UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA NO CURSO DE ENGENHARIA MECATRÔNICA, DA UNIVERSIDADE FEDERAL DE SANTA CATARINA,

VICTOR WILVERT ANTUNES

INFRAESTRUTURA DE MONITORAMENTO E OPERAÇÃO PARA APRENDIZADO DE MÁQUINA

Joinville
2022

VICTOR WILVERT ANTUNES

INFRAESTRUTURA DE MONITORAMENTO E OPERAÇÃO PARA APRENDIZADO DE MÁQUINA

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica no curso de Engenharia Mecatrônica, da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Orientador: Dr. Gian Ricardo Berkenbrock

Joinville
2022

“We can only see a short distance ahead, but we can see plenty there that needs to be done”. (Alan Turing, 1950)

RESUMO

Com o avanço da tecnologia, sistemas que aprendem com os dados estão progressivamente mais relevantes. A adoção dessa categoria de programa se faz presente em diversos setores, que vão desde aplicações comercias voltadas a população geral até sistemas industriais. Entretanto, o desenvolvimento desse tipo de programa se diferencia da forma tradicional de elaboração de aplicações, devido a um fluxo de trabalho diferente que introduz diversos novos problemas. Neste trabalho é apresentado um modelo de projeto e um conjunto de ferramentas voltados para lidar com os desafios introduzidos por esse processo de trabalho. Além disso, assim como aplicações tradicionais, em programas da área de ciência dos dados também é necessário realizar a implantação da aplicação em um sistema final para possibilitar sua utilização. Essa é outra área abordada neste trabalho, em que é apresentado uma infraestrutura voltada para a operação e implantação de projeto desse tipo. Em relação ao monitoramento, é demonstrada a importância do monitoramento desse sistema, já que ele está normalmente inserido em um ambiente dinâmico, e essas constantes mudanças tendem a causar uma degradação da qualidade do sistema. Por fim, considerando o sistema desenvolvido ao longo deste trabalho na totalidade, é então construído um experimento voltado a exemplificar e validar seu funcionamento.

Palavras-chave: Aprendizado de máquina. Desvio de conceito. Monitoramento.

ABSTRACT

As technology advances, systems that learn from data are becoming more and more relevant. The adoption of this category of program is present in various sectors, ranging from commercial applications for the general population to industrial systems. However, the development of this type of program differs from the traditional way of developing applications, due to a different workflow that introduces several new problems. In this work, it is presented a design model and a set of tools to deal with the challenges introduced by this work process. In addition, just like traditional applications, in data science programs it is also necessary to implement the application in a final system to enable its use. This is another area addressed in this work, in which an infrastructure for the operation and deployment of such a project is presented. Regarding monitoring, the importance of monitoring this system is demonstrated, since it is usually inserted in a dynamic environment, and these constant changes may cause a degradation of the program quality. Finally, considering the system developed throughout this work in its entirety, it is then built an experiment aimed at exemplifying and validating its operation.

Keywords: Machine learning. Concept drift. Monitoring.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ALM	Gerenciamento do ciclo de vida de aplicações
API	Interface de Programação de Aplicações
CDF	Função distribuição acumulada
DAG	Grafos acíclicos dirigidos
EDF	Função de distribuição empírica
MAE	Erro médio absoluto
ML	Aprendizado de máquina
RSME	Raiz quadrada do erro médio

SUMÁRIO

1 INTRODUÇÃO

Sistemas de software que aprendem com os dados têm ganhado cada vez mais visibilidade e estão sendo implantados em número crescente em todas as categorias de aplicações. O crescimento emergente da adoção do aprendizado de máquina (ML) em vários âmbitos é evidenciado pela quantidade de recursos financeiros que estão sendo alocados nesse campo. Segundo Bughin et al. (2019) o mercado de ML poderia impulsionar o crescimento da atividade econômica em toda a União Europeia em até 20% até o ano 2030. Além disso, o Fórum Econômico Mundial previu que uma rede de 58 milhões de empregos será criada nos próximos anos devido às tecnologias ML (WEF, 2018).

No desenvolvimento de *software* tradicional existem diversos desafios os quais devem ser superados, tais como testes, revisão de código e monitoramento, para se realizar o gerenciamento do ciclo de vida de aplicações (ALM) (REDHAT, 2020). Com isso, o ALM abrange as pessoas, as ferramentas e os processos que gerenciam o ciclo de vida de uma aplicação, desde o conceito até o seu fim. ALM é composto por várias disciplinas que incluem gerenciamento de projetos, gerenciamento de requisitos, desenvolvimento de aplicações, teste e controle de qualidade, implantação e manutenção.

De acordo com Zaharia et al. (2018) o desenvolvimento de aplicações de ML requer a resolução de novos problemas que não fazem parte do ciclo de vida de desenvolvimento de *softwares* gerais. Por exemplo, enquanto o *software* tradicional tem um conjunto bem definido de características relativas à aplicação a ser elaborada, o desenvolvimento de ML tende a girar em torno da experimentação. Desta forma, o desenvolvedor experimentará constantemente novos conjuntos de dados, modelos, bibliotecas de *software*, parâmetros de ajuste, de modo a otimizar alguma métrica comercial, como a precisão do modelo. Como o desempenho do modelo depende muito dos dados de entrada e do processo de treinamento, a reprodutibilidade é primordial em todo o desenvolvimento de ML.

Além disso, de modo a se obter alguma utilização real, as aplicações de ML precisam ser implantadas na produção, o que significa tanto implantar um modelo de forma que possa ser usado para inferência quanto implantar sistemas voltados a monitoramento e observabilidade do mesmo, de modo a atualizar regularmente o modelo. Isto é especialmente desafiador quando a implantação requer colaboração com outra equipe, como engenheiros de aplicação, que não são especialistas em ML (LEE et al., 2018).

Segundo Sculley et al. (2015), em sistemas de ML no mundo real apenas uma

pequena fração do projeto é composta de código voltado a ML. Há uma vasta gama de infraestrutura e processos ao redor para apoiar sua evolução. Ele também discute as muitas fontes de débito técnica que podem se acumular em tais sistemas, algumas das quais estão relacionadas à dependência de dados, complexidade de modelos, reprodutibilidade, testes, monitoramento e lidar com as mudanças no mundo externo. Desta forma, é notável que a maioria dos problemas de débito técnico no âmbito do ML são originados dos sistemas externos necessários para gerenciar o ciclo de vida destas aplicações.

Devido os ambientes nos quais modelos de ML são implementados normalmente serem dinâmicos, os modelos presentes nestes tendem a degradar devido a mudanças nas distribuições de dados ou conceitos. Segundo Bachinger, Kronberger e Affenzeller (2021) pode-se assumir que exista um único modelo melhor para a aplicação que possa ser mantido por longos períodos. Em vez disso, é necessário assumir um ambiente em contínua mudança e preparar a infraestrutura e os processos de monitoramento e adaptação iterativa ou contínua dos modelos de ML.

Considerando os problemas apresentados, foi estabelecido como objetivo deste trabalho a definição de diretrizes e uma plataforma que possibilitem o gerenciamento de ciclo de vida de aplicações de ML, definindo as ferramentas a serem usadas durante o desenvolvimento, voltadas a reprodutibilidade e cooperação, e as aplicações inclusas na infraestrutura com intuito de executar o treinamento e implantação automaticamente e o monitoramento e observabilidade dos modelos em produção.

1.1 OBJETIVO

Considerando o assunto de gerenciamento de ciclo de vida de projetos de ML, buscando uma solução simples e acessível, este trabalho propõe os seguintes objetivos.

1.1.1 Objetivo Geral

Definir as diretrizes e infraestrutura que possibilitem o gerenciamento de ciclo de vida de aplicações de ciência dos dados.

1.1.2 Objetivos Específicos

Neste trabalho, de modo a cumprir o objetivo geral proposto, são definidos os seguintes objetivos específicos:

- Propor um modelo de estrutura para projetos gerais de ciência dos dados;
- Encontrar ferramentas que facilitem colaboração e reprodutibilidade;
- Definir a infraestrutura usada para monitorar e implantar modelos de ML;
- Validar o sistema proposto por meio de um experimento.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão introduzidos os conceitos relacionados às ferramentas empregadas na formulação deste trabalho. Inicialmente serão discutidos os conceitos referentes ao desvio de conceito, entrando em detalhes sobre suas diferentes formas. Em seguida serão abordados os métodos de detecção de desvio de conceito utilizados. Após isso, serão apresentados os conceitos pertinentes sobre o gerenciamento do ciclo de vida de ML. Por fim, são apresentadas diversos trabalhos relacionados ao tema do deste trabalho.

2.1 DESVIO DE CONCEITO

Na área de ML, “conceito” é empregado para definir a distribuição de dados usada para realizar tarefas de classificação, regressão ou tarefas não supervisionadas em um determinado ponto no tempo. Normalmente, espera-se uma geração estável de dados, ou seja, que o conceito não evolua com o tempo. Entretanto, diversas situações no mundo real se encontram em ambientes dinâmicos e não-estacionários. Nesses casos, a distribuição de dados pode mudar com o tempo, produzindo o fenômeno do desvio de conceito (WIDMER, 1994).

Considere um modelo de ML treinado para detectar imagens de um monitor antes da introdução da tela plana. Mesmo se o modelo tivesse 100% de precisão, teria um desempenho inferior em imagens contemporâneas, em que praticamente todos os monitores são de tela plana, além de possuírem dimensões completamente diferentes em todos os seus aspectos. Desta forma, é possível observar que a ocorrência do desvio de conceito é relacionada à qualidade do modelo de ML.

Problemas de aprendizagem supervisionada são formalmente definidos de acordo com seguinte forma: o objetivo do sistema é realizar a previsão de uma variável alvo y, seja ela numérica ou categórica, dado um conjunto de características de entrada X. Nos dados de treinamento, usados para a construção de modelo, tanto X como y são conhecidos. Nos novos dados, nos quais o modelo preditivo é aplicado, apenas X é conhecido.

Segundo Hoens, Polikar e Chawla (2012) o conceito é definido como as probabilidade a priori de classe $p(y)$ e as probabilidades condicionais de classe $p(X|y)$. Como $p(y)$ e $p(X|y)$ determinam de forma singular a distribuição conjunta $p(X, y)$ e vice-versa, isto é equivalente a definir conceito como a distribuição conjunta $p(X, y)$, o que proposto é por Gama et al. (2014). Considerando que o conceito pode mudar no decorrer do tempo, define-se conceito para um tempo t conforme a Equação 1.

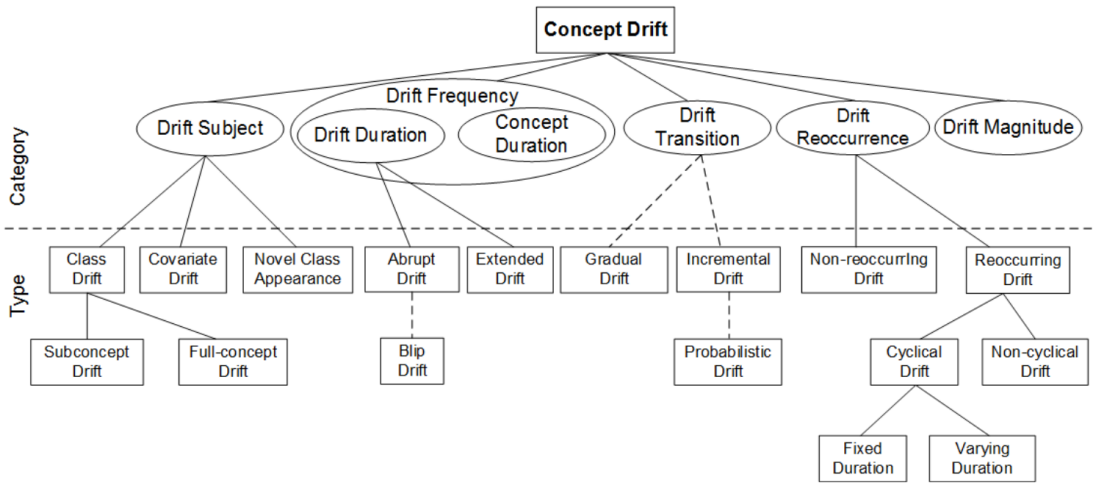
$$Conceito = p_t(X, y) \tag{1}$$

Considerando assim, desvio de conceito é o fenômeno observado da alteração do conceito de um sistema. Formalmente, o desvio de conceito entre os momentos t_0 e t_1 pode ser definido conforme a Equação 2.

$$p_{t_0}(X, y) \neq p_{t_1}(X, y) \tag{2}$$

Na literatura, o desvio de conceito é frequentemente ilustrado com base em exemplos bidimensionais simplificados mostrando o desvio temporalmente (GAMA et al., 2014). Webb et al. (2016) estende esta caracterização de base com uma taxonomia para tipos de desvio de conceito, observável na Figura 1, e fornece uma classificação adicional de desvio de conceito. Entretanto, essas e outras interpretações existentes não possuem consistência em questão à terminologia utilizada. Devido a esse fato, neste trabalho optou-se por seguir a definição proposta por Gama et al. (2014).

Figura 1 – Taxonomia de desvio de conceito



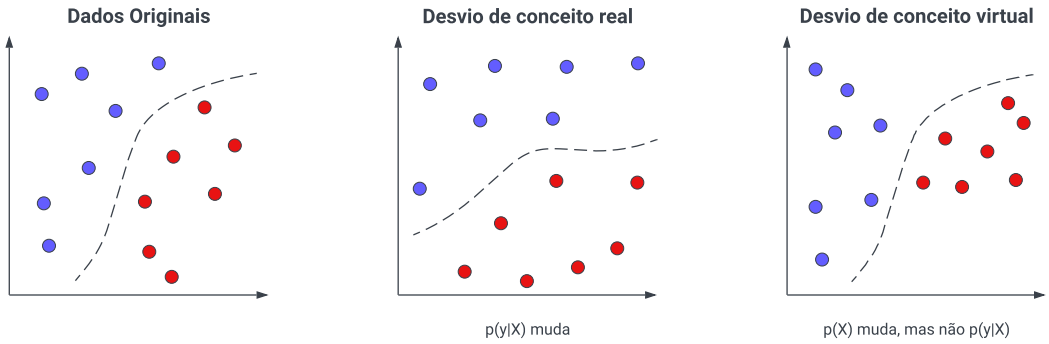
Fonte: Webb et al. (2016)

2.1.1 Tipos de desvio de conceito

Segundo Gama et al. (2014), pode-se destacar a existência de dois tipos de desvio de conceito, demonstrados na Figura 2. Esses desvios são definidos conforme o apresentado a seguir:

a) Desvio de conceito real:
   Refere-se a mudanças nas probabilidades futuras das classes $p(y|X)$. Tais mudanças podem acontecer com ou sem mudança de $p(X)$. Esse desvio define que a relação entre os dados de entrada e saída mudaram. Também é referido como mudança de conceito por Salganicoff (1997), mudança condicional por Gao et al. (2007) e desvio de classe por Webb et al. (2016);
b) Desvio de conceito virtual:

Figura 2 – Tipos de desvio de conceito



Fonte: adaptado Gama et al. (2014)

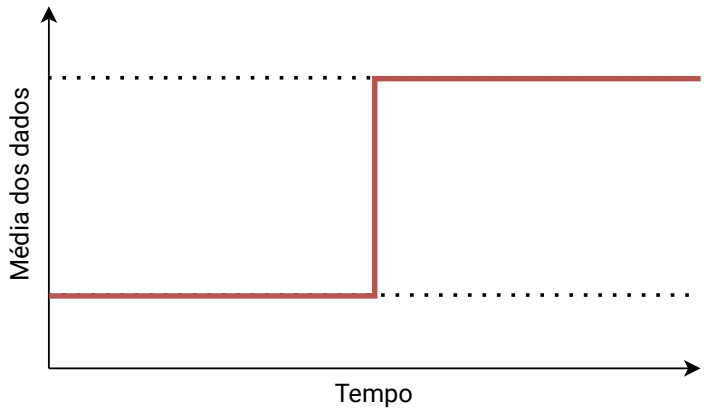
Ocorre se a distribuição dos dados recebidos muda, isto é, $p(X)$ sofre alteração, sem afetar $p(y|X)$. Também é referida como deslocamento de amostragem por Salganicoff (1997), mudança de características por Gao et al. (2007) e desvio covariável por Webb et al. (2016).

2.1.2 Classificação relativa à mudança ao longo do tempo

Mudanças nas distribuições de dados temporalmente podem se manifestar de diferentes formas. Assim, utilizando a análise de uma variável em função do tempo é possível caracterizar o desvio de conceito nesse âmbito. Segundo Gama et al. (2014) é possível observar quatro classificações de desvio nesse contexto:

a) Desvio abrupto:
Ocorre quando a distribuição alvo muda de um conceito a outro abruptamente em um momento, como mostrado na Figura 3.

Figura 3 – Desvio abrupto

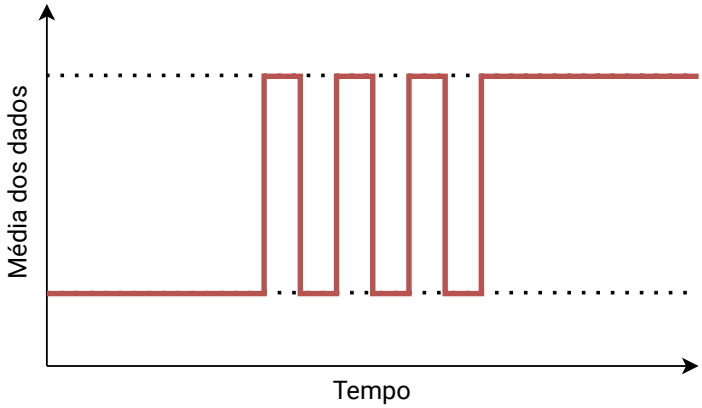


Fonte: adaptado Gama et al. (2014)

b) Desvio gradual:

Ocorre quando a distribuição alvo muda progressivamente de um conceito para outro, como mostrado na Figura 4.

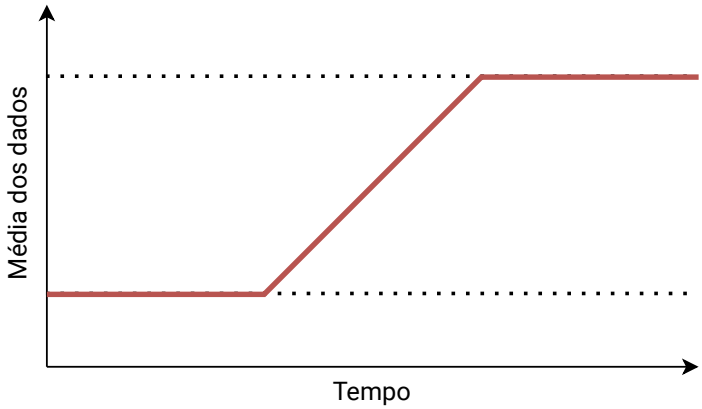
Figura 4 – Desvio gradual



Fonte: adaptado Gama et al. (2014)

c) Desvio incremental:
Ocorre quando um novo conceito substitui o antigo lentamente e de forma contínua, como pode ser observado na Figura 5. Pode ser considerado como um subtipo de desvio gradual, pois nos dois tipos de desvio, os novos conceitos surgem no sistema e substituem completamente o antigo. A diferença no desvio gradual é que não há um limite óbvio que separe a ocorrência do conceito diferente.

Figura 5 – Desvio incremental

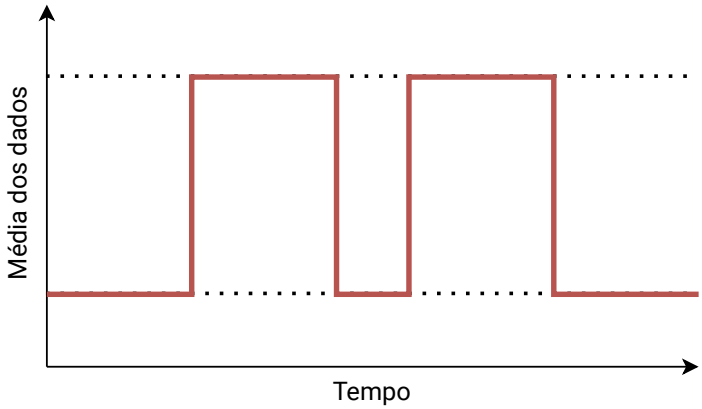


Fonte: adaptado Gama et al. (2014)

d) Conceitos recorrentes:
Ocorre quando um conceito visto anteriormente reaparece novamente após um intervalo de tempo, como mostrado na Figura 6. Este tipo é semelhante ao desvio gradual dado que os dois conceitos se intercambiam no sistema, mas a principal diferença é a fase de transição. No desvio gradual, o antigo conceito começa a

ser gradualmente eliminado e deve ser substituído pelo novo, enquanto no caso de conceitos recorrentes, o conceito anterior reaparece após algum tempo.

Figura 6 – Conceitos recorrentes



Fonte: adaptado Gama et al. (2014)

Gama et al. (2014) também menciona que anomalias e ruídos podem ser confundidos com uma classificação de desvio de conceito, porém isso não é o caso. Quando se analisa o desvio de conceito é necessário levar isso em consideração, analisando estes problemas separadamente para evitar considerações incorretas.

2.2 MÉTODOS DE DETECÇÃO DE DESVIO DE CONCEITO

Há uma grande variedade de técnicas que podem ser aplicadas para detectar o desvio de conceito. Existem métodos que utilizam abordagens baseadas em comparações entre os valores reais e as previsões do modelo, presentes no *Drift Detection Method* (GAMA et al., 2004) e *Early Drift Detection Method* (BAENA-GARCIA et al., 2006). Outra técnica disponível é o uso de modelos mais simples treinados com pequenas porções recentes de dados obtidos durante a execução do sistema, se esses modelos de teste obterem resultados melhores do que o modelo real pode-se dizer que ocorre desvio (BACH; MALOOF, 2008). Entretanto, neste trabalho a abordagem selecionada para realizar a detecção desvio de conceito utiliza métodos puramente estatísticos.

Métodos estatísticos são usados para comparar a diferença entre as distribuições de dados. Eles são empregados para encontrar diferenças entre os dados de diferentes períodos e medir as diferenças no comportamento dos dados à medida que o tempo passa. Nesse tipo de abordagem não é necessário conhecer o valor real da saída para se obter considerações relacionadas às entradas e saídas do modelo em execução. Dessa forma, essa abordagem é a mais genérica e aplicável a qualquer sistema.

São utilizados dois métodos diferentes para a detecção desvio de conceito: o

teste de Kolmogorov-Smirnov de duas amostras para variáveis numéricas e o teste qui-quadrado para variáveis categóricas.

2.2.1 Teste Kolmogorov-Smirnov

O teste Kolmogorov-Smirnov é um teste não paramétrico utilizado para decidir se uma amostra vem de uma população com uma distribuição específica (KOLMOGOROV, 1933; SMIRNOV, 1939). Esse teste é apenas aplicável a uma distribuição contínua, devido a essa limitação ele é usado apenas para dados numéricos. Ele pode ser usado para comparar um conjunto de dados com uma distribuição de probabilidade de referência ou com um segundo conjunto de dados.

Segundo Arnold e Emerson (2011), o teste é baseado na estatística de Kolmogorov–Smirnov que mede a distância suprema entre a função de distribuição empírica (EDF) de um conjunto de dados univariados e sua função distribuição acumulada (CDF), conforme a Equação 3.

$$D_n = \sup_x |F_n(x) - F(x)| \tag{3}$$

Essa estatística é usada para testar a hipótese nula que a CDF $F(x)$ é igual à alguma EDF, sob hipótese, $F_n(x)$. A hipótese relativa à forma de distribuição é rejeitada se a estatística do teste, D_n, for maior que o valor crítico obtido de uma tabela (SMIRNOV, 1948). Há diversas variações dessas tabelas na literatura que usam escalas diferentes para as estatísticas do teste Kolmogorov-Smirnov e as regiões críticas. Entretanto, as implementações existentes desse teste já definem uma tabela utilizada e retornam valores no formato de valor-p.

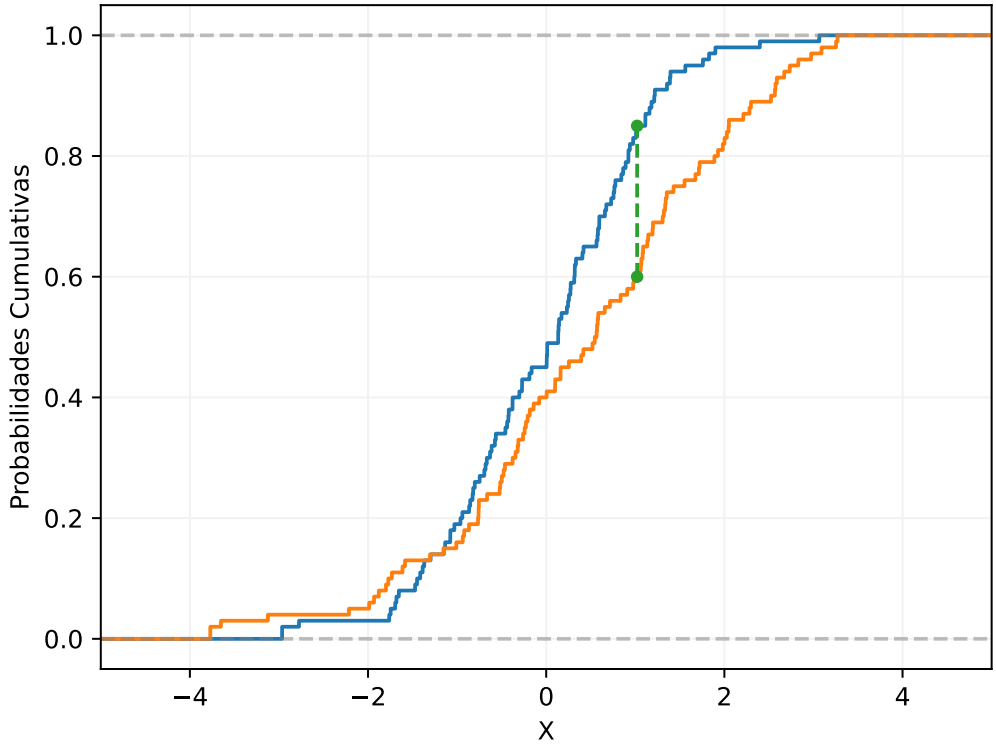
No caso da comparação de duas amostras, ou seja, na análise da diferença entre duas distribuições de dados, a estatística de Kolmogorov–Smirnov é dada conforme a Equação 4. Nesta equação $F_{1,n}$ e $F_{1,n}$ são as EDFs da primeira e da segunda amostra respectivamente.

$$D_{n,m} = \sup_x |F_{1,n}(x) - F_{2,n}(x)| \tag{4}$$

Na Figura 7 é mostrado um exemplo do teste Kolmogorov-Smirnov para duas amostras, nela as curvas azul e laranja representam as EDFs das duas amostras. A linha tracejada em verde representa o valor da estatística do teste, ou seja, a maior distância entre às duas EDFs.

Como método de detecção de desvio de conceito, é utilizado o teste de Kolmogorov–Smirnov para duas amostras. Dessa forma, um dos conjuntos de dados utilizado é composto pelos dados de treinamento e teste e o outro pelos dados obtidos enquanto o modelo está sendo utilizado para inferências. Assim, por meio do teste é

Figura 7 – Teste Kolmogorov-Smirnov para duas amostras



Fonte: elaborado pelo autor (2022)

possível observar se os dados usados no modelo estão divergindo do que o modelo foi treinado para lidar.

2.2.2 Teste qui-quadrado

O teste do qui-quadrado é usado para testar se uma amostra de dados originou de uma população com uma distribuição específica (PEARSON, 1900). Diferente do teste Kolmogorov-Smirnov que é restrito a distribuições contínuas, este teste pode ser aplicado a distribuições discretas. Em função dessa característica, este teste pode ser utilizado para analisar o desvio de conceito em distribuições de dados categóricos.

De acordo com Snedecor e Cochran (1989), a estatística de teste cumulativa de Pearson, que assintoticamente se aproxima da distribuição qui-quadrado é dada conforme a Equação 5. Nesta equação, O_i e E_i são a frequência observada e a frequência esperada da categoria i, respectivamente.

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{5}$$

Para o caso de desvio de conceito, é utilizado o teste qui-quadrado de independência, já que o objetivo do teste é analisar se os dados que o modelo de ML

está recebendo diferem dos utilizados para treinamento e teste. Para poder analisar a hipótese de que os dados são relacionados é necessário comparar o resultado da estatística do teste com valor crítico adequado. Este valor pode ser encontrado em tabelas ou obtido diretamente de uma implementação desse teste, a qual retorna resultados com um valor-p.

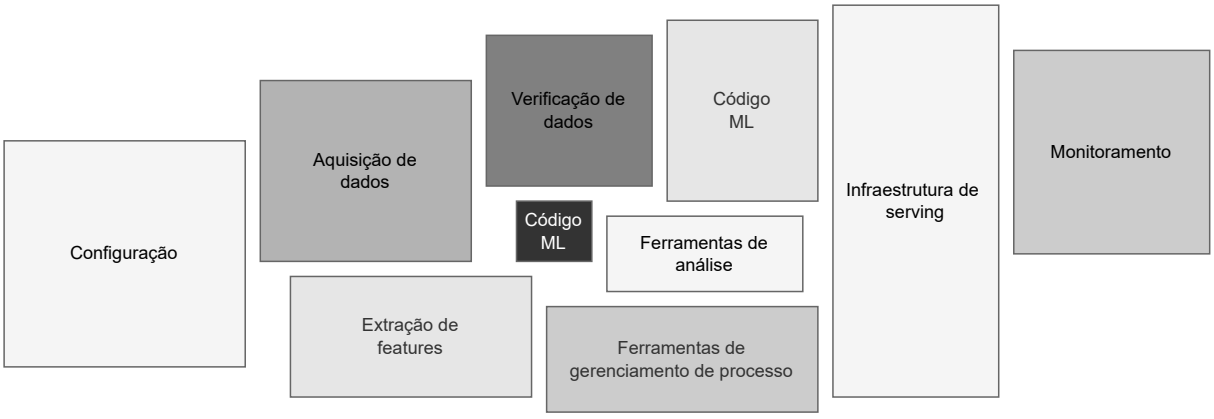
2.3 GERENCIAMENTO DE OPERAÇÕES PARA APRENDIZADO DE MÁQUINA

O termo *MLOps* é utilizado para designar o gerenciamento de operações para ML. Segundo CD Foundation (2022) este termo é definido como a extensão da metodologia *DevOps* para incluir a ML e os ativos da ciência de dados como cidadãos de primeira classe na ecologia *DevOps*. *MLOps*, assim como *DevOps*, surge do entendimento de que separar o desenvolvimento do modelo ML do processo que o implanta diminui a qualidade, a transparência e a agilidade de todo o sistema (INNOQ, 2022).

Os cientistas de dados podem implementar e treinar um modelo de ML com desempenho preditivo em um conjunto de dados de validação *off-line*, com os dados de treinamento relevantes para o caso de uso. No entanto, o verdadeiro desafio não é criar um modelo de ML, mas criar um sistema de ML integrado e operá-lo continuamente no ambiente de produção. Segundo Sculley et al. (2015) apenas uma pequena fração dos sistemas ML do mundo real é composta pelo código de ML.

Conforme mostrado na Figura 8, a menor parcela do sistema e representada pelo código de ML. Os elementos envolventes necessários são grandes e complexos. Nesta figura, o restante do sistema é composto por configuração, automação, coleta de dados, verificação de dados, teste e depuração, gerenciamento de recursos, análise de modelos, gerenciamento de processos e metadados, infraestrutura de observação e monitoramento.

Figura 8 – Elementos para sistemas de ML



Fonte: adaptado Sculley et al. (2015)

Baseado em todos os elementos necessários para a entrega e monitoramento

de um modelo ML à produção é possível definir um processo de trabalho geral para esse tipo de projeto. Segundo INNOQ (2022), todo software baseado em ML inclui três artefatos principais que precisam ser gerenciados, os quais são, dados, modelo de ML, e código. Correspondendo a esses artefatos, o típico fluxo de trabalho de aprendizagem da máquina consiste em três fases principais:

a) Engenharia de Dados: aquisição e preparação dos dados;
b) Engenharia de Modelos de ML: treinamento e validação do modelo de ML;
c) Engenharia de Código: integração do modelo ML em um ambiente de produção.

2.3.1 Engenharia de Dados

A etapa inicial de projetos desse âmbito é a aquisição e preparo dos dados a serem analisados. Tipicamente, os dados estão sendo integrados a partir de vários recursos com formatos diferentes (CHRIST; VISENGERIYEVA; HARRER, 2022). A preparação dos dados segue a etapa de aquisição de dados, sendo um processo iterativo e ágil para explorar, combinar, limpar e transformar dados brutos em conjuntos de dados para integração de dados, ciência de dados e descoberta de dados.

Mesmo que a fase de preparação seja uma fase intermediária destinada a preparar dados para análise, esta fase é a mais custosa no que diz respeito a recursos e tempo (CHRIST; VISENGERIYEVA; HARRER, 2022). A preparação de dados é uma atividade crítica nesse processo, já que é importante evitar a propagação de erros de dados para a fase seguinte, a análise de dados, já que isto resultaria na derivação de percepções errôneas dos dados.

Segundo INNOQ (2022), a *pipeline* de engenharia de dados, formada pela aquisição e preparo dos dados, inclui uma sequência de operações sobre os dados disponíveis que leva ao fornecimento de conjuntos de dados de treinamento e testes para os algoritmos de ML. Essa *pipeline* é dividida em diversas etapas conforme o apresentado a seguir:

a) Ingestão de dados:
   Coleta de dados por meio de várias estruturas e formatos. Esta etapa pode também incluir a geração de dados sintéticos ou o enriquecimento de dados;
b) Exploração e validação:
   Contém a caracterização dos dados para obter informações sobre o conteúdo e a estrutura dos dados. A saída desta etapa é um conjunto de metadados. As operações de validação de dados são funções de detecção de erros definidas pelo usuário, que escaneiam o conjunto de dados de modo a detectar alguns erros;
c) Limpeza de dados:
   O processo de reformatar atributos particulares e corrigir erros nos dados, tais

como a imputação de valores ausentes;

d) Divisão de dados:
Divisão dos dados em conjuntos de treinamento, validação e teste a serem usados durante as fases principais de ML para produzir um modelo.

2.3.2 Engenharia de Modelos

Segundo INNOQ (2022), o núcleo do fluxo de trabalho ML é a fase de escrita e execução de algoritmos de ML para obter um modelo. A *pipeline* da engenharia de modelos inclui uma série de operações que levam a um modelo final, conforme o apresentado a seguir:

a) Treinamento do modelo:
Processo de aplicação do algoritmo de ML nos dados de treinamento para treinar um modelo. Também inclui a *feature engineering* e o ajuste do hiperparâmetros para a atividade de treinamento do modelo;

b) Avaliação do modelo:
Validação do modelo treinado para garantir que ele atenda aos objetivos codificados originais antes de servir o modelo em produção;

c) Teste do modelo:
Realização do Teste de Aceitação de Modelo final, utilizando o conjunto de dados de teste;

d) Empacotamento do modelo:
Processo de exportação do modelo final para um formato específico, que descreve o modelo, de modo a ser consumido pela aplicação final.

2.3.3 Engenharia de Código

Segundo INNOQ (2022), o estágio final do fluxo de trabalho ML é a integração do modelo ML previamente projetado no *software* existente. Esta etapa inclui as seguintes operações:

a) *Model serving*:
O processo de utilização do artefato modelo ML em um ambiente de produção;

b) Monitoramento do desempenho do modelo:
Processo de observação do desempenho do modelo ML baseado em dados em tempo real e previamente não vistos, tais como previsão ou recomendação;

c) Registro de desempenho de modelo:
Toda solicitação de inferência resulta em um registro.

Nessa etapa, em relação à operação de *model serving*, que lida com a forma que o modelo é servido em um ambiente de produção, existem diversas abordagens

para realizar essa operação. De acordo com Sato, Wider e Windheuser (2019), as principais abordagens para realizar essa integração são apresentadas a seguir:

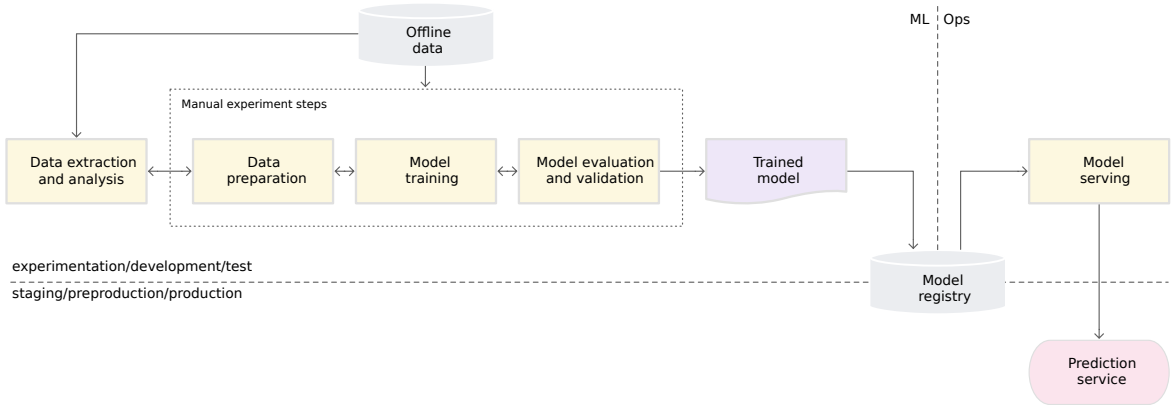
a) Modelo embutido:
   Nesta abordagem o artefato modelo é tratado como uma dependência construída e embalada dentro de uma aplicação consumidora;
b) Modelo implantado como um serviço separado:
   Nesta abordagem, o modelo é envolvido em um serviço que pode ser implantado independentemente das aplicações consumidoras. Isto permite que as atualizações do modelo sejam implantados independentemente, mas também pode introduzir latência no momento da inferência, pois haverá algum tipo de invocação remota necessária para cada previsão;
c) Modelo publicado como dados:
   Nesta abordagem, o modelo também é tratado e publicado independentemente, mas a aplicação consumidora irá ingeri-lo como dados em tempo de execução.

2.3.4 Automação do fluxo de trabalho

Considerando o fluxo de trabalho geral para projeto de ML, existem diversas formas de implementar todos os estágios necessários para a geração e implantação do modelo em um ambiente de produção. As diferentes abordagens podem ser categorizadas com base no nível de automação utilizado nesse processo.

Segundo Google (2020), uma grande parcela dos cientistas de dados e pesquisadores de ML conseguem gerar modelos de ML de última geração, mas seu processo de construção e implantação de modelos ML é inteiramente manual. O diagrama da Figura 9 apresenta o fluxo de trabalho deste processo.

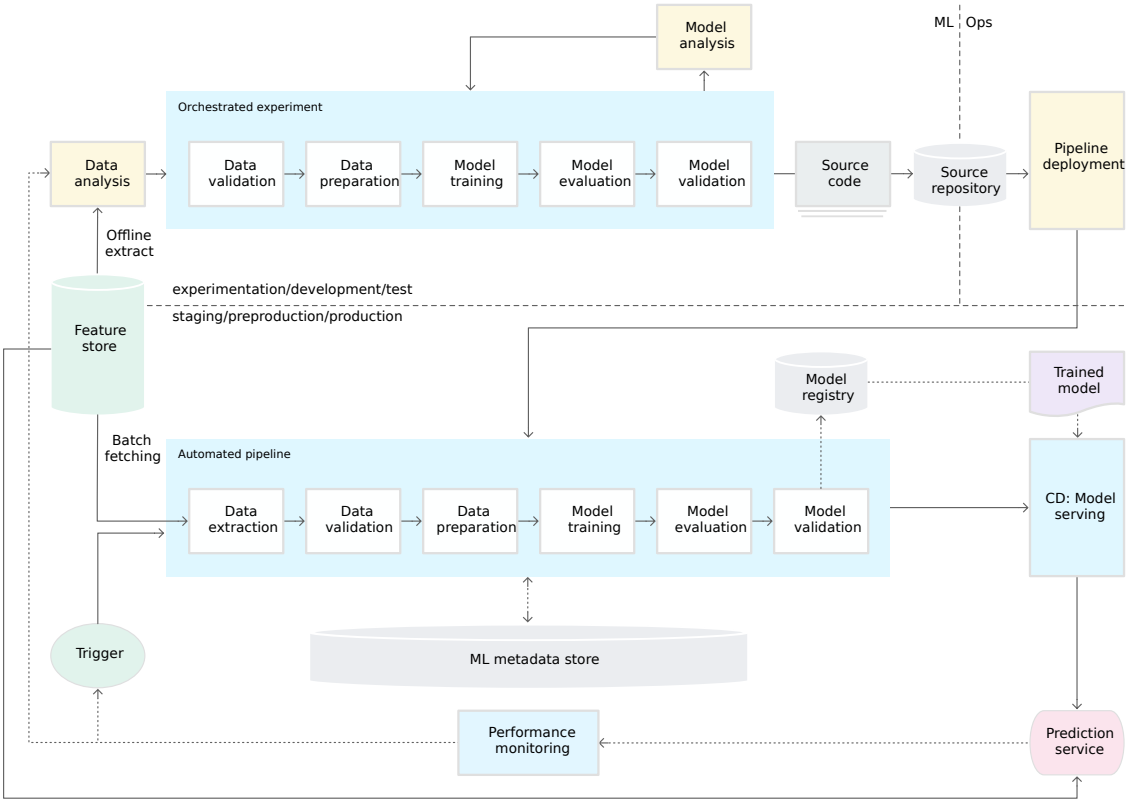
Figura 9 – Fluxo de trabalho de um projeto de ML tradicional



Fonte: Google (2020)

Fora uma solução completamente manual, Google (2020) apresenta que através do uso de sistemas de integração e implantação contínua e possível realizar a

Figura 10 – Fluxo de trabalho de um projeto com automação



Fonte: Google (2020)

automatização de todas as etapas do processo, após sua definição em um ambiente de teste e desenvolvimento. Essa abordagem também considera a possibilidade de realizar o retreinamento automático baseado no monitoramento do modelo implantado. O diagrama apresentado na Figura 10 demonstra o fluxo de trabalho deste processo automatizado.

2.4 TRABALHOS RELACIONADOS

Nesta seção são apresentados alguns trabalhos relacionados ao tema deste trabalho, dos quais, alguns consideram uma abordagem apenas teórica em relação ao tópico, discutindo soluções gerais, enquanto outros têm uma abordagem focada em um âmbito específico, como o ambiente industrial.

Sato, Wider e Windheuser (2019) apresentam uma abordagem de engenharia de software em que uma equipe multifuncional produz aplicações de ML baseadas em código, dados e modelos em pequenos e seguros incrementos que podem ser reproduzidos e liberados de forma confiável a qualquer momento, em curtos ciclos de adaptação.

Bachinger, Kronberger e Affenzeller (2021) discutem a importância de sistemas de gerenciamento de modelos voltados a melhorias e adaptação contínua de modelos

de ML no ambiente industrial. O trabalho é especificamente focado na importância de sistemas desse gênero relacionados com manufatura inteligente, já que neste âmbito a degradação de um modelo preditivo e extremamente relevante e pode gerar grandes custos.

Com o uso de um exemplo de um processo industrial recorrendo a múltiplos modelos preditivos, Bachinger, Kronberger e Affenzeller (2021) demonstram a complexidade da análise do desvio de conceito em casos similares. São apresentadas duas formas de se detectar tal problema, além de se discutir as soluções presentes para se lidar com o problema de gerenciamento de modelos e suas limitações.

Em relação ao gerenciamento de modelos preditivos, Vartak e Madden (2018) e Weber, Hirmer e Reimann (2020) apresentam soluções similares visando manusear os modelos de ML, evitando interferir com outras partes do ciclo de vida de projetos de inteligência artificial. Algo que, como mostrado por Bachinger, Kronberger e Affenzeller (2021), consiste em um problema devido à maioria das soluções presentes serem voltadas a solucionar apenas uma parte do problema sem considerar uma visão geral, o que as faz muitas vezes serem incompatíveis com as outras ferramentas presentes.

Zaharia et al. (2018) discute sobre o gerenciamento do ciclo de vida de aplicações de inteligência artificial apresentando o uso da ferramenta Mlflow (MLFLOW, 2022) para lidar com essa situação. Por meio do uso dessa ferramenta, não se faz apenas o gerenciamento dos modelos, mas também o gerenciamento do projeto em código, pipelines de execução, gerenciamento de experimentos e empacotamento dos modelos em um formato genérico.

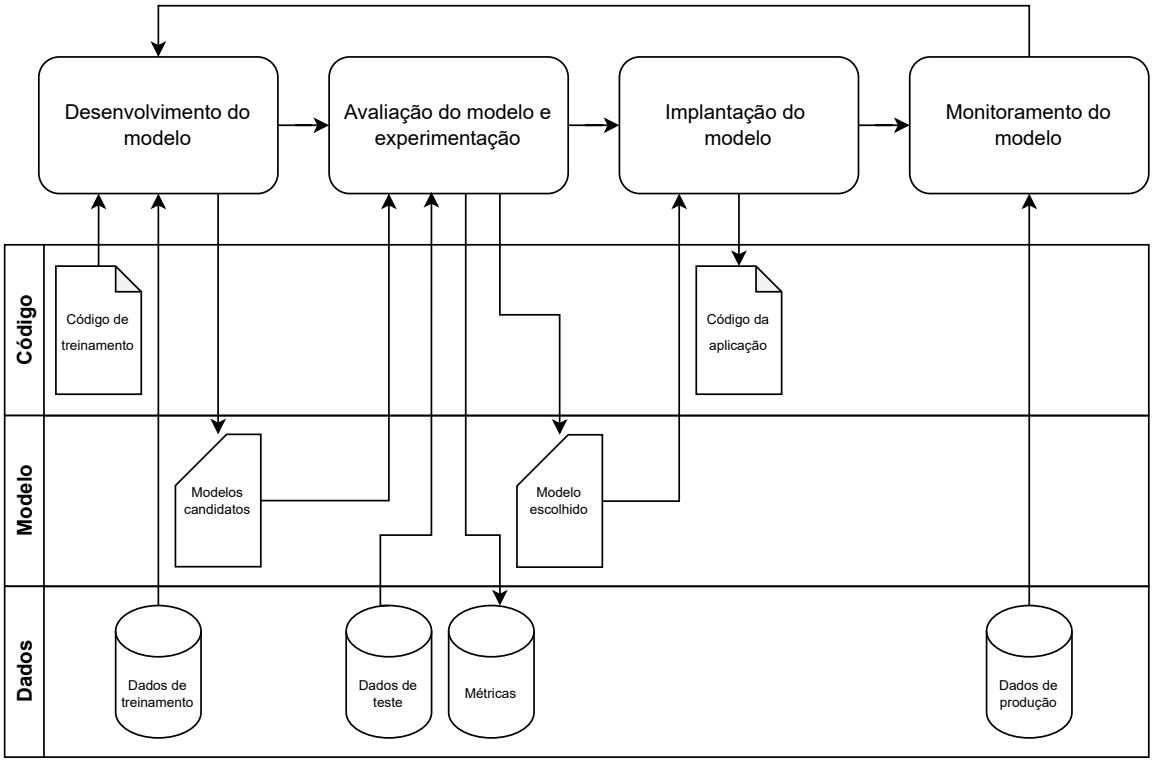
3 MATERIAIS E MÉTODOS

Neste capítulo são introduzidos os métodos utilizados para definir as diretrizes e a plataforma proposta para solução do problema de gerenciamento de ciclo de vida em projetos de ciência dos dados. Inicialmente é apresentado uma visão geral do processo elaborado. Em seguida, a estruturação e as diretrizes definidas para um projeto de ML são demonstradas, descrevendo as ferramentas utilizadas para possibilitar reprodutibilidade e rastreamento de código e dados. Por fim, é demonstrada a infraestrutura implementada para possibilitar o monitoramento e implantação dos modelos.

3.1 VISÃO GERAL

Considerando o fluxo de trabalho apresentado na Seção 2.3, neste trabalho foi desenvolvido um conjunto de diretrizes, projetos e serviços para facilitar a execução de processo. Na Figura 11 é apresentado o fluxo implementado como uma versão simplificada do sistema automatizado apresentado na Subseção 2.3.4. Nela estão denotados os artefatos utilizados em cada estágio do processo.

Figura 11 – Visão geral do fluxo de trabalho do projeto



Fonte: elaborado pelo autor (2022)

3.2 PROJETO PARA DESENVOLVIMENTO DO MODELO

Na análise de dados, normalmente são apenas considerados os relatórios, métricas e visualizações resultantes do projeto. Embora estes produtos finais sejam geralmente o objetivo principal, é fácil acabar se concentrando em produzir os resultados para parecerem esteticamente atrativos e ignorar a qualidade do código utilizado para gerá-los. Como estes produtos finais são criados de forma programática, a qualidade do código ainda é importante, já que nesta área, ela é sobre a exatidão e a reprodutibilidade do projeto.

Dito isto, nesta seção são apresentadas as diretrizes para estruturação de um projeto de ML, definindo a organização base e o método utilizado para reproduzi-la. Além disso, são também apresentadas as ferramentas utilizadas para garantir a reprodutibilidade e rastreamento de diversos experimentos.

3.2.1 Estrutura do projeto

Uma estrutura de projeto bem definida e padronizada tende a funcionar para gerar documentação automaticamente na medida em que a própria organização fornece contexto para o código. Isso facilita que outras pessoas possam trabalhar em conjunto, já que é possível entender uma análise sem se aprofundar em uma extensa documentação e não é necessário ler todo código antes de saber onde procurar partes específicas. Neste trabalho, a estrutura de projeto definida e recomendada pode ser observada na Figura 12.

Figura 12 – Estrutura do projeto

```
├── app/                <- Projeto do serviço para implementação do modelo
├── data/
│   ├── processed/      <- O conjunto final de dados para modelagem
│   └── raw/            <- Os dados originais e imutáveis
├── docs/               <- Um projeto de documentação Sphinx padrão
├── models/             <- Modelos treinados e serializados
├── reports/            <- Resultados do treinamento como metricas e gráficos
├── src/                <- Código fonte para treinamento do modelo
├── .gitignore
├── .gitlab-ci.yml      <- Arquivo de configuração da pipeline de ci/cd
├── Dockerfile
├── requirements.txt    <- O arquivo de requisitos para reproduzir o ambiente de análise
├── LICENSE.txt
└── README.md
```

Fonte: elaborado pelo autor (2022)

Nada nessa estrutura definida é obrigatório, podem ser alterados os nomes de pastas ou arquivos caso seja necessário. A ideia proposta é produzir uma estrutura de projeto que possa ser seguida independente da linguagem de programação e

framework de ML utilizada, separando o código treinamento, de implantação e os artefatos de ML. Para facilitar o acesso a essa estrutura foi criado um modelo usando a ferramenta Cookiecutter (GREENFELD, 2022) que permite reproduzir um novo projeto automaticamente.

3.2.2 Treinamento reproduzível

A reprodutibilidade é fundamental para o desenvolvimento de modelos de ML, já que sem ela, desenvolvedores correm o risco de alegar ganhos com mudanças de parâmetros sem perceber que as fontes ocultas de aleatoriedade são o verdadeiro motivo da melhoria. A reprodutibilidade reduz ou elimina as variações quando se repetem experimentos, dessa forma se fazendo essencial no contexto de tolerância a falhas e refinamento iterativo dos modelos. Esta capacidade se torna ainda mais importante quanto se considera o treinamento remoto.

Visando formalizar o processo de versionamento do código em conjunto com os artefatos de ML, como dados de treinamento, parâmetros e modelos, foi selecionada a ferramenta de código aberto chamada DVC (ITERATIVE, 2022a) em conjunto com o git. Essa seleção é devida ao fato do git não lidar de forma ideal com arquivos grandes, como os dados de treinamento, que através do DVC são salvos em uma base de artefatos externa e rastreados em conjunto com o código por integrações entre as ferramentas.

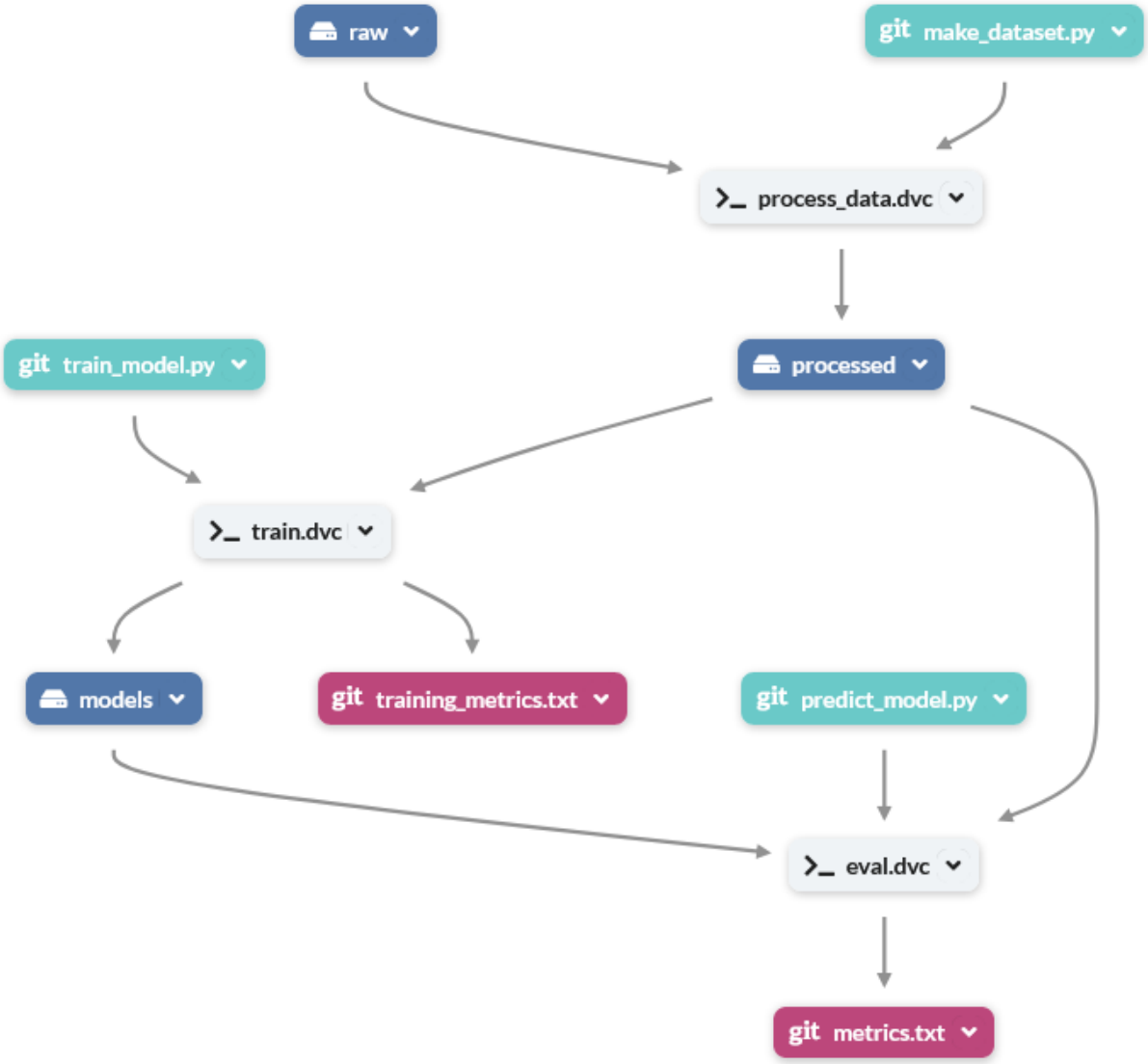
Além disso, o DVC também é utilizado para realizar a execução do projeto em forma de *pipelines* por meio do uso de apenas um comando, dividindo cada passo do processo em um diferente estágio. Por exemplo, o projeto pode ser divido em estágios de processamento de dados, treinamento e avaliação do modelo, fazendo o rastreamento de todos os dados de saída de cada passo, que pode ser visualizado conforme o grafo acíclico dirigido (DAG) na Figura 13. Essas funcionalidades garantem a reprodutibilidade do treinamento realizado.

3.2.3 Rastreamento de experimentos

O treinamento de modelos é um processo iterativo, alterando os valores de diversos parâmetros, verificando o desempenho de cada algoritmo e os aperfeiçoando para obter os resultados ideais. A fim de apoiar este processo, é importante capturar e exibir informações que permitam a escolha do modelo desejado, tanto de forma manual quanto automatizada. Como processos de ML são fortemente centrados em pesquisa, é comum possuir múltiplos experimentos sendo testados em paralelo, e muitos deles podem ser descartados.

Está abordagem de experimentação durante a fase de pesquisa difere de um processo de desenvolvimento de software tradicional, já que se espera que grande

Figura 13 – DAG representando estágios do processo de treinamento

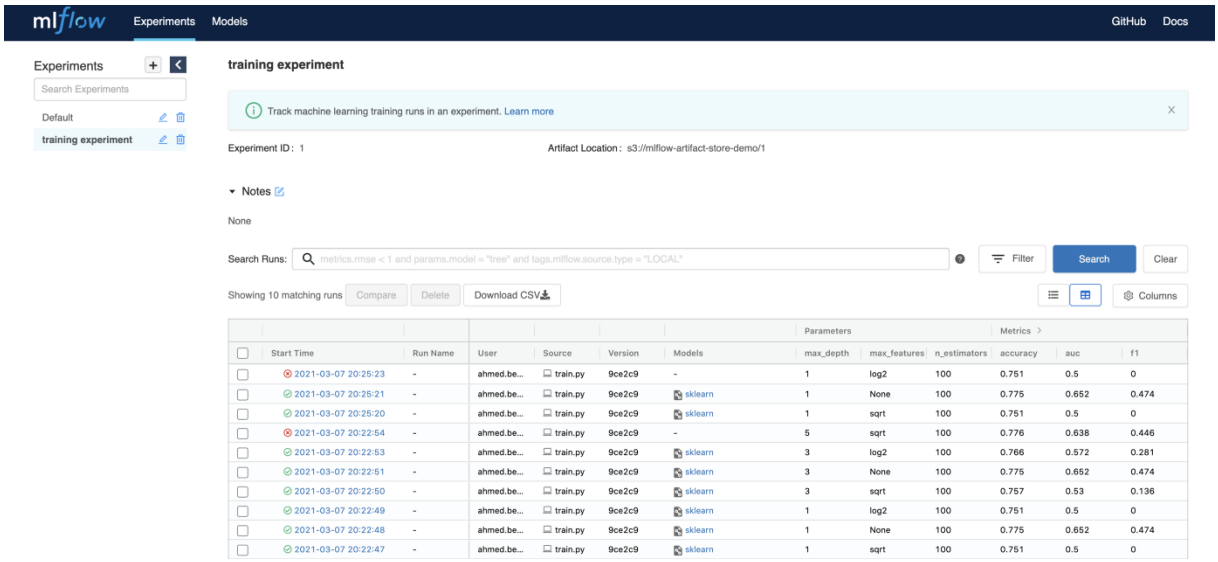


Fonte: elaborado pelo autor (2022)

parte dos experimentos sejam descartados, e apenas uma pequena parcela acabará sendo escolhida para ser utilizada em um ambiente de produção. Devido a isto, se faz necessário definir uma abordagem para gerenciamento e rastreamento de todos os diferentes experimentos e seus componentes, tais como parâmetros, métricas, modelos e outros artefatos.

Dessa forma, com objetivo de realizar o rastreamento dos experimentos foi selecionada a ferramenta MLflow Tracking (MLFLOW, 2022), a qual foi implementada como um serviço. Essa ferramenta fornece uma interface de programação de aplicações (API) para o registro das múltiplas execuções de diversos experimentos. Além disso, também é disponível uma interface gráfica para visualizar os mesmos experimentos, juntamente com seus parâmetros, métricas de desempenho, modelos, artefatos e versão do código utilizado, como mostrado na Figura 14.

Figura 14 – Interface gráfica do *MLflow Tracking*



Fonte: (MLFLOW, 2022)

A ferramenta DVC também possui capacidade para registro e análise de diferentes experimentos, porém em comparação com MLflow Tracking, existem diversas funcionalidades que ainda não estão presentes, especialmente em relação à seleção de execuções especificas, limitando a capacidade de automação. Além disso, para utilização de uma interface gráfica é necessário o uso de uma ferramenta separada, Iterative Studio (ITERATIVE, 2022b), a qual não é de código aberto e apenas funciona a partir de um repositório remoto, limitando a análise de experimentos locais.

3.3 INFRAESTRUTURA DE MONITORAMENTO E OPERAÇÃO

Buscando desenvolver uma plataforma que torne possível realizar o treinamento remotamente, a implantação do modelo de forma automática e o monitoramento do mesmo, é necessário desenvolver uma infraestrutura capaz de atender essas especificações. Dito isso, foram selecionados diversos serviços de código aberto que permitem atender essas necessidades, os quais são implantados por meio da ferramenta Docker Compose (DOCKER, 2022b).

Para realizar as operações do projeto e também funcionar como base para o versionamento de código foi utilizada a plataforma GitLab (GITLAB, 2022). Essa plataforma funciona como um sistema remoto de armazenamento de projetos Git, mas também inclui diversas outras funcionalidades, como o GitLab CI (GITLAB, 2022), utilizado para lidar com todas as operações relacionadas ao código, como o treinamento remoto e implantação do serviço do modelo. Um exemplo de um arquivo de configuração do GitLab CI esta no Apêndice A.

Em relação ao monitoramento, foram selecionados os serviços Grafana (GRAFANA LABS, 2022a) e Prometheus (GRAFANA LABS, 2022b) em conjunto.

Prometheus é uma base de dados de séries temporais utilizada para armazenamento de métricas de monitoramento, o próprio serviço já e configurado para obter as métricas diretamente de outros serviços definidos por meio de um terminal *metrics*. Enquanto isso, Grafana é utilizado para visualização dos dados, permitindo desenvolvimento de *dashboards* com as métricas obtidas do Prometheus, como apresentado na Figura 15.

Figura 15 – Interface gráfica do Grafana

Fonte: (GRAFANA LABS, 2022a)

Além disso, também faz parte da infraestrutura o serviço MLflow Tracking, responsável por armazenar as informações e artefatos de diversos experimentos, como foi demonstrado na Subseção 3.2.1. Dessa forma é possível realizar o compartilhamento dos resultados entre diversos contribuidores do projeto, já que essas informações são salvas externamente. Em conjunto com esse serviço estão inclusas as bases de dados PostgreSQL (POSTGRESQL, 2022) e Minio (MINIO, 2022), as quais funcionam como armazenamento tanto para o MLflow Tracking quanto para o DVC. Os arquivos utilizados para criar a infraestrutura estão no Apêndice B.

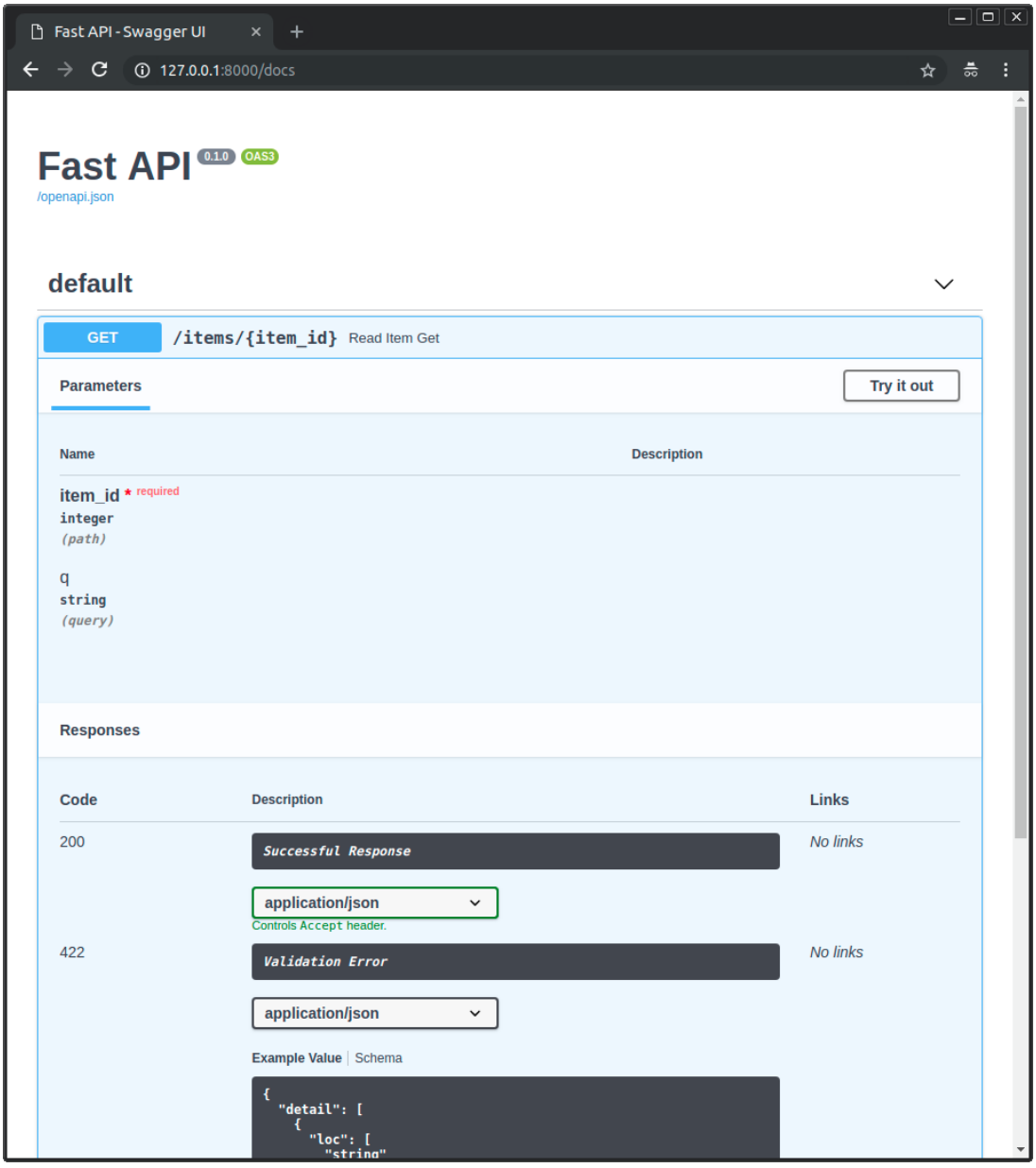
3.3.1 Implantação do modelo como serviço

Uma vez que um modelo adequado é encontrado por meio experimentação, é necessário definir como o mesmo será servido e utilizado em um ambiente de produção. Existem diversas formas de cumprir esse objetivo como apresentado na Seção 2.3.3, e a escolha é normalmente definida conforme o caso de uso em questão. Neste trabalho, buscando definir a forma mais abrangente para a implantação do modelo, foi definido que ele deve ser implantado como um serviço dentro de um contêiner, feito por meio

da plataforma Docker (DOCKER, 2022a).

Para o desenvolvimento do serviço responsável por proporcionar o acesso ao modelo, foi selecionado a *framework* FastAPI (RAMíREZ, 2022), a qual é uma *framework* moderna, de alto desempenho para a construção de APIs usando Python. Por meio dela é possível criar uma API de forma simples, já que grande parte do processo já foi otimizado previamente, reduzindo a ocorrência de erros. Além disso, essa *framework* realiza a geração de duas documentações interativas de forma automática em um de seus terminais. Uma das opções de documentação é demonstrada na Figura 16.

Figura 16 – Documentação interativa da API



Fonte: (RAMíREZ, 2022)

Essa funcionalidade de geração de documentação se faz importante nesse

serviço, já que como se busca implementá-lo da forma mais abrangente e genérica possível, possibilitando que sua utilização em diversos projetos de ML, é necessário ser possível entender o modelo em questão. Por meio disso é possível passar informações do modelo diretamente pela API, como o formato dos dados de entrada, exemplos de execução e descrição do modelo.

Além disso, outro detalhe importante em relação à implantação do modelo, especialmente tendo em consideração o objetivo de ser abrangente, é a serialização do modelo. Como apresentado na Subseção 3.2.1, a estrutura do projeto foi feita buscando ser agnóstico em relação tanto a linguagem de programação quanto a framework de ML utilizada. Para torna possível cumprir esse objetivo foi definido que a serialização do modelo deve ser feita no formato ONNX (ONNX, 2022).

O formato ONNX é um formato de código aberto construído para representar modelos de ML. Ele define um conjunto de operadores universais e um formato de arquivo comum para permitir aos desenvolvedores utilizarem uma variedade de *frameworks* de ML, já que ele permite interoperabilidade. Além disso, o ONNX também permite a execução dos modelos direto de seu formato genérico, o que é utilizado para realização de inferências no serviço proposto.

3.3.2 Monitoramento do modelo e serviço

Em relação ao monitoramento do serviço em questão é necessário analisar dois tipos diferentes de métricas, as API e as do modelo. As métricas advindas da API são compostas por informações relativas ao número de requisições e os tempos de respostas, enquanto as métricas do modelo são relacionadas à qualidade dos dados e do modelo. Para exportar essas métricas para o Prometheus foi utilizado o módulo *starlette_exporter* (HILLIER, 2022).

Enquanto as métricas da API já são geradas automaticamente pelo módulo utilizado, as métricas do modelo precisam ser definidas externamente. Para isso é necessária alguma forma de agregar as informações ao longo do tempo no serviço, algo que no caso dos dados usados para inferência não é viável devido ao seu volume. Levando isso em conta, foi utilizada a biblioteca *whylogs* (WHYLABS, 2022), a qual faz o registro de uma distribuição de dados eficientemente.

Por meio dos registros das distribuições de dados obtidas se torna possível calcular os testes KS ou qui-quadrado dependendo do tipo de dados, apresentados na Seção 2.2. Com isso, é possível analisar se o sistema está sofrendo alguma forma de desvio de conceito virtual diretamente em um ambiente de produção, sem a necessidade de analisar o conjunto de dados externamente ao serviço. Neste trabalho, para comparação dos valores-p gerados pelos testes, foi definido o nível de significância α igual a 0,05.

Além disso, por meio um terminal extra na API do serviço, voltado a receber os

valores reais em comparação as previsões, é possível realizar o cálculo de algumas métricas de qualidade do modelo em produção. Entretanto, essas métricas são limitadas às que podem ser calculados de forma *online*, como, por exemplo, no caso de modelos de regressão, os valores do erro médio absoluto (MAE) e a raiz quadrada do erro médio (RMSE).

4 RESULTADOS E DISCUSSÕES

Neste capítulo é apresentado o experimento executado para demonstração do sistema de ponta a ponta de gerenciamento de um projeto de ML funcionando em sua integridade. As seções são divididas conforme os passos que um projeto convencional tomaria desde a obtenção dos dados até a utilização do modelo implantado. Inicialmente é descrito o conjunto de dados escolhido, e em seguida é apresentado o processo de desenvolvimento de um modelo de ML utilizando esses dados. Por fim, é demonstrado como o monitoramento do modelo ocorre por meio diferentes testes.

4.1 CONJUNTO DE DADOS

Para execução do experimento, foi escolhido um conjunto de dados para ser utilizado tanto para o treinamento na modelagem como para a realização dos testes de monitoramento do modelo concebido. O conjunto escolhido é composto por informações de amostras de vinho tinto da variedade de vinho português “Vinho Verde”, com objetivo de modelar a qualidade do vinho baseado em testes físico-químicos (CORTEZ et al., 2009). Esse conjunto de dados possui 1599 amostras e as informações relativas às variáveis estão presentes no Quadro 1.

Quadro 1 – Descrição das variáveis do conjunto de dados

	Variáveis	Descrição	Unidade
Entradas	fixed acidity	A quantidade de ácidos não voláteis.	g/dm³
	volatile acidity	A quantidade de ácido acético.	g/dm³
	citric acid	A quantidade de ácido cítrico.	g/dm³
	residual sugar	A quantidade de açúcar restante após o término da fermentação.	g/dm³
	chlorides	A quantidade de sal.	g/dm³
	free sulfur dioxide	A quantidade de SO_2 livre.	mg/dm³
	total sulfur dioxide	A quantidade total de SO_2.	mg/dm³
	density	A densidade do vinho.	g/cm³
	pH	Descreve a acidez do vinho em uma escala de 0 a 14.	
	sulphates	Um aditivo para vinho que pode contribuir para os níveis de SO_2.	g/dm³
	alcohol	Teor alcoílico do vinho.	vol.%
Saida	quality	Valor da qualidade do vinho.	

Fonte: Adaptado (CORTEZ et al., 2009).

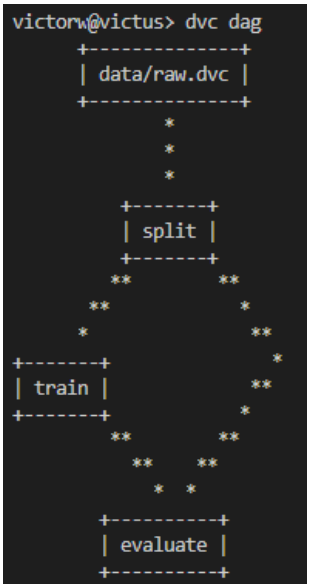
4.2 DESENVOLVIMENTO DO MODELO

Tendo o conjunto de dados definido, passamos para o fluxo de trabalho iterativo de ciência dos dados do desenvolvimento de modelos. Inicialmente realiza-se a divisão dos dados em um conjunto de treinamento e um conjunto de validação. Entretanto, do conjunto original são apenas utilizados os dados com valor de volume percentual de álcool menores ou iguais a 11%, composto de 1191 observações. Esta consideração é relacionada aos testes apresentados na Seção 4.3, em que será realizado o retreinamento com o restante dos dados.

Em seguida, para criar um modelo que será responsável pela previsão dos valores de qualidade do vinho por meio de classificação, utiliza-se o algoritmo arvore de decisão. Esse algoritmo foi escolhido por ser uma forma simples de realizar a classificação dos dados com a possibilidade de controlar um parâmetro do modelo para análise na fase de experimentação. Não foi analisado se o modelo é ideal para esse conjunto de dados, já que otimização não é o foco deste trabalho.

Cada um dos passos definidos é estruturado como um estágio de execução, por meio do uso da ferramenta DVC, definindo a pipeline de treinamento. Os arquivos de código que representam os estágios estão no Apêndice C. Na Figura 17 é possível observar o DAG gerado pelo DVC esta *pipeline*. Nela tem-se os estágios de divisão de dados, treinamento e avaliação do modelo denotados por *split*, *train* e *evaluate* respectivamente. Além disso, é possível notar que os dados puros também são considerados para execução, já que esses são uma dependência para o estágio de divisão.

Figura 17 – DAG gerado pela ferramenta DVC

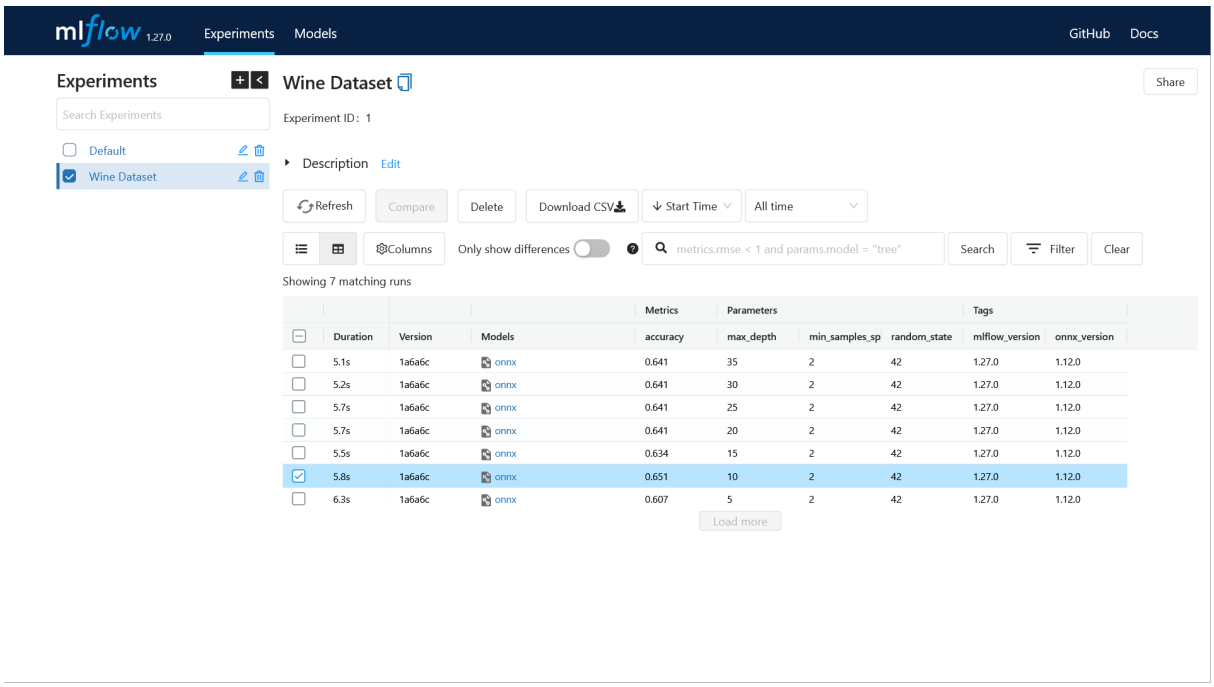


Fonte: elaborado pelo autor (2022)

4.2.1 Experimentação

Para escolher um modelo para utilização no sistema final, se faz necessário realizar diversos experimentos, alternando algoritmos, dados e parâmetros, buscando encontrar algo adequado. No caso do experimento proposto neste trabalho, foram realizados 7 experimentos alternando os valores do parâmetro *max_depth* do modelo de classificação. Os resultados dos experimentos podem ser observados na Figura 18.

Figura 18 – Experimentos utilizando diferentes parâmetros



Fonte: elaborado pelo autor (2022)

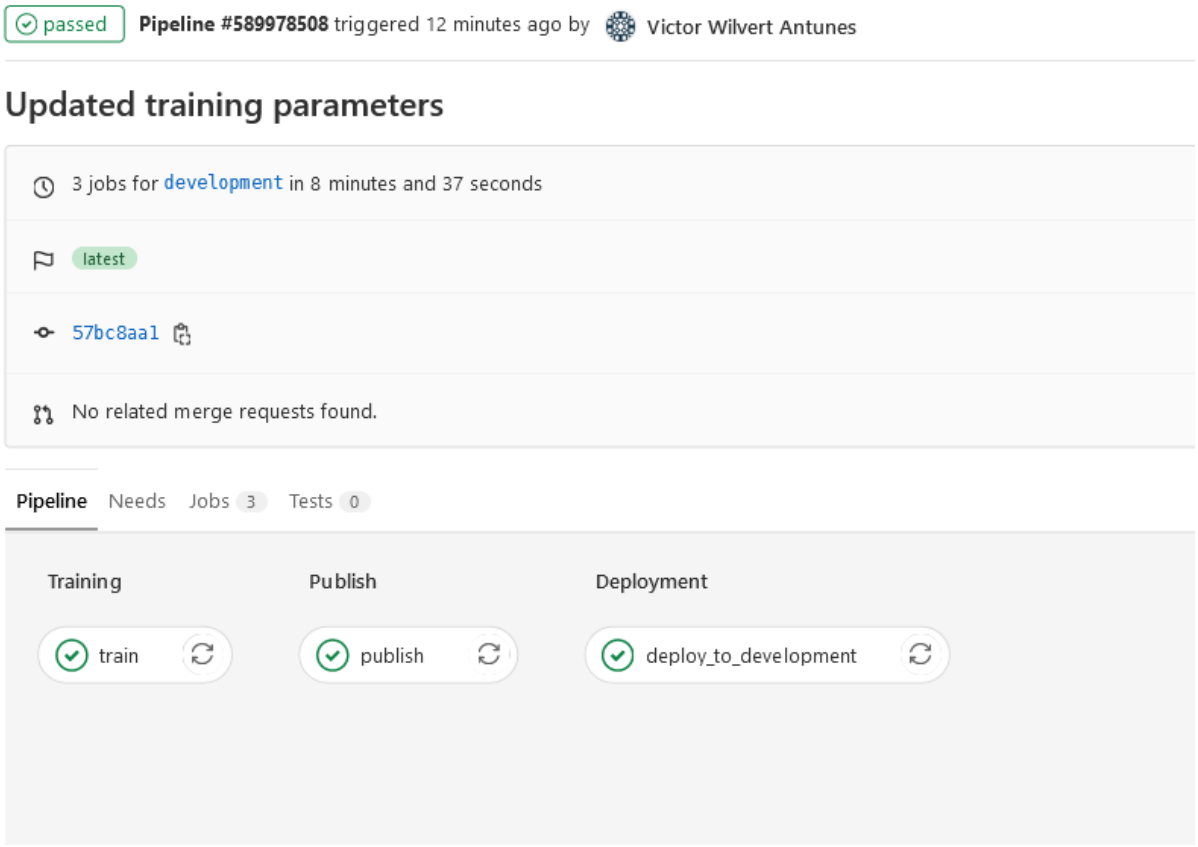
Após a execução dos experimentos foi escolhido o parâmetro *max_depth* igual a 10, já que com esse parâmetro se obteve o melhor resultado na métrica utilizada para avaliação do modelo. A métrica de avaliação utilizada é a exatidão, e o valor obtido, considerando o parâmetro escolhido, é 65,1

4.2.2 Implantação

Como apresentado na Subseção 3.3.1, tendo um modelo escolhido, para possibilitar sua utilização se faz necessário realizar sua implantação, nesse caso empacotado como um serviço. Os arquivos relevantes do código do serviço estão no Apêndice D. O processo de implantação ocorre de forma automática por meio de uma *pipeline* de integração e implantação contínua disparada a cada *commit*, observável na Figura 19. Essa *pipeline* é dividia em três estágios: treinamento, publicação e implantação.

No estágio de treinamento, é executada a *pipeline* de treinamento do projeto por meio do uso da ferramenta DVC. Caso o usuário tenha atualizado os artefatos do

Figura 19 – *Pipelines* de treinamento e implantação no Gitlab CI



Fonte: elaborado pelo autor (2022)

treinamento pelo DVC antes de realizar o *commit*, esse estágio utiliza aqueles dados ao invés de executar a *pipeline* novamente. Após isso, é gerado automaticamente um comentário no *commit* contendo as métricas e gráficos resultantes, observável na Figura 20.

Nos estágios de publicação e implantação, é realizada a construção da imagem Docker contendo o modelo empacotado como uma API e sua implantação no servido que esse serviço será executado. Após a finalização da execução desses estágios, o modelo de ML está pronto para utilização. Um exemplo da documentação interativa gerada automaticamente é apresentado na Figura 21. Nela é possível obter informações sobre as entradas necessárias, as saídas que serão geradas, e a execução de requisições de exemplo.

4.3 SIMULAÇÃO DO TRÁFEGO EM PRODUÇÃO

Para validar o funcionamento do sistema de monitoramento concebido, foram desenvolvidos quatro testes. Dentre eles, os três primeiros são voltados a monitorar a qualidade do modelo e o último monitora as capacidades do sistema. Esse conjunto de testes realiza uma simulação de situações que um modelo implantado como serviço

Figura 20 – Comentário automático sobre o treinamento no *commit*

Fonte: elaborado pelo autor (2022)

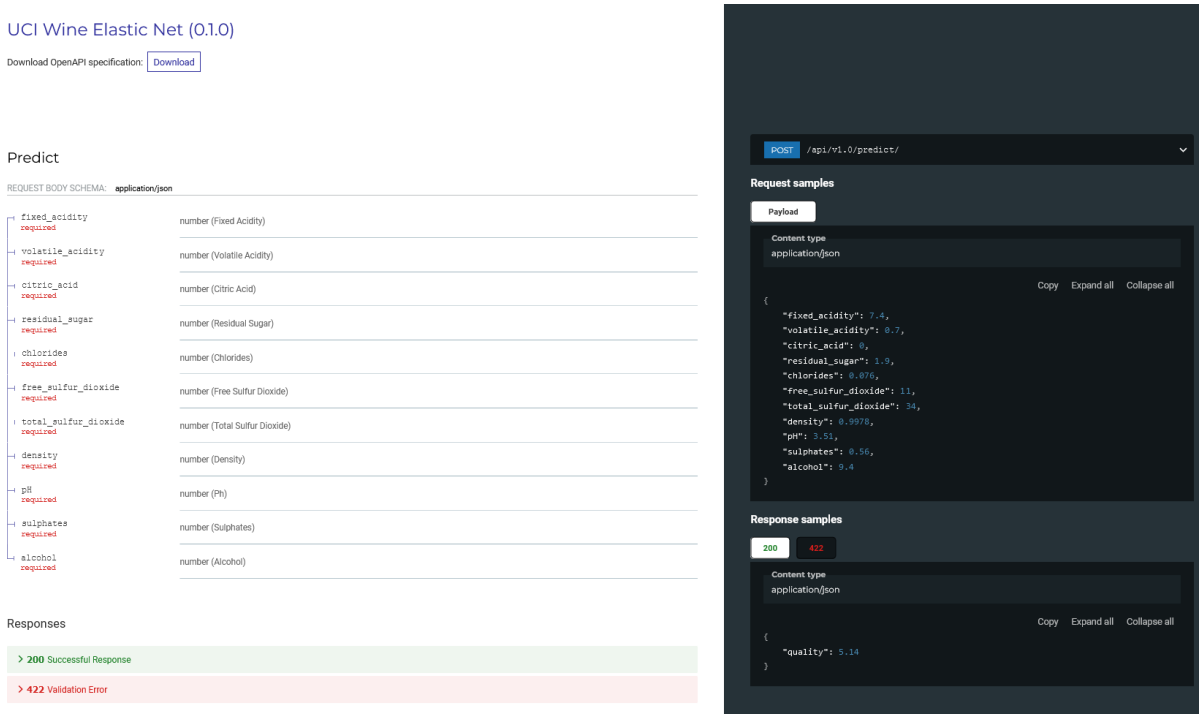
poderia encontrar em um ambiente de produção.

4.3.1 Ausência de desvio de conceito

Esse teste tem o objetivo de verificar o comportamento do modelo em casos que as requisições feitas utilizam dados similares aos utilizados para treinamento e teste, ou seja, na ausência de desvio. Dessa forma, está sendo analisado um caso em que o sistema está operando segundo o planejado durante a modelagem. Para isso são feitas requisições de inferência para o serviço utilizando dados amostrados uniformemente do conjunto de dados usado no treinamento e validação.

Os resultados deste teste podem ser observados na Figura 22, a qual apresenta o *dashboard* de monitoramento de qualidade do modelo e dados. É possível notar duas regiões nos gráficos, já que nos primeiros momentos de execução os valores flutuam consideravelmente. Isso ocorre em função de uma elevada variação

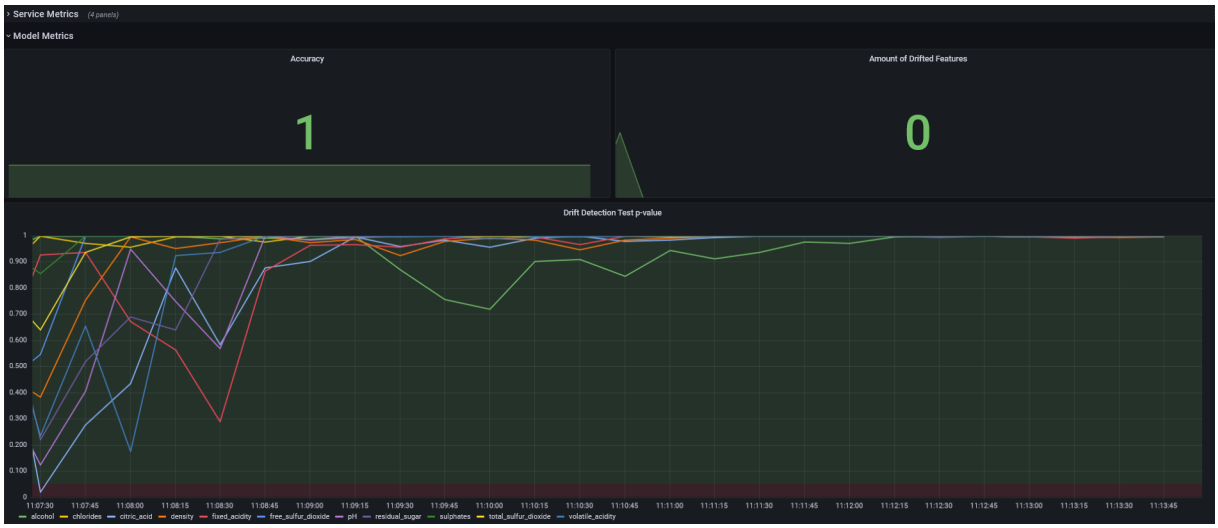
Figura 21 – Documentação interativa *redoc* gerada pela FastAPI



Fonte: elaborado pelo autor (2022)

nos cálculos empregados durante os primeiros valores devido a serem feitos usando distribuições de dados, para qualidade de dados, e médias calculadas em tempo real, para qualidade do modelo.

Figura 22 – *Dashboard* após o teste de ausência de desvio de conceito



Fonte: elaborado pelo autor (2022)

Considerando a segunda região dos gráficos, após o regime em que as métricas estão variando, é possível notar que não há presença de desvio nas variáveis de entrada. Isso é denotado pelo fato de que os valores-p dos testes de desvio de cada variável se encontram todos em torno de 1, significando ausência. Dessa forma é possível

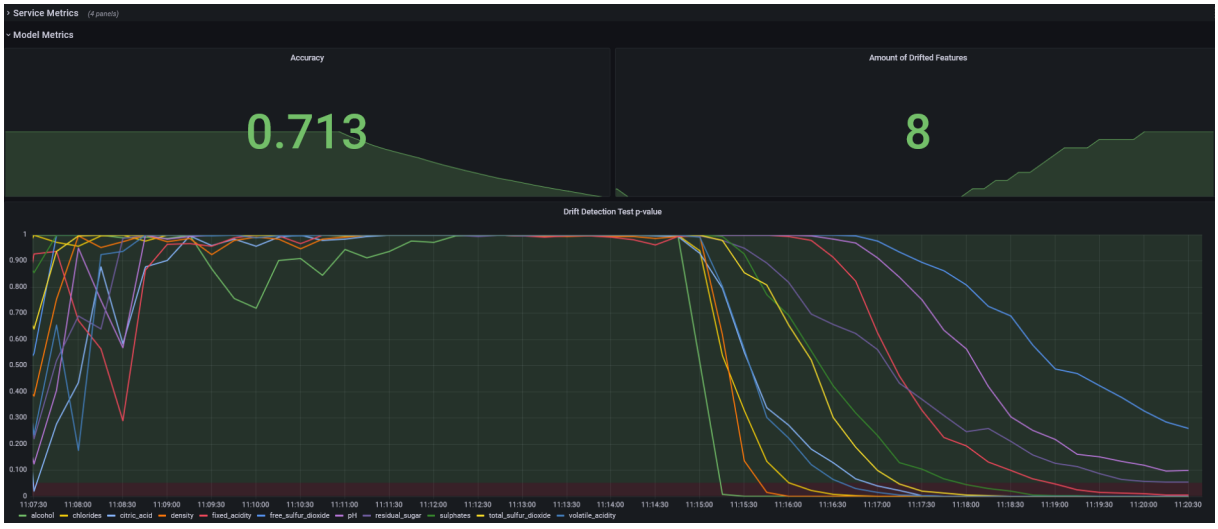
concluir que não há presença de desvio de conceito no sistema em questão por meio do *dashboard*.

4.3.2 Presença de desvio de conceito virtual

Este teste tem o objetivo de verificar o comportamento do modelo em casos que as requisições deixam de utilizar uma distribuição de dados similar à empregada no treinamento de forma abrupta. Dessa forma, está sendo analisado um caso em que o sistema sobre um desvio de conceito virtual abrupto. Para isso são feitas requisições de inferência utilizando dados amostrados uniformemente dos valores do conjunto de dados completo que possuam o valor do volume percentual de álcool acima de 11%.

Os resultados deste teste podem ser observados na Figura 23, a qual apresenta o *dashboard* de monitoramento de qualidade do modelo e dados após a introdução do desvio de conceito virtual abrupto. É possível que tenha havido uma queda nos valores-p dos testes de desvio e que o valor da exatidão do modelo reduziu, denotando uma degradação do modelo de ML.

Figura 23 – *Dashboard* após o teste de desvio de conceito virtual



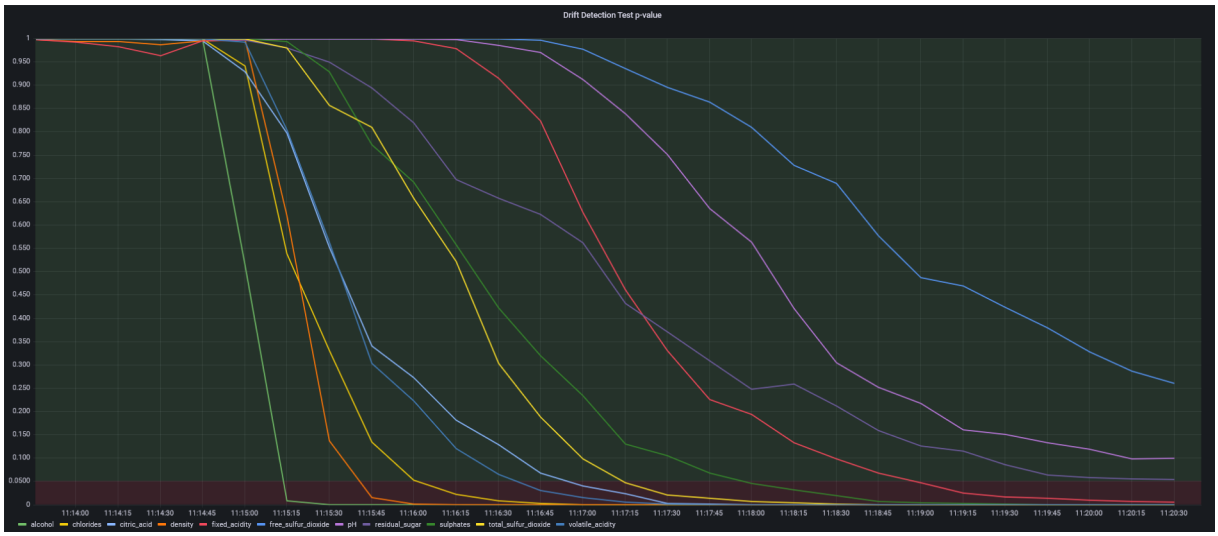
Fonte: elaborado pelo autor (2022)

Na Figura 24 que apresenta o gráfico dos valores-p dos testes de desvio, é possível notar que todos esses valores sofreram redução. Vários deles ficaram abaixo da faixa de 0,05, denotada no gráfico pela região vermelha, que é o valor do nível de significância α, isso significa que o sistema está sofrendo desvio de conceito.

4.3.3 Retreinamento do modelo

Este teste tem o objetivo de verificar o comportamento da implantação de um novo modelo retreinado com os dados observados após a ocorrência de desvio de conceito virtual. Dessa forma, está sendo analisado como seria a resposta após a

Figura 24 – Gráfico dos valores-p dos testes de desvio

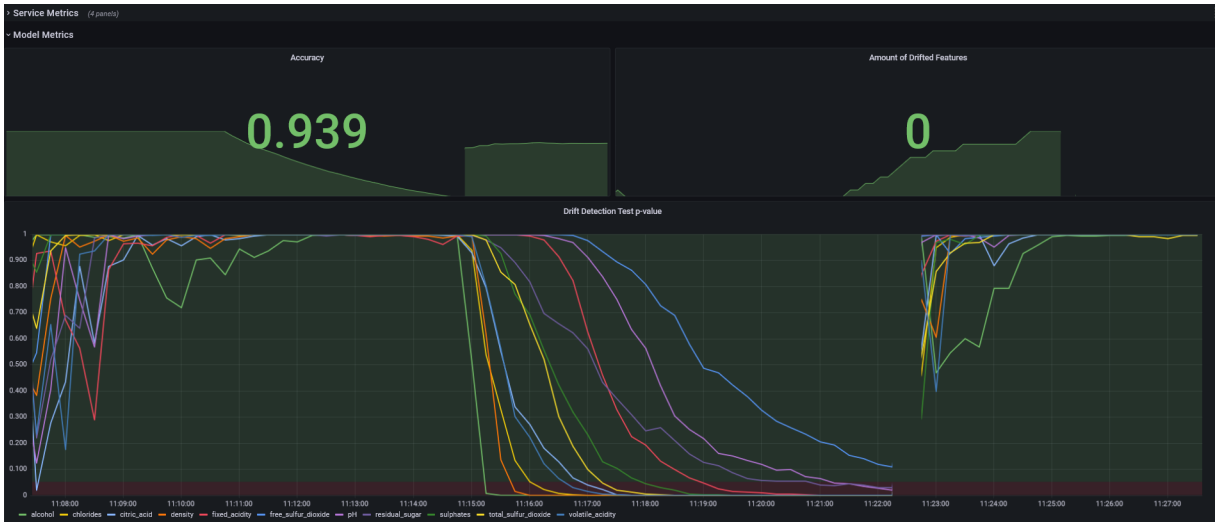


Fonte: elaborado pelo autor (2022)

ocorrência do desvio. Para isso o modelo foi retreinado utilizando os dados do conjunto completo contendo valores de volume de álcool acima de 11%, e o conjunto utilizado para as requisições foi mantido do teste anterior.

Os resultados do teste são mostrados na Figura 25. É possível notar que após o final da região correspondente a introdução do desvio de conceito virtual existe um intervalo sem dados, ocasionado pela implantação do novo modelo. Logo após a introdução desse modelo no sistema, nota-se que os valores-p dos testes de desvio voltam a se aproximar do valor 1, ou seja, não há mais desvio de conceito presente no sistema.

Figura 25 – *Dashboard* após implantação do novo modelo



Fonte: elaborado pelo autor (2022)

Entretanto, ao analisar métricas de qualidade do modelo, pode-se notar que ocorreu uma mudança significativa em relação ao teste anterior, porém o valor ainda não atingiu o mesmo patamar do modelo anterior com os dados sem desvio. Isso ocorre

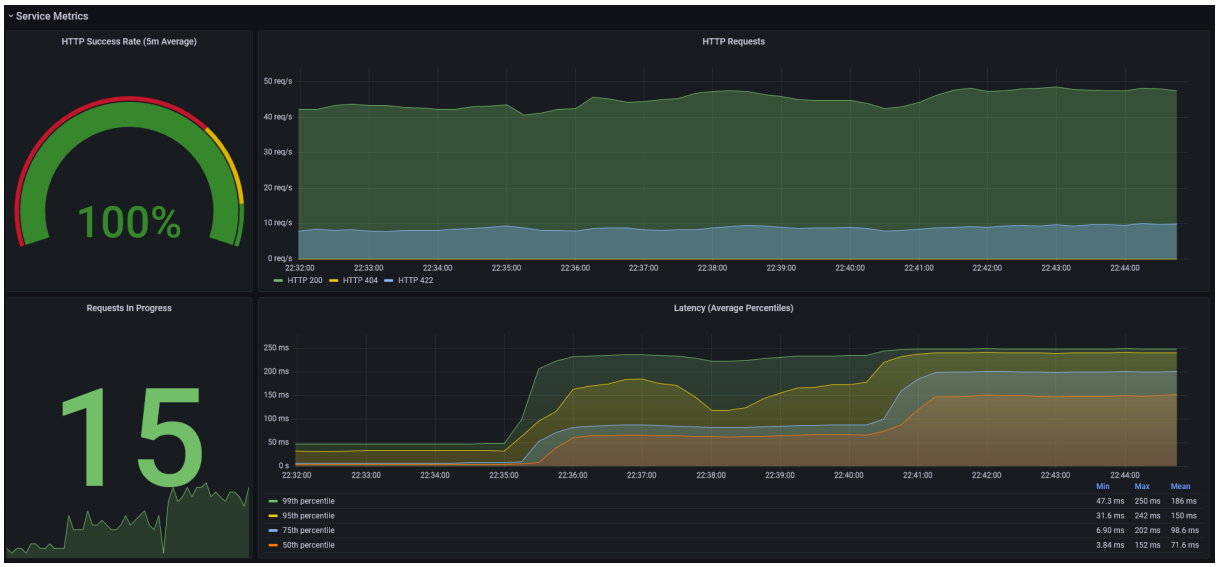
especialmente devido ao tamanho do conjunto de dados utilizado para o retreinamento. Nesse novo conjunto existem apenas 408 observações em comparação às 1191 observações, presentes no conjunto utilizado para o treinamento inicial.

4.3.4 Teste de carga

Este teste tem o objetivo de analisar as capacidades do serviço, para verificar o limite de dados processados pelo software até que ele não consiga mais processar requisições. O teste foi realizado em três etapas, divididas com base no número de usuários simulados executando requisições ao serviço. As etapas foram compostas por 1, 5 e 10 usuários, executando a maior quantidade de requisições possíveis para a API. As requisições enviadas foram dividas em dois grupos, de forma que 80% delas são requisições válidas e 20% possuem uma das variáveis de entrada faltando.

Na Figura 26 é mostrado o *dashboard* com as métricas do sistema durante o teste de carga, nele estão presentes informações como as quantidades totais de requisições em uma faixa de tempo e as latências das requisições. É possível notar que existem dois tipos de retornos para as requisições feitas ao serviço, os retornos com código 200 são para as requisições válidas e os retornos com código 422 são para as que possuem erro no *schema* de dados.

Figura 26 – *Dashboard* durante o teste de carga



Fonte: elaborado pelo autor (2022)

Esse tipo de teste é utilizado para testar as limitações tanto do serviço como do sistema que está hospedando ele. O sistema utilizado para esse tipo de hospedagem é um servidor em caso reais, porém no caso apresentado foi utilizado um computador pessoal. A máquina usada possui 16Gb de memória e um processador AMD Ryzen 7 5800H com um *clock* de 3,2Ghz e 8 *cores*. Por ser um computador convencional é notável que a capacidade de lidar com as requisições não foi muito alta, respondendo a

em torno de 50 requisições por segundo. Um servidor normalmente consegue suportar um fluxo muito maior de dados.

4.4 LIMITAÇÕES

Nesta seção são discutidas algumas das limitações encontradas no desenvolvimento deste trabalho. Essas limitações são advindas das ferramentas e serviços selecionados ou de problemas observados durante o desenvolvimento do trabalho.

4.4.1 Suporte a diferentes tipos de dados

No sistema apresentado na totalidade existe uma limitação relacionada aos tipos de dados e algoritmos usados no modelo na questão do monitoramento. Os métodos utilizados para o cálculo de desvio proporcionados pela biblioteca whylogs são apenas adequados para regressão e classificação de dados tabulares. Dessa forma, dados mais complexos como imagens ou texto solicitam que outros métodos de análise sejam implementados.

4.4.2 Obtenção de métricas de qualidade do modelo

A obtenção de métricas de qualidade do modelo apresentada neste trabalho foi feito de forma basicamente ilustrativa. Isso ocorre por se esperada uma resposta logo em seguida a uma inferência para possibilitar uma visualização coerente. Em casos reais, situações como essa seriam extremamente raras, pois normalmente não se sabe o resultado verdadeiro no mesmo momento realizado que é feita uma previsão.

Além disso, as métricas foram calculadas *online*, ou seja, é feita uma agregação de valores e não se usa o conjunto de dados completo. Esse detalhe impossibilita a implementação de outras métricas que não podem ser analisadas dessa forma, como o coeficiente de determinação. Além disso, isso também impossibilita a análise por meio de uma janela de tempo, importante para reduzir o viés de dados anteriores que vão sendo agregados nessa métricas.

4.4.3 Automatização do treinamento e Implantação

No projeto de gerenciamento de ciclo de vida apresentado já existe a capacidade para se iniciar um novo treinamento de forma automática. Isso poderia ser implementado por meio do uso de *Webhooks* advindos de alarmes no Grafana que poderia iniciar uma nova *pipeline* de treinamento no repositório do projeto do modelo. Entretanto, isso não considera a seleção do conjunto de dados utilizado para esse treinamento.

Para fazer essa seleção são necessárias diversas análises e experimentos, buscando encontrar um modelo adequado ao novo ambiente. Além disso, mesmo que seja possível definir os dados adequados para o treinamento, também é necessário validar se a quantidade de dados disponível é suficiente. Esses problemas requerem um projeto ou serviço separado responsável por essas análises e também um registro de todas as inferências realizadas pelo modelo original. Na Subseção 4.3.3 é possível observar que realizar o retreinamento sem considerar esta situação não necessariamente traz beneficio para o sistema.

5 CONCLUSÕES

Este trabalho realizou a definição de diretrizes, ferramentas, modelos e infraestrutura para o desenvolvimento de um projeto de ML, que inclui não só o treinamento, mas também a análise do modelo após ele ser desenvolvido.

O trabalho é dividido em duas partes principais, as quais são o modelo do projeto de código de ML e a infraestrutura de operação e monitoramento. O primeiro é composto por uma estrutura de projeto de código e ferramentas necessárias para garantir organização, colaboração e reprodutibilidade. Enquanto o segundo garante a automatização e análise do sistema fora do âmbito de experimentação clássico.

Para demonstração do funcionamento do sistema elaborado na totalidade, foi desenvolvido um experimento que aborda um projeto comum de ciência dos dados. Esse experimento consistiu no desenvolvimento e utilização de um modelo de classificação para prever a qualidade de vinhos baseado em suas características químicas, no qual foi implementado em um ambiente de produção simulado.

Desse desenvolvimento foi possível notar que a parcela que lida com a definição e criação do modelo é uma das menores partes em questão de tamanho e trabalho, mesmo que nela esteja o objetivo principal projeto. Todas as outras partes do projeto, que servem para suportar a utilização e desenvolvimento desse processo, demandaram mais tempo e trabalho para serem desenvolvidas.

Isso corrobora o problema do débito técnico oculto apresentado por projetos desse âmbito, em que a maioria do trabalho é voltado para os sistemas que servem de base para o código de ML. Dessa forma, como neste trabalho, como foi desenvolvido todas essas etapas de forma genérica, se torna possível reutilizá-las em outros projetos, tornando viável a simplificação do fluxo de trabalho.

Além disso, também foi demonstrada a importância de se lidar com a degradação dos modelos de ML que se encontram em ambientes dinâmicos. De modo em que as mudanças de contexto em que o sistema está inserido podem causar a degradação do desempenho do modelo utilizado. Entretanto, além da capacidade de analisar a necessidade do sistema precisa ser atualizado, e necessário tomar cuidado com a própria atualização.

5.1 TRABALHOS FUTUROS

A partir do que foi desenvolvido neste trabalho, obteve-se um sistema que apresenta algumas limitações que poderiam ser tratadas em trabalhos futuros. Delas é possível observar três possibilidades de trabalho os quais expandem o que foi desenvolvido aqui.

O primeiro ponto que poderia ser abordado é a expansão dos métodos usados para análise das mudanças de distribuições de dados para mais categorias de dados, como texto e imagens. Assim também aumentaria as possibilidades de algoritmos de ML que o sistema suporta. Para isso seria necessário modificar os métodos de detecção de desvio utilizados, já que eles são feitos apenas para dados tabulares.

Outro trabalho poderia ser feito na análise do desvio de conceito real, o qual precisaria ser implementado em um serviço separado, já que o retorno do *ground truth* não virá com as inferências em todos os casos, similar ao que foi ilustrado neste trabalho. Para atingir esse objetivo também seria provavelmente necessário incluir um *feature store*, algo que não foi implementado na infraestrutura.

Por fim, por meio de uma análise mais detalhada e inclusão de uma *pipeline* de dados é possível desenvolver um trabalho que trata do treinamento continuo e automático do modelo de ML.

REFERÊNCIAS

ARNOLD, T. B.; EMERSON, J. W. Nonparametric goodness-of-fit tests for discrete null distributions. **The R Journal**, v. 3, n. 2, p. 34–39, 2011.

BACH, S. H.; MALOOF, M. A. Paired learners for concept drift. In: **ICDM 2008. Eighth IEEE International Conference on Data Mining**. Los Alamitos, CA, USA: IEEE Computer Society, 2008. p. 23–32. Disponível em: https://doi.ieeecomputersociety.org/10.1109/ICDM.2008.119. Acesso em: 22 jun. 2022.

BACHINGER, F.; KRONBERGER, G.; AFFENZELLER, M. Continuous improvement and adaptation of predictive models in smart manufacturing and model management. **IET Collaborative Intelligent Manufacturing**, Wiley Online Library, v. 3, n. 1, p. 48–63, 2021.

BAENA-GARCIA, M. et al. Early drift detection method. In: AGRAWAL, R.; STOLORZ, P. (Ed.). **Fourth international workshop on knowledge discovery from data streams**. New York: AAAI Press, 2006. v. 6, p. 77–86.

BUGHIN, J. et al. Notes from the ai frontier: tackling europe's gap in digital and ai. **McKinsey Global Institute**, 2019.

CD FOUNDATION. **MLOps SIG**. 2022. MLOps Roadmap 2020. Disponível em: https://github.com/cdfoundation/sig-mlops/blob/master/roadmap/2020/MLOpsRoadmap2020.md. Acesso em: 24 jun. 2022.

CHRIST, J.; VISENGERIYEVA, L.; HARRER, S. **Data Mesh Architecture**: Data mesh from an engineering perspective. 2022. Disponível em: https://www.datamesh-architecture.com/. Acesso em: 24 mar. 2022.

CORTEZ, P. et al. Modeling wine preferences by data mining from physicochemical properties. **Decision support systems**, Elsevier, v. 47, n. 4, p. 547–553, 2009.

DOCKER. **Docker**. 2022. Página inicial. Disponível em: https://www.docker.com/. Acesso em: 24 jun. 2022.

DOCKER. **Docker Compose**. 2022. Documentação. Disponível em: https://docs.docker.com/compose/. Acesso em: 24 jun. 2022.

GAMA, J. et al. Learning with drift detection. In: BAZZAN, A. L. C.; LABIDI, S. (Ed.). **Advances in Artificial Intelligence - SBIA 2004**. Berlin: Springer, 2004. p. 286–295. Disponível em: https://link.springer.com/chapter/10.1007/978-3-540-28645-5_29. Acesso em: 18 jun. 2022.

GAMA, J. et al. A survey on concept drift adaptation. **ACM computing surveys**, Association for Computing Machinery, v. 46, n. 4, p. 1–37, 2014.

GAO, J. et al. A general framework for mining concept-drifting data streams with skewed distributions. In: SIAM. **Proceedings of the 2007 SIAM International Conference on Data Mining**. 2007. p. 3–14. Disponível em: https://epubs.siam.org/doi/abs/10.1137/1.9781611972771.1. Acesso em: 15 jun. 2022.

GITLAB. **GitLab: The One DevOps Platform**. 2022. Página inicial. Disponível em: https://about.gitlab.com/. Acesso em: 24 jun. 2022.

GOOGLE. **MLOps: Continuous delivery and automation pipelines in machine learning**. 2020. Disponível em: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning. Acesso em: 29 mar. 2022.

GRAFANA LABS. **Grafana**. 2022. Página inicial. Disponível em: https://grafana.com/oss/grafana/. Acesso em: 24 jun. 2022.

GRAFANA LABS. **Prometheus**. 2022. Página inicial. Disponível em: https://grafana.com/oss/prometheus/. Acesso em: 24 jun. 2022.

GREENFELD, A. R. **Cookiecutter**. 2022. Repositório GitHub. Disponível em: https://github.com/cookiecutter/cookiecutter. Acesso em: 20 jun 2022.

HILLIER, S. **starlette_exporter: Prometheus exporter for Starlette and FastAPI**. 2022. Repositório GitHub. Disponível em: https://github.com/stephenhillier/starlette_exporter. Acesso em: 24 jun. 2022.

HOENS, T. R.; POLIKAR, R.; CHAWLA, N. V. Learning from streaming data with concept drift and imbalance: an overview. **Progress in Artificial Intelligence**, Springer, v. 1, n. 1, p. 89–101, 2012.

INNOQ. **Machine Learning Operations**. 2022. Página inicial. Disponível em: https://ml-ops.org/. Acesso em: 20 mar. 2022.

ITERATIVE. **Data Version Control (DVC)**. 2022. Página inicial. Disponível em: https://dvc.org/. Acesso em: 24 mar. 2022.

ITERATIVE. **Studio**. 2022. Página inicial. Disponível em: https://studio.iterative.ai/. Acesso em: 24 jun. 2022.

KOLMOGOROV, A. Sulla determinazione empirica di una legge di distribuzione. **Giornale dell'Istituto Italiano degli Attuari**, v. 4, p. 83–91, 1933.

LEE, Y. et al. From the edge to the cloud: Model serving in ml. net. **Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**, v. 41, n. 4, p. 46–53, 2018.

MINIO. **MinIO**. 2022. Página inicial. Disponível em: https://min.io/. Acesso em: 24 jun. 2022.

MLFLOW. **MLflow Project**. 2022. Página inicial. Disponível em: https://mlflow.org/. Acesso em: 12 jun. 2022.

ONNX. **Open Neural Network Exchange**. 2022. Página inicial. Disponível em: https://onnx.ai/. Acesso em: 24 jun. 2022.

PEARSON, K. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. **The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science**, Taylor & Francis, v. 50, n. 302, p. 157–175, 1900.

POSTGRESQL. **PostgreSQL: The World's Most Advanced Open Source Relational Database**. 2022. Página inicial. Disponível em: https://www.postgresql.org/. Acesso em: 24 jun. 2022.

RAMíREZ, S. **FastAPI**. 2022. Repositório GitHub. Disponível em: https://github.com/tiangolo/fastapi. Acesso em: 24 jun. 2022.

REDHAT. **O que é o gerenciamento do ciclo de vida de aplicações (ALM)**. 2020. Disponível em: https://www.redhat.com/pt-br/topics/devops/what-is-application-lifecycle-management-alm. Acesso em: 25 jun. 2022.

SALGANICOFF, M. Tolerating concept and sampling shift in lazy learning using prediction error context switching. In: AHA, D. W. (Ed.). **Lazy learning**. 1. ed. Dordrecht: Springer, 1997. p. 133–155. Disponível em: https://link.springer.com/chapter/10.1007/978-94-017-2053-3_5. Acesso em: 08 jun. 2022.

SATO, D.; WIDER, A.; WINDHEUSER, C. Continuous delivery for machine learning. **Martin Fowler**, 2019. Disponível em: https://martinfowler.com/articles/cd4ml.html. Acesso em: 28 mar. 2022.

SCULLEY, D. et al. Hidden technical debt in machine learning systems. **Advances in Neural Information Processing Systems**, v. 28, 2015.

SMIRNOV, N. V. Estimate of deviation between empirical distribution functions in two independent samples. **Moscow University Mathematics Bulletin**, v. 2, n. 2, p. 3–14, 1939.

SMIRNOV, N. V. Table for estimating the goodness of fit of empirical distributions. **The Annals of Mathematical Statistics**, Institute of Mathematical Statistics, v. 19, n. 2, p. 279–281, 1948.

SNEDECOR, G. W.; COCHRAN, W. G. **Statistical Methods**. 8. ed. Ames: Iowa State University Press, 1989.

VARTAK, M.; MADDEN, S. Modeldb: Opportunities and challenges in managing machine learning models. **Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**, v. 41, n. 4, p. 16–25, 2018.

WEBB, G. I. et al. Characterizing concept drift. **Data Mining and Knowledge Discovery**, Springer, v. 30, n. 4, p. 964–994, 2016.

WEBER, C.; HIRMER, P.; REIMANN, P. A model management platform for industry 4.0 – enabling management of machine learning models in manufacturing environments. In: ABRAMOWICZ, W.; KLEIN, G. (Ed.). **Business Information Systems**. Cham: Springer, 2020. p. 403–417. Disponível em: https://link.springer.com/chapter/10.1007/978-3-030-53337-3_30. Acesso em: 30 mar. 2022.

WEF. The future of jobs report 2018. **Centre for the New Economy and Society**, 2018.

WHYLABS. **whylogs: The open standard for data logging**. 2022. Repositório GitHub. Disponível em: https://github.com/whylabs/whylogs. Acesso em: 24 jun. 2022.

WIDMER, G. Combining robustness and flexibility in learning drifting concepts. In: **Proceedings of the 11th European Conference on Artificial Intelligence**. Amsterdam: Wiley & Sons, 1994. p. 468–472. Disponível em: https://dl.acm.org/doi/10.5555/3070217.3070307. Acesso em: 03 jun. 2022.

ZAHARIA, M. et al. Accelerating the machine learning lifecycle with mlflow. **Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**, v. 41, n. 4, p. 39–45, 2018.

APÊNDICE A - ARQUIVO DE CONFIGURAÇÃO DA PIPELINE DO GITLAB PARA CI/CD

O Gitlab CI faz o uso de um arquivo de configuração no formato YAML que possui o nome gitlab-ci.yml, Através desse arquivo as *pipelines* de operações do repositório são produzidas. A seguir está presente um exemplo contento o arquivo usado para o experimento deste trabalho.

Código 1 – Arquivo de configuração da pipeline do GitLab para CI/CD

```
stages:
  - training
  - publish
  - deployment

variables:
  TAG_LATEST: $CI_REGISTRY_IMAGE/$CI_COMMIT_REF_NAME:latest
  TAG_COMMIT: $CI_REGISTRY_IMAGE/$CI_COMMIT_REF_NAME:
    $CI_COMMIT_SHORT_SHA

train:
  image: iterativeai/cml
  stage: training
  script:
    - pip install -r requirements.txt

    - dvc pull data/raw
    - dvc exp run
    - dvc push

    - echo "## Metrics" >> report.md
    - dvc metrics show --md >> report.md
    - echo "## Plots" >> report.md
    - dvc plots show --show-vega reports/predicted_vs_actual.json > vega
    .json
    - vl2png vega.json > plot.png
    - cml publish --md plot.png >> report.md
    - cml send-comment report.md

    - cp models/model.onnx .
    - cp reports/profile.bin .
  artifacts:
    paths:
      - model.onnx
      - profile.bin
```

```
34
35 publish:
36   image: docker:latest
37   stage: publish
38   services:
39     - docker:dind
40   script:
41     - cp .env.example .env
42
43     - docker build -t $TAG_COMMIT -t $TAG_LATEST .
44     - docker login -u gitlab-ci-token -p $CI_BUILD_TOKEN $CI_REGISTRY
45     - docker push $TAG_COMMIT
46     - docker push $TAG_LATEST
47   only:
48     - development
49     - master
50
51 deploy_to_development:
52   image: alpine:latest
53   stage: deployment
54   script:
55     - chmod og= $ID_RSA
56     - apk update && apk add openssh-client
57
58     - ssh -i $ID_RSA -o StrictHostKeyChecking=no -p $SSH_PORT
       $SERVER_USER@$SERVER_IP "uname -a"
59     - ssh -i $ID_RSA -o StrictHostKeyChecking=no -p $SSH_PORT
       $SERVER_USER@$SERVER_IP "docker version"
60     - ssh -i $ID_RSA -o StrictHostKeyChecking=no -p $SSH_PORT
       $SERVER_USER@$SERVER_IP "docker login -u gitlab-ci-token -p
       $CI_BUILD_TOKEN $CI_REGISTRY"
61     - ssh -i $ID_RSA -o StrictHostKeyChecking=no -p $SSH_PORT
       $SERVER_USER@$SERVER_IP "docker pull $TAG_COMMIT"
62     - ssh -i $ID_RSA -o StrictHostKeyChecking=no -p $SSH_PORT
       $SERVER_USER@$SERVER_IP "docker container rm -f wine-elastic-net-
       development || true"
63     - ssh -i $ID_RSA -o StrictHostKeyChecking=no -p $SSH_PORT
       $SERVER_USER@$SERVER_IP "docker run -d -p 8000:80 --network monitor-
       net --name wine-elastic-net-development $TAG_COMMIT"
64   only:
65     - development
66
67 deploy_to_production:
68   image: alpine:latest
69   stage: deployment
70   script:
71     - chmod og= $ID_RSA
```

```
72      - apk update && apk add openssh-client
73
74      - ssh -i $ID_RSA -o StrictHostKeyChecking=no -p $SSH_PORT
      $SERVER_USER@$SERVER_IP "uname -a"
75      - ssh -i $ID_RSA -o StrictHostKeyChecking=no -p $SSH_PORT
      $SERVER_USER@$SERVER_IP "docker version"
76      - ssh -i $ID_RSA -o StrictHostKeyChecking=no -p $SSH_PORT
      $SERVER_USER@$SERVER_IP "docker login -u gitlab-ci-token -p
      $CI_BUILD_TOKEN $CI_REGISTRY"
77      - ssh -i $ID_RSA -o StrictHostKeyChecking=no -p $SSH_PORT
      $SERVER_USER@$SERVER_IP "docker pull $TAG_COMMIT"
78      - ssh -i $ID_RSA -o StrictHostKeyChecking=no -p $SSH_PORT
      $SERVER_USER@$SERVER_IP "docker container rm -f wine-elastic-net-
      production || true"
79      - ssh -i $ID_RSA -o StrictHostKeyChecking=no -p $SSH_PORT
      $SERVER_USER@$SERVER_IP "docker run -d -p 8010:80 --network monitor-
      net --name wine-elastic-net-production $TAG_COMMIT"
80    only:
81      - master
```

APÊNDICE B - ARQUIVOS PARA INICIAR A INFRAESTRUTURA EM UM AMBIENTE DOCKER

Através da ferramenta Docker Compose é possível iniciar diversos serviços em conjunto. A seguir é demonstrado o código de configuração para realizar isso e um arquivo Dokcerfile necessário para os serviços do MLflow.

Código 2 – Arquivo para iniciar a infraestrutura

```
version: '3'

networks:
  monitor-net:
    name: monitor-net
    external: true

volumes:
    prometheus_data: {}
    grafana_data: {}

services:
  prometheus:
    image: prom/prometheus:v2.36.1
    container_name: prometheus
    volumes:
      - ./prometheus:/etc/prometheus
      - prometheus_data:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--web.console.libraries=/etc/prometheus/console_libraries'
      - '--web.console.templates=/etc/prometheus/consoles'
      - '--storage.tsdb.retention.time=200h'
      - '--web.enable-lifecycle'
    restart: unless-stopped
    ports:
      - "9090:9090"
    expose:
      - 9090
    networks:
      - monitor-net
  grafana:
    image: grafana/grafana:8.5.5
    container_name: grafana
    volumes:
```

```
37      - grafana_data:/var/lib/grafana
38      - ./grafana/provisioning/dashboards:/etc/grafana/provisioning/
    dashboards
39      - ./grafana/provisioning/datasources:/etc/grafana/provisioning/
    datasources
40    environment:
41      - GF_SECURITY_ADMIN_USER=${ADMIN_USER:-admin}
42      - GF_SECURITY_ADMIN_PASSWORD=${ADMIN_PASSWORD:-admin}
43      - GF_USERS_ALLOW_SIGN_UP=false
44    restart: unless-stopped
45    ports:
46      - "3000:3000"
47    expose:
48      - 3000
49    networks:
50      - monitor-net
51  minio:
52    image: minio/minio
53    expose:
54      - "9000"
55    ports:
56      - "9000:9000"
57      # MinIO Console is available at http://localhost:9001
58      - "9001:9001"
59    environment:
60      MINIO_ROOT_USER: "user"
61      MINIO_ROOT_PASSWORD: "password"
62    healthcheck:
63      test: ["CMD", "curl", "-f", "http://localhost:9000/minio/health/
    live"]
64      interval: 1s
65      timeout: 10s
66      retries: 5
67    # Note there is no bucket by default
68    command: server /data --console-address ":9001"
69  minio-create-bucket:
70    image: minio/mc
71    depends_on:
72      minio:
73        condition: service_healthy
74    entrypoint: >
75      bash -c "
76      mc alias set minio http://minio:9000 user password &&
77      if ! mc ls minio | grep --quiet bucket; then
78        mc mb minio/bucket
79      else
80        echo 'bucket already exists'
```

```
      fi
      "
  postgres:
    image: postgres
    restart: always
    environment:
      POSTGRES_DB: db
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
  tracking-server:
    build:
      context: .
      dockerfile: "${DOCKERFILE:-Dockerfile}"
    depends_on:
      - postgres
      - minio-create-bucket
    expose:
      - "5000"
    ports:
      # MLflow UI is available at http://localhost:5000
      - "5000:5000"
    environment:
      MLFLOW_S3_ENDPOINT_URL: http://minio:9000
      AWS_ACCESS_KEY_ID: "user"
      AWS_SECRET_ACCESS_KEY: "password"
    command: >
      mlflow server
      --host 0.0.0.0
      --port 5000
      --backend-store-uri postgresql://user:password@postgres:5432/db
      --artifacts-destination s3://bucket
      --gunicorn-opts "--log-level debug"
      --serve-artifacts
```

Código 3 – Dockerfile para MLflow

```
FROM python:3.7
WORKDIR /app
RUN pip install mlflow psycopg2 boto3
```

APÊNDICE C - CÓDIGOS DA PIPELINE DE TREINAMENTO

Os estágios da *pipeline* de treinamento são a parte que incluí o código de ML no projeto. A seguir estão os códigos de cada estágio usado no experimento deste trabalho.

Código 4 – Estágio de separação dos dados

```
from sklearn.model_selection import train_test_split
import pandas as pd
import yaml
import sys
import os

# Load stage parameters
parameters = yaml.safe_load(open("params.yaml"))["split"]
test_size = parameters["test_size"]
random_state = parameters["random_state"]

# Check script arguments
if len(sys.argv) != 2:
    sys.stderr.write("Arguments error. Usage:\n")
    sys.stderr.write("\tpython split.py data-file\n")
    sys.exit(1)

# Load raw data
data = pd.read_csv(sys.argv[1])

# Split the data into training and test sets.
train, test = train_test_split(
    data,
    test_size=test_size,
    random_state=random_state
)

# The predicted column is "quality" which is a scalar from [3, 9]
train_x = train.drop(["quality"], axis=1)
test_x = test.drop(["quality"], axis=1)
train_y = train[["quality"]]
test_y = test[["quality"]]

# Save prepared data
os.makedirs(
    os.path.join("data", "processed"),
    exist_ok=True
```

```
)
train_x.to_csv(
    os.path.join("data", "processed", "train_input.csv"),
    index=False
)
train_y.to_csv(
    os.path.join("data", "processed", "train_output.csv"),
    index=False
)
test_x.to_csv(
    os.path.join("data", "processed", "test_input.csv"),
    index=False
)
test_y.to_csv(
    os.path.join("data", "processed", "test_output.csv"),
    index=False
)
```

Código 5 – Estágio de treinamento

```
import sys
import pandas as pd
import numpy as np
import os
import yaml
from sklearn.tree import DecisionTreeClassifier
from skl2onnx import to_onnx
import mlflow
import onnx
import dotenv

# Load stage parameters
parameters = yaml.safe_load(open("params.yaml"))["train"]
max_depth = parameters["max_depth"]
min_samples_split = parameters["min_samples_split"]
random_state = parameters["random_state"]

# Check script arguments
if len(sys.argv) != 3:
    sys.stderr.write("Arguments error. Usage:\n")
    sys.stderr.write("\tpython train.py train-input-data-file train-
    output-data-file\n")
    sys.exit(1)

# Load train data
train_input = pd.read_csv(sys.argv[1])
train_output = pd.read_csv(sys.argv[2])

```

```
28 # Load environment variables
29 dotenv.load_dotenv(".env")
30 os.environ["MLFLOW_RUN_ID"] = ""
31
32 # Execute MLFlow run
33 with mlflow.start_run() as run:
34
35     # Update run_id
36     run_id = run.info.run_id
37     dotenv.set_key(".env", "MLFLOW_RUN_ID", run_id)
38
39     # Train model
40     model = DecisionTreeClassifier(
41         max_depth=max_depth,
42         min_samples_split=min_samples_split,
43         random_state=random_state
44     )
45     model.fit(train_input, train_output.values.ravel())
46
47     # MLflow params
48     mlflow.log_param("max_depth", max_depth)
49     mlflow.log_param("min_samples_split", min_samples_split)
50     mlflow.log_param("random_state", random_state)
51     mlflow.set_tag("mlflow_version", mlflow.__version__)
52     mlflow.set_tag("onnx_version", onnx.__version__)
53
54     # Export model in the onnx format
55     onnx_model = to_onnx(model, train_input[:1].astype(np.float32).
    values)
56     mlflow.onnx.log_model(onnx_model, "onnx-model")
57     os.makedirs("models", exist_ok=True)
58     with open(os.path.join("models", "model.onnx"), "wb") as f:
59         f.write(onnx_model.SerializeToString())
```

Código 6 – Estágio de avaliação

```
1 import sys
2 from sklearn import metrics
3 import numpy as np
4 import onnxruntime as rt
5 import pandas as pd
6 import os
7 import json
8 import whylogs as why
9 import mlflow
10 import dotenv
11
12 # Check script arguments
```

```
13 if len(sys.argv) != 6:
14     sys.stderr.write("Arguments error. Usage:\n")
15     sys.stderr.write("\tpython evaluate.py model-file train-input-data-
       file train-output-data-file test-input-data-file test-output-data-
       file\n")
16     sys.exit(1)
17 
18 # Load test data
19 test_input = pd.read_csv(sys.argv[4])
20 test_output = pd.read_csv(sys.argv[5])
21 
22 # Load regression model
23 session = rt.InferenceSession(sys.argv[1])
24 input_name = session.get_inputs()[0].name
25 
26 # Load environment variables
27 dotenv.load_dotenv(".env")
28 
29 # Execute MLFlow run
30 with mlflow.start_run() as run:
31 
32     # Make predictions
33     input_value = test_input[:].astype(np.float32).values
34     predicted = session.run(None, {input_name: input_value})[0].tolist()
35     actual = sum(test_output.values.tolist(), [])
36 
37     # Calculate evaluation metrics
38     accuracy = metrics.accuracy_score(actual, predicted)
39 
40     # MLflow metrics
41     mlflow.log_metric("accuracy", accuracy)
42 
43     # Save model metrics
44     os.makedirs("reports", exist_ok=True)
45     with open(os.path.join("reports", "metrics.json"), 'w') as f:
46         scores = {
47             "accuracy": accuracy,
48         }
49         json.dump(scores, f)
50 
51     # Save confusion matrix
52     classes = pd.DataFrame({"actual": actual, "predicted": predicted})
53     classes.to_csv(os.path.join("reports", "classes.csv"), index=False)
54 
55     # Load train data
56     train_input = pd.read_csv(sys.argv[2])
57     train_output = pd.read_csv(sys.argv[3])
```

```
58
59      # Make predictions
60      input_value = train_input[:].astype(np.float32).values
61      predicted = session.run(None, {input_name: input_value})[0].tolist()
62      actual = sum(train_output.values.tolist(), [])
63      train = train_input
64      train["actual"] = actual
65      train["predicted"] = predicted
66
67      # Save data profile
68      results = why.log(pandas=train)
69      #results.writer("mlflow").write()
70      profile = results.profile()
71      os.makedirs("reports", exist_ok=True)
72      why.write(profile,os.path.join("reports", "profile.bin"))
```

APÊNDICE D - PRINCIPAIS CÓDIGOS DO PROJETO DO SERVIÇO DO MODELO

O serviço do modelo utilizado para realizar inferências com o modelo implantado durante o experimento faz parte de um projeto separado do código de ML. Nos trechos de código a seguir são demonstradas as principais partes do código, como os *schemas* e terminais definidos.

Código 7 – Arquivo principal

```python
from fastapi import FastAPI
from starlette.middleware.cors import CORSMiddleware
from starlette_exporter import PrometheusMiddleware, handle_metrics
from starlette_exporter.optional_metrics import response_body_size,
    request_body_size
from app.api.api_v1.api import api_router
from app.core.config import settings

app = FastAPI(
    title=settings.PROJECT_NAME, openapi_url=f"{settings.API_V1_STR}/
    openapi.json"
)

if settings.BACKEND_CORS_ORIGINS:
    app.add_middleware(
        CORSMiddleware,
        allow_origins=[str(origin) for origin in settings.
    BACKEND_CORS_ORIGINS],
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"],
    )

app.add_middleware(PrometheusMiddleware, optional_metrics=[
    response_body_size, request_body_size])
app.add_route("/metrics", handle_metrics)

app.include_router(api_router, prefix=settings.API_V1_STR)
```

Código 8 – Classe de configurações

```python
from typing import List, Union
from pydantic import AnyHttpUrl, BaseSettings, validator

class Settings(BaseSettings):
    API_V1_STR: str = "/api/v1.0"
    PROJECT_NAME: str
```

```
    MODEL_PATH: str
    PROFILE_PATH: str
    BACKEND_CORS_ORIGINS: List[AnyHttpUrl] = []

    @validator("BACKEND_CORS_ORIGINS", pre=True)
    def assemble_cors_origins(cls, v: Union[str, List[str]]) -> Union[
   List[str], str]:
        if isinstance(v, str) and not v.startswith("["):
            return [i.strip() for i in v.split(",")]
        elif isinstance(v, (list, str)):
            return v
        raise ValueError(v)

    class Config:
        case_sensitive = True
        env_file = ".env"

settings = Settings()
```

Código 9 – Roteador da API

```
from fastapi import APIRouter
from app.api.api_v1.endpoints import predict, feedback

api_router = APIRouter()
api_router.include_router(predict.router, prefix="/predict")
api_router.include_router(feedback.router, prefix="/feedback")
```

Código 10 – Terminal de previsão

```
from typing import Any
from app import schemas
from fastapi import BackgroundTasks, APIRouter
from onnxruntime import InferenceSession
from whylogs.viz.utils.drift_calculations import calculate_drift_values
import numpy as np
import pandas as pd
from prometheus_client import Gauge
import whylogs as why
from app.core.config import settings

DRIFT_VALUES = Gauge('drift_values', 'p-value for applied statistical
   test', ['column', 'test'])
session = InferenceSession(settings.MODEL_PATH)
reference = why.read(settings.PROFILE_PATH)
target = None
router = APIRouter()

def export_metrics(dataframe: pd.DataFrame):
```

```
    # Calculate drift values
    global target
    if target is None:
        results = why.log(pandas=dataframe)
        target = results.profile()
    else:
        target.track(pandas=dataframe)
    drift_values = calculate_drift_values(target.view(), reference.view
    ())

    # Update drift values
    for key, value in drift_values.items():
        DRIFT_VALUES.labels(key, value["test"]).set(value["p_value"])

@router.post("/", response_model=schemas.Prediction)
def predict(
    *,
    background_tasks: BackgroundTasks,
    payload: schemas.Payload
) -> Any:
    # Process payload data
    payload = [payload.dict()]
    payload = pd.DataFrame(payload)

    # Export data drift values
    background_tasks.add_task(export_metrics, payload)

    # Predict output
    input_value = payload.to_numpy().astype(np.float32)
    input_name = session.get_inputs()[0].name
    output_value = session.run(None, {input_name: input_value})[0]

    # Generate response
    prediction = schemas.Prediction(quality=output_value[0])
    return prediction
```

Código 11 – Terminal de retorno

```
from ctypes.wintypes import WORD
from typing import Any, List, Dict
from app import schemas
from fastapi import BackgroundTasks, APIRouter
import math
import pandas as pd
from prometheus_client import Gauge

METRICS = Gauge('metrics', 'model quality metrics', ['metric'])
ACCURACY = 0.0
```

```
CORRECT = 0
COUNTER = 0
router = APIRouter()

def export_metrics(error: float):
    # Calculate regression metrics
    global ACCURACY, CORRECT, COUNTER
    COUNTER = COUNTER + 1
    if error == 0:
        CORRECT = CORRECT + 1
    ACCURACY = CORRECT / COUNTER

    # Export data drift values
    METRICS.labels("accuracy").set(ACCURACY)

@router.post("/", response_model=Dict)
def feedback(
    *,
    background_tasks: BackgroundTasks,
    predicted: schemas.Prediction,
    actual: schemas.Prediction,
) -> Any:
    # Process payload data
    predicted = [predicted.dict()]
    actual = [actual.dict()]
    predicted = pd.DataFrame(predicted)
    actual = pd.DataFrame(actual)

    print(predicted)
    print(actual)

    # Calculate metrics
    error = (predicted - actual).values.tolist()[0][0]

    # Export data drift values
    background_tasks.add_task(export_metrics, error)

    # Return error values
    return {"error": error}
```

Código 12 – Schema das entradas

```
from pydantic import BaseModel, Field

class Payload(BaseModel):
    fixed_acidity: float = Field(example=7.4)
    volatile_acidity: float = Field(example=0.7)
    citric_acid: float = Field(example=0.0)
```

```
    residual_sugar: float = Field(example=1.9)
    chlorides: float = Field(example=0.076)
    free_sulfur_dioxide: float = Field(example=11.0)
    total_sulfur_dioxide: float = Field(example=34.0)
    density: float = Field(example=0.9978)
    pH: float = Field(example=3.51)
    sulphates: float = Field(example=0.56)
    alcohol: float = Field(example=9.4)
```

Código 13 – Schema da previsão

```
from pydantic import BaseModel, Field

class Prediction(BaseModel):
    quality: float = Field(example=5.14)
```