isec
Engenharia

MESTRADO EM INFORMÁTICA E SISTEMAS

Ensemble SGE

Autor

Rui Manuel Rodrigues Abreu

Orientador

Prof. Francisco José Baptista Pereira

Co-Orientador

Prof. Nuno António Marques Lourenço

INSTITUTO POLITÉCNICO DE COIMBRA

INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

Coimbra, Julho 2021

isec
Engenharia

DEPARTAMENTO DE INFORMÁTICA E SISTEMAS

Ensemble SGE

Relatório de Trabalho de Projeto para a obtenção do grau de Mestre em Informática e Sistemas

Especialização em Tecnologias da Informação e do Conhecimento.

Autor

Rui Manuel Rodrigues Abreu

Orientador

Prof. Francisco José Baptista Pereira

Co-Orientador

Prof. Nuno António Marques Lourenço

INSTITUTO POLITÉCNICO DE COIMBRA

INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

Coimbra, Julho 2021

RESUMO

Este documento diz respeito a um projeto de investigação que decorreu no âmbito de projeto de final de curso do Mestrado em Informática e Sistemas, ramo de Tecnologias de Informação e Conhecimento que decorreu no Instituto Superior de Engenharia de Coimbra. Está integrado na área da aprendizagem automática e tem como principal objetivo desenvolver uma nova *framework* suportada pelo SGE para resolver problemas de aprendizagem supervisionada, e tem o nome de Ensemble SGE.

O Ensemble SGE, utiliza o SGE que é um algoritmo de evolução automática de programas, para gerar vários modelos capazes de resolver um problema. E posteriormente utiliza técnicas de aprendizagem por Ensemble para agregar alguns dos modelos gerados e produzir um Ensemble.

Neste trabalho foram abordados 3 problemas de regressão simbólica. Duas aproximações a funções conhecidas, polinómio de quarto grau e o polinómio de *Pagie* e por fim *Boston Housing*, um problema em que dadas características de uma casa é necessário prever o seu preço.

Os resultados deste projeto são positivos, é demonstrado que é possível obter Ensembles capazes de resolver alguns problemas de uma melhor forma, que o melhor modelo gerado pelo SGE. A performance obtida pela utilização de Ensembles é maior comparativamente a modelos simples gerados pelo SGE. A *framework* foi implementada e disponibilizada com possíveis casos de teste.

Concluindo, a escolha dos modelos constituintes do Ensemble é a decisão mais importante, pois não foi encontrada nenhuma maneira exata de o fazer, ou seja, apenas por métodos experimentais. O Ensemble SGE também consegue detetar situações de *overfitting* mais cedo que o melhor modelo do SGE ao longo das gerações. Isto porque o Ensemble SGE utiliza vários indivíduos de uma população.

Palavras-Chave: aprendizagem automática, aprendizagem supervisionada, programação genética, evolução gramatical, SGE, aprendizagem por ensemble

This work was funded by FEDER funds through the Operational Programme Competitiveness Factors - COMPETE and national funds by FCT - Foundation for Science and Technology (POCI-01-0145-FEDER-029297, PTDC/BTM-SAL/29297/2017

ABSTRACT

This document is the product of a research project that took place within the scope of the final project of the Master's course in Informatics and Systems, which took place at the Instituto Superior de Engenharia de Coimbra. It is integrated in the field of machine learning and its main objective is to develop a new framework supported by the SGE algorithm to solve supervised learning problems, and it’s called Ensemble SGE.

Ensemble SGE uses SGE, which is an automatic program evolution algorithm, to generate several models capable of solving problems, it also uses Ensemble learning techniques to aggregate some of the generated models and produce an Ensemble.

In this work 3 symbolic regression problems were tackled to test the framework. Two function approximations, the quartic polynomial and Pagie's polynomial. And finally the Boston Housing problem, where the point is to predict the price of a house based on some of its features.

The results of this project are positive, it is shown that is possible to obtain Ensembles capable of solving some problems in a better way than the best model generated by the SGE algorithm. The Ensemble SGE framework was implemented and made available to the scientific community with all the executed tests.

It was concluded that the choice of the base models that build the Ensemble is the most importante decision to be made on the proposed solution. Since no easy way was found to this, to find the best base models experimental methods need to be relied on. Ensemble SGE can also detect overfitting earlier than the best SGE model across generations. This is because the SGE pools several models from a population.

Keywords: machine learning, supervised learning, genetic programming, gramatical evolution, SGE, ensemble learning

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ÍNDICE DE EQUAÇÕES

TABELA DE ACRÓNIMOS

SGE	*Structured Grammatical Evolution*
CRISP-DM	*Cross Industry Standard Process for Data Mining*
AED	Análise Exploratória de Dados
PCA	*Principal Component Analysis*
CV	*Cross Validation*
LOOCV	*Leave One Out Cross Validation*
TP	*True Positives*
TN	*True Negatives*
FP	*False Positives*
FN	*False Negatives*
TPR	*True Positive Rate*
FPR	*False Positive Rate*
TNR	*True Negative Rate*
RSS	*Residual Sum of Squares*
MSE	*Mean Squared Error*
RMSE	*Root Mean Squared Error*
MAE	*Mean Absolute Error*
TSP	*Travelling Salesman Problem*
GP	*Genetic Programming*
GE	*Grammatical Evolution*
CFG	*Context Free Grammar*
BNF	*Backus Naur Form*
IQR	*Interquartile Range*

1. INTRODUÇÃO

A aprendizagem automática nos dias de hoje é uma das mais importantes áreas de estudo devido ao seu potencial no que diz respeito à resolução de problemas complexos. As tarefas realizadas por qualquer ser humano como falar, reconhecer determinados tipos de padrões, podem ser banais para os próprios, no entanto conseguir que um computador as realize desenvolvendo explicitamente um algoritmo pode tornar-se difícil ou até impossível de desenvolver. Aqui entra a aprendizagem automática que com um vasto conjunto de técnicas consegue resolver problemas de diferentes tipos como classificar peças de fruta a partir de uma imagem, prever a Temperatura que amanhã irá estar em Coimbra ou até descobrir os produtos que mais são comprados em conjunto num supermercado.

Este documento pretende descrever o trabalho realizado no âmbito do projeto final de curso do Mestrado em Informática e Sistemas, ramo de Tecnologias de Informação e Conhecimento que decorreu no Instituto Superior de Engenharia de Coimbra. O trabalho foi orientado em todos os momentos pelo Prof. Dr. Francisco Pereira numa perspetiva integral do projeto e pelo Prof. Dr. Nuno Lourenço noutra mais focada no *Structured Grammatical Evolution* (SGE) e seu funcionamento.

1.1 - Motivações

A aprendizagem automática engloba várias técnicas de aprendizagem que utilizam conjuntos de exemplos para resolver problemas complexos. Tem como principal objetivo resolver problemas que não são facilmente programáveis e para isso existem vários tipos de aprendizagem, como a aprendizagem supervisionada, aprendizagem por Ensemble entre outras que serão mais bem exploradas ao longo deste documento.

As vantagens de aprendizagem por *Ensemble* e da robustez dos modelos gerados que podem ser produzidos utilizando *ensembles* [1]. *Ensembles* são aplicados numa imensa panóplia de áreas [2] e com resultados promissores. Muitas destas aplicações conseguem ter resultados tao bons ou melhores que a aprendizagem profunda [3] que são atualmente vistas como técnicas bastante preformantes.

Por outro lado o projeto utiliza também métodos de produção de programas por via de algoritmos evolucionários que por sua vez são algoritmos de pesquisa baseados em populações [4]. Existem diversos benefícios para a utilização de algoritmos baseados em populações como por exemplo a descoberta de diferentes partes da solução, isto é, cada indivíduo da população irá identificar padrões diferentes que podem fazer parte da solução. Podem ser vistos em [5] outros benefícios deste tipo de técnicas. Por sua vez os custos deste tipo de algoritmos são altos estando a performance computacional diretamente proporcional ao número de indivíduos por população, Tendo em conta que versões avançadas deste tipo de algoritmos produzem

programas capazes de resolver problemas bastante complexos [6], torna este tipo de pesquisa bastante atraente.

Analisando ambas as técnicas e potencialidades de cada uma delas parece surgir uma oportunidade de fazê-las funcionar em conjunto. Alguns estudos tentam utilizar estas técnicas em conjunto com algum tipo de sucesso em problemas reais, em [7] é desenvolvida a ideia de evoluir *ensembles* utilizando uma estratégia multiobjectivo com o propósito de resolver o problemas de classificação com *datasets* desequilibrados. Outro exemplo em [8] é a utilização de algoritmos de evolução de programas para a seleção de características a selecionar por modelos também eles produzidos pelo algoritmo. Entre outros casos como pode ser visto em [9] onde é feita uma revisão a alguns casos de estudo que combinam as duas técnicas.

Os conceitos mencionados neste subcapítulo serão explorados nos subsequentes capítulos do documento.

1.2 - Objetivos

Descritos neste subcapítulo estão os objetivos deste projeto de investigação:

1. Desenvolver uma *framework* de aprendizagem chamada *Ensemble SGE*, que utiliza todas as vantagens de um algoritmo de produção de populações de modelos para criar *Ensembles* baseados nessas mesmas populações.
2. Análise sistemática a diversas áreas da aprendizagem automática:
    a. Com introdução à aprendizagem automática, exploração dos conceitos mais importantes e principais técnicas.
    b. Exploração de técnicas de aprendizagem por *Ensemble*, com vantagens e desvantagens das principais técnicas.
    c. Algoritmos de evolução automática de programas.

É uma investigação que pretende comprovar as vantagens da utilização de *Ensembles* para a resolução de problemas e combinar também as vantagens das técnicas de evolução automática de programas para potencialmente obter melhores *performances* ao resolver problemas complexos.

1.3 - Contribuições

Este trabalho teve como principais contribuições os seguintes pontos:

1. Proposta de uma nova framework para resolver problemas de aprendizagem supervisionada.
2. Desenvolvimento e Implementação da framework, disponibilização da mesma à comunidade
3. Comparação de resultados experimentais da nova abordagem, com técnicas já existentes (neste caso ao SGE)

1.4 - Estrutura do Documento

Para além do capítulo introdutório o presente documento está dividido em 7 capítulos:

- Capítulo 2 – Introdução à aprendizagem automática sistematizando processos e técnicas, fazendo uma análise e exploração ao funcionamento dos mesmos
- Capítulo 3 – Introdução ao conceito de *Ensemble* explorando utilidade e potencialidade, explorando algumas técnicas específicas
- Capítulo 4 – Sistematização rápida de conceitos de otimização e algoritmos específicos culminando na introdução a algoritmos de evolução automática de programas.
- Capítulo 5 – Proposta da nova *framework* de aprendizagem automática.
- Capítulo 6 – Alguns resultados experimentais relativos à nova proposta descrita no capítulo 5.
- Capítulo 7 – Considerações finais acerca do trabalho realizado, propostas de trabalho futuro e melhorias.

2. APRENDIZAGEM AUTOMÁTICA

Sendo aprendizagem automática tão vasta e abrangente existem vários casos de uso de sucesso e que podem comprovar o potencial ainda existem destas técnicas.

No dia a dia, de todas as pessoas que utilizam *smartphones*, existe um conjunto de algoritmos de aprendizagem automática que facilitam diversas tarefas a efetuar no telemóvel, o corretor automático ao escrever mensagens que utiliza técnicas de processamento de linguagem natural para prever a próxima palavra. Normalmente este tipo de mecanismos são treinados à medida do utilizador sendo que é treinado continuamente com os exemplos que utilizador realiza.

Outro exemplo muito comum são motores de recomendação, por exemplo, em *websites* de comércio *online*. Em que são utilizadas técnicas de associação produtos e previsão de produtos similares baseadas em exemplos de, por exemplo, clientes anteriores.

No âmbito da investigação destacam-se dois casos interessantes AlphaZero [10] e o AlphaFold [11] nas áreas de jogos clássicos como xadrez, go e shogi e biologia respetivamente. Ambos os projetos foram promovidos e financiados pela empresa Deepmind. As técnicas utilizadas nestes projetos são muito avançadas e não serão exploradas no presente documento. No entanto mostram a potencialidade da aprendizagem automática e a importância da área para o futuro.

Os três jogos em que [10] se foca, xadrez, shogi e go são três jogos de tabuleiro clássicos que sempre foram problemas relativamente simples de representar no entanto jogos extremamente complexos devido à possibilidade de combinações possíveis a cada jogada.

O AlphaZero é um motor de jogo que utiliza técnicas de aprendizagem por reforço. Este tipo de técnicas, dadas as regras de um determinado jogo como xadrez ou shogi produz um agente que joga contra si próprio de maneira a aprender as diferentes variantes do jogo e como vencer iterativamente. Um antecessor deste motor o AlphaGo [10][12] utilizando técnicas similares foi o primeiro programa de computador a vencer o número 1 mundial do jogo de Go.

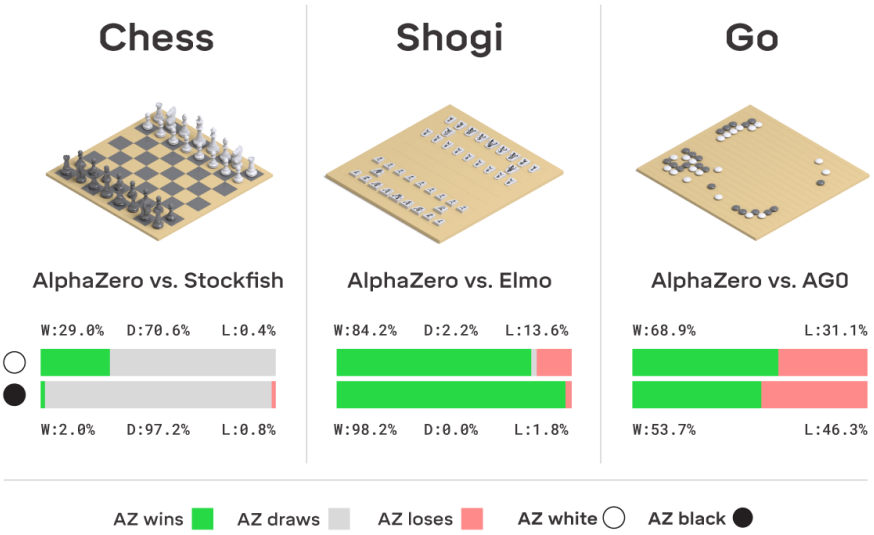

Figura 1 -Taxas de vitória do AlphaZero

Na Figura 1 podem ser vistas as taxas de vitória do AlphaZero contra os respetivos melhores motores de jogo no momento em que o AlphaZero foi lançado. Pode ser visto que nos três jogos AlphaZero dominou o adversário mostrando a potencialidade da aprendizagem automática.

Este trabalho está integrado no ramo da aprendizagem automática. Esta é a área, dentro da ciência da computação, que pretende fazer com que computadores respondam a determinado tipo de problemas sem que sejam especificamente programados para a resolução desses mesmos problemas. A Aprendizagem Automática encontra-se dentro da Inteligência Artificial que por sua vez é uma área da Ciência de Dados que por sua vez é uma área da Ciência da Computação [13].

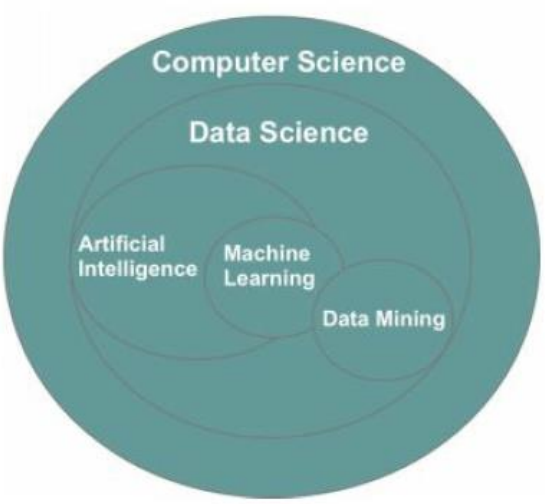

Figura 2 – Hierarquia da Aprendizagem automática [13].

A Figura 2 mostra de uma forma simples como a aprendizagem automática se relaciona com diversas áreas da ciência da computação. Um subconjunto da ciência da computação chamada ciência de dados engloba todas áreas como inteligência

artificial, *data mining* e também a aprendizagem automática. De notar que a aprendizagem automática utiliza e partilha técnicas e mecanismos com outras áreas da ciência de dados.

Este capítulo tem como principal objetivo introduzir os conceitos básicos da aprendizagem automática. Conceitos estes que serão de extrema importância para a realização deste trabalho. No subcapítulo 2.1. será explorada a relevância dos dados e como estes afetam a aprendizagem automática, em 2.2. uma comparação entre diferentes tipos de aprendizagem, em 2.4. são exploradas as típicas etapas de um projeto de aprendizagem automática, em 2.5. é revista a análise exploratória de dados o que é e porque é importante, 2.6 é explorado o como é gerado e avaliado um modelo por fim em 2.7. serão abordados alguns dos algoritmos clássicos mais utilizados na aprendizagem automática.

2.1 - Dados

Para a aprendizagem automática os dados são a grande base para a resolução de um problema, pois qualquer tipo de resultado depende maioritariamente do conjunto de dados com que se está a trabalhar. Posto isto, assume-se que ao iniciar um projeto nesta área que exista um conjunto de dados normalmente designado *dataset*. Os datasets são a base de qualquer projeto na aprendizagem automática.

O *dataset* é um conjunto de exemplos sendo que cada exemplo possui uma série de características chamadas de *features*. Os exemplos podem ter várias representações, uma imagem, uma linha de uma tabela, um conjunto de páginas web ou uma sequência genética entre muitos outros.

Figura 3 - Dataset MNIST [14]

A Figura 3 mostra um exemplo de um *dataset* em que cada exemplo é uma imagem, no entanto a representação mais tradicional são simples tabelas em que cada linha representa um exemplo e cada coluna uma *feature* como pode ser observado na Figura 4.

As *features* de um *dataset* podem ser de diferentes tipos como dados numéricos, categóricos, séries temporais, texto, imagens entre outros.

Os dados numéricos podem ser divididos em duas grandes categorias, dados discretos ou contínuos. Para dados contínuos a *feature* pode assumir qualquer valor dentro de um determinado intervalo sendo que a sua contagem seria infinita, enquanto com dados discretos os valores são distintos e finitos. Por exemplo, o número de alunos de uma turma é uma *feature* discreta pois os valores admissíveis seriam apenas inteiros, seria impossível ter 10,5 alunos numa turma. A medição da Temperatura em Coimbra seria considerada um valor contínuo pois é possível encontrar qualquer valor numérico neste tipo de features.

Dados categóricos são dados representam uma determinada característica como cidade, nacionalidade ou satisfação do cliente. Podem ser divididos também em dois grandes tipos, dados nominais ou ordinais. Dados do nominais assumem um valor textual como o nome de uma determinada cidade ou nacionalidade. Dados categóricos ordinais assumem valores numéricos, no entanto têm um valor discreto e praticamente não têm qualquer significado matemático.

Dados de séries temporais são normalmente sequências tal e qual como os dados numéricos referidos anteriormente, mas aqui existe uma ordem temporal sendo que cada número representa algo dentro de um intervalo temporal. Um exemplo de uma série temporal seria um dataset com a evolução anual das vendas de uma loja.

Dados de texto como o nome indica são apenas texto, ou seja, conjuntos de *features* que possuem apenas informação textual como comentários a um restaurante ou descrições de determinado serviço.

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10

Figura 4 - Boston Housing Dataset [15]

Na Figura 4 está o exemplo de um *dataset* apenas com *features* numéricas sendo que algumas são discretas e outras contínuas.

2.2 - Tipos de Aprendizagem

As técnicas e métodos utilizados na aprendizagem automática podem separar-se em alguns tipos, sendo que cada um deles utiliza diferentes maneiras de aprender. Existem vários tipos de aprendizagem, mas para este trabalho serão apenas

considerados dois grandes tipos da aprendizagem clássica, a aprendizagem supervisionada e a aprendizagem não supervisionada.

2.2.1 - Aprendizagem Supervisionada

A aprendizagem supervisionada [16] é o tipo de aprendizagem que necessita de dados que estejam previamente marcados de um modo especial. Neste tipo de aprendizagem uma ou mais features são marcadas ou definidas como *target* ou *label*. Para que com base nos exemplos, já com target definido, seja definida a relação entre as features e o target utilizando um processo de construção chamado treino. *Targets* ou *labels* são *features* especiais que normalmente correspondem à *feature* que se pretende prever ou classificar para um dado problema.

Os parâmetros de entrada de determinado modelo é o subconjunto de *features,* total ou parcial, sem o *target*. O parâmetro de saída será o *target* definido anteriormente.

Estes parâmetros são combinados de maneiras diferentes para prever o *target*, dependendo do algoritmo que se está a utilizar. E para mais informações sobre algoritmos consultar o subcapítulo 2.7.

Assume-se que para este tipo de aprendizagem existe uma relação entre os inputs e outputs. Daí que ao conjunto de input também se pode chamar de conjunto de variáveis independentes e ao target, variável dependente.

Por exemplo, considerando um problema de deteção de fraudes num banco, existe uma base de dados em que os dados de todas as transações se encontram categorizados e com toda a informação que os define e que, por sua vez estão também rotuladas como fraude ou não. Com estes exemplos aplicamos algumas técnicas de treino criando um modelo que define a relação entre as entradas e saídas deste problema.

Na aprendizagem supervisionada clássica existem dois tipos de aprendizagem principais a Classificação e a Regressão.

2.2.1.1 - Classificação

Os problemas de classificação são problemas em que o *output* do modelo a criar seja um valor discreto. Os algoritmos que resolvem este tipo de problemas, podem ter como *target* um ou mais valores, desde que o número de possibilidades para o output seja finito.

Os seguintes exemplos mostram exemplos de problemas de classificação que podem ser resolvidos com aprendizagem supervisionada:

- *Recebemos um e-mail com o título duvidoso, mas tem como remetente uma entidade fiável. Trata-se de spam ou não? Este pode ser considerado um problema de classificação, que recebe como input o email e como output*

produz valor Spam ou Não Spam. Este tipo de problemas por ter apenas dois possíveis outputs pode ser chamado de problema de classificação binária.

- *Um doente chega às urgências com um conjunto de sintomas, que podem ser atribuídas a três condições médicas. Qual das condições se trata? Este problema também é considerado um problema de classificação pois como output existe um número finito de condições médicas a classificar. Os sintomas do doente são considerados as features deste problema.*

2.2.1.2 - Regressão

São considerados problemas de regressão o conjunto de problemas que tem como *target* um valor contínuo. Significa que o output de um algoritmo que pretende resolver este tipo de problemas possui um valor infinito. Como por exemplo:

- *Estamos à procura de uma nova casa para viver e possuímos um conjunto de características de diversas casas. Qual é o preço de cada uma?*
- *Possuímos todas as características necessárias para definir a Temperatura do ar em determinada hora. Que temperatura estará amanhã pelas 11 da manhã?*

Ambos os problemas acima descritos são problemas de regressão pois o *target* pode ser definido por um conjunto contínuo de valores.

2.2.2 – Aprendizagem Não Supervisionada

Ao contrário dos problemas supervisionados, problemas não supervisionados [16] não possuem dados rotulados pois normalmente o objetivo destes é detetar conjuntos ou associações nos dados. Mais concretamente padrões difíceis de detetar.

Os algoritmos que resolvem este tipo de problemas são algoritmos que pretendem agrupar de determinada forma um *dataset*, por exemplo utilizando *Clustering* [17]. Os exemplos que forem parecidos, segundo determinada métrica de similaridade, devem pertencer ao mesmo conjunto e por sua vez devem ser distantes de conjuntos diferentes.

Como exemplo, podemos agrupar conjuntos de pontos geográficos singulares de maneira a perceber possíveis pontos de interesse de determinada área metropolitana de uma cidade ou encontrar conjuntos de itens que são comprados em simultâneo num supermercado.

Outro exemplo de aprendizagem não supervisionada são os sistemas de recomendação que por sua vez agregam padrões com o objetivo de sugerir itens similares, existem em *websites* de comércio eletrónico ou em qualquer rede social para sugerir conexões entre pessoas.

2.3 - Etapas de um Projeto

Sabendo que a aprendizagem automática é baseada principalmente em dados, existiu a necessidade de serem criadas algumas normas e etapas para a realização de um projeto deste cariz.

Foram já desenvolvidas algumas *frameworks* de desenvolvimento para planear e gerir os projetos com base em dados. Devido à origem experimental deste tipo de projetos, as metodologias que gerem um projeto são iterativas e com muitas comutações entre fases. O *Cross Industry Standard Process for Data Mining* (CRISP-DM), é das mais conhecidas metodologias para projetos de aprendizagem automática [18][19].

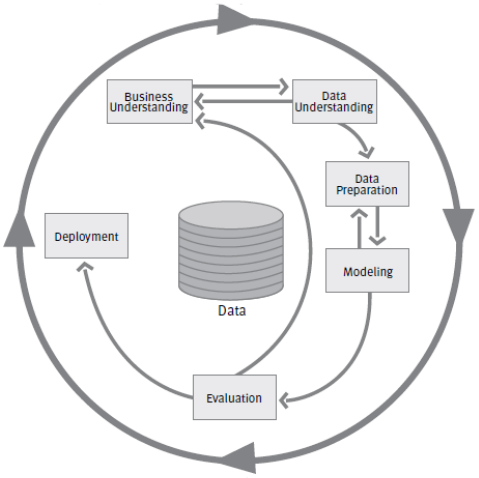

Figura 5 - Ciclo de vida CRISP-DM [6]

O ciclo de vida de um projeto proposto pelo CRISP-DM é representado pela Figura 5. Consiste em 6 fases que não são totalmente rígidas, ou seja, é possível voltar atrás em determinado ponto do processo as setas apenas representam as dependências mais frequentes. O resultado de uma fase particular vai determinar a ação a tomar na fase seguinte do processo. O círculo externo na figura representa a natureza cíclica deste processo, pois mesmo finalizado o processo as lições aprendidas podem ser reinseridas no processo gerando novas e melhoradas versões [20].

2.4.1 - Conhecimento do negócio

A fase inicial do processo onde é feito o levantamento de objetivos e requisitos do projeto de uma perspetiva do negócio. Após feito este levantamento o problema é definido formalmente e é gerado um plano preliminar de atuação para atingir os objetivos do negócio.

2.4.2 - Conhecimento dos Dados

Esta é a fase de análise exploratória que começa com a aquisição de dados e atividades de exploração dos mesmos, de maneira a conhecer melhor os dados a trabalhar. Nesta fase são identificados problemas na qualidade dos dados, são encontradas algumas especificidades nos dados e são detetados alguns subconjuntos de dados para formar hipóteses e possível informação escondida.

2.4.3 - Preparação dos dados

Nesta fase estão incluídas todas as atividades para a construção do *dataset* final pronto para as atividades de construção do modelo a partir do conjunto de dados inicial conhecido também por *raw data*. Tarefas de transformação, seleção de *features* e limpeza de dados são realizadas múltiplas vezes e em nenhuma ordem específica.

2.4.4 - Construção do Modelo

Nesta etapa, são selecionadas e aplicadas várias técnicas de modelação. Normalmente existem várias técnicas para resolver o mesmo de tipo de problemas, no entanto, algumas destas técnicas têm requisitos específicos e que leva frequentemente a ser necessário voltar à fase de preparação de dados para refazer o *dataset*.

2.4.5 - Validação

Esta é uma das fases mais importantes pois é analisado o desempenho do/os modelo/os gerado/os. Aqui são avaliados todos os passos e decisões tomadas para criar o modelo para garantir que o todas as premissas discutidas na fase inicial do projeto foram tidas em conta e se o resultado esperado resolve efetivamente o problema e atinge propriamente os objetivos propostos inicialmente.

Nesta fase é decidido se o modelo está pronto para ser *deployed* ou não, se não volta novamente à primeira fase e é repetido todo o processo.

2.4.6 - Deployment

Finalmente depois de todo o processo temos de fazer com que o modelo criado gere o pretendido valor. Ou seja, nesta fase é criada a forma de o utilizador possa utilizar o modelo, seja simplesmente a criação de um relatório com os resultados da análise do modelo ou uma complexa integração nas regras de negócio existentes, por exemplo um modelo que identifique spam no email poderá ser integrado no sistema de receção de emails para que sejam automaticamente redirecionados para o lixo, entre outros.

O CRISP-DM trata-se apenas de um conjunto de tarefas que irão aumentar a probabilidade de sucesso de um projeto. Esta metodologia tenta tratar todos os pontos importantes de um projeto baseado em dados e experimentação, no entanto não se trata de um conjunto estrito de regras que tem de ser seguido à risca, muito pelo contrário deve tudo ser visto como um conjunto de *guidelines*, normas e boas práticas num projeto deste tipo.

2.4 - Análise Exploratória de Dados

A Análise Exploratória de Dados (AED) é um processo iterativo que irá extrair a informação importante sobre o conjunto de dados a trabalhar. Na AED são realizadas algumas tarefas de tratamento e processamento de dados como a Normalização e Standardização, *Feature Selection*. Também são realizadas tarefas de análise ao dataset, análises univariadas e multivariadas, utilizando a visualização gráfica dos dados e registando sempre todas as informações importantes. Normalmente quanto mais informação se conseguir extrair nesta fase melhores são os modelos possíveis de gerar [21].

A primeira análise ao *dataset* é muito importante, são analisadas semanticamente todas as *features*, ou seja, compreender o significado de cada uma para o caso de uso. Esta análise faz parte da análise univariada do *dataset* feita para cada *feature*, como os tipos de dados de cada *feature*, as medidas de tendência central como moda, média e mediana, é realizada também uma análise de dispersão verificando a variância e desvio padrão e todas as outras medidas como mínimos/máximos e também quantis/percentis. Para isto são usadas algumas ferramentas de análise estatística como gráficos de dispersão e/ou *boxplots* representado na Figura 6.

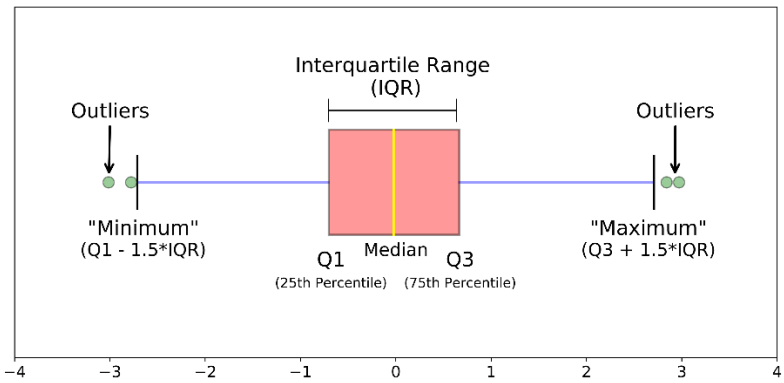

Figura 6 - Exemplo Boxplot [22]

Um *boxplot* é um a representação gráfica de um determinado conjunto de dados, é tipicamente usado para visualizar *features* singularmente, podendo ser usados para analisar qualquer tipo de dispersão. Em conjunto com histogramas e gráficos de dispersão são ferramentas indispensáveis para a fase de análise de um projeto.

A Análise Multivariada também é importante, como o nome indica, é uma análise tão cuidada como a anterior, mas neste caso são cruzadas todas as *features* entre si. São selecionadas medidas de correlação e é analisada a relação entre *features* do *dataset.* Estas análises iniciais vão originar um conjunto de meta-dados (dados sobre os dados), que por sua vez serão cruciais para tomar algumas decisões mencionadas de seguida.

Um *dataset* cru é o conjunto de dados inicial onde na maioria dos casos encontramos problemas com dados inconsistentes, dados em falta ou até dados com ruído. Para isso possíveis soluções são remover exemplos ou remover *features,* no entanto o que pode acontecer, é que os *datasets* não possuem muitos registos e remover algo poderá ser tão ou mais problemática do que não fazer nada. Tudo depende do contexto e *dataset* com que se está a trabalhar.

A deteção e tratamento de *outliers* é uma das atividades relativas ao pré processamento dos dados sendo que um *outlier* pode representar um grande peso ou erro que irá reduzir, e por vezes drasticamente, a performance do modelo a criar.

Na Figura 7 podem ser observados *outliers*, um *outlier* é um valor extremo comparativamente aos outros exemplos do *dataset.* Qualquer inconsistência ou erro terá de ter algum tipo de tratamento pois se se tratar de um erro de leitura poderá ser simplesmente removido ou se existir possibilidade ser corrigido. No entanto existem *outliers* que podem ser naturais, ou seja, são valores que efetivamente não seguem a norma dos restantes, no entanto são acontecimentos reais e que podem ter de ter tratamento especial pois podem representar novas *features* ou possíveis novas análises.

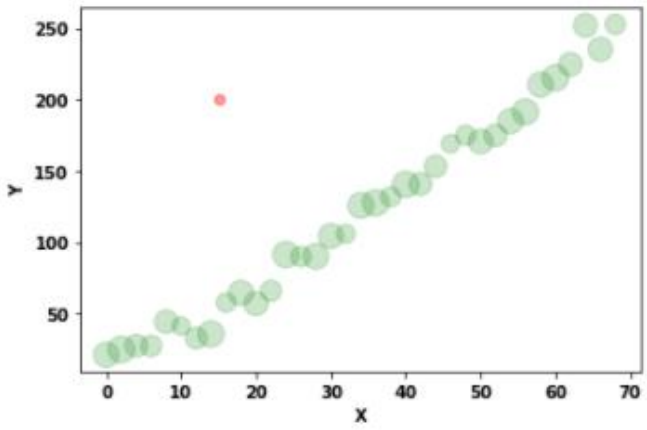

Figura 7 - Outlier

Outro tipo de problemas que podemos encontrar é a falta de dados, ou seja, *features* que para determinados registos não possuem valor. Tal e qual como nos *outliers* podemos ter diferentes abordagens para cada problema específico, no entanto a abordagem mais simples e direta poderá ser remover exemplos que possuam *features* em falta, ou se esta falta de dados se verificar ao longo de toda a *feature* pode-se optar por remover a própria *feature*.

Se o *dataset* for demasiado pequeno e remover *features* ou exemplos esteja fora de questão, dever-se-á analisar se é possível realizar imputação. Que é quando é necessário atribuir um determinado valor aos exemplos que não tenham dados. O valor a imputar pode ser um valor médio (se for um a *feature* numérica) ou até um valor recorrente (se for uma *feature* categórica) relativo a essa mesma *feature*. Existem outro tipo de técnicas mais complexas que podem ser aplicadas como se pode ver na literatura [23].

Nesta fase podem ser aplicadas algumas técnicas para transformar os dados e alterar a sua interpretação de maneira a conseguir extrair mais informação, é comum realizar nesta fase o *Feature Scaling* que se trata de homogeneizar a gama de valores das *features*. Existem dois grandes métodos de *Feature Scaling* a Normalização e a Standardização.

- Normalização: Quando se quer transformar de gama original de valores para um dado intervalo.
- Standardização: Quando se quer ajustar os dados de maneira que a média dos valores seja 0 e o desvio padrão 1.

A seleção de *features* ou *Feature Selection* trata-se de uma simples técnica de selecionar *features* que realmente interessam no contexto da análise, por exemplo, se o dataset possuir uma *feature* 'ID' que o único propósito é ter um código único por registo podemos assumir que a *feature* não tem qualquer utilidade para a análise pois é assumido que cada registo se trata de uma observação independente das outras (isto pode não ser verdade para determinados tipos de dados como séries temporais).

A seleção de *features* é realizada para simplificar o modelo a criar, irá também reduzir o tempo de construção do modelo pois estamos a reduzir a complexidade do *dataset* por exemplo ao utilizar um *dataset* com 1000 *features* em que apenas 5 das *features* são relevantes para a análise, selecionam-se apenas essas *features.* Esta seleção tem como base a análise de correlações feita anteriormente ao começar a análise exploratória de dados.

Entre outros simples métodos de seleção de *features* encontram-se o *Forward Stepwise Selection* ou *Backwards Stepwise Selection*. Ambos tem o mesmo princípio, o de experimentar cada modelo que se pretende construir por cada *feature*. Sendo que no *Forward Stepwise Selection* [16] ir-se-á numa primeira iteração testar o modelo apenas com uma *feature* e nas iterações seguintes adicionando *features* uma a uma até chega ao máximo número de *features,* após feito isto escolher o número de *features* que tiver melhor resultado. Em cada passo a *feature* escolhida para adicionar é a feature que trará um melhor aumento na performance do modelo, ou seja, a cada passo é testada cada *feature* e a que trouxer o maior aumento é adicionada.

O *Backwards Stepwise Selection* [16] funciona exatamente da mesma maneira, no entanto começa-se com todas as features até chegar a apenas uma *feature*. A *feature*

a retirar tal e qual como no método anterior é a *feature* que prejudica menos a performance do modelo.

Ambos os métodos são eficazes, no entanto são sôfregos, não garantindo o melhor modelo, e perdem rapidamente viabilidade quanto maior o número de *features* do dataset.

Feature Extraction é outro ponto que é utilizado para a redução de complexidade do modelo a utilizar, pois este conjunto de métodos gera novas *features,* ou seja, a partir do dataset inicial gera novas características baseado dos dados do próprio *dataset*. O *Principal Component Analysis* (PCA) [17] é um destes métodos que vai combinar linearmente as *features* originais criando um novo referencial. As direções no espaço deste novo referencial são escolhidas com base na variabilidade dados desde a maior variabilidade até à menor, as *features* geradas pelo PCA chamam-se componentes, podem ser criadas tantas componentes como quantas *features* existirem no *dataset*. O PCA é mais eficaz quando o *dataset* possui features muito relacionados pois vai criar novas componentes diretamente não estão relacionadas com nenhuma das *features* em específico.

Toda a informação nesta fase desde o tipo de *features* que possuímos até aos padrões e relações entre *features* deve ser identificado pois são informações relevantes para a escolha das técnicas e algoritmos a utilizar. É aqui então realizada a criação dos modelos que irão resolver os problemas mais complexos identificados anteriormente. Estes modelos utilizam técnicas e algoritmos que serão mais aprofundados no subcapítulo 2.6.

2.5 - O Modelo

Na fase posterior á análise e processamento dos dados é necessário aplicar diversos tipos de algoritmos e técnicas para resolver efetivamente o problema.

Este capítulo é orientado a problemas de aprendizagem supervisionada onde os modelos a gerar possuem um *input* e um *output,* sendo que o primeiro trata-se do conjunto de *features* escolhidas para representar o exemplo e um *output* o *target*.

Inicialmente deve ser escolhido então o algoritmo a utilizar para resolução do problema, alguns destes serão analisados à frente no documento mais ao pormenor. Cada algoritmo possui uma série de especificações e limitações desde apenas aceitar um tipo específicos de dados, como dados numéricos ou categóricos, como a necessidade de um processamento adicional de tratamento de *output*. Esta decisão, e como demonstrado anteriormente, pode fazer com que seja necessária voltar à fase de processamento dos dados para serem aplicadas algumas técnicas de transformação de *features*.

Apesar de cada um dos algoritmos possuir uma maneira específica para ser construído e utilizado, seguem todos um conjunto de ideias gerais que serão apresentadas de seguida.

2.5.1 - Construção do Modelo

A fase de construção é a fase em que o modelo vai aprender e ajustar os parâmetros internos de maneira a otimizar uma função/estrutura objetivo com base dos dados do *dataset* original. Esta construção é diferente para os diferentes tipos de aprendizagem, supervisionada e não supervisionada.

Na aprendizagem não supervisionada sabe-se que o principal objetivo é encontrar padrões e construir uma estrutura a partir dos dados. Para cada algoritmo tem-se maneiras diferentes de o fazer, mas normalmente os modelos são gerados a partir da totalidade do *dataset* formando determinadas estruturas ou padrões tendo em conta as similaridades entre exemplos e métricas de distância entre *features*.

Já no caso da aprendizagem supervisionada a construção do modelo é chamada de fase de treino. A técnica mais comum para construir um modelo é dividir o *dataset* percentualmente, ou seja, utilizar 70% do *dataset* e treinar o modelo com essa amostra e os restantes 30% são utilizadas para validação (valores 70/30 são os mais comuns, mas podem variar consoante o problema). Isto acontece, pois assim é garantido que o modelo é testado com exemplos que nunca viu podendo assim ser possível verificar se o mesmo está ou não a generalizar, no entanto existe a possibilidade de o modelo criado se adaptar em demasia aos dados de treino, a isto chama-se *Overfitting*. O contrário também pode acontecer onde o modelo gerado não consegue identificar a estrutura correta nos dados e a isso chamamos *Underfitting*.

Para detetar ambas das situações referidas anteriormente são utilizadas diversas técnicas sendo que a mais simples é a comparação da *performance* do modelo gerado em treino e em teste. No caso do *Overfitting* verifica-se que a taxa de acerto é perfeita numa fase de treino, no entanto na fase de teste esta taxa desce bastante, já no caso de *Underfitting* quando a taxa de acerto no treino é extremamente baixa significa que o modelo não está a conseguir aprender, ou seja, não está a conseguir criar a estrutura necessária prever corretamente os targets.

Existem várias maneiras de prevenir este tipo de situações. Existem soluções que são específicas para cada tipo de algoritmos como a regularização, que se trata de um conjunto de técnicas que forçam o modelo a ser mais simples, alterando parâmetros internos do algoritmo de maneira a desvalorizar determinados exemplos corretamente previstos.

No entanto, muitas das vezes é possível utilizando apenas ajustes nos dados de input. Uma das abordagens para a prevenção do *overfitting* pode ser a utilização de mais dados para treinar o modelo ou apenas a remoção de algumas *features* tendo que cada uma destas alterações ser testadas para cada caso.

É comum reorganizar aleatoriamente todos os exemplos antes da separação treino/teste com o objetivo de treinar o modelo com outros exemplos que nunca foram usados para treino antes. No entanto nem sempre se obtém grandes resultados quando, por exemplo, se possui um *dataset* muito tendencioso (elevado *bias*) ao nível das *labels*, ou seja, com muito mais *labels* de um tipo do que de outro. Este é um problema comum [24] e existem vários tipos de soluções para o resolver como pode ser visto [25] em que é proposta um solução relativamente complexa onde é atribuída uma métrica para quantificar o valor do bias e por sua vez é executado um algoritmo de otimização para minimizar este valor. No entanto uma simples maneira de resolver o problema pode ser utilizando, uma sobre amostragem com repetição. Ou seja, acrescentar ao dataset original uma amostra de exemplos das *labels* que tenham menos observações repetindo algumas dessas observações. Podem também em simultâneo ser usadas técnicas como o *Cross Validation* para ajudar a combater este tipo de problemas.

O *Cross Validation (CV)* é uma técnica que tenta iterativamente utilizar diferentes partes do dataset como treino e teste [26], isto é, o Cross Validation seleciona uma pequena amostra do *dataset* para utilizar como teste e o restante para treinar, numa iteração seguinte fará o mesmo processo com amostras diferentes. Posto isto destacam-se por maior utilização [26], duas estratégias para a escolha das amostras o *Leave One Out Cross Validation (LOOCV)* e *K-Fold Cross Validation.*

O *LOOCV* é uma abordagem que como referido anteriormente irá separar o *dataset* em duas partes treino e teste, irá fazer *n* vezes, sendo que *n* representa o número total de registos do *dataset,* utilizando sempre de cada vez apenas um registo como *dataset* de teste.

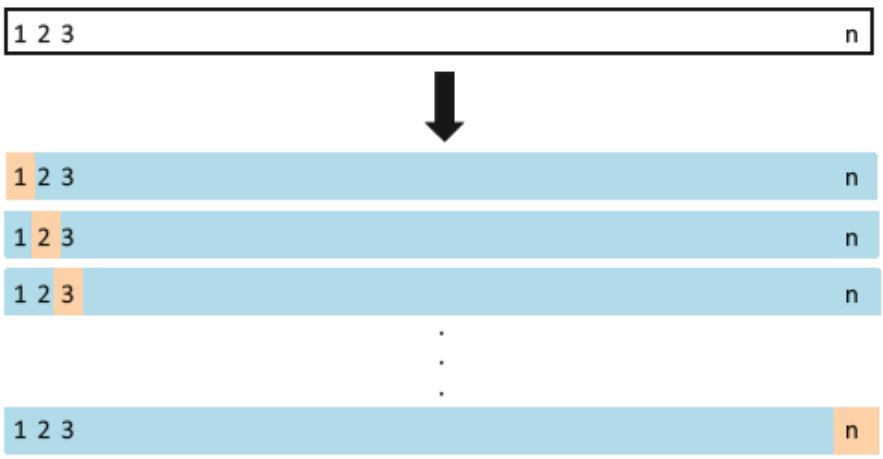

Figura 8 - *Leave One Out Cross Validation*[17]

A Figura 8 representa o funcionamento do LOOCV separando o dataset original em n grupos separando treino e teste (azul e amarelo respetivamente). O erro pode ser calculado pelo MSE fazendo a média de cada um dos grupos. O primeiro grupo de

treino possui todos os registos excepto o registo 1, o segundo grupo de treino possui todos os registros excepto 2 e assim sucessivamente [17] .

Todos os datasets gerados alimentam o modelo e ao calcular o erro de cada iteração e por sua vez a média dos resultados de um determinado modelo percebe-se que o *bias* é muito mais baixo que utilizando o método clássico de separação de treino e teste e os resultados são melhores. Este método é geral o suficiente para ser aplicado a qualquer problema, pode tornar-se, no entanto, muito dispendioso para valores de *n* muito elevados.

O *K-Fold* por sua vez também tenta separar o *dataset* e treinar o modelo da mesma forma, no entanto recebe um *K* que representa o número de grupos, ou *folds*, que o *dataset* original será dividido. Esta divisão é feita de forma aleatória e com dimensões similares, ou seja, a cada iteração do *Cross Validation* é escolhido um conjunto de treino e de teste com um número similar de registos que na iteração anterior.

A Figura 8 representa a maneira como o *K-Fold Cross Validation* separa o dataset neste caso para K = 5 em que vai re-treinar o modelo 5 vezes com os valores de treino e teste a azul e amarelo respetivamente. As métricas e os grupos utilizados são as mesmas que para o LOOCV.

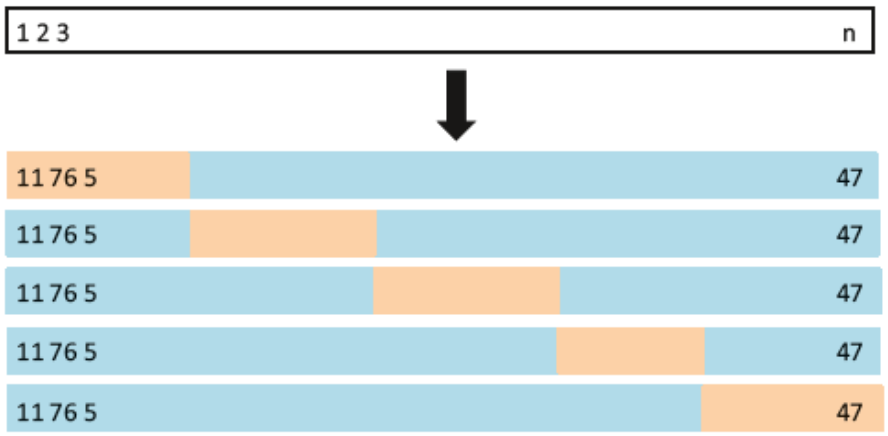

Figura 9 – K-Fold Cross Validation Imagem retirada de [17]

Visualizando a Figura 9 facilmente se percebe que o *LOOCV* é um caso especial do *K-Fold* onde *K = n.* Em que a maior vantagem do K-Fold é o processamento necessário, pois no caso do *LOOCV* o modelo terá de ser treinado o número de vezes igual ao número total de registos e no caso de por exemplo um *10-Fold* apenas terá de ser re-treinado 10 vezes em que computacionalmente temos uma melhoria bastante significativa para casos que tenhamos um *n* grande.

2.5.2 - Validação do Modelo

Os diferentes tipos de problemas restringem as métricas a utilizar, devido há maneira como são calculadas ou a própria semântica de cada uma.

De uma forma geral o processo de validação tanto para problemas de classificação ou regressão assume que já possuímos um modelo construído. Este modelo é executado sobre todo o conjunto de validação separado *a priori* e os resultados do mesmo são comparados com os valores reais existentes no *dataset*. Aqui entram as métricas de performance que serão calculadas a partir desta comparação.

2.5.2.1 Classificação

Uma das mais importantes estruturas utilizadas em problemas de classificação é a Matriz de Confusão que expõe de uma maneira clara e explícita algumas somas vindas da comparação acima abordada e a partir dela, calculam-se algumas métricas para medir a performance do modelo.

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

Tabela 1 - Matriz de confusão [17]

Na Tabela 1 verifica-se que um determinado modelo em 10000 exemplos de testes errou 275 vezes, concluindo que temos uma taxa de erro de 2,75% que por sua vez é muito bom e para alguns problemas este resultado de 97,25% de taxa de acerto (ou precisão) é suficiente e valida o modelo. No entanto ao analisar apenas exemplos com *default status* a *Yes* que no total são 333 exemplos verifica-se um erro de 252 exemplos que resulta numa taxa de erro de 75,7%. Que para outro tipo de problemas pode ser um grande problema e o modelo terá de ser reconstruído. De notar que neste caso a matriz de confusão diz respeito a um *dataset* muito desequilibrado sendo que é necessário ter cuidados adicionais ao analisar a matriz e recorrer a outras métricas.

A Matriz de Confusão apresentada na Tabela 1 é uma matriz com um *target* com duas possibilidades, para datasets com mais *targets* todas as possibilidades serão apresentadas na matriz. A Matriz é construída de uma maneira intuitiva e simples e possui termos específicos para cada valor que a constitui. De seguida verifica-se a designação básica de cada valor da matriz:

	Actually Positive	***Actually Negative***
Predicted Positive	*True Positive* (TP)	*False Positives* (FP)
Predicted Negative	*False Negative* (FN)	*True Negatives* (TN)

Tabela 2 - Designação de cada valor de uma matriz de confusão [26]

De um modo exemplificativo define-se um problema de deteção de uma doença em que se aplica um modelo de classificação e verifica-se a matriz de confusão relativa à performance do mesmo modelo, é possível então dizer que:

- True Positives (TP) – são exemplos classificados como se possuíssem a doença e efetivamente tinham a doença
- True Negatives (TN) – são exemplos classificados como se não possuíssem a doença e efetivamente não possuem a doença
- False Positives (FP) – são exemplos classificados como se possuíssem a doença, mas não a possuem (também conhecido como erros de Tipo 1)
- False. Negatives (FN) – são exemplos classificados como se não possuíssem a doença, mas possuem (também conhecido como erros de Tipo 2)

Erros de Tipo 1 não são necessariamente piores que erros de Tipo 2 pois depende exclusivamente do caso de uso do modelo em questão, neste caso um erro de Tipo 2 é mais grave que um erro de Tipo 1 pois não é identificada uma doença que possivelmente pode ter consequências graves visto que o objetivo é apenas detetar a doença.

A partir da matriz é possível então gerar diversas métricas/rácios entre os diferentes valores que irão medir a performance de um modelo de classificação a criar. A seguinte tabela mostra algumas das mais importantes métricas possíveis de extrair da Matriz de Confusão:

Accuracy	$\frac{(TP + TN)}{Total\ Examples}$
Error Rate	$\frac{(FP + FN)}{Total\ Examples}$
True Positive Rate (TPR) / Recall	$\frac{TP}{All\ Actually\ Positive\ Examples}$
False Positive Rate (FPR)	$\frac{FP}{All\ Actually\ Negative\ Examples}$
True Negative Rate (TNR)	$\frac{TN}{All\ Actually\ Negative\ Examples}$
Precision	$\frac{TP}{All\ Predicted\ Positive}$
F Measure	$2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$

Tabela 3 – Métricas Classificação

Na Tabela 3 são apresentadas algumas das métricas mais utilizadas para medir a performance de modelos de classificação, calculadas a partir da matriz de confusão.

A *Accuracy* representa a taxa de acerto do modelo, ou seja, é dos principais indicadores para a maioria dos problemas, no entanto não pode ser utilizado sozinho

pois pode induzir em erro. O *Error Rate* como o nome indica representa a taxa de erro do nosso modelo será no fundo o contrário da *Accuracy*. O *Recall* é a taxa de acerto no universo dos exemplos que deviam ser corretamente previstos, é importante utilizar esta métrica principalmente quando um valor previsto incorretamente tem um grande impacto no âmbito do problema, alterar o *Recall* tem impacto nas restantes métricas. A *Precision* é calculada tendo em conta apenas os valores previstos e mede o quão próximos estão as previsões do modelo umas das outras. Uma alta *Precision*, não significa que o modelo está a acertar corretamente nos valores reais, mas sim que os resultados do modelo são muito próximos uns dos outros.

Pode dizer-se que a Precision e o Recall são métricas antagónicas sendo que quando uma delas aumenta a outra diminui. Idealmente quer-se que ambas sejam altas daí que é utilizada para correlacionar ambas as métricas a F-Measure.

A *F Measure* é uma métrica que relaciona o *Recall* e a *Precision,* é a média harmónica entre as duas métricas referidas e quanto maior mais coesos e acertados são os resultados. Tanto o *FPR* como o *TNR* são métricas similares ao *Recall* tendo semânticas e utilizações similares.

Como representado na Tabela 3 existem várias métricas importantes, no entanto existem outras técnicas importantes que utilizam várias destas métricas sendo uma delas a curva ROC que é uma representação gráfica em que é colocado no eixo das abcissas o *False Positive Rate* e no das ordenadas o *True Positive Rate*.

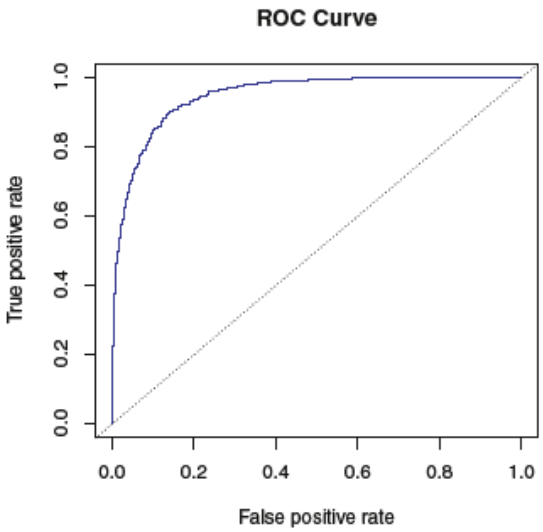

Figura 10 - Exemplo de uma Curva ROC [17]

É pretendido que a curva *Receiver Operating Characteristics* (ROC) esteja o mais próxima possível do canto superior esquerdo pois o objetivo é maximizar a *Area Under Curve* (AUC), quanto maior a AUC melhor o classificador. Na Figura 10, a linha pontilhada que vai do canto inferior esquerdo ao canto superior direito representa o classificador sem informação ou aleatório, pode ser utilizada como comparação para o classificador que se está a tentar construir.

2.5.2.1 Regressão

Para problemas de regressão estes tipos de métricas podem ser utilizadas, no entanto não têm grande utilidade pois as métricas analisadas até agora assumem que existe um espectro finito e discreto de outputs que não existe em problemas de regressão, sendo que estes resultam em *labels* contínuas. Por isso de seguida estão algumas das métricas mais comuns para medir performance de modelos aplicados a problemas de regressão:

Residual Sum of Squares (RSS)	$RSS = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$
Mean Squared Error (MSE)	$MSE = \frac{1}{n}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2$
Root-Mean-Squared-Error (RMSE)	$RMSE = \sqrt{\sum_{i=1}^{N} \frac{(y_i - \hat{y}_i)^2}{n}}$
Mean-Absolute-Error (MAE)	$MAE = \frac{1}{n}\sum_{i=1}^{N} \lvert y_i - \hat{y}_i \rvert$
R² or Coefficient of Determination	$R^2 = 1 - \frac{MSE(Model)}{MSE(Baseline)}$

Tabela 4 – Métricas Regressão

Na Tabela 4 estão apresentadas as mais comuns métricas de validação de modelos de regressão. Nas fórmulas da tabela *n* representa o número total de elementos do dataset de validação, y representa o valor real do dataset de validação e $\hat{y}$ representa o valor previsto pelo modelo.

O *RSS* é uma abordagem muito simples que apenas calcula a soma dos das diferenças entre o valor real e o valor previsto pelo modelo. Já *MSE*, muito parecido ao anterior, é a média da diferença quadrática entre o valor real e o valor previsto pelo modelo. Como as diferenças entre real e previstos estão elevadas ao quadrado a mínima diferença terá um grande impacto penalizando o valor da métrica.

No caso da *RMSE* trata-se da raiz quadrada da diferença quadrática média entre o valor real e valor previsto, como a média dos erros está a ser calculada antes da raiz quadrado faz com que penalização a erros muito grandes seja maior, ou seja, é uma métrica muito útil se grandes erros forem indesejados.

A métrica *MAE* calcula-se fazendo a diferença absoluta entre o valor real e valor previsto, neste caso a métrica é mais robusta que as anteriores no que diz respeito a *outliers*, ou seja, não penaliza muito os erros. É uma métrica linear sendo que todas as diferenças calculadas têm exatamente o mesmo peso o que significa que é uma métrica que não deve utilizada se quisermos ter em conta os *outliers*.

Por fim a R^2 é uma métrica que pretende comparar os resultados do modelo com os resultados de uma linha de base constante criada a partir dos dados do *dataset* de validação, o resultado desta métrica será sempre menor ou igual a 1.

2.6 - Algoritmos

Uma das fases de um projeto na área da aprendizagem automática é a escolha do algoritmo. Esta escolha é normalmente feita tendo em conta as características do problema que se pretende resolver e também a experiência de quem o está a implementar pois os diversos algoritmos utilizados têm características personalizáveis como iremos ver mais à frente. Nesta secção encontram-se alguns algoritmos utilizados na aprendizagem automática e respetivo funcionamento.

2.6.1 - Algoritmos Supervisionados

São descritos neste subcapítulo 3 algoritmos supervisionados apresentando a regressão linear que é um algoritmo de regressão seguindo-se a regressão logística e árvores de decisão que irão facilitar a compreensão de conceitos em seguintes capítulos. Os 3 algoritmos foram escolhidos, devido à sua simplicidade para facilitar compreensão, de entre muitos outros algoritmos supervisionados, como as Redes Neuronais [27], Naïve Bayes entre outros [17].

2.6.1.1 - Regressão Linear

Uma regressão linear é uma das abordagens mais simples da aprendizagem supervisionada sendo ponto de partida para diversos problemas e novas abordagens segundo a literatura [17].

É um algoritmo que tenta prever o valor da variável dependente *Y* dado o conjunto de variáveis independentes *X*. Existem dois tipos de regressão linear, a simples onde o conjunto *X* é composto apenas por uma variável e a múltipla que como nome indica faz com que o conjunto *X* possua mais que uma variável independente.

Uma regressão linear simples tenta obter uma linha no plano, já uma regressão múltipla tenta obter um plano ou Hiper plano dependendo do número variáveis independentes, que melhor se ajustem aos dados para a qual o erro de previsão seja o menor possível. Uma regressão linear pode ser representada matematicamente pela Equação 1.

$$Y \approx \beta_0 + \beta_1 X$$

Equação 1 - Regressão Linear

Podemos ler o símbolo "≈" como "aproximado por". Possuímos dois parâmetros β_0 e β_1 que representam respetivamente a ordenada na origem e o declive da reta que pretende representar a regressão linear entre *X* e *Y*. Estes parâmetros são

inicialmente desconhecidos, tendo então de ser calculadas com um conjunto de dados de treino.

Para uma regressão linear simples, os parâmetros são ajustados segundo a equação apresentada anteriormente, ou seja, quer-se calcular o declive e a ordenada na origem que aproxime da melhor forma os exemplos do *dataset* de treino. Para o cálculo do erro podem ser utilizadas quaisquer métricas referidas no subcapítulo anterior.

Para uma regressão múltipla os passos referidos anteriormente são adaptados para um maior número de dimensões sendo que os pressupostos se mantêm.

2.6.1.2 - Regressão Logística

A regressão logística é um algoritmo utilizado para quando o *target* de determinado problema é do tipo categórico. [16][28]

Por exemplo, um banco necessita de criar um modelo que tenha como objetivo classificar se deve ou não validar um empréstimo com um indivíduo baseado no saldo bancário. É definido um limite "Sim efetuar empréstimo" se $p(saldo) < 10000$. Se o banco achar que 10000 é um risco demasiado elevado pode definir este limite como $p(saldo) < 5000$ que faz com que ocorra menos vezes. Com esta estratégia é possível desenvolver um classificador categórico com uma regressão linear, no entanto, devido ao facto de a mesma não possuir limites, ou seja, pode prever valores até ao infinito, torna-se muito difícil e complexo definir um limite exato podendo isto ter um grande impacto no resultado do algoritmo.

Daí que para problemas de classificação é utilizada a regressão logística, que possui todos os princípios da regressão linear, no entanto é aplicada uma função sigmoide com o intuito de normalizar os resultados da previsão, por simplicidade será sempre assumido que se trata de uma classificação binário onde se classifica sempre 1 ou 0.

A função sigmoide é uma função contínua não linear tipicamente utilizada em redes neuronais [29] que varia entre 0 e 1. Esta função é aplicada diretamente ao resultado da previsão normalizando os resultados prevenindo o problema referido anteriormente de obter resultados infinitos. A função sigmoide é representada por:

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

Equação 2 - Sigmoide

O output de uma regressão logística é a probabilidade de esse acontecimento ser realizado no caso do banco referido anteriormente, seria tido como *output* por exemplo $p(emprestimo) = 1$ representando que a probabilidade do empréstimo ser validado ter 100% de probabilidade de acontecer.

O processo de classificação é então definido pelos inputs que podem ser representados por (*x1, x2…, x_n)*, e cada um dessas *features* é multiplicada por coeficientes de regressão *(w1, w2, …, w_n)*, é então realizado o somatório *z*:

$$z = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$$

Finalmente *z* passa pela função sigmoide dando assim o resultado para a previsão. Para a classificação é tipicamente utilizado o limite de 0.5, ou seja, se $z > 0.5$ então 1 senão 0.

2.6.1.3 - Árvores de Decisão

Algumas das terminologias aqui introduzidas irão ser utilizadas para tópicos mais avançados. Árvores de Decisão são dos métodos de aprendizagem automática que utilizam árvores para definir a estratégia a utilizar para atingir o objetivo. [17]

Uma Árvore de Decisão é uma árvore binária que consiste num conjunto de nós ligados entre si que formam uma árvore. Possuem um nó chamado de raiz que como o nome indica é onde a árvore começa. Todos os outros nós, possuem um ramo de entrada. Um nó, quando possui ramos de saída tem o nome de nó interno e quando um nó não possui ramos de saída tem o nome de folhas (também conhecido como nó terminal). [30]

Cada nó interno possui uma função de separação, que está associada às *features* de entrada, por exemplo se uma das *features* representar a idade de uma pessoa uma possível função para um nó interno seria $idade > 18$. Um nó interno representa uma *feature*, por exemplo da idade, os eles representam o conjunto de valores que passam ou não nessa função. Utilizando o mesmo exemplo, instâncias que se avaliem verdadeiras na condição $idade > 18$ vão para um ramo e as que forem falsas vão para outro. Importante salientar que neste caso os conjuntos que vão para os ramos têm de ser mutualmente exclusivos e completos para garantir que cada instância é mapeada para apenas uma folha.

Cada folha representa os resultados que se irão obter numa previsão. Ou seja, serão associados os valores do target mais apropriados que estão dependentes do algoritmo de construção utilizado.

Se a previsão for um valor categórico a árvore de decisão pode ser chamada de Árvore de Classificação e no caso de a previsão ser um valor contínuo pode ser chamada de Árvore de Regressão. Para realizar a previsão uma instância passa por todos os nós internos desde a raiz respeitando todas as condições de cada nó interno que passar, quando for atingida uma folha, o valor aí guardado será o resultado da previsão.

O exemplo seguinte de uma árvore de decisão de um *dataset* que pretende medir a performance de um jogador de basebol, baseado apenas no número de acertos por época e anos de experiência. É assumido para este caso que à esquerda é falso e à direta é verdadeiro

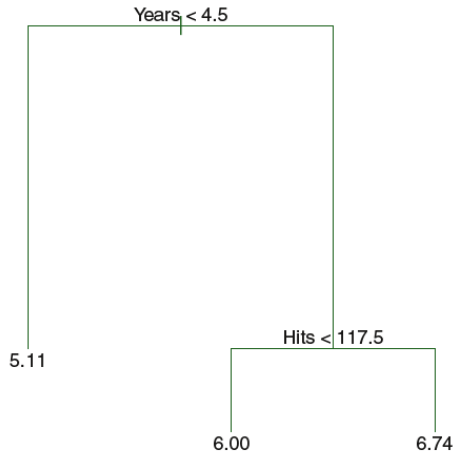

Figura 11 –Árvore de Decisão [17]

Na Figura 11 está representado um exemplo de uma árvore de decisão. Esta árvore possui 2 nós internos em que um deles é a raiz (*Years* < *4.5*) e três folhas com os possíveis resultados de (5.11, 6.00, 6.74).

Para construir uma árvore é tipicamente utilizado uma técnica chamada de *Binary Recursive Splitting* [17] que vai definir as estratégias de separação de ramos. Esta é uma técnica iterativa e "gulosa" pois separa as observações baseando-se exclusivamente no passo atual, sendo que se olhasse para passos sucessivos poderia fazer a uma melhor separação.

Para realizar o *Binary Recursive Splitting* é assumido que todos os exemplos do dataset pertencem ao mesmo espaço. São selecionadas cada uma das *features* e é aplicada uma condição (ex. *feature* < *4.5*), a esta condição dá se o nome de preditor. É então selecionado o preditor que minimize alguma função de avaliação. O primeiro preditor selecionado vai corresponder à raiz da árvore, que por sua vez vai ser o "melhor classificador", pois vai conseguir separar o maior conjunto de dado. Para cada conjunto separado é recursivamente efetuado o mesmo processo removendo as *features* associadas aos preditores já processados. Este processo continua até a uma determinada condição de paragem como um limite na função de custo ou não existirem mais *features* para criar preditores, entre outras no capítulo 6 de [30].

Existem diversas funções de avaliação que podem ser também chamados de critérios de separação [30], salientando duas das mais populares que *Gini Index* e *Entropy* (Entropia).

O *Gini Index* pode ser visto como o valor de pureza de determinado conjunto, ou seja, quanto mais baixo for o valor do *Gini Index* significa que esse conjunto contém observações apenas de uma *label*. Em suma o *Gini Index* é a mede a quantidade de vezes que ao escolher um elemento do conjunto esse estará incorretamente previsto.

$$Gini\ Index = 1 - \sum {p_j}^2$$

Equação 3 - Gini Index

O *Gini Index* calcula-se como representado na fórmula acima, *p* representa a probabilidade do target *j* acontecer, ou seja, são somadas todas as probabilidades possíveis. No caso de um dataset possuir quatro observações duas delas com target *true* e outras duas com target *false,* então:

$$Gini\ Index = 1 - ({p_{true}}^2 + {p_{false}}^2)$$

$$Gini\ Index = 1 - (0.5^2 + 0.5^2) = 0.5$$

Equação 4 - Aplicação do Gini Index

O *Gini Index* tem o valor de 0.5 que significa que este conjunto tem um valor de pureza de 0.5.

A Entropia é o chamado valor de confusão que significa que quanto maior o mesmo, maior a variedade de observações da mesma *label* em determinado nó, tal e qual como com o *Gini Index* é ideal que este valor seja baixo pois representa que determinado nó apenas contém observações de uma *label*.

$$Entropy = -\sum p_j \times \log_2 p_j$$

Equação 5 - Entropy

Na equação da *Entropy, p* representa novamente a probabilidade de um determinado *target j.*

$$Entropy = -(p_{true} \times \log_2 p_{true} + p_{false} \times \log_2 p_{false})$$

$$Entropy = (0.5 \times \log_2 0.5) - (0.5 \times \log_2 0.5) = 1$$

Equação 6 - Aplicação da Entropy

O valor da *Entropy* é o valor 1, ou seja, valor máximo de confusão que representa que o conjunto tem *targets* muito variados.

O processo acima descrito pode efetivamente produzir boas previsões no treino, no entanto irá gerar com certeza *overfitting.* Isto porque a construção da árvore continua até, por exemplo, MSE ser 0 podendo tornando a árvore muito grande e complexa fazendo com que a árvore restrinja casos específicos, fazendo com que o erro de teste aumente para conjuntos que nunca foram vistos. Outro problema de árvores complexas pode ser a dificuldade de processamento de estruturas tão complexas.

Para prevenir esta situação pode ser necessário alterar a função de custo de maneira que a árvore gerada seja mais simples, ou utilizar uma técnica chamada *prunning*. *Prunning* é uma técnica que pretende simplificar as árvores de decisão, em que basicamente tenta “cortar” ramos das árvores segundo alguma condição. Existem vários tipos de *prunning* que podem ser analisados ao pormenor em [30].

Todos os métodos de construção de árvores utilizam os métodos de separação e *prunning* acima referidos. No entanto existem configurações específicas para a construção das árvores que utilizam e conjunto de técnicas utilizadas em conjunto

como o ID3, C4.5 e CART que podem ser explorados mais aprofundadamente em [30].

As principais vantagens para a utilização das árvores de decisão é que são métodos autoexplicativos, ou seja, é possível justificar e perceber exatamente como se chega ao resultado o que não acontece por exemplo com regressões mais complexas. Conseguem facilmente lidar com valores tanto numéricos como categóricos.

Como desvantagens as árvores de decisão não têm grande capacidade de prever valores numéricos. Também possuem alguma dificuldade em problemas cujas relações e interações sejam muito complexos pois maioritariamente as árvores dividem o espaço de procura em conjuntos mutuamente exclusivos, ou seja, faz com que para determinados problemas a árvore necessite de criar subárvores duplicadas eventualmente tornando a árvore demasiado complexa.

Como vai ser possível confirmar nos próximos capítulos ao combinar vários tipos de árvores podemos criar modelos muito mais poderosos que qualquer um baseado apenas numa árvore de decisão.

2.6.2 - Algoritmos não Supervisionados (Clustering)

É sabido que existem vários tipos de aprendizagem não supervisionado como referido na literatura [17]. Para este caso é explicado o *Clustering* que é uma das técnicas mais utilizadas neste âmbito de aprendizagem não supervisionada. São abordadas variantes de algoritmos para uma melhor compreensão dos métodos não supervisionados.

Clustering refere-se ao conjunto de técnicas que pretendem separar um determinado conjunto de dados em grupos, também chamados *clusters*, tentando que cada elemento pertencente a um *cluster* seja muito similar a outro elemento desse mesmo *cluster* e em simultâneo, que elementos de *clusters* diferentes sejam muito diferentes. O ponto médio calculado a partir das observações pertencentes ao *cluster* tem o nome de centroide.

Nesta fase é necessário definir o que significa ser igual e diferente, e para isso existem várias abordagens para medir a distância e similaridade entre observações. No caso do *K-means* pretendemos dividir o conjunto de dados em um número (*K*) predefinido de *clusters*. Já no caso do *Clustering* Hierárquico não sabemos à partida o número de *clusters* até porque aqui produz-se um dendograma que conectará todas as observações do conjunto de dados assim conseguindo visualizar os todos os possíveis *clusters*.

2.6.2.1 - K-Means

O K-Means é uma das técnicas de *clustering* mais populares. No início da execução recebe um valor K que representa o número de clusters que serão considerados para

a divisão das observações. Então o algoritmo irá atribuir a cada observação exatamente um dos K *clusters*. A partir daí o procedimento tende a respeitar duas regras: cada observação pertence apenas a um só grupo K; Não existe sobreposição entre clusters, ou seja, cada observação pertence apenas a um cluster.

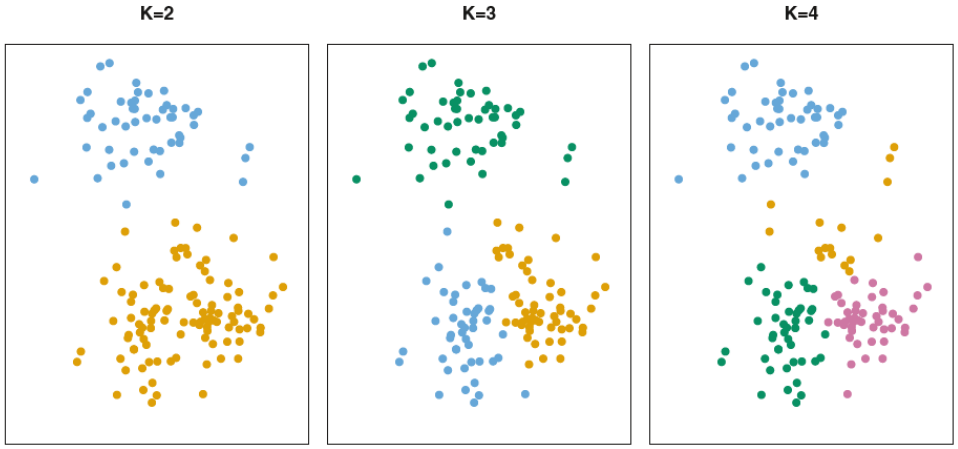

Figura 12 - Exemplo do K-means com 3 valores diferentes para K[17]

O K-Means pretende então minimizar a variação intra-*cluster* sendo esta uma medida que mede a diferença entre as observações dentro do mesmo *cluster.* Existem várias métricas de distância como a Distância Euclidiana ou de *Manhattan* entre outras [31]. Para obter a variação intra-*cluster* podemos calcular soma das distâncias entre cada uma das observações do *cluster* a dividir pelo número de observações desse mesmo *cluster*.

Para resolver o problema é utilizado um simples algoritmo iterativo que pretende encontrar um ótimo local tendo em conta que este problema se torna muito complicado se quisermos encontrar o valor ótimo para a variação intra-*cluster*, o algoritmo é o seguinte:

1. Atribuir aleatoriamente um número de 1 a K a cada observação
2. Iterar $n = 0;\ i = n, n+1, \ldots, K$:
    a. Para cada K *clusters* computar o seu centroide.
    b. Atribuir a cada observação ao cluster cujo centroide se encontra mais próximo, sendo que mais próximo é definido pela distância Euclidiana.

Este algoritmo não garante um ótimo global os resultados irão depender do estado inicial que é gerado por métodos estocásticos. Daí que é importante que o algoritmo itere várias vezes para poder ser escolhida uma melhor solução. Eventualmente irá convergir para devolver todas as observações corretamente agrupadas.

2.6.2.2 - Clustering Hierárquico

O *Clustering* Hierárquico também conhecido como *Clustering* Aglomerativo não necessita da especificação do número de clusters pretendido, nesse aspeto é vantajoso utilizar este método pois poderemos extrair diferentes padrões em casos que não saibamos à partida quantos clusters poderão existir.

Este método constrói uma espécie de árvore chamada dendograma, construída a partir das observações que são representadas como folhas no dendograma, combinando-as em pequenos clusters iterativamente até chegar à raiz.

Quando esta árvore começa a ser construída inicialmente começam-se por fundir as observações mais similares, ou seja, as folhas. À medida que se vai subindo na árvore começamos a ver ramos a combinarem-se o que significa que os grupos correspondentes são similares. Este processo continua até todos os pontos e grupos estarem conectados entre si. Pode então ser dito que quanto mais abaixo da árvore encontrarmos uma ligação mais similares são as observações constituintes dessa ligação e quanto mais acima menos similares pois o resultado deste método retorna sempre um cluster único constituído por todas as observações.

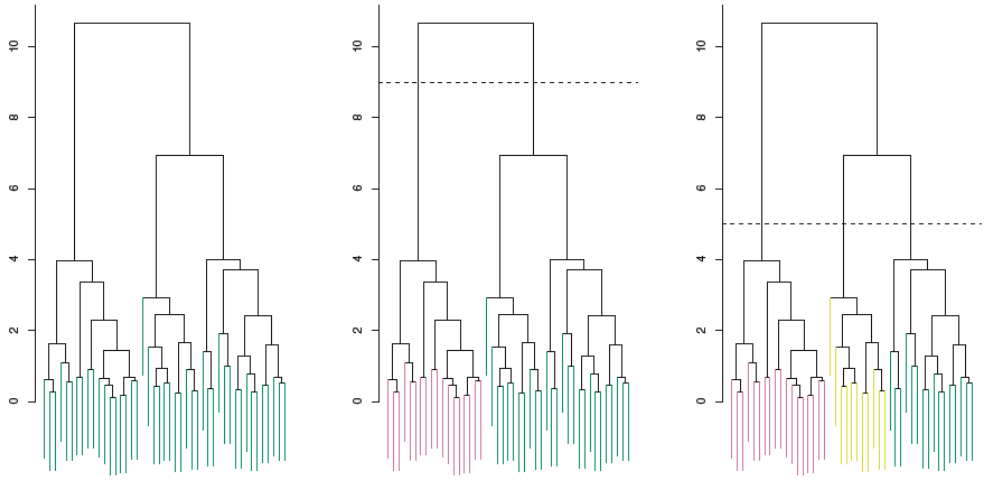

Figura 13 - Exemplo de 3 dendogramas com 3 pontos de corte diferentes [17]

Compreendido então o funcionamento e representação do Clustering Hierárquico devemos então definir o ponto de corte a aplicar no dendograma gerado. Este ponto de corte será escolhido como representado na Figura 13 com uma linha horizontal no dendograma definindo como clusters todos os conjuntos conectados abaixo dessa linha. Este ponto de corte irá definir o número de clusters a obter no final do processamento tendo assim agrupado todas as observações. O seguinte algoritmo é utilizado para construir o dendograma:

1. Iniciar com *n* observações e todas as medidas $(n_2) = \frac{n(n-1)}{2}$ de similaridade. Cada observação nesta fase é considerada o seu próprio cluster.
2. Iterar $i = n, n-1, \dots, 2$:

a. Calcular as distâncias entre pontos inter-cluster e identificar o par de clusters mais similar entre si (dependendo da métrica de similaridade utilizada). Agregar os clusters identificados. A distância entre esses clusters é a altura a que aparece no dendograma.

b. Calcular as novamente as distâncias entre cada cluster assumindo que existem i-1 clusters restantes.

As mais comuns métricas de similaridade utilizadas no Clustering Hierárquico são as seguintes:

1. *Complete Link:* Similaridade máxima inter-cluster. São calculadas todas as distâncias entre os pares de observações de um Cluster A e das observações de um Cluster B. É selecionada a maior distância para a ligação
2. *Single Link:* Similaridade mínima inter-cluster. São calculadas todas as distâncias entre os pares de observações de um Cluster A e das observações de um Cluster B. É selecionada a menor distância para a ligação
3. *Average Link:* Similaridade média inter-cluster. São calculadas todas as distâncias entre os pares de observações de um Cluster A e das observações de um Cluster B. É selecionada a média da distância para a ligação
4. *Centroid:* É assumida a distância entre os atuais centroides de cada *cluster.*

Todos os tipos de *clustering* são válidos e úteis para tarefas específicas desde problemas na área da biologia até ao mercado financeiro. Em todas as áreas existem diversos tipos de padrões para serem identificados, e para isso são utilizadas muitos outros tipos de algoritmos não supervisionados como por exemplo o PCA referido anteriormente, ou o DBSCAN [32].

3. ENSEMBLES

Até agora é percetível que os vários métodos de aprendizagem automática existentes, utilizam um conjunto de dados para construir um modelo para resolver um determinado problema. Neste capítulo é introduzido o conceito de aprendizagem de grupo ou aprendizagem por *ensemble*, técnica que pretende construir vários modelos e agregá-los de determinado modo para resolver um problema.

Em 3.1 é introduzido o conceito de *Ensemble* sistematizando alguns termos e conceitos. De seguida em 3.2 são explorados alguns métodos de combinação utilizados em *Ensembles*. São então exploradas algumas técnicas mais específicas nos próximos subcapítulos como o AdaBoost em 3.3, o Bagging em 3.4, Meta-*Ensemble* em 3.5 e Random Forests em 3.6.

3.1 - Ensemble

Um Ensemble é um modelo de aprendizagem automática que combina a previsão de vários modelos de maneira a dar uma única previsão. Na seguinte Figura 14 vê-se uma arquitetura comum para um *Ensemble*.

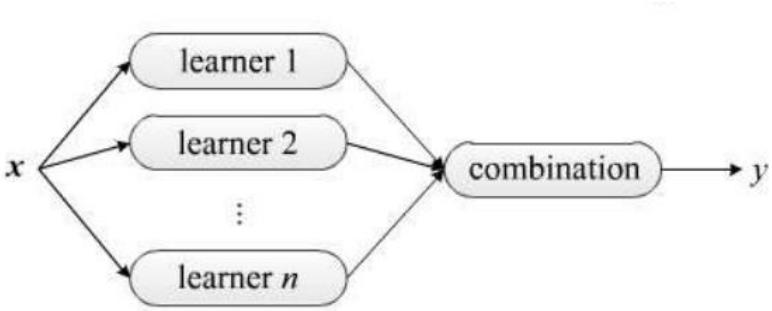

Figura 14 - Arquitetura Básica de um *Ensemble* em que o treino de cada *base learner* é realizado em paralelo. [33]

Um *Ensemble* contém um conjunto de modelos chamados *base learners* que são gerados a partir dos dados do *dataset* de treino por um algoritmo base como uma árvore de decisão, rede neuronal ou outro. Os *base learners* podem ser chamados de *weak* ou *strong learners*. Um *weak learner* é um modelo que é apenas um pouco melhor que a escolha aleatória, em termos gerais é um modelo fraco que sozinho não terá grande impacto em qualquer problema que tente resolver [34]. Tipicamente um *Decision Stump* (é específico para árvores de decisão) é a representação básica e simples de explicar de um *weak learner,* trata-se de uma pequena árvore de decisão apenas com apenas um nível de profundidade com uma capacidade de generalização muito reduzida. Um *strong learner* é um modelo que utiliza determinado algoritmo e que possui uma taxa de acerto maior que que uma escolha aleatória. Por exemplo, uma árvore de decisão simples com uma taxa de acerto de 90% pode ser considerada um *strong learner*.

Um *Ensemble* que utilize um conjunto de modelos com o mesmo algoritmo base é considerado um *Ensemble* Homogéneo, se possuírem diferentes algoritmos é chamado de *Ensemble* Heterogéneo. Se cada *base learner* for uma árvore de decisão é considerado um *Ensemble* Homogéneo. No entanto, se um dos *base learners* for uma Regressão Linear o *Ensemble* é considerado um *Ensemble* Heterogéneo.

A utilização do *Ensemble* traz um grande custo computacional, tempo e complexidade. Isto porque é necessário treinar e manter todos os *base learners* constituintes. Por sua vez, a utilização de *Ensembles*, como se pode confirmar na literatura [1], tem um melhor desempenho que a utilização de um *learner* simples.

De realçar que os *Ensembles* para além de possuírem taxas de acerto mais elevadas [35] que outras técnicas de aprendizagem automática clássica, possuem uma maior robustez reduzindo a variância e o *bias* dos resultados obtidos[36][37]. A variância é a medida de dispersão dos resultados obtidos e *bias* a medida de tendência do *dataset.*

Tipicamente quanto mais diversificados forem os *learners* constituintes do *Ensemble* melhor é o modelo gerado[30]. Isto deve-se ao facto de pôr todos *learners* trabalharem em conjunto, conseguem cobrir de uma melhor forma o espaço de procura que por sua vez irá levar a melhores resultados.

Na literatura [38][39], a diversidade é descrita como a medida em que diferentes modelos discordam onde cometem erros, ou seja, como exemplo numa classificação binária, em que se possui um *dataset* de validação com seis exemplos. É construído um *Ensemble* com três *base learners* do tipo *Decision Stump,* o primeiro *learner* apenas consegue prever os primeiros quatro exemplos corretamente, o segundo *learner* consegue apenas prever os quatro últimos exemplos corretamente, já o terceiro *learner* prevê corretamente os dois primeiros e os dois últimos exemplos do *dataset*. No exemplo descrito anteriormente pode dizer-se que o *Ensemble* construído possuí uma grande diversidade pois cada um dos *base learners* possuí a mesma capacidade de previsão, mas para exemplos diferentes do *dataset,* favorecendo o resultado agregado do *Ensemble*.

3.2 - Métodos de Combinação

Há vários métodos válidos para combinar cada um dos *learners* de um determinado *Ensemble*, uns mais robustos outros com um melhor desempenho. Neste caso diz-se que um método de combinação é robusto quando este é capaz de lidar com a grande variância dos outputs dos *base learners* e com *outliers.* Por exemplo, um *Ensemble* que resolve um problema de regressão pode utilizar a média para agregar os *base learners,* no entanto se algum dos *outputs* resultar num valor infinito o output do *Ensemble* será também ele esse valor. Daí que para casos como este é utilizada a mediana que é considerada uma medida mais robusta.

O método mais simples e mais usado [1] para problemas de regressão em que o *output* é um valor numérico é a média. Uma simples média dos outputs de cada *base learner* será o *output* dado pelo *Ensemble.*

$$Ensemble(X) = \frac{1}{n}\sum_{i=1}^{n} L_i(X)$$

Equação 7 – *Ensemble* utilizando o somatório

Acima representada a equação que agrega os *bases learners* para obter o output de um *Ensemble* utilizando a média aritmética para um problema de regressão. A variável *X* representa o conjunto de inputs, *n* representa o número de *base learners* que constituem o *Ensemble* e *L* representa o *output* de um *base learner*. Em alguns casos para regressão é utilizada a média ponderada onde é associado um peso a cada modelo que pode ser atribuído automaticamente utilizando princípios de *Boosting* Cap. 3.2.

Para problemas de classificação são utilizados métodos de votação. Como votação da maioria em que é escolhido como output do *Ensemble* o *output* mais frequente de cada *learner* constituinte.

$$Ensemble(X) = \text{mode}\,(L)$$

Equação 8 – *Ensemble* utilizando a moda

Neste caso também pode ser realizada uma votação ponderada onde cada *base learner* tem algum tipo de peso associado que mostra o quanto contribui esse *Ensemble* para a previsão final. Esse peso pode por exemplo ser a *Accuracy* desse base learning, dando mais importância a base learners mais, mais exatos. Podem ser encontradas na literatura [40] [30] imensas técnicas de agregação, umas mais complexas que outras.

3.3 - AdaBoost

O *Boosting* é uma técnica proposta em [41], que parte do princípio que é muito mais simples para um determinado problema identificar vários regras ou padrões simples no *dataset* do que identificar uma única regra muito precisa. O AdaBoost é um algoritmo que utiliza os princípios básicos do *Boosting*. É um algoritmo que tenta encontrar pequenas regras iterativamente, cada uma das regras geradas tem o nome de *weak prediction* ou previsão fraca, a cada iteração o *input* dos modelos é sempre um subconjunto diferente do *dataset* original, ou uma distribuição diferente dos exemplos com algum tipo de ponderação associada a cada exemplo. Após *n* iterações todos os modelos gerados terão de ser agregados para obter o *output* do *Ensemble* utilizando técnicas referidas anteriormente.

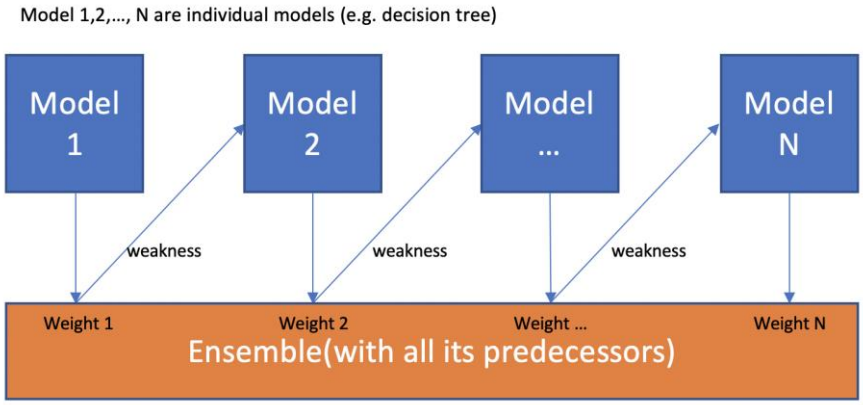

Figura 15 - Estrutura de um *Ensemble* Boosting [33]

A Figura 15 representa a estrutura básica de um *Ensemble* construído utilizando o *Boosting*. É então escolhido um subconjunto aleatório utilizando uma amostragem com repetição baseada no *dataset* original e é treinado o primeiro o modelo com esse subconjunto. O teste desse modelo é realizado com todo o conjunto de treino original, isto fará com que alguns dos exemplos tenham um erro considerável. É então associada uma ponderação a esses exemplos com maior erro com o objetivo de aumentar a probabilidade de serem selecionados para a construção do próximo *base learner,* na Figura 15 representado na seta *weakness*. Utilizando este processo aumenta a probabilidade de cada modelo subsequente consiga lidar melhor com problemas específicos que os anteriores não conseguiram lidar.

Este processo é repetido para todos os N modelos constituintes e por fim são agregados os outputs de cada um dos N modelos com um peso associado ao erro que cada um produziu no processo de treino para obter o *output* do *Ensemble*.

O *Boosting* consegue identificar iterativamente exemplos mais difíceis de prever é claro que qualquer tendência que existir nos dados será dissolvida, ou seja, a utilização do *boosting* é ideal quando cada modelo constituinte possui uma baixa variância e um alto *bias*.

3.4 - Bagging

O *Bootstrap Aggregating,* ou mais conhecido por *Bagging* é um método agrega vários modelos utilizando uma técnica chamada *bootstraping* para gerar diferentes *datasets* para cada modelo constituinte do *Ensemble* ser treinado [42][40].

O *Bootstraping* é um método de amostragem com repetição em que é gerado um novo *dataset* baseado no original onde podem existir exemplos do repetidos. A este novo *dataset* é dado o nome de *boostraped dataset*. Este processo é repetido N vezes dependendo do número de *base learners* que se pretende criar.

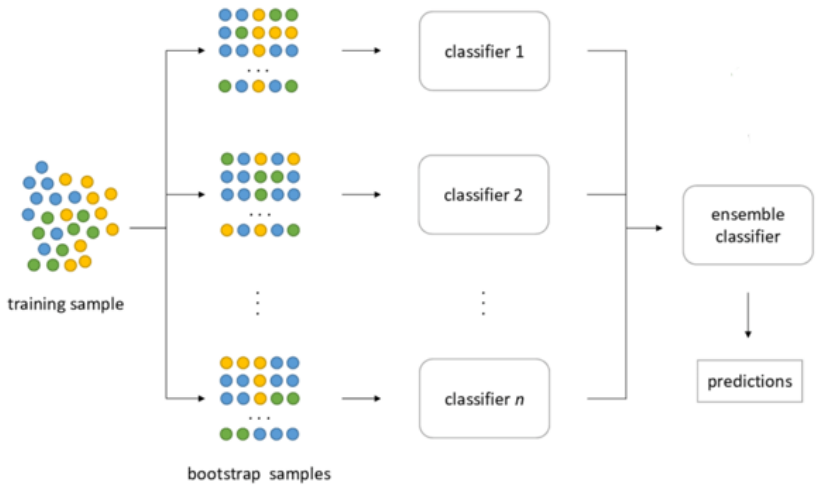

Figura 16 – Bagging e um conjunto n de classificadores

A Figura 16 pretende demonstrar a estrutura base de um *Ensemble* utilizando o *Bagging*. O *Ensemble* é constituído por N modelos, em que cada um dos modelos é treinado com um *boostraped dataset* diferente. Tal e qual como os outros métodos, são posteriormente agregados os *outputs* de cada um dos *base learners* para obter o resultado do *Ensemble*.

A utilização do Bagging tende a melhorar mais a variância dos resultados que o *bias*, ou seja, é aconselhável a sua utilização quando cada *base learners* possui pouco uma maior variância e um baixo bias.

3.5 - Meta-Ensemble

A generalização em *Stack* ou *Stacking,* proposta em [35], é uma técnica que agrega diversos modelos com o principal objetivo de reduzir o erro de generalização. A principal diferença entre o *Stacking* e os métodos mencionados anteriormente é o método de agregação em que neste caso é ele próprio um modelo, chamado de meta modelo. Ou seja, é criado um modelo que recebe como input o *output* de todos os *base learners* constituintes do *ensemble* e o seu *output* será o *output* do *Ensemble*.

A seleção do *dataset* de treino para cada *base learner* no Stacking é feita de utilizando princípios do *Cross Validation.* Em que o dataset é divido N *folds,* sendo que N é o número de *base learners,* treinando cada *base learner* com diferentes partes do *dataset*. De seguida são utilizados os *outputs* dos N *base learners* como *input* do meta modelo.

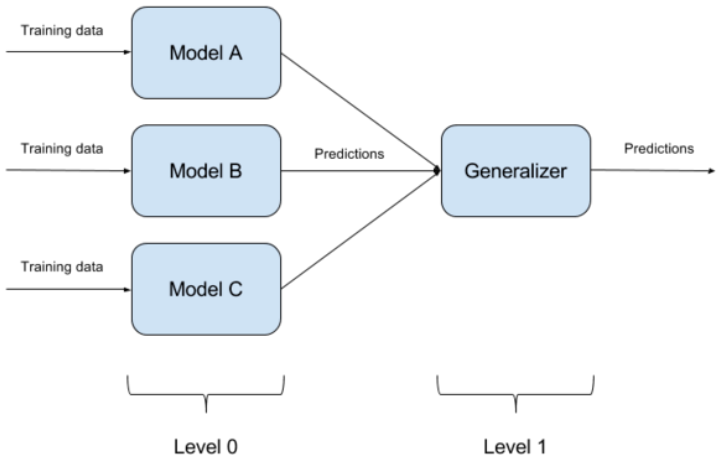

Figura 17 – Stacking , sendo que Model A, Model B e Model C são os base learners e o Generalizer é o meta modelo [33]

A Figura 17 acima ilustra a estrutura base de um *Ensemble* utilizando *Stacking*. Cada *base learner* é treinado com dados de treino e cada previsão é utilizada como *input* da meta modelo resultando assim no *output* do *Ensemble*.

O Stacking é uma técnica versátil no que diz respeito à seleção de *datasets* de treino, pois esta pode ser feita de diversas formas desde utilização de métodos iterativos como o *Boosting* ou um *bootstrapping* como é feito no *Bagging.* Ou seja, o *Stacking* consegue produzir resultados que tendem a aumentar o poder de generalização e como podem possuir alguns atributos de outros métodos de *ensembling* também podem reduzir a variância e *bias* das previsões.

No entanto, a utilização do *stacking* exige um grande processamento e gera modelos muito complexos e com pouca explicabilidade. Neste caso para além de ser necessário treinar e manter os *base learners*, tal e qual como as outras técnicas, é também necessário treinar e manter o meta modelo. Pode também tornar-se muito difícil explicar o *output* pois a complexidade do modelo gerado é extremamente alta.

3.6 - Random Forests

Dos métodos mais poderosos na aprendizagem por *ensemble* são as *Random Forests.* Este tipo de *ensemble* foram introduzidos em [43] como evolução dos métodos existentes à altura de *ensemble* que utilizavam *Bagging* e *Boosting* para melhorar modelos simples que utilizavam árvores de decisão.

As *Random Forests* são *ensembles* em que os *base learners* são árvores de decisão. Estas árvores utilizam amostras aleatórias do *dataset* original utilizando técnicas como *bootstrapping,* mas também é realizada uma seleção estocástica de *features* a ter em conta para cada *dataset,* são também realizadas escolhas estocásticas ao nível do nó da árvore de decisão pois cada uma destas é construída baseada em métodos já referidos. Estas escolhas reduzem a correlação entre os *base learners*, resultando em

modelos diversificados que analisam diferentes condições e locais do *dataset*. Esta diversificação é normalmente benéfica como tem vindo a ser referido no documento.

Para finalizar existe uma função que agrega os resultados utilizando técnicas referidas anteriormente como a votação por maioria ou apenas uma média em caso de regressão [44]. Na Figura 18 está a arquitetura básica de uma instancia deste algoritmo.

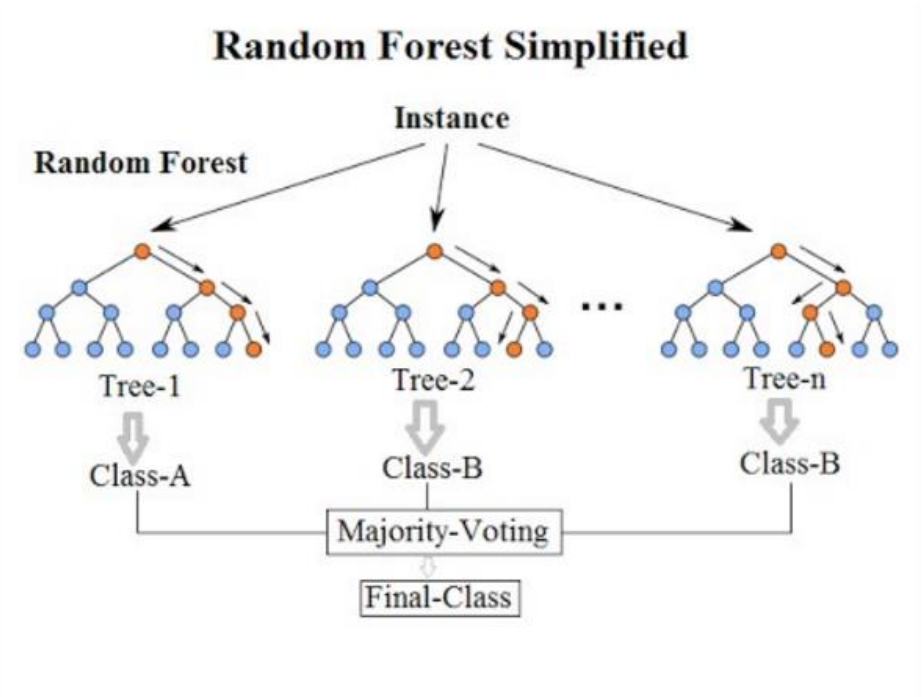

Figura 18 - Estrutura básica de uma Random Forest [33]

Comparando o *base learner* e o *Ensemble*, Uma árvore de decisão simples tende a fazer *overfitting* com os dados de treino no entanto a agregação de várias árvores como demonstrado em [43] e [33] tende a melhorar a performance.

O conceito de *Random Forest* possui uma estrutura relativamente modular e versátil quanto à maneira como os processos de *bootstrapping* ou de combinação são efetuados. Existem estratégias diferentes para criar *random forests* como adotar o Boosting, tentando criar cada árvore com princípios do *boosting* entre outros como pode ser visto em [30] no capítulo de *Decision Forests.* São aplicados os métodos referidos anteriormente de agregação de modelos como outros métodos de votação para classificações ou utilização da média ou mediana dos *outputs* das *decision trees* constituintes para definir o resultado de uma *random forest*.

As *Random Forests* são métodos que têm uma *performance* elevada comparado a outros métodos de aprendizagem automática. No entanto, as árvores de decisão sofrem dos problemas gerais de um *Ensemble* como poder tornar-se lento devido a vários *learners* terem de ser mantidos, e também podem possuir *base learners* extremamente complexos com árvores muito grandes e com muitos parâmetros.

4. EVOLUÇÃO AUTOMÁTICA DE PROGRAMAS

Este capítulo tem como objetivo introduzir uma área muito específica da aprendizagem automática chamada de Evolução Automática de Programas. Aqui estão introduzidos alguns conceitos de otimização, algumas das técnicas mais utilizadas nesta área e por fim uma análise ao algoritmo em que a proposta deste trabalho é baseada.

4.1 - Otimização

A otimização é uma área da Matemática podendo ser descrita como a escolha do melhor elemento, baseada em algum critério, num certo espaço de procura. Integrando este conceito na ciência da computação é possível desenvolver algoritmos e técnicas de otimização para resolver problemas complexos.

É possível dividir algoritmos de otimização em dois grandes grupos, os algoritmos exatos e algoritmos heurísticos. Os algoritmos exatos garantem uma solução ótima para os problemas, no entanto para determinados problemas onde o espaço de procura é demasiado grande e complexo podem não ser apropriados devido ao processamento necessário para os resolver.

Quando não compensa utilizar algoritmos exatos, são utilizados algoritmos heurísticos que normalmente não garantem uma solução ótima, no entanto são tipicamente muito mais rápidos que métodos exatos.

4.2 - Algoritmos de Melhoramento Iterativo

Algoritmos de melhoramento iterativo tentam de certo modo partir a resolução do problema principal em pequenos passos, ou seja, inicialmente produzem por exemplo uma solução utilizando métodos estocásticos e iterativamente vão melhorando essa mesma solução até atingir uma solução ideal. A grande desvantagem destes algoritmos é que não garantem a melhor solução possível, ou também chamado de ótimo global.

Todos os algoritmos heurísticos têm um conjunto de conceitos básicos [45]. Uma delas é a escolha da representação das soluções, esta escolha irá afetar e por sua vez definir o espaço de procura, sendo este o conjunto de todas as soluções para resolução do problema.

Para saber como é que uma solução é melhor que outra é necessário definir também uma função de avaliação e função objetivo. É necessário perceber a qualidade de uma solução para ser possível melhorá-la e também qual o objetivo que se pretende atingir. Por exemplo, um problema em que a representação de cada solução é um

vetor de inteiros pode ter como função de avaliação a soma de todos os seus elementos e a função objetivo é soma ser maior que 1000.

Todos os algoritmos de melhoramento iterativo partem de uma solução completa, isto é, uma solução que pode ser utilizada à partida e não necessita de ser construída ao longo da execução.

Para o melhoramento acontecer devem ser efetuadas algumas transformações às soluções selecionadas em cada iteração. Para resolver um determinado problema em que a representação de cada solução é um vetor binário, uma possível pequena transformação seria inverter um bit do vetor gerando assim um vizinho da solução atual. Ao conjunto de todos os possíveis vizinhos de uma solução dá-se o nome de vizinhança. Existem diversos operadores de transformação como se pode ver na literatura [46].

O *Hill-Climbing* é algoritmo de melhoramento iterativo que implementa todos os princípios básicos referidos até agora num problema de otimização. É um algoritmo que começa por selecionar um ponto aleatório do espaço de procura e seleciona a melhor solução da vizinhança para a próxima iteração.

Este processo continua até nenhuma das soluções da vizinhança for melhor que a atual. Tendo em conta que o *Hill-Climbing* escolhe sempre melhores soluções é frequente o mesmo ficar preso em ótimos locais. Existem outros algoritmos de otimização que podem ser analisados na literatura [46] como *Simulated Annealing* ou um algoritmo evolucionário analisado mais à frente. O algoritmo base de um *Hill-Climbing* pode ser descrito da seguinte forma:

1. Selecionar uma solução aleatoriamente
2. Avaliar resultado da solução atual *a,* com a função de avaliação *f*
3. Iterar enquanto existirem vizinhos v_i, onde $f(a) < max(f(v_i))$
    a. Aplicar operador de vizinhança e gerar vizinho *v*
    b. Se $f(a) < f(v)$, a = v

Este tipo de algoritmos é utilizado para resolver problemas como o *Traveling Salesman Problem (TSP)* que é um problema que dada uma lista de cidades e as distâncias entre cada uma das cidades pretende descobrir qual o itinerário mais curto partindo de uma cidade origem e percorrendo todas as outras cidades uma única vez até voltar à cidade de origem.

Para este problema podemos representar uma solução como um ciclo hamiltoniano que agregue todas as cidades conectadas entre si em que cada nó é uma cidade e cada aresta guarda a distância entre os dois nós que conecta.

Depois de escolhido o espaço de procura é necessário definir o objetivo do problema, isto significa que é necessário definir a expressão matemática que vai avaliar uma

solução, por exemplo, no caso do *TSP* e tendo em conta que se *pretende* minimizar o somatório das distâncias, podemos definir a função objetivo como:

$$\min \left(\sum dist(x, y) \right)$$

Equação 9 - Objetivo do TSP

A função objetivo apresentada significa que se pretende obter o valor mínimo para o somatório de todas as distâncias entre os pontos a visitar. Na fórmula *x* representa uma das cidades e *y* a cidade a subsequente na solução. A função *dist* recebe duas cidades e retorna a distância entre os pontos.

O *TSP* é um problema difícil de resolver em que têm de ser utilizados algoritmos heurísticos para obter resultados em tempo útil, mesmo não sendo resultados ótimos.

4.3 - Algoritmos Evolucionários

Dentro dos algoritmos de melhoramento iterativo existe um subconjunto de algoritmos chamados de algoritmos evolucionários. Estas técnicas são inspiradas nos processos evolucionários biológicos [6].

Utilizando um algoritmo evolucionário como exemplo, o início de uma execução é marcado pela criação de um conjunto de soluções candidatas para a resolução do problema chamado de população. São tipicamente utilizados métodos estocásticos para gerar a população [4]. Serão referidos mais à frente no documento alguns métodos específicos para gerar populações iniciais.

É também definida a função de avaliação ou *fitness function,* é a função que irá avaliar a aptidão (ou *fitness*) de uma solução em particular. O valor do *fitness* de cada indivíduo pode ser determinante para a sua sobrevivência ao longo da execução. A cada iteração é guardado o melhor indivíduo global para no final ser utilizado para resolver o problema e questão [4].

O processo de seleção é tipicamente estocástico, no entanto existem abordagens diferentes como a utilização de um limite do *fitness*. Por exemplo é definido um limite para o *fitness,* em que se o indivíduo não atingir esse valor não sobreviverá e será descartado.

Outro exemplo de seleção é a seleção por torneio que também é um método estocástico que avalia um par de indivíduos um contra o outro e compara o seu *fitness* selecionando o melhor. De notar que a seleção por torneio não tem em consideração o quão melhor é uma solução perante outra, ou seja, se um dos indivíduos for melhor que outro é selecionado mesmo que seja por uma mínima diferença. Em existem inúmeros outros métodos como o a seleção por elitismo ou por tamanho de vizinhança como pode ser visto em [47].

Um desses métodos é o *Crossover* ou Recombinação, método que combina a informação genética de dois indivíduos originando assim indivíduos descendentes. A

informação genética de cada indivíduo depende da sua representação, por exemplo se um indivíduo for representado por um vetor de números a sua informação genética são os números que compõem o individuo. Existem vários tipos de *crossover* como o *one-point crossover* ou o *two-point crossover* que podem ser consultados na literatura [48].

Outro método é a Mutação, é a operação de alteração espontânea de um ponto genético ou gene. No exemplo de um indivíduo ser representado por um vetor de números, uma possível mutação seria a alteração de um dos números gerando um descendente com um gene diferente do indivíduo anterior.

Ambos os métodos referidos anteriormente têm uma probabilidade de acontecer ao longo de uma execução e essa probabilidade é um dos parâmetros iniciais da execução de um algoritmo evolucionário.

É também necessário definir uma condição de paragem, que por sua vez previne a execução infinita do algoritmo. Tipicamente um dos parâmetros definidos inicialmente é o número de gerações que define *a priori* quando é que o algoritmo irá parar a execução, no entanto podem ser definidas outras condições como atingir o melhor *fitness* possível num dos indivíduos ou outros tipos de abordagens que podem ser explorados na literatura [49].

Um exemplo de um algoritmo evolucionário é ilustrado na seguinte descrição de alto nível:

1. Inicialização da população: *populaçãoAtual*
2. Aplicar a função de avaliação *f* a todos os indivíduos da *populaçãoAtual*
3. Guardar a melhor solução da *populaçãoAtual: melhorSolução*
4. Iterar enquanto não se atingir *N* gerações
    a. Seleção dos pares para aplicação de operações genéticas
    b. Iterar sobre todos os pares: *pai1, pai2*
        i. Aplicação do Crossover entre o *pai1* e *pai2*: *filho1* e *filho2*
        ii. Aplicar operação de mutação nos *filho1* e *filho2*
        iii. Guardar *filho1* e *filho2* para a nova geração: *novaGeração*
    c. Aplicar a função de avaliação *f* a todos os indivíduos da *novaGeração*
    d. Se a melhor solução da *novaGeração* for melhor que *melhorSolução*
        i. Substituir *melhorSolução* pela melhor solução da *novaGeração*
    e. *populaçãoAtual* = *novaGeração*
5. Devolver *melhorSolução*

Este algoritmo Necessita de alguns parâmetros iniciais como por exemplo, tamanho da população, número de iterações/gerações e taxas de recombinação e mutação.

4.4 - Genetic Programing

A evolução automática de programas é especificamente focada na evolução de programas também chamada de síntese de programas. De seguida são apresentadas algumas técnicas de evolução de programas.

Genetic Programming(GP) é uma técnica de computação evolucionária que resolve automaticamente um problema sem que seja necessário especificar a forma ou estrutura da solução a priori. Tipicamente a representação de um algoritmo evolucionário simples é de tamanho fixo e complexidade limitada, no caso de GP e outros métodos de evolução automática de programas é definido um conjunto de regras que permite haver estruturas dinâmicas ao longo da execução. Para além de que em GP devido a essa definição inicial de regras os resultados inválidos são muito raros e normalmente acontecem por má definição dessas mesmas regras.

Em GP é evoluído um conjunto de programas de computador, tal e qual como num algoritmo evolucionário normal melhorando os indivíduos de geração em geração. Não é garantido o melhor programa para resolver o problema como qualquer outro algoritmo evolucionário. A seguinte figura mostra o funcionamento geral do GP.

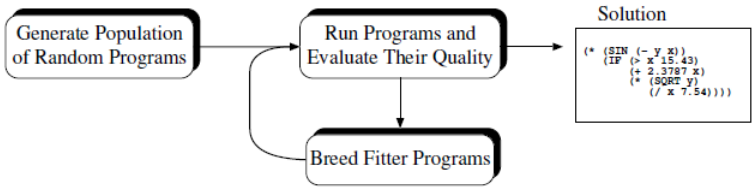

Figura 19 - Diagrama do funcionamento interativo de GP [50]

Na Figura 19 no retângulo mais à esquerda é o primeiro passo, o de gerar a população inicial de programas, de seguida são executados os programas para produzirem o resultado esperado do mesmo e são avaliados segundo uma função objetivo, por exemplo é necessário criar um modelo que aproxime uma equação conhecida como um polinómio de segundo grau $f(x) = x^2 + x + 1$. Um programa é uma qualquer função que receba um *x* e que produza um *output* e a função de avaliação pode ser a diferença entre o *output* do polinómio e o *output* do programa gerado em que objetivo é minimizar esta diferença. São então gerados novos programas aplicando os conceitos evolucionários com o objetivo de gerar melhores indivíduos por sua vez na figura à esquerda surge a solução provinda do melhor programa gerado na execução.

Para qualquer problema deste tipo é necessário existir uma representação de uma solução, um conjunto de funções e de terminais. O conjunto de funções é o conjunto de operações possíveis de realizar como por exemplo operações aritméticas (+, -, *,

/) por sua vez o conjunto de terminais é o conjunto de variáveis, constantes ou funções sem argumentos.

Sabe-se que para serem resolvidos por GP uma possível solução é um programa de computador, no entanto, é utilizada usualmente a representação em árvore sintática em vez de linhas de código. A seguinte figura mostra, por exemplo, como pode ser representado o programa $max(x + x, x + 3 * y)$.

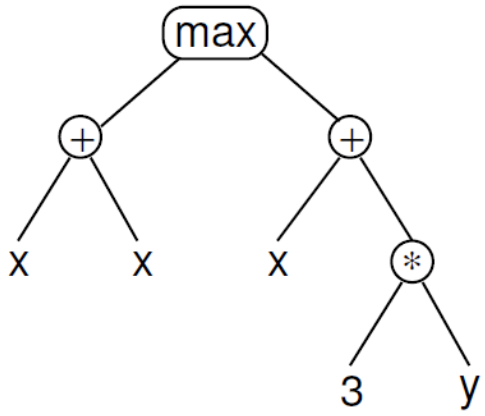

Figura 20 - Árvore sintática do programa $max(x + x, x + 3 * y)$[50]

Na Figura 20 as variáveis e constantes x, y e 3 representados como folhas da árvore tem o nome de nós terminais, no caso das operações aritméticas +, * e *max* são o conjunto de funções que estão representados pelos nós internos da árvore, ambos funções e terminais formam o *primitive set*.

As árvores sintáticas são muito boas no que diz respeito à visualização e explicabilidade de soluções, no entanto em muitos casos tornam-se ineficientes devido ao espaço que ocupam em memória e ao seu processamento em casos complexos. Nas seguintes secções temos outras formas de evolução automática de programas que aceitam e utilizam outras representações e serão mais aprofundadas.

A inicialização da população é normalmente feita de um modo aleatório, no entanto, existem diferentes métodos de gerar a população. Este passo tem algum impacto no desempenho de qualquer tipo de algoritmo pois sendo um algoritmo de melhoramento iterativo o início da pesquisa pode levar-nos à solução ótima ou apenas a um ótimo local.

Dois dos métodos utilizados para a inicialização da população, o método *full* e o método *grow*. Em ambos o caso é necessário especificar a profundidade máxima, a profundidade do nó é o número de ramos que necessitam de ser passados para chegar da raiz (profundidade 0) a esse nó, ou seja, o caminho até à folha. A profundidade da árvore é o valor da profundidade da folha mais profunda da árvore.

O método *full* como o nome indica gera apenas árvores completas, ou seja, árvores em que todas as folhas têm a mesma profundidade. Os nós são gerados aleatoriamente a partir do conjunto de funções até que a profundidade especificada

seja atingida, aí são gerados também aleatoriamente folhas dos nós finais a partir do conjunto de terminais. Apesar do método *full* gerar árvores com a mesma profundidade não significa que vá gerar árvores com a mesma estrutura ou tamanho pois esta depende da aridade das funções utilizadas. Este método pode gerar pouca variedade de árvores o que pode ser limitado em determinados contextos.

O método *grow* ao contrário do método *full*, pode gerar árvores com diversas formas e tamanhos. Em vez de utilizar apenas o conjunto de funções até à profundidade especificada este método utiliza todo o *primitive set* (funções e terminais). Ou seja, se for escolhido um terminal para ser colocado em determinada posição é automaticamente considerada uma folha da árvore.

Em [51] é proposto um método que tenta combinar ambos os métodos propondo o *ramped half-and-half* onde metade da população é criada utilizando o método *full* e outra metade utilizando o método *grow*, que tenta assegurar que as árvores geradas para a população inicial são variadas com tamanhos e formas diferentes.

Como em todos os algoritmos evolucionários também existem operadores genéticos em GP que são aplicados a cada indivíduo com base no seu *fitness*. Os principais operadores genéticos utilizados neste tipo de métodos são o *crossover* e a mutação, no entanto devido à representação dos indivíduos têm de ser feitas algumas alterações aos mesmos.

Das mais utilizadas formas de *crossover* em GP é o *subtree crossover*, onde são dados dois pais e é escolhido aleatoriamente um ponto de *crossover* em cada indivíduo, pontos esses que dizem respeito a nós na respetiva árvore. Estes pontos são as raízes de subárvores que serão trocados formando assim indivíduos descendentes de ambos os programas pai.

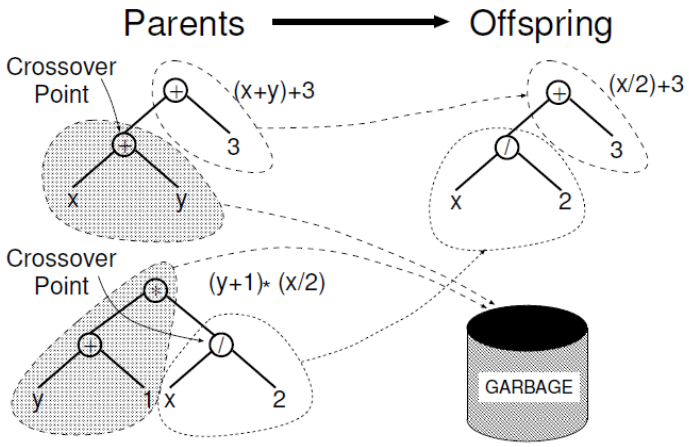

Figura 21 – Exemplo do funcionamento do Subtree Crossover.[50]

A Figura 21 demonstra este processo e como são agregadas as duas árvores para formarem a descendência, à esquerda estão os indivíduos que representam os dois pais com o respetivo ponto de crossover, e que são posteriormente agregados para formar o indivíduo descente à direita. De notar que os indivíduos pai representados são cópias dos originais daí que o resta da árvore original pode ser perdido.

O operador de mutação também é aqui utilizado e tal e qual como o de recombinação existem várias técnicas para o aplicar, sendo que os mais comuns são o *subtree mutation* e o *point mutation*. O *subtree mutation* tal e qual como o *subtree crossover*, é aleatoriamente selecionado um ponto de mutação e a subárvore que tem como raiz esse ponto é removida e é gerada uma nova árvore de forma aleatória para a substituir.

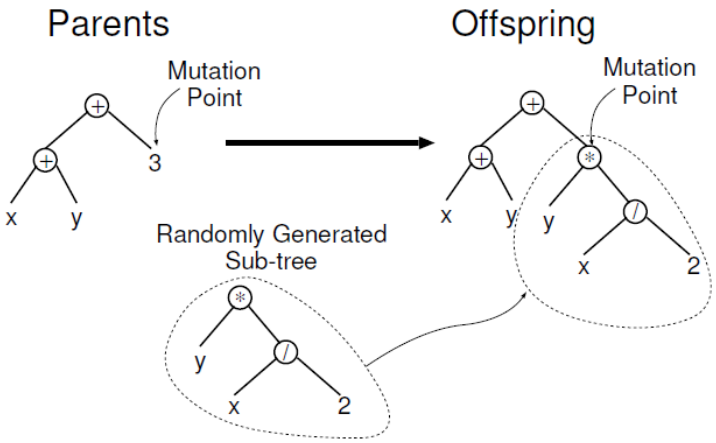

Figura 22 - Exemplo do funcionamento do Subtree Mutation [50]

Na Figura 22 à esquerda está um indivíduo selecionado para ser mutado, de seguida é escolhido o *mutation point,* é também gerada aleatoriamente uma subárvore e por sua vez essa árvore é imputada no indivíduo pai dando origem a um novo indivíduo.

Já no *point mutation* é também escolhido um ponto de mutação aleatoriamente, no entanto é apenas alterado o valor do nó selecionado por um valor do *primitive set* com a mesma aridade. Ou seja, se o nó selecionado tiver como valor +, é uma função do *primitive set,* que por sua vez tem aridade dois, apenas pode ser substituído por uma função equivalente também ela com aridade dois. O *point mutation* pode ser comparado ao *"bitflip"* sendo que é apenas alterado um valor em toda a árvore.

A grande diferença de implementação dos operadores genéticos em GP perante outros algoritmos evolucionários é a característica de estes poderem ser mutuamente exclusivos. Tipicamente, o operador de crossover tem uma probabilidade maior por volta dos 90% já o operador de mutação tem um valor muito mais pequeno, de 1% ou menos dependendo de outros parâmetros. Estas probabilidades chamam-se de *operator rates*, quando a soma de ambas as *rates* de mutação e *crossover* não resultam em 100% é aplicado outro operador chamado de reprodução, que é simplesmente a seleção de um indivíduo e a passagem do mesmo para a gerações seguintes [50].

4.5 - *Grammatical Evolution*

Outra variante de algoritmos de evolução automática de programas é a *Grammatical Evolution* (GE) que é uma forma de algoritmo evolucionário que possui um vetor de

inteiros como representação dos indivíduos, que por sua vez são mapeadas com ajuda de uma gramática para gerar um programa executável. Uma das vantagens perante *PG* é a utilização desta gramática pois faz com que o espaço de procura utilizado seja mais restrito ao domínio do problema a resolver, pois como pode ser visto de seguida neste capítulo a gramática possui um conjunto de regras que dizem respeito ao domínio do problema.

As gramáticas podem ser implícitas ou explícitas. Sendo que gramáticas implícitas são gramáticas utilizadas de forma subentendida, como acontece no caso de GP, em que a representação dos indivíduos é feita com árvores e não necessário explicitamente definir a gramática ou regras pois a morfologia da árvore automaticamente define a estrutura. No caso de várias técnicas de GE é necessário definir explicitamente a gramática a utilizar.

Em GE são utilizadas *Context-Free Grammars (CFG)* [6]. As CFG são utilizadas para mapear o genótipo de um indivíduo, que contém todas as especificações entre terminais e não-terminais. Sendo que terminais representam itens que irão aparecer na estrutura final, e não terminais representam normalmente os operadores que combinam os terminais.

A *Backus Naur Form* (BNF) [52] trata-se de uma notação para exprimir a gramática de uma linguagem num formato de regras produção. O tuplo {N, T, P, S} define a gramática a utilizar, onde N representa o conjunto de não terminais, que tal e qual como em GP, são operadores intermédios que conectam os terminais (T), sendo mapeados pelo conjunto de regras de produção P. Existe um valor não terminal especial que representa o símbolo por onde toda a derivação começa, está representado em S. O seguinte exemplo ilustra um exemplo de uma gramática na notação BNF.

$$S = \{< string >\}$$

$$N = \{< string >, < letter >, < vowel >, < consonant >, < character >\}$$

$$T = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, ", ?, ', ', ., ;\ , :, ' '\}$$

Neste exemplo o símbolo de início de derivação é o não terminal <*string*>. O conjunto de não terminais é dado por N. O conjunto de terminais T é composto por letras, espaços e pontuação, enquanto o conjunto de não terminais é composto por cinco símbolos sendo que o símbolo *string* se trata do não terminal especial que marca o início. Estes conjuntos são então ligados pelo conjunto P, conjunto esse que mapeia a forma legal para formar o fenótipo ou as soluções. O conjunto P é composto pelas seguintes regras:

(1) $< string > ::= < letter > \mid < letter >< string >$

(2) $< letter > ::= < vowel > \mid < consonant > \mid < character >$

(3) $< vowel > ::= a \mid e \mid i \mid o \mid u$

(4) $< consonant > ::= b \mid c \mid d \mid f \mid g \mid h \mid j \mid k \mid l \mid m \mid n \mid p \mid q \mid r \mid s \mid t \mid v \mid w \mid x \mid y \mid z$

(5) $< character > ::= " \mid ? \mid , \mid . \mid ; \mid : \mid ' '$

Gramática 1 – Regras de produção de para construção de frases [6]

Em GE a representação de um indivíduo é um vetor de inteiros também conhecido como genoma ou genótipo e o programa originado como fenótipo. A cada inteiro desse vetor tem o nome de códão. O genótipo e fenótipo tratam-se de termos provindos da biologia que representa a estrutura genética e o aspeto físico de um indivíduo respetivamente.

O genótipo por si só, não constitui uma solução explícita do problema, para extrair a solução é necessária a utilização das gramáticas que irão mapear os vetores para efetivamente obter uma solução.

O processo de mapeamento é feito iterativamente derivando cada codão do genoma da esquerda para a direita. É então escolhido S para iniciar o processo de derivação, sendo que sempre que uma escolha tem de ser feita, é calculado o resto da divisão inteira entre códão sobre o número de regras existentes resultando no índice da regra a escolher.

Por exemplo, o primeiro codão de um indivíduo for 38 e utilizando a gramática referida anteriormente, verifica-se que o mapeamento deve começar por <*string*> e este possui duas possibilidades de derivação por isso a operação efetuada é: 38 mod 2 = 0 (sendo que mod é o operador do resto da divisão inteira). Ou seja, é selecionada a opção <*letter*> pois é a regra de índice 0 que mapeia <*string*>, o processo continua com os restantes codões.

A derivação termina quando não existirem mais não terminais por mapear. No caso de o indivíduo não estar totalmente mapeado e já não existirem mais codões ou se abandona o indivíduo atribuindo-lhe o *fitness* mínimo ou é realizado *wrapping* [6], onde é utilizado novamente o primeiro códão e repetindo este processo. Quando é utilizado o *wrapping* é necessário por vezes definir o número de vezes que um indivíduo pode ser *wrapped*. Por outro lado, se sobrarem codões são chamados de “cauda” do indivíduo e são apenas codões que não contribuem para o mapeamento apesar de fazerem parte do indivíduo.

Para exemplificar o processo de mapeamento será utilizada a Gramática 1 e também o seguinte indivíduo [1,1,13,1,0,4,0,0,2] que por sua vez é mapeado para a string *rui*. Na Figura 23 podemos acompanhar a evolução do fenótipo na variável fen que vai sendo alterada à medida que é feita a operação de modulo de cada um dos codões com o número de opções da regra mais à esquerda nesse momento do genótipo.

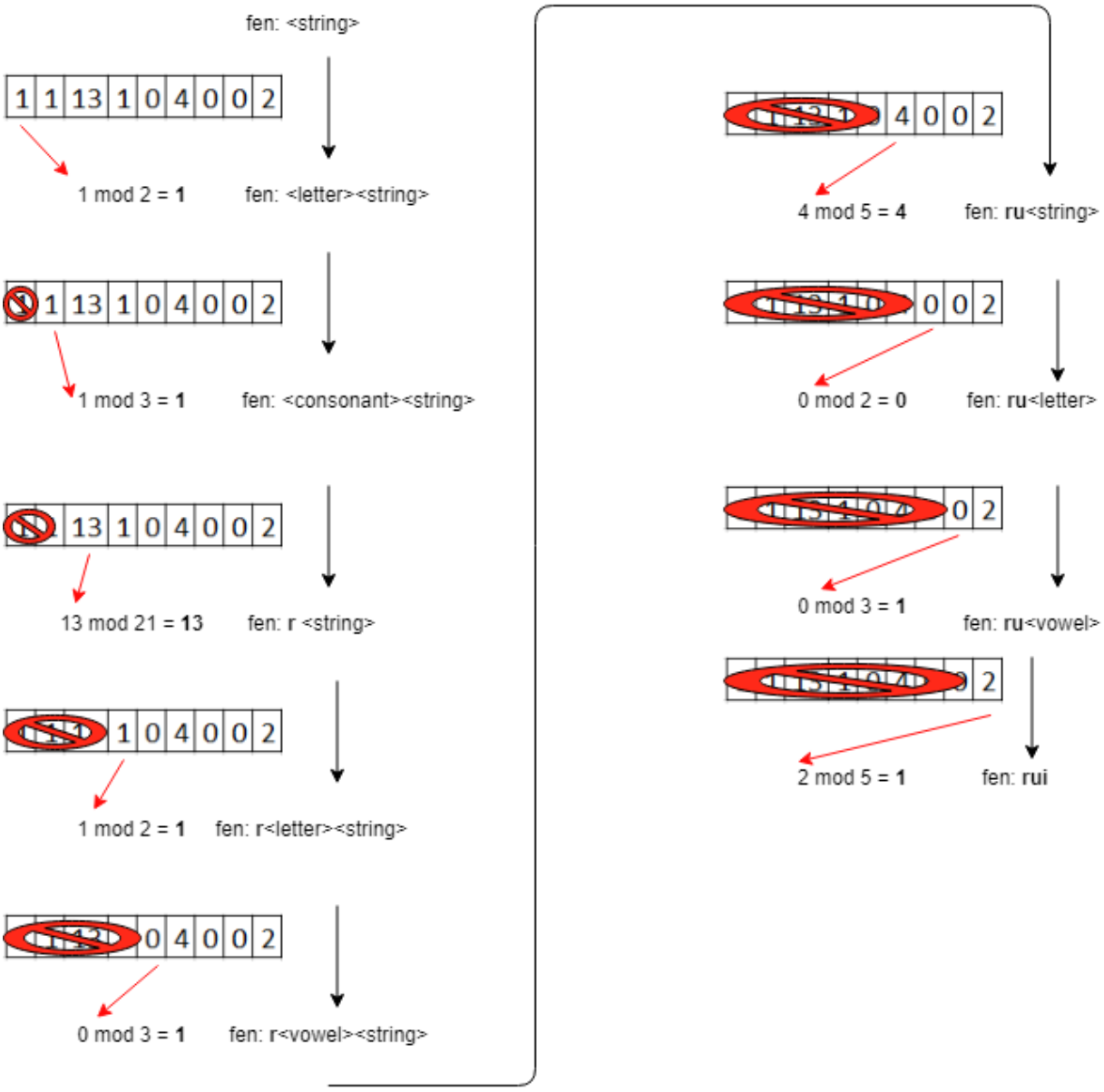

Figura 23 - Exemplo do processo de mapeamento

A definição da gramática é extremamente importante devido ao impacto que tem no espaço de procura [6]. A gramática vai garantir a produção de programas válidos, mas por esta ser explícita deve ser definida com algum rigor, para garantir uma melhor performance dos indivíduos. Na literatura [53] é possível ver que a definição da gramática pode ter impacto nos resultados e acabar por reduzir a performance de um modelo.

Como referido anteriormente a inicialização da população de qualquer método de evolucionário pode ter grandes impactos na performance do algoritmo. A inicialização aleatória foi a primeira abordagem para inicializar a população, mas como é possível identificar aqui [54], este tipo de abordagens pode ter grande impacto na população inicial.

A inicialização da população feita de forma uniforme, sabendo que a possibilidade da escolha de regras é igual, utilizando a gramática apresentada anteriormente, no caso

de $< S > ::= < letter >$ conclui-se que metade da população gerada seria apenas uma *<letter>, <consonant> ou <character>.* Limitando a variedade indivíduos da população inicial tornando mais difícil a evolução. São propostas na literatura [55] algumas opções para inicialização que tentam combater o problema de inicialização tendenciosa.

Os operadores genéticos como *crossover* mais comum [56], entre outros, chama-se de *single point crossover* onde basicamente são selecionados dois indivíduo e é escolhido aleatoriamente um ponto de *crossover* em ambos e a metade do lado direito dos indivíduos é trocada de um para outro e vice-versa.

Ao nível de mutação em [57] são comparadas três estratégias a mutação estrutural e por inteiros para verificar o impacto destes operadores genéticos no *fitness* antes da execução do GE. Onde são analisadas estratégias de mutação ao nível no fenótipo, diretamente a árvore de derivação em comparação a uma mutação mais convencional em GE [6] ao nível do genótipo.

GE é considerado um algoritmo modular, o que significa que qualquer parte do processo pode ser alterado desde a linguagem do problema, alterando a gramática, ou até o próprio algoritmo de pesquisa, não sendo sequer necessário utilizar um algoritmo evolucionário.

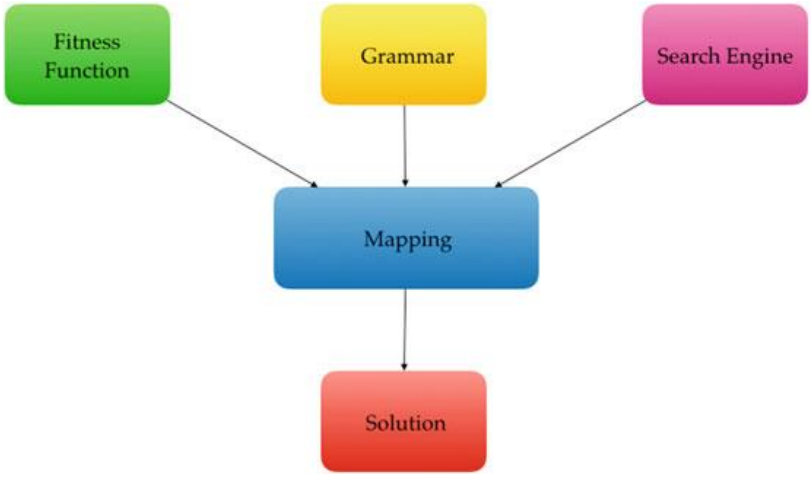

Figura 24 - Natureza modular de GE [6]

A Figura 24 mostra com grande clareza que existem blocos que podem ser alterados independentemente de qualquer outro. Se for necessário alterar a linguagem do programa que se está a trabalhar basta alterar a gramática utilizada. Se for necessário definir pesos dinâmicos para qualquer indivíduo a função de avaliação (a verde) pode ser alterada a função de avaliação. No caso de o algoritmo de pesquisa também poderá ser alterado não estando restrito a qualquer tipo de algoritmo evolucionário, sendo que qualquer algoritmo de otimização é válido neste contexto.

GE possui uma baixa localidade de representação que pode ser visto como um problema [58][59]. O conceito de localidade de representação diz respeito ao quão parecidos são os vizinhos de determinado indivíduo na sua representação genotípica

perante sua representação fenotípica. Ou seja, diz-se que a localidade de representação é alta quando os vizinhos de determinado indivíduo são os mesmos tanto ao nível do genótipo como do fenótipo. Ou quando pequenas alterações efetuadas ao nível do genótipo produzem pequenas alterações ao nível do fenótipo.

Concluindo GE é uma técnica extremamente versátil devido à sua origem modular e também ao funcionamento. Outra vantagem de GE é a sua expressividade, sendo que utiliza uma gramáticas acaba por ter melhor expressividade ao nível do fenótipo. Mas possui a desvantagem de ter uma baixa localidade de representação, onde mínimas alterações no genótipo originam uma abruta alteração do fenótipo, e também uma grande redundância no mapeamento onde vários genótipos podem dar origem aos mesmos fenótipos. Estas características são apontadas na literatura como grandes limitações de GE [60][59].

4.6 - *Structured Grammatical Evolution*

O Structured Grammatical Evolution (SGE) é uma alteração proposta em [60] que altera a forma como cada indivíduo é representado ao nível do genótipo. Esta abordagem é uma versão de GE, mas com alterações nas suas configurações. Será explorado especificamente pois irá ser utilizado na proposta relativa a este documento.

Visto anteriormente, é possível perceber que em GE certas variações de genótipo poderão ter consequências menos boas para aprendizagem. Seja pela baixa localidade já vista anteriormente como por vários elementos da população poderem ter fenótipos exatamente iguais percebe-se que por exemplo um crossover terá pouco impacto e poderá reduzir a velocidade da aprendizagem [59]. O SGE surge no contexto de reduzir esta redundância e balancear a localidade e tentar acelerar a aprendizagem com o objetivo de melhorar os modelos gerados.

O SGE possui uma representação dos indivíduos mais estruturada, uma lista composta por genes. Cada gene está diretamente ligado a um não terminal específico. Por sua vez cada gene é composto por uma lista de inteiros usada para selecionar a regra de derivação. Cada um desses inteiros corresponde diretamente ao índice da regra a utilizar, não sendo necessário utilizar a operação do resto da divisão inteira.

$$< start > ::= < expr >< op >< expr > \mid < expr >$$

$$< expr > ::= < term >< op >< term > \mid (< term >< op >< term >)$$

$$< term > ::= \ x \mid 0.5$$

$$< op > ::= \ + \mid - \mid * \mid /$$

Gramática 2 - Regras de produção de operações matemáticas

Para exemplo será utilizada a Gramática 2 - Regras de produção de operações matemáticas apresentada anteriormente onde existem quatro regras, sendo que

<start>, <expr> e <term> possuem duas possibilidades de derivação e *<op>* possui quatro possibilidades de derivação. Baseado na gramática um possível indivíduo em SGE pode ser representado da seguinte forma:

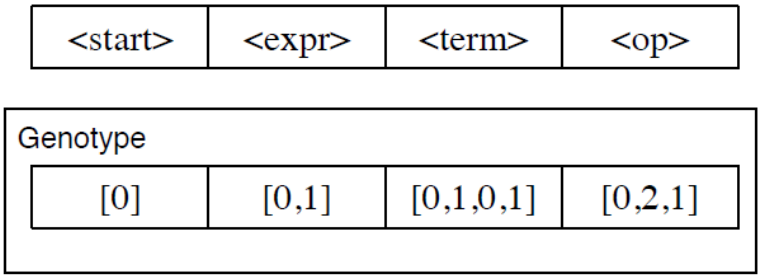

Figura 25 - Exemplo de como pode ser representado um indivíduo em SGE.

Como se pode observar na Figura 25 cada gene está diretamente associado a um não terminal que por sua vez estão associados a uma regra de produção.

O funcionamento SGE é semelhante ao do GE no que à evolução diz respeito, no entanto os operadores genéticos são específicos pois estão diretamente relacionados com a representação dos indivíduos. Estes são adaptados neste caso para funcionar com o SGE, operador de recombinação e de mutação.

O operador de recombinação é baseado num *crossover* uniforme, ou seja, a probabilidade de cada gene ser selecionado para ser trocada é igual. O processo inicia pela criação uma lista, do tamanho do conjunto de não terminais chamada de máscara, com valores binários em que cada representa o gene que será recombinado entre indivíduos. Na recombinação não existem alterações dentro das listas de cada gene, ou seja, o operador de recombinação apenas troca conjuntos de genes na totalidade como mostra na Figura 26

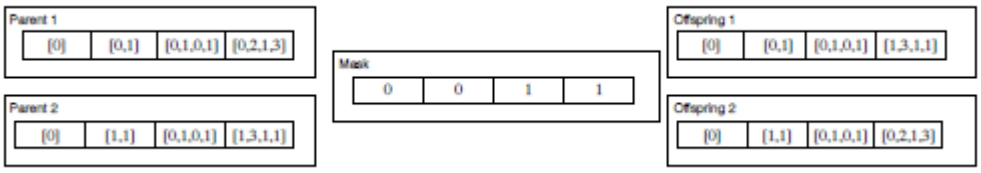

Figura 26 - Exemplo Recombinação SGE[60]**.**

Os dois indivíduos à esquerda são recombinados segundo a máscara, ou seja, são trocados os dois genes mais à direta de cada indivíduo. Dando então origem aos indivíduos descendentes do lado direito. De notar que o gene número três é igual em ambos os indivíduos, no entanto foram trocados pois na máscara o índice relativo a esse gene está a 1.

O operador de mutação altera diretamente a lista dentro do gene fazendo uma espécie de mutação apenas de um inteiro, neste caso atribuindo um novo valor aleatório de zero ao número de regras que o não terminal ligado esse gene possuir. No exemplo da Figura 27 ilustra uma possível mutação.

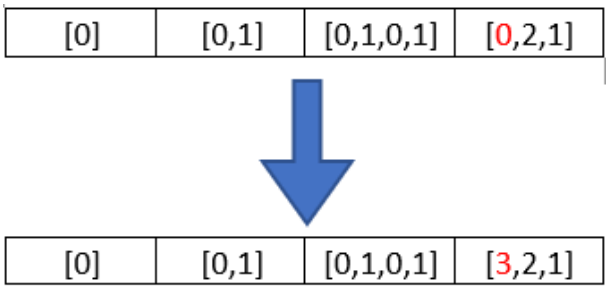

Figura 27 - Exemplo Mutação SGE.

É então possível observar uma mutação do primeiro elemento do quarto gene do indivíduo em cima em que passa de 0 para 3. O valor alterado tem obrigatoriamente de respeitar as regras de produção da gramática, ou seja, como no caso da figura os valores possíveis são, 0, 1, 2, 3 pois a regra de produção associada ao não terminal do quarto gene, possui quatro possibilidades de derivação.

A cada execução deste método é necessário definir determinados parâmetros que como visto anteriormente podem variar para as determinadas fases do processo. Os principais parâmetros a serem definidos são parâmetros normalmente encontrados em algoritmos evolucionários [6] do mesmo tipo como:

- População Inicial – É necessário definir o número de indivíduos que a população inicial deverá ter.
- Número de Iterações – Número de Iterações, ou Gerações, a realizar na execução.
- Elitismo – Elitismo trata-se do número dos melhores indivíduos, de acordo com a função de *fitness,* de uma determinada geração a serem transportados para a geração seguinte.
- Probabilidade de Crossover – Probabilidade o operador de crossover ser disparado.
- Probabilidade de Mutação - Probabilidade o operador de mutação ser disparado, normalmente este valor tem uma probabilidade muito baixa.

O SGE é uma técnica introduzida em [60] que contém resultados promissores, que em comparação a GE possui melhores resultados em todos os problemas selecionados para o teste. Provou ser eficiente pois com um menor número de avaliações obteve boas soluções.

5. ENSEMBLE SGE

Neste capítulo encontra-se a proposta de uma *framework* chamada *Ensemble SGE*, uma nova abordagem para a utilização de algoritmos de evolução automática de programas para a resolução de problemas de aprendizagem supervisionada.

O principal objetivo traçado no início da investigação foi a criação de uma nova técnica que tivesse algum sustento teórico com potencial de utilização futura para resolução de problemas.

Depois de analisado o funcionamento de *Ensembles* que pode ser visto no Capítulo 3, percebe-se o potencial de técnicas que agregam modelos, que para além da *performance* alta em termos de taxas de acerto e robustez de resultados, possuem um funcionamento peculiar em que cada *base learner* aprende um pequeno padrão dos dados e o grupo irá chegar a uma decisão final.

E também de uma análise sistemática a técnicas de evolução automática de programas baseadas em algoritmos evolucionários, registada no Capítulo 4, em que cada algoritmo irá produzir um conjunto de *learners* que por via de um processo evolucionário aprenderam diversas partes de um problema específico e segundo o seu *fitness* têm melhor ou pior aptidão para resolver esse mesmo problema.

É então possível encontrar pontos de contacto entre estas duas técnicas, em que uma delas produz diversos modelos para resolver um problema e outra que utiliza diversos modelos em conjunto para resolver um problema.

Surge a oportunidade de criação de uma *framework* que irá utilizar todas as potencialidades de um algoritmo de produção de programas para produzir um conjunto de *base learners* de *Ensemble* e posterior estratégia de agregação.

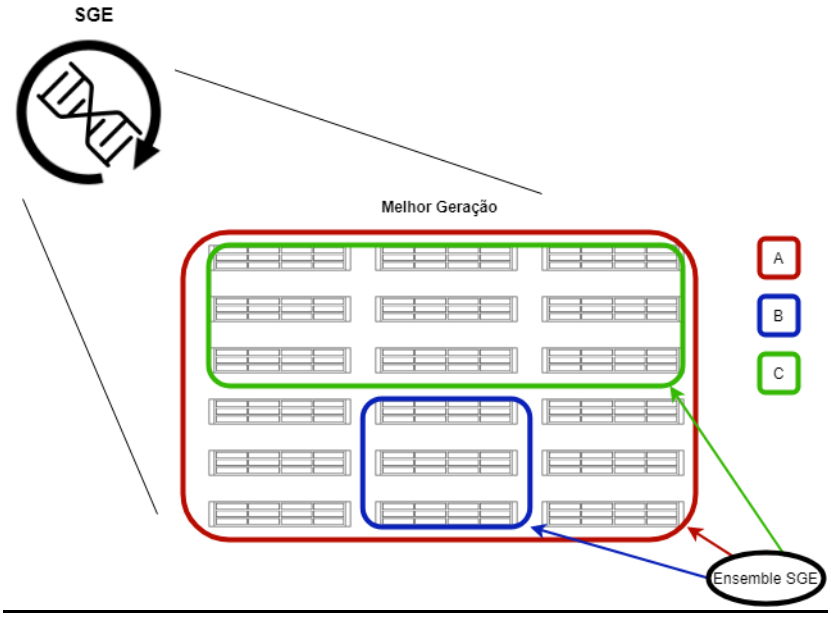

Figura 28 - *Ensemble* SGE, seleção de modelos provindos de uma geração do SGE e produção de 3 *Ensembles* A, B e C.

A Figura 28 tenta representar o funcionamento base onde o SGE, representado à esquerda, irá produzir uma geração de modelos e posteriormente o *Ensemble SGE* irá selecionar um subconjunto de modelos da população. Neste caso são exemplificados três *Ensembles* em instâncias diferentes da *framework* para o mesmo conjunto de modelos. O *Ensemble* A a vermelho utiliza a totalidade dos modelos criados. Noutra instância *Ensemble* B a azul, utiliza apenas 3 modelos da população. E por sua vez na terceira instância o *Ensemble* C a verde que utiliza metade da população.

Para a produção de modelos seria possível a utilização de qualquer algoritmo de evolução automática de programas como motor. Sendo que para cada um, em termos de implementação será necessária adaptação da *framework* à respetiva representação dos indivíduos. Daqui para frente é assumida a utilização do SGE para a produção dos modelos.

Sabe-se que o motor que irá produzir os modelos é baseado em algoritmos evolucionários, por isso, em cada execução irá criar diversas gerações com potenciais *base learners* para o *Ensemble*.

É necessário guardar alguns parâmetros relativos à execução do motor. O *dataset* utilizado será eventualmente separado em subconjuntos de treino e teste. O subconjunto de teste deve ser guardado para garantir que os testes realizados posteriormente aos Ensembles são feitos com os dados que nenhum modelos “viu” anteriormente.

Relativamente a resultados do motor de produção é necessário guardá-los na sua totalidade, ou seja, todas as gerações e todos os seus respetivos indivíduos com fenótipo e/ou genótipo (neste caso é necessário guardar também a gramática utilizada pelo motor) e também o *fitness* de cada um.

Para gerar os resultados dos *Ensembles* é necessário executar os modelos criado pelo SGE, daí a necessidade do fenótipo ou genótipo pois são as estruturas que permitem transformar o indivíduo num programa executável. Por sua vez o *fitness* é uma métrica importante para escolhas posteriores a este passo daí a importância de guardar estes valores para cada indivíduo.

5.1 - Seleção de *Base Learners*

O primeiro passo para construção dos modelos em *Ensemble* é a seleção dos *base learners.* Não só quais, mas também quantos *base learners* utilizar.

Numa primeira fase não é possível perceber qual a melhor geração para obter o melhor *Ensemble* por isso é aconselhável o teste de várias gerações, sabendo que, as gerações mais aptas serão obtidas em gerações mais próximas do final da execução que por sua vez trazem também melhores resultados como pode ser visto no Capítulo 6. Tipicamente a melhor geração será a última geração da execução do motor de produção.

A segunda escolha a fazer é a seleção de quantos modelos da geração a utilizar para isso são propostas duas estratégias diferentes para selecionar quantos e quais os *base learners* a utilizar para formar o *Ensemble*.

1. Selecionar os *n* modelos com melhor *fitness* da geração - *n* pode ter um valor no mínimo 2 e no máximo o tamanho da população da geração selecionada.
2. Selecionar modelos em que o *fitness* dos mesmos se encontra no intervalo interquartil da distribuição de todos os valores de *fitness* de cada indivíduo da geração. Abordagem *Interquartile Range* (IQR)

Por sua vez pode ser também utilizada uma abordagem híbrida em que são selecionados modelos provindos de várias gerações diferentes de uma execução do motor de produção. Desta maneira não é necessário selecionar a geração pois serão utilizadas todas as gerações da execução. Em que são selecionados os *n* melhores indivíduos de cada geração de uma execução do motor. Ou seja, se o motor executou durante 50 gerações, o *Ensemble* produzido pelo *Ensemble SGE* possui 50 *base learners.*

5.2 - Seleção da Função de Agregação

Após a seleção dos modelos é necessário agregá-los, para isso, é aplicada uma estratégia de *Stacking* onde cada modelo é independente e o *output* dos mesmos será enviado para um função de agregação para por sua vez calcular o resultado do *Ensemble.* Na Figura 29 vê-se este processo utilizando um dos exemplos da Figura 28:

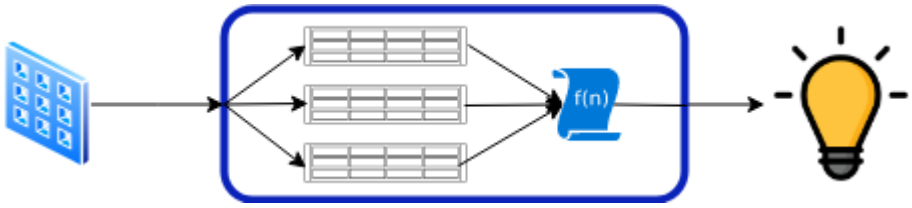

Figura 29 – Selecionando o *Ensemble* B, como input entram os dados no Ensemble e os mesmos são servidos a cada um dos base learners, os mesmos produzem os resultados que são posteriormente agregados por um função.

Por fim a escolha da função de agregação é totalmente dependente do problema a resolver. A principal distinção será para problemas de classificação e para problemas de regressão.

Para regressões é utilizada a média ou mediana dos resultados do base *learners* para fazer a previsão e dar o output do *Ensemble*.

Para classificações é utilizado o *majority voting* sendo que o resultado *Ensemble* será o output mais frequente no conjunto de outputs dos *base learners.*

5.3 - Implementação

Encontra-se disponível para a comunidade a implementação do Ensemble-SGE, num repositório do *GitHub* https://github.com/rumaroab/ensemble-sge.

Toda a implementação da ferramenta foi realizada com base no projeto original do SGE que se encontra também num repositório do GitHub https://github.com/nunolourenco/dsge.git , acrescentando funcionalidades capazes de selecionar *base learners* e também de os agregar e por sua vez executar experiências.

A ferramenta foi implementada em *Python* versão 2. As *diretrizes* de implementação foram baseadas na implementação do *SGE,* inclusive algum código reutilizado no que diz respeito a métricas e separação de treino e teste. Por sua vez o projeto possui um clone do repositório do *SGE.* Sendo que de momento a implementação está totalmente dependente da estrutura dos componentes internos produzidos pelo *SGE.*

5.4 - Problemas

Foram selecionados três problemas para testar a técnica proposta, com base nos testes efetuados em [60], para ter uma base de comparação dos resultados. Serão analisados o três problemas de Regressão Simbólica, *Quartic Polynomial*, *Pagie Polynomial* e *Boston Housing*.

Uma Regressão Linear Simples possui pelo menos dois parâmetros que podem ser ajustados, no entanto a sua representação matemática é sempre a mesma. A regressão simbólica [61] tenta com base em métodos de evolução automática de programas, encontrar uma função que se aproxime ao conjunto de dados sem que

previamente seja necessário definir forma ou parâmetros. Para os testes foram escolhidos dois polinómios para serem aproximados pelo *ensemble* SGE. Sendo que um deles se trata de uma equação de quarto grau:

$$quartic\ polynomial\ = f(x) = ax^4 + bx^3 + cx^2 + dx + e$$

Equação 10 - Polinómio de quarto grau

E também o polinómio de Pagie:

$$pagie\ polynomial = f(x,y) = \frac{1}{1+x^{-4}} + \frac{1}{1+y^{-4}}$$

Equação 11 - Polinómio de Pagie

Foi escolhida uma equação simples de quarto grau devido à facilidade com o SGE resolve esse exemplo em particular e por sua vez o Pagie exatamente pelas razões opostas onde o SGE teve mais dificuldade em resolvê-lo.

O *Boston Housing* mencionado na literatura [62], é um problema baseado num *dataset* com 506 registos construído em com *features* relativas a casas na zona de Boston provindo de censos realizados em 1970. O objetivo principal é prever o preço das casas com base nas suas características.

A função de avaliação estará diretamente relacionada com a função que resolve problema em questão, visto que cada indivíduo será um modelo válido para a sua resolução. Por isso, pode ser utilizado o erro do modelo para definir a aptidão de cada indivíduo. Este é calculado pelo *Root Relative Squared Error (RRSE)* que calcula o erro relativo comparativamente a um preditor aleatório simples. Ou seja, são calculados os erros, elevado ao quadrado, que um indivíduo produz e divide-se pelo mesmo erro elevado ao quadrado de um preditor aleatório. É então calculada a raiz quadrada da divisão anterior para reduzir o erro às mesmas dimensões dos valores previstos. É utilizado este método para manter a coerência com os valores por defeito do SGE.

$$RRSE = \sqrt{\frac{\sum_{j=1}^{N}(P_j - T_j)^2}{\sum_{j=1}^{N}(T_j - \bar{T})^2}}$$

Equação 12 - Root Relative Square Error

Na Equação 12 - Root Relative Square Error *N* representa o número total de exemplos, *P* representa o valor previsto pelo modelo para o exemplo *j*, *T* representa o valor do *target*. O símbolo $\bar{T}$ representa o a média dos *targets* de todos os exemplos utilizados.

No caso do *Boston Housing* é utilizada a abordagem treino e teste onde são selecionados os conjuntos antes da execução do SGE. No caso de ambos os polinómios são selecionados intervalos diferentes da distribuição para cada uma das

equações para treino e para teste. Os exemplos de treino são utilizados em toda a execução do SGE ao longo de todas as gerações seguindo o processo normal do algoritmo.

5.5 - Configurações de Teste

De seguida serão descritas algumas configurações para as experiências a realizar utilizando o *Ensemble SGE*. As configurações utilizadas para a execução do SGE são as mesmas para todos os problemas, e estão representadas na seguinte tabela:

Tamanho da População	1000
Número de Iterações	50
Elitismo	100
Probabilidade de Mutação	0.01
Probabilidade de Crossover	0.9

Tabela 5 – Parâmetros do SGE utilizados para executar os testes

Para cada problema foram feitas cinco execuções do SGE. Foi selecionada a execução com o melhor indivíduo por geração em média, ao longo das gerações. Isto é realizado devido à grande variância dos resultados do SGE devido à natureza estocástica de alguns componentes do algoritmo.

Cada execução regista um log da população por geração, ou seja, um conjunto de indivíduos/modelos prontos a utilizar por cada geração da execução. De notar que cada experiência produz 50 gerações que por sua vezes possuem 1000 indivíduos cada uma o que dá aproximadamente um total 50000 potencias *base learners* de um *Ensemble* para a execução escolhida.

Para cada geração dessa execução serão criados Ensembles seguindo diferentes estratégias:

1. Utilização da totalidade da população
2. Utilização da melhor metade da população
3. Utilização da abordagem IQR
4. Utilização dos 20 melhores indivíduos
5. Utilização dos 10 melhores indivíduos

Serão também utilizadas duas diferentes abordagens de agregação para os Ensembles. Será usada numa instância a média para agregar os modelos selecionados para produzir o resultado e noutra instância a mediana.

Ou seja, para cada um dos três problemas referidos serão criados dez Ensembles por geração de uma execução. O resultado de cada Ensemble será comparado ao melhor indivíduo da respetiva geração. Desta maneira será possível observar a evolução do melhor indivíduo do *SGE* como a evolução dos Ensembles gerados.

De notar que os resultados apresentados estão maioritariamente apresentados em gráficos com a respetiva descrição e também uma análise crítica. Apresentado nos gráficos está o valor de *RRSE* de teste. Isto é, o *SGE* foi executado utilizando o *dataset* de treino para avaliação dos indivíduos e produção das 50 gerações. No entanto para a comparação com os Ensembles os melhores indivíduos de cada geração foram testados com o conjunto de teste, também para efetivamente perceber a capacidade de generalização de cada um dos métodos.

6. RESULTADOS EXPERIMENTAIS

Neste capítulo são analisados os resultados experimentais dos testes mencionados anteriormente. A única métrica utilizada para fazer a comparação é o erro produzido pelos modelos. Para uma mais rápida análise às seguintes figuras, a verde é representado o *RRSE* do melhor indivíduo de cada geração, a amarelo o *RRSE* do *Ensemble* que possui como função de agregação a mediana e a vermelho o *RSSE* do *Ensemble* que possui como função de agregação a média.

6.1 - Quartic Polynomial

Para o problema da equação de quarto grau foi efetuado o teste selecionando toda a população de cada geração que dá um total de 1000 *base learners*. Foram criados *ensembles* com a média e a mediana como funções de agregação. O melhor indivíduo produzido pelo SGE simples teve um erro de 0.07 enquanto o *ensemble* obteve 0.18 quando utilizada a mediana e obteve $6.89 * 10^{62}$ de erro quando utilizada a média. Isto deve-se ao facto de alguns elementos da população final não serem de todo aptos para resolver o problema.

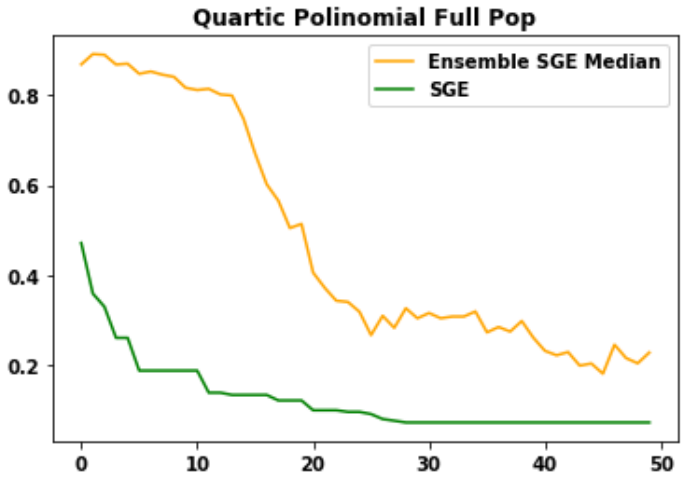

Figura 30 - Erro por geração da equação de quarto grau comparando SGE e *Ensemble* SGE utilizando a totalidade da população

O *Ensemble* SGE mostra uma melhoria ao longo das gerações não conseguindo obter melhores resultados comparativamente ao melhor indivíduo do SGE. A linha que representa o erro da versão do *Ensemble* SGE que utiliza a média não está apresentada devido aos maus resultados, os valores de erro são demasiado altos e ficaria fora de escala.

Ao usar a totalidade da população e a média é esperado que os resultados sejam estes pois os indivíduos maus de uma geração são tipicamente mesmo muito maus e por sua vez utilizando a média esses indivíduos irão ter uma contribuição muito

negativa para o resultado Ensemble daí que para estes casos a mediana seja mais robusta.

Numa segunda fase foi apenas utilizada a melhor metade da população, ou seja 500 *base learners*, em que são esperados melhores resultados que a experiência anterior pois a seleção dos indivíduos restringe a indivíduos com melhor aptidão para resolver o problema.

O melhor resultado foi obtido pelo *ensemble* utilizando a média e o erro é de 0.12, como esperado um resultado mais aproximado do melhor indivíduo. No entanto utilizando a mediana ainda se obtém um erro menor de 0.10.

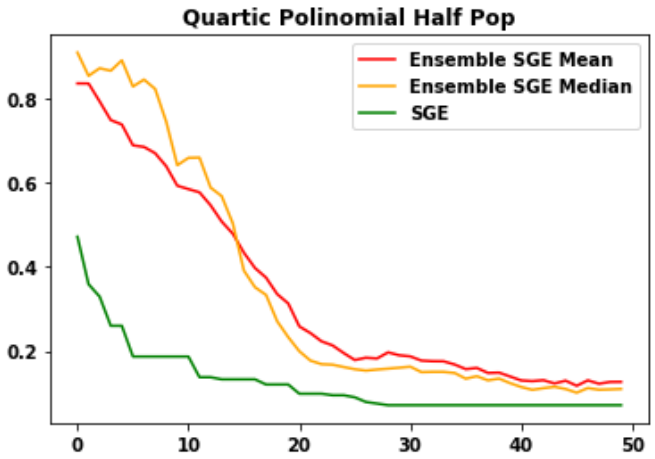

Figura 31- Erro por geração da equação de quarto grau comparando SGE e *Ensemble* SGE utilizando a melhor metade da população

Na Figura 31 observa-se uma melhoria substancial relativamente à utilização da média e por sua vez uma melhoria também utilizando a mediana, sendo que esta última consegue ser melhor que a utilização da média.

De notar que na Figura 30 e Figura 31 os melhores resultados para o *ensemble* não são na última geração como era esperado. É de esperar que a última geração seja melhor que todas as anteriores, no entanto neste caso a última geração não obtém os melhores resultados, estes são obtidos por volta da geração 45.

Uma explicação para isto ocorrer é que a cobertura do espaço de procura que o conjunto de *base learners* que forma o *Ensemble* na geração 45 é maior que na última geração. Apesar de existir elitismo no SGE a evolução ocorre na mesma indivíduos mais fracos e esses conseguirem evoluir ao ponto de cobrir pontos do espaço de procura que os melhores não conseguiam. Uma demonstração do poder de generalização dos *Ensembles*.

A terceira foi a abordagem IQR, maneira a perceber se um conjunto que não contivesse nem os melhores nem os piores indivíduos pode obter melhores resultados. Não são esperados melhores resultados desta abordagem, no entanto o objetivo desta abordagem é perceber se a capacidade de generalização do Ensemble com indivíduos que não são os melhores.

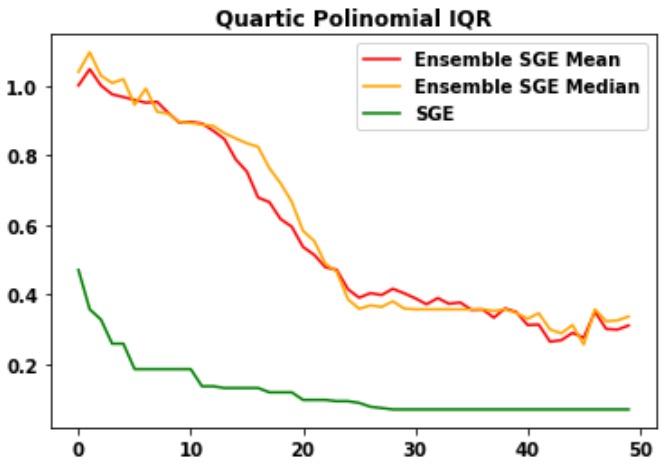

Figura 32 - Erro por geração da equação de quarto grau comparando SGE e *Ensemble* SGE utilizando a medida IQR

Como esperado não foi possível obter melhores resultados que utilizando a melhor metade da população, mesmo em comparação à experiência da, não foi possível obter melhores resultados com a abordagem IQR. Na abordagem vista na Figura 32, obtém-se um erro de 0.26 e 0.28 relativo à mediana e à média respetivamente.

As duas experiências seguintes têm como objetivo reduzir a complexidade do Ensemble utilizando um número reduzido de *base learners.* É esperado obter melhores resultados devido à concentração de *base learners* muito aptos.

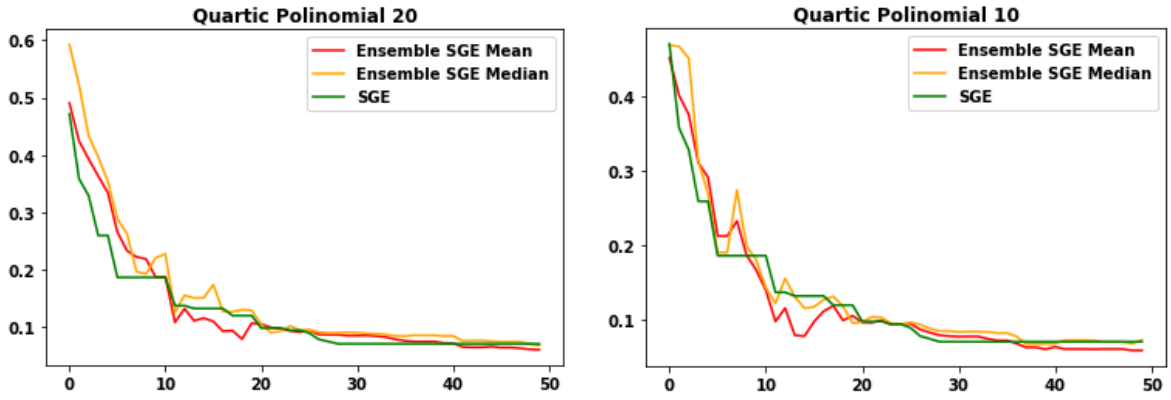

Figura 33 - Erro por geração da equação de quarto grau comparando SGE e Ensemble SGE utilizando os 20 melhores elementos (à esquerda) e os 10 melhores elementos (à direita) de cada geração

Na Figura 33 é possível verificar que os resultados utilizando *ensembles* bem mais pequenos que os anteriores são bastante promissores. Ambas as abordagens conseguem ultrapassar a performance de 0.07 do melhor modelo gerado pelo SGE simples. Com o melhor *Ensemble*, para os 20 melhores e para os 10 melhores a atingir um erro de 0.06 e 0.059 respetivamente, ambos utilizando a média como função de agregação.

Como esperado a concentração dos *base learners* originou resultados promissores ultrapassando a performance do melhor indivíduo gerado pelo SGE. A melhoria dos resultado é pequena e o progresso ao longo das gerações é muito similar ao melhor

indivíduo produzido pelo SGE o que mostra que os *base learners* utilizados por estas abordagens são similares, no entanto diferentes o suficiente para conseguir generalizar um pouco melhor que um *base learner* apenas.

6.2 - Pagie Polynomial

Tal e qual como mostrado anteriormente a média não será mostrada neste caso pois devido aos indivíduos com resultados inválidos não foi possível calcular a média utilizando toda a população da geração. Para o melhor modelo gerado pelo SGE o melhor resultado é atingido na geração 43 com um erro de 0.14.

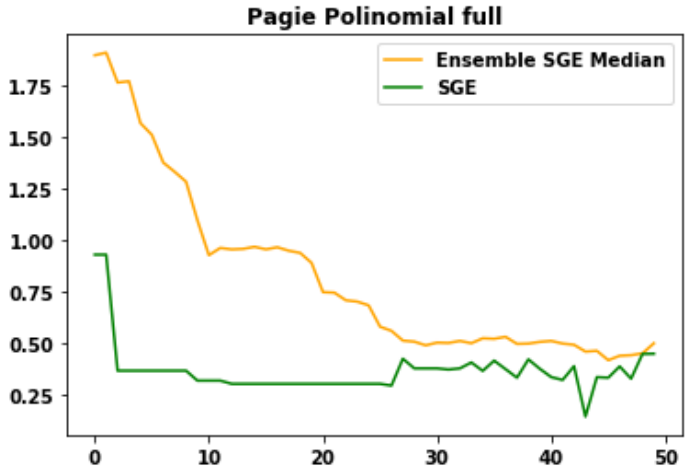

Figura 34- Erro por geração do pagie polynomial comparando SGE e Ensemble SGE utilizando a totalidade da população

De notar a Figura 34, que para este caso possuímos um problema de *overfitting*. O SGE possui elitismo, ou seja, faz com que o melhor indivíduo de uma geração passe para a próxima, no entanto a partir da geração 25 parece que o melhor modelo não é capaz de continuar a aprender e claramente fica descontrolado sem conseguir generalizar para os dados de teste que são utilizados nas experiências. As experiências procederam com o mesmo conjunto para perceber o comportamento dos *ensembles* gerados em casos como este.

O valor da mediana para este caso apresenta uma descida constante do erro, no entanto quando o SGE começa com o overfitting o ensemble deixa também de progredir com o melhor resultado de 0.45 na geração 49.

Novamente para a segunda e terceira abordagem são utilizadas a melhor metade da população e a abordagem IQR de maneira com os mesmos objetivos do problema anterior.

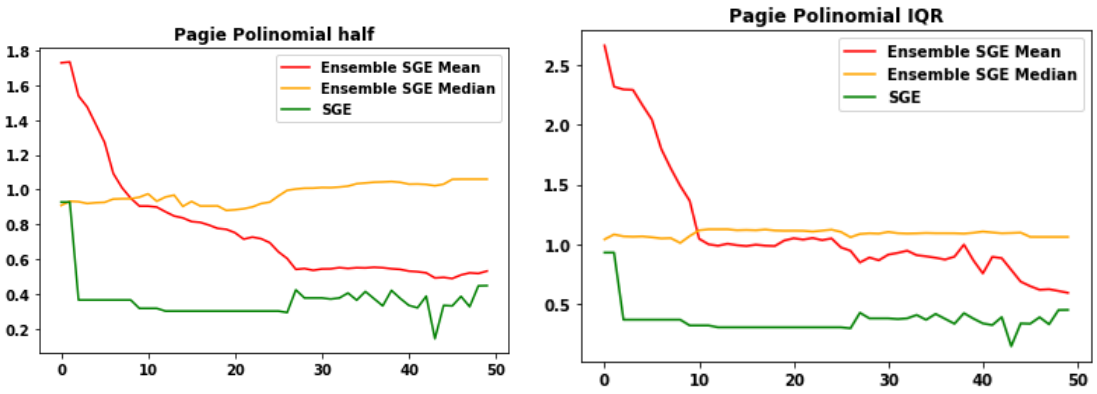

Figura 35 - Erro por geração do pagie polynomial comparando SGE e Ensemble SGE utilizando metade da população (à esquerda) e a abordagem IQR (à direita)

Na Figura 35 é possível verificar que em ambas as abordagens utilizando a mediana os resultados são quase constantes e para a melhor metade da população piora ao longo das gerações coincidente ao *overfitting*.

É possível por um lado perceber que ao longo das gerações efetivamente na maioria dos indivíduos não está a existir uma grande aprendizagem e por sua vez os resultados dos *Ensembles* também não são capazes de corrigir o overfitting.

Já utilizando a média como função de agregação os resultados são diferentes mostrando um bom progresso ao longo da execução conseguindo estabilizar a evolução sempre com alguma melhoria mesmo após o descontrolo iniciado na geração 25.

Para a quarta e quinta abordagem novamente reduzindo a complexidade dos *Ensembles* criados, verifica-se uma melhoria notória na execução comparando às outras abordagens.

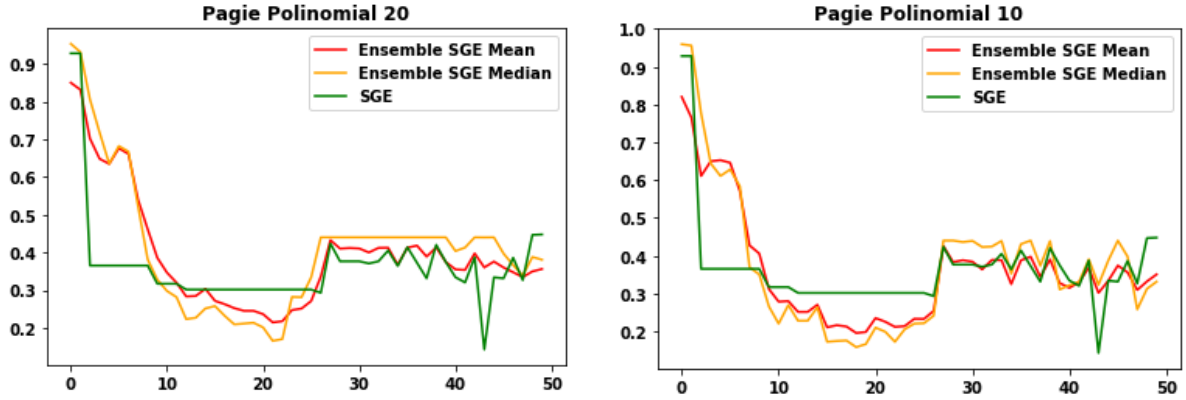

Figura 36 - Erro por geração da equação de quarto grau comparando SGE e Ensemble SGE utilizando os 20 melhores elementos (à esquerda) e os 10 melhores elementos (à direita) de cada geração

Nos dois casos da Figura 36 o melhor resultado obtido é do SGE simples, mas até ao descontrolo que começa a ocorrer por volta da geração 25, os Ensembles estavam com uma melhor performance ao longo das gerações.

A robustez que se possuía anteriormente onde o ensemble conseguia estabilizar a evolução ao longo das gerações não ser observado em nenhum destes casos pois a partir da geração 25 percebe-se então um *overfitting* igual ao do SGE. Apesar de o descontrolo ser mais suave e não existirem tantos picos no erro, especialmente utilizando 20 *base learners*. Uma potencial causa para isto é similaridade entre *base learners.* Por serem menos e devido ao elitismo referido anteriormente pode admitir-se que os constituintes do *Ensemble* serão muito similares entre si.

Outro facto curioso sobre esta experiência é que ambos os Ensembles conseguem identificar o início do *overfitting* mais cedo que o melhor modelo produzido pelo SGE. Isto porque como são utilizados vários modelos existe a possibilidade de o descontrolo ter começado antes nos constituintes e só depois ter chegado ao melhor.

6.3 - Boston Housing

Como em ambos os exemplos anteriores para o problema do Boston Housing foram realizados os mesmos 4 testes. O SGE simples possui também alguma dificuldade em resolver este problema e é expectável, após os resultados anteriores que os exemplos com 10 e 20 *base learners* tenham uma performance tão boa ou melhor que o SGE simples.

Neste caso apenas será apresentada a próxima figura relativa à abordagem que utiliza metade da população para formar o ensemble. Para este problema tanto a população total como a abordagem IQR não produziram resultados válidos devido à enorme dimensão do erro. Devido à dificuldade de aprendizagem do Ensemble SGE para este problema foram produzidos ao longo das gerações muitos indivíduos pouco aptos, que dão origem a resultados do ensemble inválidos. Como se pode verificar nas falhas observadas na Figura 37.

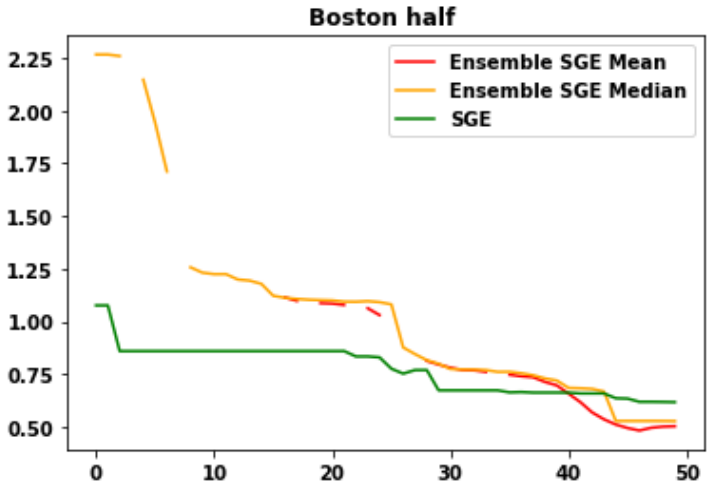

Figura 37 - Erro por geração do Boston Housing comparando SGE e Ensemble SGE utilizando metade da população

Como se pode ver na figura até utilizado a melhor metade da população existem muitos valores inválidos como pode ser visto nas quebras que existem nas gerações

iniciais da execução. Para este caso os melhores resultados obtidos são os resultados obtidos pelo Ensemble SGE. Sendo que o melhor de todos aparece utilizando a média com um erro total de 0.48, obtido pelo *ensemble* que utiliza a média.

Apesar dos resultados inválidos no início da execução são observados melhores resultados para as últimas gerações conseguindo obter melhores resultados que o SGE simples pela primeira vez utilizando metade da população.

Tendo em conta a performance de experiências anteriores é esperado que reduzindo a complexidade dos *Ensembles* tendencialmente os resultados têm uma melhor performance.

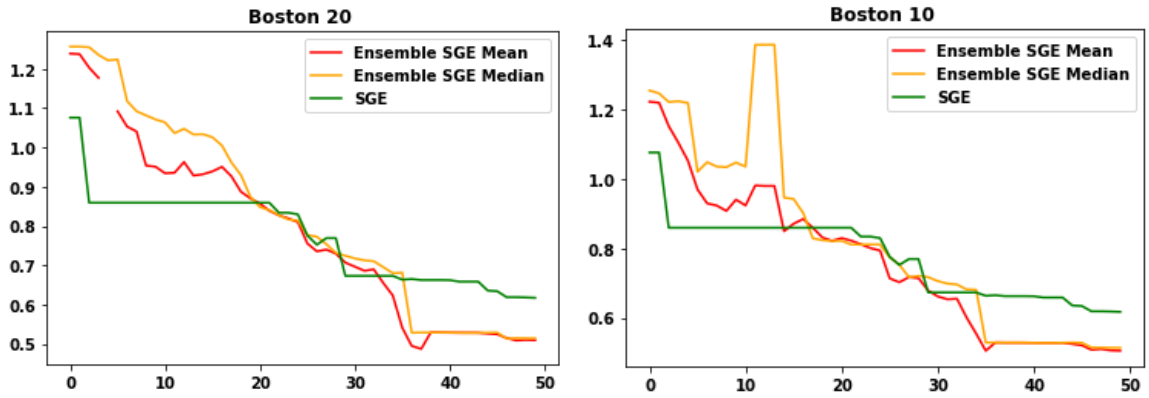

Figura 38 - Erro por geração do Boston Housing comparando SGE e Ensemble SGE utilizando os 20 melhores elementos (à esquerda) e os 10 melhores elementos (à direita) de cada geração

O método que melhor resolve o *Boston Housing* foi produzido pelos 20 melhores programas da execução utilizando a média como função de agregação. É obtido na geração 37 com um erro de 0.48.

De notar que o Ensemble que utiliza os 20 melhores elementos possui uma evolução mais suave produzindo um conjunto de resultados mais coeso e com menos picos. Já o de 10 *base learners* piora por volta da geração número 10 voltando novamente a ter melhores resultados algumas gerações a seguir.

A razão para o *Ensemble* com 20 *base learners* ter uma evolução mais regular que o *Ensemble* com 10 acontece devido ao facto de com 10 a similaridade entre *base learners* é maior, aproximando os resultados ao melhor exemplo do SGE, com divergências, relativas aos outros constituintes do *Ensemble*, que tornam um pouco erráticos os resultados com picos instáveis ao longo das gerações. Com 20 *base learners* os resultados são mais regulares ao longo das gerações e a performance unitária mantém-se praticamente a mesma.

7. CONCLUSÕES

Este trabalho tem com principal objetivo a criação de uma *framework* que funciona com base num algoritmo já existente o SGE[6]. *Framework* que utiliza o SGE para gerar modelos e posteriormente aplicar técnicas de *Ensembling* [40] para gerar novos modelos potencialmente melhores.

Para atingir este objetivo foi necessário realizar um conjunto de passos inclusive uma análise sistemática à área da aprendizagem automática. Esta análise permitiu aprender primeiramente como criar modelos inteligentes e também como podem ser usados em casos reais percebendo o seu ciclo de vida deste tipo muito específico de projetos.

Por sua vez aprofundando conceitos mais centrais para o projeto estudando potencialidades da aprendizagem por *Ensemble*, prós, contras e principalmente as várias maneiras de os implementar. Também foi feita uma análise mais profunda a métodos de otimização especialmente método evolucionários que por sua vez evoluem programas automaticamente.

Após esta foi análise foi possível definir uma estratégia para começar a implementação da *framework*. Após várias iterações foi então concluída com sucesso a implementação e por sua vez colocada num repositório público. De seguida tendo validado a implementação do Ensemble SGE passou-se para a avaliação do mesmo e comparação com o SGE já existente.

Os resultados do Ensemble SGE em comparação ao SGE foram um sucesso, tendo em conta que alguns dos Ensembles gerados conseguiram ultrapassar a *performance* do SGE. Das experiências realizadas foi possível perceber algumas conclusões.

Claramente a decisão mais importante a tomar para garantir a melhor performance do Ensemble é escolha dos *base leaners.* Neste documento a estratégia utilizada para encontrar o melhor número de *base learners* foi à base da experimentação que para muitos casos de uso não é ideal.

A utilização da totalidade da população não é uma boa ideia devido aos indivíduos maus ou até inválidos, existentes especialmente em gerações iniciais, que fazem que seja muito difícil ou até impossível, respetivamente gerar *Ensembles* com boa *performance*.

Ao reduzir o número de *base learners* presentes no *Ensemble*, a performance do *Ensemble* tem tendência a subir. Para casos em que se utiliza a totalidade da população ou a melhor metade não é uma grande diferença, no entanto para todos os problemas foi possível obter melhores resultados utilizando 10 e/ou 20 *base learners* por *Ensemble.*

Analisando especialmente as abordagens que utilizam 10 e 20 modelos constituintes é possível ver uma perda de robustez, ou seja, um comportamento mais errático com

maiores picos e performance instável ao longo das gerações, por parte dos Ensembles com 10 *base learners,* no entanto uma maior aproximação ao melhor indivíduo gerado pelo SGE, isto pelos constituintes desse *Ensemble* serem muito similares ao melhor modelo. O que faz concluir que o número de constituintes do modelo deve ser bem explorado para cada problema a resolver e também para cada geração pois para alguns casos este número pode ser mais baixo em outros mais alto.

Para casos de *overfitting* o Ensemble SGE consegue suavizar a evolução no entanto quanto menor o número de *base learners* mais visível é impacto deste *overfitting*. Neste caso também foi possível identificar o *overfitting* mais cedo, ao longo das gerações, devido à utilização de várias indivíduos que começam a adaptar-se demasiado aos dados de treino mais cedo na execução.

Em suma os objetivos propostos no início do trabalho foram concluídos com sucesso e com resultados muito positivos, mostrando o potencial do Ensemble SGE.

7.1 - Limitações

Como todos os algoritmos existentes o Ensemble SGE possui um conjunto de limitações que podem abrandar o desenvolvimento ou até impedir a sua utilização.

Uma das limitações dos algoritmos de evolucionários, ao qual pertencem os algoritmos de evolução automática de programas, é o elevado tempo de processamento. Por utilizar o resultado de um destes algoritmos o Ensemble SGE sofre com o tempo de processamento inicial necessário do SGE e também com o próprio processamento pois quando são trabalhados *Ensembles* muito grandes o tempo de processamento tende a aumentar.

Outra limitação é o número de *base learners* ideal ser totalmente dependente do problema, que por sua vez tem que ser encontrado com base em experiências. Podendo ser visto como uma otimização de Hiper parâmetros.

7.2 - Trabalho Futuro

Como primeira versão o Ensemble SGE consegue obter bom resultados baseado nos três problemas de regressão em que foi testado em comparação aos indivíduos do SGE. No entanto existem outras abordagens que potencialmente podem atingir bons resultados.

Neste momento o Ensemble SGE não foi testado com problemas de classificação o que torna uma incógnita a sua viabilidade para este tipo de problemas.

Podem ser utilizadas também no futuro abordagens diferentes para agregação dos modelos, por exemplo uma média ponderada utilizando o fitness normalizado de cada indivíduo para dar mais importância a exemplos mais aptos, uma abordagem utilizando conceitos de *boosting*.

Tendo em conta que o melhor indivíduo de cada geração possui tipicamente uma evolução constante, pode ser interessante a utilização do melhor indivíduo de cada geração para formar o Ensemble. Pode ser necessário remover indivíduos iguais do Ensemble para prevenir os casos em que se está a utilizar exatamente o mesmo indivíduo pois pode enviesar os resultados.

As experiências realizadas podem ser expandidas inclusive utilizando execuções mais extensas do SGE gerando populações mais aptas que potencialmente Ensembles mais fortes. Nem sempre acontece, no entanto, quanto mais tempo correr o SGE teoricamente os indivíduos gerados tornar-se-ão mais aptos o que pode vir a criar *Ensembles,* também eles mais fortes.

Podem também ser alterados os parâmetros do SGE ao gerar as populações de maneira diferente que poderá fazer com que a agregação de modelos seja mais rápida, ou seja alteração do tamanho da população ou das taxas de ativação de operadores genéticos, por exemplo.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] M. R. Oleg Okun, Giorgio Valentini, *Ensembles in Machine Learning Applications*. Springer, 2011.

[2] M. A. Ganaie, M. Hu, M. Tanveer, and P. N. Suganthan, “Ensemble deep learning: A review,” 2021.

[3] S. Hamori, M. Kawai, T. Kume, Y. Murakami, and C. Watanabe, “Ensemble Learning or Deep Learning? Application to Default Risk Analysis,” *J. Risk Financ. Manag.*, 2018, doi: 10.3390/jrfm11010012.

[4] T. Bartz-Beielstein, J. Branke, J. Mehnen, and O. Mersmann, “Evolutionary Algorithms,” *WIREs Data Min. Knowl. Discov.*, 2014, doi: https://doi.org/10.1002/widm.1124.

[5] A. Pr, “Benefits of a Population : Five Mechanisms That Advantage Population-Based Algorithms,” *IEEE Trans. Evol. Comput.*, 2010, doi: 10.1109/TEVC.2009.2039139.

[6] C. Ryan, M. O’Neill, and J. J. Collins, *Handbook of grammatical evolution*. Springer International Publishing, 2018.

[7] U. Bhowan, M. Johnston, M. Zhang, and X. Yao, “Evolving diverse ensembles using genetic programming for classification with unbalanced data,” *IEEE Trans. Evol. Comput.*, 2013, doi: 10.1109/tevc.2012.2199119.

[8] K. Nag and N. R. Pal, “A Multiobjective Genetic Programming-Based Ensemble for Simultaneous Feature Selection and Classification,” *IEEE Trans. Cybern.*, 2016, doi: 10.1109/tcyb.2015.2404806.

[9] Y. Jin and B. Sendhoff, “Pareto-Based Multiobjective Machine Learning: An Overview and Case Studies,” *Syst. Man, Cybern. Part C Appl. Rev. IEEE Trans.*, 2008, doi: 10.1109/tsmcc.2008.919172.

[10] D. Silver *et al.*, “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play,” *Science*, 2018, doi: 10.1126/science.aar6404.

[11] A. W. Senior *et al.*, “Improved protein structure prediction using potentials from deep learning,” *Nature*, 2020, doi: 10.1038/s41586-019-1923-7.

[12] D. Silver *et al.*, “Mastering the game of Go without human knowledge,” *Nature*, 2017, doi: 10.1038/nature24270.

[13] O. Theobald, *Machine Learning for Absolute Beginners*. Scatter Plot Press, 2017.

[14] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” *AT&T Labs*, 2010. http://yann.lecun.com/exdb/mnist/ (accessed Jan. 13, 2021).

[15] D. Harrison and D. L. Rubinfeld, “Hedonic housing prices and the demand for clean air,” *Reveal. Prefer. Approaches to Environ. Valuat.*, 2019, doi: 10.1016/0095-0696(78)90006-2.

[16] M. Akinkunmi, “Introduction to Statistics Using R,” *Synth. Lect. Math. Stat.*, 2019, doi: 10.2200/s00899ed1v01y201902mas024.

[17] G. James, D. Witten, R. Tibshirani, and T. Hastie, *An Introduction to Statistical Learning with Applications in R*, Seventh. Springer, 2013.

[18] G. Mariscal, Ó. Marbán, and C. Fernández, “A survey of data mining and

knowledge discovery process models and methodologies," *Knowl. Eng. Rev.*, 2010, doi: 10.1017/s0269888910000032.

[19] G. Piatetsky, "CRISP-DM , still the top methodology for analytics , data mining , or data science projects," *KDnuggets.com*, 2014. http://www.kdnuggets.com/2014/10/crisp-dm-top-methodology-analytics-data-mining-data-science-projects.html (accessed Dec. 20, 2020).

[20] C. Pete *et al.*, "Crisp-Dm 1.0," *CRISP-DM Consortium*, 2000. https://www.the-modeling-agency.com/crisp-dm.pdf (accessed Dec. 20, 2020).

[21] A. Samvelyan, R. Shaptala, and G. Kyselov, "Exploratory data analysis of Kyiv city petitions," *IEEE 2nd Int. Conf. Syst. Anal. Intell. Comput.*, 2020, doi: 10.1109/saic51296.2020.9239185.

[22] "Boxplot Example Image." https://miro.medium.com/max/18000/1*2c21SkzJMf3frPXPAR_gZA.png (accessed Mar. 23, 2021).

[23] G. Chang and T. Ge, "Comparison of missing data imputation methods for traffic flow," *Proc. 2011 Int. Conf. Transp. Mech. Electr. Eng. TMEE 2011*, 2011, doi: 10.1109/tmee.2011.6199284.

[24] A. A. e T. A. Efros, "Unbiased Look at Dataset Bias," 2011.

[25] Y. Li and N. Vasconcelos, "Repair: Removing representation bias by dataset resampling," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2019, doi: 10.1109/cvpr.2019.00980.

[26] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2013.

[27] A. D. Dongare, R. R. Kharde, and A. D. Kachare, "Introduction to Artificial Neural Network," *Int. J. Eng. Innov. Technol.*, 2012, doi: -.

[28] X. Zou, Y. Hu, Z. Tian, and K. Shen, "Logistic Regression Model Optimization and Case Analysis," *Proc. IEEE 7th Int. Conf. Comput. Sci. Netw. Technol. ICCSNT 2019*, 2019, doi: 10.1109/iccsnt47585.2019.8962457.

[29] M. S. Imtiaz and K. A. Wahid, "Color enhancement in endoscopic images using adaptive sigmoid function and space variant color reproduction," *Comput. Math. Methods Med.*, 2015, doi: 10.1155/2015/607407.

[30] L. Rokach and O. Maimon, *Data Mining With Decision Trees: Theory and Applications*, 2nd ed. USA: World Scientific Publishing Co., Inc., 2014.

[31] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[32] S. Fong, S. U. Rehman, K. Aziz, and I. Science, "DBSCAN : Past , Present and Future," *Fifth Int. Conf. Appl. Digit. Inf. Web Technol.*, 2014, doi: 10.1109/icadiwt.2014.6814687.

[33] Y. Liu, Y. Wang, and J. Zhang, "New machine learning algorithm: Random forest," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 2012, doi: 10.1007/978-3-642-34062-8_32.

[34] V. B. Vaghela, A. Ganatra, and A. Thakkar, "Boost a weak learner to a strong learner using ensemble system approach," *2009 IEEE Int. Adv. Comput. Conf. IACC 2009*, 2009, doi: 10.1109/iadcc.2009.4809227.

[35] D. Wolpert, “Stacked Generalization (Stacking),” *Neural Networks*, 1992, doi: 10.1016/S0893-6080(05)80023-1.

[36] D. Opitz and R. Maclin, “Popular Ensemble Methods: An Empirical Study,” *J. Artif. Intell. Res.*, 1999, doi: 10.1613/jair.614.

[37] L. Rokach, “Ensemble-based classifiers,” *Artif. Intell. Rev.*, 2010, doi: 10.1007/s10462-009-9124-7.

[38] D. W. Opitz and J. W. Shavlik, “Actively Searching for an Effective Neural Network Ensemble,” *Conn. Sci.*, 1996, doi: 10.1080/095400996116802.

[39] L. K. Hansen and P. Salamon, “Neural Network Ensembles,” *Pattern Anal. Mach. Intell.*, 1990, doi: 10.1109/34.58871.

[40] Z.-H. Zhou, *Ensemble Methods, Foundations and Algorithms*. CRC Press, A Chapman & Hall Book, 2012.

[41] R. E. Schapire, “The Boosting Approach to Machine Learning: An Overview,” in *Nonlinear Estimation and Classification*, Springer New York, 2003.

[42] Y. Freund and R. E. Schapire, “Experiments with a New Boosting Algorithm,” *Proc. 13th Int. Conf. Mach. Learn.*, 1996, doi: 10.1.1.133.1040.

[43] L. Breiman, “Random Forests,” *Mach. Learn.*, 2001, doi: 10.1023/a:1010933404324.

[44] I. Reis, D. Baron, and S. Shahaf, “Probabilistic random forest: A machine learning algorithm for noisy datasets,” *arXiv*, 2018, doi: 10.3847/1538-3881/aaf101.

[45] Z. Michalewicz, D. B. Fogel, Z. Michalewicz, and D. B. Fogel, “Why Are Some Problems Difficult to Solve?,” *How to Solve It Mod. Heuristics*, 2004, doi: 10.1007/978-3-662-07807-5_2.

[46] H. Hoos and T. Stützle, “Stochastic Local Search Algorithms: An Overview,” 2015, doi: 10.1007/978-3-662-43505-2_54.

[47] J. Sarma and K. A. De Jong, “An Analysis of Local Selection Algorithms in a Spatially Structured Evolutionary Algorithm,” *Proc. 7th Int. Conf. Genet. Algorithms (ICGA 1997)*, 1997, doi: -.

[48] M. Coli, G. Gennuso, and P. Palazzari, “New crossover operator for genetic algorithms,” *Proc. IEEE Conf. Evol. Comput.*, 1996, doi: 10.1109/icec.1996.542361.

[49] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri, and V. B. S. Prasath, “Choosing mutation and crossover ratios for genetic algorithms-a review with a new dynamic approach,” *Inf.*, 2019, doi: 10.3390/info10120390.

[50] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programing*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008.

[51] J. Koza, “Genetic programming - on the programming of computers by means of natural selection,” 1993, doi: 10.1007/bf00175355.

[52] M. O’Neill and C. Ryan, *Grammatical evolution - evolutionary automatic programming in an arbitrary language*. Springer, 2003.

[53] M. Nicolau and A. Agapitos, “Understanding grammatical evolution: Grammar

design," in *Handbook of Grammatical Evolution*, 2018.

[54] R. Harper, "GE, explosive grammars and the lasting legacy of bad initialisation," *2010 IEEE World Congr. Comput. Intell. WCCI 2010 - 2010 IEEE Congr. Evol. Comput. CEC 2010*, 2010, doi: 10.1109/cec.2010.5586336.

[55] D. Fagan, M. Fenton, and M. O'Neill, "Exploring position independent initialisation in grammatical evolution," *2016 IEEE Congr. Evol. Comput. CEC 2016*, 2016, doi: 10.1109/cec.2016.7748331.

[56] M. O'Neill, C. Ryan, M. Keijzer, and M. Cattolico, "Crossover in Grammatical Evolution: The Search Continues," 2001, doi: 10.1007/3-540-45355-5_27.

[57] J. Byrne, M. O'Neill, and A. Brabazon, "Structural and nodal mutation in grammatical evolution," 2009, doi: 10.1145/1569901.1570215.

[58] F. Rothlauf, *Representations for genetic and evolutionary algorithms*. Springer Berlin Heidelberg, 2006.

[59] P. A. Whigham, J. MacLaurin, G. Dick, and C. A. Owen, "Examining the 'best of both worlds' of grammatical evolution," *GECCO 2015 - Proc. 2015 Genet. Evol. Comput. Conf.*, 2015, doi: 10.1145/2739480.2754784.

[60] N. A. M. Lourenço, "Enhancing Grammar-Based Approaches for the Automatic Design of Algorithms," 2015.

[61] G. S. Imai Aldeia and F. O. De Franca, "Lightweight Symbolic Regression with the Interaction - Transformation Representation," *2018 IEEE Congr. Evol. Comput. CEC 2018 - Proc.*, 2018, doi: 10.1109/cec.2018.8477951.

[62] S. S. Kim and K. C. Kwak, "Development of quantum-based adaptive neuro-fuzzy networks," *IEEE Trans. Syst. Man. Cybern.*, 2010, doi: 10.1109/tsmcb.2009.2015671.