Universidade Estadual de Campinas
Instituto de Computação

Thales Eduardo Nazatto

Construção de um *Pipeline* de *Machine Learning* autônomo com métricas de *Fairness*

CAMPINAS
2022

Thales Eduardo Nazatto

Construção de um *Pipeline* de *Machine Learning* autônomo com métricas de *Fairness*

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Orientadora: Profa. Dra. Cecília Mary Fischer Rubira
Coorientador: Prof. Dr. Leonardo Montecchi

Este exemplar corresponde à versão da Dissertação entregue à banca antes da defesa.

CAMPINAS
2022

Na versão final, esta página será substituída pela ficha catalográfica.

De acordo com o padrão da CCPG: "Quando se tratar de Teses e Dissertações financiadas por agências de fomento, os beneficiados deverão fazer referência ao apoio recebido e inserir esta informação na ficha catalográfica, além do nome da agência, o número do processo pelo qual recebeu o auxílio."
e
"caso a tese de doutorado seja feita em Cotutela, será necessário informar na ficha catalográfica o fato, a Universidade convenente, o país e o nome do orientador."

Na versão final, esta página será substituída por outra informando a composição da banca e que a ata de defesa está arquivada pela Unicamp.

Alguma frase
Para colocar,
Mas é um dos últimos itens a se fazer.

(Eu mesmo)

Agradecimentos

Agradecer posteriormente

Resumo

Esta dissertação de Mestrado possui o objetivo de contribuir com o tema de Inteligência Artificial (IA) do ponto de vista da Engenharia de Software, visto que o uso de IA envolvendo grandes volumes de dados vem crescendo conforme nossa sociedade migra processos manuais de trabalho para soluções digitais e necessita de tomadas de decisão mais rápidas e assertivas. Entretanto, as métricas usadas inicialmente para definir a eficácia de um algoritmo se mostraram limitadas devido a barreiras éticas e legais, resultando em vieses que refletem a sociedade de maneira que não era esperada pelos desenvolvedores da solução. Desta forma, o uso de novas métricas, como métricas de Fairness, e proveniência de dados pode ajudar neste problema. Desenvolvendo um pipeline autônomo utilizando a arquitetura MAPE-K, documentando métodos e ferramentas, e realizando estudos de caso, é esperado que este problema seja controlado e facilite o desenvolvimento de softwares com o uso responsável de dados.

Abstract

This Master's Degree dissertation aims to contribute to the theme of Artificial Intelligence (AI) from the Software Engineering point of view, since the use of AI involving large volumes of data has been growing as our society migrates manual work processes to digital solutions and needs faster and more assertive decision making. However, the metrics used initially to define the effectiveness of an algorithm proved to be limited due to ethical and legal barriers, resulting in biases that reflect society in a way that was not expected by the solution developers. Thus, the use of new metrics, like Fairness metrics, and data provenance can help with this problem. By developing an autonomous pipeline using MAPE-K architecture, documenting methods and tools, and making case studies, it is expected that this problem will be controlled and facilitate software development with the responsible use of data.

Lista de Figuras

Sumário

Capítulo 1

Introdução

Técnicas de Inteligência Artificial e Aprendizado de Máquina já são utilizadas há bastante tempo no ramo da Computação. Ramos como robótica e jogos são grandes exemplos, dada a necessidade nos mesmos de automatizar comportamentos que seriam tidos como triviais para um ser humano. Entretanto, nos últimos anos ocorreu um crescimento no uso dessas tecnologias em aplicações tradicionais, com previsão de US$ 57 bilhões em investimentos em 2021, 480% maior em relação a 2017 [6]. No Brasil, o número de empresas de IA aumentou de 120 em 2018 para 206 em 2020 [7].

Isso se mostra possível devido a grande quantidade de dados processada diariamente pelas empresas, que coletam estatísticas toda vez que um usuário acessa suas aplicações. Com esses dados, podem traçar diferentes perfis e usar soluções de IA para ter tomadas de decisão mais assertivas com o objetivo de melhorar a experiência de usuário e corrigir problemas. Porém, muitas dessas soluções foram projetadas sem pensar em governança de dados como requisito de projeto, e se mostram ineficientes quando ela é tratada em consideração.

Governança de dados é um tema que entrou em evidência recentemente em países como o Brasil: Iniciativas como a LGPD - Lei Geral de Proteção de Dados [8], de 14 de agosto de 2018, mostram como as aplicações e seus dados possuem cada vez mais influência na sociedade moderna. E nesse ponto muitas aplicações de IA falham: muitas implementações foram implementadas como *black boxes*, onde o determinante para estabelecer a confiança no modelo implementado é sua entrada e sua saída.

Um outro efeito colateral dessa estratégia é a exposição de vieses que, embora sejam vistos como não-intencionais pelos desenvolvedores por ter a possibilidade de ser um *outlier* no modelo treinado, refletem preconceitos escancarados da sociedade atual. Uma entrada de dados enviesada resulta em um algoritmo que realiza discriminações em sua classificação [16], e uma vez que as métricas utilizadas para medir a qualidade de um modelo *black box* são geralmente baseadas em acurácia, precisão e recall, discriminações não são facilmente percebidas por tais métricas. Ao mesmo tempo, um modelo *black box* pode ter uma alta dependência de poucos dados, determinando problemas de acoplamento. Para resolver este problema, é possível que a criação de um novo modelo seja uma melhor opção que realizar a correção em apenas parte dele.

Devido a esses tipos de problemas, o termo *Explainable AI* (XAI) ganha força para envolver o desenvolvimento de uma IA que seja acurada e simultaneamente transparente.

Como IA possui diversos tipos de métodos diferentes para enquadrar diversos tipos de dados, o mesmo acaba se aplicando em Explainable AI, podendo enquadrar em diversos tipos de dados [38], ou dados específicos como imagens [26] e tabelas [32]. Como o objetivo em XAI é fazer com que os resultados alcançados pela solução de IA sejam compreendidos por humanos, é possível considerar este fato como requisito no design de uma solução de IA, fazendo com que a mesma seja reusável e testável.

No mesmo tema, é possível estabelecer métricas para determinar o quão o modelo está preparado para dados sensíveis [13], termo que é conhecido como *Fairness*. Com a evolução das pesquisas na comunidade acadêmica, foram descobertos algoritmos para redução dos vieses presentes nos conjuntos de dados, como *Reweighing* [24], *Adversarial Debiasing* [40] e *Reject Option Classification* [25]. Por consequência, ocorre melhora nas métricas em questão, mas pode desfavorecer métricas que já são amplamente utilizadas como garantia de um bom modelo desenvolvido com técnicas de Aprendizado de Máquina.

O objetivo desta dissertação de mestrado é desenvolver uma estrutura de *Pipeline* para *Machine Learning* que seja completamente autônoma, por três fatores principais:

- Facilitar a criação de modelos justos e confiáveis com a automatização da escolha dos algoritmos, cuja complexidade aumenta com a escolha dos algoritmos a serem utilizados e suas execuções nas etapas corretas do processo, onde eles foram escolhidos para atuar.
- Estabelecer um balanceamento entre métricas para avaliar bons modelos com métricas para avaliar modelos justos.
- Considerar proveniência de dados como requisito no design de uma solução de IA, e como uma alternativa a XAI através da utilização de metadados.

Para isso, além do desenvolvimento do *Pipeline* em questão, será utilizada a arquitetura MAPE-K [5] para analisar uma base de conhecimento e prover o melhor pipeline seguindo regras pré-determinadas.

O restante da dissertação se organizará da seguinte forma: o Capítulo 2 descreve os conceitos que irão ser abordados neste projeto; o Capítulo 3 mostra a metodologia e detalhamento do processo de desenvolvimento; o Capítulo 4 discute os resultados obtidos e, finalmente, o Capítulo 5 estabelece as conclusões, considerações finais e sugestões de trabalhos futuros e evoluções.

Capítulo 2

Conceitos Relacionados

2.1 Ciência de Dados e Engenharia de Dados

2.2 Machine Learning

Aprendizado de Máquina (*Machine Learning*, em inglês) pode ser definido como "a prática de usar algoritmos para coletar dados, aprender com eles, e então fazer uma determinação ou predição sobre alguma coisa no mundo. Então em vez de implementar as rotinas de software manualmente, com um gama específica de instruções para completar uma tarefa em particular, a máquina é 'treinada' usando uma quantidade grande de dados e algoritmos que dão e ela a habilidade de aprender como executar a tarefa" [1]. Com isso, o computador consegue a habilidade de realizar determinado cálculo ou tarefa sem que necessite de programação adicional ou interferência humana para isso.

O *Machine Learning* é fortemente relacionado com a Estatística, uma vez que seus métodos e parte de seus algoritmos, como regressões, tiveram como base modelos estatísticos e a análise de seus dados. As tarefas de aprendizado podem ser classificadas em três categorias básicas [4] [3]:

- **Aprendizado supervisionado**: O treinamento é realizado por meio de exemplos rotulados, como uma entrada na qual a saída desejada é conhecida. Através de métodos como classificação, regressão e *gradient boosting*, o aprendizado supervisionado utiliza padrões para prever os valores de rótulos em dados não-rotulados adicionais. O aprendizado supervisionado é comumente empregado em aplicações nas quais dados históricos preveem eventos futuros prováveis.

- **Aprendizado não-supervisionado**: É utilizado em dados que não possuem rótulos históricos. A "resposta certa" não é informada ao sistema, o algoritmo deve descobrir o que está sendo mostrado. O objetivo é explorar os dados e encontrar alguma estrutura dentro deles. Técnicas populares incluem mapas auto-organizáveis, mapeamento por proximidade, agrupamento *k-means* e decomposição em valores singulares. Esses algoritmos também são utilizados para segmentar tópicos de texto, recomendar itens e identificar pontos discrepantes nos dados.

- **Aprendizado por reforço**: O algoritmo descobre através de testes do tipo "tentativa e erro" quais ações rendem as maiores recompensas. Este tipo de aprendizado possui três componentes principais: o agente (o aprendiz ou tomador de decisão), o ambiente (tudo com que o agente interage) e ações (o que o agente pode fazer). O objetivo é que o agente escolha ações que maximizem a recompensa esperada em um período de tempo determinado. O agente atingirá o objetivo muito mais rápido se seguir uma boa política, então o foco do aprendizado por reforço é descobrir a melhor política.

O termo *Machine Learning* se tornou muito mais evidente com a possibilidade da implementação do *Deep Learning*, que é uma técnica que utiliza Redes Neurais Artificiais para atingir seus resultados. Redes Neurais Artificiais são modelos computacionais inspirados no sistema nervoso do cérebro, onde temos neurônios divididos em camadas e conectados entre si, podendo ser abstraído conforme ilustração na Figura 2.1. Dependendo da tarefa a ser realizada, cada neurônio atribui um peso para os dados que entram e a saída final é determinada pelo total desses pesos [1]. As redes neurais utilizadas em *Deep Learning* possuem, ao menos, duas camadas de neurônios entre a camada que recebe os dados de entrada e a camada final que faz o tratamento final dos dados de saída.

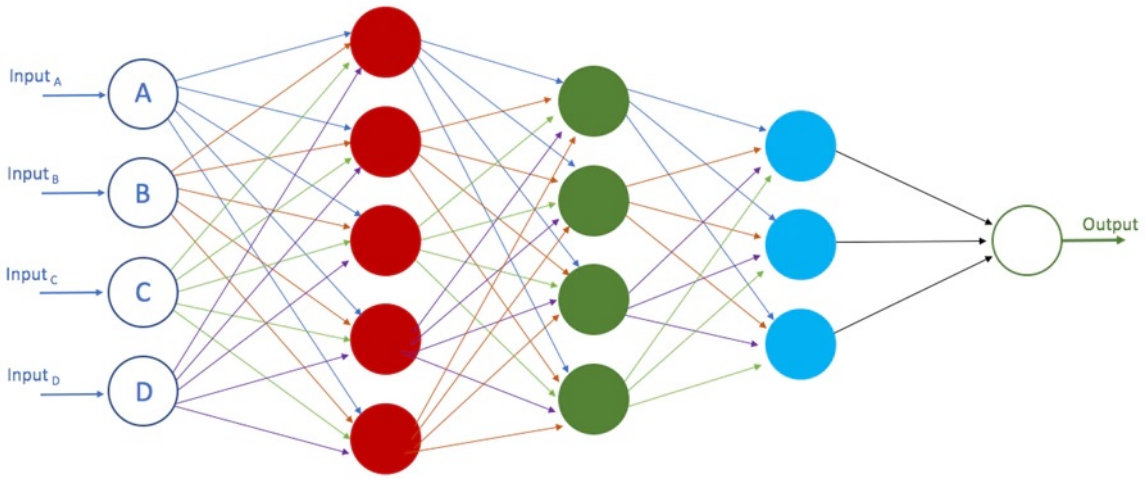

Figura 2.1: Exemplo de uma rede neural utilizada em *Deep Learning*.

Com a evolução da computação, o treino de uma tarefa passou a ser cada vez mais viável, uma vez que a execução de algoritmos de *Machine Learning* é computacionalmente muito custosa, especialmente quando redes neurais são utilizadas. E sua viabilidade é acompanhada de efetividade: Como exemplo, reconhecimento de imagens por máquinas treinadas através de deep learning em alguns cenários possuem uma taxa de acerto maior que a de humanos [1].

2.2.1 Métricas de Avaliação

Tradicionalmente em um problema de classificação binária, uma previsão do classificador pode ter 4 tipos de resultados em uma matriz de confusão, presente na Tabela 2.2.1. Os

Verdadeiros Positivos (VP, ou TP pelo termo em inglês *True Positives*) e Verdadeiros Negativos (VN, ou TN pelo termo em inglês *True Negatives*) são classificações corretamente previstas pelo classificador. Um Falso Positivo (FP) ocorre quando um caso previsto para estar na classe positiva quando o resultado real pertence à classe negativa, e um Falso Negativo (FN) ocorre quando um caso previsto para estar na classe negativa quando o resultado real pertence à classe positiva.

– TODO reformatar Tabela

Tabela 2.1: Matriz de confusão

		Classe prevista	
		Positiva	**Negativa**
Classe atual	**Positiva**	Verdadeiros Positivos (VP/TP)	Falsos Positivos (FP)
	Negativa	Falso Negativo (FN)	Verdadeiros Negativos (VN/TN)

A taxa de sucesso, também conhecida como **acurácia**, é o número de previsões corretas dividido pelo número total de resultados:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

Ela é considerada um indicador da realização de um bom treinamento e de um bom funcionamento do modelo obtido. Entretanto, não é a melhor métrica para interpretar situações quanto a aplicação do modelo ao problema, como classes desequilibradas.

A **precisão** ajuda quando os custos de falsos positivos são altos e mostra a precisão das previsões positivas. É a fração de casos positivos previstos corretamente para estar na classe positiva de todos os casos positivos previstos no modelo:

$$P = \frac{TP}{TP + FP} \tag{2.2}$$

O **recall**, ou sensibilidade, ajuda quando o custo de falsos negativos é alto. É a fração de casos positivos previstos corretamente para estar na classe positiva de todos os casos positivos reais:

$$R = \frac{TP}{TP + FN} \tag{2.3}$$

O **F1-score** é uma medida geral da acurácia de um modelo que combina precisão e recall. Pode ser interpretado como uma ponderação entre ambos e usa a média harmônica para realizar a medição:

$$F1 = 2 \times \frac{P \times R}{P + R} \tag{2.4}$$

O **AUC** (*Area under the ROC Curve*), também chamada de precisão balanceada, pode imterpretar a capacidade do classificador de evitar uma classificação falsa. Se sai melhor que a acurácia em situações que ela não é apropriada, como o desequilíbrio entre classes já citado anteriormente:

$$AUC = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right) \tag{2.5}$$

2.2.2 Algoritmos

2.3 Fairness em Machine Learning

Como a coleta de dados está presente atualmente no dia-a-dia de variados setores da sociedade, o uso de *Machine Learning* é extremamente versátil para tomadas de decisão, podendo ser utilizados em problemas como admissão de universidades, contratações, análise de crédito e reconhecimento de doenças. Com o aumento dessa influência, o uso de dados sensíveis em um contexto determinado também aumentou, e temas como uma IA ética e conceitos como vieses nos dados e *Fairness* passaram a serem discutidas não apenas na Computação, mas em áreas como Direito. Algoritmos são mais objetivos, rápidos e são capazes de considerar uma grande magnitude de recursos que pessoas não são capazes. Entretanto, até o presente momento eles não são capazes de diferenciar contextos sociais, onde um resultado mais eficiente de acordo com os dados disponíveis podem amplificar as desigualdades sociais e tomar decisões de modo injusto [33].

Estes dados sensíveis, tendo como exemplos cor de pele, raça, sexo, idade e altura, são considerados atributos protegidos, que precisam ser classificados e processados antes da execução de um algoritmo de *Machine Learning*, determinarão como o algoritmo se comportará e, consequentemente, afetará suas métricas [34]. Os grupos de dados provenientes destes atributos protegidos são considerados grupos protegidos, que podem ser divididos em dois grupos: o grupo privilegiado, que possui vantagens no contexto do problema, e o grupo não-privilegiado, que possui desvantagens no contexto do problema e, portanto, sujeito a discriminação.

É possível descrever o conceito de *Fairness* no contexto de aprendizagem supervisionada, onde um modelo f pode prever um conjunto de resultados y a partir de um conjunto de *features* x, evitando discriminação injusta em relação a um atributo protegido a. É permitido, mas não exigido, que a seja um componente de x [13]. Em outras palavras, um modelo de ML considerado justo é aquele onde a correlação de seu resultado é baixa em relação a dados de entrada considerados como sensíveis a discriminações.

Geralmente, as descrições de justiça se dividem em dois grupos principais: justiça individual e justiça de grupo. O objetivo da justiça individual é que indivíduos semelhantes

devem obter resultados semelhantes, enquanto na justiça de grupo, cada um dos grupos definidos pelo atributo protegido deve ser tratado igualmente. ser tratados igualmente.

– TODO continuar

2.3.1 Métricas de Fairness

Considerando $Y = 1$ a classe positiva, $Z = 0$ o grupo não-privilegiado e $Z = 1$ o grupo privilegiado, algumas das definições de *Fairness* mais usadas são as seguintes:

- **Diferença de paridade estatística, ou discriminação [39]:** Esta métrica é baseada na seguinte fórmula:

$$Pr(Y = 1|Z = 0) - Pr(Y = 1|Z = 1) \tag{2.6}$$

  Aqui, o viés ou paridade estatística é a diferença entre a probabilidade de que um indivíduo aleatório retirado dos não-privilegiados seja rotulado como 1 e a probabilidade de que um indivíduo aleatório dos privilegiados seja rotulado como 1. Portanto, um valor próximo de 0 é considerado justo.

- **Diferença de oportunidade igual [14]:** É a diferença entre a taxa positiva verdadeira do grupo não privilegiado e a taxa positiva verdadeira do grupo privilegiado:

$$TPR_{Z=0} - TPR_{Z=1} \tag{2.7}$$

  Um valor próximo de 0 é considerado justo. Um classificador binário satisfaz a igualdade de oportunidades quando a taxa positiva verdadeira de ambos os grupos são iguais [23]

- **Diferença de probabilidade média [14]:** Essa métrica usa a taxa de falsos positivos e a taxa positiva verdadeira para calcular a tendência, calculando a igualdade de probabilidades com a fórmula:

$$\frac{1}{2}(|FPR_{Z=0} - FPR_{Z=1}| + |TPR_{Z=0} - TPR_{Z=1}|) \tag{2.8}$$

  Precisa ser próximo a 0 para ser considerado justo.

- **Impacto de disparidade [14]:** Para esta métrica, é usada a seguinte fórmula:

$$\frac{Pr(Y = 1|Z = 0)}{Pr(Y = 1|Z = 1)}$$

  Usa as mesmas probabilidades da diferença de paridade estatística, mas aqui são calculadas como proporção. Desta forma, um valor próximo de 1 é considerado justo.

- **Indíce de Theil [37]:** Esta medida também é conhecida como índice de entropia generalizado, mas com α igual a 1 [37]. É calculado com a seguinte fórmula:

$$\frac{1}{n}\sum_{i=0}^{n}\frac{b_i}{\mu}\ln\frac{b_i}{\mu}$$

  Onde $b_i = \hat{y}_i - y_i + 1$, y_i é o conjunto de saídas e $\hat{y}_i$ é o conjunto de previsões dadas pelo modelo. Também precisa ser próximo a 0 para ser considerado justo.

2.3.2 Algoritmos para redução de vieses

Há diversos tipos de algoritmos diferentes na inteligência artificial para redução de vieses, a fim de garantir *Fairness* nos projetos de Aprendizado de Máquina. É possível classificá-los em três categorias diferentes: Algoritmos de pré-processamento, processamento e de pós-processamento.

Os algoritmos de **pré-processamento** tentam eliminar a discriminação transformando os dados, antes de executar o algoritmo de treinamento. Tais algoritmos podem ser usados caso seja permitida a modificação dos dados de treinamento [19], e a ideia por trás de tais algoritmos é que suas previsões serão mais balanceadas se o classificador for treinado com os dados já balanceados [**?**]. Nesta categoria se enquadram os seguintes algoritmos:

- **Reposição (*Reweighing*) [24]:** Pondera os exemplos em cada combinação de grupo e rótulo de maneira diferente para garantir a justiça antes da classificação.
- **Removedor de impacto de disparidade (*Disparate impact remover*) [20]:** Edita valores de *features* aumentando a justiça de cada grupo enquanto preserva a ordem de classificação dentro dos mesmos.
- **Aprendizado de representações justas (LFR, ou *Learning fair representations*) [39]:** Encontra uma representação latente que codifica os dados, mas ofusca informações dos atributos protegidos.
- **Pré-processamento otimizado (*Optimized pre-processing*) [17]:** Aprende uma transformação probabilística que edita *features* e rótulos nos dados efetuando justiça em cada grupo, distorção individual, garantindo a fidelidade de dados através de restrições.

Os algoritmos de **processamento** tentam realizar modificações nos algoritmos de treinamento para mitigar a discriminação durante o processo de treinamento do modelo. Se for permitido fazer mudanças no processo de treinamento, então os algoritmos podem ser usados incorporando mudanças na função de custo ou impondo restrições [**?**]. Nesta categoria se enquadram os seguintes algoritmos:

- **Remoção de viés adversário (Adversarial debiasing) [40]:** Aprende um classificador que maximiza a precisão e reduz a capacidade de um adversário de depender

do atributo protegido nas previsões. Essa abordagem leva a um classificador justo, pois as previsões realizadas pelo classificador não possuem nenhuma informação de discriminação nos grupos.

- **Removedor de preconceito (Prejudice remover) [20]:** Técnica que adiciona no algoritmo escolhido um termo de regularização baseado na discriminação (No caso da biblioteca AI Fairness 360 é usado o programa da publicação, que usa a regressão logística como algoritmo base).
- **Meta-Algoritmo para classificações justas (Meta-Algorithm for Fair Classification) [18]:** Aprende um classificador compatível com uma gama grande de métricas de Fairness, sendo prático o suficiente para abrangê-las sem grande perda de performance.
- **Justiça por Subgrupos Ricos (Rich Subgroup Fairness) [27]:** Aprende um classificador que procura equalizar as taxas de falsos positivos e falsos negativos entre os dados que envolvem atributos protegidos, considerados como subgrupos.
- **Redução por Gradiente Exponencial (Exponentiated Gradient Reduction) [10]:** Aprende um classificador baseado em Gradiente Exponencial que tende a minimizar o erro de uma classificação ponderada.
- **Redução por busca em grid (Grid Search Reduction) [10] [11]:** Aprende um classificador baseado na busca em um grid de valores que tende a minimizar o erro de uma classificação ponderada. É mais simples e impreciso que a Redução por Gradiente Exponencial, mas sua escolha pode ser razoável se a quantidade de métricas de Fairness a serem consideradas for pequena.

Os algoritmos de **pós-processamento** utilizam um conjunto de validação, que não foi envolvido no processo de treinamento para melhorar a imparcialidade das previsões [19]. Quando não há possibilidade de fazer alterações nos dados de treinamento ou no treinamento do modelo, apenas algoritmos de pós-processamento podem ser usados. Nesta categoria se enquadram os seguintes algoritmos:

- **Igualdade de probabilidade calibrada (Calibrated Equalized odds) [36]:** Otimiza as previsões do classificador obtido, calibrando para alterar os rótulos de saída e obter probabilidades igualadas entre os grupos.
- **Igualdade de probabilidade (Equalized odds) [23]:** Resolve um problema linear para alterar os rótulos de saída e obter probabilidades igualadas entre os grupos.
- **Classificação baseada em Rejeição de Opções (Reject Option-based Classification) [25]:** Dá resultados favoráveis para grupos não privilegiados e resultados desfavoráveis para grupos privilegiados de acordo com uma faixa de confiança.

2.4 Engenharia de Software

2.4.1 Engenharia de Software para Aplicações de IA

Quando se fala de Arquitetura e Engenharia de Software, se fala da definição dos componentes de software, suas propriedades externas, e seus relacionamentos com outros softwares para fazer com que um sistema seja documentável, reusável e testável. A preocupação está em como um sistema deve ser organizado e com a estrutura geral desse sistema. Dado as definições sobre IA já detalhadas, é possível encaixar Machine Learning na forma de um processo bem definido, de forma que é possível sistematizar todo esse processo na forma de componentes e definir formas em que o modelo resultante do mesmo é disponibilizado para aplicações externas.

No processo, ilustrado na Figura 2.2, o conjunto de dados passa por um pré-processamento e dividido em dois conjuntos, um para treinamento e outro para teste. O conjunto de treino é utilizado para o algoritmo realizar o processo de treinamento, obtendo um modelo após o término desse processo. O conjunto de testes é utilizado para mensurar se o modelo obtido no processo de treinamento realiza previsões confiáveis ou não.

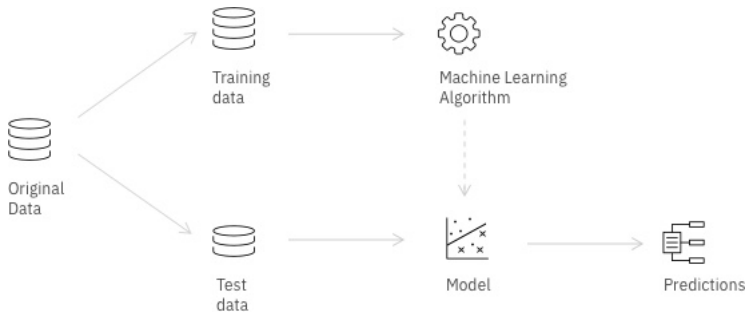

Figura 2.2: Processo padrão para aprendizado de máquina

Um exemplo de arquitetura que generaliza todo o processo, desde a necessidade de negócio até o deploy do modelo de IA, é a IBM Analytics and AI Reference Architecture [2], ilustrada na Figura 2.3. Nela, são definidos os seguintes requisitos não-funcionais: Performance, estabilidade, segurança, escalabilidade, manutenibilidade e regulamentações de privacidade/*compliance*, e pode ser classificada em 4 grupos principais envolvendo diversos tipos de processos e componentes:

- **Coleta:** Relaciona os processos de coleta, armazenamento e transformação de diversas fontes de dados, estruturadas ou não estruturadas, para determinados repositórios (*Data Lakes*).
- **Organização:** Relaciona os processos de organização e estruturação dos dados nos presentes nos *Data Lakes* e necessários para o uso das aplicações que envolvem a análise dos dados. Dependendo do uso pode-se aplicar processos de Governança.
- **Análise:** Relaciona os processos para desenvolvimento de aplicações de IA e relatórios após a organização dos dados para tomadas de decisão e uso de aplicações externas.

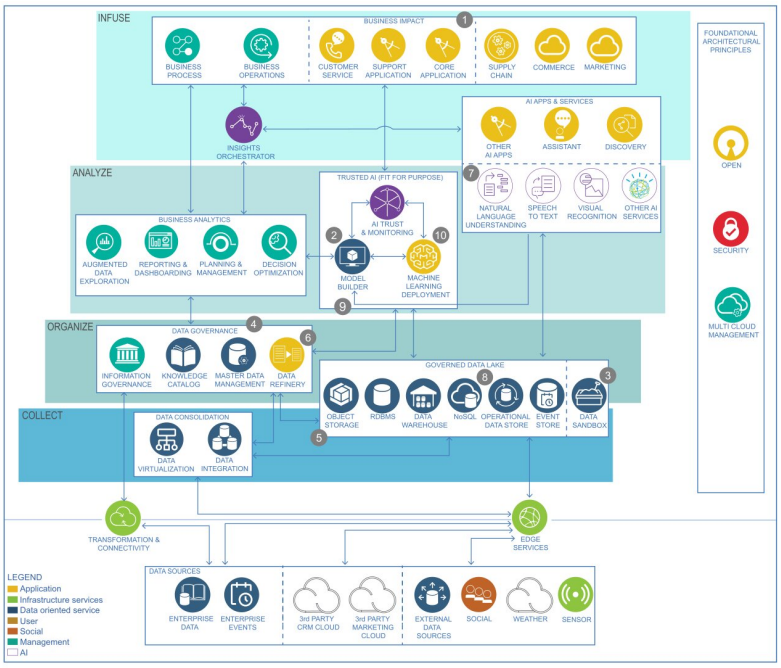

Figura 2.3: IBM Analytics and AI Reference Architecture.

- **Infusão:** Relaciona os processos de disponibilização desses dados e conhecimento obtidos na fase de análise para aplicações externas.

2.5 Proveniência de Dados

2.6 Arquitetura de Software

O consenso sobre a definição de arquitetura de software foi adotado com a adoção do padrão IEEE 1471, que define arquitetura de software como *"a organização fundamental de um sistema incorporado em seus componentes, seus relacionamentos entre si e com o meio ambiente e os princípios que orientam seu projeto e evolução"*. Com esta definição, o componente e o conector são reforçados como conceitos centrais da arquitetura de software [15].

O nível de design da arquitetura de software em um projeto vai além dos algoritmos e estruturas de dados da computação. Incluem fatores como organização, protocolos de comunicação, acesso a dados, atribuição de funcionalidades, escalabilidade, performance, composição e seleção do *design* ideal [22]. É possível tratar uma arquitetura de um sistema específico como uma coleção de componentes juntamente com uma descrição dos conectores, que define as interações entre os componentes.

Um estilo de arquitetura define uma família de tais sistemas em termos de um padrão de organização estrutural, e determina o vocabulário de componentes e conectores que podem ser usados em instâncias desse estilo, juntamente com um conjunto de restrições sobre como eles podem ser combinados [22]. A decisão sobre tal estilo depende da solução e dos requisitos de um sistema. Ela pode adicionar novos componentes, incrementá-los

com novos requisitos ou adicionar restrições sobre eles [15].

2.6.1 Arquitetura MAPE-K

Em 2001, Paul Horn introduziu o conceito de Computação Autônoma como alternativa a solução para a crescente complexidade dos sistemas da época, onde previa-se que os mesmos se tornariam muito grandes e complexos até mesmo para os profissionais mais qualificados configurarem e realizarem manutenção. Tal conceito qualifica sistemas de computação que podem se autogerenciar com relação aos objetivos de alto nível dados pelos administradores e é derivado da biologia, dado a grande variedade e hierarquia de sistemas autônomos presentes na natureza e na sociedade [28].

Em um ambiente autônomo e autogerenciado, os componentes de sistema podem incorporar como funcionalidade um *loop* de controle. Embora estes *loops* sejam divididos nos mesmos procedimentos, é possível categorizá-los em 4 categorias principais. Essas categorias são consideradas atributos dos componentes do sistema e são definidas como [5]:

- **Auto-configuração:** Pode se adaptar dinamicamente a mudanças no ambiente. Um componente autoconfigurável realiza esta adaptação usando políticas fornecidas pelo profissional. Tais mudanças podem incluir a implantação de novos componentes ou a remoção dos existentes, ou mudanças drásticas nas características do sistema. A adaptação dinâmica ajuda a garantir força e produtividade contínuas da infraestrutura, resultando em crescimento e flexibilidade dos negócios.

- **Auto-cura:** Pode descobrir, diagnosticar e reagir a interrupções. Um componente auto-curável pode detectar falhas no sistema e iniciar ações corretivas baseadas em políticas sem interromper o ambiente. A ação corretiva pode envolver um produto alterando seu próprio estado ou efetuando mudanças em outros componentes do ambiente. Com isso, o sistema se torna mais resiliente porque as operações cotidianas possuem menos probabilidade de falhar.

- **Auto-otimização:** Pode monitorar e ajustar recursos automaticamente. Um componente auto-otimizável pode se ajustar para atender às necessidades do usuário. As ações de ajuste podem significar realocar recursos para melhorar a utilização geral, como em resposta a cargas de trabalho que mudam dinamicamente, ou garantir que processamentos possam ser concluídos em tempo hábil. A auto-otimização ajuda a fornecer um alto padrão de serviço para quem vai utilizar o sistema. Sem funções de auto-otimização, não há uma maneira fácil de re-escalonar os recursos de infraestrutura quando um aplicativo não os usa totalmente.

- **Auto-proteção:** Pode antecipar, detectar, identificar e proteger contra ameaças de qualquer lugar. Um componente de autoproteção pode detectar comportamentos hostis à medida que ocorrem e tomar ações corretivas para se tornarem menos vulneráveis. Os comportamentos hostis podem incluir acesso e uso não autorizados, infecção e proliferação de vírus e ataques de negação de serviço. Os recursos de autoproteção permitem que as empresas apliquem consistentemente políticas de segurança e privacidade.

Para a Computação Autônoma acontecer, é implementado um Elemento Autônomo [9], um componente de software que gerencia partes do sistema baseando-se em um *loop* MAPE-K (*Monitor, Analyze, Plan, Execute, and Knowledge*), ilustrado na Figura 2.4. O MAPE-K é um conceito que constitui um *loop* de controle, usado para monitorar e controlar um ou mais elementos gerenciados. Um elemento gerenciado (*Managed Element*) pode ser um hardware, como uma impressora, um software, como um banco de dados, outro Elemento Autônomo ou funcões específicas, como balanceamento de carga. Um *loop* de controle MAPE-K é dividido da seguinte forma:

- **Monitoramento (*Monitor*):** Esta parte é responsável por monitorar os recursos gerenciados e coletar, agregar e filtrar dados. O monitoramento é feito por meio de um sensor (*Sensor*) ou mais sensores.
- **Análise (*Analyze*):** Analisa os dados relatados pela parte do monitor. A análise visa compreender qual é o estado atual do sistema e se há medidas para serem tomadas.
- **Planejamento (*Plan*):** Um plano de ação é preparado no base dos resultados da análise. O plano é uma série de medidas que irão mover o sistema de seu estado atual para um estado desejado.
- **Execução (*Execute*):** O plano é executado e controlado. Um efetor (*Effector*) ou mais executam as ações planejadas no recurso.
- **Conhecimento (*Knowledge*):** A base de conhecimento é central e acessível por todas as partes do *loop*. Separado a partir de dados coletados e analisados, ele contém conhecimento adicional, como modelos de arquitetura, modelos de metas, políticas e planos de mudança.

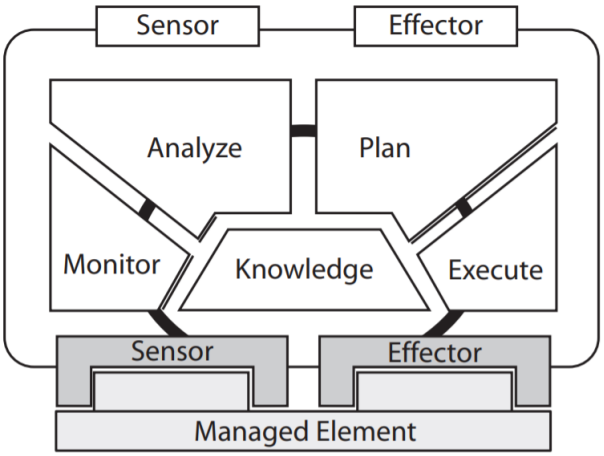

Figura 2.4: Diagrama de funcionamento da arquitetura MAPE-K [9].

2.6.2 Arquitetura *Pipe-and-Filter*

Em uma arquitetura *Pipe-and-Filter*, ilustrado na Figura 2.5, cada componente tem um conjunto de entradas e um conjunto de saídas. Um componente lê *streams* (ou fluxos) de dados em suas entradas e produz *streams* de dados em suas saídas, abstraindo a entrega de um resultado como um todo. O *stream* de entrada é transformado de modo que a saída comece a ser produzida antes da entrada ser completamente consumida. Por isso, os componentes são chamados de filtros (*filters*). Os conectores deste estilo servem como condutores para os *streams*, transmitindo as saídas de um filtro para as entradas de outro. Por isso, os conectores são chamados de tubos (*pipes*) [22].

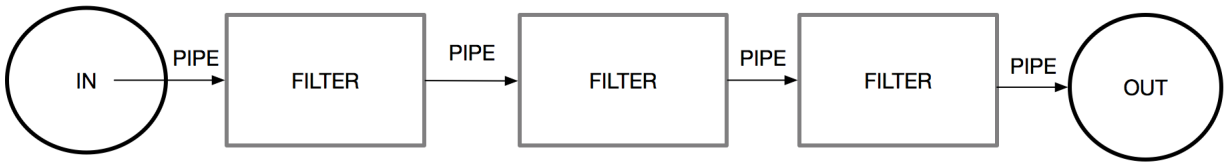

Figura 2.5: Diagrama de uma arquitetura *Pipe-and-Filter*.

Os filtros devem ser independentes, isto é, não devem compartilhar estado com outros filtros e, durante a programação, o ideal é que os filtros independam da ordem de processamento. Suas especificações podem restringir os dados transportados nos *pipes* de entrada e nos *pipes* de saída, mas não conhecem quaisquer outros filtros, ou componentes, conectados por seus *pipes*. Especializações comuns desse estilo incluem *pipelines*, que restringem as topologias a sequências lineares de filtros; *Pipes* limitados, que restringem a quantidade de dados que podem ser passados em um *pipe*, e *pipes* tipados, que exigem que os dados passados entre dois filtros tenham um tipo bem definido.

O uso da arquitetura *Pipe-and-Filter* possuí como vantagens:

- Permitem que o Arquiteto de Software/Desenvolvedor entendam o comportamento geral de entrada/saída de um sistema como uma composição simples dos comportamentos dos filtros individuais.
- Suportam a reutilização: quaisquer dois filtros podem ser conectados, desde que concordem com os dados que estão sendo transmitidos entre eles.
- Os sistemas podem ser facilmente mantidos e aprimorados: novos filtros podem ser adicionados a sistemas existentes e filtros antigos podem ser substituídos por outros melhorados.
- Permitem certos tipos de análise especializada, como análise de rendimento e de impasse.
- Naturalmente suportam a execução simultânea: Cada filtro pode ser implementado como uma tarefa separada e potencialmente executado em paralelo com outros filtros.

Como desvantagens, é possível citar:

- Geralmente levam a uma organização de processamento em lote. Embora os filtros possam processar dados de forma incremental, uma vez que os filtros são inerentemente independentes, o Arquiteto de Software é forçado a pensar em cada filtro como fornecendo uma transformação completa dos dados de entrada em dados de saída.

- Por sua natureza de transformação de dados, os sistemas que usam a arquitetura *Pipe-and-Filter* normalmente não são bons para lidar com aplicativos interativos. Esse problema é mais grave quando são necessárias atualizações de exibição incrementais, porque o padrão de saída para atualizações incrementais é radicalmente diferente do padrão para saída de filtro.

- Podem ser prejudicados por terem que manter correspondências entre dois *streams* separados, mas relacionados.

- Podem forçar um resultado médio na transmissão de dados em situações onde muitos filtros sejam encadeados com um único filtro, resultando em trabalho adicional para cada filtro separar seus dados e analisar o que for necessário. Isso, por sua vez, pode levar tanto à perda de desempenho quanto ao aumento da complexidade na escrita dos próprios filtros.

Capítulo 3

Metodologia

3.1 Detalhamento do processo e diagramas elaborados

3.2 Arquitetura do código

3.3 Experimentos e limitações

Capítulo 4

Resultados e Discussões

Capítulo 5

Conclusões

Referências Bibliográficas

[1] What's the difference between artificial intelligence, machine learning and deep learning? URL https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/.

[2] IBM - Analytics and AI architecture. URL https://www.ibm.com/cloud/architecture/architectures/aiAnalyticsArchitecture/reference-architecture/.

[3] Machine learning: o que é e qual sua importância? URL https://www.sas.com/pt_br/insights/analytics/machine-learning.html.

[4] Aprendizado de máquina. URL https://pt.wikipedia.org/wiki/Aprendizado_de_máquina.

[5] An architectural blueprint for autonomic computing. Technical report, IBM, 2005. URL https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf.

[6] Machine learning: things are getting intense, 2018. URL https://www2.deloitte.com/content/dam/Deloitte/global/Images/infographics/technologymediatelecommunications/gx-deloitte-tmt-2018-intense-machine-learning-report.pdf.

[7] Brasil se destaca com 42% das iniciativas de IA na América Latina, em 2020, 2021. URL https://cio.com.br/tendencias/brasil-se-destaca-com-42-das-iniciativas-de-ia-na-america-latina-em-2020/.

[8] Proteção de Dados - LGPD, 2021. URL https://www.gov.br/defesa/pt-br/acesso-a-informacao/lei-geral-de-protecao-de-dados-pessoais-lgpd.

[9] Nadeem Abbas, Jesper Andersson, and Welf Löwe. Autonomic software product lines. pages 324–331, 2010.

[10] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna Wallach. A reductions approach to fair classification. In *International Conference on Machine Learning*, pages 60–69, 2018.

[11] Alekh Agarwal, Miroslav Dudík, and Zhiwei Steven Wu. Fair regression: Quantitative definitions and reduction-based algorithms. In *International Conference on Machine Learning*, pages 120–129, 2019.

[12] Alejandro Barredo Arrieta, Natalia Díaz Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.

[13] Tom Begley, Tobias Schwedes, Christopher Frye, and Ilya Feige. Explainability for fair machine learning, 2021.

[14] Sumon Biswas and Hridesh Rajan. Do the machine learning models on a crowd sourced platform exhibit bias? an empirical study on model fairness. page 642–653, 2020.

[15] Jan Bosch. Software architecture: The next step. pages 194–199, 2004.

[16] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on Fairness, Accountability and Transparency*, volume 81 of *Proceedings of Machine Learning Research*, pages 77–91, 2018.

[17] Flavio Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Natesan Ramamurthy, and Kush R Varshney. Optimized pre-processing for discrimination prevention. In *Advances in Neural Information Processing Systems*, 2017. URL https://proceedings.neurips.cc/paper/2017/file/9a49a25d845a483fae4be7e341368e36-Paper.pdf.

[18] L Elisa Celis, Lingxiao Huang, Vijay Keswani, and Nisheeth K Vishnoi. Classification with fairness constraints: A meta-algorithm with provable guarantees. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 319–328, 2019.

[19] Brian d'Alessandro, Cathy O'Neil, and Tom LaGatta. Conscientious classification: A data scientist's guide to discrimination-aware classification. *Big Data*, pages 120–134, 2017.

[20] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 259–268, 2015. URL https://doi.org/10.1145/2783258.2783311.

[21] Martin Fowler. Fluentinterface, 2005. URL https://martinfowler.com/bliki/FluentInterface.html.

[22] David Garlan and Mary Shaw. An introduction to software architecture. In *Advances in Software Engineering and Knowledge Engineering*, 1993.

[23] Moritz Hardt, Eric Price, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In *Advances in Neural Information Processing Systems*, 2016. URL https://proceedings.neurips.cc/paper/2016/file/9d2682367c3935defcb1f9e247a97c0d-Paper.pdf.

[24] Faisal Kamiran and Toon Calders. Data pre-processing techniques for classification without discrimination. *Knowledge and Information Systems*, 2011. URL https://www.researchgate.net/publication/228975972_Data_Pre-Processing_Techniques_for_Classification_without_Discrimination.

[25] Faisal Kamiran, Asim Karim, and Xiangliang Zhang. Decision theory for discrimination-aware classification. In *2012 IEEE 12th International Conference on Data Mining*, pages 924–929, 2012.

[26] Andrei Kapishnikov, Tolga Bolukbasi, Fernanda B. Viégas, and Michael Terry. XRAI: better attributions through regions. In *IEEE/CVF International Conference on Computer Vision*, pages 4947–4956, 2019.

[27] Michael Kearns, Seth Neel, Aaron Roth, and Zhiwei Steven Wu. Preventing fairness gerrymandering: Auditing and learning for subgroup fairness. In *International Conference on Machine Learning*, pages 2564–2572, 2018.

[28] Jeffrey Kephart and D.M. Chess. The vision of autonomic computing. pages 41 – 50, 2003.

[29] Bahador Khalegi. The how of explainable ai: Explainable modelling, 2019. URL https://towardsdatascience.com/the-how-of-explainable-ai-explainable-modelling-55c8c43d7bed.

[30] Bahador Khalegi. The how of explainable ai: Post-modelling explainability, 2019. URL https://towardsdatascience.com/the-how-of-explainable-ai-post-modelling-explainability-8b4cbc7adf5f.

[31] Bahador Khalegi. The how of explainable ai: Pre-modelling explainability, 2019. URL https://towardsdatascience.com/the-how-of-explainable-ai-pre-modelling-explainability-699150495fe4.

[32] Sasan Maleki, Long Tran-Thanh, Greg Hines, Talal Rahwan, and Alex Rogers. Bounding the estimation error of sampling-based shapley value approximation with/without stratifying. 2013.

[33] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *ACM Computing Surveys*, pages 1–35, 2021.

[34] Carlos Mougan, José Manuel Álvarez, Gourab K. Patro, Salvatore Ruggieri, and Steffen Staab. Fairness implications of encoding protected categorical attributes. 2022.

[35] Tomoki Nakamaru and Shigeru Chiba. Generating a generic fluent api in java. *The Art, Science, and Engineering of Programming*, 2020. URL http://dx.doi.org/10.22152/programming-journal.org/2020/4/9.

[36] Geoff Pleiss, Manish Raghavan, Felix Wu, Jon Kleinberg, and Kilian Q Weinberger. On fairness and calibration. In *Advances in Neural Information Processing Systems*, 2017. URL https://proceedings.neurips.cc/paper/2017/file/b8b9c74ac526fffbeb2d39ab038d1cd7-Paper.pdf.

[37] Till Speicher, Hoda Heidari, Nina Grgic-Hlaca, Krishna P. Gummadi, Adish Singla, Adrian Weller, and Muhammad Bilal Zafar. A unified approach to quantifying algorithmic unfairness. 2018. URL http://dx.doi.org/10.1145/3219819.3220046.

[38] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328, 2017.

[39] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *Proceedings of the 30th International Conference on Machine Learning*, pages 325–333, 2013. URL https://proceedings.mlr.press/v28/zemel13.html.

[40] Brian Zhang, Blake Lemoine, and Margaret Mitchell. Mitigating unwanted biases with adversarial learning. pages 335–340, 2018. URL https://www.researchgate.net/publication/330299272_Mitigating_Unwanted_Biases_with_Adversarial_Learning.

Apêndice A

Conceitos complementares

A.1 AI Explainability

AI Explainability é um conceito em IA que propõe a criação de um conjunto de técnicas de Aprendizado de Máquina (ou Machine Learning/ML) que produz modelos mais explicáveis, mantendo qualidade em suas métricas, e permite que os humanos entendam, confiem e gerenciem aplicações baseadas em IA. XAI também absorve conceitos das Ciências Sociais e considera a psicologia da explicação [12]. Um algoritmo de ML explicável precisa não apenas mostrar tomadas de decisão, mas também mostrar o processo que o levou ao tomar tal decisão, de modo que seja compreensível e transparente para humanos.

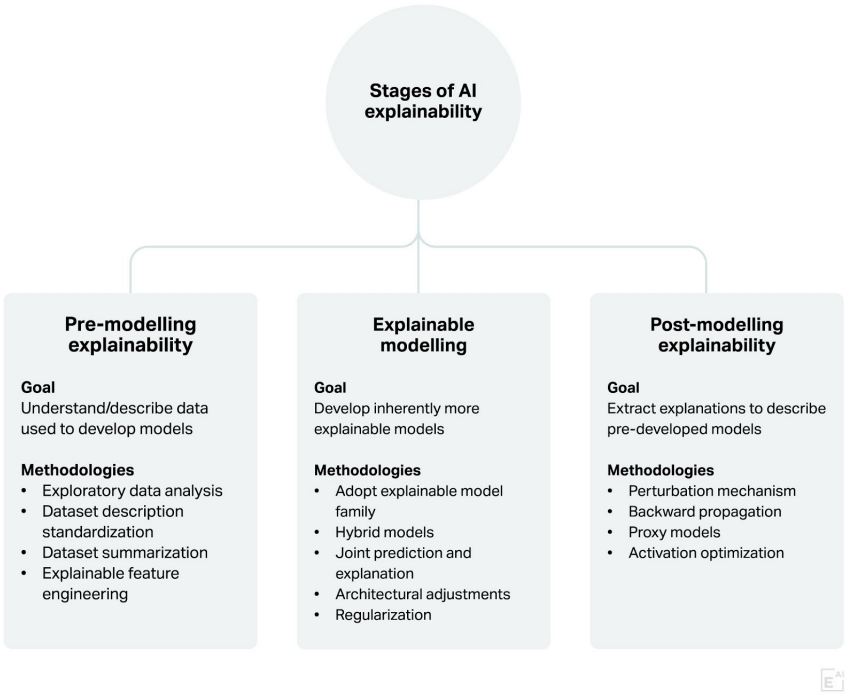

Figura A.1: Fases de um processo baseado em IA explicável.

Conforme ilustrado na Figura A.1, o um processo baseado em IA explicável pode ser classificado em 3 fases:

- ***Pre-Modelling Explainability*:** Esta fase se caracteriza por obter explicações no conjunto de dados usado para treino e validação, tendo como principal motivação o fato do comportamento de um modelo depender muito dos dados que o alimentam. Análises e visualizações dos dados, ou mesmo através de documentações do conjunto de dados se enquadram neste estágio. [31]

- ***Explainable Modelling*:** Esta fase se caracteriza por obter explicações por modelos considerados transparentes (ver adiante). Ao adotar modelos deste tipo, ele já pode ser considerado como explicável por um humano. Também é possível obter a explicação através de regularizadores, ou adotar abordagens onde explicação e resultado sejam obtidos simultaneamente, sejam via junção de dados ou sendo intrínseco a arquitetura do modelo. [29]

- ***Post-Modelling Explainability*:** Também chamada de *Post-hoc Explainability* e onde a maioria dos trabalhos disponíveis baseados em XAI tenta focar, esta fase se dedica a explicar modelos desenvolvidos anteriormente, ou modelos que não são considerados como transparentes. Pode-se explicar um modelo por métodos como árvores de decisão e estimativas como valores de Shapley e propagação inversa. [30]

Entretanto, estas 3 fases não necessariamente são executadas de maneira sequencial: Por ser realizada no conjunto de dados e não no modelo, a fase de *Pre-modelling Explainability* pode ser opcional, e a aplicação das fases de *Explainable Modelling* e *Post-Modelling Explainability* e seus métodos dependem exclusivamente se o modelo é classificado como transparente ou não, e tal critério é determinado pela adoção de uma (ou mais) das seguintes características [12]:

- **Simulabilidade:** Denota a capacidade de um modelo de ser simulado ou pensado estritamente por um humano. A complexidade assume um lugar dominante nesta classe: Sistemas com uma grande quantidade de regras, por mais simples que essas sejam, já não podem ser classificados como tal. O modelo que se adequa a essa característica precisa ser autocontido o suficiente para que um ser humano pense e raciocine sobre ele como um todo.

- **Decomponibilidade:** Também considerado como inteligibilidade, significa a capacidade de explicar cada uma das partes de um modelo. Se o modelo não for simples o suficiente, ele precisa ser divisível em várias pequenas partes e cada parte do modelo deve ser compreensível por um ser humano, sem a necessidade de ferramentas adicionais.

- **Transparência do algoritmo:** Trata-se da habilidade do usuário de entender o processo seguido pelo modelo para produzir qualquer saída dada a partir de seus dados de entrada. Colocando de outra forma, um modelo linear é considerado transparente porque sua superfície de erro pode ser entendida e fundamentada, permitindo ao usuário entender como o modelo irá agir em cada situação que pode enfrentar.

A.2 *Fluent API*

O conceito de *Fluent Interface* [21] (também conhecido como *Fluent API* [35]) é uma abstração de código que o organiza de modo a criar uma *Domain Specific Language* (DSL) interna, tendo como prioridade em seu *design* a legibilidade e a fluidez. Embora a construção de uma *Fluent API* consuma tempo, esse tempo pode ser recuperado com o tempo devido a maior facilidade no desenvolvimento e na manutenção dos componentes presentes no sistema.

Seu desenvolvimento lembra o *Design Pattern Builder*, onde seus métodos realizam acões específicas e retornam a referência do Objeto. Sua diferença está no objetivo em que foi desenvolvido: Enquanto no *Builder* o objetivo está na construcão da instância de um Objeto, na *Fluent API* o objetivo está em desenvolver uma série de ferramentas e ações de modo a resolver um problema específico.

A melhor forma de descrever e entender as diferenças e objetivos da Fluent API é através de um exemplo: O Código A.1 representa uma implementação padrão de um sistema, com instanciações de novos Objetos e métodos determinando atribuições e adição do elemento em uma lista.

```
1    private void makeNormal(Customer customer) {
2        Order o1 = new Order();
3        customer.addOrder(o1);
4        OrderLine line1 = new OrderLine(6, Product.find("TAL"));
5        o1.addLine(line1);
6        OrderLine line2 = new OrderLine(5, Product.find("HPK"));
7        o1.addLine(line2);
8        OrderLine line3 = new OrderLine(3, Product.find("LGV"));
9        o1.addLine(line3);
10       line2.setSkippable(true);
11       o1.setRush(true);
12   }
```

Código A.1: Implementação padrão presente em um sistema [21]

O Código A.2 representa o mesmo sistema usando uma implementação com *Fluent API*. Além na diminuição no número de linhas, a implementação é mais legível devido ao menor número de elementos presentes no código: Todas as chamadas e instanciações do Código A.1 estão ímplicitas, e o encadeamento das funções possui uma progressão lógica, mais sucinta e de simples entendimento.

```
1    private void makeFluent(Customer customer) {
2        customer.newOrder()
3                .with(6, "TAL")
4                .with(5, "HPK").skippable()
5                .with(3, "LGV")
6                .priorityRush();
7    }
```

Código A.2: Implementação com o uso de *Fluent API* [21]