isec
Engenharia

MESTRADO EM INFORMÁTICA E SISTEMAS

TILS: um método de engenharia de features baseado em algoritmos de pesquisa local

Autor

João Filipe Pereira de Sousa

Orientador

Francisco José Baptista Pereira

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

Coimbra, novembro de 2022

isec
Engenharia

DEPARTAMENTO DE INFORMÁTICA E SISTEMAS

TILS: um método de engenharia de features baseado em algoritmos de pesquisa local

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

João Filipe Pereira de Sousa

Orientador

Francisco José Baptista Pereira

Co-Orientador

Teresa Cunha Oliveira

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

Coimbra, novembro de 2022

AGRADECIMENTOS

Concluída mais uma fase na minha vida académica, é necessário agradecer a todos aqueles que, de uma ou de outra forma, me ajudaram a chegar até aqui. Desta forma gostaria de deixar um agradecimento especial:

Ao meu orientador - Doutor Francisco José Baptista Pereira – por toda a disponibilidade e ajuda prestada ao longo deste trabalho.

À minha co-orientadora - Doutora Teresa Cunha Oliveira – por toda a disponibilidade e ajuda prestada ao longo deste trabalho.

Ao Instituto Superior de Engenharia de Coimbra e a todos os professores por me terem oferecido uma formação superior de elevada qualidade, bem como excelentes condições de aprendizagem.

Aos colegas de curso por toda a motivação, companhia e disponibilidade para trabalhar em grupo.

A todos os meus amigos por toda a motivação, amizade e companhia ao longo de todo este tempo.

Aos meus pais por todo apoio dado e pelo esforço que tiveram para me ajudarem a chegar até aqui.

À minha namorada por todo o apoio, paciência, companhia e motivação prestadas ao longo desta fase da minha vida.

RESUMO

Engenharia de features é o termo utilizado para definir um conjunto de técnicas utilizado na criação e manipulação de features, tendo como objetivo melhorar o resultado dos algoritmos de machine learning. É um dos passos mais importantes na construção de um modelo. Contudo a engenharia de features não se trata de apenas selecionar as melhores features para um modelo, é um processo que abrange a transformação das features de forma a extrair o maior potencial possível bem como a criação de novas features.

Neste trabalho é realizado um estudo sobre métodos de engenharia de features e é desenvolvido um novo método de engenharia de features (TILS) baseado em algoritmos de pesquisa local como o trepa colinas e o ILS – Iterated Local Search. O método desenvolvido é comparado com alguns métodos de engenharia de features existentes e testado com um dataset biológico real. O método desenvolvido tem como objetivo a seleção e criação de novas features para qualquer problema de forma a retirar o melhor proveito possível do dataset. A aplicação do TILS em comparação com os métodos de engenharia de features acabou por ser benéfica e resultou num resultado superior face aos métodos de comparação selecionados.

Palavras-Chave: Engenharia de features, Seleção de features, Extração de features, Pesquisa Local, ILS, Trepa Colinas

ABSTRACT

Feature engineering is the term used to define a set of techniques used in the creation and manipulation of features, with the aim of improving the result of machine learning algorithms. It is one of the most important steps in building a model. However, feature engineering is not just about selecting the best features for a model, it is a process that encompasses the transformation of features in order to extract the greatest possible potential as well as the creation of new features.

In this work a study on feature engineering methods is carried out and a new feature engineering method (TILS) is developed based on local search algorithms such as hill climbing and ILS – Iterated Local Search. The developed method is compared with some existing feature engineering methods and tested with a real biological dataset. The developed method aims to select and create new features for any problem in order to take the best possible advantage of the dataset. The application of TILS in comparison with the feature engineering methods turned out to be beneficial and resulted in a superior result compared to the selected comparison methods..

Key-words: Feature engineering, Feature selection, Feature extraction, Local Search, ILS, Hill Climbing

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ÍNDICE DE GRÁFICOS

ÍNDICE DE FÓRMULAS

SIMBOLOGIA E ABREVIATURAS

ELA	Esclerose Lateral Amiotrófica
ALS	Amyotrophic Lateral Sclerosis
χ^2	Qui-Quadrado
ERF	Eliminação Recursiva de Features
GI	Ganho de Informação
LARS	Least-angle regression
PG	Programação Genética
ML	Machine learning
PL	Pesquisa Local
TC	Trepa Colinas
ILS	Iterated Local Search
TILS	Tendency Iterated Local Search

1 Introdução

A engenharia de features tem atraído cada vez mais a atenção na área de machine learning (ML), desempenhando um papel significativo na melhoria da qualidade dos modelos de ML (Fontes, 2020). Dado o elevado número de abordagens existentes é difícil identificar qual o melhor método, e qual se adapta a cada tipo de problema. Com o avanço da tecnologia e da ciência é cada vez mais fácil obter maiores volumes de dados, dados estes que podem conter informação essencial ou não, informação correta ou incorreta, informação em falta ou até informação em excesso. É desta forma que a engenharia de features ajuda no processo de seleção e extração da melhor informação para os processos de ML. A engenharia de features é um aspeto importante do processo de ML (Domingos, 2012) e nunca deve ser ignorado (Banerjee et al., 2016).

A engenharia de features pode ser entendida como o processo de seleção, manipulação e transformação de dados não tratados em dados que possam ser usados em modelos de ML, desta forma pode ser categorizada dois grupos: extração de features e seleção de features. A extração de features é o processo de obtenção de recursos uteis através dos dados existentes, extrair informações valiosas através de dados em bruto. A seleção de features é o processo de escolher um subconjunto de features através do conjunto original de features.

Este trabalho teve como objetivo desenvolver um método de engenharia de features baseado em algoritmos de pesquisa local, capaz de extrair e manipular features, bem como investigar alguns dos métodos existentes para engenharia de features.

No presente capítulo é apresentado o enquadramento do tema, a motivação, os objetivos e metodologia utilizada, bem como a estrutura do relatório.

1.1 Motivação

Uma das principais motivações para a realização deste trabalho foi o desenvolvimento de um novo método de engenharia de features de forma a ajudar o grupo de investigação MitoXT (*MitoXT - Toxicologia e Terapêutica Experimental Mitocondrial*, 2020) do Centro de Neurociências e Biologia Celular (*Centro de Neurociências e Biologia Celular*, 2020) a identificar os melhores biomarcadores de bioenergética mitocondrial para a identificação e distinção de doentes com esclerose lateral amiotrófica (ELA). Outro fator de motivação é a intervenção no processo de engenharia de features visto ser um passo importante e onde os resultados são importantes. Outro fator de motivação são os dados fornecidos, pois dizem respeito a um conjunto de indivíduos onde foram recolhidas amostras para que este estudo fosse possível. Tratando-se de um conjunto de dados real, a motivação é sempre superior pois o nosso contributo pode ser uma mais-valia para o grupo de investigação. Por fim

é motivador perceber se o nosso contributo foi ou não positivo para o grupo de investigação.

1.2 Descrição do problema

Quando estamos a estudar problemas biológicos, por vezes as amostras existentes são raras comparadas com outro tipo de problemas, onde a quantidade de dados existente é enorme. Por outro lado, podem ser dados difíceis de interpretar devido à complexidade do problema ou até da qualidade das amostras recolhidas. Em certos casos, estes dados podem estar desorganizados, difíceis de interpretar e dai a necessidade de obter o melhor conjunto de atributos para a aplicação de ML.

O grupo de investigação MitoXT (*MitoXT - Toxicologia e Terapêutica Experimental Mitocondrial*, 2020) pretende através de processos de ML mostrar que é possível classificar um doente com ELA com a maior certeza possível. Desta forma foi recolhido um conjunto de dados relativos a amostras de pacientes com e sem ELA e será a partir deste conjunto de dados que é pretendido obter a melhor classificação possível.

1.3 Objetivos do trabalho

O trabalho descrito neste documento terá como principal objetivo desenvolver um método de engenharia de features baseado no ILS (Iterated Local Search) (H. Lourenço et al., 2010) e usar um dataset preliminar como caso de teste. Depois pretende-se comparar o método desenvolvido com as abordagens existentes e validar com o dataset fornecido pelo grupo de investigação MitoXT.

De forma a validar o novo método de engenharia de features, foi usado um dataset preliminar como caso de teste, depois de afinado e testado o novo método foi validado com o dataset fornecido pelo grupo de investigação MitoXT.

Outro objetivo foi identificar algumas abordagens para engenharia de features e perceber a sua forma de funcionamento e comparar as abordagens escolhidas com o método desenvolvido. Por último verificar se o método desenvolvido é relevante ou não para o problema.

1.4 Estrutura do relatório

Este relatório é composto por 7 capítulos, sendo o presente capítulo uma breve descrição do problema para o qual se pretendeu obter os melhores resultados possíveis, seguida da apresentação dos principais objetivos do trabalho desenvolvido.

No segundo capítulo são explicados os conceitos gerais sobre ML e é feito um estudo sobre alguns métodos para engenharia de features. São explorados de uma forma geral e onde é conhecido o seu funcionamento.

No terceiro capítulo é descrito o método desenvolvido e as opções tomadas.

No quarto capítulo é apresentado o dataset preliminar “MicroRNA”. É descrito o design experimental. São aplicados os métodos e o método desenvolvido ao dataset. São afinados os parâmetros do método desenvolvido e por fim os resultados.

No quinto capítulo é descrito e analisado em detalhe o dataset principal “Amyotrophic Lateral Sclerosis (ALS)” utilizado para este trabalho.

No sexto capítulo são aplicados os métodos ao dataset principal bem como o método desenvolvido. São também analisados os resultados obtidos.

Por fim, no sétimo capítulo são apresentadas as conclusões sobre o trabalho realizado e trabalho futuro.

2 Métodos existentes para engenharia de features

Neste capítulo abordamos alguns conceitos gerais sobre ML e dois tipos de abordagens para engenharia de features, sendo a primeira abordagem sobre métodos tradicionais de seleção e a segunda abordagem relativa a frameworks de engenharia de features.

2.1 Conceitos Gerais de Machine Learning

Para a realização deste trabalho é necessário perceber e compreender alguns conceitos básicos de ML:

- **Machine Learning**: é a ciência (e a arte) de programar computadores para que possam aprender através dos dados (Géron, 2019; Hamet & Tremblay, 2017), ou de uma forma mais genérica, uma área de estudo que dá aos computadores a capacidade de aprender sem ser explicitamente programado para isso (Wiederhold & McCarthy, 1992). Um exemplo muito utilizado de ML é o filtro de spam do email, que através de exemplos de emails considerados "spam" (por exemplo, identificados pelos utilizadores) e exemplos de emails "normais" (não spam) pode aprender a sinalizar o que é spam ou não. Os exemplos que o sistema usa para aprender são chamados de conjunto de treino. Cada exemplo de treino é chamado de instância (ou amostra). Neste caso, a tarefa X é assinalar spam para novos emails, a experiência Y são os dados de treino e o desempenho Z a medida que precisa de ser definida; por exemplo o rácio de classificação de emails correta. Neste caso concreto a performance obtida é chamada de accuracy e é usada em problemas de classificação.
- **Tipos de problemas de ML**: Existem muitos tipos de problemas de ML que podem ser categorizados em várias categorias (Géron, 2019). Neste trabalho vamos dar importância aos problemas que são treinados com supervisão humana. Esta categoria pode ser dividia em problemas de ML supervisionados e não supervisionados. Nos problemas supervisionados o conjunto de treino que é fornecido ao algoritmo inclui o objetivo desejado, chamada de label ou target. Nos problemas supervisionados encontramos dois tipos de problemas: classificação e regressão. O exemplo do filtro de spam é um bom exemplo de um problema de classificação, ele é treinado com exemplos de emails junto com a sua classe (spam ou não) e deve aprender a classificar novos exemplos. Nos problemas de regressão o objetivo é prever um valor numérico, como por exemplo o preço de um carro através de características do mesmo (quilómetros, idade, marca, etc.). Para poder treinar estes sistemas é necessário fornecer muitos exemplos de carros incluindo as suas características e o seu preço (label ou target). Os problemas não

supervisionados são aqueles em que os dados não possuem a informação da solução e o sistema tenta aprender sem ajuda. Estes tipos de problemas são usados para explorar dados desconhecidos. Eles podem revelar padrões que podem ter sido perdidos ou examinar grandes conjuntos de dados que seriam demais para serem resolvidos por um ser humano. Os problemas de ML não supervisionados usam uma variedade de algoritmos para ajustar os dados em grupos amplos, agrupamentos e associações. Um exemplo de agrupamento ou clustering pode ser o seguinte, existe bastante informação acerca dos utilizadores de um determinado site. Podemos correr um algoritmo de clustering para tentar detetar grupos semelhantes de utilizadores, mas em momento algum dissemos ao algoritmo a qual grupo a que o utilizador pertence, ele próprio forma os grupos. Por exemplo, pode identificar que 50% dos utilizadores são homens, gostam de banda desenhada e utilizam o site à noite, enquanto 30% dos utilizadores são jovens e utilizam o site ao fim de semana. Não vamos entrar em mais detalhes porque o objetivo é perceber a diferença entre problemas de ML supervisionados e não supervisionados.

- **Dados**: é um pilar fundamental para um processo de ML. São a base utilizada para que os algoritmos consigam aprender, evoluir e exibir seus resultados. Os datasets são bases de dados especificas que servem de amostras para o treino de algoritmos de ML. Geralmente possuem um formato tabular, com linhas e colunas bem definidas e organizadas com informações claras sobre a sua finalidade. Nas linhas, podemos ter, por exemplo o número do registo e nas colunas as características. As linhas normalmente refletem o número de amostras ou quantidade de registos e as colunas as características das linhas. Por exemplo, numa loja de vendas as linhas representam as vendas da loja, produto x ou y e as colunas o tipo de pagamento, o valor da venda, a data da venda, o colaborador, etc., dessa mesma venda. Datasets com muitas colunas e poucas linhas tendem a prejudicar a performance do processo de ML (Urbanowicz et al., 2018) e datasets com poucas colunas e muitas linhas também tendem a prejudicar a performance dos modelos de ML (Vasconcelos, 2017). O seu formato pode variar, mas por norma são ficheiros TXT, CSV e XLS. Podem ser também em formato de base de dados, numa tabela ou várias.
- **Algoritmos de ML:** Existem vários algoritmos de ML que podem ser aplicados nos vários tipos de problemas de ML, neste documento vamos abordar os principais algoritmos de ML (Mahesh, 2019) como regressão linear, regressão logística, árvore de decisão, SVM, Naive Bayes, KNN, K-Means e Random Forest. A regressão linear é usada para estimar valores reais (custos de casas, número de ligações, número de vendas, etc.) com base em variáveis continuas (Maulud & Abdulazeez, 2020). É estabelecida uma relação entre as variáveis independentes e variáveis dependentes ajustando a melhor linha, essa linha de ajuste é conhecida como a linha de regressão e é representada por uma equação linear $Y = aX + b$. A regressão Logística é utilizada para determinar

valores discretos (valores binários como 0 ou 1, sim ou não, verdadeiro ou falso) com base num determinado conjunto de variáveis independentes (Nusinovici et al., 2020). A regressão logística prevê a probabilidade da ocorrência de um evento ajustando os dados a uma função logística, prevendo a probabilidade de valores de saída entre 0 e 1. Árvore de decisão é um mapa dos possíveis resultados de uma série de escolhas relacionadas, geralmente começa com um único nó, que se divide em possíveis resultados (Charbuty & Abdulazeez, 2021). Cada um desses resultados leva a nós adicionais que se ramificam em outras possibilidades, como resultado temos a forma de uma árvore. SVM (Support Vector Machine) é um algoritmo no qual representamos os dados em bruto na forma de pontos num espaço n-dimensional (onde n é o Número de recursos existentes) (Somvanshi et al., 2016). O valor de cada recurso é então vinculado a uma coordenada especifica, facilitando a classificação dos dados. As linhas chamadas de classificadores podem ser usadas para dividir os dados e representá-los em forma de um gráfico. Por exemplo, se tivéssemos apenas duas características como a altura e o comprimento do cabelo de um individuo, primeiro, marcamos cada ponto correspondente as duas variáveis num espaço bidimensional onde cada ponto tem as duas coordenadas (estas coordenadas são conhecidas como vetores de suporte). Agora são encontradas as linhas que dividem os dados entre os dois grupos, a linha que melhor dividir os dois grupos é considerada a melhor. Naive Bayes é um algoritmo baseado no teorema de Bayes («Teorema de Bayes», 2022) com pressuposto de independência entre os preditores. Em termos simples, um classificador Naive Bayes assume que a presença de uma característica particular de uma classe não está relacionada à presença de qualquer outra característica (Ampomah et al., 2021). Por exemplo, uma fruta pode ser considerada uma maçã se for vermelha, redonda e com cerca de 6 cm de diâmetro. Mesmo que essas características dependam umas das outras ou da existência de outras características, um classificador ingênuo de Bayes consideraria que todas essas propriedades contribuem independentemente para a probabilidade de que essa fruta seja uma maçã. KNN (k- Vizinhos mais próximos) é um algoritmo que armazena todos os casos disponíveis e classifica novos casos por maioria de votos de seus k vizinhos. O caso atribuído à classe é mais comum entre seus K vizinhos mais próximos medidos por uma função de distância (Euclidiana, Manhattan,etc.). O KNN pode ser facilmente entendido num caso da vida real, se pretendemos obter informações sobre uma pessoa, faz sentido conversar com seus amigos e colegas. K-Means é um algoritmo que resolve problemas de agrupamento (Itoo et al., 2021). Os conjuntos de dados são classificados num determinado número de clusters (K) de forma que todos os pontos de dados dentro de um cluster sejam homogêneos e heterogêneos em relação aos dados de outros clusters. O K-Means começa por escolher um número de k pontos, chamados centroides, para cada cluster. Cada ponto forma um cluster com os centroides mais

próximos (K clusters). Depois cria centroides com base nos membros dos clusters existentes, com esses novos centroides, a distância mais próxima para cada ponto é determinada. Este processo é repetido até que os centroides não mudem. O Random Forest é um conjunto de árvores de decisão (conhecidas como “Forest”) (Abdulkareem & Abdulazeez, 2021), para classificar um objeto com base nos seus atributos cada árvore dá uma classificação e dizemos que a árvore “vota” para aquela classe. A floresta escolhe a classificação com mais votos (sobre todas as árvores da floresta).

- **Treino e teste do modelo**: Nos problemas de ML existe sempre a necessidade de saber o quão bom é o nosso modelo e de que forma podemos melhorar o modelo com base nas nossas necessidades. Nos processos de ML temos um problema a resolver, que geralmente está relacionado com a avaliação dos dados e fazer previsões. Para resolver o problema temos algumas ferramentas à nossa disposição, os algoritmos (como os mencionados no tópico acima). Como sabemos qual o melhor modelo para o problema? Para responder a esta questão temos de usar ferramentas de medição, mas para isso temos de treinar, testar, avaliar e validar os nossos modelos de forma a tomar as melhores decisões possíveis com os dados existentes. Ao determinar o melhor modelo com base no algoritmo que se achar mais vantajoso para cada tipo de problema, queremos que o resultado generalize bem para dados não vistos pelo modelo. para fazer isso, dividimos o dataset em dois conjuntos distintos: conjunto de treino e conjunto de teste. Normalmente o conjunto de treino é sempre maior que o conjunto de teste, cerca de 80% para o conjunto de teste e 20% para conjunto de treino.

- **Tipos de avaliação do modelo:** Depois de desenvolvido o modelo queremos descobrir se o seu desempenho é bom, para isso existem várias métricas que se podem usar e dependendo do tipo de problema podem ser mais adequadas ou não (Zhou et al., 2021). Podemos dividir as métricas de avaliação do modelo consoante o tipo de problema: classificação ou regressão. Nos problemas de classificação a melhor forma de avaliar a sua performance é utilizando a matriz de confusão (Géron, 2019). A matriz de confusão é uma tabela que mostra a frequência de classificação para cada classe de modelo em quatro classes distintas: Verdadeiros Positivos (VP), Falsos Positivos (FP), Falsos Verdadeiros (FV) e Falsos Negativos (FN). Verdadeiros Positivos (VP) ocorre quando no conjunto teste, a classe que estamos a considerar foi prevista corretamente. Por exemplo, quando a mulher está grávida e o modelo previu corretamente que ela está grávida. Falsos Positivos (FP) ocorre quando no conjunto de teste, a classe que estamos a considerar foi prevista incorretamente. Exemplo: a mulher não está grávida, mas o modelo disse que ela está. Falsos Verdadeiros (FV) ocorre quando no conjunto de teste, a classe que não estamos a considerar foi prevista corretamente. Exemplo: a mulher não estava grávida, e o modelo previu corretamente que ela não está. Falsos Negativos (FN) ocorre

quando no conjunto de teste, a classe que não estamos a considerar foi prevista incorretamente. Por exemplo, quando a mulher está grávida e o modelo previu incorretamente que ela não está grávida. A matriz de confusão dá-nos bastante informação, mas por vezes precisamos de uma medida mais concisa, desta forma podemos retirar várias métricas da tabela como por exemplo: accuracy, precision, recall e f1 score. A accuracy diz quanto o modelo acertou das previsões possíveis. É a razão entre o somatório das previsões corretas (verdadeiros positivos com verdadeiros negativos) sobre o somatório das previsões. A precision é uma métrica que indica, das classificações positivas do modelo quantas foram acertadas. Portanto, é definida como o número de verdadeiros positivos dividido pelo número de verdadeiros positivos mais os falsos positivos. Recall é uma métrica que indica, das amostras positivas existentes, quantas o modelo consegui classificar corretamente. Portanto, é definida como o número de verdadeiros positivos dividido pelo número de verdadeiros positivos mais os falsos negativos. F1 Score é uma métrica que permite avaliar em conjunto a precision e a recall juntas, sendo uma média harmónica entre ambas. Se o seu valor for baixo, é indicador que a precision ou a recall é baixa. Nos problemas de regressão também existem várias métricas, mas as mais usadas (Géron, 2019) são a Raiz do Erro Quadrático Médio (RMSE) e o Erro Médio Absoluto (MAE). A Raiz do Erro Quadrático Médio dá uma ideia da quantidade de erros que o sistema tipicamente comete nas suas previsões, com um peso maior par grandes erros. Embora o RMSE seja geralmente a medida de desempenho preferida para tarefas de regressão (Géron, 2019), em alguns contextos pode-se preferir a utilização de outra métrica. Nesse caso, poderemos utilizar o MAE. O MAE mede a média da diferença entre o valor real com o valor predito, por haver valores positivos e negativos, é adicionado um módulo entre a diferença dos valores. Quanto mais baixo for o valor do MAE melhor são os resultados do modelo.

- **Afinação de parâmetros**: Depois de avaliado o modelo treinado, existe a necessidade de tentar melhorar o mesmo através da afinação de parâmetros (Elgeldawi et al., 2021). Estes parâmetros também podem ser conhecidos como “hiperparâmetros”. São parâmetros cujos valores controlam o processo de ML e determinam os valores dos parâmetros do modelo de ML. O prefixo “hiper” sugere que são parâmetros de “nível superior” que controlam o processo de aprendizagem e os parâmetros do modelo que dele resultam. A afinação de coeficientes ou pesos de uma regressão linear ou de uma regressão logística são exemplos de afinação de parâmetros bem como a afinação do número de centroides no K-Means. A afinação dos melhores valores para os parâmetros dos algoritmos de ML é muito importante porque tem impacto direto no desempenho do modelo.
- **Seleção do modelo de ML:** Com tantos modelos possíveis para utilizar em problemas de ML é preciso ter algumas considerações no processo de seleção

do modelo de ML (Zimányi et al., 2021). No processo de ML existem dois tipos de erros que são mais frequentes de acontecer (Ghosh & Dasgupta, 2022): "Underfitting" e "Overfitting". Underfitting diz respeito a um modelo que é incapaz de identificar a relação entre as variáveis de entrada e as variáveis de saída. Overfitting diz respeito a um modelo que se ajusta demasiado aos dados de treino, ou seja, memoriza os dados de treino e não generaliza para dados novos. Uma solução para este problema é a utilização de "cross validation" (Fuhr, 2022). Cross Validation (CV) é uma técnica de reamostragem que ajuda a assegurar a performance do modelo e que consiste em separar os dados em conjuntos, onde um conjunto é utilizado para treino e o outro é utilizado para teste e avaliação do modelo. Num processo de CV básico existe um parâmetro "K" ou "Kfold" que é o responsável por dividir o dataset de forma aleatória em K subconjuntos idênticos em tamanho. A cada iteração é formado um conjunto de K-1 subconjuntos que serão usados para treino e o restante para teste. Este processo é repetido K vezes de forma a usar todos os K subconjuntos sendo atribuído para cada um deles uma avaliação. No fim o resultado é a média das K iterações. Existem várias técnicas de CV mas optamos por utilizar neste trabalho a variante LOOCV (leave-one-out cross validation). LOOCV é uma versão do CV mais exaustiva uma vez que requer que o modelo seja criado e avaliado para cada exemplo do conjunto de dados de treino. A vantagem deste método é a estimativa mais afinada do modelo e a sua grande desvantagem é o custo computacional (Chaturvedi, 2021). A diferença do LOOCV para o CV é que em vez de um K=10 por exemplo temos um K igual ao número de exemplos do dataset.

2.2 Métodos Tradicionais de Engenharia de Features

Esta secção diz respeito aos métodos tradicionais para seleção de features. Os métodos de seleção de features são responsáveis por selecionar as melhores features para o modelo de ML através das features existentes. Existem muitos métodos de seleção de features mas optamos por escolher apenas cinco que consideramos relevantes.

2.2.1 Qui-Quadrado

O qui-quadrado é um método matemático criado por Karl Pearson no ano de 1900 (Pearson, 1900) que avalia os atributos/features individualmente em relação à classe através da medida de χ^2. Quanto maior for o valor da medida χ^2 mais provável é a correlação entre o atributo e a classe, portanto deve ser considerado como relevante. Se o valor for próximo de zero quer dizer que o atributo e a classe são independentes, nestes casos não devem ser considerados relevantes. Antes de poder aplicar a

fórmula do qui-quadrado é necessário criar uma tabela de contingência, esta tabela contém a contagem ou frequência dos registos/observações correspondentes às categorias das linhas e colunas. Imaginemos que pretendemos criar uma tabela de contingência entre o sexo de uma pessoa (masculino, feminino) e os seus interesses (arte, cinema, matemática, etc.). A tabela de contingência a criar contém a relação entre o sexo e o número de ocorrências dos interesses. Por exemplo, uma pessoa do sexo masculino, na categoria de arte aparece 10 vezes, na categoria de cinema 2 vezes e na categoria de matemática 5. A intenção desta tabela é ajudar a perceber se uma variável é dependente de outra. Mesmo assim, usando a tabela de contingência é difícil perceber se as variáveis são ou não dependentes, é então que surge o teste do qui-quadrado.

Formula do Qui-quadrado:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Fórmula 1 - Qui-Quadrado

- O_i é o número de observações na classe i
- E_i é o número esperado de observações na classe i

Seguindo o exemplo do sexo/interesse acima referido, o número de observações para cada categoria (masculino, feminino) pode ser ou não o mesmo, então podemos calcular a frequência esperada em cada grupo de interesse e verificar se os interesses por sexo resultam em frequências semelhantes ou diferentes.

Portanto, para cada um dos atributos/features calcula-se a estatística de qui-quadrado, consoante o valor obtido este atributo pode ser considerado relevante ou não. Se o valor obtido for elevado, maior é a probabilidade de correlação entre o atributo e a classe, logo o atributo é dependente da classe e deve ser considerado como relevante, caso contrário, o valor é próximo de zero indica que o a relação entre a classe e o atributo é independente, logo não deve ser considerado.

2.2.2 Eliminação Recursiva de Features

A eliminação recursiva de features (ERF) é um método comum para selecionar as melhores “n” features para construção de modelos (Senan et al., 2021). Dado um estimador externo que atribui pesos às features (por exemplo, coeficientes de um modelo linear), o objetivo da ERF é selecionar as features considerando recursivamente conjuntos cada vez menores de features. Em primeiro lugar, o modelo

é treinado no conjunto inicial de features e a importância de cada uma é obtida através de qualquer atributo específico. Em seguida, as features menos importantes são removidas do conjunto original de features. Esse procedimento é repetido recursivamente no dataset até que o número desejado de características a serem selecionadas seja obtido. A Figura 1 mostra o processo.

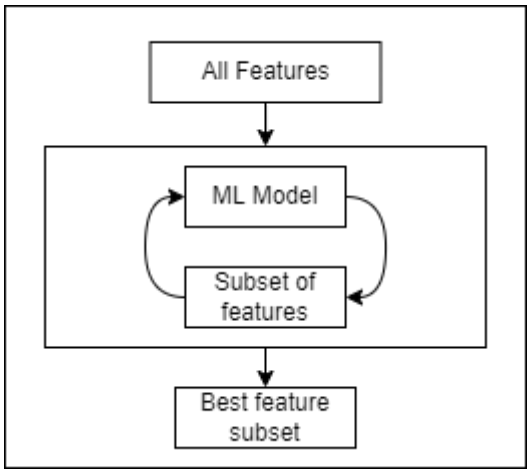

Figura 1 - Funcionamento geral do ERF

Este método é mais dispendioso em termos de processamento porque percorre todos os subconjuntos de forma exaustiva, e quanto maior for o espaço de procura maior é o tempo de execução e por sua vez o seu custo a nível de processamento também é superior (Rani et al., 2021).

2.2.3 Regressão de Ridge

A regressão de Ridge (Hoerl & Kennard, 1970) é uma versão regularizada da regressão linear, onde é acrescentado um fator de regularização à função de custo. O processo de regularização visa adicionar uma penalidade aos parâmetros de forma que o modelo seja generalizado e não específico. A regressão de Ridge também conhecida como L2, tem como objetivo aproximar os coeficientes de algumas features a zero através de penalização e dessa forma descartar essas mesmas features. Esta penalização é o parâmetro “alfa” ou “L2”. Este parâmetro minimiza o tamanho dos coeficientes, evitando que sejam removidos do modelo. Para isso a regressão de Ridge baseia-se na soma dos valores dos coeficientes ao quadrado. Para controlar esta penalização é utilizado um parâmetro chamado “lambda” que controla a ponderação da penalidade para a função de perda. O seu valor por defeito é 1, mas pode ser ajustado dependendo do objetivo. Utilizando o valor de 1 faz com que o peso da penalização seja máximo, utilizando um valor perto de 0, a penalização é mínima e a fórmula da regressão de ridge transforma-se numa simples regressão linear. A fórmula abaixo mostra a regressão de Ridge.

$$\mathrm{RSS}_{\mathrm{ridge}} = \sum_{i=1}^{n} \left[y_i - (\mathbf{w} \cdot \mathbf{x}_i + b) \right]^2 \boxed{+ \alpha \sum_{j=1}^{p} w_j^2}$$

$$\text{regularização } \ell_2$$

Fórmula 2 - Regressão de Ridge

Em resumo, a regressão de Ridge é um método de regularização que tem como objetivo atenuar atributos que estejam relacionados e que podem aumentar o ruido do modelo.

2.2.4 Regressão de Lasso

A regressão de Lasso (Tibshirani, 1996) é também um método de regularização. Tal como acontece com a regressão de Ridge, a regressão de Lasso utiliza um fator de penalização chamado de “L1”, este é idêntico ao “L2”, mas ao invés de utilizar o valor ao quadrado utiliza o valor absoluto. Se duas features estiverem linearmente correlacionadas, a sua presença em simultâneo fará aumentar a função de custo, como a regressão de lasso tenta reduzir o coeficiente das features menos importantes para zero, as melhores features apresentam valores superiores a zero. A fórmula abaixo reflete a regressão de Lasso.

$$\mathrm{RSS}_{\mathrm{lasso}} = \sum_{i=1}^{n} \left[y_i - (\mathbf{w} \cdot \mathbf{x}_i + b) \right]^2, \boxed{+ \alpha \sum_{j=1}^{p} |w_j|}$$

$$\text{regularização } \ell_1$$

Fórmula 3 - Regressão de Lasso

Em resumo, a regressão de Lasso é um método de regularização que tem como objetivo é descartar features com coeficientes iguais a zero.

2.2.5 Ganho de Informação

O ganho de informação (GI) é bastante utilizado para seleção de features em problemas de classificação (Jadhav et al., 2018). Está habitualmente relacionado com árvores de decisão e é um método baseado na entropia. A entropia é o termo utilizado

para medir a pureza ou impureza de um determinado conjunto. Entropia pode ser definida como a medida que nos diz o quanto os dados estão desorganizados e misturados. Quanto maior for a entropia menor é o ganho de informação. Quanto menor for a entropia maior é o ganho de informação. O primeiro passo para obter o ganho de informação de uma feature é calcular o grau de entropia do conjunto de treino, definido pela Fórmula 4.

$$Entropia(N) = -\sum_{j=1}^{m} p_j * \log_2 p_j$$

Fórmula 4 - Entropia

Onde N é qualquer conjunto de amostras, podendo representar a base completa (nó raiz) ou partições da base de dados, m é o número de classes e p_j é a proporção de amostras pertencendo à classe onde ocorre N.

O ganho de informação é calculado quando estamos a comparar a entropia do estado atual dos dados com a entropia que seria obtida com uma nova ramificação. O valor do ganho de informação pode ser obtido através da seguinte formula:

$$Ganho\_de\_informação = Entropia(pai) - \sum_{i=1}^{n} peso(filho_i) + Entropia(filho_i)$$

Fórmula 5 - Ganho de Informação

O número de filhos presentes no somatório depende de como a ramificação será feita, normalmente são gerados dois nós filhos, onde o peso para cada um desses filhos é calculado dividindo o total de elementos em um nó filho dividido pelo número de elementos do nó pai. A fórmula do peso dos filhos pode ser representada da seguinte forma:

$$peso(filho) = \frac{nº\ amostrasFilho}{nº\ amostrasPai}$$

Fórmula 6 - Peso de cada filho

Calculado o ganho de informação para cada feature, os valores obtidos são ordenados por ordem decrescente. Definindo um limite, incluíamos todas as features acima desse limite e o resultado são as melhores features segundo o método.

2.3 Frameworks de Engenharia de Features

Esta secção diz respeito aos métodos de extração de features. Os métodos de extração de features são responsáveis combinar as features existentes de forma a

obter novas features com mais utilidade. Existem vários métodos de extração de features, optamos por escolher apenas quatro que consideramos mais relevantes.

2.3.1 AutoFeat

Autofeat é uma biblioteca em Python lançada em janeiro de 2019 por Franziska Horn, que fornece ferramentas para classificação e seleção de features de forma automática (Horn et al., 2019). Esta ferramenta funciona em três etapas para criar features: a primeira etapa é a construção de features não lineares, ou seja, são criadas features através da aplicação de fórmulas matemáticas como por exemplo: $\log x$, $\sqrt{x}$, $\frac{1}{x}$, x^2, x^3, $|x|$, e^x, $2x$, $\sin x$, $\cos x$. Depois são combinados pares de features com operadores de adição, subtração e multiplicação. Por fim são aplicadas novamente as mesmas fórmulas matemáticas do primeiro passo. Isto resulta num número de features maior, por exemplo, se o conjunto inicial de features for de 3, a aplicação da primeira fase, transformações não lineares, resulta em cerca de 20 features, depois fazendo combinações entre elas com os operadores resulta em cerca de 750 novas features, aplicando o terceiro passo ultrapassamos as 4000 features (Horn et al., 2019). Depois de criadas as features a próxima fase é escolher entre o universo de novas features aquelas que trazem benefício ao modelo. O primeiro passo é remover as features que estão altamente correlacionadas. O segundo passo é aplicar o algoritmo de Lasso LARS («Least-Angle Regression», 2021). Este foi criado por Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani e é uma evolução da regressão de Lasso, ou seja, não requer um valor de lambda fixo para a função de penalização, o valor é ajustado automaticamente pelo algoritmo (Efron et al., 2004). No fim de aplicar o algoritmo a todas as features, são escolhidas aquelas que apresentarem menor correlação.

Este é o funcionamento geral do AutoFeat, mas também pode ser utilizado só para seleção das melhores features de um dataset, para isso segundo a documentação o AutoFeat utiliza cinco funções para identificar as features a remover, ou seja, aquelas que não trazem benefício ao modelo. Sendo assim as cinco funções são as seguintes: “identify_missing”, “identify_single_unique”, “identify_collinear”, “identify_zero_importance” e “identify_low_importance”. “identify_missing” é responsável por encontrar as features com valores em falta, por defeito o método considera que se uma feature tem mais de 60% de valores em falta é motivo para ser considerada má. “identify_single_unique” identifica as features que possuem só um único valor único. “identify_collinear” é responsável por identificar as features que estão correlacionadas através do coeficiente de correlação de Pearson e remove uma delas. “identify_zero_importance” identifica as features com zero importância através do algoritmo gradient boosting. Por último o “identify_low_importance” utiliza dos valores obtidos pelo “identify_zero_importance” de forma a encontrar as features de menor importância não necessários para atingir o valor do target. Por defeito o

AutoFeat considera que 99% é o valor necessário para que as features sejam necessárias para o resultado, então se uma feature não é usada em 99% dos resultados obtidos do target não é relevante. Aplicando estas cinco funções obtemos o resultado do AutoFeat na seleção de features.

2.3.2 TPOT

Figura 2 - Logotipo do TPOT

TPOT é o acrónimo de Tree-based Pipeline Optimization Tool e é uma framework open source desenvolvida em python (*TPOT*, 2022). Foi lançada em novembro de 2015 por um grupo de investigadores da Universidade da Pensilvânia (Olson et al., 2016). Como o próprio nome indica é uma framework baseada no conceito de árvore e pipeline onde o objetivo é a construção de pipelines ideais através do uso de programação genética (PG) (Le et al., 2020). As possíveis soluções são avaliadas segundo determinado critério e as características das soluções promissoras são misturadas de forma a criar soluções. O processo é repetido até que a condição de paragem seja atingida.

O TPOT quando foi projetado não teve em consideração o tempo que demora a ser executado (Le et al., 2020), pode demorar minutos como dias até terminar. Embora que em problemas menos complexos e com datasets pequenos seja possível obter bons resultados em poucos minutos. É possível ajustar diversos parâmetros para que o TPOT demore menos tempo a ser executado, reduzindo a qualidade da solução apresentada. Com os parâmetros por defeito do TPOT, a framework avalia mais de 10,000 pipelines, afina mais de 10,000 combinações de parâmetros para o modelo de ML, conjugado com CV para avaliação do modelo passamos rapidamente para 100,000 modelos que são afinados durante cada iteração.

O objetivo do TPOT não é ser utilizado para extração de features, mas pode ser utilizado para tal. A forma mais simples de obter um conjunto de features ideal segundo o TPOT é elaborar um pequeno script de forma a configurar o TPOT e deixar a framework avaliar todos os tipos de pipelines possíveis de forma automática, automatizando as partes de seleção, processamento e construção de features, além da seleção de modelo e otimização de parâmetros. No fim é mostrada a melhor

pipeline criada pelo TPOT, consoante o tipo de pipeline gerada é possível obter através dos coef_ ou feature_importance_ do modelo as melhores features. Também é possível obter as features criadas pelo TPOT se na pipeline existir algum passo de transformação de features, como por exemplo o “Polynomial Features” ou o “PCA”. Nestes casos basta listar o resultado da aplicação da fase da pipeline em questão para obter as features e os seus valores.

2.3.3 H2O AutoML

Figura 3 - Logotipo do H20 AutoML

H2O AutoML é uma framework de ML distribuída, open source e projetada para tratar datasets de grandes dimensões, dai ser uma framework de ML distribuída (*AutoML: Automatic Machine Learning — H2O 3.36.1.1 documentation*, 2022). Pode ser configurada para correr localmente numa máquina, perdendo o poder de processamento distribuído, mas funcionando exatamente da mesma forma, a única diferença é o tempo de execução da framework. Esta foi desenvolvida em JAVA mas pode ser utilizada por diversas linguagens como R, Python e Javascript por exemplo. O principal objetivo do H20 AutoML é ajudar o utilizador a criar o melhor modelo possível utilizando o menor nível de esforço possível, ou seja, utilizar o menor número de linhas de código possível (LeDell & Poirier, 2020). Todo o trabalho é realizado de forma automática pela ferramenta e resultado é um “leaderboard” com a lista dos modelos criados com a sua classificação.

Para trabalhar com esta ferramenta é necessário fornecer um dataset já com algum tratamento, por exemplo, sem a existência de valores em falta. No próximo passo são criados vários modelos baseados em Generalized Linear Models (GLM), Distributed Random Forests (DRF), XGBoost, Gradient Boosting Machines (GBM) e Deep Learning (NN). Depois de criados todos os modelos são afinados individualmente os seus parâmetros. Depois são treinados todos os modelos e no fim é retornado uma lista com os melhores modelos ou pipelines geradas pelas framework. Através dos modelos criados é possível obter a lista de features com maior importância para o modelo selecionado e desta forma obter a lista de features mais importantes.

2.3.4 FeatureTools

Figura 4 - Logotipo do FeatureTools

Featuretools é uma framework open source em python desenvolvida para a criação de features de forma automática (*Featuretools*, 2022). Esta biblioteca está assente em três pilares essenciais (Pelison, 2018):

- Entidades ou entities
- Deep Feature Synthesis (DFS)
- Feature primitives

Entidades são as tabelas que contêm os dados, se tivermos várias tabelas o conceito altera-se para entityset.

Deep Feature Synthesis ou DFS é o core desta framework, este algoritmo foi desenvolvido em 2014 por um grupo de investigadores do Laboratório de Ciência da Computação e Inteligência Artificial do MIT. Este algoritmo é o responsável pelo processo de criação de novas features.

Feature primitives são funções que o DFS usa na criação de novas features, estas podem ser classificadas em dois conjuntos:

- Funções de transformação (transformation primitives)
- Funções de agregação (aggregation primitives)

Funções de transformação são funções que transformam informações existentes em novos dados. Funções de agregação são funções que agregam as informações a partir de alguma chave.

O Featuretools permite realizar a seleção de features através da Deep Feature Synthesis com três funções:

- Remover features nulas (Remove highly null features)
- Remover features sem variação (Remove single value features)
- Remover features correlacionadas (Remove highly correlated features)

A função de remover features nulas analisa o dataset e se uma coluna/feature tiver muitos valores nulos ou vazios é descartada. Por defeito o valor para descarte é de 95%. A função de remover features sem variação analisa cada feature e verifica se existe alguma variação nos dados que apresenta. Se a variação for pouca essa feature é descartada. A função para remover features correlacionadas analisa o dataset e verifica quais as features provavelmente seriam redundantes para o modelo

considerando a correlação entre as features. Quando duas features são consideradas altamente correlacionadas é removida a que tiver maior correlação e que seja superior a 95%.

3 TILS - Tendency Iterated Local Search

Um dos objetivos deste projeto é desenvolver um método de engenharia de features baseado no “Trepa Colinas” (TC) e no “Iterated Local Search” (ILS) de forma a extrair os melhores atributos para serem usados em qualquer modelo. Uma das principais motivações deste algoritmo foi a necessidade de ajudar a descobrir os melhores biomarcadores da bioenergética mitocondrial para identificar doentes com ELA.

3.1 Pesquisa Local

Os algoritmos de pesquisa local (PL) baseados no trepa colinas (TC), são algoritmos iterativos que partem de uma solução inicial e a cada iteração a solução atual é substituída por uma da vizinhança que melhore a solução atual. O algoritmo termina quando na vizinhança não exista nenhuma solução que melhore a atual. Sendo assim, os algoritmos de PL baseiam-se em: representação, objetivo e a função de avaliação (Michalewicz & Fogel, 2013). A representação diz respeito ao espaço de procura onde estão disponíveis as possíveis soluções candidatas, o objetivo diz respeito ao objetivo que se quer ver cumprido e a função de avaliação remete para um valor específico que indica a qualidade de uma solução. Por vezes é útil olhar para um espaço de procura em termos geométricos, pensando na vizinhança que rodeia uma determinada solução e se podemos ou não encontrar melhores soluções dentro dessa vizinhança. Se não conseguirmos, então descobrimos um ótimo local. Pode ser que a solução seja realmente um ótimo global, isto é, independentemente do tamanho da vizinhança que escolheríamos, a solução continuaria a ser a melhor. É muitas vezes dispendioso em termos computacionais expandir a procura para além das vizinhanças mais próximas. Como resultado, enfrentamos o perigo de ficarmos presos num ótimo local, particularmente quando se usa o TC. Uma forma de evitar ficar preso num ótimo local é introduzir o conceito de perturbação, ou seja, fazer com que o algoritmo “salte” para uma zona diferente, evitando assim ficar preso num ótimo local.

O TC, tal como todos os métodos de PL utiliza uma técnica de melhoria iterativa. A técnica é aplicada a um único ponto (ponto atual) dentro do espaço de procura. A cada iteração, um novo ponto é selecionado a partir da vizinhança do ponto atual. Se esse novo ponto fornecer um valor melhor face à função de avaliação, o novo ponto torna-se o ponto atual. Caso contrário, é selecionado um ponto da vizinhança e testado em relação ao ponto atual. O algoritmo termina se não for possível melhorar mais ou se a condição de paragem for atingida (Michalewicz & Fogel, 2013).

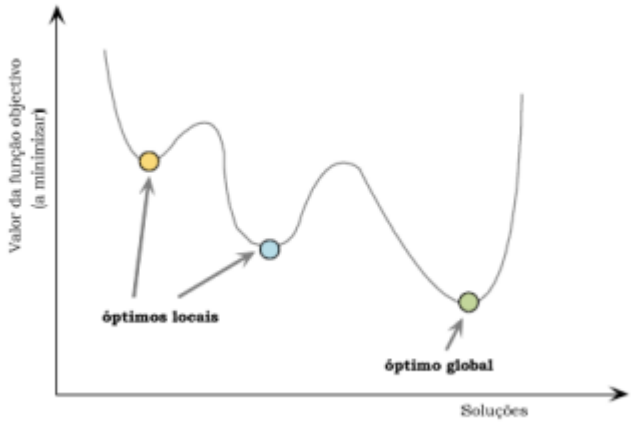

Figura 5 - Ótimos Locais e Ótimos Globais (Marques, 2015)

Uma das limitações dos algoritmos de PL são os ótimos locais, visto que podem existir melhores soluções (Hoos & Stützle, 2005), mas por outro lado, são algoritmos fáceis de aplicar. Tudo o que é necessário é a representação, uma função de avaliação e uma medida que defina a vizinhança em torno de uma dada solução (Paquete, 2005).

Os algoritmos de PL não garantem uma solução ótima para o problema, mas a resposta obtida no tempo de execução é satisfatória sem a necessidade de grandes recursos computacionais (Marques, 2015).

Neste projeto optamos por nos focar em dois algoritmos de pesquisa local, o tradicional “Trepa Colinas” (TC) e o “Iterated Local Search” (ILS), sendo estes a base do algoritmo a ser desenvolvido.

O TC examina os nós vizinhos um por um e seleciona o primeiro nó vizinho que melhora a solução atual, sendo assim o TC pode ser representado da seguinte forma:

Trepa Colinas

1. Definir o objetivo, gerar uma solução inicial e avaliar.
   1.1. **Se** a solução for o objetivo, parar e retornar a solução.
   1.2. **Senão**, a solução inicial passa a ser a solução atual
2. **Enquanto** o número máximo de iterações não for atingido **Repetir**:
   2.1. Gerar uma solução vizinha e verificar
      2.1.1.**Se** a solução vizinha for o objetivo **Então**
         2.1.1.1. retornar a solução vizinha e parar
      2.1.2.**Senão**, **Se** a solução vizinha for melhor que a solução atual **Então**
         2.1.2.1. a solução vizinha passa a ser a solução atual
      2.1.3.**Senão**, **Se** a solução vizinha não for melhor que a atual **Então**
         2.1.3.1. continuar iterativamente
3. Mostrar a melhor solução

Figura 6 - Algoritmo Trepa Colinas

3.2 Iterated Local Search - ILS

O ILS é uma evolução do TC que consiste em explorar as soluções possíveis através de perturbações quando se está perante ótimos locais. Esta perturbação precisa de ser suficientemente forte para poder sair de um ótimo local, mas também não pode ser extremamente forte para evitar um reinício aleatório (H. R. Lourenço et al., 2003; Toksari, 2016). O ILS está assente em quatro pilares essenciais (H. Lourenço et al., 2010):

- criação de uma solução inicial: O algoritmo determina aleatoriamente uma solução inicial dentro do universo existente.
- Método de pesquisa local: Processo iterativo de procura de uma solução melhor que a atual.
- Perturbação: Alteração da solução atual que acontece quando se está perante um ótimo local.
- Critério de Aceitação, diz respeito ao processo de seleção, avalia se a solução antes ou depois da perturbação é mais benéfica que a solução atual.

Sendo assim o ILS pode ser representado da seguinte forma:

ILS

1. $s_0 = GerarSoluçãoInicial$
2. $s^* = PesquisaLocal(s_0)$
3. **Repetir** até encontrar condição de paragem
   3.1. $s' = Perturbação(s^*)$
   3.2. $s^{*\prime} = PesquisaLocal(s')$
   3.3. $s^* = CriterioAceitação(s^*, s^{*\prime})$
4. Mostrar a melhor solução

Figura 7 - Algoritmo ILS

De forma a entender melhor o funcionamento do ILS optamos por colocar uma figura ilustrativa da aplicação do ILS.

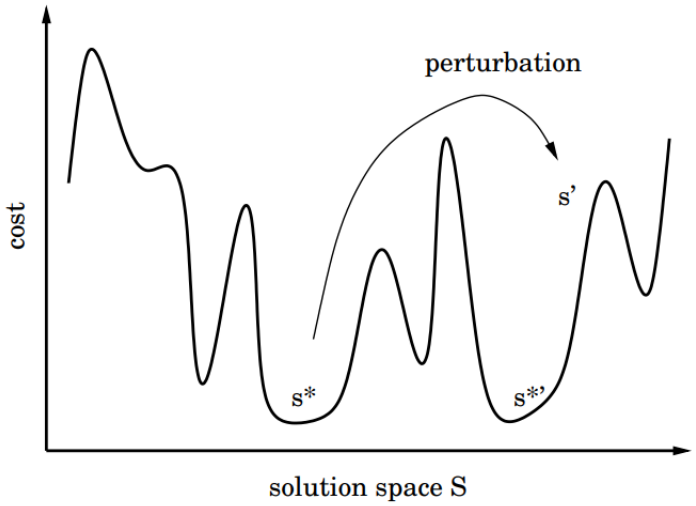

Figura 8 - Representação do ILS (H. Lourenço et al., 2010)

A Figura 8 é um exemplo da aplicação do ILS, depois de obter um mínimo local s^*, é aplicada a perturbação que dá origem a uma nova solução s'. Depois é aplicado a pesquisa local e obtém-se um novo mínimo local $s^{*\prime}$ que pode ser melhor ou não que o mínimo local anterior.

3.3 TILS

Depois de conhecer e perceber o funcionamento do TC e do ILS vamos proceder ao desenvolvimento do algoritmo proposto. Desta forma é necessário perceber o que é uma solução, como se avalia uma solução e quais são os operadores de transformação existentes. Uma solução é um conjunto de features inicial escolhido de

forma aleatória ou não, ou seja, uma solução inicial é a escolha aleatória de um determinado número de features ou uma determinada percentagem em relação ao dataset inicial. A função de avaliação será um valor indicativo da qualidade da solução em relação ao objetivo, neste caso será a accuracy do modelo criado com essas features através de uma árvore de decisão. As operações existentes estão divididas em duas categorias: transformação e seleção. Os operadores de transformação são os seguintes:

- Operações matemáticas com a utilização de duas features
    - Adição
    - Subtração
    - Multiplicação
    - Divisão
- Operações matemáticas com a utilização de uma feature
    - Raiz quadrada
    - Exponenciação

Os operadores de seleção são os seguintes:

- Operações não matemáticas
    - Adicionar feature nova
    - Apagar feature existente

O resultado das operações matemáticas com a utilização de duas features é uma nova feature com o resultado obtido entre as duas features. Exemplo: $F1 + F2 = F3$, onde $F3$é uma nova feature com o resultado da soma entre $F1$ e $F2$.

As operações matemáticas referentes apenas a uma feature é a aplicação direta de uma das duas operações matemáticas onde o resultado é uma nova feature. Exemplo:$\sqrt{F4} = F5$, $F5$ é uma nova feature com o resultado da operação raiz quadrada.

Os operadores não matemáticos são responsáveis por adicionar ou remover uma feature, se a operação for adicionar é adicionada uma nova feature do dataset ao subconjunto, no caso de ser a operação de apagar, é eliminada uma feature do subconjunto. No caso da operação selecionada for adição de uma nova feature, é escolhida uma feature que não pertença ao subconjunto do dataset e se disponível. Pode acontecer não existir mais features para adicionar e a operação escolhida ser essa mesmo, nestes casos o método está protegido para não escolher uma operação

de acrescentar uma nova feature quando não existem mais features disponíveis. Nestes casos em vez de ter 8 operações para escolha só tem 7.

Em todas as categorias de transformação a operação de escolha é feita de forma aleatória bem como a escolha da feature ou features a serem utilizadas. Esta escolha aleatória torna o algoritmo estocástico. Estes tipos de algoritmos são muito comuns porque a certo ponto utilizam a aleatoriedade para tomar decisões. Alem da utilização da aleatoriedade na escolha da operação, este fator também é utilizado no processo de seleção do tamanho inicial do dataset a ser usado no método, chamado de subconjunto.

Antes de iniciarmos o processo de desenvolvimento do método optamos por usar uma árvore de decisão como algoritmo para criação do modelo e usarmos a accuracy como medida de avaliação do modelo. Optamos por utilizar uma árvore de decisão para porque apresenta bons resultados em problemas de classificação e tem sido aplicada em várias áreas da saúde com bons resultados (Mitchell, 1997; Podgorelec et al., 2002).

3.3.1 TC

Numa fase inicial optamos por desenvolver um método baseado em pesquisa local, neste caso o TC, de forma a perceber a sua eficácia, rapidamente percebemos que era uma técnica robusta e então optamos por introduzir algumas alterações e melhorias de forma a conseguir a primeira abordagem do método proposto. Esta abordagem começa por escolher de forma aleatória um subconjunto de features através do dataset inicial e avalia a sua accuracy. Iterativamente escolhe aleatoriamente duas features do subconjunto e realiza uma das operações possíveis. No caso da operação selecionada for adicionar uma nova feature, esta é selecionada através das features disponíveis do dataset. Depois o novo subconjunto é novamente avaliado através da accuracy, se o novo subconjunto for melhor que o anterior então o novo passa a ser o subconjunto principal, caso contrário reverte-se a operação e volta-se ao início. Esta abordagem só termina quando atingir determinado número de iterações, no fim devolve o melhor subconjunto que encontrar.

Desta forma começamos por desenvolver a primeira abordagem do nosso método.

1ª Abordagem – Baseado no Trepa Colinas

1. Escolher aleatoriamente um determinado número de features do dataset
2. Avaliar o dataset através da accuracy da árvore de decisão e guardar como melhor solução
3. **Enquanto** o critério de fim não for atingido **Repetir**:
   3.1. Escolher aleatoriamente duas features das features disponíveis do subconjunto
   3.2. Realizar uma das 8 operações (+, -, *, /, sqrt, exp, add, del) aleatoriamente
   3.3. Avaliar através da accuracy da árvore de decisão
   3.4. **Se** a solução atual for melhor que a anterior **então**
      3.4.1.1. Melhor solução = Solução atual
   3.5. **Senão** reverte operação e volta ao passo 3.1
4. Mostrar a melhor solução

Figura 9 - 1ª Abordagem do método baseado no Trepa Colinas

A 1ª abordagem está assente em dois parâmetros fundamentais: “Número de features Iniciais” e “Critério de terminação”. O “Número de features iniciais” diz respeito ao subconjunto de features inicial a ser utlizado pelo método, nesta fase optamos por utilizar valores fixos, como por exemplo 20 features aleatórias. O “Critério de fim” também foi fixado no número máximo de iterações, por exemplo 100 iterações. A Tabela 1 mostra os parâmetros desta abordagem.

Tabela 1 - Parâmetros da 1ª abordagem

Parâmetros
“Número de Features Iniciais”
“Critério de Terminação”

Por norma antes de treinar qualquer modelo é separado aleatoriamente o dataset para obter dois subconjuntos, um para treino e outro para teste. Normalmente os valores usados são de 80% para treino e 20% para teste (Rácz et al., 2021). Depois do modelo treinado usamos o conjunto de teste para avaliar o desempenho do modelo. Não deixa de ser uma boa abordagem, mas pode não ser suficiente para avaliar o modelo e a utilização do cross validation é uma mais-valia. Como tal optamos por usar uma variante do CV mais exaustiva, o LOOCV.

3.3.2 TC com fator de tendência

Depois de explicada a primeira abordagem do método decidimos fazer uma pequena alteração no método e criar uma segunda abordagem para tentar perceber se o seu desempenho melhorava ou piorava em relação à primeira abordagem. A principal diferença entre a primeira abordagem e a segunda é a introdução do fator de

tendência. O fator de tendência é um parâmetro que é ativado ou não consoante o resultado da última operação seja benéfico ou não, ou seja, se a última operação foi benéfica para o problema então na próxima iteração volta a ser usada, caso não seja benéfica é escolhida aleatoriamente.

A Figura 10 mostra a 2ª abordagem do algoritmo.

2ª Abordagem – Baseado no Trepa Colinas com fator de tendência

1. Escolher aleatoriamente um determinado número de features do dataset
2. Avaliar o dataset através da accuracy da árvore de decisão e guardar como melhor solução
3. **Enquanto** o critério de fim não for atingido **Repetir**:
   3.1. Escolher aleatoriamente duas features das features disponíveis do subconjunto
   3.2. **Se existir** última operação com melhorias **Então**
      3.2.1.Realizar última operação com melhorias
   3.3. **Senão** Realizar uma das 8 operações (+, -, *, /, sqrt, exp, add, del) aleatoriamente
   3.4. Avaliar através da accuracy da árvore de decisão
   3.5. **Se** a solução atual for melhor que a solução anterior **então**
      3.5.1.Melhor solução = Solução atual
      3.5.2.última operação com melhorias = operação escolhida aleatoriamente
   3.6. **Senão** reverte operação, última operação com melhorias = null e volta ao passo 3.1
4. Mostrar a melhor solução

Figura 10 - 2ª Abordagem do método baseado no Trepa Colinas com fator de tendência

Acreditamos que a utilização do fator de tendência na escolha da operação possa ser uma mais-valia no resultado do método, se uma operação traz benefícios ao subconjunto porque não utilizar a mesma operação na próxima iteração? A utilização deste fator de tendência é analisada no Capítulo 4. A Tabela 2 mostra os parâmetros desta abordagem.

Tabela 2 - Parâmetros da 2ª abordagem

Parâmetros
“Número de Features Iniciais”
“Critério de Terminação”
"Fator de Tendência"

3.3.3 TILS

Depois de desenvolvida a segunda abordagem do nosso método optamos por implementar o conceito de perturbação do ILS, resultando na terceira abordagem.

A principal diferença entre a segunda e a terceira abordagem do nosso método foi a implementação do fator de perturbação (H. R. Lourenço et al., 2003). Esta otimização do método necessita de um “trigger” que seja disparado para que possa ser aplicado, esse “trigger” foi chamado de “Número de Iterações para Perturbar”. Este parâmetro é a responsável por ativar ou não a aplicação do conceito de perturbação. Este conceito de perturbação consiste em detetar se o método está parado num ótimo local, não conseguindo obter melhores resultados. Caso esteja parado o “trigger” é disparado e o método comporta-se de forma diferente. Neste comportamento é necessário também usar outro parâmetro que é o “Número de Operações a Adicionar”. Este parâmetro diz respeito ao número de operações que vamos inferir/introduzir na solução de forma a perturbar a sua performance. Este parâmetro é fundamental para que o algoritmo consiga obter melhores resultados, pois se a perturbação for muito pequena podemos ficar parados no mesmo local e se a perturbação for muito grande podemos saltar para uma zona desconhecida onde os resultados podem ser melhores ou piores de onde estávamos inicialmente. Dai a necessidade de este fator ser fundamental para a eficácia do algoritmo. O “Número de Operações a Adicionar” ideal para o algoritmo e o “Número de Iterações para Perturbar” serão analisados no Capítulo 4.

A Figura 11 mostra o esquema da terceira abordagem, chamada de TILS - Tendency Iterated Local Search.

TILS – Tendency Iterated Local Search

1. Escolher aleatoriamente um determinado número de features do dataset
2. Avaliar o dataset através da accuracy da árvore de decisão e guardar como melhor solução
3. **Enquanto** o critério de fim não for atingido **Repetir**:
   3.1. **Se** número de interações sem melhorias >= número de iterações para perturbar **Então**
      3.1.1.**Se** existir copia do dataset antigo **Então**
         3.1.1.1. Avaliar através da accuracy da árvore de decisão ambos os datasets
         3.1.1.2. **Se** o dataset antigo for melhor que o atual **então**
            3.1.1.2.1. Guardar dataset antigo como o atual
            3.1.1.2.2. Melhor solução = solução do dataset antigo
         3.1.1.3. **Senão** eliminar dataset antigo, Melhor solução = solução do dataset atual
      3.1.2.**Fim se**
      3.1.3. Guardar copia do dataset atual como antigo
      3.1.4.**Enquanto** número de operações a adicionar não for atingido **Repetir**
         3.1.4.1. Escolher aleatoriamente duas features das features disponíveis
         3.1.4.2. Realizar uma das 8 operações (+, -, *, /, sqrt, exp, add, del) aleatoriamente
      3.1.5.número de interações sem melhorias = 0
   3.2. **Senão**
      3.2.1.Escolher aleatoriamente duas features das features disponíveis
      3.2.2.**Se existir** última operação com melhorias **Então**
         3.2.2.1. Realizar última operação com melhorias
      3.2.3.**Senão** Realizar uma das 8 operações (+, -, *, /, sqrt, exp, add, del) aleatoriamente
      3.2.4.Avaliar através da accuracy da árvore de decisão
      3.2.5.**Se** a solução atual for melhor que a solução anterior **então**
         3.2.5.1. Melhor solução = Solução atual
         3.2.5.2. última operação com melhorias = operação escolhida aleatoriamente
         3.2.5.3. número de interações sem melhorias = 0
      3.2.6.**Senão** reverte operação, última operação com melhorias = null, incrementa número de interações sem melhorias e volta ao passo 3
4. Mostrar a melhor solução

Figura 11 - TILS

Acreditamos que a método desenvolvido é bastante poderoso e com a correta afinação de todos os parâmetros vai ser possível obter bons resultados. A Tabela 3 mostra os parâmetros do TILS

Tabela 3 - Parâmetros do TILS

Parâmetros
“Número de Features Iniciais”
“Critério de Terminação”
"Fator de Tendência"
“Número de Iterações para Perturbar”
“Número de Operações a Adicionar”

No “Anexo A – Tendency Iterated Local Search – TILS” encontra-se o código Python do algoritmo desenvolvido.

4 Engenharia de features no dataset preliminar - MicroRNA

4.1 Introdução

O objetivo principal deste capítulo é a aplicação do TILS ao dataset preliminar, em seguida a afinação dos seus parâmetros e por último a comparação com os métodos escolhidos no Capítulo 2. Desta forma optamos por utilizar um dataset preliminar como forma de teste a análise. Em primeiro lugar iremos fazer uma breve descrição e análise do dataset preliminar. Em segundo lugar vamos explicar o design experimental. Em terceiro, iremos aplicar os métodos escolhidos a este dataset e verificar os seus resultados. Em quarto, iremos fazer algumas afinações do modelo para que se possa verificar se existem melhorias ou não com a otimização de parâmetros. Em quinto lugar iremos testar e verificar quais os melhores valores para os diversos parâmetros do nosso método, de forma, a optar pela melhor configuração na obtenção dos resultados. Por último, iremos aplicar a TILS ao dataset preliminar, com as afinações e otimizações consideradas benéficas nos pontos anteriores.

4.2 Dataset MicroRNA

O dataset preliminar obtido em (Witten et al., 2010) é baseado em amostras reais de pacientes com e sem tumores onde o objetivo foi estudar a relação entre os níveis de MicroRNA e desenvolvimento de tumores (Witten et al., 2010). Os MicroRNA's (miRNA) são pequenas biomoléculas presentes nas células dos seres vivos e que participam dos processos bioquímicos dos organismos («MicroRNA», 2022). Sabe-se, por exemplo, que o genoma humano codifica mais de 1000 miRNAs. Estas pequenas moléculas produzem efeitos ao nível do desenvolvimento e dos mecanismos de defesa biológicos. A desregulação dos níveis de miRNA está muitas vezes associada a diferentes doenças. Este dataset diz respeito a um estudo sobre a descoberta de MicroRNA baseado em sequenciamento de biomarcadores referentes a uma coleção de tumores cervicais e respetivas amostras de controlo (Witten et al., 2010). Neste caso pretendeu-se estudar a eventual relação entre os níveis de expressão de miRNA e o surgimento de tumores.

O dataset consiste em 58 amostras de RNA: 29 de pacientes com tumores e 29 amostras de indivíduos de controlo. Para cada uma das amostras foi recolhida uma gama exaustiva de valores obtidos por sequenciação de miRNA. No total foram recolhidos 714 valores, 626 correspondentes aos níveis de expressão moléculas de miRNA já conhecidas e referidas em bibliotecas de dados biológicos e 88 novos candidatos a miRNA.

De forma a fazer um breve análise sobre o dataset usamos a ferramenta Orange Data Mining Toolbox (Ljubljana, 2021) para nos ajudar a compreender melhor o mesmo.

Começamos por observar que o dataset continha 58 linhas e 715 colunas. Destas 715 colunas uma era refente ao target, sendo este target alfanumérico e com duas classes: N e T, onde T significa presença de tumor e N ausência de tumor. Verificamos que o dataset não contém valores em falta e não procuramos valores que pudessem ser consideramos como outliers, já que mesmo que existissem não seriam removidos porque se trata de um dataset biológico e os valores potencialmente outliers podem ser um fator decisivo na boa classificação do target. Verificamos também que todos os valores das features são numéricos, inteiros e sem nenhuma casa decimal.

4.3 Design experimental

O primeiro passo foi instalar o software Anaconda (*Anaconda*, 2021). Este software é o responsável por instalar todos os componentes necessários para trabalhar com Python (*Python.Org*, 2021) de uma forma rápida e simples, disponibilizando diversas ferramentas para que os utilizadores consigam trabalhar em Python.

O segundo passo foi observar o dataset, perceber o seu conteúdo, compreender os valores, identificar o target e as features, e verificar se existiam valores em falta (o dataset não continha valores em falta). Optamos por colocar o dataset numa escala de [-1,1] através da função MinMaxScaler (*Sklearn.Preprocessing.MinMaxScaler*, 2021). Com esta modificação obtemos os mesma relação entre as features, mas numa escala menor, permitindo maior performance nos algoritmos e mantendo a coerência dos dados iniciais.

O passo seguinte foi desenvolver um script para cada método de forma a aplicar o mesmo ao dataset e recolher os resultados. Desta forma desenvolvemos 9 scripts em Python para aplicar cada método ao dataset:

- Começamos pelo qui-quadrado onde utilizamos a função Chi2 (*Sklearn.Feature_selection.Chi2*, 2022). Segundo a documentação do sklearn o score obtido ordenado por ordem decrescente, indica quais os atributos/features mais relevantes em relação à classe.
- Para aplicar a eliminação recursiva de features ao dataset foi utilizada a biblioteca scikit-learn para Python (*Sklearn.Feature_selection.RFE*, 2022). Segundo a documentação do sklearn é obrigatório usar um “estimator”, que no nosso caso optamos por usar uma árvore de decisão e como segundo parâmetro utilizamos o “n_features_to_select” para que o algoritmo retornasse um número determinado de features, neste caso optamos por escolher o valor de 40.
- Para aplicar a regressão de ridge foi utilizada a biblioteca scikit-learn para Python (*Sklearn.Linear_model.Ridge*, 2022). Segundo a documentação do sklearn para utilizar esta regressão é necessário fornecer um valor para “alpha”, optamos por utilizar os valores por defeito que é 1.

- Para aplicar a regressão de lasso utilizamos a biblioteca scikit-learn para Python (*Sklearn.Linear_model.Lasso*, 2022). Segundo a documentação do sklearn para utilizar esta regressão é necessário fornecer um valor para “alpha”, optamos por utilizar o valor por defeito que é 1.
- Para aplicar o ganho de informação utilizamos a biblioteca scikit-learn para Python (*Sklearn.Tree.DecisionTreeClassifier*, 2022). Segundo a documentação do sklearn para utilizar o ganho de informação temos de utilizar a class DecisionTreeClassifier. Depois é necessário utilizar o parâmetro criterion = “entropy”. Outro parâmetro que utilizamos foi o max_features = “auto”, que segundo a documentação o número máximo de features retornadas é igual ao valor da raiz quadrada do número total de features.
- Para aplicar o Autofeat foi necessário instalar os componentes da biblioteca para Python (Horn et al., 2019). Depois utilizamos o método AutoFeatClassifier com os parâmetros por defeito. Para identificar as features criadas pela framework utilizamos o “new_feat_cols_”.
- Para utilizar o TPOT foi necessário instalar os componentes da biblioteca para Python (*TPOT*, 2022). Depois criamos um script para configurar o TPOT para problemas de classificação com o método TPOTClassifier, utilizamos os valores por defeito. Depois de executado este script o resultado é uma pipeline criada pelo TPOT de forma a obter a melhor performance possível. Com a pipeline escolhida voltamos a criar um script para determinais quais as features utilizadas pelo TPOT. Caso na pipeline criada exista algum passo para criação ou transformação de features como por exemplo “Polynomial Features”, “PCA” ou “RFE” é possível obter a listagem de features através das propriedades das mesmas. Caso contrário é porque a framework utilizou todas as features, desta forma podemos determinar quais são as mais importantes para o modelo através dos coef_ ou feature_importance_.
- Para utilizar o H2O foi necessário instalar os componentes da framework para Python (*AutoML: Automatic Machine Learning — H2O 3.36.1.1 documentation*, 2022). Depois criamos um pequeno script de forma a configurar o servidor local do H2O com apenas dois parâmetros: número de threads e memória máxima a ser usada, neste caso optamos por utilizar o valor de -1 para o número de threads, que segundo a documentação indica que o H2O utiliza todos os CPU’s disponíveis. Definimos também um limite máximo de memória para 12GB de forma a ter a maior quantidade de memoria disponível a ser utilizada e porque a máquina usada só tinha 16GB de memória disponível. Depois de configurado o servidor para o H2O, configuramos o H2O para o nosso dataset com os valores por defeito. Depois de executado este script o resultado obtido é um leaderboard com os melhores modelos que o H2O conseguiu criar. Desse quadro escolhemos o primeiro registo, e através dos coef_ ou feature_importance_ do modelo retiramos as melhores features.

- Para utilizar o FeatureTools foi necessário instalar os componentes da framework para Python (*Featuretools*, 2022). Criamos um pequeno script de forma a aplicar o "Deep Feature Synthesis" no dataset. Depois com o conjunto de dados retornado pelo DFS começamos por remover as features nulas, depois remover as features sem variação e por fim remover as features correlacionas. Para fazer estas operações foi só aplicar os métodos diretamente sem a necessidade de afinação de nenhum parâmetro. No fim simplesmente mostramos as features restantes que são as que o featuretools considerou como melhores.

O passo seguinte foi criar um script para criar um modelo de ML através de uma árvore de decisão ao resultado obtido da aplicação de cada método. Este script aplica uma árvore de decisão com os valores por defeito da biblioteca DecisionTreeClassifier (*Sklearn.Tree.DecisionTreeClassifier*, 2022) e depois foi usado o cross_val_predict (*Sklearn.Model_selection.Cross_val_predict*, 2022) para obter os resultados da árvore de decisão. Optamos por usar LOOCV na função cross_val_predict de forma a obter um resultado mais fiável. Este processo foi usado repetidamente para cada método e os resultados registados em tabelas que serão analisadas nos próximos capítulos.

De forma a obter algumas métricas optamos por recolher a informação sobre o tempo de execução de cada método em cada script, o tempo de execução do modelo, nº de features e por fim o seu desempenho. O tempo de execução é importante para termos um termo de comparação entre as várias abordagens e perceber se o tempo é um fator de decisão na escolha método. Outra métrica é a qualidade da solução, optamos por usar a performance de cada um, para obtermos essa métrica usamos a accuracy do modelo. Estes valores foram retirados da aplicação direta da função de avaliação cross_val_predict (*Sklearn.Model_selection.Cross_val_predict*, 2022) da biblioteca sklearn.

No processo de afinação do modelo optamos por criar um script onde aplicamos o GridSearchCV (*Sklearn.Model_selection.GridSearchCV*, 2022) de forma a afinar os parâmetros "max_depth" e "min_samples_split" da árvore de decisão, o mesmo script serve para testar os valores obtidos e aplicar na árvore de decisão.

No "Anexo B – Ficheiros relativos ao dataset MicroRNA" encontra-se os ficheiros Python referentes ao design experimental.

4.4 Resultados da aplicação dos métodos ao dataset MicroRNA

Tabela 4 - Resultados da aplicação dos métodos ao dataset MicroRNA

Método	Nº Features Originais	Nº Features Novas	Δ t (s) Seleção de features	Δ t (s) LOOCV	Accuracy
-	714	0	-	1,53	0,86
Qui-Quadrado	40	0	1,31	0,93	0,71
Eliminação Recursiva de Features	40	0	28,65	1,06	0,72
Regressão de Ridge	174	0	1,07	1,17	0,91
Regressão de Lasso	61	0	3,90	1,09	0,62
Ganho de Informação	9	0	0,76	0,92	0,67
Autofeat	714	1	45,12	1,55	0,86
TPOT	360	0	33470,93	1,33	0,87
H20 AutoML	25	0	3577,87	0,99	0,93
Featuretools	714	48363	8424,12	26,22	0,54

A Tabela 4 é composta por 6 colunas, a primeira coluna diz respeito ao método utilizado. A segunda coluna diz respeito ao número de features originais resultantes da aplicação do método. A terceira coluna indica o número de features criadas pelo método. Na quarta coluna temos o tempo em segundos que o método demorou a ser executado. Na quinta coluna temos o tempo de execução da avaliação do modelo. Por último, na sexta coluna temos a accuracy geral do método.

Depois de percebida a estrutura da tabela, vamos olhar agora para os seus valores e pela análise da Tabela 4 podemos rapidamente observar que a quantidade de features utilizadas por cada método é bastante diferente, não existindo nenhum tipo de padrão ou tendência. Temos métodos a utilizar menos de 10 features e outros a utilizar mais de 1000. Portanto estamos perante uma realidade muito díspar de resultados. No que toca a tempos de execução também temos várias realidades, uma das realidades é que temos métodos a demorar aproximadamente 1 segundo e outros a demorar mais de 8 horas. No que toca a tempos de execução da avaliação do modelo não temos grandes diferenças nos resultados, quase todos rondam 1 segundo. Agora no que toca a valores gerais de performance já temos valores bem diferentes. A framework que obteve a melhor classificação foi o H20 AutoML com 93% de accuracy. O método tradicional que obteve melhor classificação foi a regressão de ridge com 91% de acurracy.

Observando estes resultados é fácil perceber que o H20 AutoML obteve os melhores resultados, mas também foi dos que demorou mais tempo, apesar de não ser o método que demorou mais tempo, foi o segundo que mais tempo demorou a ser executado. Logo de seguida temos a regressão de ridge que demorou em relação ao H20 AutoML menos 300% e obteve quase a mesma pontuação. Outra análise que

podemos fazer desta tabela é que a quantidade de features não é sinal de qualidade, vejamos o seguinte caso, o Featuretools selecionou 49077 features e obteve uma accuracy de 54%, já o Autofeat utilizou 715 features o obteve uma accuracy de 86%, portanto a diferença de features não se refletiu em melhor performance.

Outra observação é que a regressão de ridge foi o método tradicional que obteve melhores resultados, indicando que é um método que traz benefícios em datasets com um número elevado de features e poucas amostras. Ao contrário dos restantes métodos que revelaram não ser benéficos neste tipo de dataset. Das frameworks a que se destacou foi a H2O AutoML que com apenas 25 features conseguiu obter um bom resultado comparado com as restantes frameworks. O Featuretools foi o método que criou mais features mas esse aumento não se refletiu na sua performance.

4.5 Afinação de parâmetros do modelo de ML

Para criação do modelo de ML optamos por usar o algoritmo da árvore de decisão sem qualquer afinação de parâmetros. Nesta secção vamos afinar alguns parâmetros da árvore de decisão de forma a verificar se traz benefícios ou não. Sendo assim, optamos por escolher os dois piores e os dois melhores conjuntos de features obtidos pelos métodos para afinar os parâmetros da árvore de decisão. Optamos por escolher os piores resultados e os melhores para termos duas realidades distintas, assim conseguimos verificar se afinação é benéfica num resultado que era mau e se ajuda a obter melhores resultados num resultado que já era considerado bom.

Os parâmetros que vamos afinar são o “min_samples_split” e “max_depth”. O “min_samples_split” é um parâmetro da árvore de decisão que determina o número mínimo de amostras para separação, ou seja, supondo que existem 7 amostras e temos min_samples_split = 5 a divisão é possível, podendo um ramo da árvore por exemplo ficar com 6 folhas e a outra com 1 folha. O parâmetro “max_depth” define o número máximo de profundidade da árvore. O parâmetro de profundidade é uma das formas de afinar a árvore ou então limitar a maneira como ela cresce para evitar o overfitting. De forma a poder testar quais os melhores valores para estas duas variáveis, usamos um método em Python chamado “GridSearchCV”. Este método faz diversas combinações dos parâmetros e depois avalia, no fim retorna o ou os melhores valores possíveis. Sendo assim aplicamos este método ao nosso dataset MicroRNA e tentamos perceber quais eram os melhores valores para “min_samples_split” e “max_depth”. De forma a tornar o processo mais rápido optamos por definir os valores distintos para os parâmetros, sendo eles [5,10,20,40,50,100] para “max_depth” e [2,3,4,5] para “min_samples_split”. A afinação destes dois parâmetros na maioria dos casos é suficiente para obter melhores resultados (Mantovani et al., 2018). Os resultados foram transcritos para a Tabela 5. No “Anexo C – Afinação de parâmetros do modelo” encontra-se o código python usado para afinar o modelo.

Tabela 5 - Aplicação do GridSearchCV para afinar árvore de decisão

Método	Nº Features	Δ t (s) LOOCV	min samples split	max depth	Accuracy
Featuretools	49077	10,54	3	20	0,60
Regressão de Lasso	61	0,26	3	20	0,71
H20 AutoML	25	0,20	3	20	0,95
Regressão de Ridge	174	0,43	5	10	0,93

A Tabela 5 é composta por 6 colunas, a primeira coluna diz respeito ao método utilizado. A segunda coluna diz respeito ao número de features resultantes da aplicação do método. Na terceira coluna temos o tempo em segundos da avaliação do modelo. Na quarta coluna temos o valor obtido para “min_samples_split”. Na quinta coluna temos o resultado para o “max_depth”. Por último, na sexta coluna temos a accuracy geral do método depois de afinada a árvore de decisão com os valores sugeridos.

Pela análise da Tabela 5 podemos reparar que os tempos de execução da árvore de decisão são bastante inferiores aos tempos da Tabela 4 para os mesmos métodos. Esta melhoria deve-se ao facto de restringirmos a profundidade máxima da árvore evitando problemas de overfitting e underfitting. Outro pormenor que podemos verificar é que os valores obtidos para os dois parâmetros não são muito dispares entre si, ou seja, para o parâmetro “min_samples_split” temos 2 opções [3,5] e para o parâmetro “max_depth” temos [10,20]. Os melhores valores para o “min_samples_split” variam entre 3 e 5, sendo que o valor 3 é o mais utilizado, logo será o valor que vamos usar para este parâmetro. No que toca ao parâmetro “max_depth” temos 2 valores como ótimos, sendo o valor 20 escolhido em maioria. Ao analisarmos a accuray global podemos reparar que em comparação com a Tabela 4 todos os valores são superiores, por exemplo o valor obtido para a accuracy na regressão de lasso na Tabela 4 foi de 62% e nesta tabela foi de 71%, obtivemos uma melhoria de 9%. No Featuretools tivemos um ganho de 6%, no H20 AutoML tivemos uma melhoria de 2% e na Regressão de Ridge uma melhoria de 2%. Portando, a utilização do “GridSearchCV” na otimização dos 2 parâmetros escolhidos foi benéfica em todos os aspetos, nos métodos com menores performances o ganho foi mais acentuado do que nos métodos que já possuíam uma boa performance, mas mesmo assim com a afinação foi possível obter uma pontuação ainda maior.

Pela análise dos resultados acima podemos concluir que a afinação de parâmetros da árvore de decisão é uma mais-valia. A afinação destes parâmetros depende sempre do tipo de problema, neste dataset MicroRNA e perante os moldes utilizados os valores obtidos tendem a melhorar a performance deste modelo, deste modo optamos por usar estas afinações no nosso projeto, sendo assim optamos por usar os valores

de “max_depth” = 20 e “min_samples_split” = 3 quando utilizarmos uma árvore de decisão.

4.6 Afinação de parâmetros do TILS

O TILS possui 5 parâmetros para o seu funcionamento, a listagem de parâmetros já foi descrita no Capítulo 3, sendo assim a Tabela 6 ilustra a relembra de parâmetros do TILS.

Tabela 6 - Parâmetros do TILS

Parâmetro	Valor
“Número de Features Iniciais”	
“Critério de Terminação”	1000
"Fator de Tendência"	
“Número de Iterações para Perturbar”	
“Número de Operações a Adicionar”	

De forma a saber quais os melhores valores para cada parâmetro do TILS, começamos por afinar os valores referentes à 2ª abordagem: “Número de Features Iniciais”, “Critério de Terminação” e “Fator de Tendência”. Para o “Critério de Terminação” optamos por usar sempre o valor de 1000 porque achamos que é um número razoável de iterações para obter um resultado, nada impede em outras utilizações a alteração deste valor, temos de ter em conta, que quanto maior for o valor, maior é o tempo de execução. Para encontrar os melhores valores para “Número de Features Iniciais” e “Fator de Tendência” começamos por fazer uma análise comparativa entre a percentagem inicial de features e considerar a tendência ou não do nosso método. Para tal corremos o algoritmo na 2ª abordagem 30 vezes, por cada vez registamos os resultados numa tabela. O Gráfico 1 é o resultado da média deste processo.

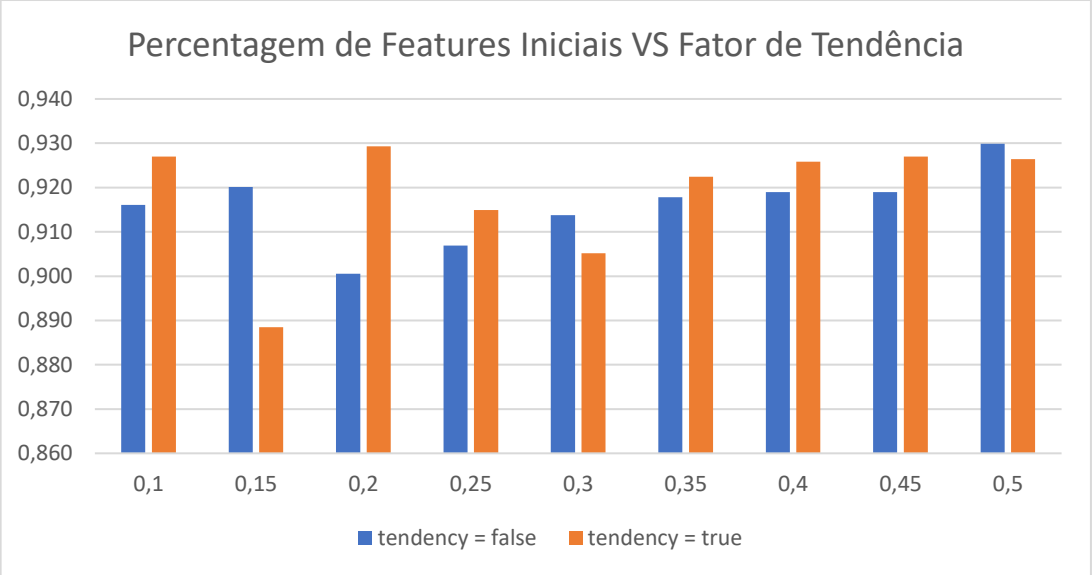

Gráfico 1 - Percentagem de Features Iniciais VS Fator de Tendência

No eixo dos X temos os valores referentes à percentagem inicial do dataset a ser usado. No eixo dos Y temos os valores obtidos pela accuracy do modelo. As barras com a cor azul dizem respeito aos valores obtidos pelo algoritmo quando o fator de tendência está desativado e as colunas laranja dizem respeito aos valores obtidos com a o fator de tendência ativo. Pela análise do gráfico, maior parte das vezes a utilização do fator de tendência trouxe melhorias. Em 9 casos, 6 são benéficos e os restantes prejudiciais, como existem mais casos com vantagens do que casos com desvantagens optamos por usar o fator de tendência. No que toca ao parâmetro "Número de features Iniciais", referente à percentagem inicial de features optamos por utilizar o valor de 0.2 porque foi o melhor resultado obtido quando utilizamos o fator de tendência. A Tabela 7 mostra os valores escolhidos para os parâmetros da 2ª abordagem.

Tabela 7 - Parâmetros da 2º abordagem afinados

Parâmetro	Valor
"Número de Features Iniciais"	0,2 (20%)
"Critério de Terminação"	1000
"Fator de Tendência"	1

Afinados os parâmetros referentes à 2ª abordagem ficam a faltar os parâmetros da versão final do algoritmo – TILS. O TILS possui mais 2 parâmetros que necessitam de ser afinados, são eles o "Número de Iterações para Perturbar" e o "Número de Operações a Adicionar". De forma a perceber quais seriam os melhores valores a utilizar corremos o TILS com os parâmetros afinados da 2ª abordagem 30 vezes e registamos os valores numa tabela. De forma a restringir a gama de valores para cada parâmetro, optamos por testar cada um dos parâmetros num determinado intervalo

de valores. Sendo eles [50, 100, 150, 200] para o “Número de iterações para perturbar” e [1, 3, 5, 10] para “Número de operações a adicionar” depois de um determinado número de iterações sem melhorias. Optamos por começar a nossa análise por descobrir quais os melhores valores para o número de iterações sem melhorias. O Gráfico 2 reflete as médias dos resultados obtidos.

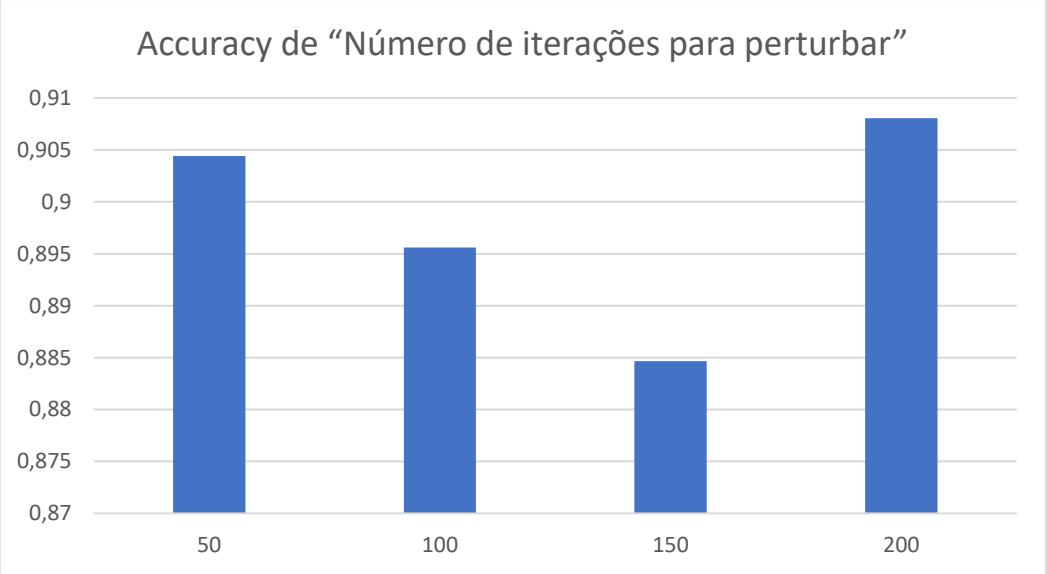

Gráfico 2 - Accuracy de “Número de iterações para perturbar”

No eixo horizontal temos o número de iterações sem melhorias de performance para que o método opte por utilizar a perturbação. No eixo vertical temos os valores da accuracy para cada um dos casos. Podemos concluir que a utilização do valor de 50 e 200 são os mais benéficos, quando “Número de Iterações para Perturbar” é igual a 50 o valor da accuracy é de 0.904 e quando é igual a 200 o valor da accuracy é de 0.908. Sendo o valor de 200 o mais benéfico, para os nossos testes e análise consideramos o valor de 50 pelo simples facto de o método com um valor mais baixo, poder aplicar mais vezes a perturbação e desta forma tentar obter ainda melhores resultados.

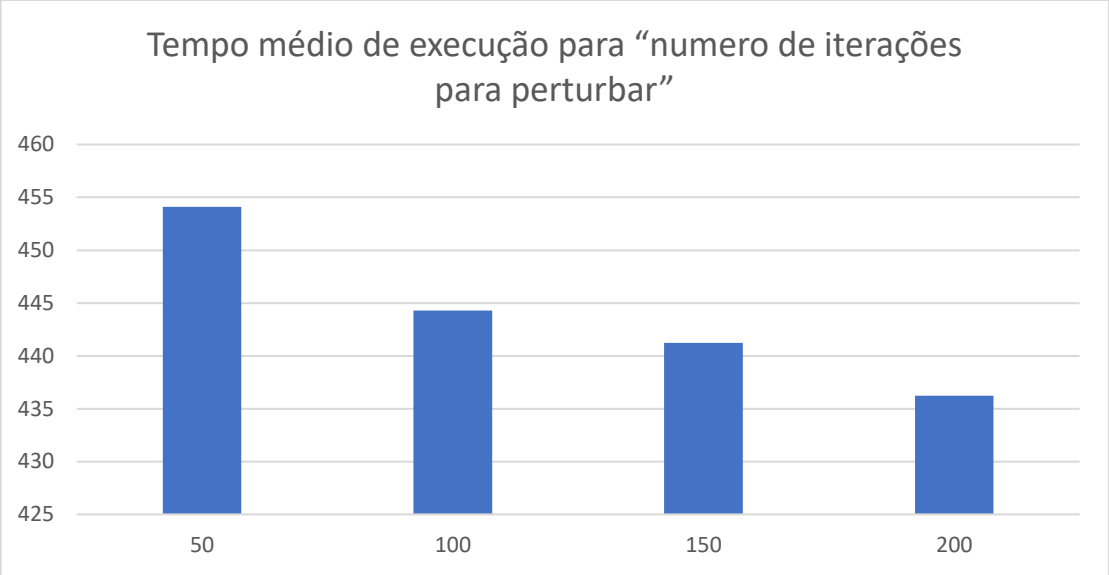

Gráfico 3 - Tempo médio de execução para “Número de iterações para perturbar”

O Gráfico 3 reflete os tempos de execução do TILS referentes ao Gráfico 2. O tempo de execução diminui à medida que o “Número de iterações para perturbar” aumenta, isto deve-se ao facto de cada vez que é aplicada uma “perturbação” o subconjunto de features é avaliado, e como esta operação é feita mais vezes, o tempo de execução aumenta. Ou seja, a cada 50 iterações é aplicado a perturbação e por sua vez o subconjunto é avaliado e este processo demora tempo, se o aplicarmos várias vezes o tempo de execução aumenta.

Seria mais benéfico escolher o valor 200, porque era o melhor valor para que o tempo de execução do algoritmo fosse menor, no entanto, como pretendemos perceber se a “perturbação” é benéfica ou não, optamos por escolher o valor de 50, mesmo com impacto na performance do tempo de execução do algoritmo porque acreditamos que quantas mais vezes a perturbação for aplicada, maior a probabilidade de o resultado ser melhor.

Depois de escolhido o valor que pensamos ser o adequado para o “Número de iterações para perturbar” optamos por tentar descobrir qual seria o melhor valor para a próxima variável. O “Número de operações a adicionar” é uma variável muito importante porque temos dois cenários possíveis de acontecer. O primeiro é adicionar pouca “perturbação” e o algoritmo ter um comportamento fraco e não ter melhorias de performance ou então ser prejudicial. Outro cenário é acrescentar “perturbação” a mais e aqui também não ser uma mais-valia, ou porque “tanta perturbação” acaba por ser prejudicial ou então é tao prejudicial que afeta negativamente o método e não teve benefício nenhum. Para realizar esta análise usamos o mesmo método para a escolha da variável “Número de iterações para perturbar”, mas já tivemos em atenção as variáveis escolhidas anteriormente. Sendo assim o Gráfico 4 mostra o resultado dos testes.

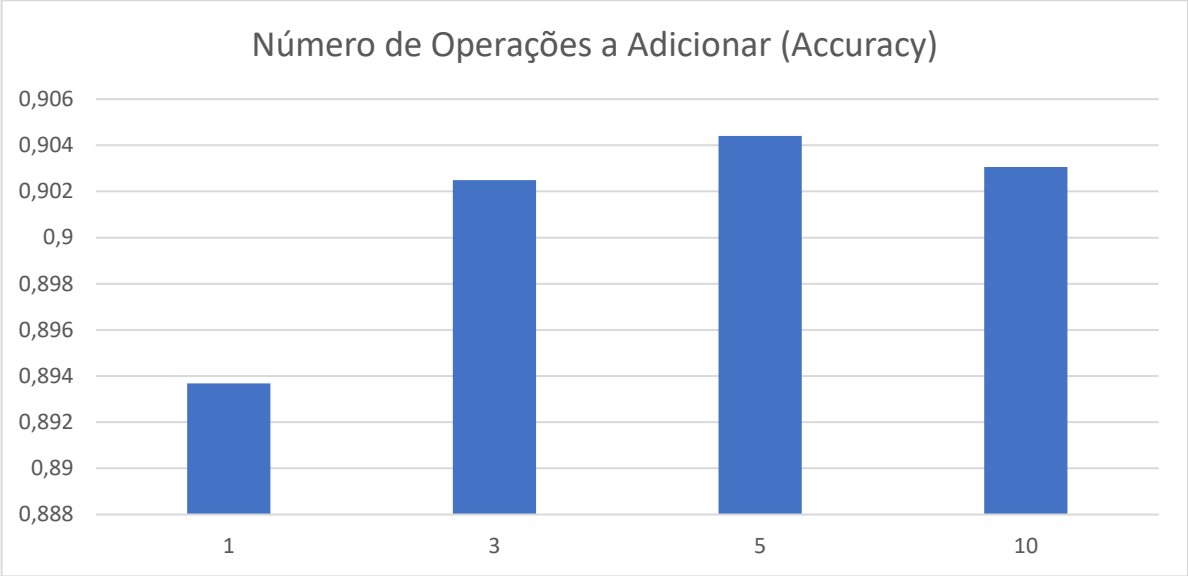

Gráfico 4 - Número de Operações a Adicionar (Accuracy)

Pela análise do Gráfico 4 o melhor valor é 5, apesar de o valor de 3 e de 10 ser muito semelhante. De forma a verificar o impacto desta escolha em termos de tempo, recolhemos dados sobre os tempos de execução do algoritmo de forma a perceber o impacto referente a cada opção da variável. Desta forma o Gráfico 5 mostra o resultado do tempo de execução.

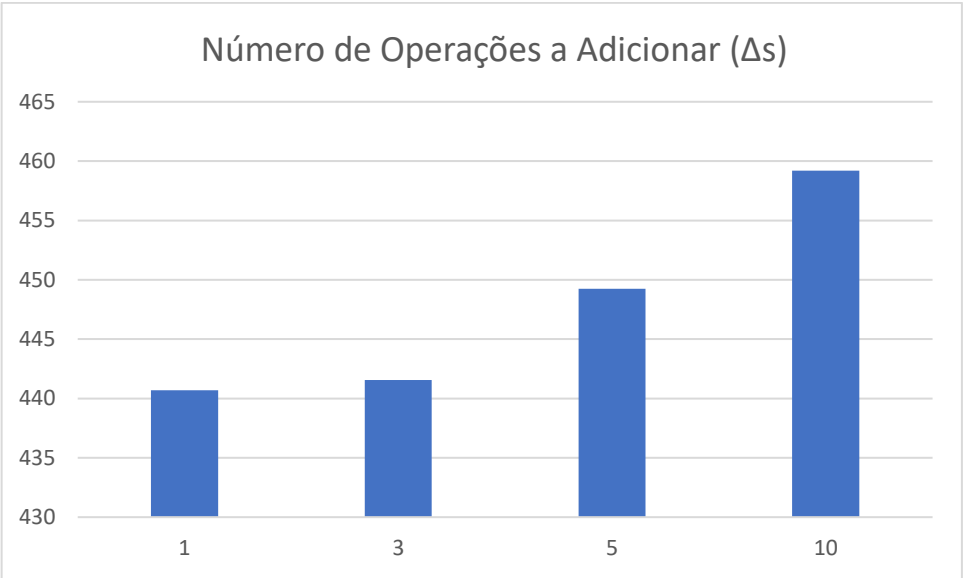

Gráfico 5 - Número de Operações a Adicionar (Δs)

Pela análise do gráfico é fácil perceber que quantas mais operações adicionamos à perturbação mais tempo demora o algoritmo a correr. Sendo o tempo linearmente maior em relação ao número de operações, ou seja, quantas mais operações mais tempo o algoritmo demora a terminar.

Não restando dúvidas optamos por utilizar o valor de 5 para a variável de “Número de operações a adicionar” visto ter sido o valor que obteve melhor performance, apesar do tempo que demora a correr ser mais dispendioso o mesmo não teve muita importância na escolha.

No fim de analisarmos todos os parâmetros para o TILS, optamos por transcrever as mesmas para a Tabela 8 de forma a ficar mais claro o valor de cada um dos parâmetros do método.

Parâmetro	Valor
“Número de Features Iniciais”	0.2 (20%)
“Critério de Terminação”	1000
"Fator de Tendência"	1
“Número de Iterações para Perturbar”	50
“Número de Operações a Adicionar”	5

Tabela 8 - Parâmetros finais do TILS

Da mesma forma que transcrevermos os valores usados pelo TILS, optamos também por descrever na Tabela 9 os valores que decidimos usar na árvore de decisão. Estes valores foram obtidos através da análise da Tabela 5.

Parâmetro	Valor
min_samples_split	3
max_depth	20

Tabela 9 - Parâmetros da Árvore de Decisão

Os valores presentes na Tabela 8 e na Tabela 9 irão ser os valores utilizados daqui em diante para o TILS.

4.7 Resultados

Depois de analisados os melhores valores para os parâmetros do TILS, voltamos a aplicar o método ao dataset MicroRNA. De forma a obter uma melhor perceção dos resultados optamos por correr o método 30 vezes e por cada vez que o método era executado, os valores eram apontados numa tabela, sendo o Gráfico 6 o resultado desses registos.

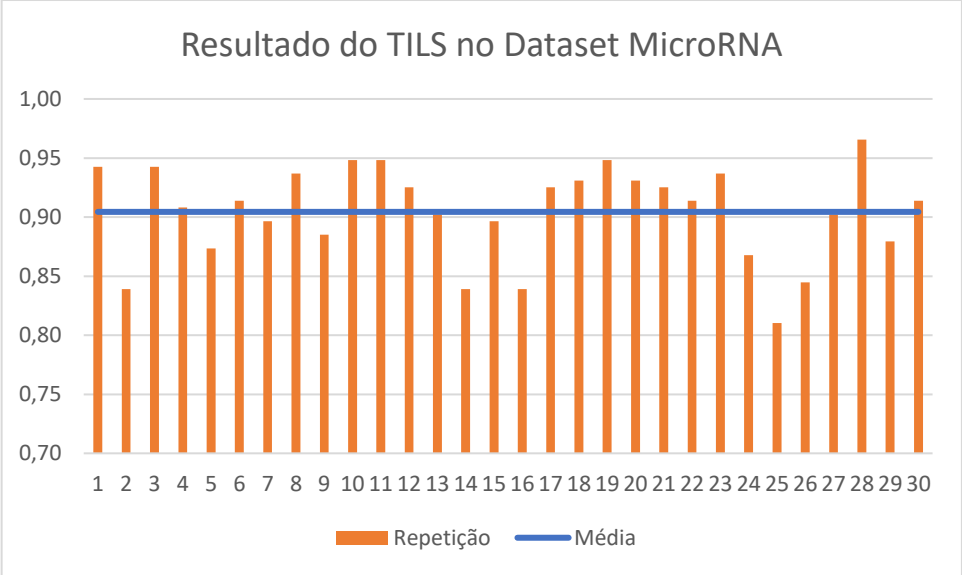

Gráfico 6 - Resultado do TILS no Dataset MicroRNA

No Gráfico 6 o eixo vertical diz respeito à accuracy obtida pelo algoritmo e o eixo horizontal diz respeito ao número da repetição do algoritmo. A linha horizontal diz respeito à média das 30 repetições, tendo esta o valor de 90%. O melhor resultado foi de 97% na repetição 28 e o resultado pior foi na iteração 25 com 81%. Comparando os resultados com os obtidos na Tabela 4, o método desenvolvido apresenta resultados semelhantes aos melhores resultados da Tabela 4. Sendo o melhor resultado da tabela o valor de 93%, o TILS em média obtém 90%. Se compararmos o melhor resultado obtido do TILS, 97%, não existe nenhum método testado que consiga alcançar este valor. Mesmo comparando o pior resultado do TILS, 81%, comparado com os resultados da Tabela 4, o TILS tem um comportamento satisfatório visto que maior parte da accuracy da Tabela 4 é inferior a 81%.

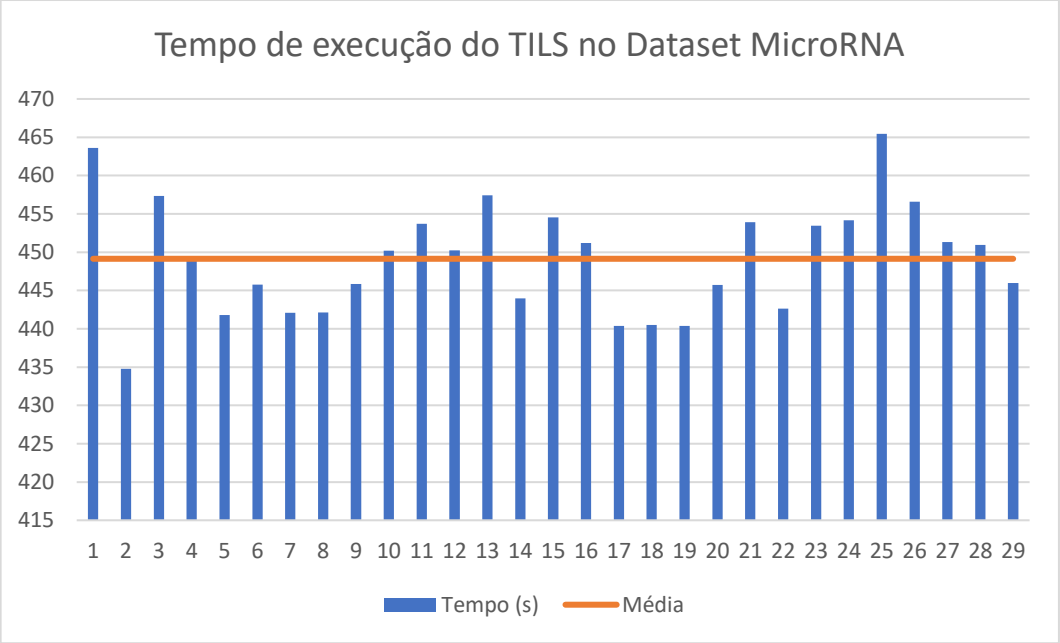

Gráfico 7 - Tempo de execução do TILS no Dataset MicroRNA

No Gráfico 7 o eixo vertical diz respeito ao tempo de execução em segundos do algoritmo e o eixo horizontal diz respeito ao número da repetição do algoritmo. A linha horizontal diz respeito à média das 30 repetições, tendo esta o valor de 449 segundos. O tempo de execução foi menor da iteração 2 com o valor de 435 segundos e o valor máximo foi de 465 na repetição 25. Comparando os resultados com a Tabela 4 o TILS é dos que demora mais tempo até terminar, ficando só atrás do TPOT, os restantes métodos são todos mais rápidos. O TILS poderia ser parametrizado para demorar menos tempo a ser executado se o “Critério de Terminação” fosse ajustado, mas ao ser ajustado poderia ser prejudicial para a sua performance e só benéfico no tempo de execução.

Depois de analisado a performance do algoritmo e o seu tempo de execução é importante avaliar também o subconjunto de features gerado pelo algoritmo. Para isso criamos vários gráficos de forma a perceber melhor os resultados obtidos. O Gráfico 8 mostra a relação entre o número de features inicias e o número de features finais.

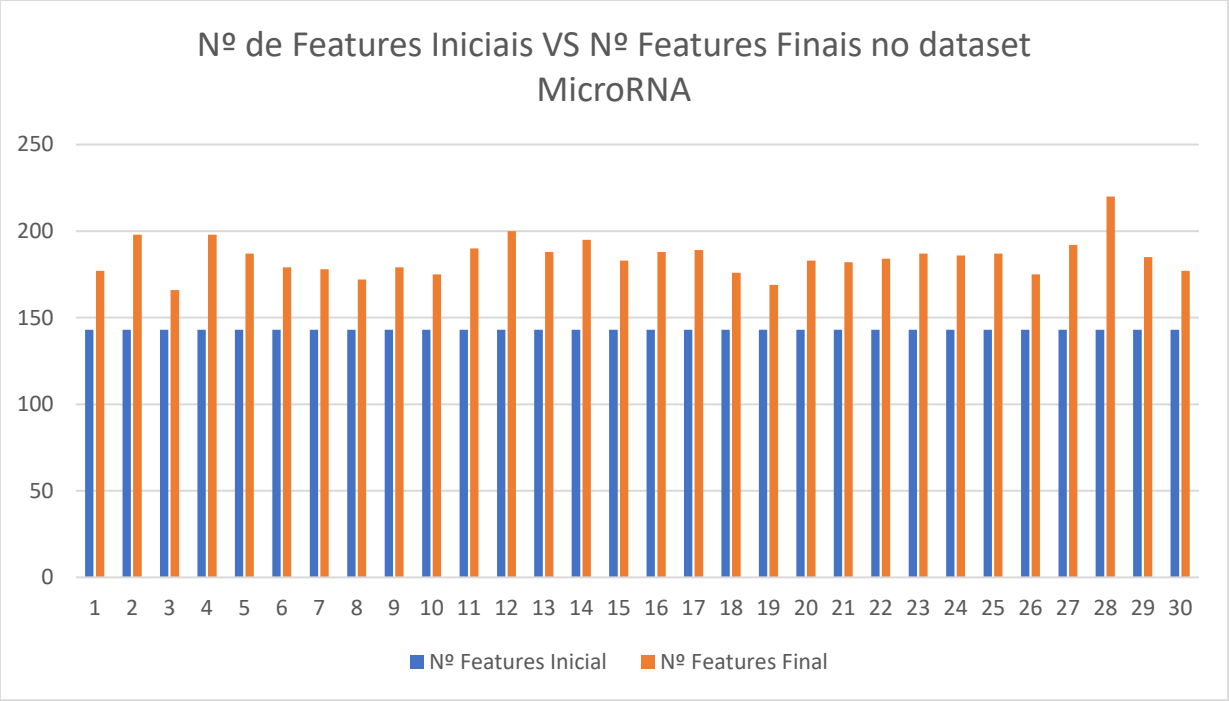

Gráfico 8 - Nº de Features Iniciais VS Nº Features Finais no dataset MicroRNA

No eixo vertical temos o número de features e no eixo horizontal temos o número de repetições do algoritmo. O número de features inicial é sempre igual a 143 porque começamos sempre com 20% de features do dataset inicial. O número de features final é que varia consoante as repetições. O número máximo de features final foi de 220 e diz respeito à repetição 28, neste caso temos uma diferença de 77 features. A iteração 3 é aquela que apresentou um número inferior de features finais, obteve apenas 166. A média de features finais é de 185, ou seja, um aumento médio de 29%

O Gráfico 9 mostra o resultado detalhado das operações matemáticas referentes às operações com duas features.

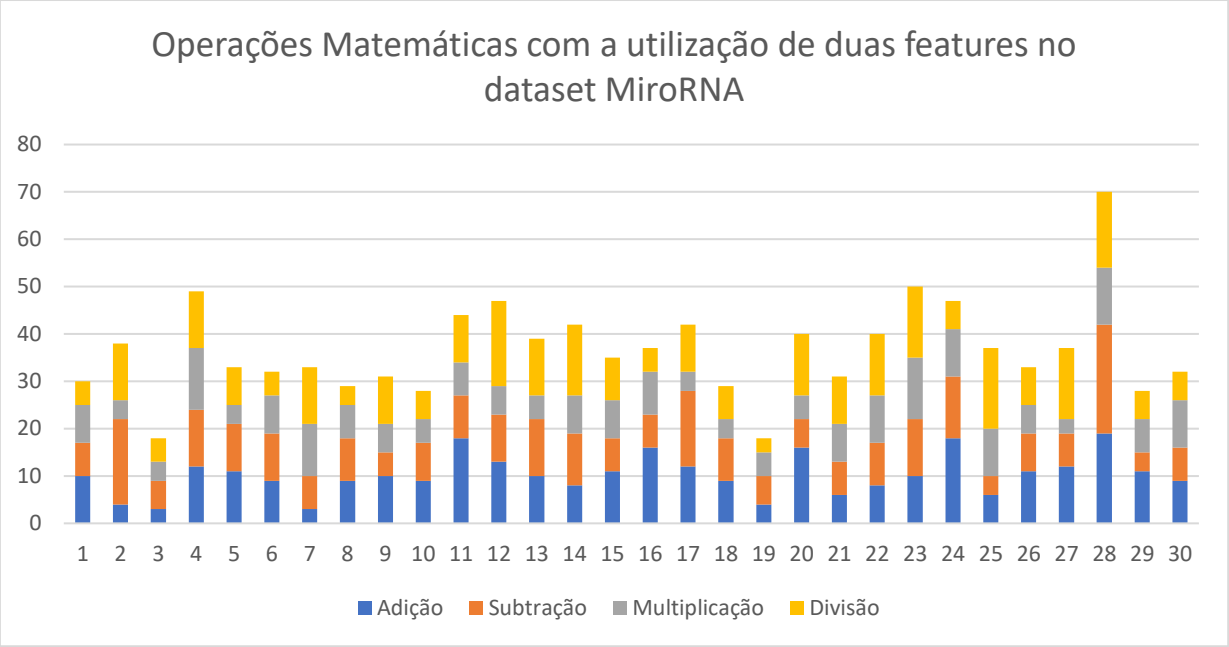

Gráfico 9 - Operações Matemáticas com a utilização de duas features no dataset MicroRNA

No eixo vertical temos o número de operações e o eixo horizontal temos o número de repetições. Pela análise do Gráfico 9 rapidamente se identifica que a repetição 28 foi a que teve mais operações matemáticas com duas features e a repetição 3 a que teve menos. A média de operações é de 37. A cor azul representa o número de operações referentes à soma de duas features, sendo a sua média de 10. A repetição com maior número é a 28 com 19 operações de adição. A repetição com menor número de adições foi a 7 com apenas 3. A cor laranja representa o Número de operações referentes a operações de subtração de duas features, sendo a sua média de 9. O valor máximo foi registado na iteração 28 com o valor de 23. O valor mínimo foi registado na iteração 29 com o valor de 4. A cor cinza diz respeito ao Número de operações referentes à operação de multiplicação, sendo a sua média de 7. O maior valor foi registado na repetição 23 com o valor de 13. O valor mínimo foi registado na iteração 27 com o valor de 3. Por fim, temos o amarelo que diz respeito às operações de divisão, tendo estas uma média de 10. O número máximo de divisões foi registado na iteração 12 com o valor de 18. Na repetição 19 foi registado o valor mínimo de 3.

Depois de analisadas as operações matemáticas com a utilização de duas features vamos agora analisar as operações matemáticas com a utilização de apenas uma feature. O Gráfico 10 mostra os resultados obtidos.

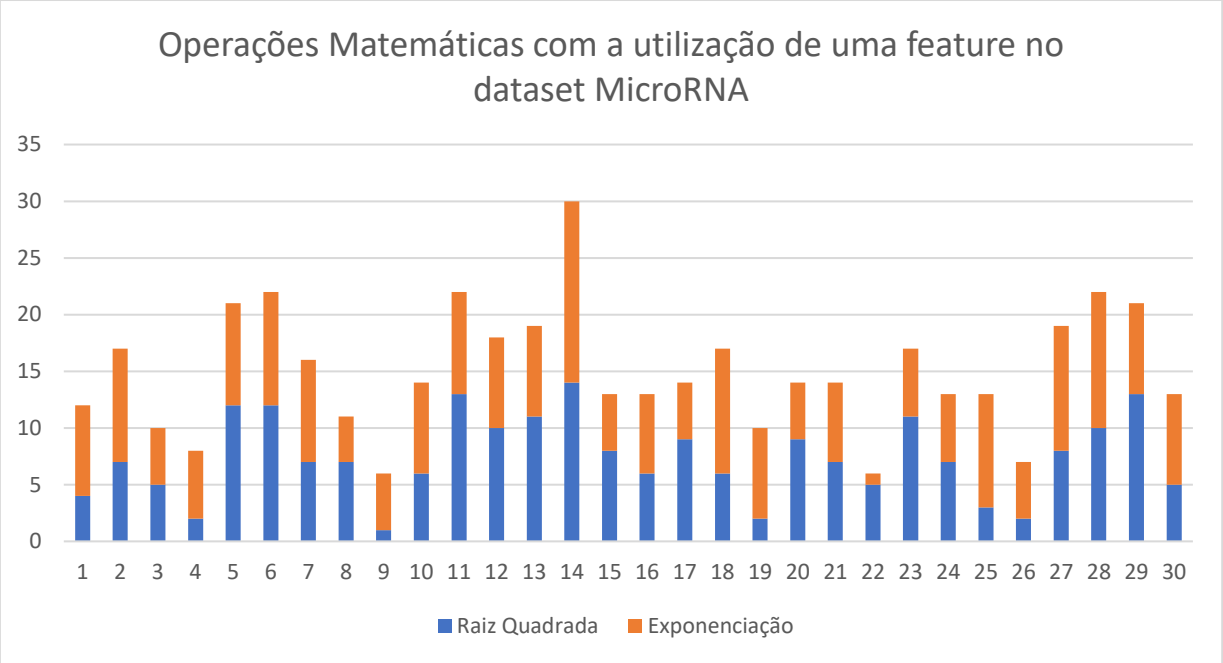

Gráfico 10 - Operações Matemáticas com a utilização de uma feature no dataset MicroRNA

No eixo vertical temos o número de operações e no eixo horizontal temos o número da repetição. A média de operações foi de 15. A iteração 14 foi a que obteve o maior número de operações com o valor de 30. O menor valor de operações foi registado na iteração 9 e 22 com o valor de 6. A cor azul diz respeito ao número de operações referentes à aplicação da raiz quadrada, sendo a sua média de 7. O valor máximo foi registado na iteração 14 com 14 operações. O valor mínimo foi obtido na iteração 9 com apenas uma utilização. A cor laranja representa o número de operações referentes à operação de exponenciação, sendo a sua média de 8. O número máximo foi de 16 registado na iteração 14. O valor mínimo foi de 1 e foi registado na iteração 22.

Depois de analisado os valores referentes às operações matemáticas com a utilização de apenas uma feature vamos analisar as operações restantes que são as não matemáticas. O Gráfico 11 mostra os resultados obtidos.

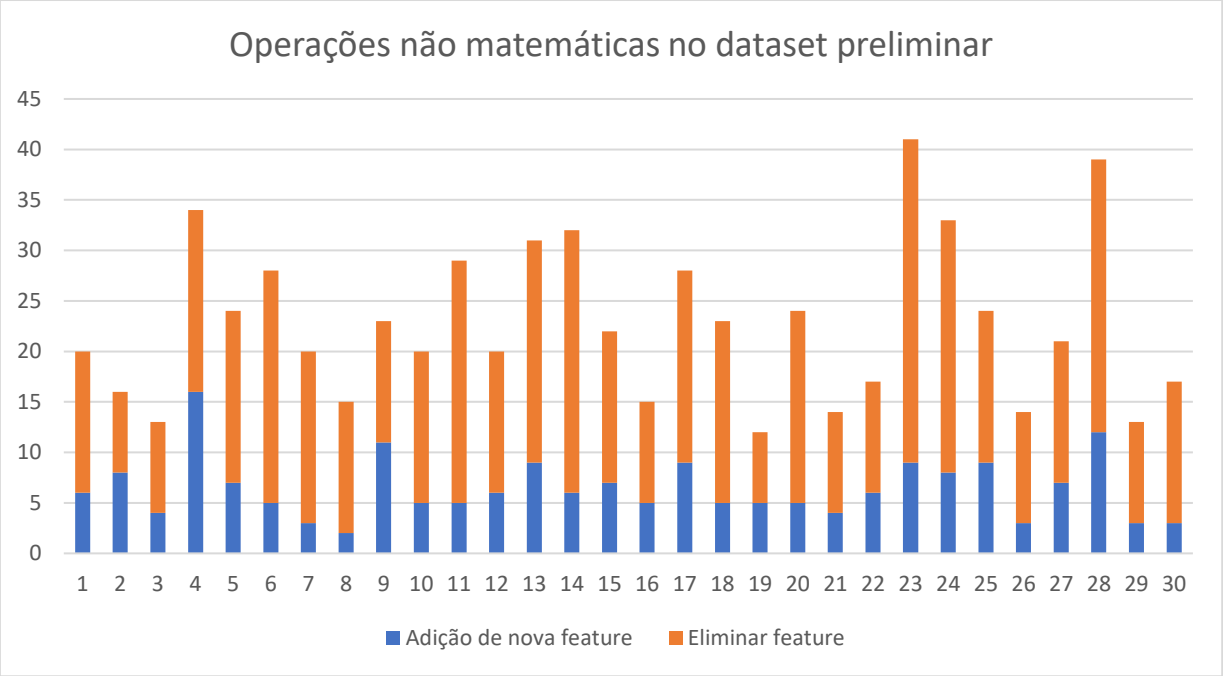

Gráfico 11 - Operações não matemáticas no dataset MicroRNA

No eixo vertical temos o número de operações e no eixo horizontal temos o número da repetição. A média de operações foi de 23. O número de máximo de operações não matemáticas foi registado na repetição 23 com o valor de 41 e o menor valor foi registado na iteração 19 com o valor de 12. A cor azul representa a adição de uma feature nova ao dataset, sendo a sua média de 6. O valor máximo registado foi de 16 na iteração 4 e o valor mínimo foi de 2 na iteração 8. A cor laranja representa a operação de eliminação de uma feature e a sua média foi de 16. O valor máximo registado foi de 32 na repetição 23 e o valor mínimo foi de 7 na iteração 19.

Analisados todos os conjuntos de operações possíveis, decidimos criar um gráfico com todas as operações possíveis de forma a ter uma visão geral de todas as operações. O Gráfico 12 mostra o resultado.

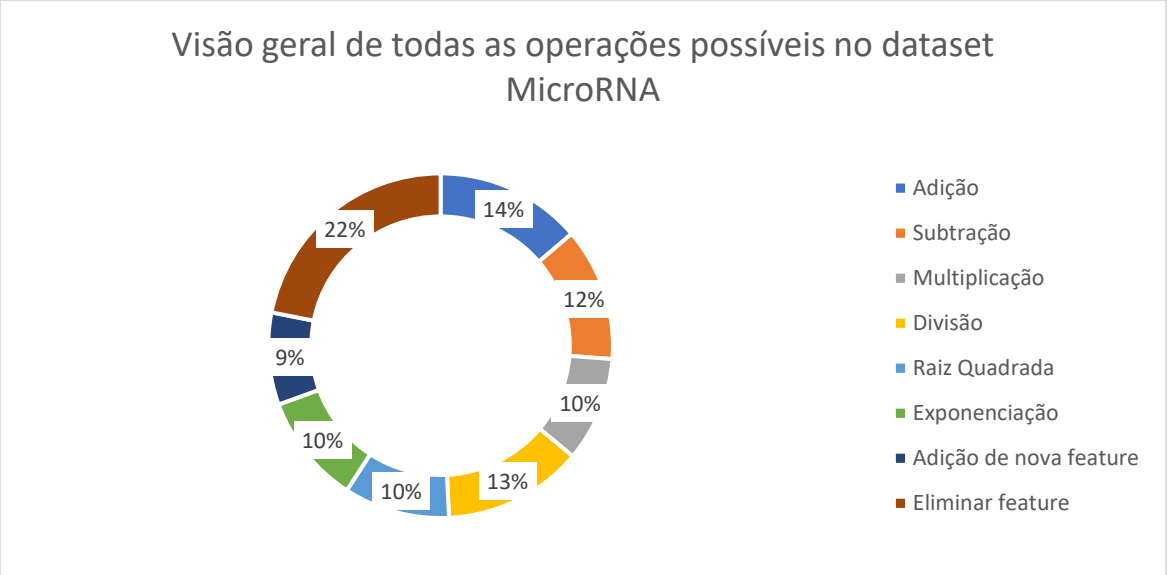

Gráfico 12 - Visão geral de todas as operações possíveis no dataset MicroRNA

A operação de eliminação de feature é aquela que é utilizada mais vezes pelo algoritmo. A operação de adicionar uma nova feature é a menos utilizada. As restantes operações são uniformes.

As operações matemáticas com uma ou duas features são as mais uniformes e são usadas de forma semelhante no algoritmo. Podemos concluir que as operações matemáticas são bons operadores para a transformação de features.

Podemos concluir que a operação de eliminação de features é aquela que traz maior benefício no nosso algoritmo e que a operação de adicionar uma nova feature é que traz menor benefício. A operação de eliminação acabou por ser mais relevante no nosso algoritmo porque com a adição de perturbação e quando a operação de eliminação era escolhida trazia benefícios em questão de performance ao nosso modelo, como optamos por usar o fator de tendência a operação voltava a ser utilizada até deixar de trazer benefício. Dai os valores de eliminação de feature serem maiores.

O facto de trazermos uma nova feature para o subconjunto de features acabou por ser pouco usado porque não trazia benefícios, dai o seu número ser o mais baixo.

5 Dataset “Amyotrophic Lateral Sclerosis (ALS)”

5.1 Descrição do problema

A esclerose lateral amiotrófica (ELA), do inglês Amyotrophic Lateral Sclerosis (ALS), também conhecida como doença de Lou Gehrig ou doença de Charcot, é uma doença neurodegenerativa progressiva rara do sistema nervoso central, que evolui causando atrofia progressiva da musculatura respiratória e dos membros além de sintomas de origem bulbar como disartria e disfagia, resultando em morte ou ventilação mecânica permanente (Pontes et al., 2001). É a forma mais frequente de Doença do Neurónio Motor (DNM) (Apela, 2022).

Esta doença afeta mais de 200.000 pessoas em todo o mundo, com cerca de 800 doentes em Portugal (*Esclerose lateral amiotrófica* | *CUF*, 2021). A fraqueza muscular progressiva dos músculos respiratórios é responsável por Insuficiência Respiratória, a principal causa de morte na ELA. Os doentes ficam mais suscetíveis a infeções respiratórias, atelectasias (colapso de segmentos dos pulmões) e a má oxigenação sanguínea (Apela, 2022).

A sobrevida média é de cerca de 5 anos nas formas medulares, sendo inferior na forma bulbar, e pior nas formas respiratória e difusa. Existem formas muito rapidamente progressivas, mas também existem formas lentamente progressivas nas quais os doentes sobrevivem 10 a 15 anos do início dos primeiros sintomas. Por vezes, são observadas situações atípicas, de evolução ainda mais lenta da progressão. A evolução de cada caso é sempre imprevisível.

Em 1993, um grupo de investigação demonstrou que algumas formas hereditárias de ELA são causadas por mutações. Uma das mudanças genéticas altera uma enzima abundante dentro de células chamada de cobre-zinco superóxido dismutase (SOD1). Esta enzima serve para proteger as células de resíduos metabólicos, que podem causar danos se não forem neutralizados. A SOD1 converte oxigénio reativo em água. Um ião de cobre presente dentro da enzima realiza esta reação química. Um átomo de zinco também é importante para a função da enzima (Donnelly et al., 2013).

Até hoje, mais de 100 mutações diferentes na SOD1 foram ligadas a ELA familiar (hereditária). A vasta maioria destas mutações altera a sequência de aminoácidos da proteína SOD1 numa única posição. Essa posição é diferente para a maioria das famílias.

O que define a ELA e muitas outras doenças neurodegenerativas é a acumulação anormal de proteínas chamadas depósitos ou agregados. Os cientistas agora sabem que estes agregados em murganhos que possuem ELA, contêm SOD1 mutante. Muitos locais diferentes dentro da proteína SOD1 são alterados pelas diferentes mutações que os investigadores identificaram em famílias com ELA familiar. O aspeto comum para todas estas mutações é a alteração da estrutura da SOD1.

É comum encontrar na literatura especializada e em materiais dirigidos a pacientes a informação de que o gene SOD1 é o principal responsável pelos dos casos de ELA hereditários (cerca de 20%) – casos nos quais a doença é transmitida de uma geração para outra. Entretanto, a descoberta de uma expansão de 6 nucleotídeos no cromossoma 9 modificou drasticamente este cenário (Oliveira, 2017), uma vez que esta representa entre 5% a 10% dos casos de ELA familiar. No entanto a mutação da SOD1 é interessante do ponto de vista molecular, uma vez que está diretamente ligada com alterações do equilíbrio redox e mitocondrial.

5.2 Análise do dataset

Este dataset foi obtido no âmbito do projeto Mito4ALS, financiado pela FCT (AMA – Agência para a Modernização Administrativa, 2022) e está dividido em duas categorias distintas, a primeira é “Mitostress clean” e contém os registos sobre indicadores da saúde mitocondrial recolhidos através de amostras biológicas. A segunda é “ATP Rate clean” e contém a informação relativa aos parâmetros relacionados com a produção de energia celular.

Devido à variabilidade da cultura celular, as medições foram repetidas por diversas vezes, sendo assim, os datasets contêm, para cada amostra, réplicas biológicas e réplicas técnicas. Para melhor perceção do estudo, é preciso perceber a diferença entre esses dois conceitos: Réplica biológica – consiste em analisar amostras biologicamente independentes; Réplica técnica - consiste em analisar uma mesma amostra biológica independentemente várias vezes. Várias replicas técnicas dizem respeito a uma réplica biológica, as mesmas estão ainda relacionadas com o dia em que foram recolhidas, de maneira a identificar o parâmetro N que pode ser 1,2,3, etc.

O estudo engloba um grupo controlo (LH) e doentes que poderão ter uma mutação conhecida no gene SOD 1 (LPS) ou não (LP). As células dos doentes e dos controlos foram adquiridas ao biobanco Coriell Repositories (*US Biobank* | *Browse Our Diverse Biobanks* | *Coriell Institute*, 2021) e divididas em três ‘famílias’ com o mesmo género e idades semelhantes. A tabela seguinte apresenta um resumo das características das amostras:

Tabela 10 - Tabela de características

Amostra	Grupo	Idade (anos)	Género	Família	Mutação
LH4	Controlo (LH)	46	F	1	não conhecida
LH3	Controlo (LH)	46	M	2	não conhecida
LH5	Controlo (LH)	26	M	3	não conhecida
LP1	Sem mutação (LP)	46	F	1	não conhecida
LP3	Sem mutação (LP)	46	M	2	não conhecida
LP5	Sem mutação (LP)	27	M	3	não conhecida
LPS1	Com mutação (LPS)	46	F	1	ALA4VAL
LPS3	Com mutação (LPS)	46	M	2	ILE113THR
LPS5	Com mutação (LPS)	26	M	3	ALA4VAL

O grupo LH (Controlo), contém amostras de pessoas sem doença. Os LP e LPS correspondem a amostras de pessoas com a doença. Sendo que os LP correspondem ao grupo de amostras sem mutações conhecidas e os LPS ao grupo de amostras com mutações conhecidas. Cada linha da Tabela 10, corresponde ao grupo de amostras com as mesmas características. Por exemplo, todas as amostras LH4 correspondem a um indivíduo de Controlo (LH), com 46 anos (à data da recolha), do género Feminino, da família 1 e com mutação não conhecida. Para todos os participantes no estudo foram avaliados diferentes parâmetros relacionados com as taxas de consumo de oxigénio (OCR) e acidificação extracelular (ECAR). Os parâmetros avaliados correspondem medidas do metabolismo energético de células em cultura. A Figura 12 detalha os atributos recolhidos:

MitoStress	ATP Rate
Basal Respiration (OCR)	Total ATP Production Rate
ATP-linked Respiration (OCR)*	Glyco ATP Production Rate**
Proton Leak Respiration (OCR)**	Mito ATP Production Rate*
Maximal Respiration (OCR)*	XF ATP Rate Index
Spare Capacity (OCR)*	
Non-mitochondrial Respiration (OCR)**	
Basal ECAR (ECAR)**	
Stressed ECAR (ECAR)	
Basal OCR	
Stressed OCR	

Nota: Os parâmetros indicadores da saúde mitocondrial encontram-se destacados a verde. A vermelho destacam-se os parâmetros relacionados com stress ou disfunção mitocondrial.

Figura 12 - Parâmetros Avaliados em cada amostra

De modo a poder compreender melhor o dataset fornecido, optamos por usar a ferramenta Excel para fazer uma rápida análise dos dados.

5.2.1 Aspeto geral dos dados relativos a “Mitostress Clean”

Este dataset contém cerca de 531 registos e 16 colunas tendo as colunas as seguintes características:

- **ID**: Número identificador de cada registo do dataset.
- **Family**: Identifica a família de cada amostra:
  - 1 - 46 anos de idade e do sexo feminino
  - 3 - 46 anos de idade e do sexo masculino
  - 5 - 26 anos de idade e do sexo masculino
- **Target**: Classificação do grupo de cada pessoa, é Target do dataset. O seu resultado é dividido nos seguintes campos:
  - LH - Grupo Controlo
  - LP - Doentes que possuem uma mutação conhecida no gene SOD1
  - LPS - Doentes que não possuem uma mutação conhecida no gene SOD1
- **Description**: Representa os participantes do estudo.
- **Data**: Data em que a amostra foi analisada.
- **N**: Cada N representa uma réplica biológica.
- **Non mitochondrial respiration (OCR)** - Taxa de consumo de oxigénio que ocorre fora da mitocôndria;
- **Basal respiration (OCR)** - Taxa de consumo de oxigénio quando a célula está em repouso;
- **Maximal respiration (OCR)** - Taxa máxima de consumo de oxigénio quando a célula está em stress;
- **Proton leak (OCR)** - Diferença da taxa de consumo de oxigénio entre a aplicação de Oligomicina e aplicação de Antimicina A + Rotenona.
- **ATP production (OCR)** - Diferença entre a taxa de consumo de oxigénio depois da aplicação do reagente oligomicina e o proton leak.
- **Spare respiratory capacity (OCR)** - Diferença entre as taxas de consumo de oxigénio basal e máxima das células;

- **Baseline OCR** - Taxa de consumo de oxigénio quando a célula está em repouso; equivalente à basal respiration mas sem descontar a non mitochondrial respiration.
- **Baseline ECAR** - Taxa de produção de ácido extracelular quando a mitocôndria está em repouso.
- **Stressed OCR** - Taxa de consumo de oxigénio quando a célula é estressada pelo equipamento de Respirometria; equivalente à maximal respiration mas sem descontar a non mitochondrial respiration.
- **Stressed ECAR** - Taxa de acidificação extracelular quando a mitocondria é estressada pela adição de FCCP.

Na Tabela 11 mostramos para cada coluna os valores médios, máximos, mínimos, e os valores possíveis.

Tabela 11 - Aspeto geral do dataset "Mitostress Clean"

Coluna	Tipo de Valores	Mínimo	Média	Máximo	% Valores em falta
ID	Numérico	1	283	610	0%
Familiy	Numérico	1	3,2	5	0%
Target	Alfanumérico	-	-	-	0%
Description	Alfanumérico	-	-	-	0%
Data	Data	-	-	-	0%
N	Numérico	1	5	12	0%
Non mitochondrial respiration	Numérico	1,03	24,56	64,96	0%
Basal respiration	Numérico	25,25	75,05	220,97	0%
Maximal respiration	Numérico	51,22	171,27	480,13	0%
Proton leak	Numérico	0,09	15,79	58,46	0%
ATP production	Numérico	0,39	59,25	186,85	0%
Spare respiratory capacity	Numérico	13,56	96,22	349,35	0%
Baseline OCR	Numérico	37,52	99,61	263,18	0%
Baseline ECAR	Numérico	40,60	85,22	130,65	0%
Stressed OCR	Numérico	59,17	195,83	526,98	0%
Stressed ECAR	Numérico	52,75	111,92	167,56	0%

Relativamente ao nº de amostras do dataset em análise, Mitostress Clean, é possível constatar que o nº de amostras referentes a cada um dos targets, é similar entre os diferentes tipos de target existentes. O Gráfico 13 mostra a distribuição das amostras consoante o target.

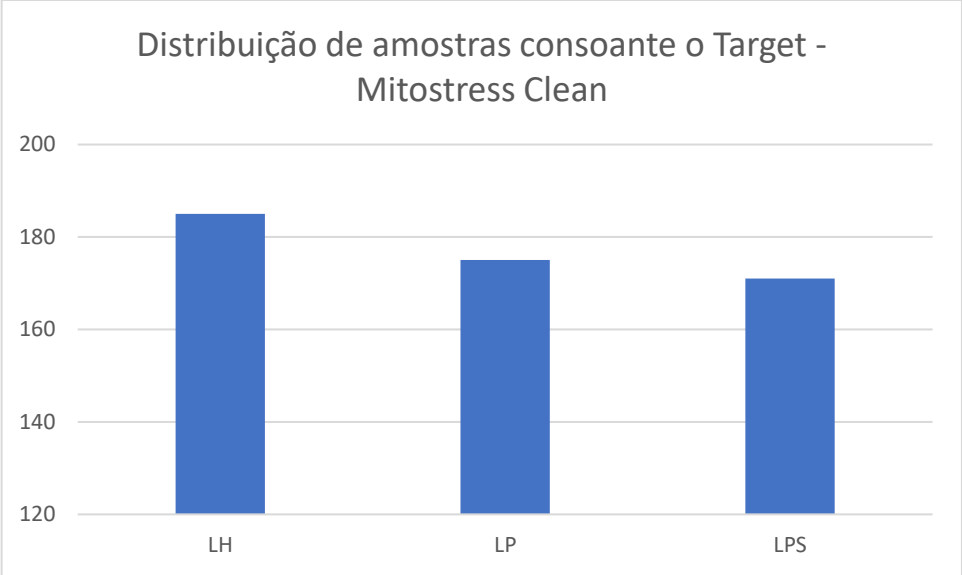

Gráfico 13 - Distribuição de amostras consoante o Target - Mitostress Clean

Dessa forma, foi possível identificar o seguinte nº de amostras, respetivamente para cada um dos targets:

- 185 amostras LH;
- 175 amostras LP;
- 171 amostras LPS;

Depois fomos analisar a quantidade de amostras para cada target de forma a identificar o número de registos por família. Sendo assim obtivemos os seguintes resultados:

- **Família 1**:
  - LP: 52 Amostras;
  - LH: 47 Amostras;
  - LPS: 49 Amostras;
- **Família 3**:
  - LP: 64 Amostras;
  - LH: 60 Amostras;
  - LPS: 59 Amostras;
- **Família 5**:
  - LP: 59 Amostras;

- LH: 78 Amostras;
- LPS: 63 Amostras;

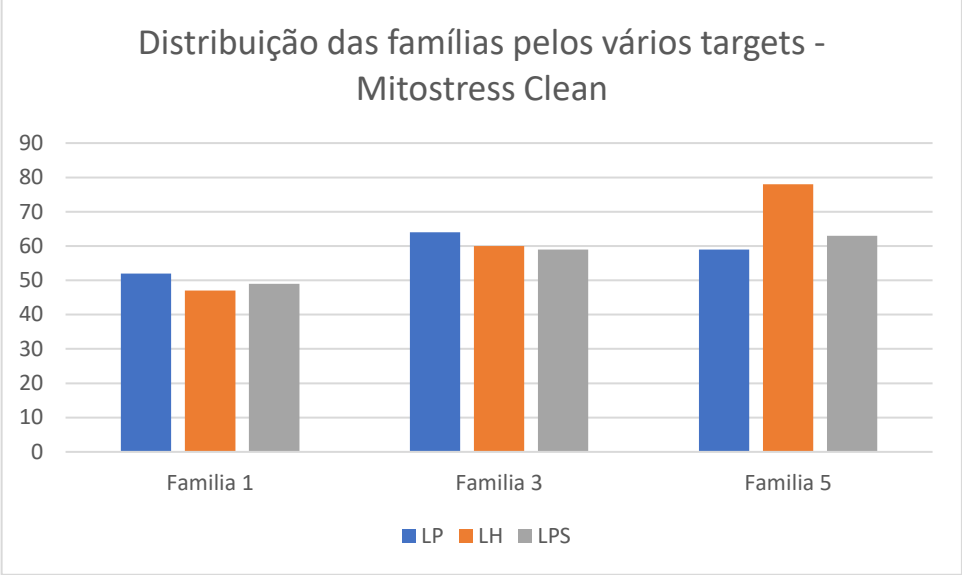

Gráfico 14 - Distribuição das famílias pelos vários targets - Mitostress Clean

Pela análise do gráfico é possível concluir que o nº de amostras inerentes à família 5 (26 anos masculino) é ligeiramente superior às restantes famílias.

Em geral os dados relativos a Mitostress Clean mostraram ser uniformes, não existem grandes variações entre grupos. Os valores por target também estão equilibrados e não existem valores em falta. Contudo optamos por retirar algumas colunas na aplicação dos nossos métodos porque não fazem sentido para os objetivos. As colunas que retiramos foram as seguintes:

- ID
- Familiy
- N
- Data
- Description

5.2.2 Aspeto geral dos dados relativos a "ATP Rate Clean"

Este dataset contém cerca de 540 registos e 10 colunas tendo as colunas as seguintes características:

- **ID**: Número identificador de cada registo do dataset.

- **Family**: Identifica a família de cada amostra:
  - 1 - 46 anos de idade e do sexo feminino
  - 3 - 46 anos de idade e do sexo masculino
  - 5 - 26 anos de idade e do sexo masculino
- **Target**: Classificação do grupo de cada pessoa, é Target do dataset. O seu resultado é dividido nos seguintes campos:
  - LH - Grupo Controlo
  - LP - Doentes que poderão não ter uma mutação conhecida no gene SOD1
  - LPS - Doentes que possuem uma mutação conhecida no gene SOD1
- **Description**: Representa os participantes do estudo.
- **Data**: Data em que a amostra foi analisada.
- **N**: Cada N representa uma réplica biológica.
- **Total ATP Production Rate (Basal) (pmol/min)** - Taxa total de produção de energia (ATP) pela célula;
- **glyco-ATP Production Rate (Basal) (pmol/min)** - Taxa de produção de energia (ATP) produzida fora da mitocondria;
- **mito-ATP Production Rate (Basal) (pmol/min)** - Taxa de produção de energia (ATP) produzida pela mitocondria;
- **XF ATP Rate Index (Basal)** - Rácio do Mito ATP Production Rate pelo Taxa de produção de energia (ATP) produzida fora da mitocondria.

Na tabela abaixo mostramos para cada coluna os valores médios, máximos, mínimos, e os valores possíveis. O campo Target representa o grupo da amostra e pode tomar os valores LH, LP, LPS.

Tabela 12 - Aspeto geral do dataset "ATP Rate Clean"

Coluna	Tipo de Valores	Mínimo	Média	Máximo	% Valores em falta
ID	Numérico	1	283	610	0%
Familiy	Numérico	1	3,23	5	0%
Target	Alfanumérico	-	-	-	0%
Description	Alfanumérico	-	-	-	0%
Data	Data	-	-	-	0%
N	Numérico	1	4	10	0%

Coluna	Tipo de Valores	Mínimo	Média	Máximo	% Valores em falta
Total ATP Production Rate (Basal) (pmol/min)	Numérico	417,64	1026,20	1671,90	0%
glyco-ATP Production Rate (Basal) (pmol/min)	Numérico	216,56	582,71	936,97	0%
mito-ATP Production Rate (Basal) (pmol/min)	Numérico	1,13	443,48	953,61	0%
XF ATP Rate Index (Basal)	Numérico	0,00	0,80	2,44	0%

Analisando o nº de amostras relativas ao dataset em análise, ATP Rate Clean, é possível constatar que o nº de amostras referentes a cada um dos targets, é similar entre os diferentes tipos de target existentes. O Gráfico 15 mostra a distribuição das amostras consoante o target.

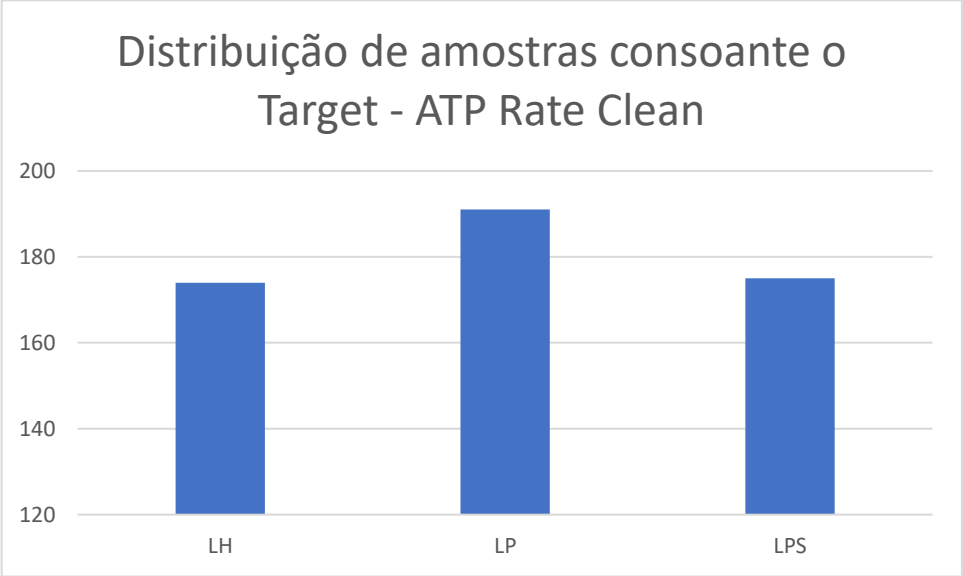

Gráfico 15 - Distribuição de amostras consoante o Target - ATP Rate Clean

Desta forma, foi possível identificar o seguinte nº de amostras, respetivamente para cada um dos targets:

- 174 amostras LH;
- 191 amostras LP;
- 175 amostras LPS;

Aqui rapidamente se percebe que os registos referentes ao target “LP” são ligeiramente superiores aos restantes targets. Neste caso temos cerca de 10% mais registos neste target do que nos restantes.

Depois fomos analisar a quantidade de amostras para cada target de forma a identificar o número de registos por família. Sendo assim obtivemos os seguintes resultados:

- **Família 1**:
  - LP: 55 Amostras;
  - LH: 52 Amostras;
  - LPS: 52 Amostras;
- **Família 3**:
  - LP: 56 Amostras;
  - LH: 53 Amostras;
  - LPS: 52 Amostras;
- **Família 5**:
  - LP: 80 Amostras;
  - LH: 69 Amostras;
  - LPS: 71 Amostras;

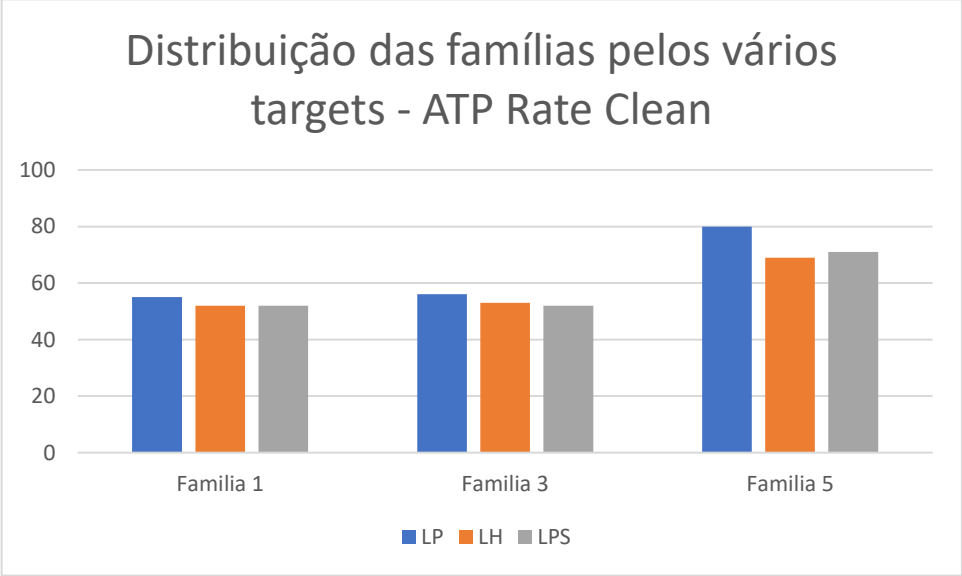

Gráfico 16 - Distribuição das famílias pelos vários targets - ATP Rate Clean

Pela análise do gráfico é possível concluir que o nº de amostras inerentes à família 5 (26 anos masculino) é ligeiramente superior às restantes famílias.

Em geral os dados relativos a “ATP Rate Clean” mostraram ser uniformes, não existem grandes variações entre grupos. Os valores por target também estão equilibrados e não existem valores em falta. Contudo, tal como para o dataset anterior, optamos por

retirar algumas colunas na aplicação dos nossos algoritmos porque achamos que não traziam mais valias para a performance do mesmo. As colunas que retiramos foram as seguintes:

- ID
- Familiy
- N
- Data
- Description

5.3 Conclusões sobre o dataset

O estudo exploratório que foi realizado focou-se na possibilidade de identificar padrões normais de medição de cada coluna em cada dataset. Sabendo isto, seria fácil excluir eventuais erros de medição ou outliers, limpando dados incorretos. Apesar da identificação de ligeiras alterações entre as medições de amostras diferentes referentes ao mesmo individuo, não se torna claro que, se pode tomar as medições do grupo do controlo como certas dos valores sobre os quais devem estar as outras medições, ou de outro grupo sem mutação. Descartar medições mediante estas observações implicaria retirar grande parte das medições realizadas para cada amostra.

No estudo exploratório realizado não foram encontrados valores em falta. Apesar de existirem valores que poderiam ser considerados outliers por serem muito diferentes dos restantes não há informação suficiente para concluir se eles são naturais ou não, podendo conter informação relevante para a separação das classes. Optamos então por não alterar os valores desses atributos.

No estudo exploratório não foi necessário efetuar nenhuma operação de limpeza dos dados porque não encontramos valores em falta nem consideramos outliers. Também não efetuamos nenhuma operação de transformação dos dados porque todos os dados estavam no mesmo formato não existindo necessidade de realizar qualquer operação.

6 Engenharia de features no dataset “Amyotrophic Lateral Sclerosis (ALS)”

6.1 Introdução

Este capítulo tem como objetivo efetuar engenharia de features ao dataset “Amyotrophic Lateral Sclerosis (ALS)”. Como este dataset está dividido em duas categorias distintas, “Mitostress clean” e “ATP Rate clean” vamos começar pelo “Mitostress clean” e depois o “ATP Rate clean”.

6.2 Dataset “Mitostress Clean”

De forma a ter um termo de comparação, aplicamos isoladamente cada método ao dataset e registamos os resultados na Tabela 13. No “Anexo C – Ficheiros relativos ao dataset Amyotrophic Lateral Sclerosis (ALS)” encontra-se os ficheiros python implementados nesta secção.

Tabela 13 - Resultado da aplicação dos métodos ao dataset “Mitostress Clean”

Método	Nº Features Originais	Nº Features Novas	Δ t (s) Seleção de features	Δ t (s) LOOCV	Accuracy
-	10	0	-	3,68	0.68
Qui-Quadrado	4	0	0,16	2,68	0.52
Eliminação Recursiva de Features	5	0	0,97	2,99	0.66
Regressão de Ridge	5	0	0,17	2,94	0.66
Regressão de Lasso	4	0	3,13	3,11	0.66
Ganho de Informação	4	0	0,17	2,74	0.60
Autofeat	10	20	380,86	8,85	0.60
TPOT	10	0	15860,69	3,61	0.67
H20 AutoML	5	0	1657,11	2,95	0.63
Featuretools	10	36	1,12	9,12	0.50

A Tabela 13 é composta por 6 colunas, a primeira coluna diz respeito ao método utilizado. A segunda coluna diz respeito ao número de features originais resultantes da aplicação do método. A terceira coluna indica o número de features criadas pelo método. A quarta coluna temos o tempo em segundos que o método demorou a ser executado. Na quinta coluna temos o tempo de execução da avaliação do modelo. Por último, na sexta coluna temos a accuracy do método.

Pela análise da Tabela 13 podemos reparar em vários fatores, um deles é o número de features, quase todos os métodos utilizaram 10 ou menos features, apenas 2 frameworks utilizaram mais de 30 features. Outro fator é o tempo de execução do método para a seleção de features, temos uma vasta disparidade, temos métodos a demorar menos de 1 segundo e outros a demorar mais de 100, ou seja, 100 vezes mais. Quase todos os métodos demoraram menos de 1 segundo a escolher as melhores features, porem, o TPOT e o H20 AutoML foram os que demoram mais tempo, com 15860segundos e 1657 segundos respetivamente. Outro fator que podemos reparar na tabela é o tempo de execução da avaliação do modelo, varia entre os 2 e os 9 segundos. Portanto não é um fator relevante porque apresenta sempre valores baixos. Em termos de accuracy os valores variam entre os 50% e os 68%, sendo os valores mais baixos registados pelo Featuretools e pelo qui-quadrado, o melhor resultado foi obtido sem a aplicação de nenhuma ferramenta, ou seja, com todas as features.

Outra observação é que o Qui-Quadrado e o Ganho de informação foram os métodos tradicionais com piores prestações, os restantes métodos tradicionais obtiveram performances semelhantes, esta observação pode indicar que estes dois métodos não se comportam bem em datasets com número reduzido de features. As frameworks não revelaram trazer grande benefício neste dataset porque os resultados não são vantajosos em comparação com a performance obtida sem a utilização de nenhum método. Este facto pode dever-se ao formato do dataset que possui um número reduzido de features em relação ao número de amostras.

6.3 Resultados da aplicação do TILS

De forma a obter uma melhor perceção dos resultados optamos por correr o TILS 30 vezes e por cada vez que o método corria, os valores eram apontados numa tabela, sendo o Gráfico 17 o resultado desses registos.

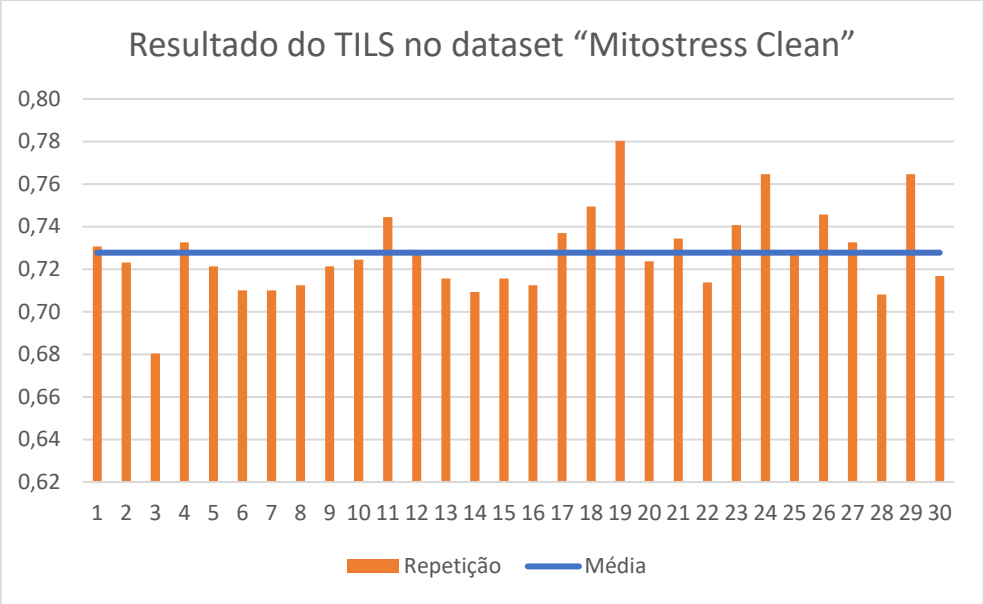

Gráfico 17 - Resultado do TILS no dataset “Mitostress Clean”

No Gráfico 17 o eixo vertical diz respeito à accuracy obtida pelo TILS e o eixo horizontal diz respeito ao número da repetição do método. A linha horizontal diz respeito à média das 30 repetições, tendo esta o valor de 73%. O melhor resultado foi de 78% na repetição 19 e o resultado pior foi na iteração 3 com 68%. Comparando estes resultados com a Tabela 13, o pior resultado do Gráfico 17 é o melhor da Tabela 13. Estes resultados podem indicar que o TILS é um método com bons resultados neste dataset, comparado com os métodos da Tabela 13.

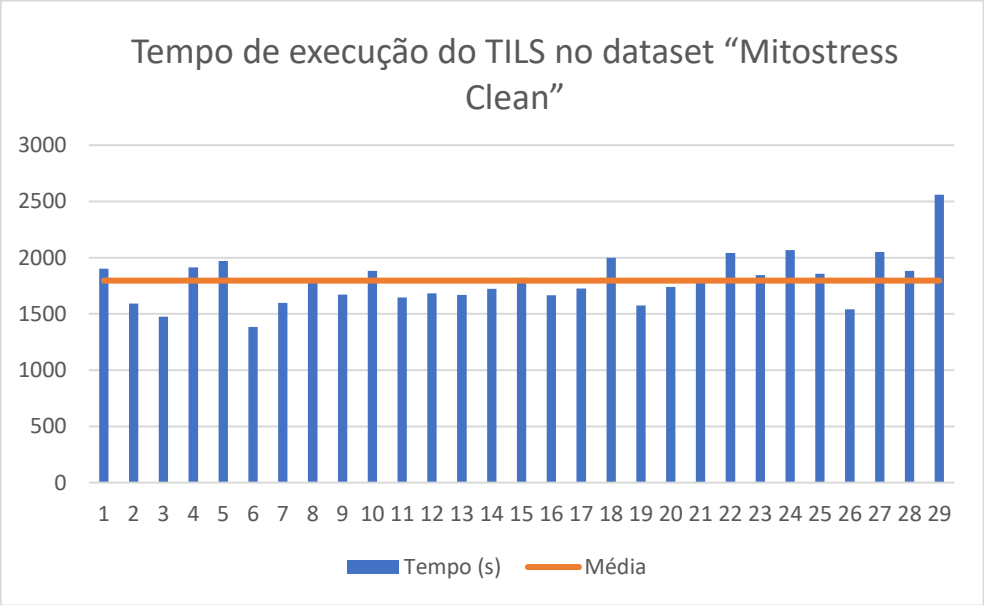

Gráfico 18 - Tempo de Execução do TILS no dataset “Mitostress Clean”

No Gráfico 18 o eixo vertical diz respeito ao tempo de execução em segundos do TILS e o eixo horizontal diz respeito ao número da repetição do método. A linha horizontal diz respeito à média das 30 repetições, tendo esta o valor de 1796 segundos. O tempo de execução foi menor da iteração 6 com o valor de 1384 segundos e o valor máximo foi de 2560 na repetição 29. Comparado com a Tabela 13 o tempo de execução do TILS é bastante maior que a maioria dos métodos, exceto o TPOT que demora mais tempo que o TILS. A média de tempo de execução do TILS é de 1796 segundos, bastante superior à maioria dos métodos.

Depois de analisada a performance do TILS e o seu tempo de execução é importante avaliar também o subconjunto de features gerado pelo método. Para isso criamos vários gráficos de forma a perceber melhor os resultados obtidos. O Gráfico 19 mostra a relação entre o número de features inicias e o número de features finais.

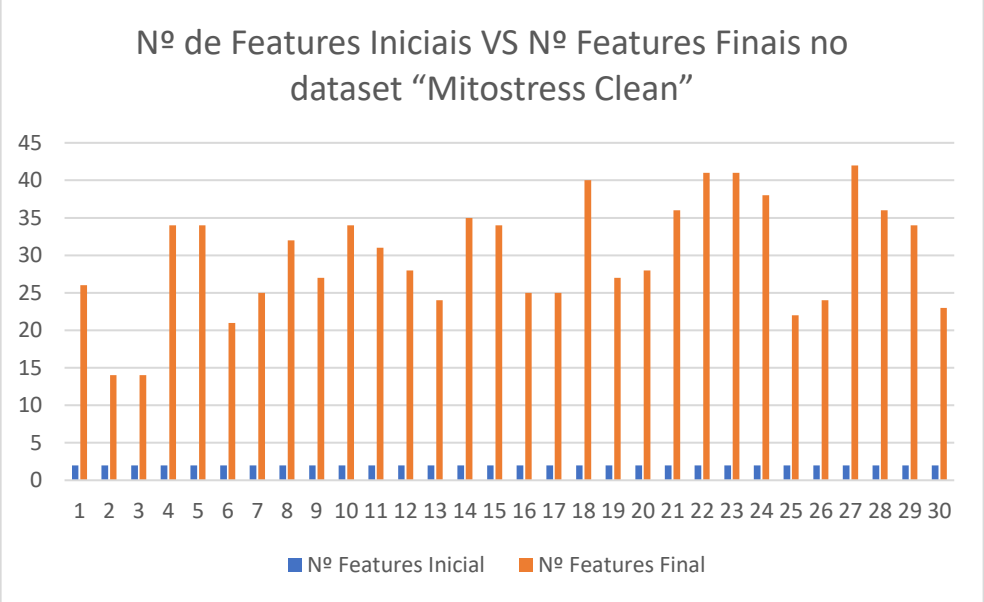

Gráfico 19 - Nº de Features Iniciais VS Nº Features Finais no dataset "Mitostress Clean"

No eixo vertical temos o número de features e no eixo horizontal temos o número de repetições do TILS. O número de features inicial é sempre igual a 2 porque começamos sempre com 20% de features do dataset inicial. O número de features final é que varia consoante as repetições. O número máximo de features final foi de 42 e diz respeito à repetição 27, neste caso temos uma diferença de 40 features. A iteração 2 e 3 são aquelas que apresentam um número inferior de features finais, apenas 14. A média de features finais é de 30, ou seja, um aumento médio de 1400%

O Gráfico 20 mostra o resultado detalhado das operações matemáticas referentes às operações com duas features.

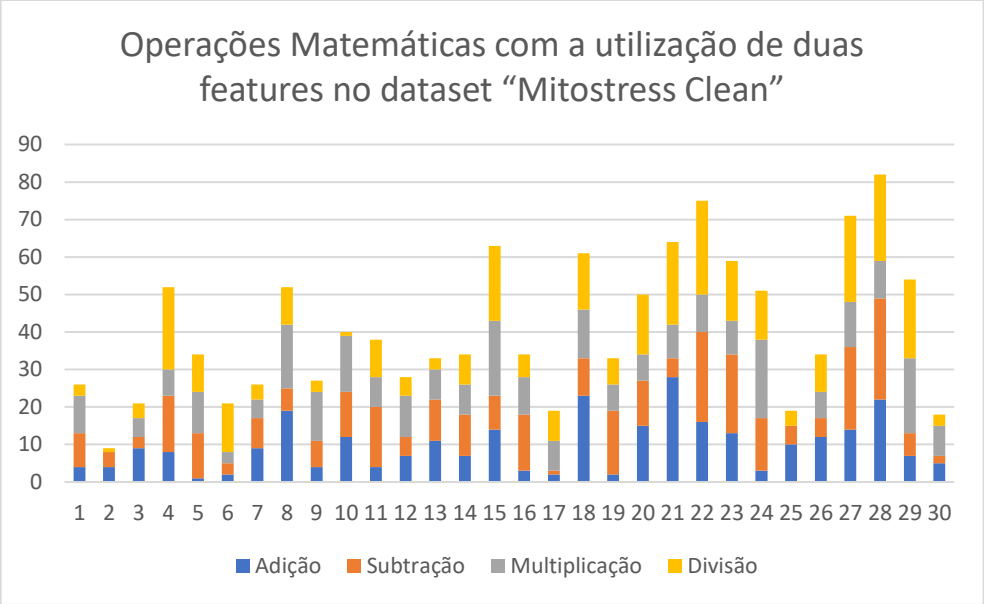

Gráfico 20 - Operações Matemáticas com a Utilização de duas features no dataset “Mitostress Clean”

No eixo vertical temos o número de operações e no eixo horizontal temos o número de repetições. Pela análise do Gráfico 20 rapidamente se identifica que a repetição 28 foi a que teve mais operações matemáticas com duas features e a repetição 2 a que teve menos. A média de operações é de 40. A cor azul representa o número de operações referentes à soma de duas features, sendo a sua média de 10. A repetição com maior número é a 21 com 28 operações de adição. A repetição com menor número de adições foi a 5 com apenas uma. A cor laranja representa o número de operações referentes a operações de subtração de duas features, sendo a sua média de 11. O valor máximo foi registado na iteração 28 com o valor de 27. O valor mínimo foi registado na iteração 27 com o valor de 1. A cor cinza diz respeito ao número de operações referentes à operação de multiplicação, sendo a sua média de 10. O maior valor foi registado na repetição 24 com o valor de 21. O valor mínimo foi registado na iteração 2 e na 25 com o valor de 0. Por fim, temos o amarelo que diz respeito às operações de divisão, tendo estas uma média de 11. O número máximo de divisões foi registado na iteração 22 com o valor de 25. Na repetição 2 e 10 foi registado o valor mínimo de 0.

Depois de analisadas as operações matemáticas com a utilização de duas features vamos agora analisar as operações matemáticas com a utilização de apenas uma feature. O Gráfico 21 mostra os resultados obtidos.

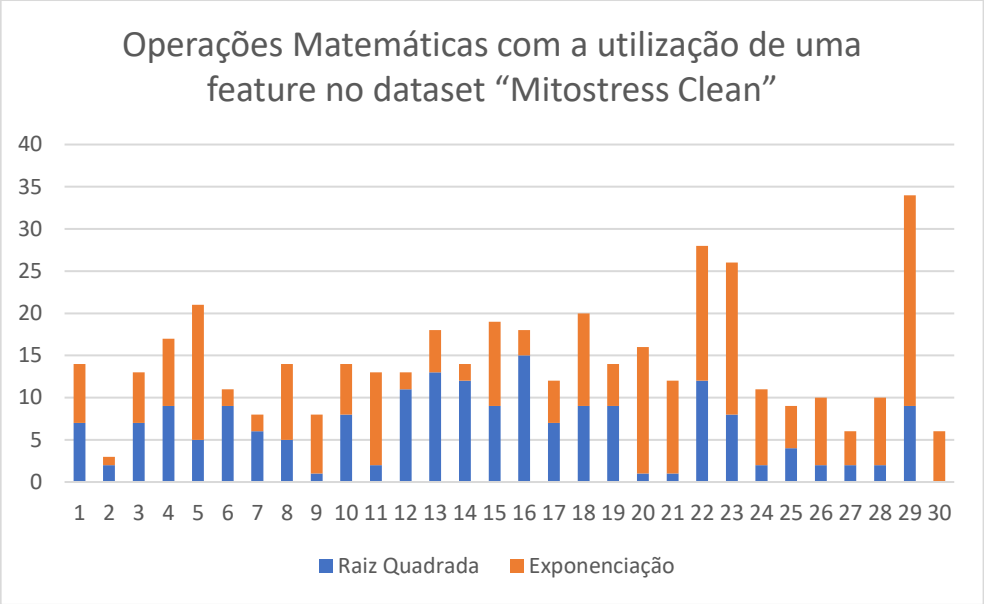

Gráfico 21 - Operações Matemáticas com a utilização de uma feature no dataset “Mitostress Clean”

No eixo vertical temos o número de operações e no eixo horizontal temos o número da repetição. A média de operações foi de 14. A iteração 29 foi a que obteve o maior número de operações com o valor de 34. O menor valor de operações foi registado na iteração 2 com o valor de 3. A cor azul diz respeito ao número de operações referentes à aplicação da raiz quadrada, sendo a sua média de 6. O valor máximo foi registado na iteração 16 com 15 operações. O valor mínimo foi obtido na iteração 30 com nenhuma utilização. A cor laranja representa o número de operações referentes à operação de exponenciação, sendo a sua média de 8. O número máximo foi de 25 registado na iteração 29. O valor mínimo foi de 1 e foi registado na iteração 2.

Depois de analisado os valores referentes às operações matemáticas com a utilização de apenas uma feature vamos analisar as operações restantes que são as não matemáticas. O Gráfico 22 mostra os resultados obtidos.

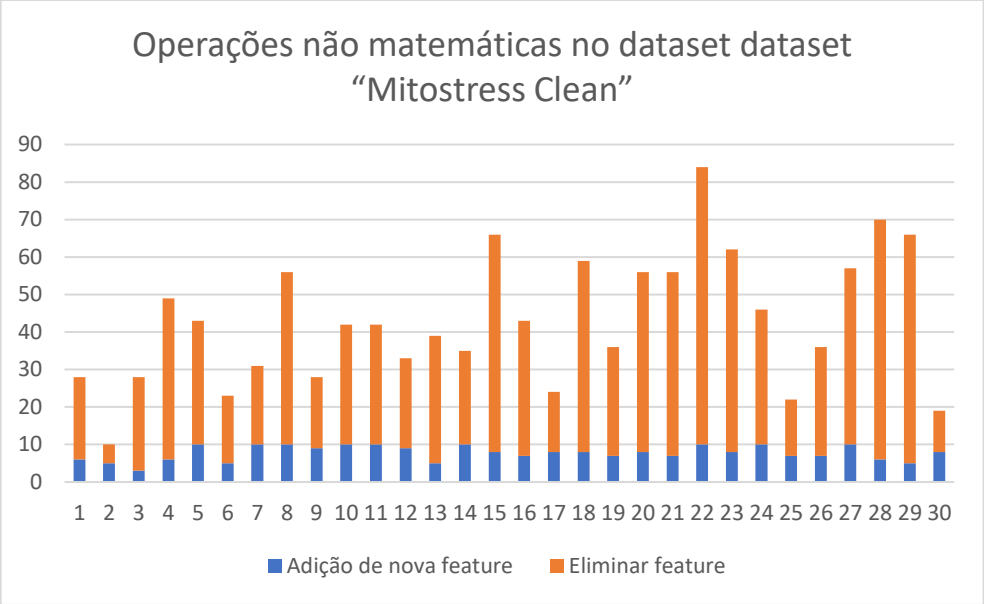

Gráfico 22 - Operações não matemáticas no dataset "Mitostress Clean"

No eixo vertical temos o número de operações e no eixo horizontal temos o número da repetição. A média de operações foi de 43. O número de máximo de operações não matemáticas foi registado na repetição 22 com o valor de 84 e o menor valor foi registado na iteração 2 com o valor de 10. A cor azul representa a adição de uma feature nova ao dataset, sendo a sua média de 8. O valor máximo registado foi de 10 na iteração 5,7,8,10,11,14,22,24 e 27. O valor mínimo foi de 3 na iteração 3. A cor laranja representa a operação de eliminação de uma feature e a sua média foi de 35. O valor máximo registado foi de 74 na repetição 22 e o valor mínimo foi de 5 na iteração 2.

Analisados todos os conjuntos de operações possíveis, decidimos criar um novo gráfico com todas as operações possíveis de forma a ter uma visão geral de todas as operações. O Gráfico 23 mostra o resultado.

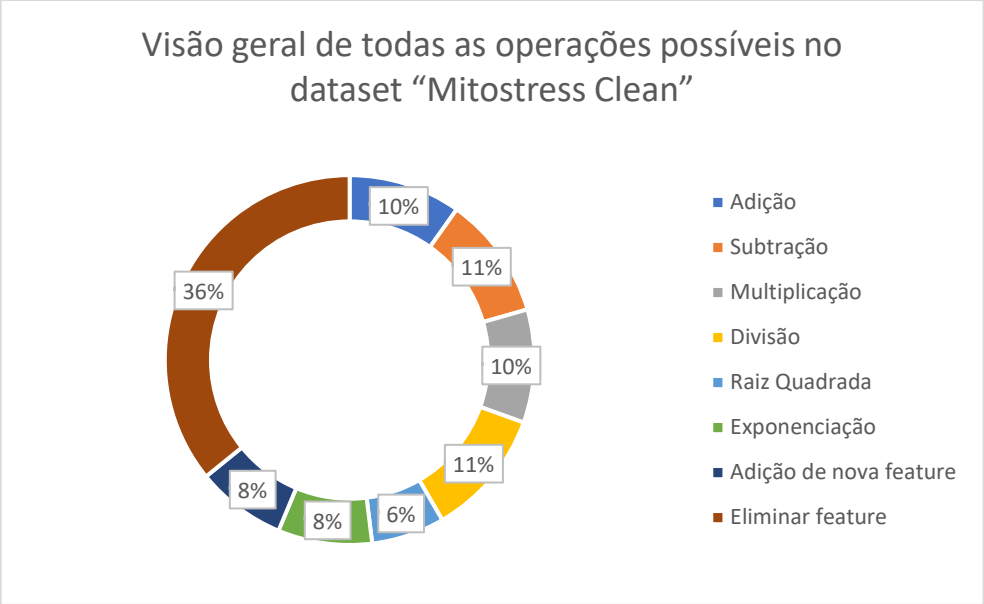

Gráfico 23 - Visão geral de todas as operações possíveis no dataset “Mitostress Clean”

Pela análise do Gráfico 23 a operação de eliminação de feature é a mais usada pelo algoritmo e a operação da raiz quadrada é a menos utilizada. As restantes operações são uniformes. Podemos concluir que a operação de eliminação de features é aquela que traz maior benefício no nosso algoritmo e que a operação de raiz quadrada é que traz menor benéfico. As operações matemáticas com uma ou duas features são as mais uniformes e são usadas de forma semelhante no algoritmo. Podemos concluir que as operações matemáticas são bons operadores para a transformação de features. A operação de eliminação acabou por ser mais relevante no nosso algoritmo porque com a adição de perturbação e quando a operação de eliminação era escolhida trazia benefícios em questão de performance ao nosso modelo, como optamos por usar o fator de tendência a operação voltava a ser utilizada até deixar de trazer benefício. Dai os valores de eliminação de feature serem maiores.

6.4 Dataset “ATP Rate Clean”

De forma a ter um termo de comparação, aplicamos isoladamente cada método ao dataset e registamos os resultados na Tabela 14. No “Anexo C – Ficheiros relativos ao dataset Amyotrophic Lateral Sclerosis (ALS)” encontra-se os ficheiros python implementados nesta secção.

Tabela 14 - Resultado da aplicação dos métodos ao dataset "ATP Rate Clean"

Método	Nº Features Originais	Nº Features Novas	Δ t (s) Seleção de features	Δ t (s) LOOCV	Accuracy
-	4	0	-	2,70	0,55
Qui-Quadrado	4	0	0,17	3,24	0,56
Eliminação Recursiva de Features	2	0	0,20	2,78	0,54
Regressão de Ridge	1	0	0,22	2,73	0,46
Regressão de Lasso	2	0	2,89	2,72	0,51
Ganho de Informação	2	0	0,10	2,75	0,53
Autofeat	4	4	52,75	5,12	0,55
TPOT	2	0	12460,83	2,80	0,52
H20 AutoML	2	0	1154,55	2,76	0,54
Featuretools	4	6	0,16	4,98	0,53

A Tabela 14 é composta por 6 colunas, a primeira coluna diz respeito ao método utilizado. A segunda coluna diz respeito ao número de features originais resultantes da aplicação do método. A terceira coluna indica o número de features criadas pelo método. A quarta coluna temos o tempo em segundos que o método demorou a ser executado. Na quinta coluna temos o tempo de execução da avaliação do modelo. Por último, na sexta coluna temos a accuracy do método.

Pela análise da Tabela 14 podemos reparar que maior parte dos métodos demoram menos de 1 segundo no processo de seleção de features. O método que demorou mais tempo foi o TPOT com cerca de 12460 segundos, logo depois temos o H2O AutoML com cerca de 1154 segundos. O método mais rápido foi a Eliminação Recursiva de Features com menos de meio segundo. Em geral o tempo de execução dos métodos são baixos o que indica que são rápidos a executar, este fator também se deve ao número reduzido de features. No que toca ao tempo de execução do modelo os valores são muito idênticos, rondam os 3 segundos em média, portanto o tempo de execução é estável. Avaliando a accuracy dos vários métodos também não temos grandes variações, sendo a média de 53%. O valor mais alto foi do Qui-Quadrado com 56% e o valor mais baixo foi de 46% referente à Regressão de Ridge.

Outra conclusão que podemos retirar é que os métodos tradicionais obtiveram performances idênticas à exceção da regressão de ridge que obteve a pior performance, o que pode indicar que este método não funciona bem em datasets com um número reduzido de features. No que toca às frameworks os resultados foram todos idênticos, mesmo as frameworks que utilizaram novas features a performance não foi muito melhor do que aquelas que só utilizaram as features originais. Não obtivemos nenhum registo com melhores valores do que o registo inicial sem a aplicação de nenhum método.

6.5 Resultados da aplicação do TILS

Nesta secção tivemos de alterar os valores da Tabela 8 porque o dataset só possui 4 features e o TILS necessita de um mínimo de 2 features para funcionar, optamos por alterar o valor de "Número de Features Iniciais" de 20% para 50%. Com o valor original de 20% de features iniciais o TILS não tem como selecionar 2 features para realizar uma operação. Como o dataset só tem 4 features e o número mínimo de features é de 2, então 50% é o valor mínimo para o parâmetro "Número de Features Iniciais".

De forma a obter uma melhor perceção dos resultados optamos por correr o TILS 30 vezes e por cada vez que o método corria, os valores eram apontados numa tabela, sendo o Gráfico 24 o resultado desses registos.

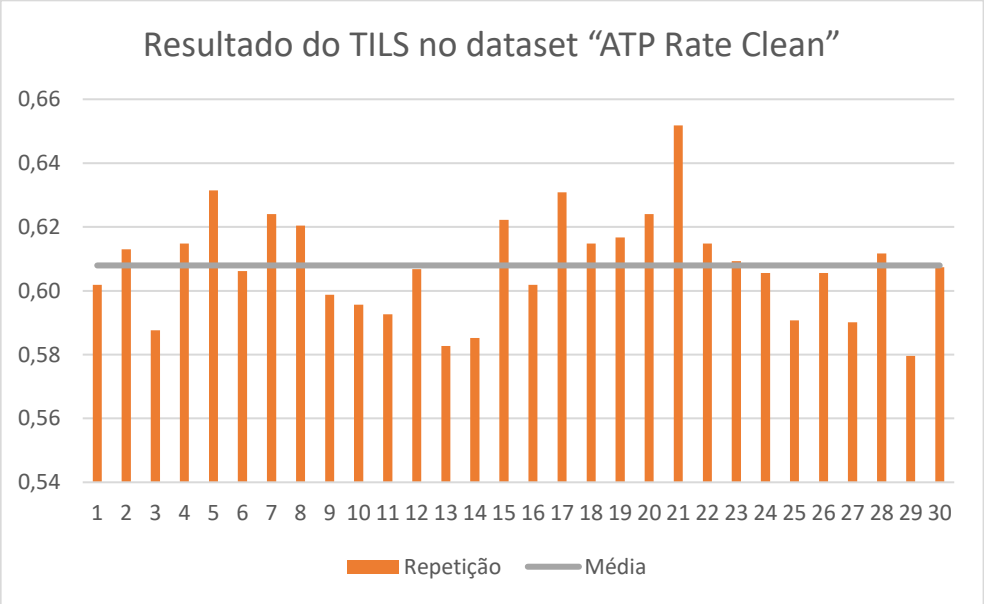

Gráfico 24 - Resultado do TILS no dataset "ATP Rate Clean"

No Gráfico 24 o eixo vertical diz respeito à accuracy obtida pelo algoritmo e o eixo horizontal diz respeito ao número da repetição do algoritmo. A linha horizontal diz respeito à média das 30 repetições, tendo esta o valor de 61%. O melhor resultado foi de 65% na repetição 21 e o resultado pior foi na iteração 29 com 58%. Comparando estes resultados com a Tabela 14, o pior resultado do Gráfico 24 é superior ao melhor resultado da Tabela 14. Estes resultados indicam que o TILS é um método com bons resultados neste dataset, comparado com os métodos da Tabela 14.

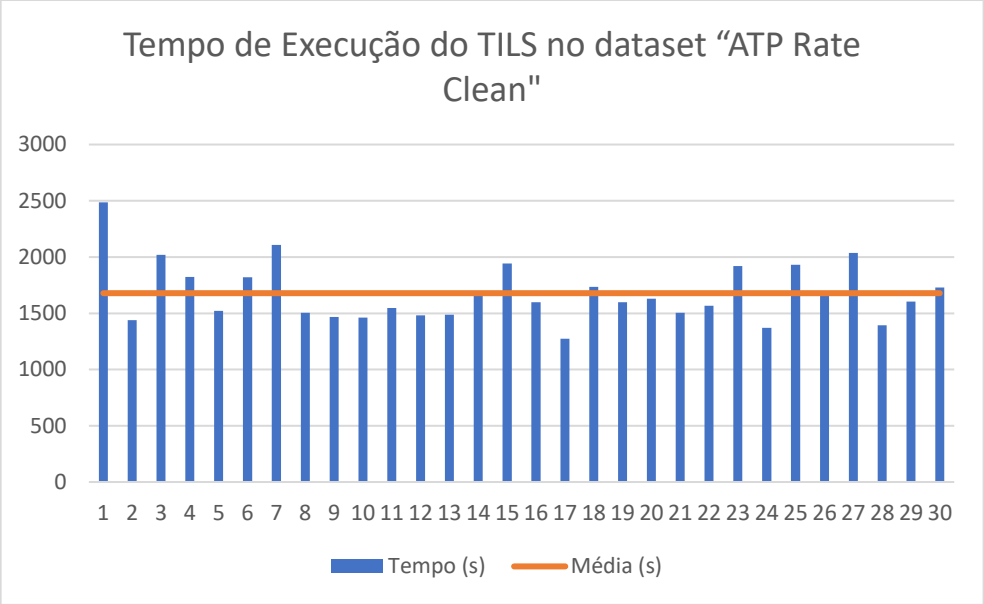

Gráfico 25 - Tempo de Execução do TILS no dataset “ATP Rate Clean”

No Gráfico 25 o eixo vertical diz respeito ao tempo de execução em segundos do TILS e o eixo horizontal diz respeito ao número da repetição do método. A linha horizontal diz respeito à média das 30 repetições, tendo esta o valor de 1679 segundos. O tempo de execução foi menor da iteração 17 com o valor de 1273 segundos e o valor máximo foi de 2485 na repetição 1. Comparado com a Tabela 14 o tempo de execução do TILS é bastante maior que a maioria dos métodos, exceto o TPOT que demora mais tempo que o TILS. A média de tempo de execução do TILS é de 1679 segundos, bastante superior à maioria dos métodos.

Depois de analisada a performance do TILS e o seu tempo de execução é importante avaliar também o subconjunto de features gerado pelo método. Para isso criamos vários gráficos de forma a perceber melhor os resultados obtidos. O Gráfico 26 mostra a relação entre o número de features inicias e o número de features finais.

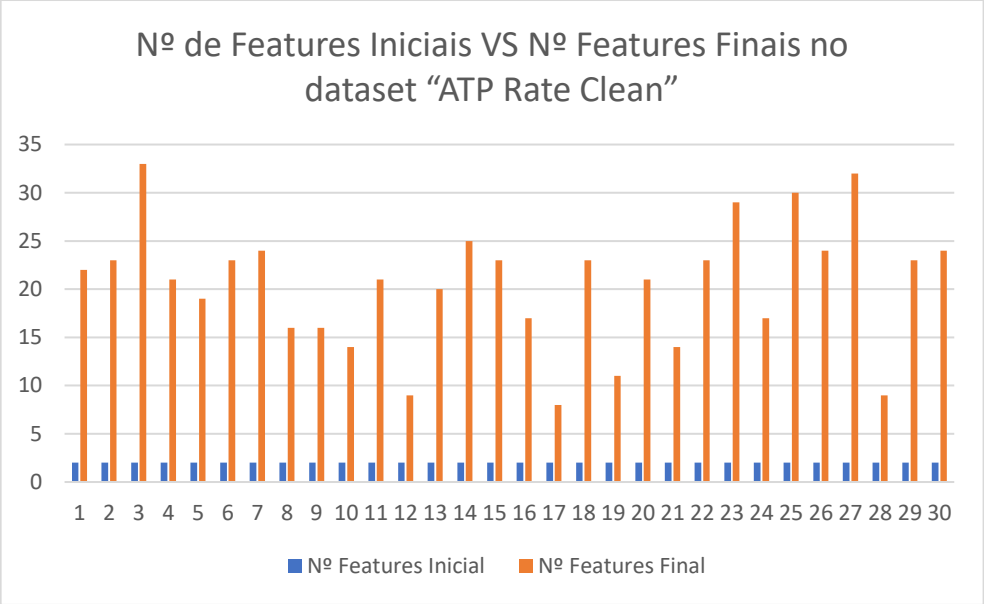

Gráfico 26 - Nº de Features Iniciais VS Nº Features Finais no dataset “ATP Rate Clean”

No eixo vertical temos o número de features e no eixo horizontal temos o número de repetições do TILS. O número de features inicial é sempre igual a 2 porque começamos sempre com 50% de features do dataset inicial. O número de features final é que varia consoante as repetições. O número máximo de features final foi de 33 e diz respeito à repetição 3, neste caso temos uma diferença de 31 features. A iteração 8 é aquela que apresenta o número inferior de features finais, apenas 8. A média de features finais é de 20, ou seja, um aumento médio de 1000%

O Gráfico 27 mostra o resultado detalhado das operações matemáticas referentes às operações com duas features.

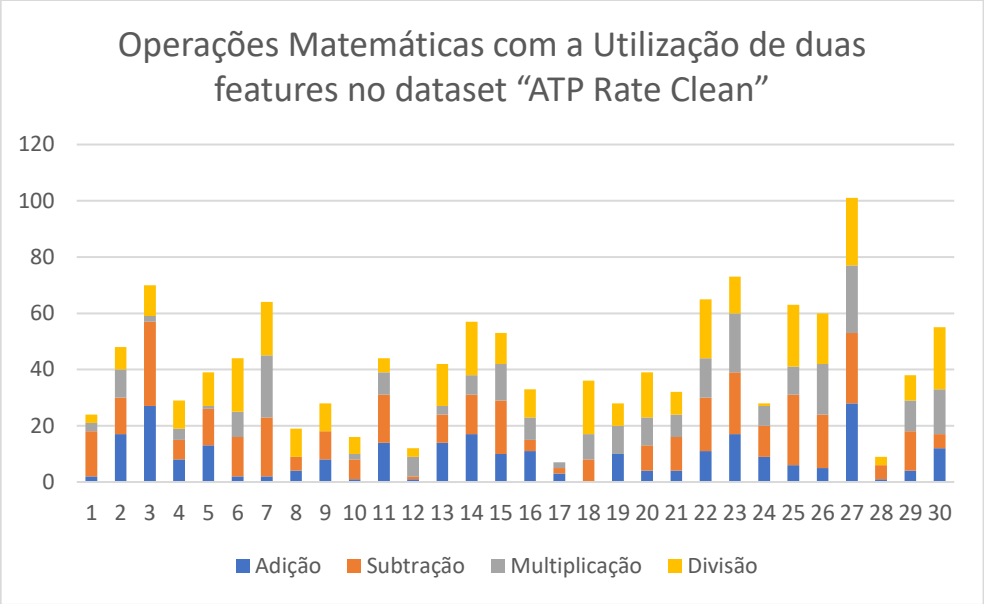

Gráfico 27 - Operações Matemáticas com a Utilização de duas features no dataset "ATP Rate Clean"

No eixo vertical temos o número de operações e no eixo horizontal temos o número de repetições. Pela análise do Gráfico 27 rapidamente se identifica que a repetição 27 foi a que teve mais operações matemáticas com duas features e a repetição 17 a que teve menos. A média de operações é de 41. A cor azul representa o número de operações referentes à soma de duas features, sendo a sua média de 8. A repetição com maior número é a 27 com 28 operações de adição. A repetição com menor número de adições foi a 18 com zero. A cor laranja representa o número de operações referentes a operações de subtração de duas features, sendo a sua média de 12. O valor máximo foi registado na iteração 3 com o valor de 30. O valor mínimo foi registado na iteração 19 com o valor de zero. A cor cinza diz respeito ao número de operações referentes à operação de multiplicação, sendo a sua média de 8. O maior valor foi registado na repetição 27 com o valor de 24. O valor mínimo foi registado na iteração 8,9 e 28 com o valor de zero. Por fim, temos o amarelo que diz respeito às operações de divisão, tendo estas uma média de 11. O número máximo de divisões foi registado na iteração 27 com o valor de 24. Na repetição 17 foi registado o valor mínimo de zero.

Depois de analisadas as operações matemáticas com a utilização de duas features vamos agora analisar as operações matemáticas com a utilização de apenas uma feature. O Gráfico 21 mostra os resultados obtidos.

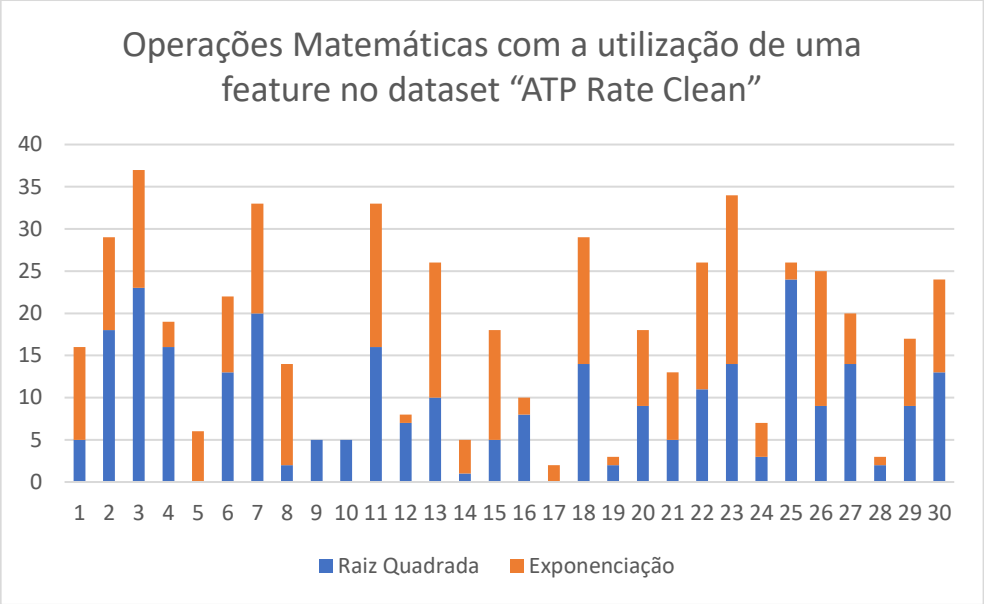

Gráfico 28 - Operações Matemáticas com a utilização de uma feature no dataset "ATP Rate Clean"

No eixo vertical temos o número de operações e no eixo horizontal temos o número da repetição. A média de operações foi de 18. A iteração 3 foi a que obteve o maior número de operações com o valor de 37. O menor valor de operações foi registado na iteração 17 com o valor de 2. A cor azul diz respeito ao número de operações referentes à aplicação da raiz quadrada, sendo a sua média de 9. O valor máximo foi registado na iteração 25 com 24 operações. O valor mínimo foi obtido na iteração 5 e 17 com nenhuma utilização. A cor laranja representa o número de operações referentes à operação de exponenciação, sendo a sua média de 8. O número máximo foi de 20 registado na iteração 23. O valor mínimo foi de 0 e foi registado na iteração 9 e 10.

Depois de analisado os valores referentes às operações matemáticas com a utilização de apenas uma feature vamos analisar as operações restantes que são as não matemáticas. O Gráfico 29 mostra os resultados obtidos.

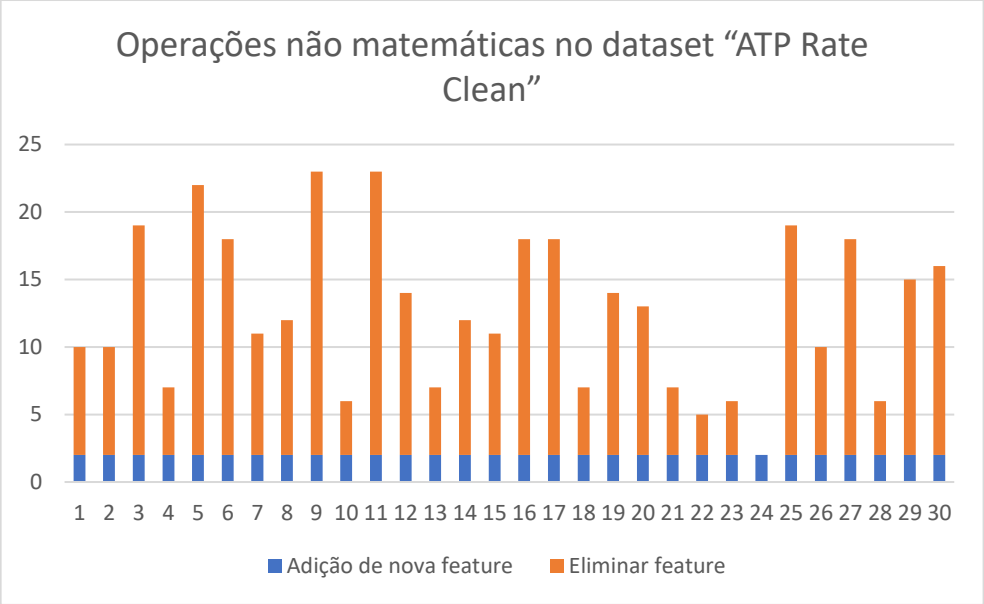

Gráfico 29 - Operações não matemáticas no dataset “ATP Rate Clean”

No eixo vertical temos o número de operações e no eixo horizontal temos o número da repetição. A média de operações foi de 12. O número de máximo de operações não matemáticas foi registado na repetição 9 e 11 com o valor de 23 e o menor valor foi registado na iteração 24 com o valor de 2. A cor azul representa a adição de uma feature nova ao dataset, sendo a sua média de 2. Este valor foi sempre constante porque o número de features disponível era reduzido. A cor laranja representa a operação de eliminação de uma feature e a sua média foi de 11. O valor máximo registado foi de 21 na repetição 9 e 11. O valor mínimo foi de 0 na iteração 24.

Analisados todos os conjuntos de operações possíveis, decidimos criar um novo gráfico com todas as operações possíveis de forma a ter uma visão geral de todas as operações. O Gráfico 30 mostra o resultado.

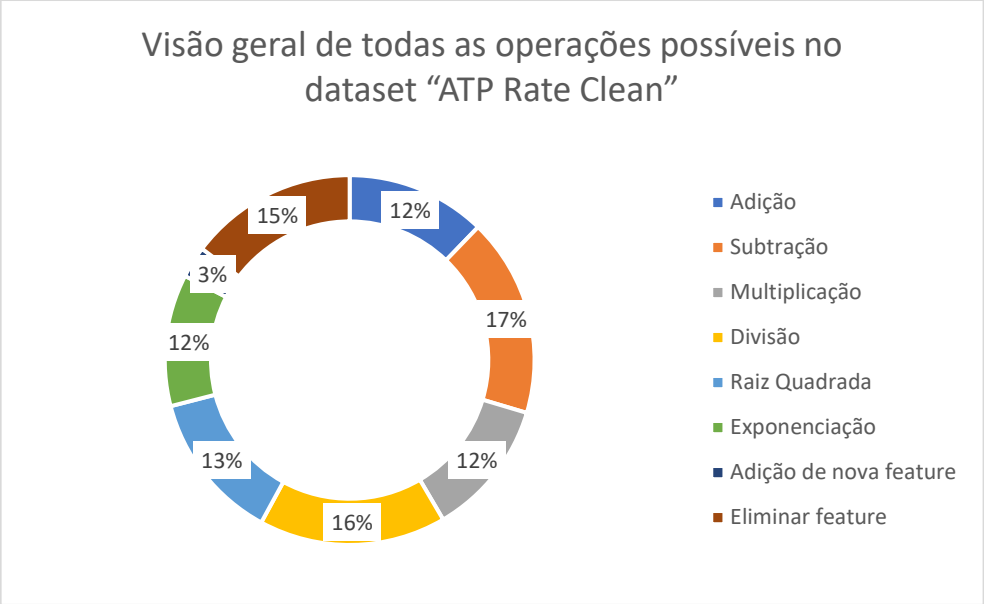

Gráfico 30 - Visão geral de todas as operações possíveis no dataset "ATP Rate Clean"

Pela análise do Gráfico 30 a operação de subtração de feature é a mais usada pelo algoritmo e a operação adicionar nova feature é a menos utilizada. As restantes operações são uniformes. Podemos concluir que a operação de subtração de features é aquela que traz maior benefício no nosso algoritmo e que a de adicionar nova feature é que traz menor benefício. As operações matemáticas com uma ou duas features são as mais uniformes e são usadas de forma semelhante no algoritmo. Podemos concluir que as operações matemáticas são bons operadores para a transformação de features. Neste dataset como o número de features era reduzido a operação de adicionar nova feature acaba por sempre igual porque em relação à percentagem inicial de features o algoritmo foi buscar sempre mais 2 features, que por sua vez eram as restantes.

7 Conclusões

O trabalho descrito neste documento resultou na criação de um método para engenharia de features – TILS, com o objetivo de obter um subconjunto de features constituído por algumas originais e outras que resultam de combinações ou transformações. Este método foi aplicado no dataset “MicroRNA” como caso de teste e comparado com os métodos escolhidos, por fim foi validado com o dataset “Amyotrophic Lateral Sclerosis (ALS)”.

O TILS é um método de PL baseado no TC e no ILS que pode ser aplicado/adaptado a qualquer problema com dados tabulares numéricos. A utilização de operações de transformação de features e a utilização do fator de tendência na escolha da operação fazem do TILS um método capaz de evoluir e trazer benefícios na sua utilização.

7.1 Principais Resultados

A aplicação do TILS no dataset “MicroRNA” foi benéfica em comparação com as restantes abordagens. A média do método desenvolvido foi de 90% e a média dos métodos testado foi de 78%. Neste dataset o TILS chegou a obter 97% de accuracy, um resultado bastante favorável face aos restantes métodos. Apesar de ser um método mais lento que a maioria dos métodos testados é um método que traz melhores resultados. O motivo deste método ser mais lento é a avaliação do modelo a cada iteração para comparar se a operação aplicada traz benefício ou não. A média de tempo de execução é de 449 segundos, ficando só atrás do TPOT e do H2O AutoML, os restantes métodos são todos mais rápidos. O TILS poderia ser parametrizado para demorar menos tempo a ser executado se o “Critério de Terminação” fosse ajustado, mas este ajuste poderia ser prejudicial para a sua performance e só benéfico no tempo de execução.

A aplicação do TILS ao dataset principal “Mitostress Clean” mostrou ser bastante benéfico, a média de accuracy foi de 73% comparado com a média dos restantes métodos que foi de 62%. Estes resultados indicam que o TILS é um método com bons resultados neste dataset. No que toca ao tempo de execução o TILS é dos métodos que demora mais tempo a ser executado, em média cerca de 1796 segundos, enquanto nos restantes métodos a maioria demora apenas alguns segundos. O TPOT e o H2O AutoML tambem demoram algum tempo a executar, mas a sua performance comparada com o TILS é inferior.

A aplicação do TILS ao dataset principal “ATP Rate Clean” também mostrou ser benéfico, a média de accuracy foi de 61% comparado com a média dos restantes métodos que foi de 52%. Estes resultados indicam que o TILS é um método com bons resultados neste dataset mesmo sendo um dataset com poucas features, apenas 4. No que toca ao tempo de execução o TILS é novamente dos métodos que demoram

mais tempo a ser executado, em média cerca de 1679 segundos. Novamente o TPOT e o H2O AutoML apresentam tempos semelhantes em relação ao TILS, mas mesmo assim o TILS consegui melhores valores.

O TILS quando aplicado a um dataset mais pequeno, com poucas features, é tendencialmente mais fraco porque tem menos opções de escolha nas features que pode usar para fazer operações, acabando por utilizar as mesmas features vezes sem conta na criação de novas features. Sendo o resultado das novas features operações com ela própria, vezes sem conta. Quando o dataset já possui mais features, a aleatoriedade na escolha das features para realizar operações é maior, mostrando que no resultado final as operações utilizando a mesma features vezes sem conta acontecem muito menos.

O TILS perde em relação à maioria dos métodos no tempo de execução, é um método lento que a cada iteração avalia o modelo, se a operação escolhida aleatoriamente for benéfica para o modelo é utilizada, caso contrário é descartada. A aplicação da perturbação também é mais um fator com impacto negativo no tempo de execução porque depois de adicionada a perturbação o conjunto de features volta a ser avaliado e comparado com o anterior, se for benéfico utilizamos esse subconjunto de features, caso contrário descartamos a perturbação.

O TILS revelou que consegue obter bons resultados em relação aos métodos escolhidos nos dois datasets utilizados, desta forma a aplicação do TILS é benéfica para o grupo de investigação MitoXT. Com a aplicação do TILS o grupo de investigação MitoXT pode obter bons resultados.

7.2 Trabalho futuro

Como trabalho futuro é possível melhorar muito o método desenvolvido, nomeadamente o tempo de execução. Este pode ser reduzido através de "Multithreading" ou até "clustering". Outra melhoria que pode ser feita é a expansão de operações possíveis de transformação, por exemplo $\log x$, $\frac{1}{x}$, x^2, x^3, $|x|$, $2x$, $\sin x$, $\cos x$. Outra melhoria que pode ser feita é a utilização de novas métricas para identificar se as features são benéficas ou não, como por exemplo correlação. Outra melhoria é o TILS efetuar operações de pré-processamento, ou seja, avaliar o dataset inicialmente, antes de qualquer manipulação e remover features com valores em falta ou remover features correlacionas. Outra melhoria que pode ser aplicada ao TILS é a evolução para uma framework, onde o utilizador pode facilmente com a afinação de parâmetros configurar o funcionamento do TILS. Outra sugestão de melhoria é a escolha do modelo a utilizar, neste caso optamos por usar sempre uma árvore de decisão, mas pode-se usar outro tipo de algoritmo na construção do modelo, como por exemplo: redes neuronais, regressões, KNN, Naïve Bayes, entre outros. Outra melhoria também seria a possibilidade de escolha da métrica de avaliação do

modelo, neste caso optamos por usar sempre a accuracy, mas podemos utilizar a precisão, a medida de F1, curva ROC, etc.

Por último, o desenvolvimento deste trabalho no âmbito do MIS permitiu-me aprofundar conhecimentos em áreas nas quais tinha algum interesse. Foi com muito agrado que consegui aprofundar as minhas capacidades de investigação e de aplicação dos resultados obtidos a um caso de estudo concreto. Foi ainda no âmbito do desenvolvimento deste projeto que tive oportunidade de ajudar o grupo de investigação MitoXT – uma experiência muito positiva que me permitiu sair da minha área de conforto em vários momentos e onde foi possível fazer uma partilha de ideias com especialistas de várias áreas de forma a evoluir no sentido correto. Em suma, foi uma experiência muito desafiante e exigente, no entanto, também foi muito enriquecedora em todas as vertentes.

Referências Bibliográficas

Abdulkareem, N. M., & Abdulazeez, A. M. (2021). Machine Learning Classification Based on Radom Forest Algorithm: A Review. *International Journal of Science and Business*, *5*(2), 128–142.

AMA – Agência para a Modernização Administrativa. (2022). *Projeto Portugal 2020 Mito4ALS .: Desenvolvimento de novos antioxidantes dirigidos para as mitocôndrias na melhoria do fenótipo da Esclerose Lateral Amiotrófica familiar SOD1 (POCI-01-0145-FEDER-029391; PTDC/MED-FAR/29391/2017)*. https://transparencia.gov.pt/pt/fundos-europeus/beneficiarios-projetos/projeto/POCI-01-0145-FEDER-029391#project_information_id

Ampomah, E. K., Nyame, G., Qin, Z., Addo, P. C., Gyamfi, E. O., & Gyan, M. (2021). Stock Market Prediction with Gaussian Naïve Bayes Machine Learning Algorithm. *Informatica*, *45*(2). https://doi.org/10.31449/inf.v45i2.3407

Anaconda. (2021). Anaconda. https://www.anaconda.com/

Apela. (2022). *Apela—Associação Portuguesa de Esclerose Lateral Amiotrófica*. Apela - Associação Portuguesa de Esclerose Lateral Amiotrófica. //www.apela.pt/

AutoML: Automatic Machine Learning—H2O 3.36.1.1 documentation. (2022). https://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html

Banerjee, S., Chattopadhyay, T., Biswas, S., Banerjee, R., Choudhury, A. D., Pal, A., & Garain, U. (2016). *Towards Wide Learning: Experiments in Healthcare*. https://doi.org/10.48550/ARXIV.1612.05730

Centro de Neurociências e Biologia Celular. (2020). Centro de Neurociências e Biologia Celular. https://cnc.uc.pt/pt

Charbuty, B., & Abdulazeez, A. (2021). Classification Based on Decision Tree Algorithm for Machine Learning. *Journal of Applied Science and Technology Trends*, *2*(01), 20–28. https://doi.org/10.38094/jastt20165

Chaturvedi, N. (2021, maio 10). *Leave-One-Out Cross-Validation*. Medium. https://medium.datadriveninvestor.com/leave-one-out-cross-validation-32fa248c1739

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, *55*(10), 78–87. https://doi.org/10.1145/2347736.2347755

Donnelly, C. J., Zhang, P.-W., Pham, J. T., Haeusler, A. R., Mistry, N. A., Vidensky, S., Daley, E. L., Poth, E. M., Hoover, B., Fines, D. M., Maragakis, N., Tienari, P. J., Petrucelli, L., Traynor, B. J., Wang, J., Rigo, F., Bennett, C. F., Blackshaw, S., Sattler, R., & Rothstein, J. D. (2013). RNA Toxicity from the ALS/FTD C9ORF72 Expansion Is Mitigated by Antisense Intervention. *Neuron*, *80*(2), 415–428. https://doi.org/10.1016/j.neuron.2013.10.015

Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least angle regression. *The Annals of Statistics*, *32*(2). https://doi.org/10.1214/009053604000000067

Elgeldawi, E., Sayed, A., Galal, A. R., & Zaki, A. M. (2021). Hyperparameter Tuning for Machine Learning Algorithms Used for Arabic Sentiment Analysis. *Informatics*, *8*(4), 79. https://doi.org/10.3390/informatics8040079

Esclerose lateral amiotrófica | CUF. (2021). https://www.cuf.pt/saude-a-z/esclerose-lateral-amiotrofica

Featuretools. (2022). https://www.featuretools.com/

Fontes, J. de A. (2020). *Abordagens de seleção de variáveis para classificação e regressão em dados espectrais para controle da qualidade*. https://lume.ufrgs.br/handle/10183/211247

Fuhr, G. T. (2022). *Uso de machine learning para a classificação do crédito de empresas por meio de demonstrativos financeiros*. http://repositorio.ufsm.br/handle/1/24624

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems* (Second edition). O'Reilly Media, Inc.

Ghosh, S., & Dasgupta, R. (2022). Model Selection for Machine Learning. Em S. Ghosh & R. Dasgupta, *Machine Learning in Biological Sciences* (pp. 51–57). Springer Nature Singapore. https://doi.org/10.1007/978-981-16-8881-2_5

Hamet, P., & Tremblay, J. (2017). Artificial intelligence in medicine. *Metabolism*, *69*, S36–S40. https://doi.org/10.1016/j.metabol.2017.01.011

Hoerl, A. E., & Kennard, R. W. (1970). Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, *12*(1), 55–67. https://doi.org/10.1080/00401706.1970.10488634

Hoos, H. H., & Stützle, T. (2005). *Stochastic local search: Foundations and applications*. Morgan Kaufmann Publishers.

Horn, F., Pack, R., & Rieger, M. (2019). *The autofeat Python Library for Automated Feature Engineering and Selection*. https://doi.org/10.48550/ARXIV.1901.07329

Itoo, F., Meenakshi, & Singh, S. (2021). Comparison and analysis of logistic regression, Naïve Bayes and KNN machine learning algorithms for credit card fraud detection. *International Journal of Information Technology*, *13*(4), 1503–1511. https://doi.org/10.1007/s41870-020-00430-y

Jadhav, S., He, H., & Jenkins, K. (2018). Information gain directed genetic algorithm wrapper feature selection for credit rating. *Applied Soft Computing*, *69*, 541–553. https://doi.org/10.1016/j.asoc.2018.04.033

Le, T. T., Fu, W., & Moore, J. H. (2020). Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*, *36*(1), 250–256. https://doi.org/10.1093/bioinformatics/btz470

Least-angle regression. (2021). Em *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Least-angle_regression&oldid=1027502988

LeDell, E., & Poirier, S. (2020). *H2O AutoML: Scalable Automatic Machine Learning*. 16.

Ljubljana, B. L., University of. (2021). *Orange Data Mining Toolbox*. Orange Data Mining - Data Mining. https://orangedatamining.com/

Lourenço, H., Martin, O., & Stützle, T. (2010). Iterated Local Search: Framework and Applications. Em *Handbook of Metaheuristics* (Vol. 146, pp. 363–397). https://doi.org/10.1007/978-1-4419-1665-5_12

Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated Local Search. Em F. Glover & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics* (Vol. 57, pp. 320–353). Kluwer Academic Publishers. https://doi.org/10.1007/0-306-48056-5_11

Mahesh, B. (2019). *Machine Learning Algorithms -A Review*. https://doi.org/10.21275/ART20203995

Mantovani, R. G., Horváth, T., Cerri, R., Junior, S. B., Vanschoren, J., & de Carvalho, A. C. P. de L. F. (2018). *An empirical study on hyperparameter tuning of decision trees*. https://doi.org/10.48550/ARXIV.1812.02207

Marques, J. A. de S. (2015). *Heurísticas de pesquisa local para problemas de máquina única*. https://recipp.ipp.pt/handle/10400.22/8112

Maulud, D., & Abdulazeez, A. M. (2020). A Review on Linear Regression Comprehensive in Machine Learning. *Journal of Applied Science and Technology Trends*, *1*(4), 140–147. https://doi.org/10.38094/jastt1457

Michalewicz, Z., & Fogel, D. B. (2013). *How to Solve It: Modern Heuristics*. Springer Nature (Textbooks & Major Reference Works) : Springer.

MicroRNA. (2022). Em *Wikipedia*. https://en.wikipedia.org/w/index.php?title=MicroRNA&oldid=1080696421

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

MitoXT - Toxicologia e Terapêutica Experimental Mitocondrial. (2020). MitoXT - Toxicologia e Terapêutica Experimental Mitocondrial. https://cnc.uc.pt/en/research-group/mitoxt-toxicologia-e-terapeutica-experimental-mitocondrial

Nusinovici, S., Tham, Y. C., Chak Yan, M. Y., Wei Ting, D. S., Li, J., Sabanayagam, C., Wong, T. Y., & Cheng, C.-Y. (2020). Logistic regression was as good as machine learning for predicting major chronic diseases. *Journal of Clinical Epidemiology*, *122*, 56–69. https://doi.org/10.1016/j.jclinepi.2020.03.002

Oliveira, D. S. de. (2017). *Pesquisa da mutação C9ORF72 e de suas características clínicas nos pacientes portadores de esclerose lateral amiotrófica, demência frontotemporal e parkinsonismo atípico* [Mestrado em Doenças Crônico-Degenerativas e Imunomediadas do

Sistema Nervoso, Universidade de São Paulo]. https://doi.org/10.11606/D.17.2017.tde-30032017-135422

Olson, R. S., Bartley, N., Urbanowicz, R. J., & Moore, J. H. (2016). *Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science*. https://doi.org/10.48550/ARXIV.1603.06212

Paquete, L. (2005). Algoritmos para o problema do caixeiro viajante multiobjectivo. *Estudos II*. https://sapientia.ualg.pt/handle/10400.1/5587

Pearson, K. (1900). X. *On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling*. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *50*(302), 157–175. https://doi.org/10.1080/14786440009463897

Pelison, L. F. (2018). *Geração Automática de Features para Modelagem Preditiva—Predição de Empresas Brasileiras de Alto Crescimento* [Universidade Federal de Santa Catarina]. https://repositorio.ufsc.br/handle/123456789/200013

Podgorelec, V., Kokol, P., Stiglic, B., & Rozman, I. (2002). Decision Trees: An Overview and Their Use in Medicine. *Journal of Medical Systems*, *26*(5), 445–463. https://doi.org/10.1023/A:1016409317640

Pontes, R. T., Orsini, M., Freitas, M. R. de, Antonioli, R. de S., & Nascimento, O. J. (2001). Alterações da fonação e deglutição na Esclerose Lateral Amiotrófica. *Revista Neurociências*, *18*(1), 69–73. https://doi.org/10.34024/rnc.2010.v18.8505

Python.org. (2021). Python.Org. https://www.python.org/

Rácz, A., Bajusz, D., & Héberger, K. (2021). Effect of Dataset Size and Train/Test Split Ratios in QSAR/QSPR Multiclass Classification. *Molecules*, *26*(4), 1111. https://doi.org/10.3390/molecules26041111

Rani, P., Kumar, R., Jain, A., & Chawla, S. K. (2021). A Hybrid Approach for Feature Selection Based on Genetic Algorithm and Recursive Feature Elimination: *International Journal of Information System Modeling and Design*, *12*(2), 17–38. https://doi.org/10.4018/IJISMD.2021040102

Senan, E. M., Al-Adhaileh, M. H., Alsaade, F. W., Aldhyani, T. H. H., Alqarni, A. A., Alsharif, N., Uddin, M. I., Alahmadi, A. H., Jadhav, M. E., & Alzahrani, M. Y. (2021). Diagnosis of Chronic Kidney Disease Using Effective Classification Algorithms and Recursive Feature Elimination Techniques. *Journal of Healthcare Engineering*, *2021*, 1–10. https://doi.org/10.1155/2021/1004767

Sklearn.feature_selection.chi2. (2022). Scikit-Learn. https://scikit-learn/stable/modules/generated/sklearn.feature_selection.chi2.html

Sklearn.feature_selection.RFE. (2022). Scikit-Learn. https://scikit-learn/stable/modules/generated/sklearn.feature_selection.RFE.html

Sklearn.linear_model.Lasso. (2022). Scikit-Learn. https://scikit-learn/stable/modules/generated/sklearn.linear_model.Lasso.html

Sklearn.linear_model.Ridge. (2022). Scikit-Learn. https://scikit-learn/stable/modules/generated/sklearn.linear_model.Ridge.html

Sklearn.model_selection.cross_val_predict. (2022). Scikit-Learn. https://scikit-learn/stable/modules/generated/sklearn.model_selection.cross_val_predict.html

Sklearn.model_selection.GridSearchCV. (2022). Scikit-Learn. https://scikit-learn/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Sklearn.preprocessing.MinMaxScaler. (2021). Scikit-Learn. https://scikit-learn/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

Sklearn.tree.DecisionTreeClassifier. (2022). Scikit-Learn. https://scikit-learn/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Somvanshi, M., Chavan, P., Tambade, S., & Shinde, S. V. (2016). A review of machine learning techniques using decision tree and support vector machine. *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, 1–7. https://doi.org/10.1109/ICCUBEA.2016.7860040

Teorema de Bayes. (2022). Em *Wikipédia, a enciclopédia livre*. https://pt.wikipedia.org/w/index.php?title=Teorema_de_Bayes&oldid=63939256

Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, *58*(1), 267–288.

Toksari, M. D. (2016). A hybrid algorithm of Ant Colony Optimization (ACO) and Iterated Local Search (ILS) for estimating electricity domestic consumption: Case of Turkey. *International Journal of Electrical Power & Energy Systems*, *78*, 776–782. https://doi.org/10.1016/j.ijepes.2015.12.032

TPOT. (2022). http://epistasislab.github.io/tpot/

Urbanowicz, R. J., Meeker, M., La Cava, W., Olson, R. S., & Moore, J. H. (2018). Relief-based feature selection: Introduction and review. *Journal of Biomedical Informatics*, *85*, 189–203. https://doi.org/10.1016/j.jbi.2018.07.014

US Biobank | Browse Our Diverse Biobanks | Coriell Institute. (2021). https://www.coriell.org/1/Browse/Biobanks

Vasconcelos, B. F. B. de. (2017). *Poder preditivo de métodos de Machine Learning com processos de seleção de variáveis: Uma aplicação às projeções de produto de países* [Master, Universidade de Brasília]. https://doi.org/10.26512/2017.03.D.23995

Wiederhold, G., & McCarthy, J. (1992). Arthur Samuel: Pioneer in Machine Learning. *IBM Journal of Research and Development*, *36*(3), 329–331. https://doi.org/10.1147/rd.363.0329

Witten, D., Tibshirani, R., Gu, S. G., Fire, A., & Lui, W.-O. (2010). Ultra-high throughput sequencing-based small RNA discovery and discrete statistical biomarker analysis in a collection of cervical tumours and matched controls. *BMC Biology*, *8*(1), 58. https://doi.org/10.1186/1741-7007-8-58

Zhou, J., Gandomi, A. H., Chen, F., & Holzinger, A. (2021). Evaluating the Quality of Machine Learning Explanations: A Survey on Methods and Metrics. *Electronics*, *10*(5), 593. https://doi.org/10.3390/electronics10050593

Zimányi, L., Sipos, Á., Sarlós, F., Nagypál, R., & Groma, G. I. (2021). Machine-learning model selection and parameter estimation from kinetic data of complex first-order reaction systems. *PLOS ONE*, *16*(8), e0255675. https://doi.org/10.1371/journal.pone.0255675

Anexos

Anexo A – Tendency Iterated Local Search – TILS
https://github.com/sousa426/tils/blob/main/TILS.py

Anexo B – Ficheiros relativos ao dataset MicroRNA
https://github.com/sousa426/tils/tree/main/miRNA

Anexo C – Afinação de parâmetros do modelo
https://github.com/sousa426/tils/blob/main/miRNA/103_Decision_Tree_Classifier_GridSearchCV.py

Anexo C – Ficheiros relativos ao dataset Amyotrophic Lateral Sclerosis (ALS)
https://github.com/sousa426/tils/tree/main/ALS