UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Flowi: uma plataforma para desenvolvimento e gerenciamento de modelos de aprendizado de máquina

Leonardo Claudio de Paula e Silva

Dissertação de Mestrado do Programa de Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria (MECAI)

ICMC USP
SÃO CARLOS

Leonardo Claudio de Paula e Silva

Flowi: uma plataforma para desenvolvimento e gerenciamento de modelos de aprendizado de máquina

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre – Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria. *VERSÃO REVISADA*

Área de Concentração: Matemática, Estatística e Computação

Orientador: Prof. Dr. Fernando Santos Osório

USP – São Carlos
Novembro de 2022

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

Claudio de Paula e Silva, Leonardo
C615f Flowi: uma plataforma para desenvolvimento e gerenciamento de modelos de aprendizado de máquina / Leonardo Claudio de Paula e Silva; orientador Fernando Santos Osório. -- São Carlos, 2022.
66 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2022.

1. Aprendizado de Máquina. 2. MLOps. 3. ML Lifecycle. 4. Plataforma. I. Santos Osório, Fernando, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Leonardo Claudio de Paula e Silva

Flowi: a platform for ML development and management

Dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Professional Master's Program in Mathematics Statistics and Computing Applied to Industry, for the degree of Master in Science. *FINAL VERSION*

Concentration Area: Mathematics, Statistics and Computing

Advisor: Prof. Dr. Fernando Santos Osório

USP – São Carlos
November 2022

Este trabalho é dedicado à minha esposa,
que me acompanhou e me auxiliou em tudo.

AGRADECIMENTOS

Primeiro gostaria de agradecer e glorificar a Deus por este trabalho, pois sem Ele nada desse projeto teria acontecido.

Também gostaria de agradecer à minha amada esposa, Ana Elisa, que me sustentou, confrontou e ouviu minhas reclamações e desabafos, também compartilhou comigo as alegrias de pequenas etapas do projeto que foram se encaixando e funcionando.

À minha família, meu porto seguro. Sempre dispostos a me ouvir e a celebrar comigo. Muito obrigado!

À Serasa Experian por me conceder a honra de trabalhar com eles, me capacitarem e me apoiarem neste projeto.

“A busca pela excelência também é
uma maneira de louvar a Deus.”
(Francis Schaeffer)

RESUMO

SILVA, L. C. P. **Flowi: uma plataforma para desenvolvimento e gerenciamento de modelos de aprendizado de máquina**. 2022. 66 p. Dissertação (Mestrado – Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Aprendizado de Máquina (*Machine Learning* - ML - em inglês) tem se tornado a principal tecnologia para automação de diversos casos de usos na indústria; desde detecção de caracteres (OCR - *Optical Character Recognition*) até veículos autônomos. Entretanto, desenvolver e gerenciar esses modelos de aprendizado de máquina em produção é complexo. Especialmente porque quem desenvolve os modelos não necessariamente tem as habilidades para colocá-los em produção e monitorá-los. Este trabalho propõe o Flowi: uma plataforma de gerenciamento do ciclo de vida de aprendizado de máquina. Ela é baseada em componentes para capacitar cientistas de dados a trazer seus conhecimentos aos modelos com escalabilidade, rastreio de experimentos, *deploy*, monitoramento e otimização de hiper-parâmetros por padrão.

Palavras-chave: Aprendizado de Máquina, MLOps, ML *Lifecycle*, Plataforma.

ABSTRACT

SILVA, L. C. P. **Flowi: a platform for ML development and management**. 2022. 66 p. Dissertação (Mestrado – Mestrado Profissional em Matemática, Estatística e Computação Aplicadas à Indústria) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Machine Learning (ML) is becoming a leading technology for several industry automation use cases, from optical character recognition (OCR) to autonomous vehicles. However, developing and managing these machine learning models in production is complex, specially because the one developing the model may not have the skills to deploy and monitor it. This work proposes Flowi as a component based ML lifecycle platform that empowers data scientists to bring their knowledge to the model with built-in scalability, experiment tracking, deploy, monitoring and parameter optimization.

Keywords: Machine Learning, MLOps, ML Lifecycle, Platform.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Interface de Programação de Aplicação *Application Programming Interface*
API REST	API *Representational State Transfer* (Transferência Representacional de Estado)
AWS	*Amazon Web Services*
Azure	*Microsoft Azure Cloud services*
CD	*Continuous Delivery*
CI	*Continuous Integration*
CT	*Continuous Training*
DAG	Directed Acyclic Graph (Grafo Acíclico Dirigido)
DevOps	Development and Operations
EDA	*Exploratory Data Analysis*
GCP	*Google Cloud Platform*
GPU	*Graphical Processing Unit (also GP-GPU General Purpose GPUs)*
ML	Aprendizado de Máquina - *Machine Learning*
MLOps	*Machine Learning Operations*
UI	Interface de Usuário - *User Interface*
vCPU	*Virtual CPU* ou *virtual processor*
VM	Virtual Machine - Máquina Virtual

SUMÁRIO

CAPÍTULO

1

INTRODUÇÃO

Aprendizado de Máquina - *Machine Learning* (ML) em inglês, tem se tornado a principal tecnologia para automação de diversos casos de usos na indústria; desde detecção de caracteres (OCR - *Optical Character Recognition*) (MORI; SUEN; YAMAMOTO, 1992) até veículos autônomos (MOHSENI *et al.*, 2019). Entretanto, desenvolver e gerenciar modelos de aprendizado de máquina em produção é complexo. Especialmente porque quem desenvolve os modelos não necessariamente tem as habilidades para colocá-los em produção e monitorá-los.

No processo de entrega de soluções reais de aprendizado de máquina, a criação de modelos que tem bom desempenho é apenas uma parte do processo (BOSE; AGGARWAL, 2020). Treinamento e implantação (*deploy*) de modelos enfrentam desafios relacionados, porém distintos. Por exemplo, problemas ultra-dimensionais demandam um grande volume de dados (Big Data) e a necessidade de modelos cada vez maiores colocando grande pressão em métodos/algoritmos de ML para escalarem a diversas máquinas (distribuído) (XING *et al.*, 2015). Isso traz complexidade a nível de gerenciamento de dados, métodos de treinamento e gerenciamento de modelos. A complexidade de gerenciamento do ciclo de vida de ML se evidencia ao se estudar a necessidade diversas empresas, de portes e segmentos distintos, quanto à padronização o fluxo de entrega e operação de seus modelos.

No caso de desenvolvimento de código, existe um processo consolidado eficiente de desenvolvimento e *deployment*: Development and Operations (DevOps) (BASS; WEBER; ZHU, 2015). Este processo consiste basicamente de três etapas de desenvolvimento (*dev*): implementação (*build*), teste (*test*), entrega (*release*) e duas etapas de operação (*ops*): monitoramento (*monitoring*) e planejamento (*planning*). Com isso, assegura-se a confiabilidade, escalabilidade, segurança e velocidade nas aplicações (AWS, 2021) mantendo a flexibilidade de cada equipe e empresa.

Tratando-se de ML, várias empresas falham em se beneficiar de um processo eficiente com reprodutibilidade de treinamento, rastreio de resultados, escalabilidade e inferência rápida.

1.1 Motivação

Assim como em DevOps, *Machine Learning Operations* (MLOps) demanda que os desenvolvedores de aplicação se integrem com os administradores de sistemas e o time de monitoramento. Para os cientista de dados, é imprescindível ter um gerenciamento de experimentos (MLflow[1] ou Polyaxon[2], por exemplo) para rastreamento dos parâmetros utilizados nos modelos e seus resultados. Para os engenheiros de dados e time de infraestrutura, por outro lado, zela-se por configurar e gerenciar os ambientes de forma reprodutível, confiável e escalável.

Este trabalho motiva-se em aplicar os princípios e boas práticas de DevOps a sistemas de ML (MLOps) além de entender que as crescentes demandas de Big Data estressam os cientistas de dados a terem códigos mais eficientes e com possibilidade de serem distribuídos em diversas máquinas (*clusters*) desviando sua atenção das demandas de *machine learning*.

Visando produtividade, escalabilidade e robustez, o cientista de dados deve se preocupar em entender quais os melhores modelos a serem treinados e não em como treinar o modelo de forma eficiente. Ele deve entender quais transformações fazer nos dados, mas não utilizar horas de trabalho mapeando os melhores valores. Esses processos deveriam ser automáticos. De igual forma, do ponto de vista dos engenheiros e time de infraestrutura, o processo de *deployment* deve ser automático e reprodutível, bem como com métricas definidas de monitoramento.

1.2 Objetivos

Este trabalho propõe Flowi: uma plataforma para desenvolvimento e gerenciamento de modelos de aprendizado de máquina baseada em componentes para capacitar cientistas de dados a aplicar seus conhecimentos aos modelos sem se preocupar com escalabilidade, gerenciamento de experimento, *deployment*, monitoramento e/ou otimização de parâmetros pois todos são aplicados e gerenciados por padrão.

Flowi é implementado utilizando Kubernetes[3] e Dask (ROCKLIN, 2015), além de diversas outras tecnologias atuais da indústria visando automatizar a infraestrutura bem como a otimizar os recursos computacionais.

1.3 Estrutura do Trabalho

Este trabalho está estruturado da seguinte forma:

- No capítulo MLOps explica-se o conceito de MLOps, quais seus desafios, sua relação com a solução proposta bem como soluções comerciais;

[1] <https://mlflow.org/>, acessado em 14/10/2022
[2] <https://polyaxon.com/>, acessado em 14/10/2022
[3] <https://kubernetes.io/>, acessado em 14/10/2022

- No capítulo Arquitetura Proposta é apresentada a arquitetura proposta ressaltando a escolha de cada tecnologia implementada e como elas resolvem as necessidades de MLOps e facilitam o trabalho dos cientistas de dados;
- No capítulo Resultados são apresentados os resultados (performance) do sistema e discute-se sua eficácia; Também é apresentada a implementação e uso do sistema como um todo descrevendo suas implementações e integrações; também demonstrado através de um vídeo o uso prático do sistema (estudo de caso), usando inclusive uma base de dados bem conhecida (Iris *Dataset*).
- No capítulo Conclusões e Trabalhos Futuros estão as conclusões sobre o trabalho. Discute-se sobre como o sistema proposto satisfaz os requisitos apresentados, além de propor melhorias e otimizações como trabalhos futuros.

CAPÍTULO

2

MLOPS

MLOps é um conjunto de práticas que visa realizar o *deploy* e a sustentação de modelos de aprendizado de máquina em produção de forma eficiente e confiável.

Em aplicações reais de ML é necessária a integração com diversos sistemas, a fim de monitorar, escalar e automatizar os processos. Por conta disso, o código de ML em si é somente uma fração do todo, exigindo que a parte de infraestrutura seja priorizada em conjunto.

Mas isso muitas vezes não ocorre e débitos técnicos podem ser acumulados na integração entre os diversos componentes necessários para a execução de uma aplicação completa de ML (SCULLEY *et al.*, 2014).

Este capítulo discute técnicas para implementar e automatizar fluxos de *Continuous Integration* (CI), *Continuous Delivery* (CD) e *Continuous Training* (CT) (GOOGLE, 2020).

2.1 Implementação de Machine Learning

As tarefas necessárias para o desenvolvimento de um modelo são:

1. *Data extraction*: Integrar dados de diversas fontes;
2. *Data analysis*: Entender o comportamento dos dados e vislumbrar as transformações necessárias;
3. *Data Preparation*: Preparar o dado para treino (limpeza, dividir em treino/teste/validação, categorizar, etc);
4. *Model Training*: Treinamento de diversos modelos utilizando os dados da etapa anterior com vários parêmetros diferentes;
5. *Model evaluation*: Avaliação do modelo utilizando a técnica adequada a fim de assegurar sua qualidade;

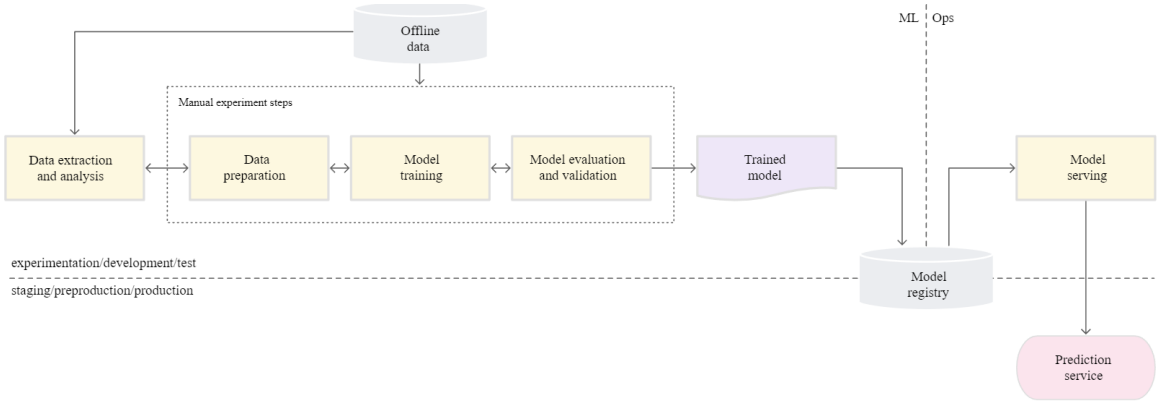
6. *Model validation*: Validação para deployment, ou seja, a performance é superior a certo valor pré-definido;

7. *Model serving*: Servir o modelo em produção como Interface de Programação de Aplicação *Application Programming Interface* (API), Batch, etc;

8. *Model monitoring*: Verificar a performance do modelo e dados em produção potencialmente invocando um novo treino quando necessário.

A maturidade do sistema pode ser definida na proporção de automatização do sistema, ou seja, quanto mais automatizado, mais maduro. A Figura 1 mostra a arquitetura manual capaz de implementar e entregar os modelos de ML em produção. A partir disso, cada equipe ou empresa pode adicionar camadas de automação e segurança conforme suas necessidades.

Figura 1 – Arquitetura manual capaz de implementar e entregar os modelos de ML em produção.



Fonte: Google (2020).

2.2 Desafios

Segundo Zaharia (ZAHARIA *et al.*, 2018), os maiores desafios no ciclo de vida de ML são: rastreio de experimentos, reprodutibilidade e operacionalização do processo de *deploy*. Estas etapas permeiam todo o ciclo de vida e aumentam significativamente a qualidade de entrega se trabalhados corretamente. Entretanto, existem diversos outras etapas a serem otimizadas.

2.2.1 Rastreio de experimentos

Esta implementação permite que cada experimento seja catalogado e rastreado possibilitando a comparação entre eles e entendimento de quais parâmetros aperfeiçoam o modelo e quais degradam. Além disso, o cientista de dados tem mais tranquilidade em desenvolver sabendo que seus testes serão gravados e bem estruturados.

2.2.2 Reprodutibilidade

É imprescindível que seja possível retreinar o modelo e ter o mesmo resultado em termos estruturais e semelhantes em termos de valor/pesos (uma vez que os processos estatísticos podem atribuir valores aleatórios durante o treino). Isso ocorre se e somente se os dados de treino forem os mesmos, bem como as transformações e seus parâmetros. Assim, para que a reprodutibilidade seja possível é necessário investir em um bom rastreio de experimentos anteriormente.

Um segundo aspecto da reprodutibilidade é a aplicação de melhores práticas de engenharia de *software*: modularização e reúso. A criação manual de diversos trechos de códigos em *jupyter notebooks*[1], apesar de boa para pequenos testes, é ineficaz para a produtização de modelos. Assim, bibliotecas e métodos bem definidos aceleram os testes (mesmo em *jupyter notebooks*) e possibilitam que as mesmas transformações realizadas nos experimentos sejam realizadas no treino e na inferência em produção.

2.2.3 Operacionalização do processo de deploy

O processo de *deploy* consiste em um *pipeline* automatizado para *test*, *build* e o *deployment* da aplicação em ambiente *dev*, *quality assurance* ou produtivo. Geralmente este processo é iniciado a partir de um *commit* ou *pull request* no código do projeto, mas existem outras formas de se iniciar o projeto. No caso do Flowi, por exemplo, não é necessário um novo código e sim um novo treino/*flow*.

O processo de deploy varia dependendo da forma de consumo do modelo. As duas formas mais comuns para ML são: API *Representational State Transfer* (Transferência Representacional de Estado) (API REST) e processamento *Batch* (*offline*). O primeiro é focado em predição em tempo real com baixa latência enquanto o segundo é focado em processamento de grandes volumes de dados com alto *throughput*.

Ambos os casos podem ser automatizados com ajuda de aplicações como Jenkins[2], Travis[3] ou alguma *pipeline* para construção de *container*, mas vale ressaltar que o processo é diferente mesmo utilizando as mesmas ferramentas. Enquanto para a API é necessário disponibilizar um *endpoint* e ter um modelo especializado em responder uma requisição ou *micro-batch* por vez rapidamente, para o *Batch*, o processamento deverá ser distribuído e capaz de realizar dezenas de milhares senão milhões de inferências por vez.

Assim, não só o *pipeline* de *deploy*, bem como a aplicação/modelo, devem ter arquiteturas otimizadas para esses fins (alto *throughput* ou baixa latência).

[1] <https://jupyter.org/>, acessado em 14/10/2022
[2] <https://www.jenkins.io/>, acessado em 14/10/2022
[3] <https://travis-ci.org/>, acessado em 14/10/2022

2.2.4 Outras otimizações

Além das principais melhorias relatadas, destacam-se as seguintes etapas:

2.2.4.1 Treino

O treino dos modelos deve ser flexível o suficiente para que o cientista de dados tenha liberdade para experimentar, testar transformações e entender o comportado dos dados e modelos ao passo que deve ser estruturado o suficiente para que os resultados e artefatos gerados tenham sempre a mesma estrutura e comportamento. Assim, é possível automatizar as demais etapas como aferição de métricas e *deployment* do modelo e transformações de *inputs* e *outputs*.

2.2.4.2 Hyper Parameter Tunning

Cada modelo de ML pode ser otimizado alterando seus hiper-parâmetros. Por exemplo, em uma estrutura de redes neurais artificiais, a quantidade de neurônios em cada camada é um hiper-parâmetro a ser otimizado. Outros exemplos são: quantidade de épocas/iterações de treino, *batch size*, otimizador, taxa de *dropout*, etc.

Um sistema capaz de trazer tais otimizações de forma transparente para o cientista de dados traz maturidade para o desenvolvimento, pois o cientista se preocupa com a lógica dos algoritmos e não com a operação deles. Ademais, os modelos treinados já estarão otimizados e terão melhor performance.

2.2.4.3 Automação de testes

Assim como todo novo código, o modelo deve ser medido e avaliado. De acordo com Jordan (JORDAN, 2017), existem três perguntas fundamentais a serem respondidas ao se medir a performance de um modelo:

- O modelo é útil? Ou seja, de acordo com o problema em questão os resultados são bons?
- Ao se adicionar mais dados de treinamento a performance do modelo aumenta?
- Ao se adicionar mais *features* a performance do modelo aumenta?

Apesar de não existir uma única forma de apresentar o melhor desempenho para todos os problemas, (REZENDE; MONARD; CARVALHO, 1999) e (RASCHKA, 2018) explicam detalhadamente em suas obras diversos algoritmos para medir o desempenho e seleção dos modelos.

2.2.4.3.1 Métodos de divisão do *dataset*

O método ***Holdout*** divide o conjunto de dados em treino e teste aleatoriamente, usualmente em 70% e 30%, respectivamente. Este método é simples e garante que os dados de treino

não são utilizados na avaliação do modelo. Isso é importante para garantir que o modelo está generalizando (conseguindo predizer valores desconhecidos) corretamente e não "decorando"as respostas. Entretanto, o método *holdout* não mantém a distribuição do *dataset* em suas divisões influenciando a análise das métricas inferidas sobre o *dataset* de teste.

Stratification é o método que assegura que cada amostragem (*subsample*) do *dataset* segue a distribuição original do *dataset*. Ou seja, se as classes tinham proporção de 30% e 70%, cada amostragem do *dataset* seguirá com a mesma proporção.

O ***Cross-Validation*** divide igualmente o *dataset* aleatoriamente em k partições mutuamente exclusivas chamadas *folds*. Em seguida, k-1 partições são utilizadas para treinamento enquanto a remanescente é utilizada para teste. Esse processo se repete k vezes até que todas as partições sejam utilizadas para teste e treinamento. Esse método traz uma grande vantagem, não é necessário escolher cuidadosamente em qual conjunto de dados treinar e em qual conjunto de dados testar. Todo o *dataset* é utilizado. Assim, é possível perceber a variabilidade das predições dos modelos e assegurar que as otimizações no treinamento são relevantes para todo o conjunto de dados, não enviasada por uma amostragem.

2.2.4.3.2 Métricas de avaliação

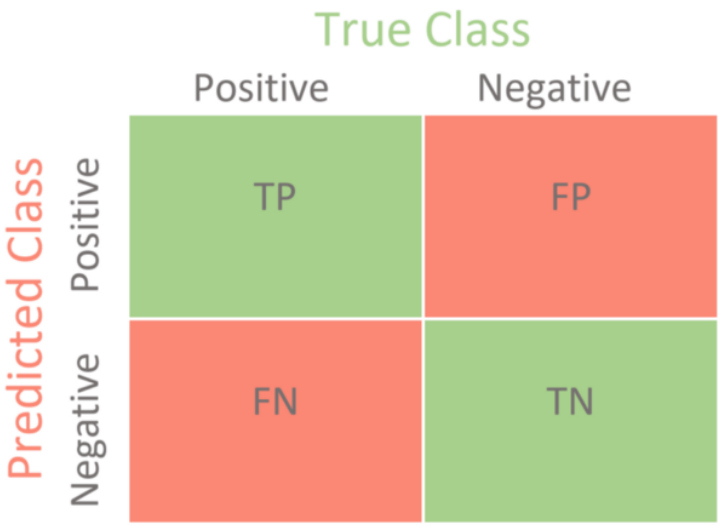
Matriz de Confusão (*Confusion Matrix*) oferece uma medida efetiva do modelo de classificação, ao mostrar as seguintes métricas (Figura 2):

- **Verdadeiro Positivo (*True Positive* - TP)**: Quando a predição diz que uma observação pertence a uma classe e ela realmente pertence à classe predita;
- **Verdadeiro Negativo (*True Negative* - TN)**: Quando a predição diz que uma observação não pertence a uma classe e ela realmente não pertence à classe;
- **Falso Positivo (*False Positive* - FP)**: Quando a predição diz que uma observação pertence a uma classe, mas ela não pertence à classe predita;
- **Falso Negativo (*False Negative* - FN)**: Quando a predição diz que uma observação não pertence a uma classe, mas ela pertence à classe;

A partir dessas métricas é possível derivar:

- **Acurácia**: Razão de predições corretas sobre o total de predições - (TP + TN) / (TP + TN + FP + FN)
- **Precisão**: Do total de predições de classe Positivo, quantas estão corretas (TP / (TP + FP));
- **Sensibilidade (*recall*)**: Do total de classe Positivo como valor esperado, quantas estão corretas (TP / (TP + FN));

Figura 2 – Matriz de confusão para classificação binária.



Fonte: Mohajon (2020).

Um caso de uso onde as classes não são bem distribuídas (99% das observações pertencem a classe A e 1% à classe B) exemplifica o uso das métricas acima. Caso o classificador aprenda a classificar todas as observações como pertencentes a classe A, ele terá acurácia de 99%, entretanto o classificador não aprendeu como identificar nenhum padrão. Olhando para a sensibilidade e a predição, logo se nota que existe algo de errado com o modelo predizendo a classe B. A sensibilidade garante que a classe B não é ignorada enquanto a precisão garante que não são muitas classificações erradas da classe A como classe B.

Por fim, em geral é desejado ter apenas uma métrica para avaliar um modelo/problema. O F-score pondera os valores de precisão e sensibilidade de acordo com a equação 2.1. Onde o parâmetro β permite controlar a importância da precisão e *recall*. $\beta < 1$ aumenta a importância da precisão enquanto $\beta > 1$ aumenta a importância do *recall*.

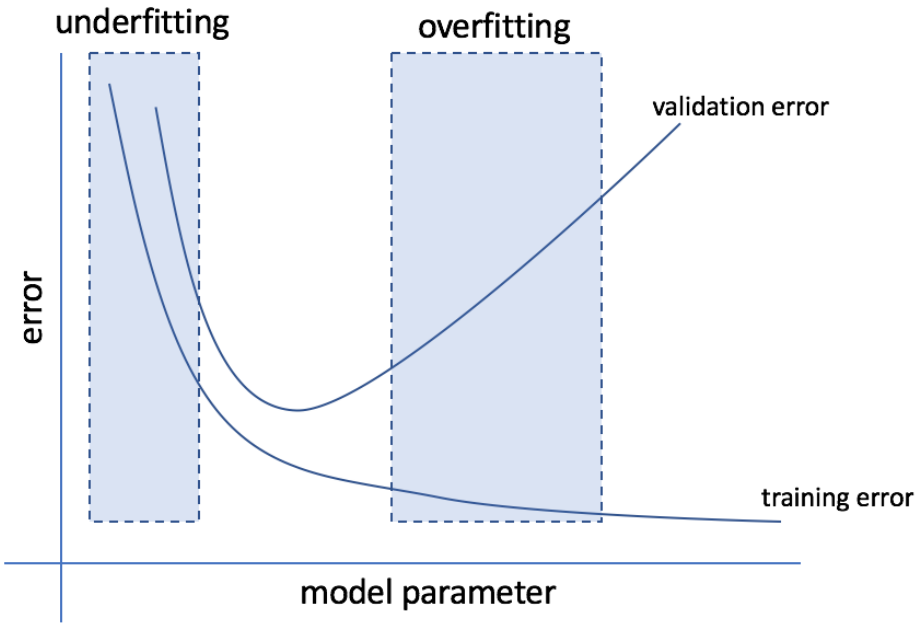
$$F_\beta = (1+\beta^2)\frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{2.1}$$

Para o caso de regressão, o erro (valor esperado contra valor predito) é a medida base. Desta pode-se extrair métricas como Erro médio quadrático (*Mean Squared Error* - MSE), erro absoluto (*absolute error*), R^2.

Validation curves (Figura 3) é uma forma de visualmente encontrar o modelo com a melhor generalização. Usualmente, o gráfico é composto do erro (eixo-y) contra algum hiperparâmetro (eixo-x) que controla a tendência do modelo realizar *overfitting* ou *underfitting*, como o grau de um polinômio. A área de *underfitting* na Figura 3 é onde o modelo não consegue aprender a real relação entre o dado de entrada e a saída esperada. Nesse caso, o modelo tem alto

bias. A região de *overfitting* indica que o modelo "decorou"as respostas esperadas no *dataset* de treino, mas isso não condiz com os valores de teste, tendo alta variância. A região intermediária é a ideal onde a generalização do modelo ocorre.

Figura 3 – Exemplo de curvas de validação para encontrar o modelo com melhor generalização.



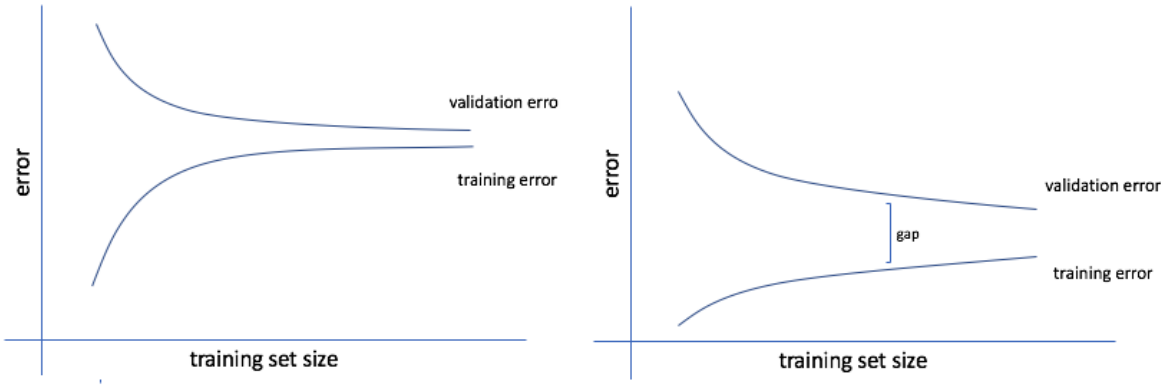
Fonte: Jordan (2017).

Por fim, as curvas de aprendizado (*learning curves*) (Figura 4) ilustram o aprendizado do modelo (erro - eixo-y) em relação a quantidade de observações de treino (eixo-y). Caso o modelo tenha alto viés (*bias*), as curvas de aprendizado convergem rapidamente, ou seja, adicionar instâncias de treino ao modelo não melhoram seu desempenho. Quando o modelo tem alta variância, existe um intervalo (*gap*) entre as duas curvas indicando que o modelo pode estar com *overfit*. Nesse caso, adicionar mais instâncias de treinamento deve melhorar o desempenho do modelo.

2.2.4.4 *Deploy de transformações*

Como visto em Implementação de Machine Learning (seção 2.1), uma etapa do processo de desenvolvimento de modelos é o *data preparation*. Nesta etapa ocorrem transformações nos dados que também devem ser realizadas em produção antes da predição do modelo ou depois. (BROWNLEE, 2020) destaca a importância deste processo e é razoável assumir que sempre

Figura 4 – Exemplo de curvas de aprendizado para ilustrar como a quantidade de observações de treino influenciam no aprendizado do modelo.



Fonte: Jordan (2017).

haverá algum tipo de preparação no dado, como *standard scaling*, que deve ser aplicado antes da inferência do modelo.

Por outro lado, algumas transformações, como a volta da categorização das classes preditas, devem ser feitas depois da inferência. No caso do *dataset* Iris[4] - conjunto de medidas das pétalas e sépalas de três espécies de flores: Setosa, Versicolor e Virginica (CHAUHAN, 2021), as classes Setosa, Versicolor e Virginica devem ser categorizadas numericamente para o treino, mas precisam voltar ao nome original para a aplicação final após a predição).

Assim, essas transformações devem ser gerenciadas juntamente com os modelos pois andam lado-a-lado e a falha de um implica a falha do outro.

2.2.4.5 Auto Scaling

Uma vez que as aplicações estão em produção é necessário manter seu desempenho (ex. tempo de resposta) ao mesmo tempo que se controla seu custo. Para isso, aplica-se o *auto scaling* (auto escalonamento) baseado em alguma métrica (ex. tempo de resposta, quantidade de requisição, etc). O escalonamento pode ser para cima, ou seja, aumentar a quantidade de réplicas da aplicação em caso de maior demanda para se assegurar o desempenho esperado ou para baixo em caso de baixa demanda, minimizando os custos.

Ao se tratar com APIs REST é razoável balancear a quantidade de réplicas da aplicação baseado em carga (ex. quantidade de *Virtual CPU* ou *virtual processor* (vCPU) utilizada) ou requisições. Enquanto para *Batch* o escalonamento visa atender um tempo médio de processamento baseado na quantidade de registros a serem preditos, em geral, a aplicação não precisa escalar durante o processamento, mas pode ser necessário caso alguma etapa interna demande mais recursos. Por exemplo, durante o desenvolvimento do modelo ocorre o pré-processamento

[4] <https://archive.ics.uci.edu/ml/datasets/Iris>, acessado em 14/10/2022

e o treino. A etapa de pré-processamento demanda mais vCPU enquanto o treino demanda mais *Graphical Processing Unit (also GP-GPU General Purpose GPUs)* (GPU). É interessante que o sistema saiba lidar com as diferentes demandas e condicione a infraestrutura para atender os requisitos do sistema em horas específicas.

2.2.4.6 Monitoramento

Como o "Ops"de MLOps significa operações, evidentemente deve existir algum tipo de otimização ou melhoria no processo após a implantação do modelo. O *Auto Scaling* beneficia-se disso, mas existem outras necessidades de monitoramento, como:

1. *Model Drift*: *Drift* ocorre quando a distribuição dos dados de entrada do modelo mudam.
2. *Model Performance*: Acompanhamento das métricas do modelo como acurácia, precisão, *recall* e *f1-score*.
3. *Model Outliers*: é a identificação de casos raros que podem ter comportamento suspeito por se diferenciar significativamente da maioria dos dados
4. *Data Quality*: significa assegurar a cardinalidade e tipo dos dados além de identificar dados faltantes e outras alterações nos dados de entrada.
5. ML *Observability*: Acompanhar os modelos a nível de aplicação e assegurar seu tempo de resposta, disponibilidade e identificar erros e *bugs* pela análise de *logs*.

2.2.4.7 Retreino

Muitos casos de uso tem como característica a adição de novos dados ou a alteração deles de tempos em tempos. Por exemplo, aplicações de *streaming* como Spotify[5], Youtube[6] ou Netflix[7] precisam retreinar seus modelos baseados nos novos gostos e interações dos usuários com o sistema. Assim, é fundamental para esses serviços aplicar treinos recorrentes utilizando os novos dados da plataforma, mas não necessariamente alterando o código de treino.

2.3 Aplicações comerciais

Diante desses desafios diversas empresas decidiram criar seus próprios sistemas de MLOps. Por exemplo: Kubeflow[8] dedica-se a tornar o *deploy* de workflows de ML em Kubernetes simples e escaláveis (BISONG, 2019); *Amazon Web Services* (AWS) SageMaker[9] é um serviço

[5] <https://www.spotify.com/us/>, acessado em 14/10/2022
[6] <https://www.youtube.com>, acessado em 14/10/2022
[7] <https://netflix.com/>, acessado em 14/10/2022
[8] <https://www.kubeflow.org/>, acessado em 14/10/2022
[9] <https://aws.amazon.com/sagemaker/>, acessado em 14/10/2022

de ML para criar, treinar e implantar modelos de ML para praticamente todos os casos de uso. Esses dois serviços agem como *frameworks* para suprimir a complexidade de gerenciamento de infraestrutura dando ao desenvolvedor a liberdade para escolher como implementar e treinar seus modelos. Entretanto, é dever do desenvolvedor utilizar o *framework* da melhor forma possível configurando o tamanho correto de máquina, estimando o uso de memória e CPU, etc. Por isso, algumas companhias como Uber[10] escolheram criar sistemas com esse tipo de solução por padrão (HERMANN; BALSO, 2017).

Por outro lado, algumas companhias focaram em provisionar serviços que, dado um conjunto de dados de treino com as *labels* (resultado esperado), fazem a seleção de *features*, treinamento e disponibilização do modelo de forma transparente. Lobe[11], Teachable Machines[12] e Clarifai[13] são algumas opções de plataformas *no-code* que removem os cientistas de dados e todo o time de TI da equação, possibilitando que o time de negócios teste e produtize suas ideias. É válido pontuar, no entanto, que existem grandes riscos desta abordagem como explicado por Abbasi e Ahmad (ABBASI; AHMAD, 2019) no Harvard Business Review. A completa automação do processo diminui drasticamente a compreensão dos processos e análise dos resultados, uma vez que as etapas de preparação, contextualização e representação dos dados foram realizadas de forma transparente. Além disso, é válido frisar que a ética é (ou deveria ser) aplicada em todas as soluções de *machine learning*. Isso se tornado um assunto cada vez mais relevante (CHEN *et al.*, 2021).

2.4 Considerações Finais

Neste capítulo, o conceito de MLOps foi apresentado, descrevendo suas etapas, correlações com DevOps e desafios. Além disso, mostrou-se como algumas empresas olham para os desafios de se operacionalizar e padronizar o processo de desenvolvimento e gerenciamento de modelos de aprendizado de máquina criando suas próprias plataformas de MLOps, sejam elas par auso interno ou comerciais. O próximo capítulo apresenta a arquitetura do Flowi, como ela implementa as etapas de MLOPs e endereça seus desafios.

[10] <https://www.uber.com/br/en/>, acessado em 14/10/2022
[11] <https://lobe.ai/>, acessado em 14/10/2022
[12] <https://teachablemachine.withgoogle.com/>, acessado em 14/10/2022
[13] <https://www.clarifai.com/>, acessado em 14/10/2022

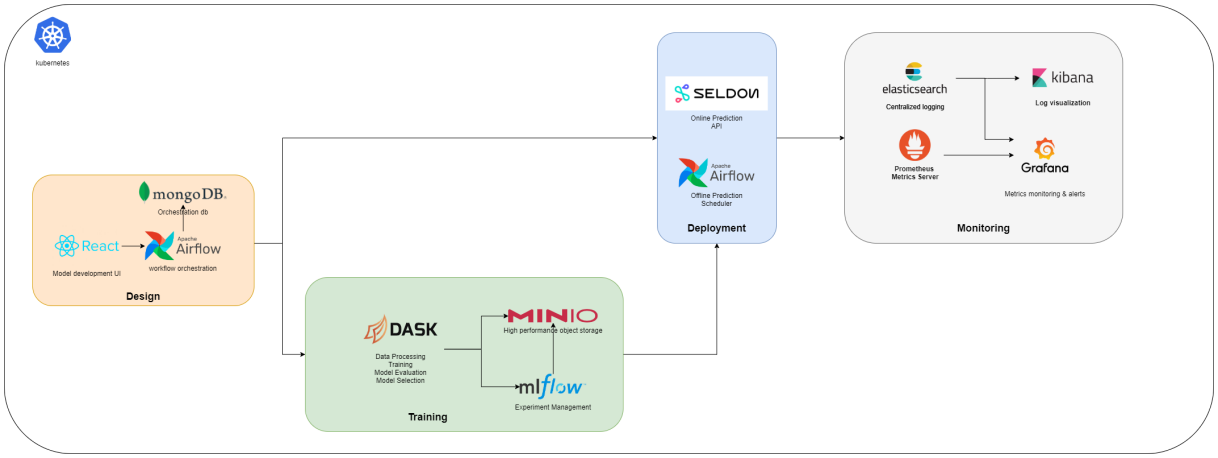
CAPÍTULO

3

ARQUITETURA PROPOSTA

Flowi é uma plataforma baseada em componentes para desenvolvimento e gerenciamento de modelos de aprendizado de máquina seguindo as diretrizes de MLOps (capítulo MLOps). Este capítulo visa apresentar a arquitetura do Flowi segundo a Figura 5 e discutir como os desafios apresentados anteriormente foram abordados e resolvidos. Vale ressaltar, no entanto, que melhorias e novas implementações já estão previstas e abordadas no capítulo Conclusões e Trabalhos Futuros.

Figura 5 – Arquitetura do Flowi implementada sobre o Kubernetes visando suprir as necessidades e filosofias de MLOps.



Fonte: Elaborada pelo autor.

O sistema é baseado em Kubernetes[1], sistema *open-source* para automação de *deployment*, escalonamento e gerenciamento de aplicações conteinerizadas. Assim, as aplicações do Flowi e sistemas correlatos são gerenciados de forma nativa por padrão e não é necessária a preocupação com o gerenciamento do sistema uma vez em produção. Semelhantemente, padroniza-se

[1] <https://kubernetes.io/>, acessado em 14/10/2022

a forma de *deployment* utilizando os arquivos de configuração do Kubernetes, facilitando o processo de *Continous Delivery* (CD).

Vale ressaltar também que diversos sistemas utilizados como MongoDB[2], Dask[3] e Seldon[4] já tem documentações de *setup* muito bem definidas para Kubernetes. O Seldon, na verdade, necessita do Kubernetes pois sua implementação é feita sobre *custom resources*[5] do Kubernetes.

Por fim, o *cluster* Kubernetes pode ser implementado *on-premise* ou na nuvem (AWS[6], *Google Cloud Platform* (GCP)[7] e *Microsoft Azure Cloud services* (Azure)[8] tem serviços de provisionamento do *cluster*) e permite a instalação de outros serviços não ilustrados na Figura 5 como:

- Istio[9]: *service mesh* para gerenciamento de rede dentro do Kubernetes
- Knative[10]: plataforma baseada em Kubernetes para *deploy* e gerenciamento de *serverless workloads*
- Keda[11]: *Kubernetes Event-driven Autoscaling*

3.1 Plataforma baseada em componentes

Uma das maiores dificuldades dos cientistas de dados é a reprodutibilidade, ou seja, ser capaz de reproduzir os experimentos diversas vezes e ter o mesmo resultado (ou semelhante, tendo em vistas variáveis aleatórios dos processos estatísticos). Além disso, uma vez que os testes foram feitos, o modelo e as transformações devem ser exatamente reprodutíveis em produção.

O Flowi se propõe a resolver esse problema de duas formas: uma Interface de Usuário - *User Interface* (UI) web[12] para criação do fluxo (*flow*) de transformações e pacote python Flowi[13], ambos implementados pelo autor.

Além disso, gerenciar diversos experimentos é uma tarefa complexa. Diversas vezes o desenvolvedor necessita de trechos de códigos que utilizou em outros projetos, deve alterar

[2] <https://www.mongodb.com/kubernetes>, acessado em 14/10/2022
[3] <https://kubernetes.dask.org/en/latest/>, acessado em 14/10/2022
[4] <https://docs.seldon.io/projects/seldon-core/en/latest/workflow/install.html>, acessado em 14/10/2022
[5] <https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/>
[6] <https://aws.amazon.com/>, acessado em 14/10/2022
[7] <https://cloud.google.com/>, acessado em 14/10/2022
[8] <https://azure.microsoft.com/en-us/>, acessado em 14/10/2022
[9] <https://istio.io/>, acessado em 14/10/2022
[10] <https://knative.dev/>, acessado em 14/10/2022
[11] <https://keda.sh/>, acessado em 14/10/2022
[12] <https://hub.docker.com/repository/docker/psilvaleo/flowi-ui>, acessado em 14/10/2022
[13] <https://pypi.org/project/flowi/>, acessado em 14/10/2022

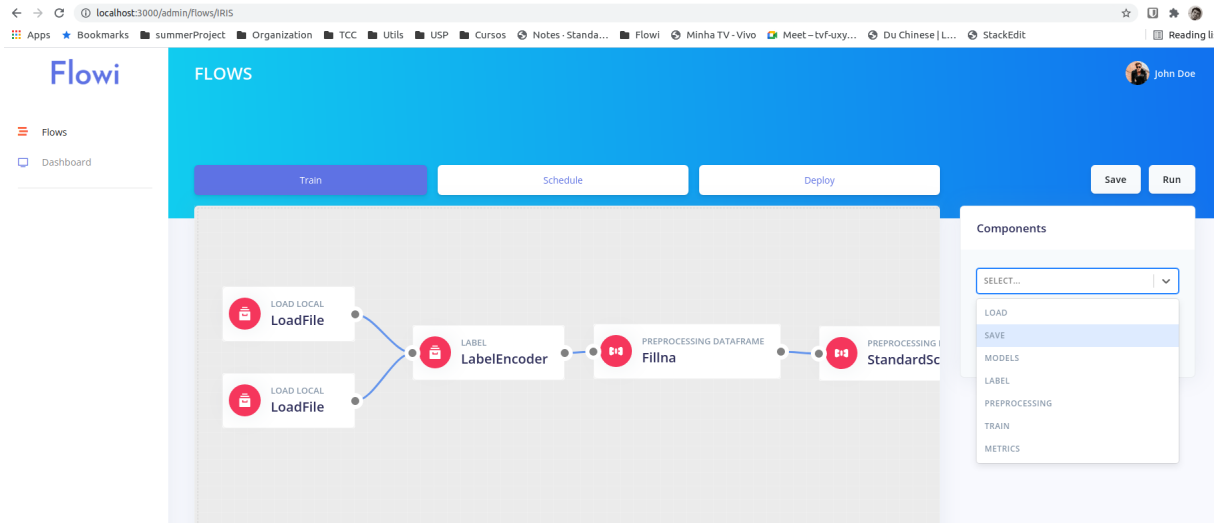
variáveis em diversos lugares para fazer uma alteração simples e alterar os *logs* nos sistemas de rastreio (ex. MLflow), caso use.

Por essa razão, o Flowi foi implementado utilizando-se do conceito de Directed Acyclic Graph (Grafo Acíclico Dirigido) (DAG) para que o cientista de dados ou desenvolvedor possa ver claramente o fluxo de dados e transformações, para que não haja nenhum *loop* que bloquei o processamento.

3.1.1 Web UI

A interface web do Flowi permite a criação de fluxos de pré-processamento e treinamento de modelos utilizando componentes pré-definidos de acordo com a Figura 6. Isso permite a rápida criação dos modelos além de assegurar a reprodutibilidade, uma vez que todos os componentes tem valores padrão (*default*) para cada variável.

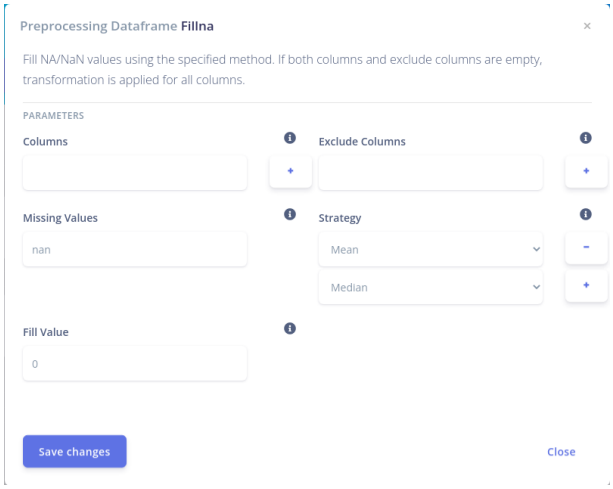
Figura 6 – Flowi UI baseada em componentes para facilitar a criação de fluxos de processamento de dados, treinamento de modelos e *deploy*.



Fonte: Elaborada pelo autor.

A Figura 7 mostra a configuração do componente Fillna. Ele é responsável por preencher valores nulos com algum método (valor constante, média, mediana, etc). Os sinais de "+"e -"permitem que mais de um método seja aplicado ao componente, isso gera dois experimentos distintos, cada qual usando seu método. Assim, a produtividade do desenvolvedor é elevada e permite testes mais seguros, pois altera-se apenas as variáveis necessárias, sem a necessidade de alteração de diversos trechos de código.

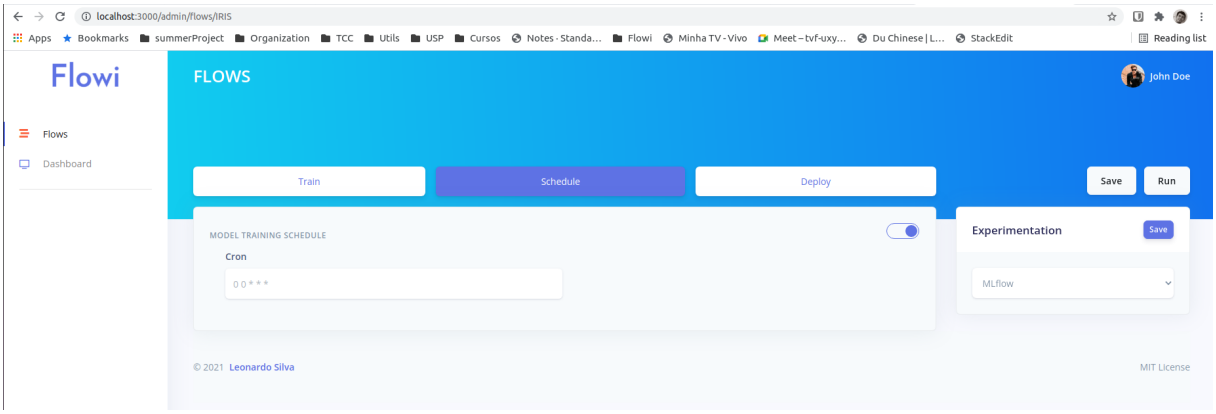
Figura 7 – Configuração do componente Fillna utilizando a UI.



Fonte: Elaborada pelo autor.

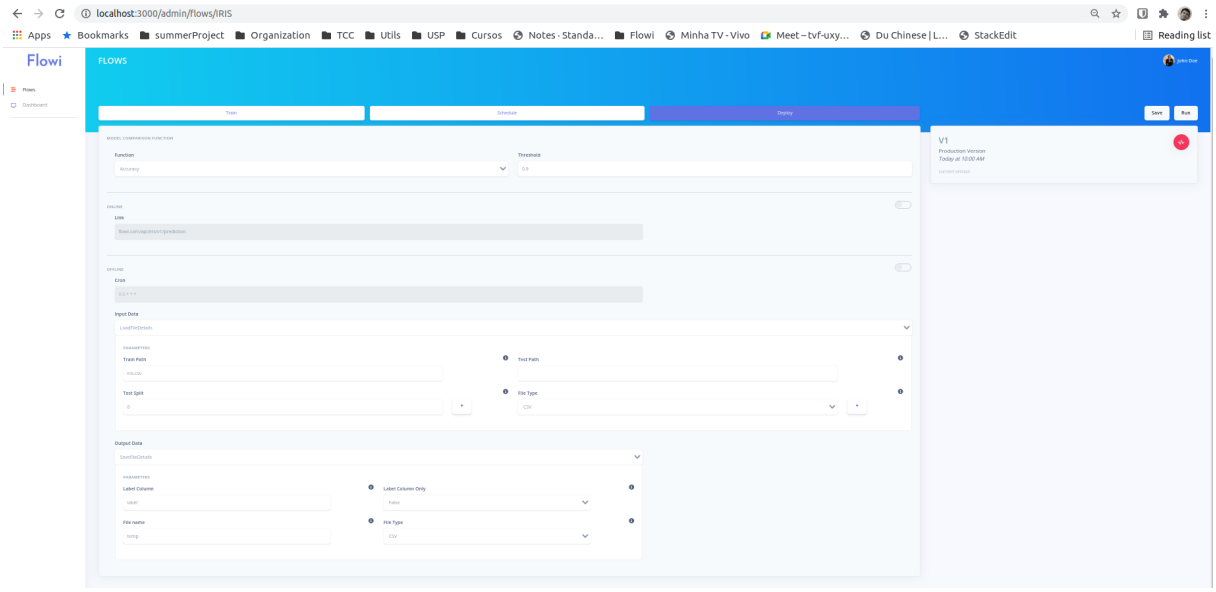
Duas outras funções estão disponíveis na interface: agendamento para treinamentos recorrentes (*schedule*) e forma de *deploy* do modelo (*deploy*). A Figura 8 mostra a configuração de treinamento recorrente utilizando *cron* para a data e permitindo habilitar ou desabilitar pelo *toggle button*. Outra configuração deste tela é a possibilidade de selecionar qual gerenciador de experimentos deve ser utilizado (MLflow ou Polyaxon) - apenas MLflow foi implementado por enquanto. De modo análogo, a Figura 9 ilustra como se configura o *deploy* do modelo. Primeiro escolhe-se qual métrica deve ser considerada e qual o threshold (valor mínimo) para que o modelo seja adequado para o *deploy*. Depois, escolhe-se a forma (*Online* - API REST ou *Offline* - Batch), cada qual com suas devidas especificações. Para o *Online* não é necessário nenhum passo adicional, a UI apenas mostra qual o *endpoint* para consumo. O *Batch* demanda o valor *cron* para recorrência, qual o dado de entrada (*input data*) e qual o dado de saída (*output data*). Para ambos os casos, é possível habilitar ou desabilitar a funcionalidade utilizando o *toggle button*.

Figura 8 – Flowi UI para agendamento para treinamentos recorrentes utilizando cron para data e permitinfo habilitar ou desabilitar a função pelo *toggle button*.



Fonte: Elaborada pelo autor.

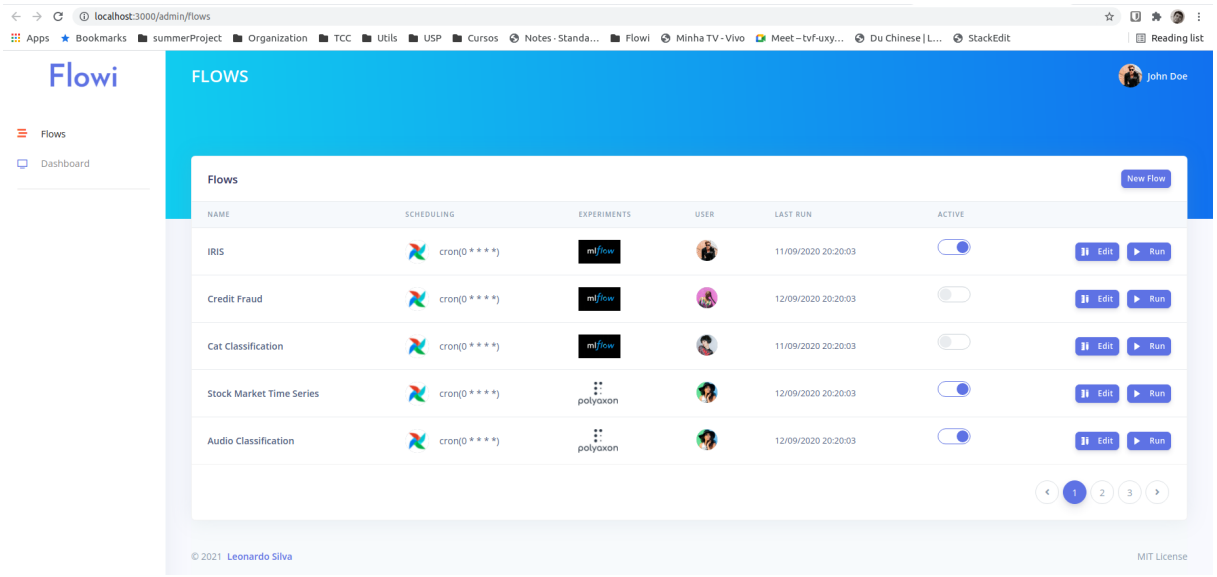
Figura 9 – Flowi UI para *deploy* dos modelos de forma online ou *batch*. A ferramenta permite a configuração de qual métrica deve ser utilizada para seleção do modelo e qual *threshold* (valor mínimo) para que o modelo seja apito a produção.



Fonte: Elaborada pelo autor.

A última tela sumariza os fluxos (Figura 10). Nota-se a presença dos aspectos principais do fluxo: *cron* do treinamento recorrente, qual o gerenciador de experimentos, usuário responsável pelo fluxo, última vez que o fluxo foi rodado, se está ativo ou não e botões para editar ou rodar o fluxo.

Figura 10 – A tela de *Flows* sumariza as principais informações de cada modelo como nome, agendamento de treino (cron), ativo ou desativo, etc.



Fonte: Elaborada pelo autor.

A interface web está diretamente ligada ao Pacote Flowi implementado em Python. O papel da interface é facilitar a criação dos fluxos explicitando os valores que serão utilizados. No final deste processo, será criado e enviado um *json*, para que o treino possa ocorrer no *backend*.

3.1.2 Pacote Flowi

O pacote/biblioteca Flowi foi implementado pensando-se em:

- Componentes: Métodos com entrada e saída padronizados;
- Execução distribuída: Computação distribuída e *out of core* (a quantidade de dados pode ser superior à memória disponível);
- Padronização de *transformers*: As transformações de *input* e *output* são padronizados em um *pipeline* do *sckikit learn*;
- Padronização de predição: O modelo gerado sempre possui um método *predict*;
- DAG (*directed acyclic graph*): O treinamento é construído utilizando DAG possibilitando o entendimento de todo o fluxo de processamento. Isso permite a adição de etapas como *hyper parameter tunning* mesmo quando não prevista no fluxo original.
- Otimização de modelos: Diversos parâmetros distintos podem ser passados por um componente. O Flowi automaticamente entende esses parâmetros e dispara diversos treinos. Ao final do processamento, o sistema seleciona o melhor e envia para *staging*.

- Rastreio de experimentos: Integração nativa com rastreio de experimentos. Qualquer gerenciador pode ser utilizado, basta implementar uma classe utilizando os métodos herdados da classe padrão.

3.2 Orquestração

Para que o processo de MLOps, ou seja, para que todo o fluxo de vida de ML ocorra, são necessárias diversas etapas. O Airflow[14] é responsável por orquestrar todas elas. Ele é uma plataforma *open-source* para programaticamente (na linguagem de programação Python[15]) criar, gerenciar e monitorar *workflows*. Com ele, é possível especificar quando um *workflow* deve rodar, tratamentos de erro, integrar com outros sistemas, visualizar os processamentos, etc. Além disso, na versão 2.0, há uma API nativa que permite interagir com as DAGs (*workflows*).

3.2.1 FlowiConfig

No caso do Flowi, a UI chama mediante a API REST nativa do Airflow a DAG *FlowiConfig* passando as configurações do novo *Flow* (fluxo/projeto/modelo) em um *payload* json (informação passada no corpo de uma requisição HTTP (MDN, 2022)). A Figura 11 mostra a tela principal do Airflow e nela vemos as DAGs FlowiTrainIRIS e FlowiBatchIRIS que foram criadas especificamente para o projeto IRIS (Dataset IRIS). As informações necessárias no *payload* são:

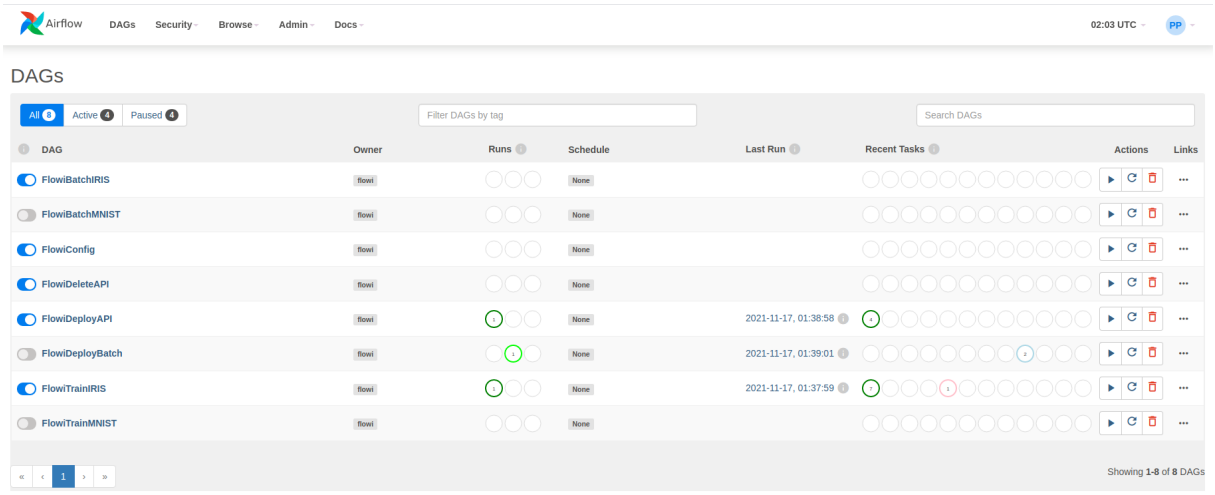
- *flow_name*: Nome do *Flow*
- *schedule_interval*: *Cron* para o agendamento do treino recorrente. "None"para manual.
- *version*: Versão do *Flow*
- *experiment_tracking*: Qual gerenciador de experimentos deve ser utilizado (ex. MLFlow)
- *deploy*: Informações sobre quais *deploys* serão feitos e como.
  - api: Para *deploy* por api
    - *enabled*: Se houver *deploy* por api, deve ser preenchido com *"true"*
  - *batch*: Para *deploy* por api
    - *enabled*: Se houver *deploy* por api, deve ser preenchido com *"true"*
    - *schedule_interval*: *Cron* para o agendamento da predição por *batch* recorrente. *"None"* para manual
    - *source*: Descrição do "*node*"especificando a origem do dado. Nele deve conter o nome, classe e parâmetros como no *flow_chart*

[14] <https://airflow.apache.org/>, acessado em 14/10/2022
[15] <https://www.python.org/>, acessado em 14/10/2022

    - ∗ *destiny*: Descrição do "*node*"especificando a saída do dado. Nele deve conter o nome, classe e parâmetros como no *flow_chart*

- *flow_chart*: contem as informações relativas ao pré-processamento e treino do modelo
  - *links*: A ligação direcionada entre cada elemento do grafo
  - *nodes*: cada componente/método é descrito como um nó no grafo
    - ∗ *name*: Nome do método (ex. *Accuracy*)
    - ∗ *class*: Classe à qual o método faz parte (ex. *MetricsClassification*)
    - ∗ *attributes*: Nome e valor de cada parâmetro (ex. "*target_column*": "*class*")

Figura 11 – Tela principal do Airflow mostrando as DAGs FlowiTrainIRIS e FlowiBatchIRIS, dentre outras mostradas na figura, que foram criadas especificamente para o projetos IRIS.



Fonte: Elaborada pelo autor.

3.2.2 FlowiTrain

A DAG iniciada por FlowiTrain é responsável pelo treino do modelo. É criada uma DAG por projeto por meio da DAG FlowiConfig. A Figura 12 ilustra as etapas de treino:

1. *generate uuid*: Cria-se um *id* para o treino para rastreio de todos os experimentos gerados durante o treino
2. *validate chart*: valida-se se o fluxo criado na UI é uma DAG válida e não contem *loop*
3. *flowi train*: Treino de todos os modelos necessários. Por exemplo, se na UI for especificado dois modelos e um componente tiver dois valores para um parâmetro, são criados 4 experimentos. Cada experimento tem suas métricas salvas no MLFlow e em um Mongo DB[16].

16 <https://www.mongodb.com/>, acessado em 14/10/2022

4. *compare models*: O melhor modelo do treino é comparado com o modelo produtivo utilizando a função especificada na UI (ex. precisão). Se for melhor, segue para o fluxo de *update model* se não, segue para *skip update*

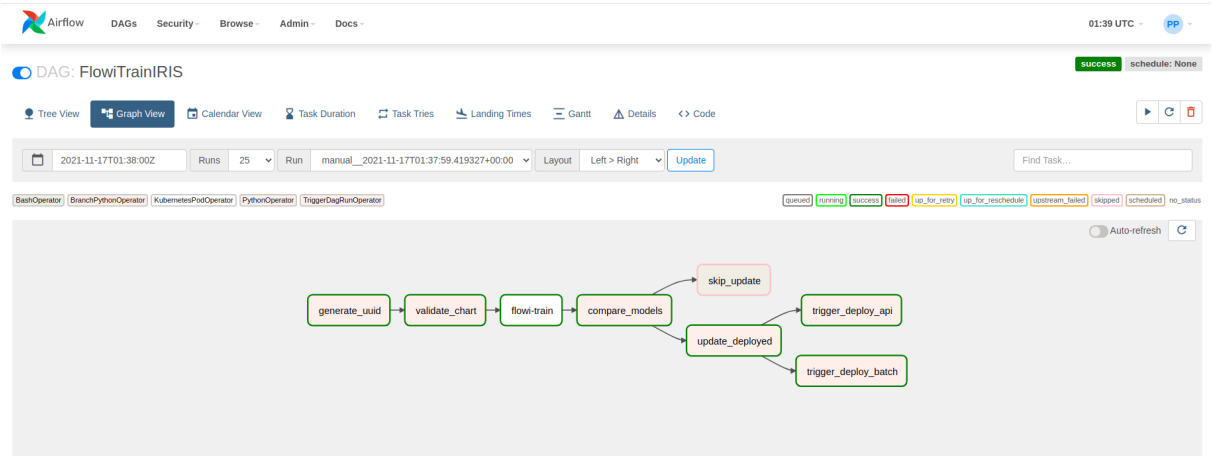
5. *skip update*: Nada a fazer, os modelos treinados não são melhores do que o produtivo e não há *deploy* do modelo

6. *update deployed*: Atualiza-se o modelo produtivo no Mongo DB

7. *trigger deploy api*: Chama a DAG FlowiDeployAPI (seção 3.2.3)

8. *trigger deploy batch*: Chama a DAG FlowiDeployBatch (seção 3.2.3)

Figura 12 – DAG FlowiTrain ilustrando as etapas para treinamento e definição de que é necessário o deploy do modelo comparando o melhor modelo do treino com o modelo produtivo.



Fonte: Elaborada pelo autor.

3.2.3 FlowiDeployAPI e FlowiDeployBatch

As DAGs FlowiDeployAPI e FlowiDeployBatch são compartilhadas entre todos os projetos, mas consomem de um arquivo de configuração interno (não visto na UI do Airflow) também criado pela DAG *FlowiConfig* com as informações necessárias para o *deploy* da API e da predição em *Batch*.

A Figura 13 ilustra as etapas de *deploy* da API:

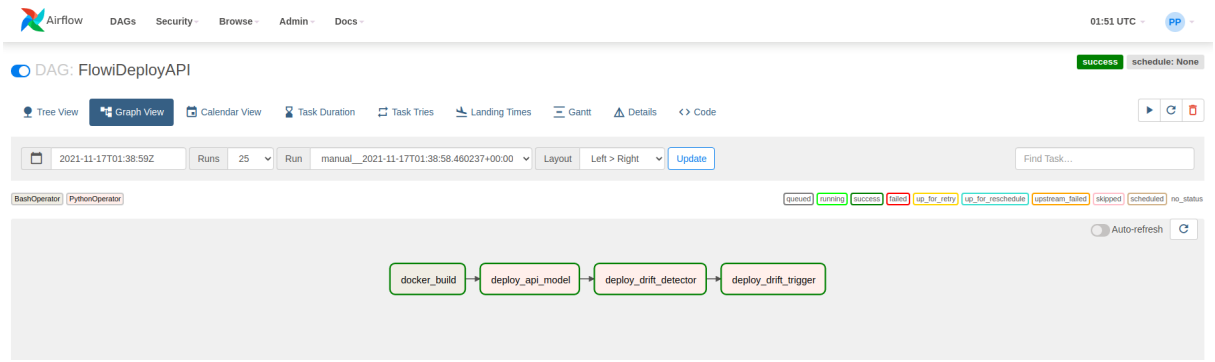
1. *docker build*: Criação do *container*

2. *deploy api model*: *Deploy* da API pelo Seldon

3. *deploy drift detector*: Todo modelo possui o *drift detector* por padrão e este é uma outra aplicação no fluxo, gerenciado pelo Seldon em conjunto com o *Knative Serving*

4. *deploy drift trigger*: *Knative eventing trigger* que notifica o *drift* detector quando há alguma requisição no modelo

Figura 13 – DAG FlowiDeployAPI no Airflow contendo as etapas para *deploy* de API.

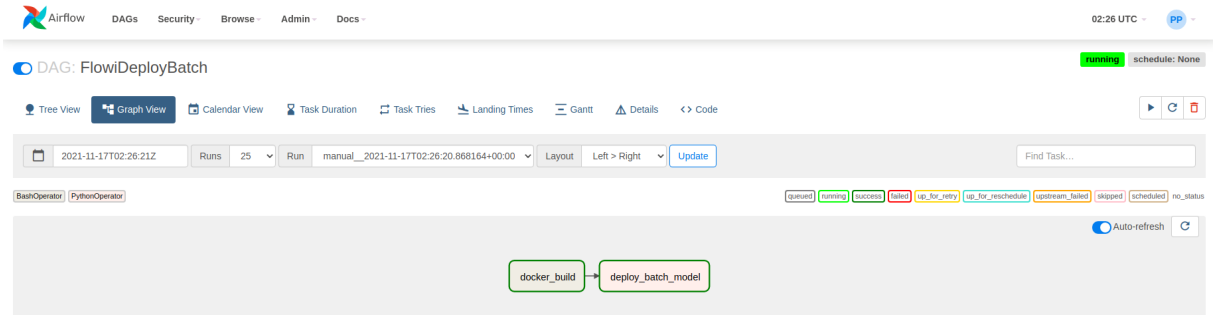


Fonte: Elaborada pelo autor.

A Figura 14 ilustra as etapas de *deploy* da Batch:

1. *docker build*: Criação do container
2. *deploy batch model*: Criação de arquivo de configuração para predição *Batch*. Não é necessário nenhum ajuste para a detecção de *drift* pois este é feito no código de predição *Batch* pela DAG (seção FlowiBatch).

Figura 14 – DAG FlowiDeployBatch no Airflow contendo as etapas para *deploy* de *Batch*.



Fonte: Elaborada pelo autor.

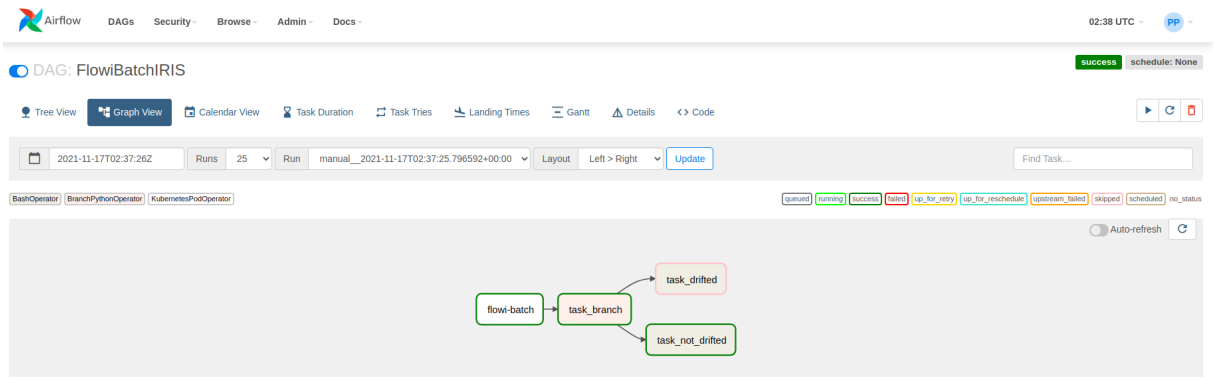
3.2.4 FlowiBatch

O processo de predição *Batch* é realizado utilizando os mesmos *transformers* da predição *online*, entretanto utiliza o processamento distribuído do Dask por trás. No caso do *Batch*, antes da predição ocorrer é realizada a detecção de *drift*, caso ele seja detectado a predição não ocorre. O resultado da predição é salva no MinIO[17] de acordo com o que foi especificado no componente da UI.

17 <https://min.io/>, acessado em 14/10/2022

A Figura 15 ilustra as etapas do processo *Batch*. Na imagem é fácil perceber a diferença no fluxo devido à presença ou não de *drift*. Assim, é possível notificar o dono do sistema para que ele tome alguma ação como retreinar o modelo.

Figura 15 – DAG FlowiBatchIRIS exemplificando o processo de predição *Batch* contendo as etapas de predição (flowi *batch*) e de detecção de *drift*.

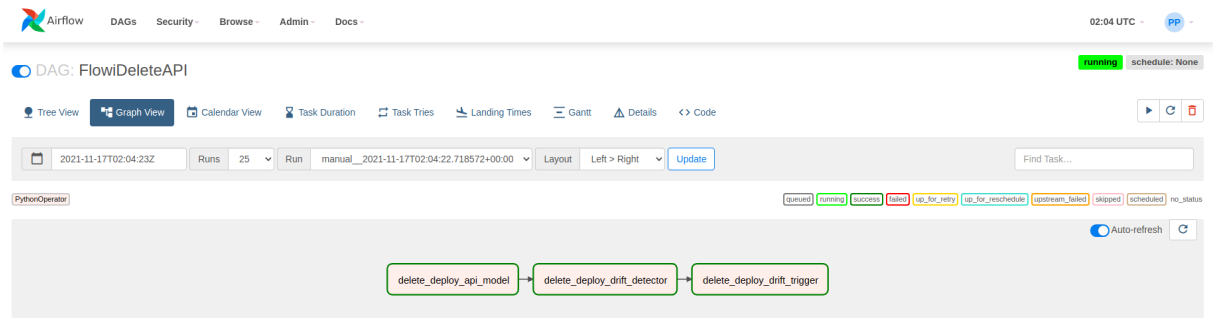


Fonte: Elaborada pelo autor.

3.2.5 FlowiDeleteAPI

A DAG FlowiDeleteAPI (Figure 3.2.5) é responsável por deletar a API de um projeto. As etapas para deletar *deploy* de API são as mesmas etapas do *deploy* exceto pela criação do *container*.

Figura 16 – DAG FlowiDeleteAPI no Airflow contendo as etapas para deletar *deploy* de API. São as mesmas etapas do *deploy* exceto pela criação do *container*.



Fonte: Elaborada pelo autor.

3.3 Rastreio de Experimentos

O rastreio de experimentos ocorre por padrão sendo a implementação padrão do MLFlow (Figura 17), mas podendo adicionar outros provedores como o Polyaxon[18] ou Neptune AI[19]. O

[18] <https://polyaxon.com/>, acessado em 14/10/2022
[19] <https://neptune.ai/>, acessado em 14/10/2022

gerenciador é escolhido na etapa de *schedule* da UI conforme a Figura 8.

O rastreio (Figura 18) salva quais parâmetros foram utilizados em cada método para que seja possível comparar os diferentes experimentos (Figura 19). Além disso, salva-se as métricas escolhidas (ex. acurácia, performance, etc), a versão do *Flow* e todos os artefatos gerados durante o treinamento (*input transformer*, *output transformer*, modelo, colunas do *dataframe*, modelo para detecção de *drift*).

Como dito anteriormente, quando mais de um valor é selecionado na UI, é gerado mais de um experimento automaticamente.

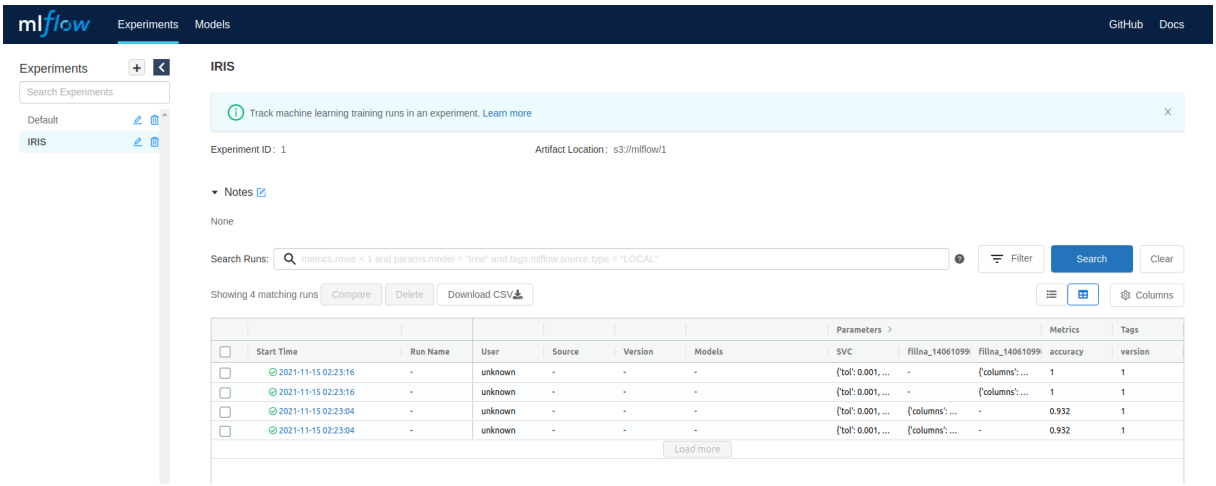
O MLFlow tem integração nativa com o S3[20] para salvar os artefatos (modelos, imagens, transformações de *input* e *output*, etc). É importante que o Flowi não tenha dependência com nenhum provedor, no caso AWS, por isso, optou-se por utilizar o MinIO como armazenamento de objetos. O MinIO tem três grandes vantagens:

1. *Hibrid* e *Multi-Cloud*: Por ser *cloud native* e construído sobre o Kubernetes, pode ser instalado em qualquer provedor de nuvem e, caso o cluster seja *multi-cloud*, ele também será.

2. *Cloud native*: conteinerização, orquestração com Kubernetes, microsserviços e multilocação

3. Alta performance: MinIO se diz o *object store* mais rápido com velocidade de leitura/escrita de 183 GB/s and 171 GB/s em hardware padrão

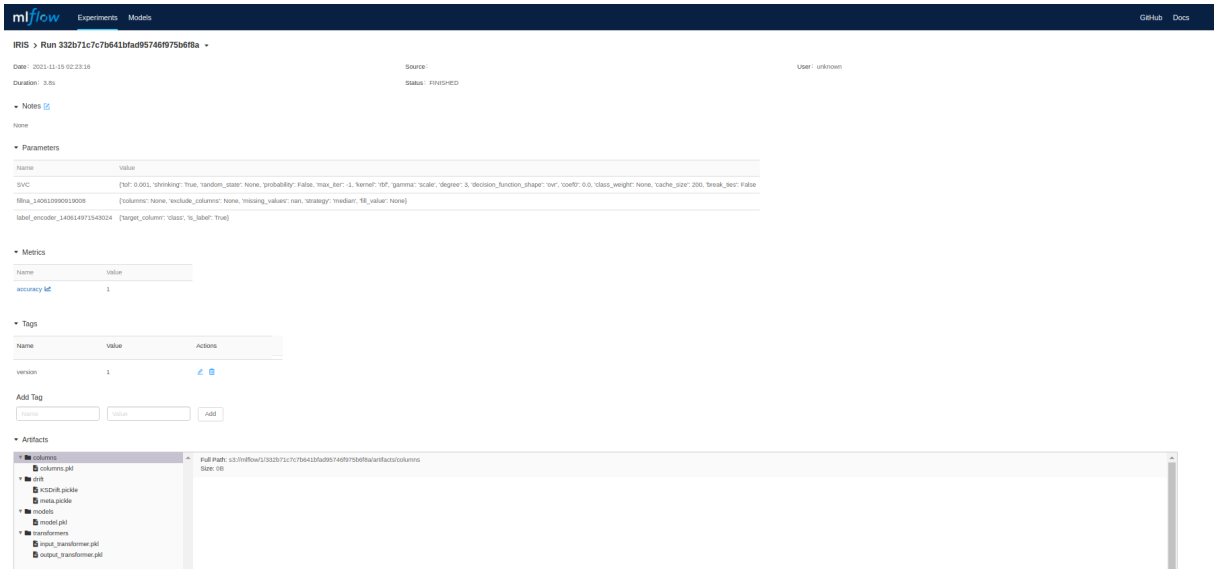
Figura 17 – Gerenciamento de experimentos utilizando o MLFlow.



Fonte: Elaborada pelo autor.

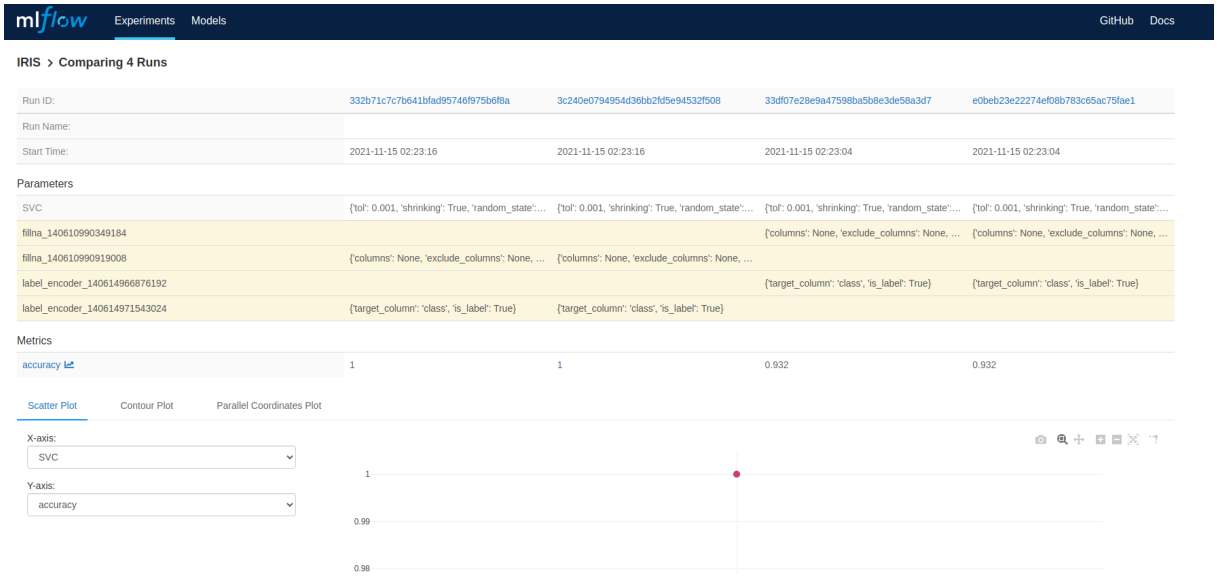
[20] <https://aws.amazon.com/s3/>, acessado em 14/10/2022

Figura 18 – Rastreio de diversas informações de treino como parâmetros, métricas e artefatos no MLFlow.



Fonte: Elaborada pelo autor.

Figura 19 – Comparação entre diversos experimentos no MLFlow.



Fonte: Elaborada pelo autor.

3.4 Data Preparation

O Flowi tem como meta ser um ambiente padrão para o desenvolvimento de modelos de ML, por isso deve lidar bem com volumes pequenos (alguns Kb) e grandes (dezenas de Gb). Tratando-se de Big Data, a tecnologia padrão é Apache Spark[21], entretanto o Spark foi feito para Big Data e tem *run time* lento para pouco volume. Ademais, o Spark não é facilmente serializável e necessita de um *Spark run time* para a predição final.

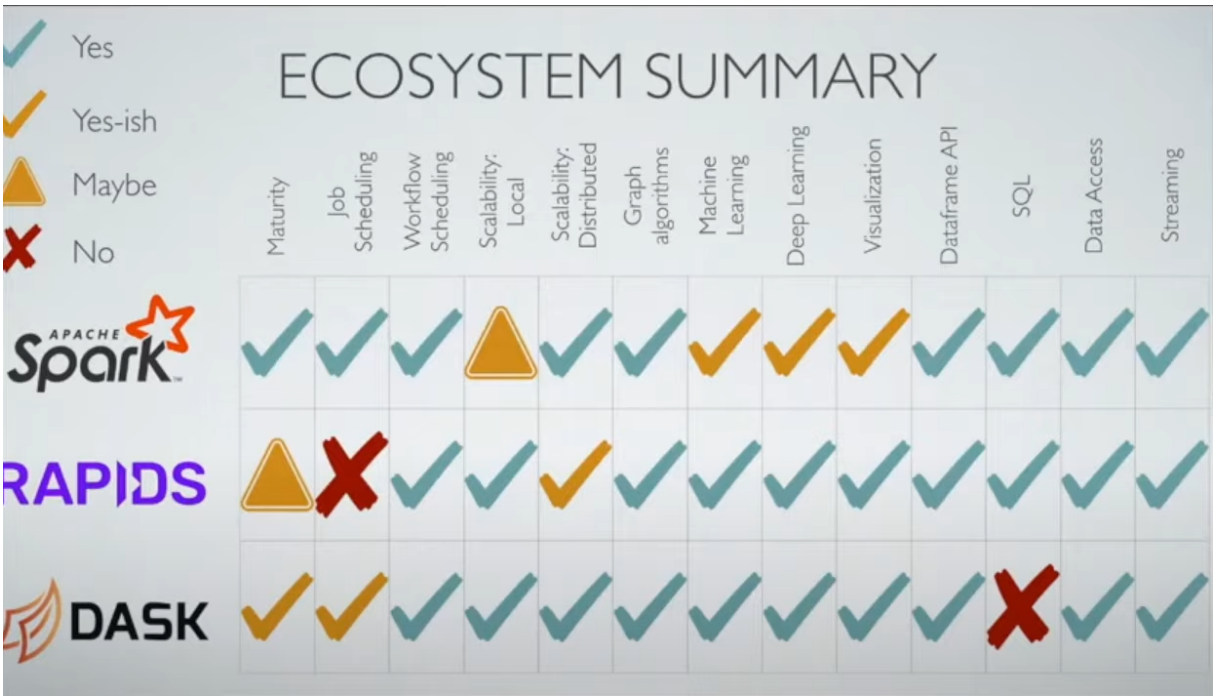
[21] <https://spark.apache.org/>, acessado em 14/10/2022

Dill (DILL, 2019) em sua palestra no PyData NYC explica vantagens e desvantagens do Dask frente ao Spark (e Rapids) especialmente voltado para o negócio e sustentação de ambiente. Em suma, por Spark ser consolidado na indústria, existem diversas empresas e serviços que mitigam os riscos de negócio como consultorias, suporte 24/7 e treinamento para equipes. Entretanto, do ponto de vista de ecossistema (Figura 20), ou seja, suas funcionalidades, Dask apresenta diversos pontos positivos especialmente por trabalhar muito bem com Escalabilidade Local (para arquivos pequenos e predição online), machine learning (para treino com *deep learning* e modelos tradicionais (scikit learn), visualização e acesso a diversos tipos de dados.

O capítulo Resultados (capítulo 4) apresenta um estudo comparativo das performances do Dask e PySpark voltado para o caso de uso do Flowi. Ou seja, compara-se a performance dos dois *frameworks* para predições online (API) e offline (*batch*). Mediante essa comparação, é notório o melhor desempenho do Dask para pequenos (Kb) e médios (Gb) arquivos enquanto que para arquivos grandes (11Gb) os resultados são equivalentes. (MEHTA *et al.*, 2016) e (DUGRÉ; HAYOT-SASSON; GLATARD, 2019) suportam que o Dask tem performance equivalente ao Spark tratando-se de Big Data.

Portanto, tendo em vista que o mesmo ambiente e transformações utilizadas durante o treinamento serão utilizadas em inferência para *Online* e *Batch* e que é necessário integração fluida com *frameworks* de *deep learning*, Dask tem as melhores características pois trabalha bem com todos os tamanhos de arquivos estudados e é baseado em pandas tornando sua integração com diversos outros *frameworks* e pacotes de ML mais simples do que com Spark.

Figura 20 – Comparação de funcionalidades entre Dask, Spark e Rapids.



Fonte: Dill (2019).

3.5 Model training

O treinamento dos modelos pode ser realizado utilizando qualquer pacote Python, entretanto terá melhor performance se for distribuído. Para modelos tradicionais usa-se a biblioteca Dask ML[22] e para *deep learning* pode-se utilizar Tensorflow[23] ou PyTorch[24].

Além de aceitar os *frameworks* padrões de aprendizado de máquina o Flowi sempre faz otimização de hiper-parâmetros utilizando o próprio Dask[25].

O treinamento pode ser realizado utilizando apenas CPU ou com GPU. Não é necessário nenhuma configuração adicional ao Flowi caso o uso de GPU seja desejado pois o *container* do Flowi é configurado para GPU e usará automaticamente caso a GPU esteja disponível.

Sabe-se que alguns modelos necessitam de *decode* uma vez que seu resultado pode ser uma matriz ou uma lista. Neste caso, é necessário criar um *wrapper* para o modelo criando um método *predict* que trata o *decode*.

Todo modelo treinado, além de salvar para o gerenciador de experimentos (ex. MLFlow), também salva as informações (não artefatos) no Mongo DB. Isto é feito para o fácil rastreio do que está em produção e para centralizar os dados caso sejam utilizados mais de um gerenciador de experimentos.

O dados do Mongo DB são utilizados na etapa de escolha de *update model* descrita no capítulo 3.2.2.

3.6 Validação e teste de modelos

Ao final de todo fluxo de treinamento, é necessário validar o modelo e as transformações de *input* e *output* bem como testar o modelo para as métricas estipuladas no desenho do fluxo na UI, como acurácia ou precisão.

Para isso, todas as transformações dos dados (fillna, normalização, categorização, etc) são condensadas em um *scikit learn pipeline*[26] chamado *input_transform* e as transformações nos *labels* (categorização) são condensadas no *output_transform*. Assim, a valiação do modelo abrange todas as etapas do processo. Caso haja alguma falha na predição do modelo na base de teste, a aplicação encerra com erro. Caso a predição ocorra, as métricas desejadas são aferidas e salvas no gerenciador de experimentos (ex. Mlflow).

Essas métricas também são usadas para selecionar o melhor modelo do treino. Esse

[22] <https://ml.dask.org/>, acessado em 14/10/2022
[23] <https://ml.dask.org/keras.html>, acessado em 14/10/2022
[24] <https://ml.dask.org/pytorch.html>, acessado em 14/10/2022
[25] <https://ml.dask.org/hyper-parameter-search.html>, acessado em 14/10/2022
[26] <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>, acessado em 14/10/2022

modelo é enviado para *staging* e será comparado com o modelo produtivo no Airflow, sendo o Airflow responsável por toda a orquestração do Flowi vide seção Orquestração deste capítulo).

3.7 Model serving

Como dito anteriormente, o Flowi disponibiliza duas formas de servir os modelos: *Online* (API) e *Offline* (Batch). Essas duas formas têm características distintas; enquanto o primeiro zela pela baixa latência a despeito do *throughput* o segundo zela pelo alto *throughput* a despeito da latência. Por isso, as tecnologias para servir devem ser distintas.

3.7.1 API

Para a API utiliza-se o Seldon como orquestrador. Com ele é possível gerenciar e escalar os modelos no Kubernetes de forma praticamente transparente utilizando-se de um Operador do Seldon instalado no *cluster*. Com isso, é apenas necessário criar um *container* com o pacote python do Seldon servindo o modelo e ele automaticamente disponibilizará uma API REST e um gRPC para consumo.

Na predição é utilizado o modelo propriamente dito, o nome das colunas para transformar o *input* em pandas *dataframe*, o *input transformer* e o *output transformer*. Dessa forma, espera-se receber os dados não tratados assim como utilizado no treino e retornar-se-á a classe esperada como dada no treino. Por exemplo, no caso do *dataset* Iris, a saída é o nome da espécie.

O escalonamento do modelo é feito naturalmente pelo Seldon utilizando o *autoscaling* por *workload* por padrão, mas pode ser alterado para quantidade de requisições ou diversas outras métricas se usar o *autoscaling* KEDA[27].

Por padrão, o Flowi também realiza a detecção de *drift* para todo modelo. É necessário criar um segundo modelo que entenda as características dos dados de entrada do modelo principal a fim de detectar se houve alguma alteração grande nos dados. O modelo de *drift* é criado utilizando o método Kolmogorov-Smirnov da biblioteca alibi-detect[28] pois é capaz de detectar desvios a nível de *feature* em dados tabulares (*dataframe*), imagens e texto.

3.7.2 Batch

O Pacote Flowi permite tanto o treino quanto a predição *Batch*. Assim, o reúso de código é maximizado enquanto se faz uso de funcionalidades importantes para a predição *Batch* como uso de GPU e pré e pós processamento distribuído pelo Dask.

Diferentemente do fluxo online, a detecção de *drift* no *Batch* é feito antes da predição para minimizar o uso de recursos.

[27] <https://keda.sh/>, acessado em 14/10/2022
[28] <https://github.com/SeldonIO/alibi-detect>, acessado em 14/10/2022

Os dados são carregados sempre do mesmo local e são salvos também no mesmo local, como indicado na UI. Para alterar o local é necessário criar uma nova versão do *Flow*.

O escalonamento é feito automaticamente pelo *cluster* Dask no Kubernetes baseado na quantidade de CPU e memória utilizados. Vale lembrar que o Dask permite trabalhar com dados que não cabem na memória (*out of core*) nativamente que é uma característica necessária para processamento *Batch*.

3.8 Monitoramento

Monitoramento é um dos pilares de MLOps e, por conseguinte, do Flowi. Ele deve ser feito a nível de automação do processo de *deploy* (Airflow) e ao se servir o modelo.

3.8.1 Automação de deploy

A seção Orquestração explica todo o processo de *deploy* e predição Batch no Airflow. Uma das grandes funcionalidades do Airflow é o monitoramento de *workflows* sendo possível identificar quando uma DAG falhou, quais os dados de entrada, *logs* da aplicação e quantidade de tentativas até falhar. Além disso, é possível configurar emails automáticos em caso de falha ou configurar *plug-ins* para notificar outro sistema como o Slack[29].

3.8.2 Predição Batch

A predição *Batch* também é orquestrada pelo Airflow (seção 3.7.2) tendo os mesmos benefícios da automação de *deploy* descritos acima. Além disso, foi implementado um fluxo para identificação de *drift* (Figura 15) onde a equipe responsável pelo sistema pode ser notificada para atuar.

3.8.3 API

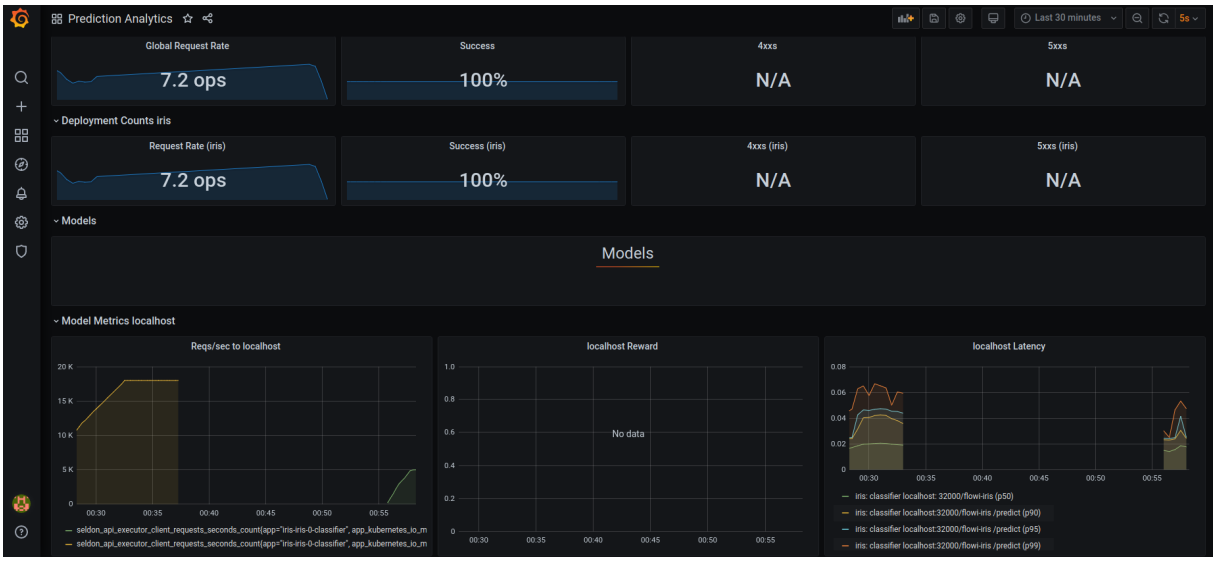
A API demanda o monitoramento de latência, taxa de sucesso e detecção de *drift*. Para isso, é necessário o uso de alguns sistemas.

O Seldon integra nativamente com o Prometheus[30] solução *open-source* líder de monitoramento de métricas. Ele é responsável por captar diversas métricas do Seldon como tempo de predição, quantidade de requisições por segundo e consumo de CPU/memória. As métricas são consumidas pelo Grafana[31] para a criação de gráficos analíticos como a Figura 21. Esse dashboard é criado automaticamente pelo Seldon e adiciona cada modelo gerenciado por ele.

[29] <https://slack.com/>, acessado em 14/10/2022
[30] <https://prometheus.io/>, acessado em 14/10/2022
[31] <https://grafana.com/>, acessado em 14/10/2022

Pelo Grafana é possível adicionar diversos alertas sobre os gráficos para que o time responsável por manter a aplicação saiba quando atuar e por qual motivo.

Figura 21 – Visualização de métricas da API Iris pelo Grafana.



Fonte: Elaborada pelo autor.

Assim como é no processo *Batch* também é importante ter acesso aos *logs* de aplicação no fluxo *Online*. Isso ocorre pela integração do Seldon com o Elasticsearch[32] e com o Kibana[33]. Em suma, todos os *logs* são enviados para o ElasticSearch e este pode ser acessado pelo Kibana ou pelo Grafana para análise dos logs. A Figura 22 mostra os *logs* no Kibana. Pelo Kibana também é possível criar os índices no Elasticsearch facilitando o acesso aos *logs*.

No Grafana foi configurada a análise de *drift* (Figura 23) consumindo os dados do Elasticsearch. A implementação no Grafana facilita pois todo o monitoramento fica no mesmo local.

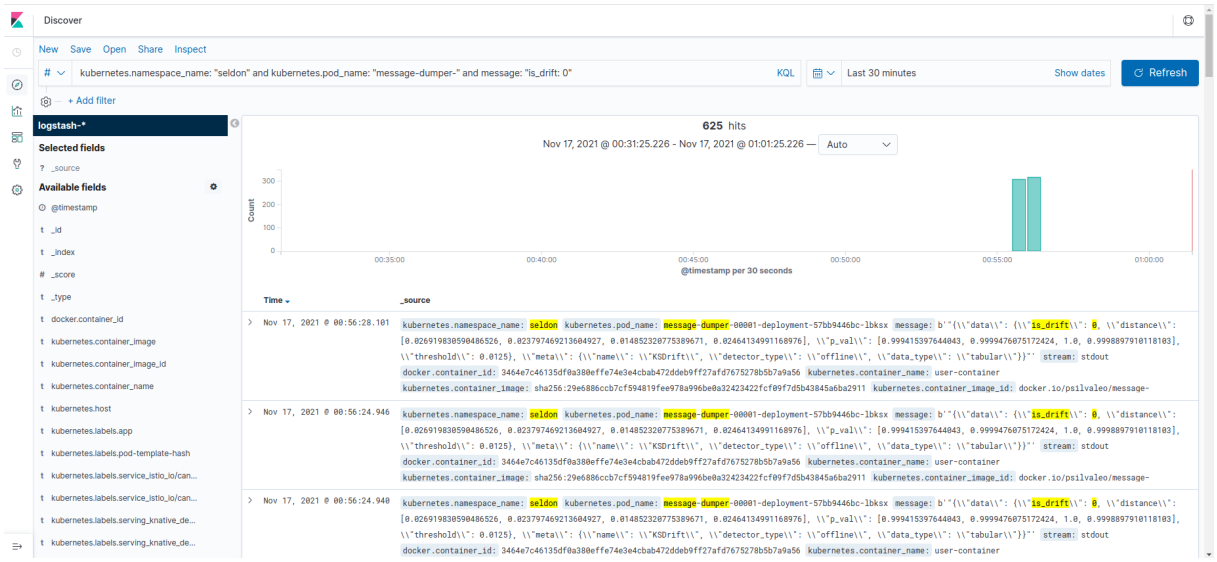
3.9 *Data extraction* e *Data analysis*

Uma das tarefas que o Flowi se propõe a facilitar é o tratamento de dados e a elaboração de experimentos pela alteração de parâmetros. Isso é feito atribuindo diversos valores para o parâmetro na UI e o pacote Python Flowi (*backend*) se encarrega de entender as entradas e gerenciar os diversos experimentos. Entretanto, o processo de *Exploratory Data Analysis* (EDA) não é o foco do Flowi.

O pacote Python Flowi pode ser usado para EDA pois seus métodos são bem definidos quanto a entrada e saída e abstraem o uso do Dask e outras tecnologias. O que se perde, no entanto, é a facilidade de uso e automações como o gerenciamento de experimentos.

[32] <https://www.elastic.co/>, acessado em 14/10/2022
[33] <https://www.elastic.co/kibana/>, acessado em 14/10/2022

Figura 22 – Visualização de *logs* de aplicação pelo Kibana.



Fonte: Elaborada pelo autor.

Figura 23 – Visualização de *logs* de aplicação pelo Kibana.

Fonte: Elaborada pelo autor.

Por ser baseado no Dask existem vantagens muito grandes em utilizar o Flowi para a etapa de EDA:

1. Pacote pré-instalados: O Flowi depende de vários pacotes de ciências de dados e, em utilizar o Flowi, diversos deles já estão instalados e assegura-se a mesma versão;

2. Baseado em pandas: Dask é baseado em pandas[34] (ferramenta de manipulação e análise de dados open source implementada em Python), sendo esta a ferramenta de *dataframe* padrão em ciência de dados. Basta converter o *dataframe* Dask para pandas e utilizar as ferramentas comuns de EDA;

3. Datashader[35]: Ferramenta gráfica para representação de *datasets* conseguindo trabalhar com Big Data.

34 <https://pandas.pydata.org/>, acessado em 14/10/2022
35 <https://datashader.org/>, acessado em 14/10/2022

3.10 Considerações Finais

Neste capítulo foi apresentada a arquitetura do Flowi e seus componentes de desenvolvimento e operação de modelos de aprendizado de máquinas divididos em três partes principais: UI, pacote Python e orquestração de fluxo pelo Airflow. Esses componentes permitem incorporar as diferentes etapas de ML e MLOps tais como preparação dos dados, treinamento de modelos com otimização de hiper-parâmetros, rastreamento de experimentos, predição online, predição offline, validação e seleção de modelos e monitoramento dos modelos produtivos constituindo-se assim de uma ferramenta de apoio completa a cientistas de dados, engenheiros de *machine learning* e outros profissionais.

Como objetivo final, o Flowi visa satisfazer as necessidades de engenharia e operação do ciclo de vida de ML por meio de uma solução estruturada (fácil de atualizar, manter, e melhorar) permitindo a empresas operar com eficiência e robustez sistema de ML em produção. Vale ressaltar que o Flowi tem por objetivo diminuir a curva de aprendizado e de entrada por ter interface intuitiva e por facilmente integrar ou criar componentes.

No próximo capítulo, serão apresentados os resultados do projeto. O sistema Flowi como contribuição principal e resultados que basearam a definição da arquitetura proposta.

CAPÍTULO

4

RESULTADOS

Diversas contribuições e resultados foram obtidos durante a construção do sistema Flowi, mas é possível sumarizar em duas frentes: Comparação entre Dask e PySpark para decidir qual a melhor tecnologia para o sistema e o Flowi propriamente dito.

4.1 Comparação entre Dask e PySpark

O processamento de dados inclui acessar dados brutos de diversas fontes (ex. banco de dados, *data lake*, etc), tratar e transformar para treinamento. Como apresentado em Data Preparation, o Flowi tem como premissa trabalhar com baixo e grande volume de dados, ou seja, de poucos Kbs a dezenas de Gbs para que seja possível realizar processos *batch* em *Big Data* e transformação online nos dados em poucos milisegundos em uma predição online.

Do ponto de vista de engenharia, as otimizações de sistemas para os dois casos de uso são distintas o que torna difícil achar um *framework* único para as duas etapas. Implementar os dois fluxos com *frameworks* distintos traz três pontos negativos: é necessário implementar as transformações duas vezes (retrabalho), adiciona-se o risco das transformações serem diferentes por algum erro de implementação e aumenta a complexidade geral do sistema. Apesar disso, é preciso ponderar a melhoria de performance que o sistema teria.

Atualmente, as duas tecnologias mais apropriadas para *Big Data* são Dask e Spark. Enquanto a tecnologia mais utilizada para dataframes pequenos é Pandas. Pandas provê uma interface abrangente para tratamento e manuseio de dados, entretanto, sua implementação é baseada em apenas um *core* e o dado necessita caber na memória. Dask distribui o processamento em várias partições Pandas, superando as limitações processamento em um único *core* e limitado a memória da máquina. Spark, é implementado em Java/Scala e necessita de uma JVM ativa. É projetado para lidar com dados em alta volumetria, mas demanda um *startup* de alguns segundos. Tanto Dask como Spark utilizam de uma arquitetura semelhante a *master/slave*, Spark tem um

driver/master para cada aplicação e este driver gerencia os executores. Dask, por sua vez, tem um scheduler global que gerencia todos os executores do cluster baseado na demanda da aplicação.

A Tabela 1 apresenta os resultados ao se comparar a performance de escrita, Fillna (transformação que troca valores nulos, comumente encontrados como NaN - not a number - em *dataframes* por valores numéricos), *max e max-min scale* para arquivos pequenos (Kb), médios (Gb) e grandes (11Gb) em uma Virtual Machine - Máquina Virtual (VM) com 12Gb de RAM e 10 vCPUs. Nota-se que para arquivos pequenos e médios, a performance do Dask é superior em todos os testes chegando a ser 35 vezes mais rápido no MinMaxScale em arquivos pequenos. Entretanto, para arquivos grandes (acima de 10Gb) os dois tiveram performance semelhante. Apesar da análise não ser exaustiva, é notório que o Dask apresenta menor tempo de execução para arquivos pequenos e médios tornando-o mais rápido para o cenário de execuções online (API). No caso de grandes arquivos, outros artigos suportam que Dask e Spark operam com performance equivalente (DUGRÉ; HAYOT-SASSON; GLATARD, 2019) e (MEHTA *et al.*, 2016) para arquivos grandes.

A avaliação de performance, em termos de execução, em sistemas de ML, é um tópico muito importante de pesquisa, pois na maioria dos trabalhos, é discutida apenas a performance relativa a qualidade do aprendizado e não em relação ao desempenho computacional. Este trabalho permitiu uma comparação, bem estruturada e modular, de duas tecnologias amplamente adotadas, permitindo assim avaliar de forma prática seus desempenhos, e inclusive, aproveitando os recursos oferecidos pela plataforma Flowi para que fosse possível realizar esta comparação.

Tabela 1 – Comparação da performance do Dask e Pyspark em operações de escrita, fillna, max and min-max scale para pequenos (Kb) médios (1Gb) e grandes (10Gb) arquivos. Melhores performances em negrito.

Operação	Engine	Arquivo Pequeno (ms)	Arquivo Médio (s)	arquivo Grande (s)
Escrita	Dask	**65.96**	**18.90**	**72.89**
	PySpark	299.64	50.07	153.69
Fillna	Dask	**56.48**	**11.82**	149.56
	PySpark	182.35	21.49	**100.39**
Max	Dask	**4.32**	**0.41**	3.79
	PySpark	115.88	0.45	**2.06**
MinMaxScale	Dask	**24.89**	**6.88**	**266.73**
	PySpark	892.30	57.95	267.17

4.2 Flowi

O principal resultado esperado por este trabalho é a própria implementação do Flowi satisfazendo as necessidade de MLOps e facilitando o desenvolvimento e reprodutibilidade de experimentos pelos cientistas de dados. O vídeo <https://youtu.be/QR2bpmcjpXU> mostra todo o funcionamento do sistema, desde a criação do fluxo (*Flow*) na UI, passando pela orquestração do Airflow e treino no Dask até a disponibilização e consumo dos modelos e a detecção de *drift* automática. No vídeo a demonstração é feita através de um estudo de caso, feito de forma prática utilizando o *dataset* IRIS devido à sua grande popularidade. Ele demonstra a praticidade, simplicidade e efetividade do uso do Flowi em tarefas de ML e MLOPs, do treino ao deploy da apliação.

4.2.1 Flowi UI

A interface web foi implementada em React[1] e está disponível em <https://github.com/psilva-leo/flowi-ui>. A interface permite a criação e edição de *Flows* para treinamento e *deploy* dos modelos. Assim como ilustrado na Figura 5, Flowi UI roda em um *container* (disponível em <https://hub.docker.com/repository/docker/psilvaleo/flowi-ui>) no Kubernetes.

O código é composto por JavaScript (47.0%), CSS (37.7%) e SCSS (15.1%) divididos em 334 arquivos e 54699 linhas de código. Para o gerenciamento dos *Flows* o Flowi UI é integrado com a API do Airflow[2].

4.2.2 Pacote Flowi

O pacote Python Flowi está disponível em <https://github.com/psilva-leo/flowi> e publicado em <https://pypi.org/project/flowi/>. O pacote Python Flowi recebe as informações de treinamento em formato json, valida se é um fluxo válido e realiza o treinamento de forma distribuída utilizando Dask. Ele também cria quantos pipelines forem necessários baseado nas estratégias de cada componente.

Para um fluxo ser válido, ele deve ser uma DAG (Directed Acyclic Graph), ou seja, é um grafo sem ciclos ou *loops* e a ligação entre os componentes segue apenas uma direção.

O Flowi também padroniza todas as transformações e manipulações nos dados e modelos mediante classes e métodos com assinaturas bem definidas. Além disso, todos os componentes tem um método *set_output* para definir a saída do componente.

O código é escrito em Python (98.1%) e contém 148 arquivos e 5518 linhas de código.

[1] <https://reactjs.org/>, acessado em 20/10/2022

[2] <https://airflow.apache.org/docs/apache-airflow/stable/stable-rest-api-ref.html>, acessado em 20/10/2022

4.2.3 Integrações

Conforme apresentado no Capítulo 3, o Flowi é composto por diversas aplicações demandando várias integrações.

- Flowi UI integra com o Airflow por meio de sua API para enviar os dados do *Flow*
- Airflow integra com o Kubernetes para orquestrar o treinamento e a predição via pacote Flowi
- Airflow integra com Mongo para decidir se deve atualizar o modelo
- Airflow integra com Docker para a criação dos containers de predição
- Flowi integra com Dask *scheduler* para processamento distribuído
- Flowi integra com GPU, Tensorflow, Pytorch e Scikit-Learn para treinamento de modelos
- Flowi integra com MLflow e Minio para rastreamento de modelos
- Todo o cluster Kubernetes integra com Prometheus para métricas de cpu e memória
- Todo o cluster Kubernetes integra com ElasticSearch para logs
- Seldon integra com Grafana para monitoramento do modelos
- Seldon integra com Knative para drift detection

4.2.4 Considerações Finais

Este trabalho apresentou diversos resultados e contribuições, desde uma interface web até toda a infraestrutura para hospedar as aplicações que compõem a plataforma Flowi. Entretanto, ainda há trabalho a ser feito. Esses trabalhos futuros são apresentados no Capítulo 5.

CAPÍTULO

5

CONCLUSÕES E TRABALHOS FUTUROS

Concluindo, neste trabalho se propôs uma nova arquitetura para desenvolvimento e gerenciamento de modelos de aprendizado de máquina com a intenção de: facilitar para os cientistas de dados o desenvolvimento de modelos com boas práticas por padrão; bem como auxiliar equipes de TI a gerenciar a infraestrutura, tendo todo o *deploy* automatizado. Além de disponibilizar para ambos um sistema com tecnologias do estado da arte para um processo robusto, eficiente e reprodutível de ciclo de vida completo de ML.

Apesar disso, ML e a área de inteligência artificial como um todo estão evoluindo rapidamente e melhorias já são vislumbradas no sistema.

5.1 Trabalhos Futuros

Apesar da arquitetura apresentada por este trabalho solucionar diversos desafios do ciclo de vida de modelos de aprendizado de máquina, existem melhorias e recomendações de trabalhos futuros que podem aprimorar ainda mais a solução proposta a fim de garantir uma melhor experiência e satisfação aos usuários da plataforma.

5.1.1 Integração com os sistemas

O Flowi aproveita diversos sistemas *open-source* para sua concepção. Apesar dessa característica ser boa e facilitar a integração com sistemas das empresas, há uma certa dificuldade em operar todos ao mesmo tempo. Existem algumas soluções possíveis para isso na interface web do Flowi tais como:

- *Embedded*: Disponibilizar o acesso aos sistemas externos colocando-os em *frames* na própria interface do Flowi
- *Links*: Disponibilizar no menu lateral botões que vão diretamente para os sistemas externos

- *Dashboard* customizados: Elaborar na UI um dashboard que sumarize as informações mais relevantes dos outros sistemas, tais como quantidade de sucesso e falha nas DAGs do Airflow e embutir os gráficos do Grafana[1]

5.1.2 UI

Para a criação dos fluxos na UI é necessário pelo menos saber quais são as colunas do *dataset* e saber se é um dado tabular, imagem, ou áudio. Apesar de não serem informações difíceis de se conseguir, é interessante que o Flowi tenha a possibilidade de interagir com os *datasets* de forma mínima pela UI identificando as colunas, tipo do dado, distribuição, etc.

5.1.3 Tipos de dados

A implementação atual do Flowi foi pensada para trabalhar com diversos tipos de dados como tabular (*dataframe*), imagens e áudio. Apesar disso, ainda existem melhorias para aperfeiçoar o treinamento com imagens e áudio. Também é interessante atualizar a plataforma para trabalhar com texto, afinal, Processamento de Linguagem Natural é uma grande área de IA.

5.1.4 Hummingbird

A Microsoft tem um projeto chamado Hummingbird[2] (NAKANDALA *et al.*, 2020) que visa compilar modelos de aprendizado de máquina tradicionais (como árvore de decisão) em *tensors* de *frameworks* de redes neurais como PyTorch. Isso possibilita que os modelos tradicionais tenham as otimizações de *hardware* utilizadas pelos modelos de *deep learning* além de unificar a plataforma de predição. Segundo os autores, Hummingbird acelera a predição tanto em CPU quanto em GPU.

5.1.5 Componentes customizados

Algumas empresas têm transformações proprietárias ou simplesmente não podem esperar a adição de um componente, portanto, é interessante que os usuários da plataforma consigam adicionar componentes customizados e utilizados em todo o *pipeline*. Para isso é necessário alterar a UI e o pacote Python.

Vale ressaltar que a adição de tais componentes pode necessitar a instalação de outros pacotes e é imprescindível que esse mapeamento seja feito no processo.

[1] <https://grafana.com/docs/grafana/v7.5/sharing/share-panel/>
[2] <https://github.com/microsoft/hummingbird>

5.1.6 Estimativa de Custo

Ao se trabalhar com Big Data e GPU os custos podem aumentar rapidamente. Por conta disso, é interessante que o sistema consiga estimar o custo de um treinamento baseado no tamanho do *dataset* e transformações especificadas. A nível de negócio, isso permite um planejamento financeiro mais preciso para as empresas que venham a adotar essa ferramenta.

5.2 Principais Contribuições

As principais contribuições deste trabalho são:

- A comparação de performance entre diversos modelos;
- O desenvolvimento e disponibilização de ferramentas ligadas ao ambiente Flowi implementadas pelo autor:
  - Interface web (UI) para criação do fluxo (flow) de transformações. Disponibilizado em: <https://hub.docker.com/repository/docker/psilvaleo/flowi-ui>
  - Pacote Python Flowi. Disponibilizado em: <https://pypi.org/project/flowi/>
- O vídeo com apresentação e demonstração de uso do Flowi. Disponibilizado em: <https://youtu.be/QR2bpmcjpXU>

E, por fim, foi produzido um trabalho publicado e apresentado no WMECAI. Disponível em: <http://cemeai.icmc.usp.br/1WMECAI/>.

REFERÊNCIAS

ABBASI, B. K. A.; AHMAD, F. **The Risks of AutoML and How to Avoid Them**. 2019. Disponível em: <https://hbr.org/2019/10/the-risks-of-automl-and-how-to-avoid-them>. Acesso em: 28/05/2022. Citado na página 36.

AWS, A. **What is DevOps?** 2021. Disponível em: <https://aws.amazon.com/devops/what-is-devops/>. Acesso em: 10/11/2021. Citado na página 23.

BASS, L.; WEBER, I.; ZHU, L. **DevOps: A software architect's perspective**. [S.l.]: Addison-Wesley Professional, 2015. Citado na página 23.

BISONG, E. Kubeflow and kubeflow pipelines. In: **Building Machine Learning and Deep Learning Models on Google Cloud Platform**. [S.l.]: Springer, 2019. p. 671–685. Citado na página 35.

BOSE, A.; AGGARWAL, A. **MLOps – "Why is it required?" and "What it is"?** 2020. Disponível em: <https://www.kdnuggets.com/2020/12/mlops-why-required-what-is.html>. Acesso em: 08/11/2021. Citado na página 23.

BROWNLEE, J. **Data preparation for machine learning: data cleaning, feature selection, and data transforms in Python**. [S.l.]: Machine Learning Mastery, 2020. Citado na página 33.

CHAUHAN, G. **Iris Dataset Project from UCI Machine Learning Repository**. 2021. Disponível em: <https://machinelearninghd.com/iris-dataset-uci-machine-learning-repository-project/>. Acesso em: 23/05/2022. Citado na página 34.

CHEN, I. Y.; PIERSON, E.; ROSE, S.; JOSHI, S.; FERRYMAN, K.; GHASSEMI, M. Ethical machine learning in healthcare. **Annual Review of Biomedical Data Science**, Annual Reviews, v. 4, p. 123–144, 2021. Citado na página 36.

DILL, E. **Is Spark still relevant?** 2019. Disponível em: <https://www.youtube.com/watch?v=obKZzFRNTxo>. Acesso em: 15/11/2021. Citado na página 50.

DUGRÉ, M.; HAYOT-SASSON, V.; GLATARD, T. A performance comparison of dask and apache spark for data-intensive neuroimaging pipelines. In: IEEE. **2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)**. [S.l.], 2019. p. 40–49. Citado nas páginas 50 e 58.

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: 10/11/2021. Citado nas páginas 27 e 28.

HERMANN, J.; BALSO, M. D. **Meet Michelangelo: Uber's Machine Learning Platform**. 2017. Disponível em: <https://eng.uber.com/michelangelo-machine-learning-platform/>. Acesso em: 05/09/2017. Citado na página 36.

JORDAN, J. **Evaluating a machine learning model**. 2017. Disponível em: <https://www.jeremyjordan.me/evaluating-a-machine-learning-model/>. Acesso em: 28/05/2022. Citado nas páginas 30, 33 e 34.

MDN, M. **Payload body**. 2022. Disponível em: <https://developer.mozilla.org/en-US/docs/Glossary/Payload_body>. Acesso em: 14/10/2022. Citado na página 43.

MEHTA, P.; DORKENWALD, S.; ZHAO, D.; KAFTAN, T.; CHEUNG, A.; BALAZINSKA, M.; ROKEM, A.; CONNOLLY, A.; VANDERPLAS, J.; ALSAYYAD, Y. Comparative evaluation of big-data systems on scientific image analytics workloads. **arXiv preprint arXiv:1612.02485**, 2016. Citado nas páginas 50 e 58.

MOHAJON, J. **Confusion Matrix for Your Multi-Class Machine Learning Model**. 2020. Disponível em: <https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826>. Acesso em: 28/05/2022. Citado na página 32.

MOHSENI, S.; PITALE, M.; SINGH, V.; WANG, Z. Practical solutions for machine learning safety in autonomous vehicles. **arXiv preprint arXiv:1912.09630**, 2019. Citado na página 23.

MORI, S.; SUEN, C.; YAMAMOTO, K. Historical review of ocr research and development. **Proceedings of the IEEE**, v. 80, n. 7, p. 1029–1058, 1992. Citado na página 23.

NAKANDALA, S.; SAUR, K.; YU, G.; KARANASOS, K.; CURINO, C.; WEIMER, M.; INTERLAND, M. **Taming Model Serving Complexity, Performance and Cost: A Compilation to Tensor Computations Approach**. 2020. Citado na página 62.

RASCHKA, S. Model evaluation, model selection, and algorithm selection in machine learning. **arXiv preprint arXiv:1811.12808**, 2018. Citado na página 30.

REZENDE, S. O.; MONARD, M. C.; CARVALHO, A. C. P. d. L. F. Sistemas inteligentes para engenharia: pesquisa e desenvolvimento. **Anais**, 1999. Citado na página 30.

ROCKLIN, M. Dask: Parallel computation with blocked algorithms and task scheduling. In: CITESEER. **Proceedings of the 14th python in science conference**. [S.l.], 2015. v. 130, p. 136. Citado na página 24.

SCULLEY, D.; HOLT, G.; GOLOVIN, D.; DAVYDOV, E.; PHILLIPS, T.; EBNER, D.; CHAUDHARY, V.; YOUNG, M. Machine learning: The high interest credit card of technical debt. In: **SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)**. [S.l.: s.n.], 2014. Citado na página 27.

XING, E. P.; HO, Q.; DAI, W.; KIM, J. K.; WEI, J.; LEE, S.; ZHENG, X.; XIE, P.; KUMAR, A.; YU, Y. Petuum: A new platform for distributed machine learning on big data. **IEEE Transactions on Big Data**, v. 1, n. 2, p. 49–67, 2015. Citado na página 23.

ZAHARIA, M.; CHEN, A.; DAVIDSON, A.; GHODSI, A.; HONG, S. A.; KONWINSKI, A.; MURCHING, S.; NYKODYM, T.; OGILVIE, P.; PARKHE, M. *et al.* Accelerating the machine learning lifecycle with mlflow. **IEEE Data Eng. Bull.**, v. 41, n. 4, p. 39–45, 2018. Citado na página 28.


ICMC
USP
SÃO CARLOS