INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA

Automatização do Processo de Ciência de Dados: Do Pré-Processamento à Implantação do Modelo Final

NUNO MIGUEL TAVARES FERNANDES
Setembro de 2024

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Automatização do Processo de Ciência de Dados: Do Pré-Processamento à Implantação do Modelo Final

Nuno Miguel Tavares Fernandes

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas de Informação e Conhecimento

Orientador: Maria de Fátima Coutinho Rodrigues

Porto, setembro 2024

Declaração de Integridade

Declaro ter conduzido este trabalho académico com integridade.

Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Portanto, o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

ISEP, Porto, 14 de setembro de 2024

Dedicatória

" Success is the sum of small efforts - repeated day in and day out." — Robert Collier

Àqueles que me apoiaram durante esta jornada.

Resumo

Com o desenvolvimento das tecnologias de Machine Learning deu-se uma revolução nos processos de tomada de decisão de diversas indústrias. Para aumentar a disponibilidade desta tecnologia para um publico mais geral existe um grande interesse na automatização progressiva de operações de Machine Learning, AutoML.

Nesta dissertação em primeiro lugar utiliza-se uma adaptação do PRISMA para fazer uma revisão do estado da arte para os conceitos do ML e as suas ferramentas de automatização. Em seguida é realizada uma pequena avaliação e comparação de algumas dessas ferramentas, com base na resolução de problemas simples, de modo a conseguir perceber as vantagens e desvantagens práticas de cada uma dessas soluções. Por último é abordado e explicado o processo de design, implementação e avaliação de uma aplicação de AutoML para problemas de classificação.

Palavras-chave: Machine Learning, Automatização, Machine Learning pipeline, AutoML

Abstract

With the development of Machine Learning technologies, a revolution has taken place in the decision-making processes of various industries. In order to increase the availability of this technology to the general public, there is great interest in the progressive automation of Machine Learning operations, AutoML.

This dissertation first uses an adaptation of PRISMA to review the state of the art in ML concepts and its automation tools. This is followed by a short evaluation and comparison of some of these tools, based on solving simple problems, to understand the practical advantages and disadvantages of each of these solutions. Finally, the process of designing, implementing and evaluating an AutoML application for classification problems is discussed and explained.

Keywords: Machine Learning, Automation, Machine Learning pipeline, AutoML

Agradecimentos

Esta dissertação representa o resultado de uma longa jornada, e não poderia ter sido concluída sem o apoio e a contribuição de várias pessoas e instituições às quais sou profundamente grato.

Em primeiro lugar, agradeço à minha orientadora, professora Maria de Fátima Coutinho Rodrigues, pela orientação, paciência e valiosos conselhos ao longo deste trabalho. A sua sabedoria e entusiasmo foram essenciais para a conclusão deste projeto.

Agradeço também aos meus colegas e amigos, que me apoiaram nos momentos difíceis, ao oferecer palavras de coragem e suporte emocional. A vossa colaboração foi crucial para me manter motivado e focado.

Aos meus professores e ao Instituto Superior de Engenharia do Porto (ISEP), que proporcionaram o conhecimento e as ferramentas necessárias para que eu pudesse desenvolver este trabalho. As vossas contribuições ao longo da minha formação académica foram inestimáveis.

Um agradecimento especial à minha família e à minha namorada, pelo apoio e compreensão durante os desafios desta jornada. Sem a força e a confiança que depositaram em mim, este trabalho não teria sido possível.

Por fim, agradeço a todos aqueles que, direta ou indiretamente, contribuíram para a realização deste projeto. Todas as contribuições, grandes ou pequenas, foram fundamentais para que eu pudesse alcançar este objetivo.

Índice

1 Introdução 1

1.1 Contexto do problema 1

1.2 Descrição do problema 2

1.3 Objetivos 2

1.4 Análise ética do problema 4

1.5 Estrutura do documento 4

2 Estado da Arte 7

2.1 Processo de pesquisa 7
2.1.1 Ética no processo de pesquisa 8

2.2 Introdução ao AutoML 8

2.3 História do AutoML 10

2.4 Estrutura da pipeline 10
2.4.1 Estrutura fixa 11
2.4.2 Estrutura variável 12

2.5 Recolha de dados 13

2.6 Limpeza de dados 13

2.7 Engenharia de *features* 14
2.7.1 Criação de features 14
2.7.2 Seleção de features 15

2.8 Seleção do algoritmo e otimização dos híper-parâmetros 15
2.8.1 Estratégia de busca simples 16
2.8.2 Otimização Derivate-Free 17
2.8.3 Método do gradiente 18
2.8.4 Algoritmo guloso 18

2.9 *Deployment* 18
2.9.1 Integração 19
2.9.2 Monitorização 19
2.9.3 Atualização 19
2.9.4 Exemplos soluções 20

2.10 Técnicas para melhorar o desempenho do AutoML 20
2.10.1 Aproximações multi-fidelidade 20
2.10.2 Paragem antecipada 20
2.10.3 Escalabilidade 21
2.10.4 Ensemble learning 21
2.10.5 Meta-learning 21
2.10.6 Refinamento do espaço de pesquisa 21
2.10.7 Sugestão de configuração candidata 22

2.11 Níveis de automatização de frameworks 22

2.12 Frameworks de AutoML ... 24
2.12.1 Automatização de visualização e limpeza ... 24
2.12.2 Automatização na engenharia de features ... 25
2.12.3 Automatização de hiperparameterização ... 26
2.12.4 Automatização da exploração de alternativas para modelos ... 27
2.12.5 Automatização na avaliação ... 28

2.13 Sistemas de AutoML ... 28

2.14 Conclusões da revisão ... 29

3 Definição dos Requisitos e Arquitetura da Solução ... 31

3.1 Engenharia de Requisitos ... 31
3.1.1 Ator do sistema e os principais benefícios do seu uso ... 31
3.1.2 Requisitos funcionais ... 32
3.1.3 Requisitos não-funcionais ... 33

3.2 Análise de ferramentas/abordagens para resolução do problema ... 34
3.2.1 Critérios de avaliação a usar ... 34
3.2.2 Análise das ferramentas ... 35
3.2.3 Escolha da ferramenta ... 37

3.3 Análise da arquitetura da solução ... 41
3.3.1 Visão geral ... 42
3.3.2 Componentes principais da Arquitetura ... 43
3.3.3 Fluxo de Dados ... 49
3.3.4 Requisitos não funcionais na arquitetura da solução ... 50

4 Implementação da solução ... 51

4.1 Ambiente de Desenvolvimento ... 51

4.2 Estrutura do código ... 52

4.3 Overview da implementação dos componentes principais ... 53
4.3.1 Interface Gráfica ... 53
4.3.2 Recolha de Dados ... 54
4.3.3 Estratégia de Pré-Processamento Customizada ... 54
4.3.4 Estratégia de Treino do Modelo ... 55
4.3.5 Estrategia Deployment ... 57

4.4 Exemplo de uso da aplicação ... 57

5 Experiências efetuadas e avaliação da solução ... 75

5.1 Metodologia ... 75

5.2 Resultados dos testes efetuados ... 76

5.3 Discussão de resultados ... 91

5.4 Avaliação de um ponto de vista ético ... 92

6 Conclusão ... 95

Lista de Figuras

Figura 1 - Especialista vs AutoML no processo de uma aplicação ML [4] 9
Figura 2 - Estrutura básica pipeline segundo boas práticas [11] 11
Figura 3 - Aspeto geral para pipelines de estrutura fixa [12] 11
Figura 4 - Aspeto de uma pipeline com estrutura variável [3] 12
Figura 5 - Esquema para criação de *features* [3] 14
Figura 6 - Níveis de automatização para framework [15] 22
Figura 7 - Cálculos para aplicação TOPSIS 39
Figura 8 - Matriz normalizada 40
Figura 9 - Matriz ponderada 40
Figura 10 - Solução ideal $Di +$ 40
Figura 11 - Solução ideal para $Di -$ 41
Figura 12 - Distâncias euclidianas 41
Figura 13 - Visão geral arquitetura do sistema 42
Figura 14 - Arquitetura da pipeline de pré-processamento customizada 44
Figura 15 - Fluxo de dados do sistema 49
Figura 16 - Estrutura da pasta da aplicação 52
Figura 17 - Exemplo de implementação de interface com recolha de dados 53
Figura 18 - Implementação da regra definida para qualificar colunas para atributo objetivo .. 54
Figura 19 - Implementação do método que estrutura a pipeline de pré-processamento 55
Figura 20 - Implementação da estratégia de divisão de dados para treino e teste 55
Figura 21 - Implementação das métricas adicionais a guardar nos resultados 56
Figura 22 - Implementação da adição de um componente para uma seção específica do Auto-sklearn 56
Figura 23 - Implementação instância automl com configurações e respetivo fit 56
Figura 24 - Implementação da invocação do componente com a estratégia de explicação 56
Figura 25 - Implementação da invocação do componente com a estratégia de guardar informação sobre o modelo 56
Figura 26 - Implementação da estratégia de *deployment* 57
Figura 27 - Exemplo de ativação de ambiente virtual e início de uma instância do MLFow 57
Figura 28 - Visão inicial da aplicação 58
Figura 29 - Vista auxiliar para escolher ficheiro csv 58
Figura 30 - Vista atualizada depois da seleção do ficheiro de dados 59
Figura 31 - Vista para escolha entre realizar pré-processamento customizado ou não 59
Figura 32 - Interface pré-processamento customizado 60
Figura 33 - Opções para primeiro campo 60
Figura 34 - Opções para segundo campo 60
Figura 35 - Opções para terceiro campo 60
Figura 36 - Opções para quarto campo 61
Figura 37 - Opções para quinto campo 61
Figura 38 - Vista do painel de treino de modelo 61

Figura 39 - Opções para variável "*Ensemble class*" 62
Figura 40 - Opções para variável "Resampling strategy" 62
Figura 41 - Opções para variável "Métrica para avaliar" 63
Figura 42 - Confirmação do início do processo de treino 63
Figura 43 - Visão geral da estratégia de explicação 64
Figura 44 - *Heatmap* sobre híper-parâmetros 65
Figura 45 - Informação sobre importância de *features* 65
Figura 46 - Informação geral sobre importância de *features* 66
Figura 47 - Matriz confusão 66
Figura 48 - Relatório de classificação 66
Figura 49 - Lista modelos ordenados por classificação média 67
Figura 50 - Tabela classificação dos modelos da tarefa 67
Figura 51 - Estatísticas da tarefa 67
Figura 52 - Performance ao longo do tempo 68
Figura 53 - Confirmação sobre processo que guarda as informações do modelo no MLFow... 68
Figura 54 - Interface inicial MLFlow 69
Figura 55 - Informação geral sobre um modelo selecionado 69
Figura 56 - Parâmetros e métrica de avaliação guardada no MLFow 70
Figura 57 - Continuação da lista de parâmetros guardados no MLFow 70
Figura 58 - Visualização da seção sobre artefactos do modelo guardados 71
Figura 59 - Lista com opções de modelos adequadas para problema 71
Figura 60 - Interface de confirmação do modelo escolhido 72
Figura 61 - Processo de download dos artefactos e *deploy* do modelo 72
Figura 62 - Simulação de uma interação com sistema através do Postman 72

Lista de Tabelas

Tabela 1 - Matriz de decisão para TOPSIS 38
Tabela 2: Testes cenário 1 77
Tabela 3: Testes cenário 2 79
Tabela 4: Teste cenário 3 81
Tabela 5: Testes CD3 83
Tabela 6: Testes CD4 83
Tabela 7 - Testes CD5 84
Tabela 8: Testes CD6 85
Tabela 9: Testes CD7 86
Tabela 10: Testes CD8 86
Tabela 11: Testes CD9 87
Tabela 12: Testes CD10 88
Tabela 13: Testes CD11 89
Tabela 14: Testes CD12 90
Tabela 15: Testes CD13 91

Acrónimos

Lista de Acrónimos

ML	*Machine Learning*
AutoML	*Automated Machine Learning*
DevOps	Desenvolvimento e Operações
CASH	*Combined algorithm selection and hyperparameter optimization*
PCA	*Principal Component Analysis*
ICA	*Independent Component Analysis*
API	*Application Programming Interface*
CSV	*Comma-separeted Values*

1 Introdução

Na primeira seção do trabalho vai se proceder a uma introdução ao problema que está a ser explorado, passando por uma pequena contextualização, uma descrição do problema, os objetivos propostos para o trabalho e por fim uma apresentação da estrutura do documento.

1.1 Contexto do problema

O *Machine Learning* (ML) inicialmente com base em modelos estatísticos tradicionais, e nos dias de hoje com o desenvolvimento de redes neurais e algoritmos de *deep learning*, sofreu uma enorme evolução.

Algoritmos cada vez mais complexos e poderosos revolucionaram várias indústrias e setores, permitindo melhorar processos de tomada de decisão. Com base no apoio dado aos profissionais das mais variadas áreas, é possível ter um processo de tomada de decisão mais ponderado, com base numa análise objetiva efetuada por uma "máquina", conseguindo retirar a subjetividade associada ao ser humano e de forma mais fiável.

Contudo há um senão neste processo que consiste na dependência do fator humano no processo de análise através do ML, pois depende muito ainda do seu fator de tomada de decisão durante a sua normal execução. Para se perceber melhor, o processo de ML pode ser divido em várias fases, que são facilmente identificadas numa das metodologias mais usadas neste âmbito, que é a metodologia CRISP-DM [1], e todas as suas fases dependem fortemente dum especialista da área para as poder compreender e executar da melhor maneira.

Estas várias etapas em que o ML pode ser dividido podem ser caracterizadas por serem repetitivas, mas mesmo assim consumindo muito tempo e recursos pois precisam de profissionais especializados nas áreas e são tarefas que consomem muito tempo.

Desta dependência de um especialista e do aumento da procura pelo mercado em geral, surge uma oportunidade para a automatização de alguns procedimentos que visam melhorar tanto a

eficácia, como a velocidade do processo, conseguindo ainda desassociar o humano do processo, o que pode ajudar a remover alguns erros associados ao mesmo.

Por outro lado, com a possibilidade de automatização surge também a oportunidade de retirar a necessidade de um especialista na área, tornando o ML mais acessível aos utilizadores. Sendo para isto também necessário realizar uma padronização e automatização principalmente do *deployment* dos modelos posteriormente criados para serem ainda mais facilmente utilizados.

Nesta dissertação vai ser feita uma exploração das tecnologias existentes para automatização do ML, estabelecendo em primeiro lugar uma comparação teórica das suas respetivas vantagens e desvantagens conseguindo assim perceber quais podem ser mais benéficas para serem usadas. Numa segunda etapa vão ser feitos alguns testes práticos com algumas das tecnologias estudadas para assim se poder escolher as que vão ser usadas para construir a solução proposta.

1.2 Descrição do problema

Na era dos grandes volumes de dados e da crescente procura de decisões baseadas em dados, as organizações enfrentam o desafio de simplificar o processo de implementação da aprendizagem automática através da sua automatização e a procura de otimização.

Embora os modelos de ML estejam a tornar-se mais potentes e sofisticados, a sua utilização eficiente e a garantia da qualidade dos dados para análise continuam a ser um estrangulamento significativo. O mesmo se pode aplicar ao desenvolvimento de soluções de AutoML, ou seja, neste grau de automatização, a necessidade de um especialista ainda é uma realidade o que em termos práticos não ajuda ainda o utilizador comum a ter acesso à tecnologia.

Neste paradigma é possível identificar vários pontos. Os processos de AutoML existem, mas ainda são difíceis de utilizar e têm a necessidade de um especialista. Algumas das fases ainda apresentam um grau muito forte de dependência do humano, como por exemplo nas fases de *data understanding*, *data preparation* e posteriormente o *deployment*.

Este problema é agravado pela necessidade de escalabilidade devido à necessidade de processamento de dados em tempo real e de diversas fontes de dados, controlo de versões e pela monitorização dos modelos depois de estarem em ambiente produção.

1.3 Objetivos

O objetivo final desta dissertação é o desenvolvimento de uma aplicação que proporcione o uso de ML a utilizadores que não sejam especialistas, conseguindo assim melhorar a sua acessibilidade para integração com outros sistemas. Para isso vão ser executadas uma lista de tarefas que contêm alguns objetivos intermédios para que com a sua conclusão o objetivo final seja alcançado:

1. Analisar possível otimização da automatização da análise de dados
    a. Investigar e testar técnicas e frameworks para automatização do pré-processamento de dados, nos campos de seleção de *features*, limpeza e exploração dos dados.
    b. Explorar estratégias para processar dados de vários tipos, por exemplo estruturados e não estruturados e de várias fontes.
    c. Considerar desafios de controlo de versões, monitorização de modelos e escalabilidade no *deployment*.
2. Analisar possível otimização de automatização da preparação dos dados
    a. Investigar e testar técnicas e frameworks para automatização de preparação, transformação e integração de dados.
    b. Considerar possíveis validações para conseguir validar a integridade dos dados usados para análise.
3. Desenvolver uma aplicação de AutoML para problemas de classificação
    a. Realizar um estudo das frameworks que existem para perceber o que pode ser usado para construir a aplicação.
    b. Efetuar testes com algumas das frameworks para perceber a verdadeira extensão das suas capacidades.
    c. Construir a aplicação tirando partido das frameworks analisadas.
    d. Considerar e implementar opções de ambientes para *deployment*.
4. Avaliar e testar o desempenho da solução
    a. Realizar testes utilizando a aplicação desenvolvida para avaliar o seu desempenho em termos de qualidade do modelo deployed e tempo de execução.
    b. Realizar os mesmos testes executando todas as fases da metodologia sem qualquer automatismo para comparação e melhor avaliação dos resultados da aplicação.
    c. Testar a aplicação em casos de estudo ou em cenários do mundo-real.
    d. Avaliar de um ponto de vista ético a aplicação e o seu impacto.

Em suma, com esta dissertação pretende-se fornecer um estudo sobre os processos que estão envolvidos quando se utiliza o ML, e de que maneira as etapas do processo de ML podem ser automatizadas. Depois de analisar os aspetos teóricos, o objetivo é fazer um estudo tanto teórico como prático de frameworks de AutoML para perceber a sua extensão e capacidades. Com base nesse estudo criar uma aplicação que use uma ou mais dessas frameworks para conseguir da forma mais automática possível receber dados e com eles fazer todo o seu processamento e como output disponibilizar ao utilizador um modelo pronto a utilizar.

1.4 Análise ética do problema

De um ponto de vista ético o problema levanta algumas considerações pois com o desenvolvimento dos mecanismos de ML, o seu uso vai ser difundido e isso faz com que se tenha de ponderar em alguns pontos. Relativamente ao enviesamento e à justiça, um sistema que é construído com base em dados enviesados vai produzir um resultado também ele enviesado o que pode afetar as previsões relativamente à sua justiça. Para combater isso é importante aplicar técnicas de redução desse enviesamento dos dados.

Outro tópico a ter em conta é a privacidade e proteção dos dados. A recolha autónoma de dados e a sua análise pode interferir com a privacidade de indivíduos e por isso caso não seja tratado duma maneira cuidadosa e responsável pode provocar usos impróprios dos mesmos. Para assegurar que isso não ocorre podem ser implementadas técnicas de anonimização, ou adotadas metodologias para lidar com dados que tenham ideais de preservação da privacidade.

Um tópico também importante é a transparência dos processos, pois os sistemas muitas vezes produzem recomendações ou decisões cujo processo de desenvolvimento não é percetível, tornando difícil para os utilizadores perceberem os resultados de determinada decisão ou previsão. De modo a combater isso podem ser implementados modelos com capacidade de explicação, de modo a dar transparência ao processo decisivo, principalmente em sistemas em contextos de decisão ou recomendação crítica.

Em relação à responsabilidade e responsabilização de sistemas de ML é considerado um dos maiores dilemas éticos da área. No caso de o sistema fazer uma previsão automatizada, é possível culpar o sistema por um erro, ou o responsável por esse erro é o humano ou entidade que o desenvolveu ou que o detém. Uma boa pratica neste caso é implementar mecanismos de validação e supervisão humana, que permitam a sua intervenção atempadamente.

Por último, como todos os processos de automatização ou robotização surge a questão social sobre a substituição do humano pela máquina nas suas funções de trabalho. Deve ser elaborada uma ponderação sobre o possível impacto da automatização do processo, neste caso de ML, para a potencial substituição dos postos de trabalho. Caso afete postos de trabalho, podem os profissionais afetados ser treinados para outras funções, pois esse trabalho específico pode ter sido substituído, mas um novo posto de trabalho pode surgir como consequência dessa automatização.

1.5 Estrutura do documento

Esta dissertação está dividida em vários capítulos para uma melhor organização da informação e de forma seguir uma linha lógica de desenvolvimento. No capítulo 1 é feita uma introdução ao tema e ao problema, fazendo uma análise do mesmo de uma vertente analítica e também ética.

No segundo capítulo, relativo ao Estado de Arte, em primeiro lugar é abordado o processo de pesquisa, seguido de algumas considerações éticas para o mesmo, e depois é feita uma revisão dos conceitos que envolvem o ML, a sua automatização e também as suas principais frameworks.

O terceiro capítulo, Definição dos Requisitos e Arquitetura da Solução, onde primeiro são abordados os requisitos funcionais e não-funcionais identificados, para além da identificação do ator do sistema e explicação dos principais benefícios seu uso. Depois é feita uma análise das ferramentas ou abordagens que podem ser aplicadas no desenvolvimento da solução, saindo desta análise uma escolha justificada. E por último é efetuada uma análise sobre a arquitetura da solução, desde uma visão geral até aos seus principais componentes, abordando também o fluxo de dados no sistema.

O capítulo 4, Implementação da Solução, aborda alguns dos principais detalhes sobre a implementação, idealizada e desenhada no capítulo anterior. Numa primeira parte detalha o ambiente de desenvolvimento, seguido de uma análise à estrutura do código e uma *overview* da implementação de detalhes importantes de alguns dos componentes principais. Por último através de um exemplo é explicado todo o funcionamento da aplicação.

O quinto capítulo, Experiências é realizada a avaliação da solução, apresentadas as estratégias utilizadas para avaliar o sistema desenvolvido e detalhadas algumas experiências que foram efetuadas durante o desenvolvimento do mesmo.

O capítulo 6, Conclusão, começa por apresentar as principais conclusões do trabalho, fazendo também referência às limitações e trabalho futuro.

2 Estado da Arte

Este capítulo é referente ao estado da arte, onde é feita uma revisão da literatura científica sobre os vários processos em que o ML é baseado, como pode ser feita a sua automatização e por último são descritos frameworks ou tecnologias nessa mesma área.

2.1 Processo de pesquisa

Para a realização da pesquisa foi usada uma adaptação do PRISMA [2]. Em primeiro lugar foram definidas as perguntas de pesquisa que se pretendem ter respondidas:

1. Qual é o estado atual teórico sobre a automatização dos processos de ML?
2. Quais são as principais frameworks e tecnologias usadas para AutoML?
3. Quais os principais problemas, falhas, ou dificuldades identificadas?
4. Que considerações éticas se podem aplicar?

Segundo passo foi definir as bases de dados para usar no processo de investigação. Para este efeito foram escolhidas as bases de dados explicitadas em baixo. Esta escolha foi baseada na quantidade de informação que cada uma delas contém e pela variedade de opções de filtragem e pesquisa.

1. B-on - https://www.b-on.pt/
2. Google Schoolar - https://scholar.google.com/

O passo seguinte consistiu na definição de uma *query* de pesquisa que buscasse agrupar todos os termos importantes para conseguir filtrar artigos com informação que forneça a capacidade de responder a todas as perguntas de pesquisa. Alguns dos exemplos que foram usados:

1. machine learning automation and feature selection in machine learning and machine learning pipeline and data collection in machine learning and deployment in machine learning and automation
2. auto machine learning or autoML or automation in machine learning and data collection automation in machine learning and data collection automation in machine learning
3. auto machine learning or autoML or automation in machine learning

Foram testadas várias *queries* de pesquisa, como se pode observar nos exemplos acima, contudo nenhuma delas apresentou resultados satisfatórios. Algumas delas mais extensas e com mais produtos de conjunção traziam muito pouco volume de resultados tanto em quantidade como em qualidade, e com um tipo de pesquisa mais aberta vinha o problema de ter demasiados resultados, uma vez que o tópico ML tem muito trabalho publicado associado,

contudo mesmo especificando a vertente de automatização quase todos os resultados que eram devolvidos eram relativos exclusivamente a ML e as suas aplicações em diversas áreas mas em nada abordando a sua automatização, ou o uso da mesma. Contudo dessa pesquisa surgiram três revisões sobre a matéria da automatização do ML especificamente, e para pesquisa foram usadas essas revisões como base e delas seguiu-se para a observação das suas referências uma vez que como foram usadas em trabalhos publicados e revistos, com alguma importância, já lhe confere um maior grau de qualidade, mas mais importante que isso, todas abordam tópicos necessários à investigação para esta dissertação.

Dessa análise surgiram os trabalhos que foram utilizados para obter as informações necessárias para realizar o estado da arte desta dissertação.

2.1.1 Ética no processo de pesquisa

Relativamente à ética no processo de pesquisa, há alguns tópicos aos quais se deve prestar atenção, com por exemplo, manter a transparência do processo. Esta transparência refere-se principalmente a mostrar explicitamente as fontes de informação utilizadas, as bases de conhecimento, os métodos usados e as restrições aplicadas, evitando assim potenciais dúvidas nos processos de obtenção do conhecimento, por exemplo, os meios utilizados para ter acesso a um determinado artigo. Outro tópico a ter em atenção consiste em ter responsabilidade nos resultados da pesquisa, por um lado, evitando enviesamentos de interpretação, e por outro lado, para manter a integridade académica, evitando o plágio ou qualquer outra prática antiética. Por último, em alguns casos pode ser importante procurar uma instituição relevante na área da ética, para em caso de alguma dúvida, poder manter sempre o processo de pesquisa o mais dentro das normas éticas definidas.

2.2 Introdução ao AutoML

Construir uma pipeline de ML consiste num esforço coletivo de vários profissionais especializados. Cientistas de dados que detém um grande conhecimento estatístico e de algoritmos de ML. Por outro lado, para cada problema existe um especialista do domínio, que é responsável por trazer todo o conhecimento específico sobre o a área do problema que se pretende resolver. Juntos possuem a habilidade para montar a pipeline de ML contendo um pré-processamento de dados específico e ajustado ao problema, uma engenharia de *features* também ela aprimorada e consistente com o domínio do problema a resolver e por final, modelos ajustados que forneçam um grande poder preditivo. Este processo é baseado numa metodologia iterativa de tentativa e erro de tal forma que obter os melhores resultados pode tornar o processo geral muito demorado e com um custo alto, o que muitas vezes faz com que se opte por uma solução sub-ótima [3].

Com isto surge a questão, quais são as possibilidades para se conseguir uma solução ótima, ou pelo menos baixar os custos associados? Uma das soluções consistiu em tentar automatizar processos. Pensando no processo de construir a pipeline, caso fosse possível retirar o especialista do processo, não ia ser mais rápido e barato para as organizações terem as suas soluções disponíveis e prontas a serem utilizadas? Com esta proposta de automatização o ML poderia tornar-se mais acessível para uso em aplicações do dia-a-dia dos utilizadores, podendo melhorar drasticamente os seus níveis de competência e customização [5].

No seguimento desta ideia surge então o AutoML. Com o objetivo de melhorar a maneira como se desenvolvem as aplicações de ML baseando-se na automatização de processos, dando a possibilidade de os especialistas beneficiarem da automatização de tarefas repetitivas e demoradas como por exemplo o processo de otimização por híper-parametrização, conseguindo manter ou até melhorar a sua eficiência. Com isto é possível dar ao especialista de domínio a capacidade de conseguir desenvolver as suas próprias pipelines sem depender tanto do cientista de dados [3].

Em seguida é posto em comparação as atividades realizadas pelo AutoML e as atividades cuja interação humana não é ainda considerada de ser alterada.

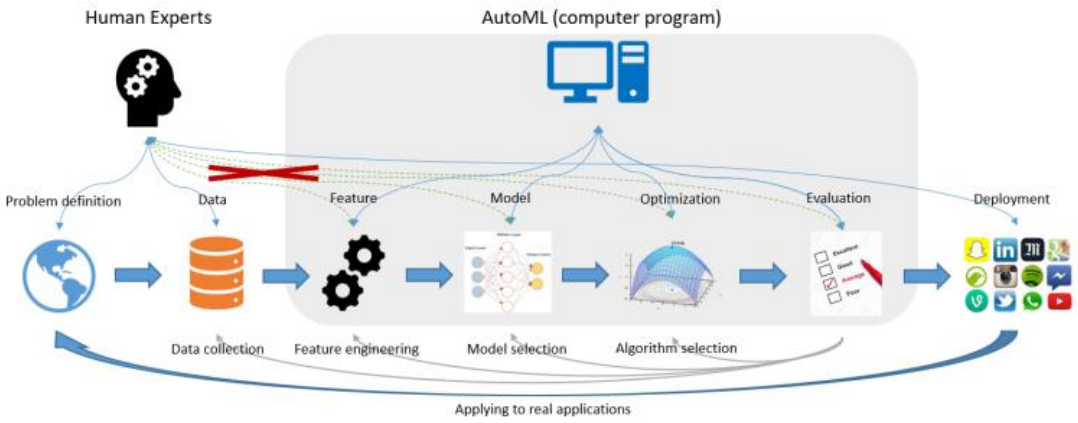

Figura 1 - Especialista vs AutoML no processo de uma aplicação ML [4]

Como se pode observar na figura 1 as áreas de definição de problema, recolha de dados e *deployment* são ainda de responsabilidade atribuída ao especialista enquanto o AutoML foca os seus esforços nos módulos de engenharia de *features*, seleção de modelo, a sua otimização por híper-parametrização e o processo de avaliação do modelo. Contudo é possível ao especialista e sistema terem interações como se pode observar pelas linhas vermelhas que os conectam.

Sintetizando, o AutoML tem como pilares as metas:

- Boa performance – através de uma boa generalização permite que sejam atingidas várias tarefas de aprendizagem, com diferentes dados de entrada;

- Menor assistência humana – configurações ótimas podem ser encontradas automaticamente;
- Alta eficácia computacional – programa é capaz de retornar uma resposta dentro de um espaço de tempo limitado e programado [4].

2.3 História do AutoML

Apesar da atualidade do tópico, AutoML não é um conceito novo. Em 2004, surgiu a primeira abordagem para automatização de seleção de *features* onde se usou uma estratégia baseada em algoritmos genéticos para selecionar quais seriam usadas como input [9].

A primeira solução de automatização de várias etapas da pipeline, construída em 2009, era um sistema onde automaticamente era feito pré-processamento, seleção de *features*, e simultaneamente a seleção do modelo em conjunto com a sua otimização designada por *Full model selection* [5]. Esta primeira solução foi depois sujeita a testes através do uso de vários conjuntos de dados, e foi provada como sendo agnóstica ao domínio [6].

Antes disso, nos anos 90, surgiu uma solução comercial que automatizava a otimização por híper-parametrização para algoritmos de classificação usando *grid search* [7]. Nos primeiros anos do século XXI foram apresentadas as primeiras formas eficientes para a sua otimização, onde foi comprovado que estratégias de busca guiadas eram mais rápidas e eficazes comparativamente ao uso do *grid search* [8].

2.4 Estrutura da pipeline

Uma pipeline de ML é uma combinação de vários algoritmos ($h: X \rightarrow Y$) que têm a capacidade de transformar um vetor de *features* ($\sim x \in X$) num valor objetivo ($y \in Y$), como por exemplo, num problema de classificação, dado um determinado conjunto de características (*features*) de um individuo conseguir atribuir a resposta ao problema definido. Esta definição pode ser traduzida por $h: X \rightarrow Y, \sim x \in X \wedge y \in Y$.

O problema da criação da pipeline é normalmente resolvido pelos especialistas de uma maneira iterativa. Em primeiro lugar uma pipeline simples é construída usando algoritmos standard e com os seus híper-parâmetros definidos como *default*. Depois vai-se testando as várias possibilidades com as diferentes combinações de híper-parâmetros de cada algoritmo até que se chegue a um resultado de performance satisfatório [3].

Com o uso do AutoML um dos fatores a considerar pode ser a criação desta estrutura que pode ser construída duma maneira fixa ou variável, contudo por indicação das boas práticas, o seu exemplo mais básico é o mostrado na figura 2.

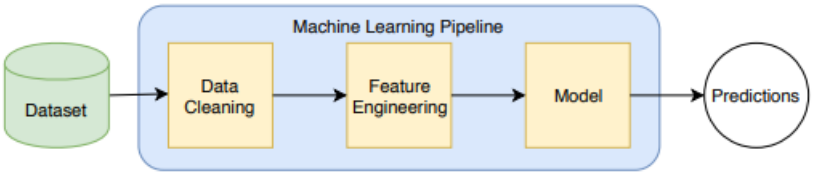

Figura 2 - Estrutura básica pipeline segundo boas práticas [11]

Pode observar-se que a pipeline é formada por três seções, limpeza de dados, a engenharia de *features* e o modelo. Primeiro passo, no processo de *data cleaning*, os dados provenientes do *dataset* são tratados através de alguns passos como por exemplo, verificação de dados em falta ou codificação (*encoding*) de atributos. Depois de tratados, começa a fase de engenharia de *features* onde se faz a criação e a seleção de *features*. Esta fase está muito conectada ao domínio específico onde está a ser aplicada. Na terceira fase os algoritmos de ML são treinados com os dados contendo as *features* resultantes da segunda fase. Este é o processo básico, sem entrar em grandes detalhes, pois cada uma das fases vai ser abordada individualmente no decorrer da dissertação.

2.4.1 Estrutura fixa

A pipeline de estrutura fixa muito parecida com a estrutura básica, indicada nas boas práticas, consiste numa sequência linear de ações como se pode ver na figura 3.

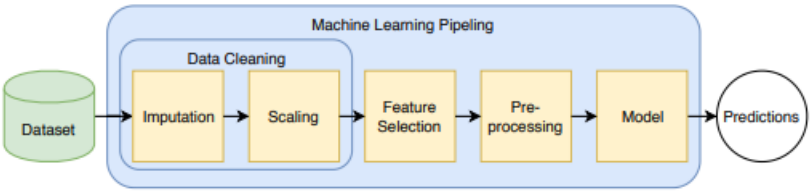

Figura 3 - Aspeto geral para pipelines de estrutura fixa [12]

Pode observar-se que consiste num processo também de limpeza de dados, mas desta vez subdividido entre imputação de valores e normalização. Esta subdivisão muitas vezes pode não ser representada, mas pelo menos estes dois processos estão presentes dentro dessa secção. A engenharia de *features* mantém-se assim como o modelo, mas existe uma seção de pré-processamento, onde se faz a escolha de um algoritmo entre um conjunto deles que é o que permite determinar qual o algoritmo de ML e a configuração que vai ser usada.

Ao usar esta pipeline a complexidade de determinar a estrutura não é considerada e o problema passa a consistir na fase de pré-processamento e modelo. Apesar desta diminuição da complexidade a performance geral da pipeline pode ser menor comparativamente com a pipeline de estrutura variável, principalmente em problemas com um maior grau de complexidade. Contudo para problemas com muitos dados e com elevada qualidade nos mesmos, ponderando entre a complexidade acrescida de computar uma estrutura variável e usar uma estrutura fixa, a segunda opção é mais vantajosa. Por isso a maior parte das

frameworks de AutoML não resolve o problema da estrutura e foram desenvolvidas para usar estrutura de pipeline fixa.

2.4.2 Estrutura variável

Os especialistas em ciências dos dados são capazes de construir pipelines altamente customizadas para um determinado problema ML, obtendo assim os melhores resultados, pois em comparação com as estruturas fixas, as estruturas variáveis apresentam maior flexibilidade para se adaptar aos contextos dos problemas. Muitas das formas para construir estas pipelines automaticamente baseiam-se nos mesmos princípios, uma pipeline consiste num conjunto de primitivos de ML:

- Algoritmos básicos de ML;
- "Clonador" do conjunto de dados;
- Mecanismo que faça união de *features* juntando conjuntos de dados.

Pode observar-se como exemplo a figura 4.

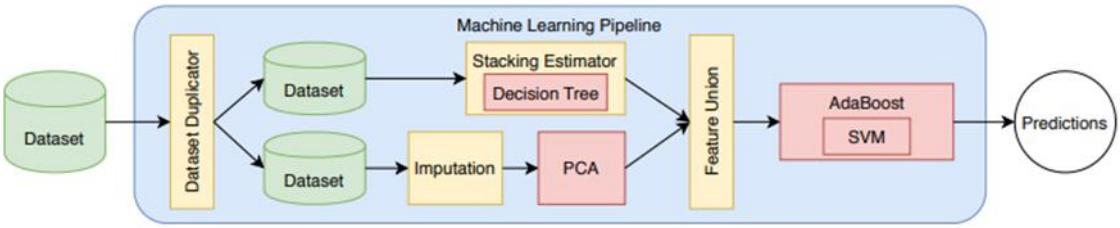

Figura 4 - Aspeto de uma pipeline com estrutura variável [3]

Como se pode observar o conjunto de dados original é copiado usando o "clonador", o que permite serem criados dois fluxos que podem correr em paralelo e serem depois novamente juntos por um mecanismo apropriado.

Pipelines podem ser interpretadas como estruturas em árvore que são criadas através de programação genética. Ou seja, há a combinação de dois indivíduos através da seleção de subgrafos da estrutura da pipeline e dai há a criação de um novo grafo. A mutação é implementada através da adição ou da eliminação de um nó, e assim pipelines mais flexíveis são criadas.

Para gerar estas pipelines de estrutura dinâmica, vários métodos podem ser usados:

- *Hierarchichal task networks* (HTNs), que conseguem pegar no problema de criar a pipeline, sub-atomizá-lo, ou seja, dividi-lo até que só restem ações atómicas bastante simples, e a partir dai resolver cada uma delas através do processo de escolher o melhor nó folha de um grafo.

- *Monte-Carlo tree search* que é um método para encontrar o melhor/mais apropriado valor numa árvore. Com este método o problema é encarado ao contrário do anterior. Em vez de reduzir a complexidade da pipeline para tomar decisões, duma forma iterativa, cria a pipeline simples e depois vai aumentando a sua complexidade. Este método encara cada decisão como uma decisão numa árvore binária de busca.
- *Self-play* é um método que segue uma estratégia de reforço de aprendizagem. Em vez de aprender com base num conjunto de dados fixo, é capaz de criar exemplos para treino jogando contra ele. Ou seja, o método encara o problema como se de um jogo se tratasse, usando os dados de treino como se fosse o seu tabuleiro e onde alterar partes da pipeline representassem as suas jogadas. Através dum processo iterativo combina redes neurais com *Monte-Carlo tree search* para através da previsão de probabilidades tomar ações [3].

Uma vez analisadas as possibilidades para a estrutura da pipeline, é importante perceber qual o papel de cada uma das etapas que as constituem, independentemente da sua estrutura. Para isso vai ser feita análise individual de cada uma dessas etapas.

2.5 Recolha de dados

No âmbito da recolha de dados o seu principal objetivo é a aquisição de dados para formar os conjuntos de dados que são usados como base para o treino de algoritmos de ML. Na literatura são apresentadas três abordagens distintas:

- Descoberta de dados – consiste em encontrar dados disponíveis para serem usados, podem ser disponibilizados diretamente por alguma entidade ou estar publicados e disponíveis online;
- Aumento de dados – Complementa a descoberta de dados pois permite adicionar dados externos a um conjunto de dados. Pode ser importante na redução do *overfiting*;
- Geração de dados – Quando não são disponibilizados ou encontrados dados é possível gerá-los duma maneira sintética [10].

2.6 Limpeza de dados

Limpeza de dados é um dos fatores mais importantes no processo do ML e tem como principal função melhorar a qualidade geral dos dados removendo erros. Alguns dos erros mais comuns são valores em falta, valores repetidos ou valores inválidos. É normalmente dividido em duas fases: deteção e posterior reparo dos erros.

Apesar de existirem sistemas de auxílio para os especialistas no processo de limpeza de dados, não há desenvolvimento para a sua automatização. Com isto, apesar da maior parte das frameworks reconhecerem a importância desta fase e a destacarem na sua pipeline, os seus processos são normalmente desenvolvidos *hard-coded*, com uma generalização praticamente nula o que impede a sua automatização, pois o objetivo principal das frameworks desenvolvidas é conseguirem ser completamente agnósticas ao domínio em que se está a trabalhar.

Umas das considerações que é feita, mas não é explorada nesta área, consiste em através da representação do conhecimento de um especialista do domínio, implementar *constraints*, ou uma lógica de deteção de dados com baixa qualidade, que podem ser usados para automaticamente corrigir problemas nos dados [3].

2.7 Engenharia de *features*

Consiste no processo de gerar e selecionar *features* de um determinado conjunto de dados. É um passo muito importante, pois a performance do modelo depende muito das *features* disponíveis. Com boas *features* a performance da pipeline de ML pode ser consideravelmente aumentada. Este processo pode ser dividido em duas sub-tarefas distintas: criação e seleção de *features*.

2.7.1 Criação de features

Criar *features* através de um mapeamento das *features* originais, que é a extração ou a descoberta de conexões entre elas. Como nesta fase é necessário um grande conhecimento de domínio é uma das etapas de maior dificuldade de automatização.

Na figura 5 pode ser observado o processo iterativo de criação de *features* que a maior parte dos mecanismos de automatização segue.

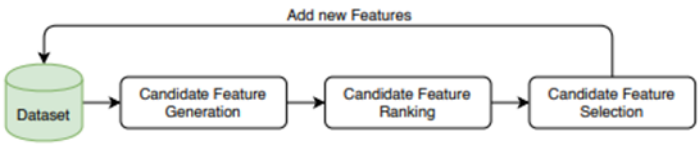

Figura 5 - Esquema para criação de *features* [3]

Com base no conjunto de dados inicial, um conjunto de *features* candidatas são selecionadas, depois disso é-lhes atribuído um ranking e conforme essa avaliação podem ser ou não adicionadas ao conjunto de dados inicial. Este processo é repetido duma maneira iterativa e através destas operações as *features* vão sendo transformadas [3].

Estas operações podem ser de diferentes tipos:

- Unários - transformam um único elemento, aplicando expansões baseadas em regras ou utilizando operadores matemáticos unários como por exemplo, o logaritmo.
- Binários - Combinam duas características através de operações aritméticas básicas. Ao utilizar testes de correlação e modelos de regressão, a correlação entre duas características pode ser expressa como uma nova caraterística.
- Operadores de alta ordem - São geralmente construídos em torno do operador *SQL Group By*, onde todos os registos são agrupados por uma caraterística e depois agregados através de mínimo, máximo, média ou contagem [3].

A criação de *features* pode ser encarado como um problema de escolha do melhor nó de uma árvore, onde o nó raiz representa as *features* originais e onde em cada folha se aplica algum operador resultando num conjunto de *features* diferentes.

2.7.2 Seleção de features

Escolhe um subconjunto das *features*, com o intuito de melhorar a velocidade do treino dos algoritmos e também a sua performance, pois visa remover *features* redundantes ou com pouco valor para os modelos.

Algoritmos como *univariate selection*, *variance treashold*, *feature importance*, *correlation matrices* ou *stability selection* estão já incorporados nas frameworks de AutoML onde são selecionados utilizando algoritmos CASH que vão ser abordados na seção de seleção e otimização de algoritmos [3].

2.8 Seleção do algoritmo e otimização dos híper-parâmetros

Depois de terminada a seleção das *features* é necessário encontrar o algoritmo mais indicado para treinar um modelo. Para isso são abordados dois temas, quais os algoritmos a escolher e quais são as configurações a usar em cada um deles.

Vários algoritmos de classificação são propostos na literatura, como por exemplo, *tree classifiers*, *linear classifiers*, *kernel machines* e, mais recentemente, *deep networks*. Cada um tem as suas vantagens e desvantagens. Cada classificador tem associado diferentes híper-parâmetros, e ao seu conjunto dá-se o nome de configuração. Tradicionalmente, a escolha entre diferentes classificadores e os seus híper-parâmetros é feita por um especialista com base na sua experiência e através dum processo iterativo de tentativa e erro [4].

O processo da escolha da melhor configuração para um algoritmo é o processo que mais consome recursos e tempo, e à medida que se dá a evolução na complexidade dos algoritmos

usados, esse consumo principalmente de recursos computacionais também aumenta muito. Contudo este é o processo que pode surtir mais efeito na eficiência e performance do modelo final, pois tem como principal objetivo aumentar a eficiência do algoritmo o que pode causar uma grande diminuição da sua performance. Com a automatização deste processo, o principal objetivo é conseguir causar um equilíbrio entre a eficácia e a performance do modelo final [4].

Quando o processo de seleção do algoritmo e a sua otimização são combinados simultaneamente, esse processo ganha o nome de *combined algorithm selection and hyperparameter optimization* (CASH). Esta nomenclatura e processo foi introduzido por Thornton et al. [13] e desde então foi adotada pela maioria das frameworks de AutoML [3].

As técnicas para resolver este desafio podem ser divididas em quatro diferentes categorias: estratégias de busca simples (*simple search approaches*), otimização a partir de amostras (*optimization from samples*), método do gradiente (*gradient descent*) e algoritmo guloso (*greedy search*) [4].

2.8.1 Estratégia de busca simples

A pesquisa simples é uma abordagem ingénua, que não faz suposições sobre o espaço de pesquisa. Cada configuração pode ser avaliada de forma independente [4]. Alguns exemplos deste tipo de busca são:

- *Grid Search* – Foi a primeira opção criada para explorar sistematicamente um espaço de pesquisa. Como o nome implica cria uma grelha de configurações e avalia cada uma delas. Apesar de ser fácil de implementar e paralelizar, não funciona bem quando se tem um espaço de pesquisa com muitas configurações pois número de avaliações de funções cresce exponencialmente com o número de híper-parâmetros a estrutura hierárquica dos mesmos não é considerada, levando a muitas configurações redundantes. Na versão tradicional, a pesquisa em grelha não explora o conhecimento das regiões com bom desempenho. Este inconveniente é parcialmente eliminado pela *contracting grid search*. Primeiro, é ajustada uma grelha grosseira e, em seguida, é criada uma grelha mais fina centrada na configuração com melhor desempenho. Este procedimento iterativo é repetido um número definido de vezes, convergindo para um mínimo local.

- *Random Search* - Uma configuração candidata é gerada escolhendo um valor para cada híper-parâmetro de forma aleatória e independente de todos os outros. Os híper-parâmetros condicionais podem ser tratados implicitamente percorrendo o grafo de dependências hierárquicas. A *random search* é fácil de implementar e paralelizar e é adequada para funções sem gradiente com muitos mínimos locais. Embora tendo maior velocidade de convergência do que a *grid search* são ainda necessárias muitas avaliações de funções, uma vez que não é explorado qualquer conhecimento de regiões com bom desempenho. Como as avaliações de funções são muito dispendiosas, a *random search* requer um longo período de otimização [3].

2.8.2 Otimização Derivate-Free

Gera de uma forma iterativa novas configurações com base em amostras previamente avaliadas. É geralmente mais eficiente que métodos de pesquisa simples, contudo não faz suposições específicas sobre o objetivo.

Pode ser dividido em três categorias diferentes: pesquisa heurística, *model-based derivate-free* e *reinforcement learning*.

2.8.2.1 Pesquisa heurística

Os métodos de pesquisa heurística são frequentemente inspirados em comportamentos e fenómenos biológicos. São amplamente utilizados para resolver problemas de otimização não convexos, não suaves ou mesmo não contínuos. A maior parte deles são métodos de otimização baseados em populações e as diferenças entre eles residem na forma de gerar e selecionar populações. Alguns métodos populares são:

- *Particle swarn optimization* - Otimização por enxame de partículas é inspirado no comportamento de comunidades biológicas que exibem tanto comportamento individual como social, exemplos dessas comunidades são bandos de pássaros ou enxames de abelhas;
- Algoritmos evolucionários - São inspirados na evolução biológica. A etapa de geração dos algoritmos evolucionários contém o cruzamento e a mutação. O crossover envolve dois indivíduos diferentes da última geração [4].

2.8.2.2 Model-Based derivate-free

A otimização baseada em modelos constrói um modelo com base nas amostras visitadas. A utilização dos feedbacks do avaliador ajuda-o a gerar novas amostras mais promissoras. Os métodos mais populares deste tipo são: a otimização Bayesiana, a otimização baseada na classificação e a otimização otimista:

- Otimização Bayesiana – Constrói um modelo probabilístico, modelo baseado em árvore ou *deep network*, que estabelece um mapeamento das configurações com a performance com incerteza;
- Otimização baseada na classificação - Com base em amostras anteriores, a otimização baseada em classificação aprende um classificador que divide o espaço de busca em áreas positivas e negativas. Em seguida, novas amostras são geradas aleatoriamente na área positiva, onde é mais provável obter melhores configurações;
- Otimização otimista simultânea - é um algoritmo de otimização *branch-and-bound*. É construída uma estrutura em árvore no espaço de pesquisa em que cada nó folha limita uma subárea [4].

2.8.2.3 Reinforcement learning

Considerado um optimizador geral e eficaz, com *feedback* atrasado, ou seja, que não é aplicado na hora e onde o desempenho real no ambiente é medido pelo avaliador do modelo treinado. Ao contrário dos métodos anteriores, os feedbacks (ou seja, recompensa e estado) não precisam de ser devolvidos imediatamente após a realização de uma ação. Por causa deste comportamento tem um consumo alto de recursos [4].

2.8.3 Método do gradiente

Com base numa função de perda diferenciável, por exemplo, a perda quadrática e a perda logística, os híper-parâmetros contínuos podem ser otimizados. Ao contrário dos problemas de otimização tradicionais cujos gradientes podem ser explicitamente derivados do objetivo, nos problemas AutoML, os gradientes têm de ser calculados numericamente.

O cálculo de gradientes exatos depende da convergência do treino do modelo. Através de gradientes inexatos, os híper-parâmetros podem ser atualizados antes da convergência do treino do modelo, o que torna o método de gradiente descendente mais eficiente.

2.8.4 Algoritmo guloso

A pesquisa gulosa é uma estratégia natural para resolver problemas de tomada de decisão em várias etapas. Segue-se uma heurística que toma decisões localmente ótimas em cada etapa, com o objetivo de encontrar um ótimo global. Por exemplo, para um mercador itinerante, a pesquisa gulosa seleciona a cidade mais próxima em cada etapa da viagem. A pesquisa gulosa não consegue encontrar o ótimo global, mas pode normalmente encontrar um ótimo local que se aproxima do global num tempo razoável [4].

2.9 *Deployment*

Deployment do modelo consiste na integração do modelo treinado na infraestrutura de software necessária para o executar. Esta fase também abrange questões relacionadas com a manutenção e a atualização do modelo. Existe uma disciplina distinta na engenharia, designada *DevOps*, que se centra nas técnicas e ferramentas necessárias para manter e apoiar com êxito os sistemas de produção existentes. Consequentemente, é necessário aplicar os princípios *DevOps* aos sistemas de ML. No entanto, embora alguns se apliquem diretamente, há também uma série de desafios exclusivos do ML.

2.9.1 Integração

A etapa de integração do modelo consiste em duas atividades principais: construir a infraestrutura para executar o modelo e implementar o próprio modelo de forma que possa ser consumido e suportado.

A reutilização de código é um tópico comum na engenharia de software, e o ML pode beneficiar da adoção da mesma mentalidade. A reutilização de dados e modelos pode traduzir-se diretamente em poupanças em termos de tempo, esforço e/ou infraestrutura.

2.9.2 Monitorização

Embora a monitorização seja crucial para a manutenção de qualquer serviço de software, a comunidade de ML está numa fase inicial de compreensão de quais são as principais métricas de dados e modelos a monitorizar e como acionar alarmes do sistema quando se desviam do comportamento normal. A monitorização da evolução dos dados de entrada, da tendência de previsão e do desempenho geral dos modelos de ML é ainda um problema em aberto.

Outra questão de manutenção destacada, que é específica da tomada de decisões baseada em dados, são os ciclos de feedback. Os modelos de ML em produção podem influenciar o seu próprio comportamento ao longo do tempo através de uma reciclagem regular. Ao garantir que o modelo se mantém atualizado, é possível criar um ciclo de feedback em que a entrada do modelo é ajustada para influenciar o seu comportamento.

2.9.3 Atualização

Depois de concluído o *deployment* inicial do modelo, é muitas vezes necessário poder atualizá-lo posteriormente para garantir que reflete sempre as tendências mais recentes dos dados e do ambiente. Existem várias técnicas para adaptar os modelos a novos dados, incluindo a reciclagem regular programada e a aprendizagem contínua. Para além da questão de quando voltar a treinar o modelo para o manter atualizado, há uma questão infraestrutural sobre como entregar o artefacto do modelo ao ambiente de produção.

O *deployment* contínuo para soluções de ML é complicado porque, ao contrário dos produtos de software normais, em que as alterações só ocorrem no código, as soluções de ML sofrem alterações em três eixos: o código, o modelo e os dados.

Embora a atualização seja necessária para manter um modelo atualizado com as flutuações recentes nos dados, também pode causar danos aos utilizadores ou aos sistemas a jusante devido às alterações no comportamento do modelo, mesmo sem causar erros de software óbvios.

2.9.4 Exemplos soluções

Muitas plataformas no mercado oferecem uma experiência completa ao utilizador, tratando de questões como o armazenamento de dados, a reciclagem e a implantação. Alguns exemplos incluem o *AWS SageMaker*, o *Microsoft ML*, o *Uber Michelangelo*, o *TensorFlow TFX* e o MLflow.

Uma plataforma de ML típica incluiria, entre outras características, uma instalação de armazenamento de dados, o alojamento de modelos com API para operações de formação e inferência, um conjunto de métricas comuns para monitorização do estado do modelo e uma interface para aceitar alterações personalizadas do utilizador. Ao oferecerem uma infraestrutura gerida e uma gama de implementações prontas a utilizar para tarefas comuns, essas plataformas reduzem consideravelmente a carga operacional associada à manutenção do modelo de aprendizagem automática na produção. O *Data Version Control* (DVC) é um projeto de código aberto que visa criar uma experiência de controlo de fontes semelhante à do *Git* para conjuntos de dados [14].

2.10 Técnicas para melhorar o desempenho do AutoML

2.10.1 Aproximações multi-fidelidade

O principal problema dos procedimentos AutoML e CASH é o tempo de execução extremamente elevado. Dependendo do conjunto de dados utilizado, a adaptação de um único modelo pode demorar várias horas e, em casos extremos, até vários dias. Consequentemente, o progresso da otimização é muito lento. Uma abordagem comum para contornar esta limitação é a utilização de aproximações multi-fidelidade. Os cientistas de dados utilizam frequentemente apenas um subconjunto dos dados de treino ou um subconjunto das características disponíveis. Ao testar uma configuração neste subconjunto de treino, as configurações com mau desempenho podem ser descartadas rapidamente e apenas as configurações com bom desempenho têm de ser testadas no conjunto de treino completo.

2.10.2 Paragem antecipada

Em contraste com a utilização de apenas um subconjunto dos dados de treino, foram propostos vários métodos para terminar precocemente a avaliação de configurações pouco promissoras. Muitas das estruturas AutoML existentes incorporam a *validação cruzada* ou *holdout* para limitar os efeitos do sobre ajustamento dos modelos aos dados. Uma aproximação bastante simples consiste em abortar o ajuste após a primeira dobra se o desempenho for significativamente pior do que o atual incumbente.

2.10.3 Escalabilidade

Uma estratégia comum para resolver um problema computacionalmente pesado é a paralelização em múltiplos núcleos ou dentro de um cluster. Como o AutoML tem normalmente de ajustar muitos modelos de ML, distribuir diferentes instâncias de ajuste num cluster é uma ideia óbvia.

O escalonamento das tarefas do AutoML para um cluster também permite a introdução de serviços do AutoML. Os utilizadores podem carregar o seu conjunto de dados e espaço de configuração - denominado estudo - para um armazenamento permanente. Os trabalhadores de um cluster testam diferentes configurações de um estudo até que o orçamento se esgote.

2.10.4 Ensemble learning

É possível vários modelos de ML para criar previsões. Dependendo da diversidade dos modelos combinados, a exatidão global das previsões pode ser aumentada significativamente. O custo da avaliação de vários modelos de ML é frequentemente negligenciável, tendo em conta as melhorias de desempenho.

Durante a procura de um pipeline de ML com bom desempenho, as estruturas AutoML criam um grande número de pipelines diferentes. Em vez de produzir apenas a configuração com melhor desempenho, o conjunto de configurações com melhor desempenho pode ser utilizado para criar um conjunto.

2.10.5 Meta-learning

Perante uma nova tarefa de ML desconhecida, os métodos de AutoML começam normalmente do zero para construir uma pipeline. No entanto, um cientista de dados nem sempre começa tudo do zero, ele aprende com tarefas anteriores. A meta-aprendizagem é a ciência de aprender como os algoritmos de ML aprendem. Com base na observação de várias configurações em tarefas de ML anteriores, a meta-aprendizagem cria um modelo para construir configurações promissoras para uma nova tarefa de ML desconhecida, conduzindo a uma convergência mais rápida com menos tentativas e erros.

2.10.6 Refinamento do espaço de pesquisa

Todos os métodos CASH apresentados requerem uma definição do espaço de pesquisa subjacente. Muitas vezes, estes espaços de pesquisa são escolhidos arbitrariamente, sem qualquer validação, o que leva a espaços muito grandes ou a espaços onde faltam regiões com bom desempenho. Em ambos os casos, o procedimento AutoML é incapaz de encontrar resultados ótimos. A meta-aprendizagem pode ser utilizada para avaliar a importância de híper-

parâmetros individuais, permitindo remover híper-parâmetros sem importância do espaço de configuração ou identificar regiões promissoras.

2.10.7 Sugestão de configuração candidata

Muitos procedimentos de AutoML geram configurações candidatas selecionando a configuração com a melhoria esperada mais elevada. A meta-aprendizagem pode ser utilizada como um critério adicional para selecionar configurações candidatas promissoras com base no desempenho previsto. Consequentemente, o risco de avaliações de configurações supérfluas é minimizado.

2.11 Níveis de automatização de frameworks

Para se melhorar a perceção atual pela comunidade do AutoML foi definido um conjunto de níveis para representar os vários sistemas automáticos desenvolvidos. Como se pode observar na figura 6 os níveis são designados de maneira que quanto mais baixo o nível menor é o seu grau de automatização, significa que é preciso de mais interação com o especialista humano enquanto um nível mais alto significa um maior grau de automatização. Para além do grau de automatização o nível também tem uma certa correlação com a capacidade de um especialista de domínio interagir com a framework [15].

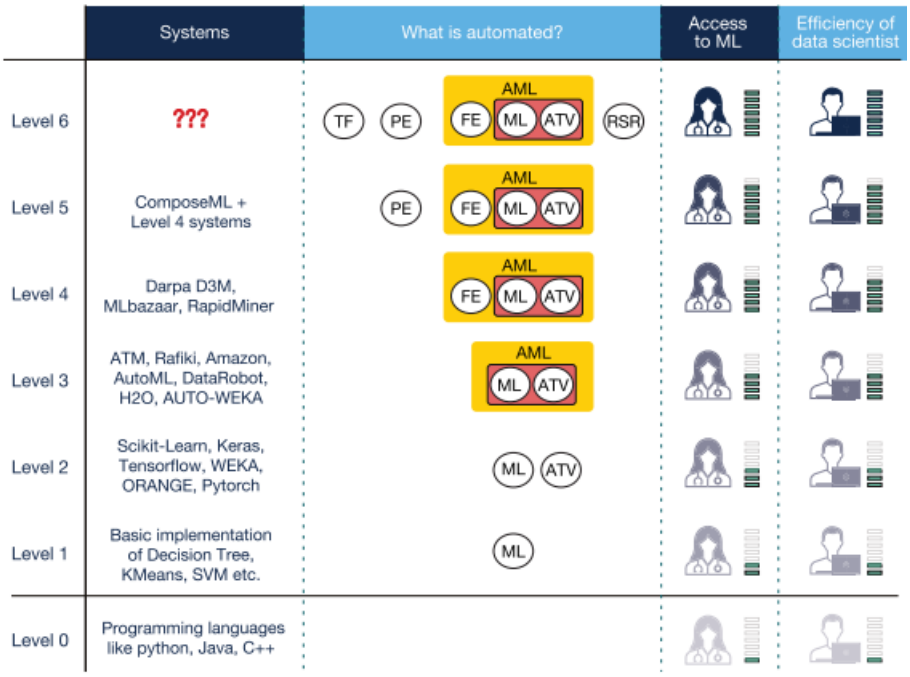

Figura 6 - Níveis de automatização para framework [15]

Analisando os vários níveis propostos temos em primeiro lugar o nível 0. Neste nível não existe qualquer tipo de automatização, sendo todas as ações de implementação executadas à “mão”

pelos especialistas de ML. Um exemplo são os investigadores que estudam e desenvolvem modelos de ML a partir do zero.

No nível 1 são fornecidas implementações básicas dos principais algoritmos de ML. Este é o nível mínimo de automatização suportado por um sistema. As ferramentas deste nível são geralmente desenvolvidas por investigadores e utilizadas por cientistas de dados. Como exemplo têm-se as bibliotecas de Python que fornecem a implementação dos vários algoritmos, como se observa na imagem, arvores de decisão, *support vector machine*, etc. Contudo todo o processo da pipeline tem de ser executado de forma manual pelo cientista de dados.

No nível 2 quando um conjunto de treino e *features* já estão bem definidas, o cientista de dados pode usar ferramentas deste nível, contudo não oferece suporte suficiente para um especialista de domínio conseguir executar nada sem assistência, pois ainda exigem muito trabalho manual. São alguns exemplos o *keras*, *scikit-learn*.

No nível 3 o nível de automatização é aumentado sendo dado aos cientistas de dados algumas ferramentas capazes de retirar algumas etapas manuais do processo, contudo ainda têm a necessidade de fazer algumas operações manualmente. Para além disso o especialista de domínio ainda não tem capacidade de conseguir trabalhar com este nível de abstração. A principal diferença do nível anterior é o processo de seleção e otimização do modelo ser feito de maneira automática.

No nível 4 é acrescentada uma automatização da parte de engenharia de *features*. Permite ao cientista de dados e ao especialista de domínio trabalharem em conjunto para validarem as *features* criadas, poupando assim horas de análises. Um exemplo de ferramenta que é usada pelos cientistas de dados é MLBazaar.

No nível 5 os especialistas de domino começam a conseguir interagir confortavelmente com os sistemas sobrando para os cientistas de dados muito poucas tarefas manuais a realizar. Neste nível é adicionada a capacidade de processar a automatização da engenharia de previsão, permitindo aos utilizadores criarem facilmente um bom conjunto de treino assim como um conjunto de teste e executar essas ações. Há alguns problemas associados a este nível, como por exemplo, a dificuldade em processar informação em falta e através de métricas computacionais preencher esses valores.

No nível 6 é fornecido o maior grau de automatização. Permitem, por exemplo, a médicos e às partes interessadas introduzir diretamente os seus objetivos e poder formular problemas de previsão de forma praticamente automática, recomendando aos utilizadores tarefas de previsão, juntamente com os resultados correspondentes. O maior diferencial neste nível é baseado em duas funcionalidades chave: formulação do problema automaticamente e capacidade de recomendação. Neste nível são apresentados diversos desafios, que para serem resolvidos apresentam a necessidade de um esforço conjunto de vários especialistas de vários domínios distintos.

A maior parte das soluções que se encontram são principalmente focadas entre o nível 3 e 4, onde é possível já se obter um grau relativamente alto de automatização e abstração, permitindo que os processos sejam mais rápidos, eficazes e que consumam menos recursos [15].

2.12 Frameworks de AutoML

2.12.1 Automatização de visualização e limpeza

Para a automatização da visualização e limpeza podem ser encontradas diversas ferramentas:

- Wisteria [15] - permite aos utilizadores especificar planos de limpeza de dados e permite a revisão iterativa do plano. Em cada iteração, apresenta uma recomendação consciente dos custos para melhorar a exatidão do plano, substituindo-o por um novo operador de limpeza;
- Open Refine - permite aos utilizadores limpar e transformar os seus dados através de uma interface baseada num navegador. É instalado na máquina do cliente e o sistema limpa os dados com base na decisão dos utilizadores;
- Trifacta Wrangler - é um serviço grátis na *cloud* que organiza e estrutura os dados de forma automática assim que são importados para o sistema conseguindo sugerir a transformação e agregação. Como a plataforma é gerida na *cloud*, a equipa pode partilhar e programar um fluxo de trabalho e ainda ligar mais dados;
- TIBCO Clarity - é uma ferramenta de preparação de dados a pedido. Deteta tipos de dados e padrões que são posteriormente utilizados para geração automática de meta dados. Categoriza os dados utilizando algumas facetas pré-definidas em padrões de texto. Também fornece distribuição numérica para identificação de variância nos dados e um algoritmo de correspondência difusa configurável para deteção de duplicados;
- Winpure - fornece capacidade de limpeza de dados que determina automaticamente as impurezas nos dados e sugere soluções para diferentes problemas de qualidade dos mesmos. Oferece aos utilizadores diferentes transformações e operações de preparação de dados através de vários modelos. Também permite ao utilizador carregar dados de diferentes tipos de fontes e guardar os dados transformados em vários tipos;
- Data Ladder - fornece uma aplicação de limpeza de dados altamente visual que automaticamente processa a informação dos ficheiros desde o início. Utiliza o reconhecimento de padrões e a lógica difusa para a redução de duplicados e a limpeza de dados não estruturados. Assegura a qualidade fornecendo uma firewall que impede a entrada de dados incorretos de terceiros através de APIs;
- Cloudingo - permite aos utilizadores criar um conjunto de filtros que são utilizados para remover entradas duplicadas em conjuntos de dados. Os utilizadores podem passar os dados por um ou vários filtros que definiram previamente e executá-los manual ou automaticamente. Também permite programar uma tarefa de filtragem que irá curar

os dados em intervalos regulares. Quando vai buscar dados, executa automaticamente uma tarefa de filtragem de base para remover os duplicados dos dados existentes e dos novos dados;

- IBM Infosphere Quality Stage - permite aos utilizadores investigar, limpar e gerir dados. Utiliza regras de qualidade incorporadas e definidas pelo utilizador para tratar os dados automaticamente. Normaliza os dados provenientes de diferentes fontes num formato comum, removendo os duplicados e fundindo vários sistemas numa única visualização [15].

2.12.2 Automatização na engenharia de features

Para resolver o problema da automatização da engenharia de *features* existem também várias soluções disponíveis, tanto livres como comerciais:

- ExploreKit - é uma ferramenta de geração automática de *features* que consiste numa implementação da abordagem baseada na transformação;
- FeatureTools – é uma ferramenta baseada no método Deep Feature Synthesis e pode efetuar eficazmente este processo para dados relacionais e temporais;
- Featexp - é uma ferramenta Python para a exploração de *features* que mostra gráficos e diagramas para ajudar os utilizadores a compreender melhor o espaço das mesmas. Divide as *features* em compartimentos de população iguais e mostra o valor médio da variável dependente (alvo) em cada compartimento, cuja tendência pode ajudar os utilizadores a compreender a relação entre o atributo alvo e a *feature*. Os utilizadores podem comparar as tendências para identificar atributos com ruído, observando o número de mudanças de direção da tendência e a correlação entre a tendência de treino e de teste. Estes gráficos também ajudam o utilizador a fazer a depuração de *features* e a deteção de fugas;
- MLFeatureSelection - Divide a seleção de características em três partes. Primeiro, seleciona uma sequência de dados em bruto que potencialmente conterá a melhor combinação de *features*. Em segundo lugar, cria grupos de *features* a partir dos dados brutos e atualiza-os iterativamente, removendo as menos importantes. Este processo produz um conjunto de *features* importantes. Em terceiro lugar, é escolhido um limiar de coerência e todas as *features* com um valor de coerência mais elevado são selecionadas como as melhores possíveis;
- FeatureHub - fornece uma framework colaborativa instanciada numa plataforma. Ela administra a colaboração entre cientistas de dados, onde os utilizadores podem registar *features* na base de dados, bem como reutilizar *features* escritas por outros utilizadores. O FeatureHub pontua automaticamente as funcionalidades para que os utilizadores possam obter feedback em tempo real sobre as suas funcionalidades.
- Feast - introduz um armazenamento centralizado de *features* que pode funcionar como uma base de dados utilizada por diferentes projetos. Permite que os utilizadores utilizem dados históricos para produzir um conjunto de *features* para treinar modelos

de ML, disponibilizando dados de *features* para modelos em produção através da API de serviço, garantindo que a consistência entre o treino e o serviço é preservada [15].

2.12.3 Automatização de hiperparameterização

Para a procura das melhores configurações de parâmetros num dado algoritmo existem várias ferramentas:

- Hyperparameter Hunter - fornece um *wrapper* para a implementação de algoritmos de ML, simplificando o processo de experiência e otimização de hiperparâmetros. Utiliza automaticamente resultados das experiências anteriores para encontrar híper-parâmetros ótimos. Também elimina códigos "boilerplate" e pode ser facilmente integrado com outras bibliotecas;
- Amazon Sagemaker - utiliza a otimização Bayesiana para descobrir automaticamente híper-parâmetros ótimos. Começa com um modelo substituto e otimiza uma função de aquisição especial chamada "Melhoria Esperada". Em suma, tenta definir e explorar um espaço de híper-parâmetros utilizando a otimização Bayesiana e encontra híper-parâmetros ótimos que atingem a melhor qualidade de modelo;
- Advisor - ferramenta de otimização de híper-parâmetros de caixa negra que suporta vários algoritmos de otimização. Os utilizadores podem escolher qualquer algoritmo com base nas suas preferências;
- Hyperband - utiliza a atribuição adaptativa de recursos e a paragem antecipada para acelerar a pesquisa aleatória e otimizar automaticamente os híper-parâmetros. Os recursos pré-definidos são atribuídos a configurações amostradas aleatoriamente e o problema de otimização dos híper-parâmetros é modelado como um problema de exploração pura, não estocástico, de *infinite-armed bandit*;
- BigML OptiML - utiliza a otimização Bayesiana para a afinar os híper-parâmetros, que se baseia na técnica de otimização SMAC. Tenta sequencialmente um grupo de parâmetros dependendo dos resultados do treino e da avaliação de modelos para esse grupo de parâmetros. Os utilizadores também podem configurar a métrica de otimização da pesquisa de modelos e selecionar entre os modelos recomendados com melhor desempenho;
- Hyperboard - facilita otimização de híper-parâmetros fornecendo uma ferramenta de visualização. Os utilizadores podem treinar modelos para diferentes conjuntos de híper-parâmetros e traçar os desempenhos para cada configuração. Os utilizadores podem então selecionar um conjunto de híper-parâmetros, inspecionando gráficos e selecionando a configuração com melhor desempenho;
- Sigopt - utiliza um conjunto de algoritmos de otimização global e Bayesiana para poder escolher a melhor técnica de otimização para o problema apresentado. Escolhe um algoritmo adequado para explorar o espaço de parâmetros sob a restrição orçamental do utilizador;
- Gentun - utiliza algoritmos genéticos para a otimizar híper-parâmetros. Treinam um modelo através do conjunto de híper-parâmetros definidos pelos seus genes. Utiliza

uma abordagem cliente-servidor para permitir que vários clientes treinem os seus modelos, e o processo de validação cruzada é gerido também por um servidor, assim como a geração e a mutação da descendência;

- Test-tube - é uma biblioteca Python agnóstica que paraleliza a pesquisa de híper-parâmetros em várias CPUs e GPUs. Utiliza a método do gradiente e a pesquisa aleatória como algoritmo de otimização e onde o utilizador pode escolher qualquer um deles [15].

2.12.4 Automatização da exploração de alternativas para modelos

Para o processo de exploração de alternativas para modelos, e onde alguns deles apresentam também otimização por configuração de parâmetros:

- AutoKeras - é um sistema AutoML de código aberto para procurar as melhores arquiteturas neuronais com orientação da técnica de otimização Bayesiana. Para a otimização, utiliza uma rede neural como *kernel* e uma estrutura em árvore como função de aquisição.
- AdaNet - é uma framework baseada em tensorflow que requer uma intervenção mínima de um perito. Fornece uma estrutura para a meta-aprendizagem de melhores modelos durante a aprendizagem da arquitetura de uma rede neuronal. Aprende a estrutura de uma rede neuronal, bem como os seus pesos.
- Cloud AutoML - torna a aprendizagem automática disponível para os utilizadores, mesmo com conhecimentos limitados de aprendizagem automática. O utilizador pode utilizar o AutoML para criar modelos de aprendizagem automática personalizados que são adaptados aos dados do utilizador e integrar esses modelos nas aplicações.
- PocketFlow - fornece uma estrutura que pode comprimir e acelerar modelos de AutoML. Esta estrutura tem dois componentes: aprendizes e optimizadores de híper-parâmetros. Os aprendizes geram um modelo comprimido a partir do modelo original com híper-parâmetros aleatórios, e o optimizador avalia a sua precisão e eficiência. Após algumas iterações, o modelo com melhor resultado é escolhido como modelo final.
- Automl-gs - Pega num ficheiro CSV de entrada, infere o tipo de dados de cada atributo e tenta uma estratégia ETL (extrair, transformar e carregar) para cada atributo para criar um modelo estatístico. As funções de ETL e de construção de modelos são consideradas como scripts de treino e, uma vez treinado o modelo, os resultados do treino são guardados juntamente com os híper-parâmetros. Esta tarefa é repetida até que a contagem de execuções atinja um determinado limiar e o melhor guião seja guardado após cada tentativa.
- MLBox é uma biblioteca de AutoML em Python que pode processar, limpar e formatar dados em linha de forma distribuída. Utiliza métodos robustos para a seleção de *features* e otimiza os híper-parâmetros num espaço de elevada dimensão. Para problemas clássicos e de regressão, fornece modelos preditivos de última geração com interpretação.

- TransmogrifAI - construído em Scala e SparkML para automatizar pipelines de AutoML. Para a seleção de modelos alternativos executa um torneio de diferentes algoritmos e escolhe automaticamente o melhor usando o erro médio de validação. Para lidar com conjuntos de dados desequilibrados, faz uma amostragem apropriada dos dados e recalibra a previsão [15].

2.12.5 Automatização na avaliação

Com intuito de avaliar a performance e eficácia dos modelos existem também algumas ferramentas que podem ajudar na sua automatização:

- OpenML100 - fornece uma avaliação comparativa aprofundada de experiências de ML em 100 conjuntos de dados públicos e permite aos utilizadores reproduzir os resultados. Recolhe experiências criadas e avaliam-nas com diferentes ferramentas. Durante a avaliação, cada algoritmo é avaliado com um conjunto de híper-parâmetros otimizado e 200 iterações de pesquisa aleatória;
- OpenML-CC18 – Avalia algoritmos em 73 diferentes conjuntos de dados. Para cada um executa 1000 configurações aleatórias do classificador de floresta aleatória, com um tempo de treino limitado a um máximo de três horas [15].

2.13 Sistemas de AutoML

Por último vão ser apresentadas algumas ferramentas que têm capacidade de realizar a automatização da pipeline do ML:

- DotData - fornece um sistema de automação para apoiar pipelines de aprendizagem automática. Prepara automaticamente os dados fornecidos para a engenharia de features, explora o espaço e seleciona as mais relevantes. Obtém um modelo de previsão preciso explorando automaticamente os algoritmos mais avançados e ajustando os seus híper-parâmetros.
- TPOT - é uma ferramenta de otimização baseada em árvores para automatizar pipelines de AutoML. Otimiza a pipeline utilizando *Gaussian Process* (GP), maximizando a precisão da classificação para um determinado conjunto de dados de aprendizagem supervisionada. Utiliza também GP para desenvolver as sequências de operadores da pipeline e os parâmetros para maximizar a precisão da classificação da mesma.
- IBM AutoAI - fornece um sistema automatizado de construção e avaliação de modelos de aprendizagem automática. Limpa os dados em bruto e categoriza-os com base no seu tipo. Testa os algoritmos candidatos num pequeno subconjunto de dados para os classificar. Gradualmente, aumenta o tamanho do subconjunto para os algoritmos mais promissores. A um nível elevado, explora o espaço de *features*, maximizando a precisão do modelo através da aprendizagem por reforço;

- Auto-Sklearn – juntou a automatização da seleção do algoritmo de aprendizagem e dos seus híper-parâmetros. Fornece ainda um conjunto de ferramentas de automatização chamado Auto-Sklearn. Usa principalmente otimização Bayesiana, *meta-learning* e *ensemble learning*;
- Azure Machine Learning – usa otimização Bayesiana e uma filtragem colaborativa para resolver a tarefa de *meta-learning* do AutoML. Para descobrir qual o algoritmo vai explorar a seguir, acrescenta um modelo probabilístico juntando assim o fato incerteza ao processo de escolha;
- Google Cloud Ai platform – tem a capacidade de testar algoritmos desenvolvidos em conjuntos de dados fornecidos pelos utilizadores. Se algum desses algoritmos desenvolvidos tiver boa performance, é selecionado, caso não tenha podem criar uma aplicação para treinar um algoritmo. Esta plataforma usa otimização Bayesiana para exploração das configurações com híper-parâmetros;
- Amazon AWS servisse H20.ai – é a solução da Amazon para AutoML. Para além de suportar automatização *end-to-end*, fornece também funcionalidades para visualização e para ML;
- ATM – é um serviço colaborativo para construir pipelines de classificação otimizadas. Usa uma estrutura fixa com um modulo opcional de PCA, e de normalização, seguido pela automatização da seleção dos algoritmos e seus híper-parâmetros. Todos os algoritmos são baseados em scikit-learn. Uma particularidade é o uso do ModelHub, capaz de armazenar informações sobre vários conjuntos de dados e basear as suas escolhas com base nas avaliações da performance guardadas, criando assim uma base que em teoria já possui uma melhor performance [15].

2.14 Conclusões da revisão

Com a revisão efetuada nesta dissertação tinha-se como objetivo aprofundar os conhecimentos sobre os processos do ML no geral, depois disso apresentar algumas técnicas que podem ser usadas para a sua automatização. O tópico seguinte abordado foi uma proposta de classificação em níveis para uma solução de AutoML seguida duma apresentação de diversos sistemas e frameworks que podem ser usados nas diversas partes da pipeline de ML e também no seu conjunto geral como sistema.

Finalizada a revisão percebe-se que maior parte dos problemas em que há mais tipos de abordagens para a sua resolução são problemas de busca dum determinado espaço, como se pode observar na seleção de híper-parâmetros e modelo, ou até na engenharia de *features*.

Problemas cuja ligação seja muito grande ao domínio, tornando-os muito específicos tornam-se pouco explorados pois é difícil produzir uma solução agnóstica ao problema nestes casos havendo menos propostas de solução.

Relativamente ao estudo de técnicas para tentar melhorar a performance pode também dizer-se que estão espelhados no desenvolvimento das frameworks que propõe as soluções o que

indica e faz sobressair a sua importância devido às dificuldades que problemas complexos podem gerar às soluções, podendo rapidamente escalar em termos de recursos computacionais, de tempo consumido e dinheiro perdido.

3 Definição dos Requisitos e Arquitetura da Solução

Neste capítulo vai ser feita em primeiro lugar a especificação da engenharia de requisitos, em segunda lugar uma análise das ferramentas/abordagens para a resolução do problema, em terceiro lugar vai ser explicada a arquitetura elaborada para a solução e por último vai ser abordado o fluxo de dados no sistema. Com a análise deste capítulo pretende-se que todas as escolhas efetuadas e desenhadas para o sistema sejam mais facilmente entendidas.

3.1 Engenharia de Requisitos

Nesta seção vai ser feita uma análise dos requisitos levantados para a aplicação de modo a responder da maneira mais completa aos objetivos definidos à priori. Esta análise vai ser dividida em três partes, sendo a primeira a análise dos requisitos funcionais, uma segunda parte, os requisitos não funcionais e por último a identificação do principal ator para o sistema desenvolvido.

3.1.1 Ator do sistema e os principais benefícios do seu uso

O principal ator do sistema é a entidade (utilizador ou sistema externo) que interage mais frequentemente com a aplicação ou que desempenha o papel mais crucial na mesma. Neste caso o ator principal pode ser considerado um cientista de dados ou um engenheiro de ML uma vez que são os utilizadores que possam ter mais interesse em usar a aplicação. Contudo com o grau de automatização implementado, que retira qualquer necessidade de programação, a aplicação permite que qualquer utilizador, com menos conhecimento da área de ML, possa usufruir de todas as capacidades da aplicação, identificando como única diferença entre quem tem conhecimento de área e não tem, a maior consciência sobre as opções a escolher e sobre os resultados apresentados.

Como principal propósito ao utilizar a aplicação o ator procura uma maneira de usufruir das várias configurações oferecidas pela framework usada para de maneira automatizada e otimizada realizar o treino de algoritmos de ML e o posterior *deployment* dos modelos desenvolvidos. Para além de procurar obter modelos com a máxima eficiência e precisão, evitando o repetitivo e iterativo processo clássico de ML, reduzindo o seu esforço manual nessas etapas, aproveita todas as capacidades do sistema para reduzir o tempo gasto e os recursos no processo do ML. Tem também o benefício de obter automaticamente informações adicionais sobre o processo de treino e o desempenho avaliado por várias métricas o que ajuda na obtenção de insights e posterior escolha do modelo que mais se adequa para ser disponibilizado para integração com outros sistemas através do processo de *deployment*.

3.1.2 Requisitos funcionais

Os requisitos funcionais consistem nas funcionalidades e comportamentos específicos que o sistema deve possuir de modo a atender os objetivos definidos. No contexto desta dissertação foram identificados os seguintes requisitos:

1. Importação de dados:
    a. Descrição: Aplicação deve permitir a importação de dados, em formato CSV, de uma forma direta e simplificada.
    b. Objetivo: Garantir que os dados necessários para executar o processo de ML sejam facilmente carregados e disponibilizados para a execução da aplicação.
2. Pré-processamento de dados:
    a. Descrição: Sistema deve fornecer funcionalidade de pré-processamento de dados, tanto limpeza como transformação, isto inclui por exemplo, tratamento de valores nulos, normalização de dados ou até codificação de variáveis categóricas.
    b. Objetivo: Preparar os dados para serem utilizados nos processos subsequentes da pipeline de ML com a premissa que tendo dados melhores, os resultados produzidos serão melhores.
3. Seleção de modelos e híper-parametrização:
    a. Descrição: Aplicação deve efetuar automaticamente a seleção de algoritmos de ML adequados e os respetivos híper-parâmetros que resultem nos melhores resultados de desempenho para um determinado conjunto, medido através das métricas mais adequadas.
    b. Objetivo: Automatizar e adequar o processo de escolha, de modo a maximizar a eficiência e precisão.
4. Treino e validação de modelos:
    a. Descrição: Sistema deve permitir efetuar o treino dos modelos, aplicando alguma estratégia de validação, como por exemplo, validação cruzada (*cross-validation),* ou *holdout*, para evitar *overfiting*.
    b. Objetivo: Garantir que o modelo treinado seja generalizável e robusto.
5. Avaliação de modelos e análise de resultados:
    a. Descrição: Aplicação deve permitir avaliar o desempenho dos modelos utilizando as métricas mais apropriadas, como por exemplo, *accuracy*, *precision*, *recall*. Para além disso deve também fornecer outras informações extra, como por exemplo, informações sobre o processo de treino ou importância das *features*.
    b. Objetivo: Fornecer feedback sobre a qualidade do modelo e apoiar o processo de tomada de decisão.
6. Implementação automatizada da pipeline:
    a. Descrição: Sistema deve permitir a criação e execução da pipeline completa de ML, integrando todas as etapas desde a importação de dados até à validação dos modelos e permitindo algumas configurações.

   b. Objetivo: Tornar o processo de ML configurável, contudo mantendo o nível de automatização.
7. Geração de relatório de treino:
   a. Descrição: Aplicação deve fornecer capacidade de documentar detalhes e informações sobre o treino e o próprio modelo, por exemplo, informações sobre o treino ou métricas de avaliação associadas.
   b. Objetivo: Assegurar que os modelos gerados contenham informação de forma que o processo de partilha e revisão do mesmo seja mais fácil e eficaz.
8. *Deployment* do Modelo:
   a. Descrição: Sistema deve suportar o *deployment* do modelo treinado.
   b. Objetivo: Facilitar a disponibilização dos modelos de ML treinados para uso em produção, permitindo que possam ser usados para previsões em tempo real.

3.1.3 Requisitos não-funcionais

Os requisitos não funcionais descrevem as qualidades e restrições do sistema. Nesta dissertação foram identificados os seguintes requisitos:

1. Escalabilidade:
   a. Descrição: Aplicação deve ser capaz de lidar com grandes volumes de dados e aumentar sua capacidade de processamento conforme necessário
   b. Objetivo: Garantir que o sistema possa crescer e se adaptar às necessidades do utilizador ou ao crescente volume de dados
2. Desempenho:
   a. Descrição: Sistema deve ser capaz de efetuar o processo de execução da pipeline de forma eficiente, minimizando o tempo de execução.
   b. Objetivo: Reduzir o tempo necessário para obter os resultados, especialmente com o aumento da complexidade ou tamanho dos dados.
3. Usabilidade:
   a. Descrição: Aplicação deve ser intuitiva e fácil de usar, principalmente para utilizadores com conhecimento limitado sobre ML
   b. Objetivo: Facilitar o uso da aplicação, permitindo que utilizadores com menos capacidades técnicas possam também tirar todo o proveito das funcionalidades oferecidas.
4. Manutenibilidade
   a. Descrição: Aplicação desenvolvida de forma modular e com boa documentação, de modo a facilitar futuras atualizações ou manutenção
   b. Objetivo: Assegura que o sistema pode ser facilmente modificado e atualizado conforme novas necessidades ou tecnologias aparecem
5. Compatibilidade:
   a. Descrição: Sistema deve ser compatível com diferentes sistemas operacionais e deve suportar integração com bibliotecas de ML e frameworks de AutoML

   b. Objetivo: Maximizar a flexibilidade e interoperabilidade da aplicação
6. Suportabilidade:
   a. Descrição: Aplicação deve apresentar um alto grau de configurabilidade, mantendo a simplicidade de uso da mesma e a seu grau de automatização.
   b. Objetivo: Assegurar que tanto pessoas com capacidade técnicas como com capacidade menos técnicas tirem todo o proveito fornecido pelas frameworks usadas no desenvolvimento.

3.2 Análise de ferramentas/abordagens para resolução do problema

Nesta seção vai ser feita uma análise de algumas das soluções que foram apresentadas na seção da revisão literária. Em primeiro lugar vão ser apresentados critérios para ajudar a efetuar a análise da melhor abordagem para a resolução do problema proposto para esta dissertação. Depois vai ser feita uma pequena análise de algumas das ferramentas em vista aos critérios apresentados e por fim uma vai ser aplicado um método de análise multicritério para ajudar e justificar da melhor maneira a ferramenta mais adequada para utilizar para a resolução do problema.

3.2.1 Critérios de avaliação a usar

Aqui vão ser apresentados critérios para ajudar a efetuar a análise da melhor abordagem para a resolução do problema proposto para esta dissertação que é determinar qual a melhor framework de AutoML é mais adequada para responder aos requisitos definidos. Vão ser usados os seguintes critérios de inclusão ou exclusão:

- Cobertura da Pipeline de Machine Learning: Consiste na capacidade da framework de cobrir todas as fases da pipeline, desde o pré-processamento dos dados até ao deployment do modelo;
- Flexibilidade e Customização: Capacidade da framework oferecer diferentes parâmetros para configurações, escolha de algoritmos, ou integrar métodos personalizados;
- Escalabilidade: Habilidade da framework lidar com grande volume de dados e a sua eficiência em ambiente de produção
- Facilidade de Uso e Documentação: Curva de aprendizagem associada à framework, qualidade da documentação e a comunidade de suporte;
- Integração com outras ferramentas: Como a framework se integra com outras ferramentas ou tecnologias existentes dentro do ecossistema de ML;
- Manutenção e suporte: Frequência de atualização, a atividade e suporte da comunidade
- Desempenho: Resultados produzidos em termos de métricas de avaliação dos algoritmos treinados e o tempo e recursos usados para o mesmo;

- Custo: Caso haja algum tipo de recurso financeiro associado ao uso da mesma, por exemplo pagamento de subscrição.

3.2.2 Análise das ferramentas

Para conseguir responder a cada um destes critérios foram feitos alguns testes e experiências usando cada uma das opções disponíveis, tendo a maior parte das soluções apresentadas na revisão da literatura sido excluídas pois eram soluções pagas e esse foi um dos critérios definidos para efetuar uma primeira triagem. De entre as soluções *open-source* restantes foram escolhidas como possíveis soluções as frameworks: Auto-sklearn, TPOT, AutoKeras, e ATM.

O passo seguinte consistiu em analisar as quatro soluções selecionadas:

- Auto-sklearn
    - Cobertura da Pipeline: Cobre a maior parte da pipeline, automatizando as etapas desde o pré-processamento até seleção de modelos e a sua híper-parametrização;
    - Flexibilidade e Customização: Apresenta um grau de flexibilidade e customização relativamente alto, contudo seguindo uma abordagem mais fiel às práticas de ML mais tradicionais;
    - Escalabilidade: Produz bons resultados para conjuntos de dados de tamanho moderado, mas pode apresentar algumas limitações em conjuntos de dados muito grandes;
    - Facilidade de Uso e Documentação: Boa documentação e curva de aprendizagem acentuada para quem tem experiência com scikit-learn;
    - Integração com outras Ferramentas: Muito fácil de integrar com as ferramentas de scikit-learn, mas também com bom grau de adaptação para outras ferramentas Python;
    - Manutenção e Suporte: Comunidade bastante ativa e com atualização regular de versões;
    - Desempenho: Desempenho sólido em *benchmarks*, que depende da qualidade dos algoritmos disponíveis no scikit-learn.
- TPOT
    - Cobertura da Pipeline: Boa cobertura da pipeline uma vez que trata o processo como um todo, desde o pré-processamento até à seleção de modelos e os seus híper-parâmetros;
    - Flexibilidade e Customização: Bom grau de customização uma vez que é possível ajustar os operadores genéticos e pipelines gerados;
    - Escalabilidade: Apresenta bom desempenho em conjuntos relativamente grandes, mas devido à sua natureza evolutiva de otimização pode ser computacionalmente caro;
    - Facilidade de Uso e Documentação: Documentação sólida, com curva de aprendizagem razoavelmente simples para quem está habituado ao scikit-learn;

    - Integração com outras Ferramentas: Boa integração com scikit-learn, mas limitado fora do ecossistema;
    - Manutenção e Suporte: Comunidade ativa que trata tanto do suporte como da manutenção;
    - Desempenho: Muito bom a gerar pipelines otimizadas, mas com tempo de execução que tende a ser longo.
- AutoKeras
    - Cobertura da Pipeline: Focado principalmente em *deep learning*, oferece cobertura para modelação e otimização de redes neuronais, mas com um suporte limitado para métodos de pré-processamento tradicional;
    - Flexibilidade e Customização: Grande grau de flexibilidade em termos de arquitetura de redes neuronais, permitindo ajustes detalhados;
    - Escalabilidade: Apresenta um grau de escalabilidade muito grande principalmente em ambientes com recursos computacionais robustos;
    - Facilidade de Uso e Documentação: Aprendizagem fácil para quem tem conhecimento de keras, mas relativamente complicado para iniciantes;
    - Integração com outras Ferramentas: Fácil integração com ecossistema de *deep learning* do keras e TensorFlow;
    - Manutenção e Suporte: Comunidade bastante ativa que fornece tanto manutenção como suporte;
    - Desempenho: Excelente opção para tarefas de *deep learning*, mas não a melhor solução no que toca a algoritmos mais tradicionais.
- ATM
    - Cobertura da Pipeline: Oferece uma plataforma para *auto-tuning* de modelos, mas com relativamente nenhuma opção para o resto da pipeline;
    - Flexibilidade e Customização: Permite customização dos parâmetros de otimização, mas é menos flexível com o suporte de diferentes algoritmos;
    - Escalabilidade: Focado em execuções em paralelo, o que vai depender profundamente da infraestrutura disponível;
    - Facilidade de uso e Documentação: Documentação é mais escassa quando comparada às outras opções apresentadas;
    - Integração com outras Ferramentas: Integração é possível com outras ferramentas de ML, mas os processos para isso não são tão robustos;
    - Manutenção e Suporte: Tanto a comunidade como a manutenção apresentam ser menos ativas;
    - Desempenho: Bom desempenho para a escolha de híper-parâmetros, mas com algum atraso relativamente às outras opções nos outros aspetos da pipeline.

Desta pequena análise comparativa é possível excluir duas das soluções à priori da aplicação do método de análise multicritério. Tanto o AutoKeras como o ATM apresentam algumas características interessantes, mas quando postas em comparação com as outras duas soluções parecem ter alguns pontos negativos. Relativamente ao AutoKeras, como identificado é uma

ferramenta muito forte e consolidada para a exploração da área de automatização de processos para problemas onde se pretenda aplicar *deep learning* como solução, pelo que decidiu-se excluir essa ferramenta. Relativamente ao ATM, provou-se como uma solução muito eficaz para problemas de automatização da híper-parametrização, dos algoritmos que têm disponíveis na sua gama de opções, mas não oferece um suporte para toda a extensão da pipeline o que a deixa relativamente afastada dos critérios definidos.

3.2.3 Escolha da ferramenta

Com esta pequena análise, a segunda fase de seleção foi aplicada e restaram duas soluções: Auto-sklearn e TPOT. Para ajudar a tomar a decisão sobre qual a solução é a mais apropriada para o problema foi então aplicado um método de análise multicritério, neste caso TOPSIS (*Technique for Order of Preference by Similarity to Ideal Solution*) [16]:

1. Definição de critérios e atribuição de pesos
    a. Como referido anteriormente os critérios são:
        i. Cobertura da Pipeline (C1)
        ii. Flexibilidade e Customização (C2)
        iii. Escalabilidade (C3)
        iv. Facilidade de Uso e Documentação (C4)
        v. Integração com outras Ferramentas (C5)
        vi. Desempenho (C6)
        vii. Custo (C7)

    b. Os pesos atribuídos a cada critério são:
        i. C1 = 0,2
        ii. C2 =0,15
        iii. C3 =0,1
        iv. C4 =0,15
        v. C5 =0,15
        vi. C6 =0,2
        vii. C7 =0,05

2. Com esta informação é possível construir a matriz de decisão, onde para cada uma das soluções possíveis vai ser dada uma avaliação entre 1 e 10, sendo 10 o mais alto e 1 o mais baixo, a cada um dos critérios.

Tabela 1 - Matriz de decisão para TOPSIS

Critério	Peso	Auto-sklearn	TPOT
C1	0,2	8	9
C2	0,15	9	7
C3	0,1	8	7
C4	0,15	9	8
C5	0,15	7	7
C6	0,2	8	8
C7	0,05	10	10

3. Depois de ter a matriz de decisão completa é efetuada a sua normalização através da aplicação da equação 1:

Equação 1:

$$P_{iJ} = \frac{x_{iJ}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}}$$

Onde x_{iJ} é o valor original e P_{iJ} é o valor normalizado.

4. Passo seguinte é a construção da matriz ponderada, aplicando a equação 2:

Equação 2:

$$v_{iJ} = w_{J^*} R_{i_J}$$

Onde são multiplicados os valores normalizados pelos pesos dos critérios.

5. O quinto passo consiste na determinação das soluções ideal e anti-ideal que consiste em ver qual o menor e o maior valor ponderado para cada critério.

6. O sexto passo é o cálculo das distâncias das soluções ideal e anti-ideal, através da distância euclidiana da solução ideal e da anti-ideal para cada uma das alternativas, ou seja, aplicação das equações 3 e 4:

Equação 3:

$$D_i^+ = \sqrt{\sum_{j=1}^{n} \left(v_{iJ} - v_j^+\right)^2}$$

Equação 4:

$$D_i^- = \sqrt{\sum_{j=1}^{n} \left(v_{iJ} - v_j^-\right)^2}$$

7. O sétimo passo consiste no cálculo do encore de preferência, através da equação 5:

Equação 5:

$$C_i = \frac{D_i^-}{D_i^+ + Di^-}$$

8. Por último com base neste último valor, é classificada como melhor escolha a alternativa que tenha o C_i mais alto que neste caso é o Auto-sklearn.

De modo a aplicar a lista de passos descritos, foi implementado o código presente na figura 7.

```
import numpy as np

# Pesos associados aos critérios
new_criteria_weights = np.array([0.2, 0.15, 0.1, 0.15, 0.15, 0.2, 0.05])

# Notas atribuidas para Auto-sklearn e TPOT
new_decision_matrix = np.array([
    [8, 9, 8, 9, 7, 8, 10],  # Auto-sklearn
    [9, 7, 7, 8, 7, 8, 10]   # TPOT
])

# Calcular a normalização da matriz de decisão
new_norm_matrix = new_decision_matrix / np.sqrt((new_decision_matrix**2).sum(axis=0))

# Calcular a matriz ponderada
new_weighted_matrix = new_norm_matrix * new_criteria_weights

# Calcular as soluções: ideal (A+) e anti-ideal (A-)
new_ideal_solution = new_weighted_matrix.max(axis=0)
new_anti_ideal_solution = new_weighted_matrix.min(axis=0)

# Calcular as distâncias Euclidianas para a solução ideal (D+) e anti-ideal (D-)
new_distances_to_ideal = np.sqrt(((new_weighted_matrix - new_ideal_solution) ** 2).sum(axis=1))
new_distances_to_anti_ideal = np.sqrt(((new_weighted_matrix - new_anti_ideal_solution) ** 2).sum(axis=1))

# Calcular o escore de preferência (C_i)
new_preference_scores = new_distances_to_anti_ideal / (new_distances_to_ideal + new_distances_to_anti_ideal

# Resultados: novos escores de preferência e classificação
print(new_preference_scores, ["Auto-sklearn", "TPOT"][np.argmax(new_preference_scores)])
```

Figura 7 - Cálculos para aplicação TOPSIS

Para os cálculos efetuados através do código presente na figura 7, vão ser mostrados os valores que vão sendo obtidos a cada uma das operações efetuada. Em primeiro lugar na figura 8, vai ser observada a matriz normalizada:

```
Matriz normalizada:
                    0       1       2       3       4       5       6
Auto-sklearn  0.6667  0.7809  0.7593  0.7428  0.7071  0.7071  0.7071
TPOT          0.7500  0.6065  0.6644  0.6614  0.7071  0.7071  0.7071
```

Figura 8 - Matriz normalizada

Em segundo lugar na figura 9 vai ser observada a matriz ponderada, obtida a partir da matriz normalizada:

```
Matriz ponderada:
                    0       1       2       3       4       5       6
Auto-sklearn  0.1333  0.1171  0.0759  0.1114  0.1061  0.1414  0.0354
TPOT          0.1500  0.0910  0.0664  0.0992  0.1061  0.1414  0.0354
```

Figura 9 - Matriz ponderada

Em terceiro lugar vai ser calculado a solução ideal para D_i^+ e para D_i^- através das figuras 10 e 11 respetivamente:

```
Solução ideal (A+):
       0
0  0.15
1  0.12
2  0.08
3  0.11
4  0.11
5  0.14
6  0.04
```

Figura 10 - Solução ideal D_i^+

```
Solução anti-ideal (A-):
       0
0  0.13
1  0.09
2  0.07
3  0.10
4  0.11
5  0.14
6  0.04
```

Figura 11 - Solução ideal para D_i^-

Com base nestes valores vai ser calculada a distância euclidiana como se pode observar na figura 12:

```
Distâncias Euclidianas (Ideal e Anti-Ideal):
               D+ (Ideal)  D- (Anti-Ideal)
Auto-sklearn       0.0167          0.0262
TPOT               0.0262          0.0167
```

Figura 12 - Distâncias euclidianas

Por último é calculado o encore com base nestas distâncias. Para os dados presentes na matriz os valores de encore consistiram nos seguintes:

- Encore de preferência para Auto-sklearn = 0,6481
- Encore de preferência para TPOT = 0,3519

Por isso a solução escolhida foi o Auto-sklearn. Para além de ser a solução recomendada como mais perto da solução ideal pelos critérios definidos e os seus pesos associados, com base em algumas experiencias elaboradas para tentar perceber de uma maneira simples o funcionamento base de cada uma destas ferramentas, percebeu-se que para este caso a solução mais indicada se mostrou mesmo ser o Auto-sklearn uma vez que ele dá uma maior grau de liberdade para se alterar e manipular cada uma das etapas da pipeline em separado, sendo que o TPOT é mais fechado para esse tipo de soluções.

3.3 Análise da arquitetura da solução

Em primeiro lugar vai ser apresentada uma visão geral do design planeado para responder ao problema. Na segunda parte vai ser analisado cada um dos componentes apresentados individualmente, para explicar o principal objetivo de cada um deles. Na terceira seção faz-se análise do fluxo de dados para ajudar a perceber o funcionamento planeado para a aplicação.

Por último, algumas considerações mais gerais sobre como se planeia responder a alguns dos requisitos não-funcionais.

3.3.1 Visão geral

De modo a responder aos requisitos apresentados anteriormente, procurou-se criar um design para a solução o mais simples e modular possível para fornecer ao utilizador a mais otimizada, intuitiva e simples interação com a aplicação, de modo a permitir que o mesmo tenha todo o proveito das capacidades da Framework usada como núcleo da aplicação para treino dos modelos, mantendo-se longe da necessidade de escrever qualquer código ou analisar conceitos específicos relacionados à Framework usada. Esta interação facilitada surge através do uso de uma interface gráfica que torna todo o processo mais user-friendly, e que para além disso permite que pessoas com menos conhecimento específico da área possam também usufruir das funcionalidades da mesma, por ser bastante intuitiva, fazendo com que através de pouquíssimos cliques o utilizador seja capaz de passar da escolha do ficheiro que quer usar como fonte de dados para ter um modelo treinado e pronto para efetuar previsões em dados enviados por outras aplicações, ou serviços. Em seguida para dar um melhor entendimento deste design proposto vai ser apresentado um diagrama de arquitetura de alto nível.

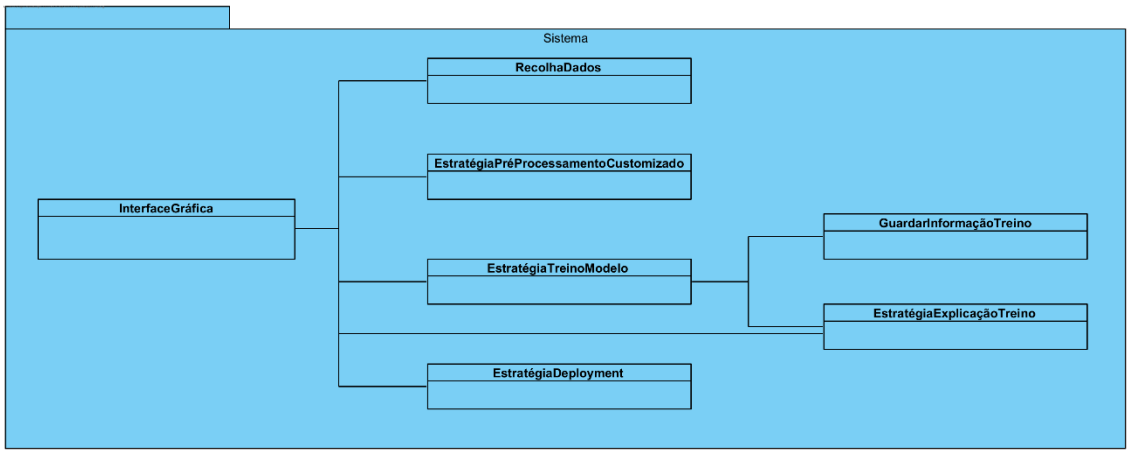

Figura 13 - Visão geral arquitetura do sistema

Como ser pode observar está presente na figura 13 uma representação da arquitetura projetada para o sistema onde se podem identificar sete componentes com funções diversas, que vão ser analisadas individualmente na próxima seção, mas que ao interagirem formam o sistema. Por alto, todo o fluxo da aplicação é controlado pela interface gráfica, pois como se pode observar esse componente está praticamente ligado a todos os outros componentes, pois para além de servir como ponte de ligação para o utilizador passar de uma fase para outra, também serve para coleta de dados relevantes como por exemplo, as configurações a usar nas diversas etapas. A partir da interface gráfica o primeiro componente a ser invocado é a RecolhaDados. Uma vez efetuado todas as ações definidas nesse componente é dada a opção de ser apresentado ao utilizador um componente que é a

EstratégiaPréProcessamentoCustomizado. Caso o utilizador não queria efetuar qualquer tipo de pré-processamento customizado, ele pode simplesmente através da interface escolher que não quer fazer e segue diretamente para a EstratégiaTreinoModelo, que neste caso segue com a informação que o utilizador não quis efetuar nenhum tipo de pré-processamento então deixa-se ativo o modulo de pré-processamento do AutoML. Caso utilizador pretenda trabalhar com algum tipo de pré-processamento mais customizado, ou simplesmente pretenda querer saber quais os métodos que foram usados para o mesmo, pode estabelecer as suas configurações para diversas formas de pré-processamento e ele vai ser automaticamente feito, passando em seguida também para a EstratégiaTreinoModelo. Neste componente como o nome indica é aplicada a estratégia para realizar o treino do modelo, sendo interno a este componente outros dois que são GuardarInformaçãoTreino e EstratégiaExplicação de treino. A primeira serve para guardar as informações relativas ao modelo quando acaba o processo de treino e o segundo para apresentar ao utilizador algumas explicações sobre o processo de treino que terminou, como por exemplo, as métricas de avaliação entre outros detalhes. Por último, o componente EstratégiaDeployment, permite realizar o deployment de um modelo.

3.3.2 Componentes principais da Arquitetura

Para se ter uma melhor perceção de qual a utilidade individual de cada um dos componentes apresentados da figura 13, vai ser feita uma análise mais detalhada das suas funções e principais objetivos.

3.3.2.1 Interface utilizador

Desenvolvida com recurso a um módulo de Python chamado tkinter, permite a criação de interfaces gráficas simples, mas que para esta situação se torna bastante útil, uma vez que permite controlar todo o fluxo de operações, estados e etapas do sistema, fazendo com que a transição de componente para componente e a passagem de dados entre os mesmos seja feita de maneira simples e eficaz. A razão pela qual foi considerado benéfico para o sistema a implementação de uma interface gráfica foi para conseguir aumentar o alcance da mesma, pois como um dos objetivos é permitir a pessoas com menos conhecimento técnico possam também usufruir das tecnologias, aumentando assim o grau de automatização, pois com esta retira-se a necessidade do utilizador ter de implementar qualquer linha de código, ou até ter de realizar qualquer ação sem o apoio do sistema desenvolvido.

No seguimento da mesma ideia a interface gráfica também ajuda a dar a conhecer as diversas alternativas de configurações disponíveis e a efetuar a sua recolha, tanto no módulo de EstratégiaTreinoModelo como no EstratégiaProcessamentoCustomizado. Tornando assim o modo de customização através das configurações mais simples e intuitivo. Neste tipo de interação a interface apresenta de uma forma simples ou em forma de campo para preencher valores inteiro, ou em campos, com uma lista de opções, através de um dropdown menu, as opções de configuração disponíveis em cada caso, para que utilizador faça as escolhas que quiser da forma mais simples, rápida e apoiada possível.

3.3.2.2 *Importar dados*

Componente projetado com dois principais objetivos. O primeiro, ler e guardar o conjunto de dados indicado pelo utilizador através da interface gráfica. Em segundo lugar, fazer uma análise desse conjunto de dados, e aplicar um filtro sobre os campos, ou seja, atributos, que estão presentes neste conjunto de dados, para que só os atributos que sejam apropriados para problemas de classificação sejam escolhidos como atributo alvo e sejam apresentados ao utilizador. Desta forma tira-se a responsabilidade ao utilizador de ter de fazer essa distinção, permitindo assim a utilizadores com menor conhecimento na área passar por esta fase sem dificuldades. A regra para definir se o atributo é considerado adequado ou não, consiste em analisar em primeiro lugar se o valor é um número ou não, caso seja um valor numérico, é necessário verificar se tem uma gama de valores inferior a 20. Caso respeite esta regra considera-se esse atributo qualificado.

3.3.2.3 *Pré-processamento customizado*

Componente de pré-processamento customizado, que é apresentado ao utilizador como opcional. Caso o utilizador queira utilizá-lo possui uma interface gráfica dedicada a recolher os valores que o utilizador pretende usar nas várias estratégias de pré-processamento escolhidas e que são aplicadas. Com a criação desta automatização pretende-se por um lado, facilitar esta etapa retirando a necessidade de fazer todo o trabalho manual de processar os dados, e por outro lado, dar a utilizadores com um conhecimento menos técnico a possibilidade de também efetuar esse processo de uma maneira simples e intuitiva. Contudo, dando a opção do utilizador usar ou não este módulo dá também ainda um maior conforto a esses utilizadores em usar a solução.

Para ter um melhor entendimento geral sobre a estratégia desenvolvida pode observar-se todas as suas etapas na figura 14.

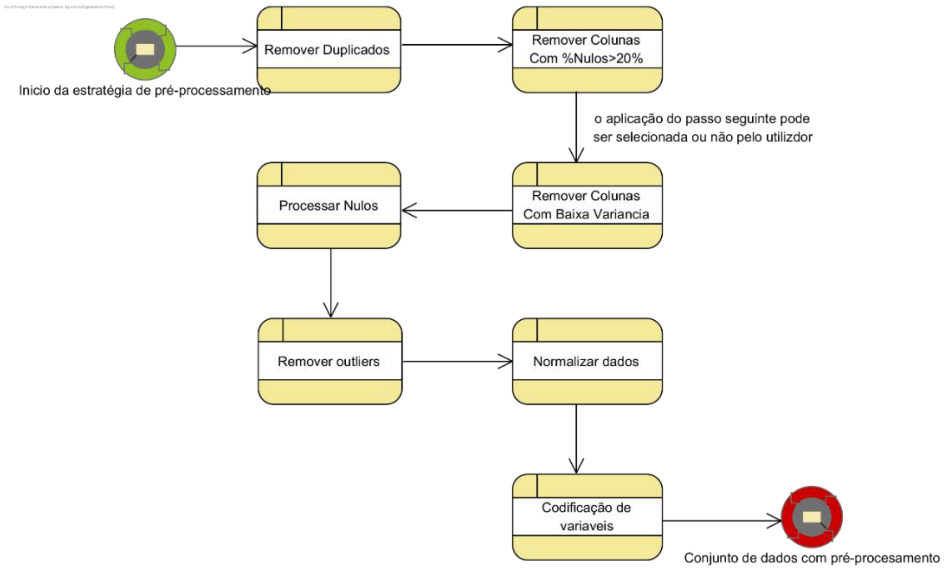

Figura 14 - Arquitetura da pipeline de pré-processamento customizada

Em primeiro lugar decidiu-se retirar os dados duplicados. Em segundo lugar aplica-se um filtro para as colunas que apresentem uma percentagem de valores nulos superior a um determinado valor em percentagem que neste caso foi definido como 20%, ou seja este valor é parametrizável pelo utilizador. O terceiro passo como pode ser observado na figura 10 é opcional, podendo ser ativo ou desativo através da interface gráfica e consiste na remoção de colunas numéricas com baixa variância de valores. O quarto passo consiste em processar os dados nulos, onde é definida uma estratégia para dados numéricos e outra para dados categóricos. Para dados numéricos é possível aplicar mediana ou média para substituir os valores. Para dados categóricos pode usar-se a moda, ou a substituição pela palavra 'Unknown'. O quinto passo consiste em remover os *outlier* e dá a possibilidade de escolher entre três estratégias para isso: *Interquartile Range* (IQR), ou seja, distância interquartil, *Z-score*, ou *Isolation Forest*. O sexto passo consiste na normalização dos variáveis numéricas, onde se dá a possibilidade de aplicar a estratégia de *StandardScaling* ou *MinMaxScaling*. E por último a codificação de varáveis categóricas através das estratégias *OneHotEncoding* ou *LabelEncoding*.

3.3.2.4 Treino do modelo

Componente de automatização principal da aplicação onde a maior parte do processo de ML acontece de forma automática e otimizada usando as capacidades da Framework escolhida como eixo principal desta aplicação. Em primeiro lugar, uma vez que o Auto-sklearn permite fornecer um conjunto de configurações, é dada a possibilidade do utilizador escolher os valores para essas configurações através da interface gráfica. Para além disso caso o utilizador tenha escolhido ou não usar o componente de pré-processamento, uma dessas configurações é alterada para usar ou não o componente interno do Auto-sklearn que lida com as etapas de pré-processamento. Caso o utilizador não queira usar nenhuma das configurações opcionais, pode escolher deixar todas elas em branco, que a aplicação está desenhada para processar esses valores e passá-los ao Auto-sklearn como se não se tivesse passado esse parâmetro, uma vez que todos eles são opcionais, possuindo um valor por defeito.

Depois de serem recolhidas as configurações, o Auto-sklearn vai usar *meta-learning* para tentar comparar os dados fornecidos com dados da internet ou da literatura previamente estudados e usados por quem desenvolveu a Framework, tentando com que caso exista alguma semelhança, procurar se têm alguma lista de técnicas, classificadores , ou conjuntos de configurações para esses mesmos classificadores e técnicas que tenham tido um bom desempenho, para depois serem usados como prioritários no processo iterativo de pesquisa pela melhor pipeline. Depois desta etapa de *meta-learning* concluída, é selecionada aleatoriamente uma estratégia para processamento de *features* e um classificador e com esse conjunto é usado um optimizador de híper-parâmetros Bayesiano para realizar uma otimização na pipeline até ao tempo definido como *treshold*. Esta estratégia é repetida para todos os classificadores até o tempo definido esgotar e dai todas as sub-pipelines construídas são combinadas e é lhes atribuída uma classificação de acordo com a métrica de avaliação definida inicialmente.

Uma característica de manipulação interessante do Auto-sklearn é que permite escolher que componentes se querem utilizar e foi a partir desta característica que através das configurações

fornecidas é possível usar o componente de pré-processamento interno do Auto-sklearn, ou o fornecido pela aplicação, sem conflitos ou sobreposições.

As configurações disponibilizadas ao utilizador consistem:

- time_left_for_this_task – tempo total disponibilizado para a tarefa de treino ser realizada, tem como valor pré-definido e por defeito 3600 segundos, e pode ser definido como não tendo limite;
- per_run_time_limit – tempo máximo que uma iteração dentro da tarefa geral de treino tem para treinar um modelo, por *default*, o valor é 1/10 do indicado na configuração anterior, mas também pode ser definido como não tendo limite;
- initial_configurations_via_metalearning – número inicial de configurações a testar como prioritárias, por defeito é dado o valor de 25 configurações, mas o mesmo processo pode ser desativado através do uso do valor zero para o campo;
- memory_limit – limite de memória em megabytes associado ao algoritmo treinado. caso o modelo passe do limite estabelecido o Auto-sklearn vai falhar a *run* associada e passar para a tentativa seguinte. O limite pode não ser estabelecido associando à variável um valor nulo;
- ensemble_class – define se usa estratégia de ensemble ou não, ou ainda se aplica a estratégia *SingleBest* para obter o modelo final associado à tarefa de treino
- resampling_strategy – define a solução a usar de modo a lidar com o *overfiting*. Como possibilidades tem-se por exemplo, *holdout* ou *cross-validation*;
- metric - caso um valor seja atribuído usa a métrica escolhida para efetuar a avaliação dos modelos de classificação produzidos em cada uma das *runs,* para depois saber quais ou qual usar para formar o modelo final que é produzido pela tarefa. No caso de nenhuma opção ser selecionada o Auto-sklearn tenta de acordo com o tipo de dados presente no conjunto de dados escolher a métrica mais apropriada;
- scoring_functions – lista de métricas de avaliação que vão ser calculadas para cada uma das execuções efetuadas;
- exclude – configuração que permite excluir etapas do processo de treino do Auto-sklearn, como por exemplo, excluir usar o pré-processador interno pois já tinha usado o disponibilizado pelo sistema anteriormente.

Relativamente ao processo de treino de sub-pipelines, ou seja, as várias runs efetuadas, dentro de cada tarefa, para cada umas das partes podem encontrar-se algumas das seguintes configurações:

- Pré-processamento:
    - Codificação One Hot de *features* categóricas;
    - Tratar nulos através de médio, moda ou mediana;
    - Normalização de dados;
    - Balanceamento do conjunto de dados através de *class weights*;
- Processamento de *features*:

    - Decomposição de matriz com PCA, *truncated* SCV, kernel PCA ou ICA;
    - Seleção de features com maior corelação com o atributo objetivo;
    - Seleção baseada em classificação da feature;
    - Clustering de *features*;
    - Expansão polinomial de *features*;
    - Kernel *approximations*;
    - *Feature embeddings*.
- Seleção classificador: Tem acesso a todos os classificadores definidos no sklearn.
- Configuração classificador: Tem também acesso a todas as configurações disponíveis para cada um dos classificadores no sklearn.

Destas etapas, no processo de treino de sub-pipeline, pode ser aleatoriamente selecionado um item das duas últimas seções apresentadas anteriormente. Relativamente ao pré-processamento, podem ser usadas todas as funcionalidades definidas e em relação ao processamento de *features*, pode ser usado uma ou mais algoritmos de features em simultâneo.

Dentro da estratégia de treino são incorporados outros dois componentes como foi explicado na visão geral que são a exploração de dados e o guardar informação do treino que são executados quando o processo de treino do modelo do Auto-sklearn termina e que quando ambos são finalizados dão por terminado o componente de treino do modelo.

3.3.2.5 Informação de treino

Componente incorporado no fim da estratégia de treino projetado para ajudar a informar o utilizador com dados sobre a tarefa de treino. Projetado com recurso a uma interface gráfica simples, fornece ao utilizador um conjunto de informações tanto sobre o modelo como sobre a tarefa executada:

- Visualização da otimização dos híper-parâmetros: Procura quais os híper-parâmetros presentes em todas a configurações de sub-pipelines geradas e através de um *heatmap* mostra a influencia de cada um deles no desempenho do modelo, neste caso através de uma métrica que faz a média dos valores de todas as métricas de avaliação usadas.
- Visualização da importância de *features*: Tem como objetivo mostrar quais a *features* que têm maior impacto na performance do modelo gerado. Esta importância é calculada através de uma técnica chamada Importância por Permutação, que mede quanto o desempenho do modelo piora quando os valores de uma *feature* são trocados aleatoriamente. Para além de serem apresentadas por ordem de importância, no fim são apresentadas as *features* que possuem mais do que 10% de percentagem de importância.
- Visualização do relatório de classificação (*classification report*): Fornece uma visão detalhada de algumas das métricas de desempenho do modelo final gerado, entre elas, Precisão, Sensibilidade, F1-Score e Suporte. Importante, pois, dá ao utilizador uma visão geral da avaliação sobre as capacidades do mesmo

- Visualização da matriz de confusão: Tabela que descreve o desempenho do modelo comparando as previsões efetuadas pelo mesmo com os valores reais, valores de teste. Permite identificar mais facilmente quais os tipos de erro que o modelo está a cometer.
- Visualização dos modelos ordenados pela avaliação média: Permite observar ordenadamente pela média das métricas de avaliação os modelos gerados pelas sub-pipelines.
- Visualização da tabela de classificação dos modelos de sub-pipelines treinados: Mostra uma tabela com informação detalhada sobre o modelo, como por exemplo métricas de desempenho e tempo de treino no modelo
- Visualização de modelos e pesos associados: Mostra uma lista de todos os modelos que foram utilizados para o ensemble final e a importância relativa de cada um desses modelos para o modelo final gerado.
- Visualização das estatísticas de treino: Mostra um resumo das estatísticas da tarefa executada, incluídas informações sobre o nome do conjunto de dados usado, a métrica usada para avaliação e a melhor nota de validação obtida, para além disso diz quantas sub-pipelines correram com sucesso e falharam devido ao tempo estabelecido como máximo para uma *run*.
- Visualização do desempenho ao longo tempo: Gera um gráfico que mostra como aumenta a performance dos modelos gerados com o passar do tempo da tarefa, o que permite por exemplo identificar se um determinado tempo pode ser adequado ou não para um treino, ao observar se a performance vai oscilando, ou vai convergindo para um valor ótimo à medida que o tempo passa.

3.3.2.6 Guardar informação do treino

Componente criado para guardar e registar o modelo e algumas informações sobre o mesmo no MLFlow, que consiste numa plataforma de gestão e monitorização de experiências de ML, que são basicamente treinos de vários modelos, para procurar o melhor modelo para usar numa determinada solução. Com isto o utilizador vai poder aceder às estatísticas dos vários modelos guardados, comparando-os para ajudar a escolher qual o mais apropriado para ser usado como solução final para produção, ou seja, ser *deploy*ed. Para além das estatísticas ajuda também a dar conhecimento ao utilizador qual a estrutura de dados que o modelo vai necessitar receber para realizar previsões quando já estiver pronto e qual o output que vai gerar através do registo da assinatura do modelo. Uma vez registado o modelo fica guardado e pode ser recuperado e usado em qualquer altura. Quando este registo é efetuado é criada uma *tag* personalizada que identifica o modelo pelo nome para que seja mais fácil de distinguir e identificar depois.

3.3.2.7 Realizar deployment

Componente criado para realizar o *deployment* do modelo escolhido. Uma vez que o utilizador esteja satisfeito com a qualidade do modelo gerado, só necessita de procurar esse modelo pelo nome através duma interface gráfica que mostra todos os modelos gerados, ou seja, o resultado da tarefa de treino para este problema. Depois de escolher o modelo, é feita uma busca no MLFLow pelo modelo sendo este carregado, ou seja, colocado dentro de um servidor web criado a partir da tecnologia Flask, que vai permitir facilmente ao modelo receber pedidos para

realizar previsões como se se tratasse de uma mera web API. Esta web API com recurso à estratégia de definir uma rota para a função que contem o modelo, fica à escuta por pedidos podendo ser facilmente acedida por qualquer aplicação ou serviço que tenha ligação à internet, através de um pedido http.

3.3.3 Fluxo de Dados

Com análise individual de cada um dos componentes foi possível perceber qual o principal objetivo e função de cada um deles. Nesta seção vai ser feita uma análise do fluxo de dados para perceber como a informação chega ao sistema e o produto final retornado por ele. Para isto foi elaborada a figura 15 que tem como função ajudar a perceber os fluxos da mesma.

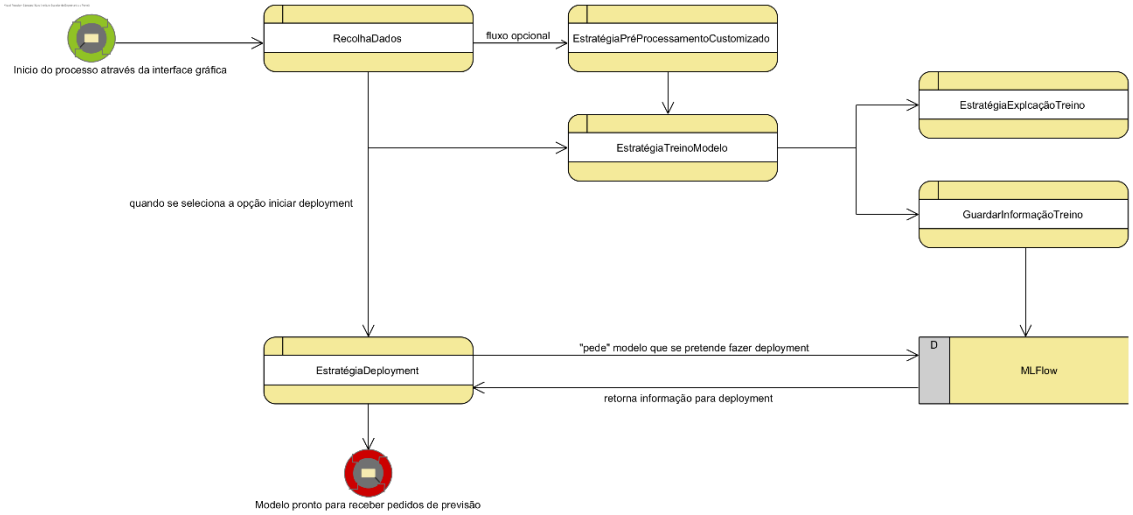

Figura 15 - Fluxo de dados do sistema

Como se pode observar na figura 15 através da interface gráfica o processo de recolha de dados acontece, onde o utilizador escolhe o ficheiro que pretende usar como *dataset* a explorar e depois disso escolhe qual o atributo objetivo que pretende. Depois deste passo o sistema através da interface gráfica permite ao utilizador realizar a estratégia de pré-processamento customizada ou passar diretamente para a estratégia de treino do modelo. Caso escolha realizar pré-processamento, através da interface gráfica o utilizador é apoiado a realizar escolhas para os parâmetros que pretende aplicar para pré-processamento. Uma vez terminada esta ação opcional é encaminhada para a estratégia de treino. Nesta componente o utilizador vai definir quais os parâmetros de configurações adicionais que quer utilizar, ou se não quer aplicar quaisquer configurações auxiliado pela interface gráfica disponível e seleciona que pretende começar o treino do modelo. Quando o processo de treino é executado uma vez que esteja terminado vai invocar o componente de exploração de dados, onde vai ser apresentado através de uma interface algumas informações da tarefa que terminou. Uma vez que o utilizador tenha terminado de observar essas informações fecha a janela da interface dessa seção. Em seguida é invocado o componente que guarda informação de treino, que vai interagir com uma instância do MLFlow, de forma a guardar lá todas as informações previamente indicadas na parte da

análise individual deste modelo. Este processo de treino pode ser repetido quantas vezes o utilizador quiser uma vez que quando o componente de treino acaba a sua execução, através da interface gráfica, volta automaticamente ao estado em que está pronto para recolher as configurações para executar uma nova tarefa. Quando o utilizador obtém um modelo que responda às suas necessidades, pode facilmente fazer o seu *deploy*. Para isto só tem de ativar a opção realizar *deployment* através da interface gráfica. Quando invocado o componente da estratégia de *deployment*, ele vai permitir ao utilizador escolher qual o modelo que pretende colocar em produção dentro de uma lista de modelos treinados para o problema que está a resolver, ou seja, modelos que vêm das mesmas seleções efetuadas no componente da recolha de dados. Depois de escolher o modelo, o sistema vai buscá-lo à mesma instância de MLFlow ativa onde foi registado e daí segue o processo explicado na parte individual deste componente, ficando o modelo pronto a receber *requests* com as informações adequadas para efetuar previsões.

3.3.4 Requisitos não funcionais na arquitetura da solução

A arquitetura da aplicação foi desenhada para atender aos diversos requisitos não funcionais previamente definidos, garantindo que o sistema seja robusto e flexível o suficiente para evoluir conforme as necessidades.

A escalabilidade é um aspeto central, permitindo que a aplicação lide com grandes volumes de dados e aumente a sua capacidade de processamento conforme necessário, devido às capacidades nativas do Auto-sklearn e das configurações que permite usar, assegurando que o sistema possa responder bem ao aumento do volume de dados a processar. O desempenho, através das mesmas capacidades é otimizado para que o processo de treino seja o mais rápido e eficiente, minimizando o tempo necessário para gerar resultados bons, mesmo a partir de conjuntos de dados complexos ou volumosos. Além disso, a usabilidade foi priorizada, aplicando um design intuitivo através da interface gráfica que facilita o uso da aplicação, especialmente para utilizadores com conhecimento técnico mais limitado, permitindo que todos possam tirar proveito das funcionalidades oferecidas. A modularidade da aplicação, aliada à documentação, assegura sua manutenibilidade, facilitando futuras atualizações e manutenções à medida que novas necessidades ou tecnologias surgem. A compatibilidade com diferentes sistemas operacionais e a capacidade de integração com várias bibliotecas de ML e frameworks de AutoML foram também consideradas, maximizando a flexibilidade e a interoperabilidade do sistema. Por fim, a suportabilidade foi cuidadosamente trabalhada, garantindo que a aplicação ofereça um alto grau de configurabilidade sem sacrificar a simplicidade de uso e a automação, permitindo que tanto usuários técnicos quanto menos técnicos possam explorar plenamente as funcionalidades proporcionadas pelo sistema.

4 Implementação da solução

Nesta seção vão ser abordados os detalhes sobre a implementação da solução desenhada no capítulo anterior. Vai ser possível perceber quais as tecnologias, linguagens e frameworks que foram utilizados e como estão interligados para permitir obter o bom funcionamento do sistema. Para além disso vai poder-se observar a estrutura do código, onde se vai conseguir ter uma melhor ideia da organização dos módulos e classes que foram utilizadas. No subcapítulo seguinte vai ser possível perceber como foram implementadas algumas das partes mais importantes dos componentes principais da aplicação. Por último, vai ser apresentado através de um exemplo prático a aplicação em funcionamento.

4.1 Ambiente de Desenvolvimento

Como núcleo desta aplicação foi escolhido o Auto-sklearn que é uma Framework de AutoML, desenvolvida em Python que está preparada para correr em ambientes Linux, tendo sido testada a compatibilidade direta em ambiente Windows e não se tendo tido sucesso nas tentativas. Uma vez que se está a desenvolver a aplicação num computador que usa Windows e para também não ser um fator limitante para os utilizadores foi testada uma maneira para conseguir usar a mesma. A solução encontrada consistiu em instalar e ativar uma máquina virtual que usasse Linux através das funcionalidades nativas oferecidas pelo Windows Hyper-V. Com isto instalou-se uma máquina com Ubuntu e trabalhou-se com o projeto hospedado e configurado nesse ambiente. Para tratar das configurações do ambiente, uma vez que se estava a usar Python, decidiu-se usar o Anaconda que tem como objetivo simplificar o processo de gestão de módulos. O processo utilizado consistiu em criar um ambiente virtual conda específico para o projeto que tivesse todos os módulos instalados e nas versões certas para que todas as ferramentas funcionassem sem conflitos. Cada vez que se quisesse ativar esse ambiente virtual só era necessário usar a função “conda activate exemplo” onde exemplo é o nome do ambiente que se deseja ativar e têm-se acesso a todos os módulos instalados no mesmo. Para instalar módulos a ferramenta mais usada foi o pip que é um instalador de pacotes ou módulos para Python. Uma vez que todas as configurações foram tratadas criou-se o projeto usando um ambiente de desenvolvimento integrado para programar em Python que se chama PyCharm, desenvolvido pela empresa JetBrains. Com ele foi possível criar o projeto usando o ambiente virtual previamente criado e podendo assim começar o processo de desenvolvimento do sistema. Para o controlo de versões foi usado o Git.

4.2 Estrutura do código

Uma vez com o ambiente todo configurado o desenvolvimento da aplicação foi feito de maneira iterativa e modular permitindo ir facilmente identificando os erros que iam decorrendo ao longo do percurso de implementação, tendo sido numa primeira parte desenvolvido todo o core da aplicação relativo aos componentes mais importantes associados ao domínio do ML e só depois desenvolvida a interface gráfica que une todos os processos e facilita a interação do utilizador com a mesma. Em seguida vai analisar-se o aspeto geral da estrutura da pasta onde aplicação está guardada.

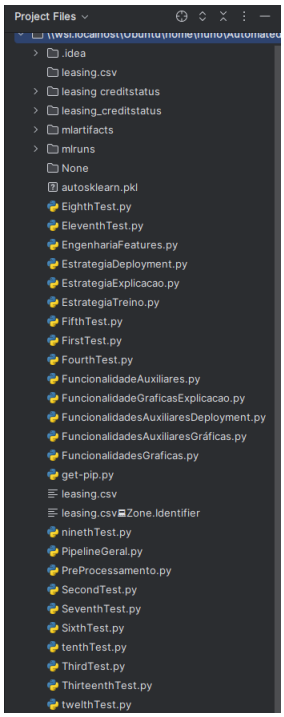

Figura 16 - Estrutura da pasta da aplicação

Como se pode observar à medida que o desenvolvimento foi efetuado foram sendo acrescentados passos de complexidade e dai surgiram alguns ficheiros que têm o nome de teste e se podem observar na figura 16. Esses ficheiros foram uteis para testar os processos que estavam a ser desenvolvidos para garantir que tudo corria como era esperado. A partir do momento em que se implementou a interface gráfica, todos os componentes foram convertidos em ficheiros individuais cada um com a sua tarefa como lhe é designado pelo nome do mesmo. O ficheiro que trabalha como o *init* da aplicação é o ficheiro que se chama PipelineGeral que o que faz é simplesmente invocar ficheiro FuncionalidadesGráficas. Depois esse ficheiro tem a capacidade de saber quando tem de invocar cada um dos outros componentes, que como se pode observar, cada um deles tem o seu próprio ficheiro, o que facilita tanto a manutenção como adição de novos componentes sem interferir com os já existentes. Para além disso pode observar-se que foram criadas para alguns ficheiros mais extensos classes com métodos auxiliares para as tarefas que têm de cumprir, de modo a manter a organização da aplicação. Para além dos ficheiros, podem observar-se algumas pastas que

foram criadas pela aplicação de modo a guardar os modelos treinados. Como se pode observar por exemplo na pasta leasing_creditstatus que guarda todos os modelos treinados a partir do ficheiro csv leasing e onde o atributo objetivo é a variável creditstatus. As pastas mlartifacts e mlruns são pastas geradas com as informações dos modelos quando são guardados os modelos no MLFlow, que usa essas pastas para fazer a persistência dos modelos e as suas informações.

4.3 Overview da implementação dos componentes principais

Nesta seção vai poder observar-se detalhes sobre a implementação de alguns dos métodos que são considerados mais importantes em cada um dos componentes desenvolvidos.

4.3.1 Interface Gráfica

Para se ter uma melhor ideia da implementação efetuada nesta seção vai-se fazer a análise do método que lida com a Recolha de Dados.

```
2 usages
def criarInterfaceRecolhaDados():
    global root, label_arquivo, label_campo_alvo, coluna_alvo_var, btn_confirmarColuna
    global frame_campos, frame_preprocessamento, frame_treino

    root = tk.Tk()
    root.title("Seleção do Ficheiro CSV e coluna alvo")

    frame_selecao = tk.Frame(root)
    frame_selecao.pack(pady=20)

    btn_selecionar = tk.Button(frame_selecao, text="Selecionar arquivo CSV", command=selecionar_arquivo)
    btn_selecionar.pack(side='left', padx=10)

    label_arquivo = tk.Label(frame_selecao, text="Nenhum arquivo selecionado")
    label_arquivo.pack(side='left', padx=10)

    frame_campos = tk.Frame(root)
    frame_campos.pack(pady=20)

    label_campo_alvo = tk.Label(frame_campos, text="")
    label_campo_alvo.grid(row=0, column=0, padx=5, pady=5)
    coluna_alvo_var = tk.StringVar(root)

    frame_preprocessamento = tk.Frame(root)
    frame_preprocessamento.pack(pady=20)

    btn_confirmarColuna = tk.Button(root, text="Prosseguir", command=perguntarPreprocessamento)
    btn_confirmarColuna.pack(pady=10)

    root.mainloop()
```

Figura 17 - Exemplo de implementação de interface com recolha de dados

Como se pode observar a interface foi implementada com recurso ao módulo de Python tkinter como idealizado previamente no design. A estratégia que foi aplicada e se pode observar neste excerto reflete também a estratégia que foi aplicada para todas as vistas que a aplicação possui.

Na figura 17 observa-se que as ações são tomadas a partir da opção “command”, que define a ação que acontece quando se clica no botão onde é definida. Pode observar-se por exemplo na linha 29 onde é invocada a função selecionar_arquivo, que define a lógica para o utilizador escolher o arquivo que pretende selecionar como conjunto de dados para o treino. Entre a linha 38 e 40 é estruturado o aspeto gráfico que suporta a exibição dos campos do ficheiro escolhido que são filtrados como qualificados para serem variáveis objetivo e permitir selecionar um deles. A linha 45 é um botão que ajuda a confirmar as escolhas efetuadas e que invoca através da opção “command” uma função que permite ao utilizador efetuar o pré-processamento customizado ou passa diretamente para o painel do treino do modelo.

4.3.2 Recolha de Dados

Um pormenor importante nesta parte é a lógica explicada anteriormente que permite identificar se uma coluna é ou não qualificada para ser atributo alvo.

```
2 usages
def coluna_disponivel_classificacao(coluna):
    if coluna.dtype == 'object' or coluna.dtype.name == 'category':
        return True
    if coluna.dtype in ['int64', 'float64'] and coluna.nunique() < 20:  # Se tiver menos de 20 classes únicas
        return True
    return False
```

Figura 18 - Implementação da regra definida para qualificar colunas para atributo objetivo

Como se pode observar na figura 18, o método classifica como verdadeira ou falsa a coluna que recebe como parâmetro pois é invocado para todas as colunas do *dataset* escolhido, fazendo assim a filtragem segundo a regra explicada no capítulo do design.

4.3.3 Estratégia de Pré-Processamento Customizada

Na figura 19 vai observar-se toda a lógica da pipeline de pré-processamento implementada e usada pela aplicação caso o utilizador queira usar a mesma.

```
9   def preprocess_data(df,estrategia_numerica, estrategia_categorica, estrategia_outliers,estrategia_codificacao,estrategia_normalizacao,escolha_remocao_variancia,treshold_variancia):
10      if df is None:
11          return None
12
13      #Remover duplicados
14      dados_processados = remover_duplicados(df)
15
16      # Remover colunas com muitos dados nulos (mais de 20%)
17      dados_processados = remover_colunas_nulos(dados_processados)
18
19      #Remover colunas com baixa variancia
20      if escolha_remocao_variancia:
21          dados_processados = remover_colunas_baixa_variancia(dados_processados, treshold_variancia)
22
23      # Tratamento de dados nulos
24      dados_processados = processar_dados_nulos(dados_processados,estrategia_numerica,estrategia_categorica)
25
26      # Deteção e eliminação de outliers
27      dados_processados = remover_outliers(dados_processados, estrategia_outliers)
28
29      # Normalização de dados numéricos
30      dados_processados = normalizar_dados(dados_processados,estrategia_normalizacao)
31
32      # Codificação de variaveis categóricas
33      dados_processados = codificar_variaveis_categoricas(dados_processados,estrategia_codificacao)
34
35      return dados_processados
```

Figura 19 - Implementação do método que estrutura a pipeline de pré-processamento

Como se pode observar a função recebe as escolhas do utilizador para cada um dos passos que implementa e vai para cada um deles, passando o modelo processado pelo passo anterior e a estratégia a aplicar nesse passo. Por exemplo, linha 24, tratamento de dados nulos, quando a função processar_dados_nulos é invocada é lhe passado o último conjunto de dados, e as estratégias escolhidas, para este caso, e retorna o conjunto de dados com as estratégias já aplicadas. Na linha 20 pode observar-se um operador condicional uma vez que é dada através da interface gráfica a possibilidade de o utilizador aplicar esse campo ou não no pré-processamento que está a efetuar.

4.3.4 Estratégia de Treino do Modelo

Esta é a seção que implementa diretamente a Framework *core* de AutoML, o Auto-sklearn, daqui é importante destacar algumas partes. Em primeiro lugar como se observa na figura 20, a divisão em conjunto de teste e conjunto de treino:

```
32      # Dividir dados em conjunto de teste e conjunto de treino
33      X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=proporcao_teste, random_state=42)
```

Figura 20 - Implementação da estratégia de divisão de dados para treino e teste

Como se pode observar consiste na invocação da função train_test_spit do módulo sklearn.model_selection, que recebe como argumento a proporção que o utilizador quis dar à variável test_size através da interface gráfica.

Outro pormenor importante é a criação do conjunto de métricas adicionais que se informa o Auto-sklearn que são importantes para serem guardadas junto com os resultados do modelo, caso essa atribuição não seja feita, essas métricas não são calculadas. Pode observar-se essa criação na figura 21 e depois pode ver-se a atribuição na figura 23.

```
# Metricas a guardar no cv_results
scoring_functions = [autosklearn.metrics.accuracy, autosklearn.metrics.f1, autosklearn.metrics.roc_auc,
                     autosklearn.metrics.precision, autosklearn.metrics.recall, autosklearn.metrics.log_loss]
```

Figura 21 - Implementação das métricas adicionais a guardar nos resultados

Um passo importante é o representado na figura 22 que é o que permite ter a opção desativar o módulo interno do Auto-sklearn através do valor dentro da variável “exclude” que depois é passada com atributo na figura 19.

```
# Adicionar componente NoPreprocessing ao autosklearn
autosklearn.pipeline.components.data_preprocessing.add_preprocessor(NoPreprocessing)
```

Figura 22 - Implementação da adição de um componente para uma seção específica do Auto-sklearn

Na figura 23 é feita a criação da instância do AutoML e o seu fit, como se pode observar.

```
# Criar e fit de um classificador autosklearn
automl = autosklearn.classification.AutoSklearnClassifier(time_left_for_this_task=tempo_tarefa,
                                                          per_run_time_limit=tempo_corrida,
                                                          initial_configurations_via_metalearning=num_config_meta,
                                                          memory_limit=memoria_limite,
                                                          ensemble_class=ensemble_class,
                                                          resampling_strategy=resampling_strategy,
                                                          metric=metrica_avaliacao,
                                                          scoring_functions=scoring_functions,
                                                          exclude=exclude)

automl.fit(X_train, y_train)
```

Figura 23 - Implementação instância automl com configurações e respetivo fit

Quando se cria a instancia automl através da função do autoskelarn.classification.AutoSklearnClassifier são passados por parâmetro todas as configurações escolhidas pelo utilizador e processadas pelo sistema que permitem ter toda a configurabilidade do processo de treino que é executado quando se faz o automl.fit com os conjuntos de teste.

Dois pormenores importantes que também são implementados dentro deste componente são a invocação da estratégia de explicação através do método presente na figura 24 e também a invocação do método que guarda o modelo no MLFlow na figura 25.

```
visualizarExplicacao(automl, X_train, y_train, X_test, y_test, y_pred,df_ajust,X)
```

Figura 24 - Implementação da invocação do componente com a estratégia de explicação

```
logModelMlFlow(automl,X_train,nome_arquivo_string,nome_modelo_guardado)
```

Figura 25 - Implementação da invocação do componente com a estratégia de guardar informação sobre o modelo

4.3.5 Estrategia Deployment

Para as etapas da estratégia de *deployment* implementada tem-se as etapas identificadas na função que está na figura 26.

```
76  def deploymentStrategy(model_name,nome_arquivo):
77
78      run_id = getRunID(model_name)
79
80      # os.environ["MLFLOW_ARTIFACT_DOWNLOAD_TIMEOUT"] = "300"
81      serve_model_locally(run_id, model_name,nome_arquivo)
```

Figura 26 - Implementação da estratégia de *deployment*

Como se constata o primeiro passo consiste em ir buscar o identificador da *run* associado ao modelo que se pretende fazer *deployment* através da função que é invocada na linha 78. Na linha 83 invoca-se a função que trata dos restantes processos, que são carregar o modelo diretamente do MLFLow e depois de ter esse modelo carregado criar a aplicação Flask e a rota que permite fazer as previsões usando o modelo carregado.

4.4 Exemplo de uso da aplicação

Nesta seção vai ser mostrado através de um exemplo a aplicação em funcionamento. Em primeiro lugar é necessário ativar a instância do mlflow, e neste caso optou-se por fazer isso dentro do ambiente virtual Linux onde a aplicação está guardada. Para além disso também se vai ativar o ambiente conda criado para o desenvolvimento da aplicação.

```
(base) nuno@DESKTOP-K8RG1A7:~$ conda activate autosklearn
(autosklearn) nuno@DESKTOP-K8RG1A7:~$ mlflow server --host 127.0.0.1 --port 8082
[2024-09-03 22:06:35 +0100] [789] [INFO] Starting gunicorn 21.2.0
[2024-09-03 22:06:35 +0100] [789] [INFO] Listening at: http://127.0.0.1:8082 (789)
[2024-09-03 22:06:35 +0100] [789] [INFO] Using worker: sync
[2024-09-03 22:06:35 +0100] [790] [INFO] Booting worker with pid: 790
[2024-09-03 22:06:36 +0100] [791] [INFO] Booting worker with pid: 791
[2024-09-03 22:06:36 +0100] [792] [INFO] Booting worker with pid: 792
[2024-09-03 22:06:36 +0100] [804] [INFO] Booting worker with pid: 804
```

Figura 27 - Exemplo de ativação de ambiente virtual e início de uma instância do MLFow

Como se pode observar na figura 27, o ambiente virtual é ativo através do primeiro comando executado e com o segundo comando pretende-se iniciar a instância do MLFlow definindo o seu endereço e a porta para ficar à escuta.

Uma vez definido o ambiente inicia-se a aplicação sendo a vista da figura 28 a primeira janela apresentada ao utilizador.

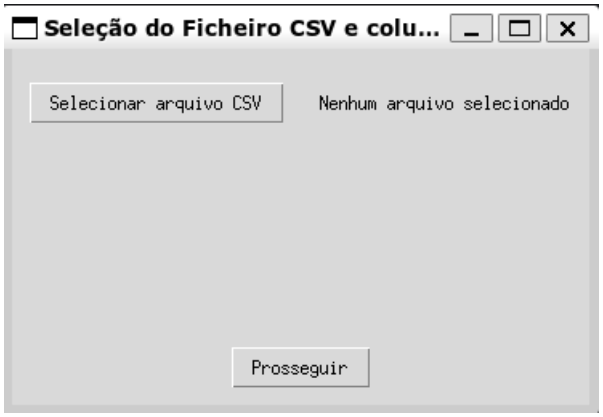

Figura 28 - Visão inicial da aplicação

Como se pode reparar tem um botão que permite escolher um arquivo csv e um botão que serve para prosseguir para a próxima seção, como o primeiro objetivo é selecionar um arquivo vai clicar-se nesse botão e vai surgir a seguinte vista:

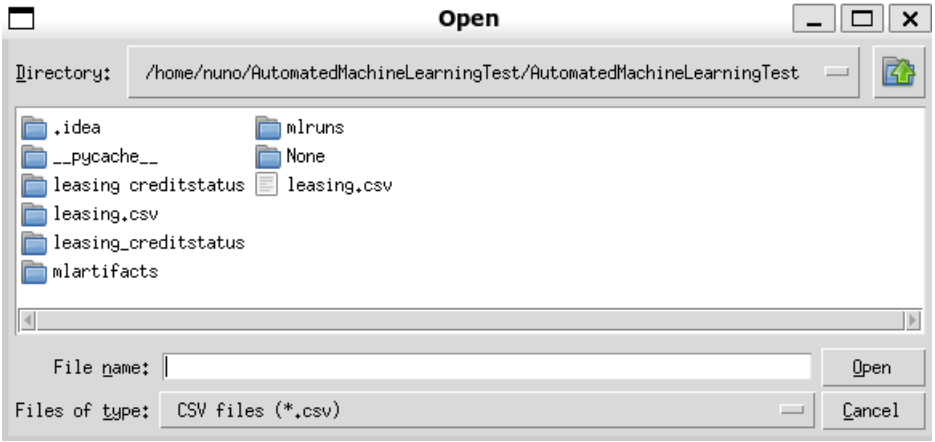

Figura 29 - Vista auxiliar para escolher ficheiro csv

Depois de selecionado a vista inicial vai automaticamente alterar-se para a seguinte:

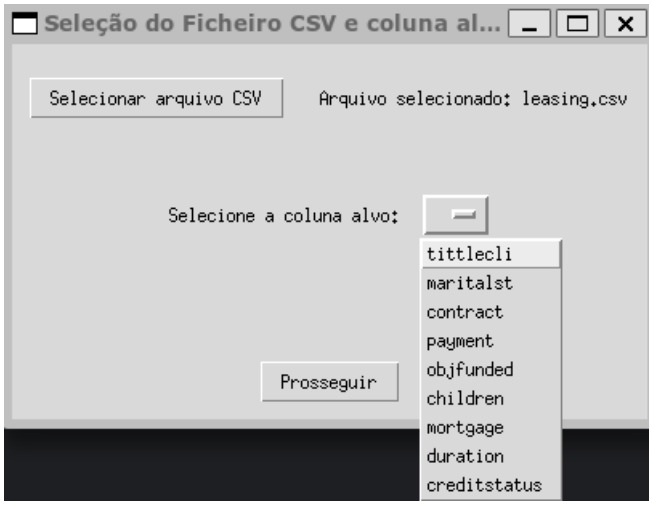

Figura 30 - Vista atualizada depois da seleção do ficheiro de dados

Como se pode observar na imagem o sistema processou o conjunto de dados escolhido e das 13 colunas que o ficheiro possuía filtrou através da regra explicada anteriormente quais dessas colunas estavam aptas para ser aplicada um algoritmo de classificação e essa lista pode observar-se no campo que permite a escolha da coluna alvo. Neste caso vai ser escolhida a variável creditstatus e vai ser selecionado o botão Prosseguir. Quando se efetua esta decisão o sistema vai verificar se existe uma pasta para modelos para esse problema específico, ou seja, para o mesmo conjunto de dados, ou pelo menos, para conjunto de dados com o mesmo nome e que depois seja para a mesma coluna alvo. Caso essa pasta exista não faz nada, no caso de a pasta não existir cria a pasta onde o modelo vai ser guardado localmente. Este passo é importante pois ajuda a organizar os modelos treinados por problema caso queiram ser reutilizados.

A segunda vista que aparece é a que está na figura 31.

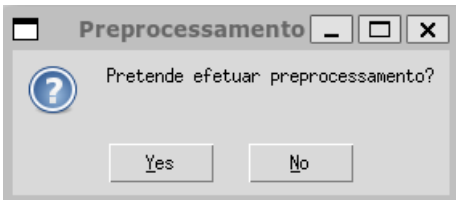

Figura 31 - Vista para escolha entre realizar pré-processamento customizado ou não

Nesta etapa o utilizador pode escolher entre aplicar o pré-processamento customizado disponibilizado pela aplicação, ou usar o modulo interno no Auto-sklearn. Neste caso vai usar-se o módulo de pré-processamento customizado desenvolvido. Caso o utilizador selecione “No”, vai ser automaticamente usado o pré-processador interno do Auto-sklearn e vai prosseguir para a vista do treino do modelo.

Quando se seleciona o “Yes”, é apresentada a informação que está presente na figura 32.

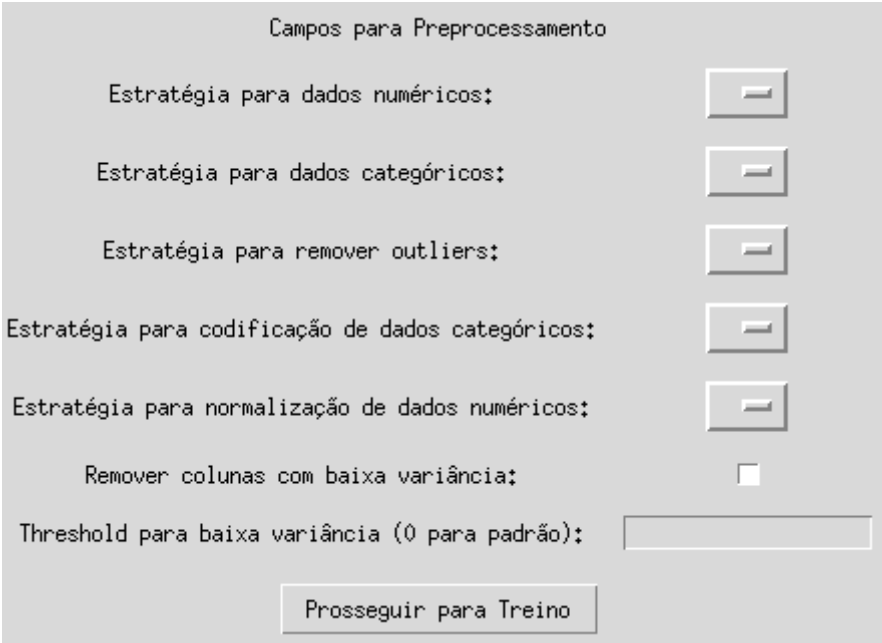

Figura 32 - Interface pré-processamento customizado

Como se pode observar, através desta interface é possível selecionar todas as configurações necessárias para realizar o pré-processamento customizado, desenvolvido para esta aplicação. Para o primeiro campo são aceites as opções que estão na figura 33.

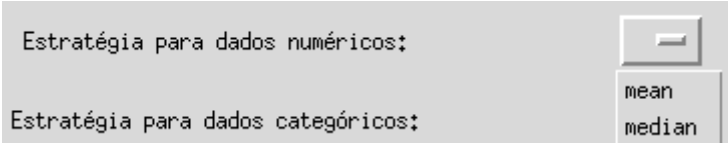

Figura 33 - Opções para primeiro campo

Relativamente à segunda opção, as opções também se podem observar na figura 34.

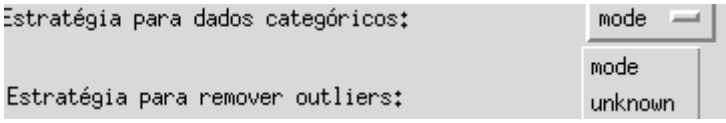

Figura 34 - Opções para segundo campo

Para o terceiro parâmetro as opções são as indicadas na figura 35.

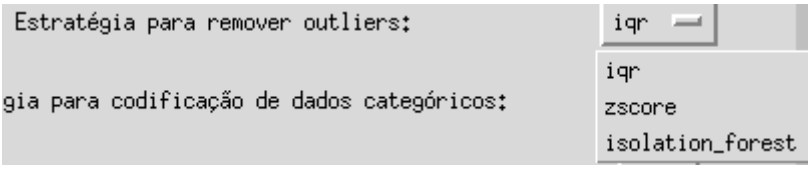

Figura 35 - Opções para terceiro campo

Para o quinto e sexto parâmetro as opções são encontradas respetivamente na figura 36 e 37.

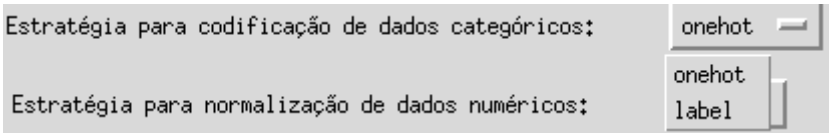

Figura 36 - Opções para quarto campo

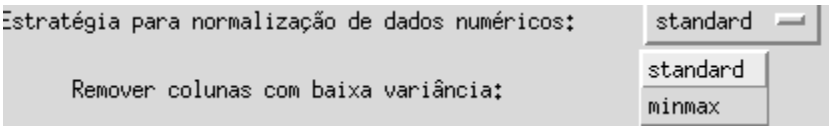

Figura 37 - Opções para quinto campo

Quanto à última opção de pré-processamento, ela pode ser escolhida de ser aplicada ou não pelo utilizador através da *checkbox* que aparece à frente da sua descrição como se observa na figura X. Uma vez selecionado esse campo, desbloqueia o campo de baixo para recolher qual o valor pretendido para o utilizador para o campo desbloqueado.

Uma vez que o utilizador esteja satisfeito com as configurações desejadas para o pré-processamento, clica no botão "Prosseguir para treino" onde é encaminhado para a interface de treino do modelo e o seu *deployment* que se pode observar na figura 38.

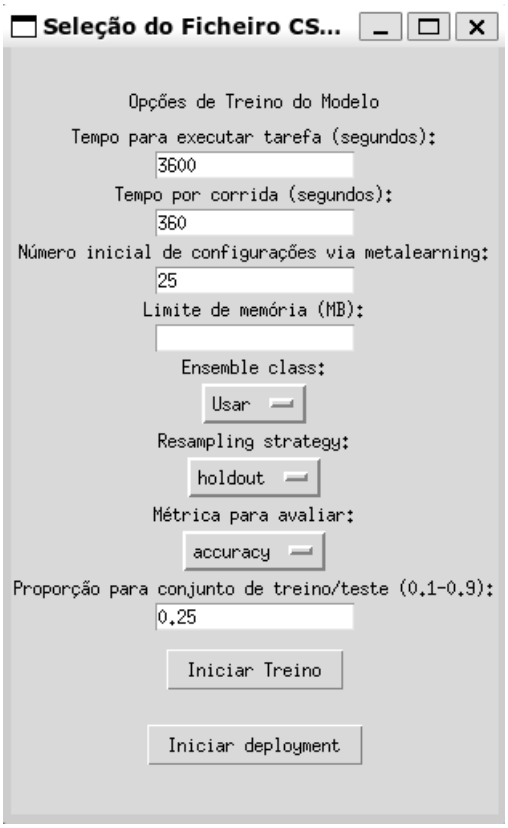

Figura 38 - Vista do painel de treino de modelo

Como se pode observar nesta interface é fornecido ao utilizador uma maneira simples de editar as configurações que pretende usar para treinar o modelo. Tendo campos para a leitura de valores numéricos como é o exemplo do tempo para executar tarefa, e também lista com as opções que outras configurações suportam como é o caso do campo métrica para avaliar. Com esta maneira de apresentação e seleção das configurações qualquer utilizador mesmo que tenha grande conhecimento técnico é capaz de usar sem qualquer dificuldade as várias opções oferecidas pela Framework. Para este caso as configurações fornecidas vão ser, para o campo tempo de execução da tarefa, que é o tempo total que tarefa pode demorar, em segundos, vai ser 600 seg.. O tempo por corrida, que é o tempo máximo que uma *run* pode demorar dentro da tarefa vai ser 45 seg.. O número inicial de configurações vai ser 25. O limite de memória não vai se aplicado, logo vai sem valor. Na variável "ensemble class" vai ser escolhida a opção "Usar", sendo que as opções disponíveis podem ser observadas na lista na figura 29.

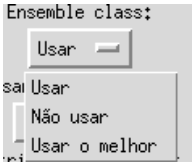

Figura 39 - Opções para variável "*Ensemble class*"

Onde "Usar" significa que no fim de correr as várias runs dentro da tarefa se vai aplicar uma estratégia de ensemble para formar o modelo final para a tarefa. A opção não usar não aplica qualquer forma de ensamble e a opção Usar o Melhor usa o modelo que tenha maior classificação de avaliação.

Para a configuração "resamplig strategy" estão disponíveis as opções representadas na figura 40.

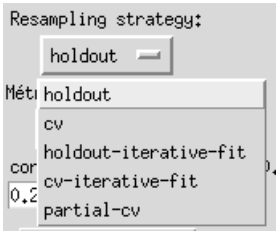

Figura 40 - Opções para variável "Resampling strategy"

Neste caso vai ser selecionada a estratégia "cv". Relativamente à configuração da métrica para avaliar, estão disponíveis as opções representadas na figura 41.

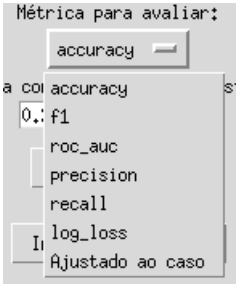

Figura 41 - Opções para variável “Métrica para avaliar”

A escolha da métrica impacta diretamente nos resultados uma vez que o Auto-sklearn vai ordenar as soluções pela avaliação da respetiva métrica para a tabela de liderança dos modelos, o que tem implicações depois nos modelos escolhidos para construir o ensamble. Nota-se que a opção “Ajustado ao caso” é a confirmação por defeito da Framework que tenha de encontrar a métrica de avaliação mais apropriada para a solução através da análise do conjunto de dados usados. Neste caso vai ser usado *accuracy*.

Por último a configuração serve para o utilizador definir a proporção da divisão de dados para treino e teste usado na função train_test_split previamente apresentada, onde neste caso foi escolhido usar o valor 0.25.

Em qualquer um dos campos que suporta valores em branco, essa é a estratégia a utilizar caso utilizador não queria usar um determinado campo. Caso queira usar os valores por defeito o utilizador não precisa de definir qualquer uma das configurações que lhe são mostradas.

Quando o utilizador tiver todas as configurações escolhidas pode efetuar o treino do modelo com essas configurações simplesmente clicando no botão Iniciar Treino.

Caso o utilizador já esteja na etapa em que quer fazer o *deploy* de um dos modelos que treinou para o problema, só necessita de clicar no botão Iniciar deployment.

Para já, neste caso vai ser treinado um modelo por isso vai ser selecionado o botão IniciarTreino e vai se apresentada a janela que se pode observar na figura 42.

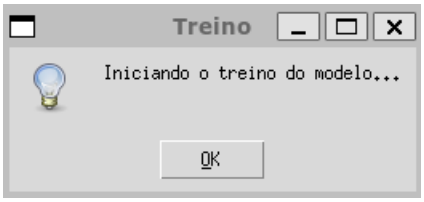

Figura 42 - Confirmação do início do processo de treino

O treino do modelo vai começar quando o botão Ok for pressionado.

Passado o tempo definido no primeiro parâmetro configurável usado na visão de treino de modelo a janela anterior desaparece e aparece a janela relativa à estratégia de explicação que se pode observar na figura 43.

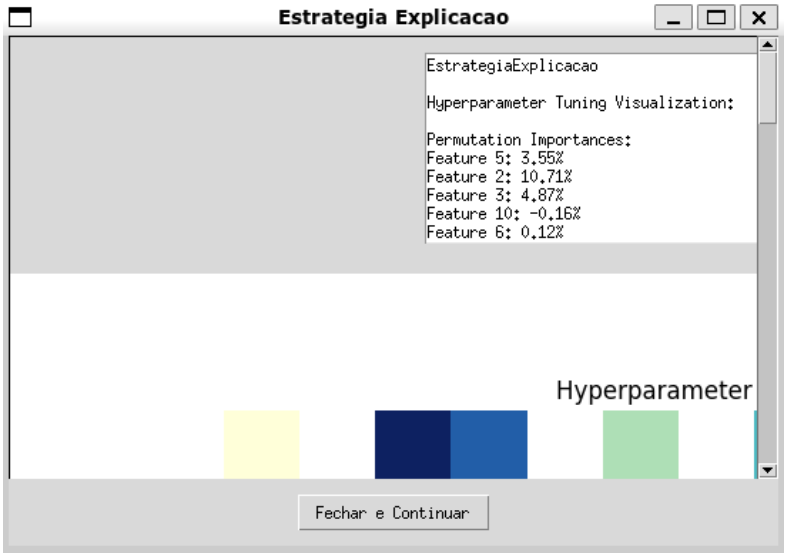

Figura 43 - Visão geral da estratégia de explicação

Nesta janela podem ser encontradas todas as informações detalhadas na parte do design da explicação, mas como a janela é grande vai ser mostrado individualmente cada passo. Repara-se que na parte inferior da janela está o botão que permite fechar a janela e que vai permitir a continuação da execução do código, neste caso, vai permitir passar para o componente que guarda o modelo no MLFlow.

Em primeiro lugar nesta janela de explicação é mostrado um *heatmap* relativo aos híper-parâmetros como se pode observar na figura 44.

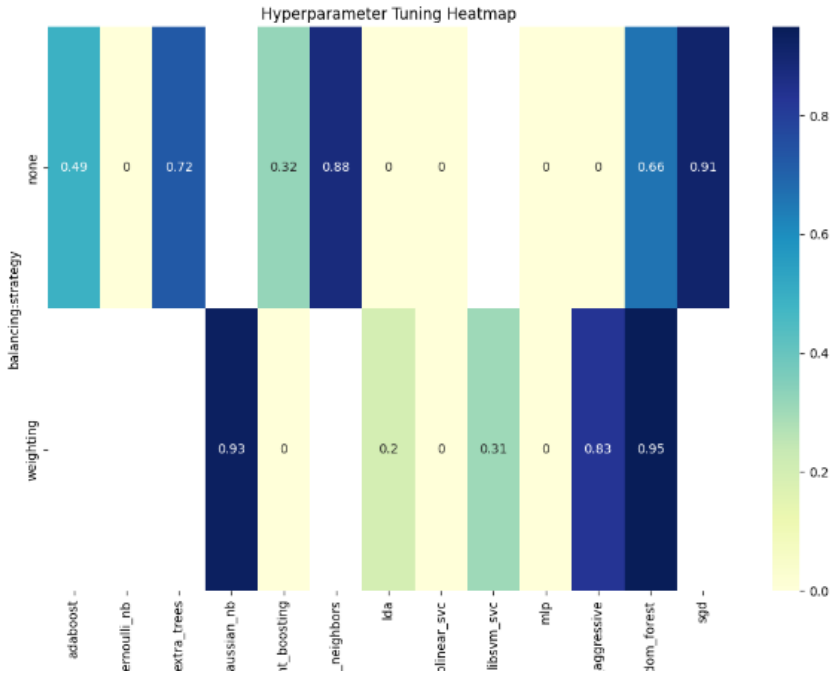

Figura 44 - *Heatmap* sobre híper-parâmetros

Em seguida é mostrada informação sobre importância de *features*, como se pode observar na figura 45, que mostra as *features* com mais de 10% de importância, e também um gráfico de bigodes que também tenta refletir a importância só que de uma maneira mais geral, como se pode observar na figura 46.

```
Features com mais de 10% de importância:
objfunded: 15.86% +/- 0.77%
maritalst: 10.59% +/- 1.12%
```

Figura 45 - Informação sobre importância de *features*

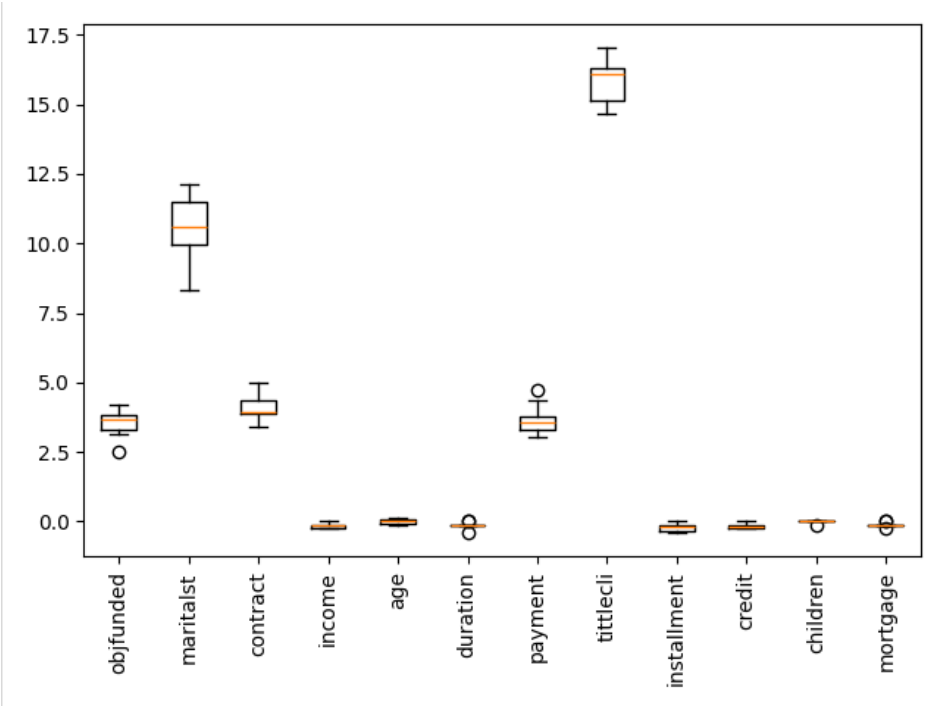

Figura 46 - Informação geral sobre importância de *features*

É também mostrado ao utilizador a matriz de confusão e o relatório de classificação, como se pode observar respetivamente nas figuras 47 e 48.

```
Confusion Matrix:
[[126   8]
 [  6 618]]
```

Figura 47 - Matriz confusão

```
Classification Report:
              precision    recall  f1-score   support

         bad       0.95      0.94      0.95       134
        good       0.99      0.99      0.99       624

    accuracy                           0.98       758
   macro avg       0.97      0.97      0.97       758
weighted avg       0.98      0.98      0.98       758
```

Figura 48 - Relatório de classificação

Outras informações mostradas ao utilizador são dados relativos aos modelos treinados dentro da tarefa geral, como se pode observar na figura 49, onde aparecem os modelos ordenados pela avaliação média.

```
Resultados ordenados pela avaliação média
    mean_test_score  ...  param_feature_preprocessor:select_rates_classification:mode
33         0.981962  ...                                                        NaN
3          0.981522  ...                                                        NaN
44         0.981522  ...                                                        NaN
5          0.981522  ...                                                        NaN
49         0.981082  ...                                                        NaN
32         0.979762  ...                                                        NaN
47         0.978883  ...                                                        NaN
26         0.978003  ...                                                        NaN
```

Figura 49 - Lista modelos ordenados por classificação média

E também uma tabela de classificação dos mesmos modelos treinados na tarefa, como se observa na figura 50.

```
Tabela das classificações dos modelos treinados
          rank  ensemble_weight  ... balancing_strategy          config_origin
model_id                         ...
35           1             0.22  ...               none           Local Search
11           2             0.04  ...               none          Initial design
46           3             0.08  ...               none          Random Search
8            4             0.16  ...               none          Initial design
51           5             0.00  ...               none          Random Search
34           6             0.10  ...               none           Local Search
49           7             0.06  ...               none           Local Search
```

Figura 50 - Tabela classificação dos modelos da tarefa

Para além disso são também mostradas as estatísticas gerais da tarefa executada, como se pode observar na figura 51.

```
Estatísticas do sprint de treino
auto-sklearn results:
  Dataset name: 77eb50ca-6a50-11ef-8171-00155dc89abf
  Metric: accuracy
  Best validation score: 0.981962
  Number of target algorithm runs: 51
  Number of successful target algorithm runs: 25
  Number of crashed target algorithm runs: 20
  Number of target algorithms that exceeded the time limit: 6
  Number of target algorithms that exceeded the memory limit: 0
```

Figura 51 - Estatísticas da tarefa

E por fim um gráfico que pretende dar o utilizador uma melhor perceção da performance, medida através da *accuracy* ao longo do tempo, como se observa na figura 52.

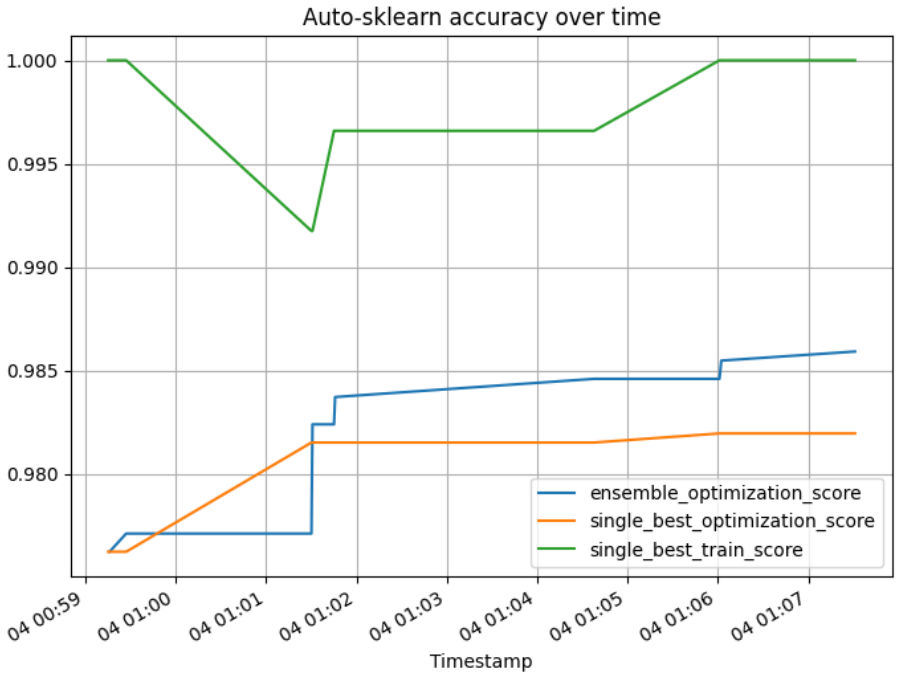

Figura 52 - Performance ao longo do tempo

Depois de analisar toda a informação disponibilizada o utilizador pode fechar a janela através do botão “Fechar e Continuar” que se encontra na figura X. Uma vez selecionado o sistema vai tentar guardar o modelo e as suas informações no MLFlow e em caso de sucesso retorna uma mensagem de sucesso como se pode observar na figura 53.

```
Successfully registered model 'leasing_8.pkl'.
2024/09/04 01:25:28 INFO mlflow.store.model_registry.abstract_store: Waiting up to 300 seconds for model version to finish creation. Model name: leasing_8.pkl, version 1
Created version '1' of model 'leasing_8.pkl'.
Model logged successfully.
```

Figura 53 - Confirmação sobre processo que guarda as informações do modelo no MLFow

Quando este processo termina, o utilizador volta para a interface de treino de modelo onde pode escolher realizar um novo treino, ou iniciar o processo de *deployment*.

Contudo o utilizador também pode aceder à instância do MLFlow para analisar os modelos guardados e as suas informações. Na figura 54 pode observar-se o aspeto da interface inicial do MLFLow.

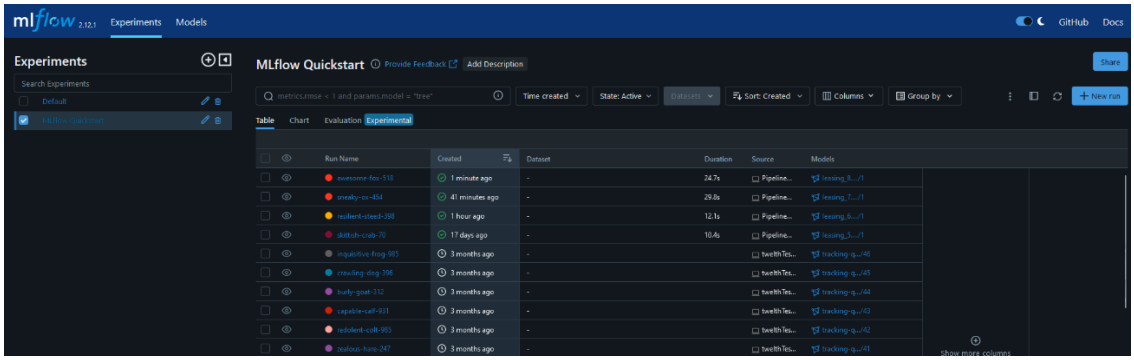

Figura 54 - Interface inicial MLFlow

Ao analisar a imagem percebe-se que um modelo foi adicionado há 1 minuto, que foi exatamente o modelo que foi treinado previamente para esta demonstração. Através desta interface podem analisar-se os diferentes modelos que foram treinados e compará-los entre si de uma maneira mais simples e intuitiva.

Se o utilizador quiser observar mais informações específicas relativas a algum dos modelos guardados só necessita de selecionar o respetivo modelo e a interface da figura 55 vai ser mostrada.

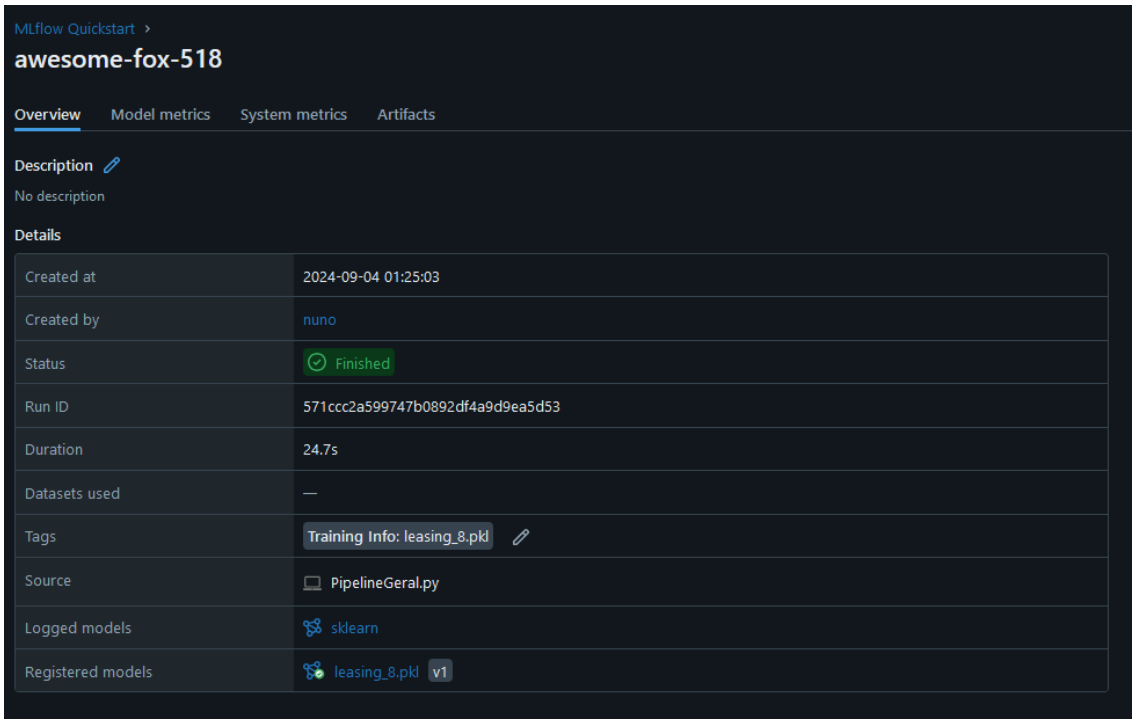

Figura 55 - Informação geral sobre um modelo selecionado

Através da figura pode observar-se que alguns detalhes gerais sobre o modelo criado são guardados para ajudar o utilizador a distinguir cada um dos modelos criados. Para além de detalhes gerais são também guardados os parâmetros de treino e algumas métricas de avaliação, como se pode observar nas figuras 56 e 57.

arameters (29)

Search parameters

Parameter	Value
memory_limit	None
metric	accuracy
include	None
seed	1
smac_scenario_args	None
initial_configurations_via_metalearning	25
resampling_strategy	cv
n_jobs	None
resampling_strategy_arguments	None
ensemble_size	None
disable_evaluator_output	False
allow_string_features	True
metadata_directory	None
max_models_on_disc	50
logging_config	None
tmp_folder	None
dataset_compression	True
load_models	True
get_trials_callback	None

Metrics (1)

Search metrics

Metric	Value
accuracy	0.45809718520052 1

Figura 56 - Parâmetros e métrica de avaliação guardada no MLFow

delete_tmp_folder_after_terminate	True
dask_client	None
scoring_functions	[accuracy, f1, roc_auc, precision, recall, log_loss]
ensemble_kwargs	None
ensemble_nbest	50
time_left_for_this_task	600
per_run_time_limit	45
exclude	None
get_smac_object_callback	None
ensemble_class	<class 'autosklearn.ensembles.ensemble_selection.EnsembleSelection'>

Figura 57 - Continuação da lista de parâmetros guardados no MLFow

Como se pode confirmar todos os parâmetros que foram selecionados como configurações pelo utilizador na interface do modelo de treino estão guardados nesta seção o que pode ser útil caso queira repetir configurações de treino, ou até ajustar alguns parâmetros de acordo com as observações dos resultados dos modelos.

Para além da seção da *Overview* que está a ser mostrada na figura 55, podem também ver-se seções, como por exemplo, sobre as métricas de avaliação guardadas e os artefactos do modelo. Relativamente aos artefactos podem ser bastante úteis uma vez que, como se pode verificar na figura 58, se pode obter informação sobre o esquema dos inputs e outputs esperados pelo modelo treinado.

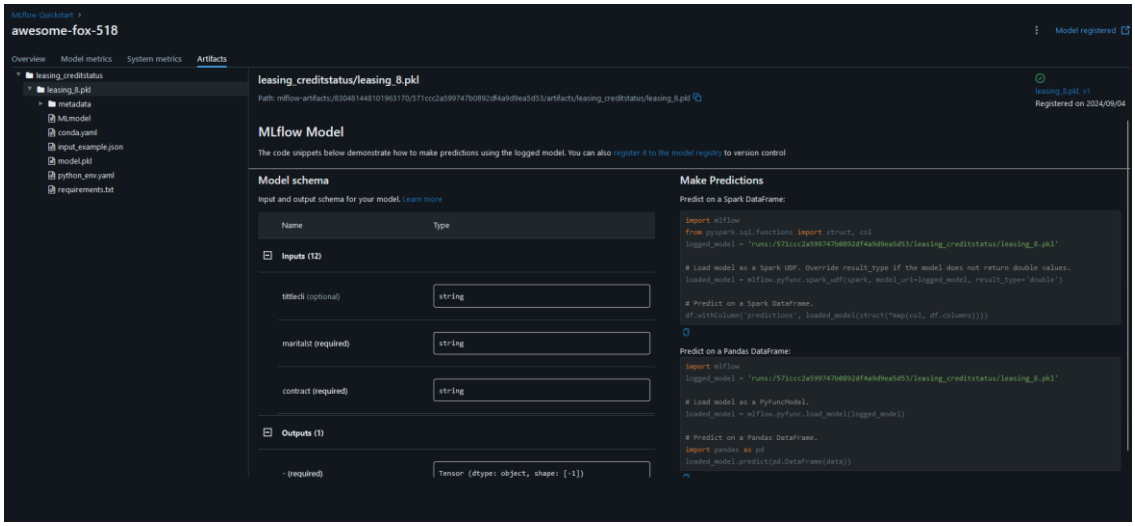

Figura 58 - Visualização da seção sobre artefactos do modelo guardados

Voltando para a aplicação o utilizador já tem a ideia de qual o modelo que acha mais adequado para ser implantado e disponibilizado para realizar previsões e então clica no botão “Iniciar deployment” que está na interface de treinar modelo. Ao clicar neste botão é mostrada uma lista de todos os modelos treinados que servem para fazer previsões para o conjunto de dados selecionado na recolha de dados, mais especificamente, na pasta cuja explicação foi feita no início desta seção, e que combina o nome do conjunto de dados com o atributo objetivo do mesmo. O aspeto da lista para este exemplo pode visualizar-se na figura 59.

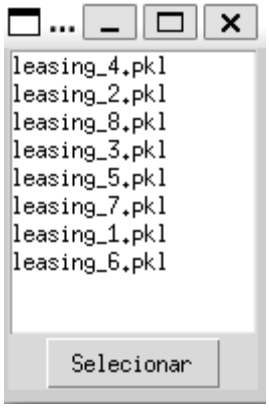

Figura 59 - Lista com opções de modelos adequadas para problema

Da lista exibida é então selecionado o modelo que foi criado em último, que neste caso é o “leasing_8”, como também se observou no MLFlow através da *tag* do modelo criada. Depois de selecionar o modelo aparece uma interface para confirmar a seleção como na figura 60.

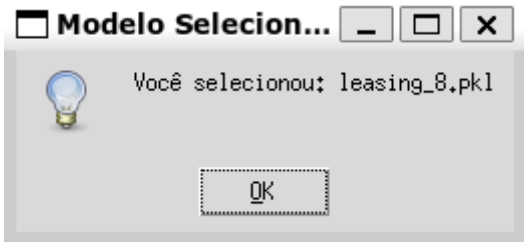

Figura 60 - Interface de confirmação do modelo escolhido

Uma vez confirmada através do botão “OK”, os dados mostrados na figura 51 aparecem.

```
Run ID: 571ccc2a599747b0892df4a9d9ea5d53
Downloading artifacts:   0%|          | 0/10 [00:00<?, ?it/s]
Downloading artifacts: 100%|██████████| 10/10 [00:03<00:00,
Starting local web server...
 * Serving Flask app 'EstrategiaDeployment'
 * Debug mode: off
```

Figura 61 - Processo de download dos artefactos e *deploy* do modelo

Como se pode observar na figura 61 o modelo foi carregado a partir da instância de MLFlow, e partir dai foi ativo o servidor para efetuar previsões com o modelo treinado.

Para simular a interação com uma aplicação foi usado o Postman como se pode analisar na figura 62.

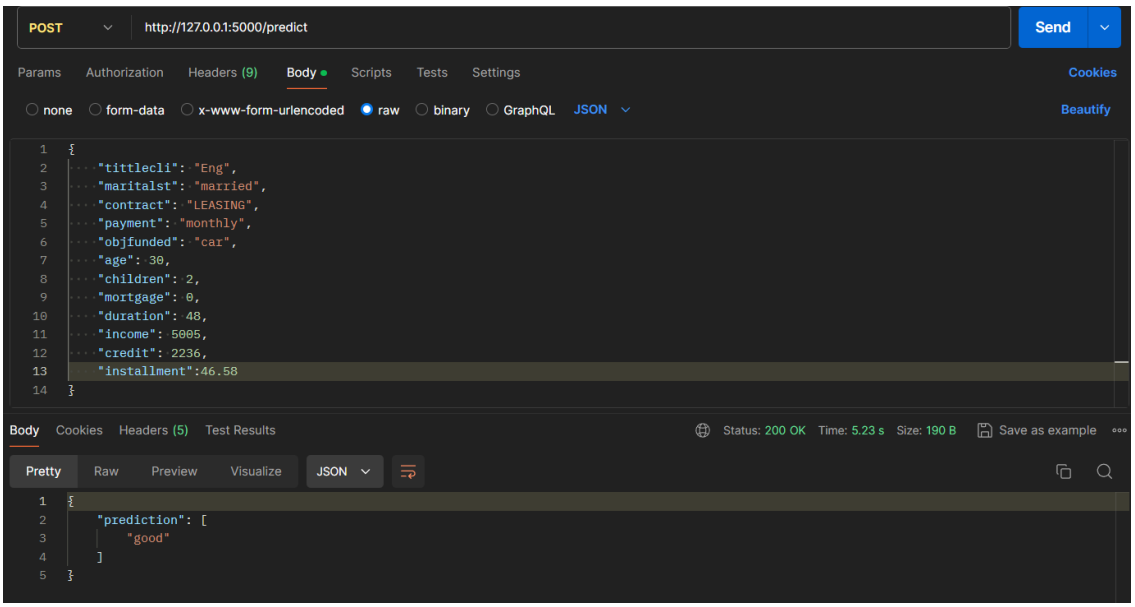

Figura 62 - Simulação de uma interação com sistema através do Postman

Como se observa definido o *url* para a aplicação Flask criada, para a rota associada à funcionalidade de executar previsões, definindo o pedido como um *POST* e colocando a informação com o esquema de input que o modelo necessita, o pedido é enviado e como se pode observar na resposta é obtido um status 200, o que significa que conseguiu contactar sem problema com o sistema Flask e para além disso recebeu a previsão que recebeu para os dados submetidos foi "good", ou seja o "creditstatus" para esta instância é previsto como sendo bom.

5 Experiências efetuadas e avaliação da solução

Nesta seção vão ser explicados os testes que vão ser realizados de modo a avaliar o sistema desenvolvido. Para isso vão ser descritos os cenários de avaliação, depois os resultados dos testes, uma avaliação de um prisma ético e por último uma discussão dos testes apresentados.

5.1 Metodologia

Em primeiro lugar, vão ser definidos alguns cenários para as experiências de forma avaliar diferentes situações e parâmetros, podendo assim avaliar melhor o impacto e o valor da solução desenvolvida.

Para realizar os testes foram escolhidos 2 conjuntos de dados com características diferentes:

- CD1 - Titanic Dataset (https://www.kaggle.com/datasets/yasserh/titanic-dataset): representa um conjunto de dados simples e mais pequeno (891 entradas e 12 variáveis diferentes);
- CD2 - Adult Census Income (https://www.kaggle.com/datasets/uciml/adult-census-income): representa um conjunto de dados mais complexo e maior (32600 entradas e 15 variáveis diferentes).

Em seguida vão ser explicados os vários cenários onde os conjuntos de dados vão ser usados de modo a avaliar a aplicação desenvolvida:

- Teste 1: Aplicação vs métodos tradicionais
    - Cenário 1: Neste cenário vai ser usado CD1
        - Critérios de avaliação: Tempo execução, métricas de avaliação de modelos, neste caso accuracy e precision, comparação de curva de aprendizagem, interações a executar e tempo de desenvolvimento.
    - Cenário 2: Neste cenário vai ser usado CD2
        - Critérios de avaliação: Tempo execução, métricas de avaliação de modelos, neste caso accuracy e precision, comparação de curva de aprendizagem, interações a executar e tempo de desenvolvimento.

- Teste 2: Aplicação vs Auto-sklearn
    - Cenário 1: Neste cenário vai ser usado CD1
        - Critérios de avaliação: Tempo execução, métricas de avaliação de modelos, neste caso accuracy e precision, comparação de curva de aprendizagem, interações a executar e tempo de desenvolvimento.

    - Cenário 2: Neste cenário vai ser usado CD2
        - Critérios de avaliação: Tempo execução, métricas de avaliação de modelos, neste caso accuracy e precision, comparação de curva de aprendizagem, interações a executar e tempo de desenvolvimento.

- Teste 3: Aplicação com pipeline customizada vs Aplicação sem pipeline customizada
    - Cenário 1: Neste cenário vai ser usado CD1
        - Critérios de avaliação: Métricas de avaliação de modelos, neste caso *accuracy* e *precision*.
    - Cenário 2: Neste cenário vai ser usado CD2
        - Critérios de avaliação: Métricas de avaliação de modelos, neste caso *accuracy* e *precision*.

O principal objetivo com o primeiro teste é perceber qual o impacto e a diferença entre usar a aplicação e usar os métodos tradicionais de ML tanto em termos de performance, ou seja, métricas de avaliação de desempenho dos modelos, e também pela acessibilidade a utilizadores com menos capacidades técnicas na área através da análise da curva de aprendizagem. Para além disso também vai ser comparado o tempo de desenvolvimento de cada uma delas, ou seja, o tempo de execução desde o início do processo de treino até ter o modelo pronto a efetuar previsões e também comparar o número de interações que o utilizador necessita de executar.

Com o segundo teste o objetivo é perceber o que a aplicação acrescentou ao Auto-sklearn. Para isto vão ser avaliados tempo de execução do processo, métricas de avaliação, curva de aprendizagem e tempo de desenvolvimento e número de interações.

Com o terceiro teste o objetivo é perceber quais os impactos da pipeline customizada na performance dos modelos e para isso vão ser testados tempo de execução e métricas de avaliação.

Nos três testes vai ser possível comparar os resultados para os dois conjuntos de dados propostos de maneira a testar a escalabilidade da solução uma vez que o segundo conjunto de dados é 10 vezes maior em termos de entradas do que o primeiro conjunto de dados.

5.2 Resultados dos testes efetuados

A tabela 2 apresenta os testes relativos ao primeiro cenário.

Tabela 2: Testes cenário 1

Solução para treino	Conjunto de dados	Tempo de execução (segundos)	Accuracy	Precision	Curva Aprendizagem	Interações até obter resultado	Tempo de desenvolvimento
Aplicação	CD1	600	O,82	Negativo: 0,82; Positivo: 0,82	Rápida	Interface	Imediato
Aplicação	CD2	600	0,87	<=50k:0,89; >50k:0,79	Rápida	Interface	Imediato
Tradicional	CD1	115.56	0,81	Negativo: 0,84; Positivo: 0,74	Lenta	Implementação de código	Médio/Alto
Tradicional	CD2	10232.77	0,85	<=50k:0,88; >50k:0,74	Lenta	Implementação de código	Médio/Alto

Como se pode observar na tabela, foram postas em comparação para as duas soluções diferentes, a aplicação desenvolvida e uma solução mais tradicional, tempo de execução, métricas de avaliação, mais precisamente accuracy e precision e para além disso três parâmetros, curva de aprendizagem, interações até obter resultado e tempo de desenvolvimento. A curva de aprendizagem que neste contexto significa principalmente a facilidade de usufruir das funcionalidades oferecidas pelas soluções, para estas duas soluções é praticamente oposta. Para a aplicação é rápida devido à interface que torna todo o processo intuitivo e simples para qualquer utilizador. Enquanto para o processo tradicional o utilizador tem de possuir um grande conhecimento da área científica e da área do conjunto de dados com que está a trabalhar. Em relação às interações, todas as ações que o utilizador necessita realizar são automatizadas através da interface, por outro lado no processo mais tradicional é necessário escrever código para alem de ter um conhecimento grande da área uma vez que todos os processos da pipeline têm de ser implementados individualmente pelo utilizador. O terceiro parâmetro, o tempo de desenvolvimento, vem diretamente do último parâmetro analisado. Para a aplicação o processo de desenvolvimento é imediato uma vez que o utilizador só necessita de interagir com a interface para fazer as suas escolhas e efetuar o treino do modelo e o do seu respetivo *deploy*. Para além de imediato o processo ainda acrescenta automaticamente etapas de explicação dos processos e dos resultados. Nas soluções tradicionais, o tempo de desenvolvimento é grande, pois mesmo que o utilizador possua um conhecimento extensivo na área necessita de realizar a implementação de todas as etapas o que acaba sempre por consumir tempo. Relativamente ao tempo de treino e às métricas de avaliação. Para o CD1 a diferença da accuracy foi de 1% vinda de uma diferença entre 0.82 e

0.81, com uma ligeira vantagem para o método tradicional para além de o tempo que o método tradicional demorou a efetuar este processo foi um quinto do tempo que a aplicação demorou. Em termos de precision a variação para o valor “Negativo” vem da diferença entre 0.84 e 0.82, que é 0.02 e do valor “Positivo”, vem da diferença entre 0.82 e 0.74 que é 0.08. Para o CD2 o tempo de execução do método tradicional foi mais de dez vezes maior do que o tempo definido para a aplicação. Em termos de resultados, a accuracy tem uma variação de 2% a favor da aplicação. Relativamente à precision para “<=50k” a variação foi de 0.01 a favorecer a aplicação e para “>50k” foi 0.05. Ou seja, analisando estes valores pode-se concluir que com o aumento da complexidade e volume de dados, a aplicação, com base nestes testes, tem pelo menos um desempenho tão bom quanto os métodos tradicionais testados, possuindo a aplicação alguma vantagem relativamente à facilidade do processo e ao tempo do processamento.

Em seguida vão ser apresentados na tabela 3 os resultados do segundo teste.

Tabela 3: Testes cenário 2

Solução para treino	Conjunto de dados	Tempo de execução (segundos)	Accuracy	Precision	Curva Aprendizagem	Interações até obter resultado	Tempo de desenvolvimento
Aplicação	CD1	600	O,82	Negativo:0,82; Positivo: 0,82	Rápida	Interface	Imediato
Aplicação	CD2	600	0,87	<=50k:0,89; >50k:0,79	Rápida	Interface	Imediato
Auto-sklearn	CD1	600	0,81	Negativo:0,81; Positivo: 0,80	Normal	Implementação de código	Médio/Baixo
Auto-sklearn	CD2	600	0,87	<=50k:0,84; >50k:0,77	Normal	Implementação de código	Médio/Baixo

Observando os dados da tabela pretende-se perceber quais são as grandes diferenças entre o uso da aplicação e o uso do Auto-sklearn diretamente. De notar que para efetuar este teste fez-se o uso da pipeline de pré-processamento da aplicação de modo a ter uma distância ainda maior entre os processos usados pela aplicação e os processos usados diretamente pelo Auto-skelarn. Em primeiro lugar, comparando as métricas de avaliação definidas. Para os dois conjuntos de dados o valor da accuracy parece ser praticamente constante nas duas abordagens uma vez que para o CD2 não apresenta qualquer variação e para o CD1 apresenta uma variação de 1%, residual. Relativamente à precision do CD1, para "Negativo" apresenta uma variação de X, diferença entre o valor máximo 0.82 e o valor mínimo 0,.81. Para o "Positivo", apresenta uma variação de 2%, devido também à diferença entre 0.82 e 0.80. Para o CD2, a precision para "<=50k" tem uma variação de 0.5, devido à diferença entre 0.89 e 0.84 e para ">50k" tem variação de 0.02, da diferença entre 0.79 e 0.77. Ou seja, em termos de métricas podemos dizer que as soluções são muito parecidas, o que faz sentido, uma vez que o core principal da aplicação desenvolvida é a solução com a qual está a ser comparada. A diferença chave das duas soluções vem principalmente da análise das variáveis seguintes. Em primeiro lugar curva de aprendizagem e que tem o mesmo significado que na tabela anteriormente analisada. Enquanto para a aplicação a curva é praticamente constante e pode ser considerada rápida devido ao uso de uma interface gráfica que aumenta em muito a usabilidade da solução, para a outra solução é necessário ter conhecimento da área para se analisar a documentação oficial e conseguir depois de algum estudo usar as funcionalidades básicas da mesma, requerendo que para utilizar toda a sua capacidade seja feito um estudo ainda mais elaborado da mesma. O segundo ponto de comparação e distinção é o critério das interações até obter

resultado. Enquanto na aplicação o utilizador interage com uma interface que lhe permite tomar todas as decisões através de botões, o que dá grandes oportunidades a utilizadores que não tenham conhecimento da área, a outra solução necessita que seja implementado código para usufruir do seu funcionamento. Como último critério, o tempo despendido para desenvolvimento. Uma vez que a aplicação é uma solução *no-code*, o seu tempo de desenvolvimento para o utilizador conseguir treinar um modelo está associado ao tempo que o utilizador vai demorar a escolher o modelo e as configurações que pretende aplicar. Por outro lado, exclusivamente usando o Auto-sklearn, apesar de muitas das etapas serem automatizadas, o que diminui em muito o tempo em comparação com o método mais tradicional, uma vez que mantém a necessidade de se escrever código continua a ter um tempo de desenvolvimento associado. Comparando o tempo atribuído à tarefa com as métricas de avaliação para os dois conjuntos e para as duas soluções pode-se afirmar que em termos de escalabilidade as duas soluções respondem da mesma maneira, com o aumento do tamanho dos conjuntos de dados uma vez que o seu core na implementação é o mesmo.

Por último vão ser apresentados na tabela 4 os resultados do terceiro teste.

Tabela 4: Teste cenário 3

Solução para treino	Conjunto de dados	Uso da pipeline customizada	Tempo de execução (segundos)	Accuracy	Precision	Ensemble	Tempo máximo de run (segundos)
Aplicação	CD1	Não	600	0,81	Negativo:0,83; Positivo: 0,78	Sim	60
Aplicação	CD1	Não	600	0,82	Positivo:0,82; Negativo: 0,83	SingleBest	60
Aplicação	CD1	Sim	600	0,82	Negativo:0,82; Positivo: 0,82	Sim	60
Aplicação	CD2	Não	600	0,86	<=50k:0,89; >50k:0,76	SingleBest	60
Aplicação	CD2	Não	300	0,87	<=50k:0,88; >50k:0,79	Sim	60
Aplicação	CD2	Não	600	0,87	<=50k:0,88; >50k:0,81	Sim	60
Aplicação	CD2	Não	600	0,87	<=50k:0,89; >50k:0,79	Sim	120
Aplicação	CD2	Não	1200	0,87	<=50k:0,89; >50k:0,78	Sim	120
Aplicação	CD2	Sim	300	0,87	<=50k:0,88; >50k:0,79	Sim	60
Aplicação	CD2	Sim	600	0,87	<=50k:0,89; >50k:0,79	Sim	60
Aplicação	CD2	Sim	600	0,87	<=50k:0,89; >50k:0,76	Sim	120
Aplicação	CD2	Sim	1200	0,87	<=50k:0,88; >50k:0,79	Sim	120
Aplicação	CD2	Sim	600	0,87	<=50k:0,89; >50k:0,77	Não	60

Com a análise dos resultados presentes na tabela observa-se que em termos de accuracy e precision dos modelos treinados, para o intervalo de tempo definido para as tarefas que foi entre 300, 600 ou 1200 segundos, que é respetivamente, 5, 10 ou 20 minutos, observou-se que a performance obtida é praticamente constante, mantendo-se praticamente igual em termos de accuracy, e tendo uma variação pequena em termos de precision. Para o CD1 a variação entre o valor máximo e o valor mínimo obtidos para a precision de “Positivo” foi 0.01, vindo da diferença entre o valor 0.83 e o valor 0.82, já para “Negativo”, a variação foi de 0.05, diferença entre os valores 0.83 e 0.78. Para o CD2, a variação para a precision de “<=50k” foi de 0.01, diferença entre o valor 0.89 e 0.88, já para “>50k” foi de 0.05, da diferença entre o valor 0.81 e

o valor 0.76. A partir destes dados pode concluir-se que para o intervalo de tempo estudado as manipulações das diversas configurações disponíveis como tempo máximo definido para uma run, tempo para execução da tarefa e uso de ensemble, têm pouco impacto nas métricas de avaliação, pois a variação máxima não é maior do que 5%. O mesmo se pode concluir com o uso da pipeline de pré-processamento customizada ou não para os dois conjuntos de dados explorados, pois nas situações em que se aplicou não houve praticamente diferença nas métricas de avaliação para as mesmas configurações de tempo. Pode ainda concluir-se que o aumento do parâmetro de configuração tempo máximo definido para uma run, não acrescenta valor nas métricas de avaliação, pois devido à análise das estatísticas das tarefas podia-se constatar que o número de modelos que falhavam porque excediam esse tempo não diminuía drasticamente com o aumento dessa variável. Por último, ao analisar e comparar os resultados dos dois conjuntos, que com o aumento do tamanho e complexidade dos dados não houve uma perda de performance significativa, ou seja, o CD1 é 10x mais pequeno que o CD2, contudo para o mesmo tempo definido por tarefa foi possível obter resultados até melhores do que para o CD1 se analisarmos exclusivamente a proporção tamanho do conjunto com a avaliação obtida para as métricas comparadas na tabela.

Após a análise dos 3 cenários propostos, onde para além de se analisar algumas métricas de avaliação e tempo de execução, foram também analisados fatores de usabilidade das várias estratégias usadas. Depois desta primeira análise, as soluções vão ser novamente testadas com o recurso a mais 11 conjuntos de dados com diversas características distintas de modo a ajudar a fundamentar melhor as primeiras conclusões retiradas ou até mesmo retirar novas. Nestes conjuntos de dados vão só ser exploradas as soluções usando a aplicação com pipeline de pré-processamento customizado, a aplicação sem o pré-processamento customizado e o Auto-sklearn, uma vez que nos dois conjuntos de dados anteriores não se obtiveram grandes variações nas métricas avaliadas. Por outro lado, os resultados com o uso da solução tradicional tiveram uma diferença muito expressiva quando comparado às soluções automatizadas e para além disso alguns destes novos conjuntos de dados são muito grandes tanto em número de entradas quanto em número de variáveis o que vai tornar este processo de teste muito demorado e não vai adicionar ou mostrar diferenças nas observações previamente apresentadas.

Em seguida vão ser apresentados os diversos conjuntos de dados usados e uma tabela com as experiências efetuadas para cada um deles.

CD3 -Default of Credit Card Clients Dataset (https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset): 30000 entradas e 25 variáveis diferentes. Como principais desafios este conjunto apresenta um grande desbalanceamento da variável objetivo usada “default.payment.next.month”, para além de algumas variáveis apresentarem uma grande correlação. Relativamente a este conjunto de dados foram efetuadas as experiências registadas na tabela 5.

Tabela 5: Testes CD3

Solução para treino	Tempo de execução (segundos)	Tempo máximo de run (segundos)	Accuracy	Precision
Aplicação com pipeline customizada de pré-processamento	600	60	0.82	0:0.84; 1:0.66
Aplicação sem pipeline customizada de pré-processamento	600	60	0.82	0:0.84; 1:0.66
Auto-sklearn	600	60	0.82	0:0.84; 1:0.68

Como se pode observar para as três soluções analisadas, as métricas não apresentam uma variação significativa, sendo essa maior variação de 2% na precisão para o valor '1'.

CD4 - Bank Marketing Dataset (https://www.kaggle.com/datasets/janiobachmann/bank-marketing-dataset):11200 entradas; 17 variáveis diferentes. Como principais desafios para este conjunto identificam-se o desbalanceamento da variável alvo "deposit", a presença de algumas variáveis temporais críticas e a falta de valores em alguns campos. Relativamente a este conjunto de dados foram efetuadas as experiências registadas na tabela 6.

Tabela 6: Testes CD4

Solução para treino	Tempo de execução (segundos)	Tempo máximo de run (segundos)	Accuracy	Precision
Aplicação com pipeline customizada de pré-processamento	600	60	0.84	no:0.86; yes:0.83
Aplicação sem pipeline customizada de pré-processamento	600	60	0.85	no:0.87; yes:0.83
Auto-sklearn	600	60	0.83	no:0.86; yes:0.81

Como se pode observar para as soluções apresentadas as métricas não apresentam uma variação considerável, sendo esta não superior a no máximo 2%.

CD5 – Heart Disease Dataset (UCI Cleveland) (https://www.kaggle.com/datasets/ritwikb3/heart-disease-cleveland): 303 entradas; 14 variáveis diferentes. O principal desafio é a escassez de dados, devido ao tamanho reduzido do conjunto de dados. Relativamente a este conjunto de dados foram efetuadas as experiências registadas na tabela 8.

Tabela 7 - Testes CD5

Solução para treino	**Tempo de execução (segundos)**	**Tempo máximo de run (segundos)**	**Accuracy**	**Precision**
Aplicação com pipeline customizada de pré-processamento	**600**	**60**	**0.84**	**0:0.83; 1:0.86**
Aplicação sem pipeline customizada de pré-processamento	**600**	**60**	**0.88**	**0:0.89; 1:0.87**
Auto-sklearn	**600**	**60**	**0.90**	**0:0.87; 1:0.93**

Como se pode observar houve uma diferença de 4% entre a accuracy de solução usando a aplicação com pipeline de pré-processamento customizada e sem e de 6% entre a primeira referida e o Auto-sklearn. Para além disso analisando a precision também se analisa uma diferença máxima de 7% entre a aplicação com pipeline de pré-processamento e o Auto-sklearn. Como é um conjunto pequeno de dados o uso das ferramentas de pré-processamento presentes na pipeline pode ter eliminado alguns dados o que deixou a amostra que é fornecida para o treino do modelo ainda menor, o que pode ter resultado numa pior performance.

CD6 – Telco Customer Churn (https://www.kaggle.com/datasets/blastchar/telco-customer-churn/data): 7043 entradas; 21 variáveis diferentes. Os principais desafios são o desbalanceamento da variável alvo “Churn” e a correlação entre variáveis. Relativamente a este conjunto de dados foram efetuadas as experiências registadas na tabela 9.

Tabela 8: Testes CD6

Solução para treino	**Tempo de execução (segundos)**	**Tempo máximo de run (segundos)**	**Accuracy**	**Precision**
Aplicação com pipeline customizada de pré-processamento	**600**	**60**	**0.80**	**no:0.83; yes:0.67**
Aplicação sem pipeline customizada de pré-processamento	**600**	**60**	**0.79**	**no:0.83; yes:0.66**
Auto-sklearn	**600**	**60**	**0.81**	**no:0.83; yes:0.71**

Como se pode observar não há grande variação das métricas de avaliação para as diferentes soluções. Sendo a variação máxima para a accuracy de 2% e para a precision em 'yes' 4%. De notar a diferença considerável entre a precision para os dois valores possíveis de classificação o que aponta neste caso para alguma dificuldade em lidar com o desbalanceamento de dados.

CD7 – Heart Failure Prediction Dataset (https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction/data): 918 entradas; 12 variáveis diferentes. Principais características, são a o tamanho pequeno do conjunto de dados e a presença exclusiva de variáveis numéricas. Relativamente a este conjunto de dados foram efetuadas as experiências registadas na tabela 10.

Tabela 9: Testes CD7

Solução para treino	Tempo de execução (segundos)	Tempo máximo de run (segundos)	Accuracy	Precision
Aplicação com pipeline customizada de pré-processamento	**600**	**60**	**0.88**	**0:0.84;1:0.91**
Aplicação sem pipeline customizada de pré-processamento	**600**	**60**	**0.88**	**0:0.84;1:0.91**
Auto-sklearn	**600**	**60**	**0.88**	**0:0.85;1:0.90**

Como se pode observar não há praticamente variação das métricas para as três soluções apresentadas.

CD8 – Haberman's Survival Data Set (https://www.kaggle.com/datasets/gilsousa/habermans-survival-data-set): 306 entradas; 4 variáveis diferentes. Os principais desafios são o número limitado de varáveis e o desbalanceamento do atributo alvo "surv_status". Relativamente a este conjunto de dados foram efetuadas as experiências registadas na tabela 11.

Tabela 10: Testes CD8

Solução para treino	Tempo de execução (segundos)	Tempo máximo de run (segundos)	Accuracy	Precision
Aplicação com pipeline customizada de pré-processamento	**600**	**60**	**0.70**	**1:0.76;2:0.47**
Aplicação sem pipeline customizada de pré-processamento	**600**	**60**	**0.69**	**1:0.75;2:0.43**
Auto-sklearn	**600**	**60**	**0.65**	**1:0.72;2:0.33**

Como se pode observar para a accuracy e precision da solução com Auto-sklearn sofre uma ligeira diferença da solução que usa a aplicação com a pipeline de pré-processamento customizada, sendo 5% para o primeiro e 4% para possibilidade '1' e 14% para possibilidade '2'. Um aspeto que se pode observar é a diferença da precision para as duas possibilidades, tendo uma variação máxima de 43% entre o melhor e o pior valor dos 3 e de 39% para a maior diferença entre os dois valores dentro da mesma solução.

CD9 – Thyroid Disease Data Set (https://www.kaggle.com/datasets/yasserhessein/thyroid-disease-data-set): 3772 entradas; 30 variáveis diferentes. Apresenta uma dimensionalidade média/alta como um dos desafios principais. Relativamente a este conjunto de dados foram efetuadas as experiências registadas na tabela 12.

Tabela 11: Testes CD9

Solução para treino	**Tempo de execução (segundos)**	**Tempo máximo de run (segundos)**	**Accuracy**	**Precision**
Aplicação com pipeline customizada de pré-processamento	**600**	**60**	**0.98**	**N:0.96; P:0.98**
Aplicação sem pipeline customizada de pré-processamento	**600**	**60**	**0.98**	**N:0.98; P:0.98**
Auto-sklearn	**600**	**60**	**0.92**	**N:0.00; P:0.92**

Como se pode observar em relação à accuracy das soluções que usam a aplicação à uma diferença de 6% em relação à que usa Auto-sklearn. Relativamente à precision, para as soluções que usam a aplicação tem uma variação máxima de 2%. Para a precision também se notou uma variação quando comparando aplicação e o Auto-skelearn variando para 'N' 96% e para 'P' 6%.

CD10 – Indian Liver Patient Records (https://www.kaggle.com/datasets/uciml/indian-liver-patient-records): 583 entradas; 11 variáveis diferentes. Os principais desafios são o tamanho moderado do conjunto de dados e desbalanceamento da variável alvo "Dataset". Relativamente a este conjunto de dados foram efetuadas as experiências registadas na tabela 13.

Tabela 12: Testes CD10

Solução para treino	**Tempo de execução (segundos)**	**Tempo máximo de run (segundos)**	**Accuracy**	**Precision**
Aplicação com pipeline customizada de pré-processamento	**600**	**60**	**0.71**	**1:0.75;2:0.25**
Aplicação sem pipeline customizada de pré-processamento	**600**	**60**	**0,68**	**1:0.74;2:0.15**
Auto-sklearn	**600**	**60**	**0.72**	**1:0.74;2:0.20**

Como se pode observar em termos de accrucay não há praticamente variação dos valores para as 3 soluções. Relativamente à precision pode-se notar que a solução com a pipeline customizada de pré-processamento apresenta uma diferença de 10% da segunda solução e de 5% da terceira, que usam o mesmo módulo interno de pré-processamento. De notar também a diferença de precision entre o valor '1' e '2', sendo associada ao desbalanceamento do atributo objetivo, apresentando no seu melhor caso, a primeiro solução, identificado uma diferença de 50%.

CD11 – Breast Cancer Wisconsin (Diagnostic) Data Set (https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data): 569 entradas; 32 variáveis diferentes. Como principal desafio é identificado a correlação e redundância de variáveis. Relativamente a este conjunto de dados foram efetuadas as experiências registadas na tabela 14.

Tabela 13: Testes CD11

Solução para treino	**Tempo de execução (segundos)**	**Tempo máximo de run (segundos)**	**Accuracy**	**Precision**
Aplicação com pipeline customizada de pré-processamento	**600**	**60**	**0.97**	**B:0.98; M:0.96**
Aplicação sem pipeline customizada de pré-processamento	**600**	**60**	**0.97**	**B:0.97; M:0.96**
Auto-sklearn	**600**	**60**	**0.98**	**B:0.97; M:1.0**

Como se pode observar não há praticamente variação das métricas de avaliação para as 3 soluções apresentadas.

CD12 – Santander Customer Transaction Prediction (https://www.kaggle.com/competitions/santander-customer-transaction-prediction/overview): 200000 entradas; 202 variáveis diferentes. Os principais desafios são a alta dimensionalidade e o grande volume de dados. Relativamente a este conjunto de dados foram efetuadas as experiências registadas na tabela 15.

Tabela 14: Testes CD12

Solução para treino	**Tempo de execução (segundos)**	**Tempo máximo de run (segundos)**	**Accuracy**	**Precision**
Aplicação com pipeline customizada de pré-processamento	**600**	**60**	**0.90**	**0:0.90; 1:0.68**
Aplicação sem pipeline customizada de pré-processamento	**600**	**60**	**0.90**	**0:0.90; 1:0.68**
Auto-sklearn	**600**	**60**	**0.90**	**0:0.90; 1:0.00**

Como se pode observar não há praticamente variação das métricas de avaliação para as 3 soluções apresentadas.

CD13 – Credit Card Fraud Detection (https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud): 284807 entradas; 31 variáveis diferentes. Os principais desafios são o desbalanceamento da variável objetivo “Class” e o grande volume de dados. Relativamente a este conjunto de dados foram efetuadas as experiências registadas na tabela 16.

Tabela 15: Testes CD13

Solução para treino	Tempo de execução (segundos)	Tempo máximo de run (segundos)	Accuracy	Precision
Aplicação com pipeline customizada de pré-processamento	**600**	**60**	**0.99**	**0:1.0; 1:0.86**
Aplicação sem pipeline customizada de pré-processamento	**600**	**60**	**0.99**	**0:1.0; 1:0.86**
Auto-sklearn	**600**	**60**	**0.99**	**0:1.0; 1:0.85**

Como se pode observar não há praticamente variação das métricas de avaliação para as 3 soluções apresentadas. Para a diferença entre os dois valores da precision pode-se identificar o desbalanceamento da variável objetivo o que pode ser um dos motivos que provoque essa diferença.

5.3 Discussão de resultados

Com estes três testes foi possível comparar desde soluções tradicionais, soluções de automatização e a aplicação desenvolvida. Em termos de performance pode se dizer que para conjuntos de dados pequenos e pouco complexos qualquer uma das soluções consegue responder com métricas de avaliação boas e tempos de execução reduzidos. À medida que a complexidade e o volume dos conjuntos de dados aumentam tende a ter-se uma melhor resposta da solução automatizada e da aplicação uma vez que na sua implementação abordam estratégias para lidar com a escalabilidade como por exemplo o uso de *meta-learning* para encontrar híper-parâmetros que gerem bons resultados mais rapidamente. Relativamente à facilidade de uso e acesso a utilizadores com menos conhecimento técnico, a aplicação desenvolvida destaca-se quando comparada com as outras abordagens principalmente por tirar a necessidade de implementação de código ao utilizador, tornando o processo simples e intuitivo. Para além de permitir esse perfil de utilizadores de se beneficiar com as funcionalidades das soluções automatizadas sem o esforço de implementar nenhuma linha de código, também auxilia quem tem conhecimento técnico.

Por outro lado, analisando a performance da pipeline de pré-processamento implementada na aplicação, pode concluir-se que em termos de influência nas métricas de avaliação possui tanta

quando o módulo de pré-processamento interno do Auto-sklearn uma vez que não há uma variação de valores acentuada. Para além disso foi testada a manipulação de configurações de treino para intervalos relativamente curtos de tempo, no máximo até 20 minutos, sendo os outros valores testados 5 e 10. Para essa gama de valores de tempo conclui-se que a performance não sofre um aumento significativo uma vez que as métricas se mantêm com uma variação praticamente nula. Para além dessa configuração foi também testada alteração do tempo máximo para uma *run* e conclui-se que o número de runs que falham por causa do tempo sem mantém praticamente constante o que indica que para este intervalo de tarefas e mantendo o tempo de *run* pelo menos em um décimo do que é o valor de tempo da tarefa não se vai ter alterações negativas na performance devido a um escasso número de modelos treinados.

Por último, para os intervalos de tempo testado não houve diferenças significativas entre usar o ensemble e não usar, contudo por observação do gráfico de *performance over time* apresentado na seção de explicação, é possível criar uma hipótese que diga que para intervalos de tempo maiores, o ensemble venha a melhorar os seus resultados e até a passar o melhor modelo treinado individualmente, uma vez que a sua linha ao longo do tempo, nos casos em que foi efetuado o teste, foi sempre observada a subir.

5.4 Avaliação de um ponto de vista ético

De um ponto de vista ético, o sistema desenvolvido pode ser avaliado tendo em conta seis parâmetros diferentes: Impacto social, viés e justiça do algoritmo, transparência e interpretabilidade dos dados, privacidade e proteção de dados, responsabilidade e sustentabilidade e impacto ambiental.

Do ponto de vista do impacto social, através da automatização de processos podem ser afetadas questões relacionadas com emprego e deslocamento de profissionais, pois ela pode reduzir a procura de especialistas humanos. Por um lado, a necessidade de intervenção humana em algumas tarefas é reduzida, e não eliminada, uma vez que a solução pode se vista como uma ferramenta para auxiliar o especialista a melhorar a sua produtividade e resultados. Por outro lado, do ponto de vista de um utilizador que não é especialista, permite com que ele tenha acesso a uma tecnologia sem ter um conhecimento de especialista da área. Relativamente ao viés e justiça dos algoritmos, para tentar mitigar o seu efeito a solução possui capacidade de efetuar o balanceamento de classes.

Para a transparência e interpretabilidade foi desenvolvido o componente explicado no design e implementação, explicação do modelo.

Para a privacidade e proteção dos dados, não são implementados quaisquer mecanismos de encriptação de dados ou outras estratégias com o mesmo intuito. Contudo durante todo o processo de execução do sistema o utilizador também não tem acesso direto ao conteúdo dos

conjuntos de dados que usa, servindo eles exclusivamente para treinar os algoritmos que depois dão origem ao modelo final.

Para a responsabilidade, o sistema ajuda o utilizador a tomar as suas alternativas de uma maneira mais guiada possível, contudo as escolhas finais são da autoria do utilizador, uma vez que o sistema não tem autonomia para tomar as decisões pelo mesmo, nem implementa qualquer estratégia de recomendação.

Por último, para sustentabilidade e impacto ambiental, o processo de treino de algoritmos de ML pode consumir uma grande quantidade de recursos computacionais, o que pode levar a um aumento do consumo de energia. Para combater esse aumento a aplicação através do Auto-sklearn implementa algumas técnicas que permitem reduzir os recursos computacionais usados durante o processo, principalmente na diminuição do tempo necessário para o processamento.

6 Conclusão

O principal objetivo desta dissertação foi analisar as diferentes fases do processo de ML assim como frameworks de AutoML e desenvolver uma aplicação que automatizasse o processo completo de uma pipeline de ML, desde a preparação de dados até ao deployment do modelo. A implementação proposta mostrou-se eficaz, conseguindo reduzir significativamente o tempo necessário para o treino de algoritmos de ML, tornando ainda o processo acessível para utilizadores com conhecimento limitado em ciência de dados.

Ao longo deste trabalho, foi possível verificar que a utilização de frameworks de AutoML, como as estudadas, facilita a criação de modelos preditivos de forma eficiente e rápida, sem sacrificar o desempenho dos algoritmos. A solução implementada foi capaz de escalar conforme o volume de dados aumentava, provando-se adequada para lidar com conjuntos de dados de diferentes dimensões, sem comprometer a qualidade dos resultados. Além disso, a aplicação demonstrou ser intuitiva e fácil de usar, permitindo que utilizadores menos experientes pudessem tirar proveito da automatização das tarefas de ML, o que cumpre um dos principais objetivos da dissertação.

Com o decorrer do trabalho algumas limitações foram identificadas. Em primeiro lugar, é relevante salientar a dependência dos algoritmos e métodos pré-definidos nas frameworks de AutoML selecionadas. Apesar de fornecerem um alto grau de automatização e otimização, essas frameworks restringem a personalização avançada de certos algoritmos e não permitem ajustes finos que poderiam ser necessários em cenários mais específicos. Isso pode limitar a capacidade dos utilizadores mais avançados de ajustar determinados parâmetros ou testar abordagens customizadas, que podem ser essenciais em alguns contextos. Adicionalmente, a execução local da aplicação em máquinas com recursos limitados impôs restrições ao processamento de grandes volumes de dados, limitando a escalabilidade da solução em ambientes com conjuntos de dados mais extensos. Outra limitação foi a compatibilidade da biblioteca de AutoML implementada na solução, com algumas ferramentas, como por exemplo Lime, ou SHAP que exigem os dados com um determinado formato o que se mostra difícil de uniformizar para todos os conjuntos de dados.

Com base nos resultados obtidos e nas limitações identificadas, há várias oportunidades para expandir este trabalho no futuro. Uma proposta é migrar a aplicação para um ambiente de cloud, o que permitiria melhorar a escalabilidade e superar as limitações computacionais enfrentadas em um ambiente local. Também seria interessante explorar a integração de frameworks adicionais de AutoML, especialmente aqueles que ofereçam maior flexibilidade na customização das pipelines de ML, permitindo que o utilizador tenha maior controle sobre o processo de desenvolvimento dos modelos. Outra oportunidade identificada é o desenvolvimento de uma pipeline de engenharia de *features* para ser usada em oposição às estratégias dos módulos internos do Auto-sklearn, de modo a dar ao utilizador mais capacidades de personalização. Para além disso, é também importante desenvolver o módulo

de explicação dos modelos, para suportar o uso de alguma framework que ajuda na explicação do modelo, de modo a garantir que as decisões tomadas possam ser facilmente explicadas e auditadas, especialmente em contextos sensíveis. Por fim, propõe-se a continuação de estudos sobre a ética na automação de ML, abordando questões como viés algorítmico e responsabilidade nas decisões automatizadas, à medida que a regulamentação em torno da inteligência artificial avança.

Em suma, este trabalho demonstrou com sucesso os benefícios do uso de AutoML para a simplificação do desenvolvimento de pipelines de ML, alcançando os objetivos propostos e evidenciando como a automatização pode tornar esse processo mais acessível. As limitações encontradas abrem caminho para futuras investigações e melhorias, visando expandir ainda mais a aplicabilidade e o impacto das soluções automatizadas no campo da ciência de dados.

Referências

[1] - Schröer, C., Kruse, F., & Gómez, J. M. (2021). A Systematic Literature Review on Applying CRISP-DM Process Model. *Procedia Computer Science*, *181*, 526–534. https://doi.org/https://doi.org/10.1016/j.procs.2021.01.199

[2] - Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E., Chou, R., Glanville, J., Grimshaw, J. M., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E. W., Mayo-Wilson, E., McDonald, S., … Moher, D. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *International Journal of Surgery*, *88*, 105906. https://doi.org/https://doi.org/10.1016/j.ijsu.2021.105906

[3] - Q. Yao et al., 'Taking Human out of Learning Applications: A Survey on Automated Machine Learning', 2018, [Online]. Available: https://arxiv.org/abs/1810.13306v4.

[4] – ZÅ Noller MA, Huber MF (2021) Benchmark and survey of automated machine learning frameworks. Journal of artificial intelligence research. pp 409-72. https://doi.org/10.1613/jair.1.11854

[5] - Escalante, H. J., Montes, M., & Luis, V. (2009). Particle Swarm Model Selection for Authorship Verificatio. Iberoamerican Congress on Pattern Recognition, pp. 563–570.

[6] - Guyon, I., Saffari, A., Dror, G., & Cawley, G. (2008). Analysis of the IJCNN 2007 Agnostic Learning vs. Prior Knowledge Challenge. Neural Networks, 21 (2-3), 544–550.

[7] - Dinsmore, T. (2016). Automated Machine Learning: A Short History.. Available at https: //blog.datarobot.com/automated-machine-learning-short-history.

[8] - Momma, M., & Bennett, K. P. (2002). A Pattern Search Method for Model Selection of Support Vector Regression. In SIAM International Conference on Data Mining, pp. 261–274.

[9] - Samanta, B. (2004). Gear fault detection using artificial neural networks and support vector machines with genetic algorithms. Mechanical Systems and Signal Processing, 18 (3), 625–644

[10] - Y. Roh, G. Heo and S. E. Whang, "A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective," in IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 4, pp. 1328-1347, 1 April 2021, doi: 10.1109/TKDE.2019.2946162.

[11] - Zhou, L. (2018). How to Build a Better Machine Learning Pipeline.. Available at https://www.datanami.com/2018/09/05/how-to-build-a-better-machine-learning-pipeline/

[12] - McGushion, H. (2019). HyperparameterHunter.. Available at https://github.com/HunterMcGushion/hyperparameter_hunter.

[13] - Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In ACM International Conference on Knowledge Discovery and Data Mining, pp. 847–855.

[14] - Andrei Paleyes, Raoul-Gabriel Urma, and Neil D. Lawrence. 2022. Challenges in Deploying Machine Learning: A Survey of Case Studies. ACM Comput. Surv. 55, 6, Article 114 (June 2023), 29 pages. https://doi.org/10.1145/3533378

[15] - KARMAKER, S. K., HASSAN, M. D. M., SMITH, M. J., U, L. E. I. X., ZHAI, C., & VEERAMACHANENI, K. (2022). AutoML to Date and Beyond: Challenges and Opportunities. ACM Computing Surveys, 54(8), 1-1–36. https://doi.org/10.1145/3470918

[16] - Behzadian, M., Khanmohammadi Otaghsara, S., Yazdani, M., & Ignatius, J. (2012). A state-of the-art survey of TOPSIS applications. *Expert Systems with Applications*, *39*(17), 13051–13069. https://doi.org/https://doi.org/10.1016/j.eswa.2012.05.056