Integrador de metadados modular para aprendizado de máquina com visualização em tempo de execução

Filipe Silva[1], **Marta Mattoso**[1]

[1]PESC/COPPE - Universidade Federal do Rio de Janeiro (UFRJ)

{filipe,marta}@cos.ufrj.br

Abstract. *In recent years, data visualization during the training of a machine learning (ML) model, as well as the structured storage of metadata for future analysis, has emerged as a key abstraction to help the human in selecting a model. Existing solutions have two limitations: the first relates to the frameworks used in training, which tend to have a tightly coupled aspect, while the second, to data governance risks. Thus, humans, data scientists and analysts, face the following barriers: i) vendor lock-in and ii) application management at commercial level. This paper aims at presenting a reference architecture for ML environments, which can be applied to data visualization. Therefore, it has three main focuses: modularization, interoperability and data governance. This architecture is based on serverless computing, as it favors tight, simple and interoperable coupling. An instantiation experiment of the architecture shows runtime visualization based on independent components.*

Resumo. *Nos últimos anos, a visualização de dados durante o treinamento de um modelo de aprendizado de máquina (ML), bem como o armazenamento estruturado dos metadados para análises futuras, tem emergido como uma abstração fundamental para ajudar o humano na seleção de um modelo. As soluções existentes possuem duas limitações: a primeira relaciona-se aos frameworks utilizados no treinamento, que tendem a possuir um aspecto fortemente acoplado, enquanto a segunda, aos riscos à governança de dados. Desta forma, os humanos, cientistas e analistas de dados, deparam-se com as seguintes barreiras i) aprisionamento tecnológico e ii) gerenciamento da aplicação em nível comercial. O objetivo deste trabalho é apresentar uma arquitetura de referência para ambientes de ML, que pode ser aplicada à disposição gráfica de dados. Assim, tem-se três principais focos: modularização, interoperabilidade e governança de dados. Essa arquitetura é baseada em computação sem servidor, pois favorece o acoplamento fraco, simples e interoperável. Um experimento de instanciação da arquitetura mostra a visualização em tempo de execução com base em componentes independentes.*

1. Introdução

O Aprendizado de Máquina (ML) - chamado de *Machine Learning* em inglês - tornou-se referência para estudos que desejam aprender com uma grande massa de dados. O aprendizado baseado em dados cresce em importância quando situações sem soluções analíticas aparecem na tomada de decisão; assim, recorre-se aos dados - históricos ou sintéticos - para construir uma solução empírica [LeCun et al. 2015].

Desta forma, os experimentos de ML ganharam complexidade pela aplicação intensa em diferentes domínios. Assim, considerando que esta experimentação é um processo iterativo e exploratório, o humano responsável também encara novos desafios [Kumar et al. 2016]. Vê-se que a necessidade de armazenamento estruturado dos metadados em tempo de execução [Yuan et al. 2021] - isto é, a disponibilização dos metadados gerados pelo próprio experimento para consulta - é uma demanda do humano que precisa de ajuda na seleção de modelos de ML [Schelter et al. 2017].

Considerando o processo iterativo e exploratório para atingir a performance desejada de um modelo, muitos metadados são gerados seguindo um caminho aleatório de armazenamento e gerência dos resultados [Schelter et al. 2017]. Neste sentido, tem-se três níveis de autonomia no ciclo de vida de ML: i) liderança humana - em cada iteração o humano é responsável por tomar a decisão; ii) liderança automática - o humano especifica as amostras para aprendizado e o conjunto de hipóteses, assim, o algoritmo encontra a solução empírica ótima; iii) liderança híbrida - o humano orienta o caminho a ser explorado pelo algoritmo enquanto novos metadados são gerados [Lee and Macke 2020]. De acordo com [De Bie et al. 2022], a seleção de modelo é um desafio no contexto de ML, assim, a liderança híbrida abre caminho para exploração de alternativas com a colaboração entre sistemas inteligentes e humanos [Gil et al. 2019].

Este trabalho está organizado em cinco seções. A seção 2 discute as limitações das principais soluções de apoio à análise de dados no ciclo de vida de ML. A proposta de arquitetura de referência é apresentada na seção 3. A seção 4 apresenta uma instanciação explorando recursos ricos em análise e a flexibilidade na escolha de sistemas de gerência de dados e visualizadores. Finalmente, a seção 5 apresenta as considerações finais.

2. Trabalhos relacionados

O levantamento realizado em [Schlegel and Sattler 2023] contempla mais de 60 sistemas e plataformas para a gerência de artefatos em ML. Segundo os autores, há uma grande lacuna na exploração de dados no processo de ML. O critério ”exploração”inclui quaisquer operações que ajudem a obter ideias tangíveis sobre os resultados dos pipelines de processamento de dados, experimentos, resultados de treinamento de modelos ou estatísticas de monitoramento. Essas operações possuem subcritérios como consultabilidade, comparabilidade, proveniência e visualização. Alguns dos serviços mais populares para gerência do ciclo de vida de ML foram postos à prova e nenhum cumpriu completamente o critério de exploração, a saber: AzureML, SageMaker e Watson Studio.

Além disso, ao avaliar ferramentas como TensorBoard (código aberto) e Neptune (código fechado), voltadas para acompanhamento de experimentos, estas também apresentam lacunas importantes. Devido às limitações do TensorBoard, a comunidade de visualização criou uma área chamada VIS4ML com inúmeras propostas de visualização de dados de ML que visa a explicação e o refinamento de modelos de ML [Wang et al. 2023]. Nesse levantamento da literatura, que compreende desde 2016 a 2022, os autores analisaram 143 artigos com sistemas voltados à visualização e análise de dados escalares. Tais soluções poderiam ser adotadas na etapa de visualização de dados da arquitetura proposta, entretanto, além das soluções serem fortemente acopladas, observa-se que não há uma preocupação em adotar algum padrão na representação de armazenamento desses dados e, consequentemente, na consulta. Do ponto de vista de análise de

dados, duas principais limitações são notadas: i) desenvolvimento dedicado a *frameworks* de ML específicos, logo não são facilmente adaptáveis a outros [Pang et al. 2020]; ii) apenas opções de visualizações básicas são oferecidas, logo não se espera uma customização a usuários que precisam definir métricas complexas (escalares personalizados). Esta característica traz luz à primeira barreira: aprisionamento tecnológico.

Além disso, principalmente em soluções de código aberto, como TensorBoard e Deep-Water Framework [Schlegel and Sattler 2023], estas falham no apoio ao desenvolvimento colaborativo e aspectos de segurança, como gestão de usuários e permissões, apesar de serem mais personalizáveis. Em um caso real, o acompanhamento de modelos em tempo de execução se torna complexo; assim, dificulta-se o trabalho colaborativo. Neste contexto, surgem outras alternativas como o TensorBoard.dev, que permite compartilhar experimentos de ML com qualquer pessoa. Entretanto, a ressalva desta solução reside sobre a publicidade de todos os dados enviados, ou seja, não é possível compartilhar resultados de forma restrita [Spinner et al. 2020]. Por fim, em contraposição aos visualizadores de dados mais maduros, que oferecem uma integração simplificada aos principais SQL *engines*, a arquitetura do TensorBoard acaba entregando um ambiente bastante limitado de consulta aos artefatos do ciclo de vida. Esta característica traz luz à segunda barreira: dificultosa aplicação real.

3. Arquitetura de referência

A arquitetura visa a prover uma referência que evite o aprisionamento tecnológico e favoreça a governança da aplicação comercial ou de dados privados. Do ponto de vista técnico, pretende-se evitar as duas barreiras: i) forte acoplamento (FA) e ii) riscos à governança de dados (GD). Em relação ao FA, o desenvolvimento de experiências fechadas tendem a refletir, de acordo com [Victorino and Bräscher 2009], "uma aplicação estreitamente inter-relacionada em função e forma", trazendo prejuízo à interoperabilidade - ou compartilhamento - do serviço com outras ferramentas que poderiam apoiar o processo de seleção de modelos. Em relação à GD, nota-se uma falta de apoio para um ambiente colaborativo para a tomada de decisão e a garantia de um *compliance* comercial.

Os insumos gerados no visualizador de dados enseja descobertas, embasa decisões para interrupção antecipada da execução, permite definir métricas complexas durante o treinamento do modelo de ML, bem como realizar consultas históricas com SQL e compartilhar o progresso da performance do modelo com pares e *stakeholders*. Assim, o objetivo desta arquitetura é oferecer uma experiência aberta em termos de desenvolvimento, ou seja, a coleta e a disposição gráfica são independentes. A figura 1 representa a arquitetura a ser utilizada como referência em um projeto de ML, com os seus respectivos componentes e a comunicação entre eles. São estes:

Figura 1. Arquitetura genérica do experimento

1. **Ambiente de ML**. Representa o ambiente produtivo de ML. Assim, neste primeiro componente, o código de ML é executado.
2. **API**. Representa o ambiente onde é criada e curada a API, que será requisitada pelo Ambiente de ML. Assim, neste segundo componente, é definido o *endpoint*.

3. **RPC**. Representa a chamada de procedimento remoto orientada a evento (API). Assim, neste terceiro componente, é feito o tratamento dos parâmetros da requisição da API para armazenamento.
4. **Armazenamento e consulta de dados**. Representa o ambiente onde os dados tratados serão armazenados e consultados via SQL. Assim, neste quarto componente, armazena-se e consulta-se os metadados de treinamento do modelo de ML.
5. **Visualizador de dados**. Representa a ferramenta de disposição gráfica com base em consultas SQL. Assim, neste quinto componente, é feita a disposição gráfica dos metadados em tempo de execução em caráter analítico.

Além da compreensão dos componentes acima, a forma de comunicação entre os mesmos é fundamental para que seja possível atingir o objetivo do experimento. Para tanto, são estas as formas de comunicação ensejadas na figura 1:

- **C1**. A comunicação entre os componentes 1 e 2 é feita por meio de uma requisição HTTP (POST). Os parâmetros da requisição (body), leia-se metadados gerados pela sobrescrita dos métodos de *callback* do *framework* de ML, são serializados em formato JSON e enviados no POST. Neste cenário, considerando uma requisição síncrona, é importante que o componente 2 responda rapidamente o POST, pois esta é uma nova borda de latência para o treinamento de ML.
- **C2**. A comunicação entre os componentes 2 e 3 é feita via API. Assim, todo o escopo da requisição é enviado como parâmetro ao código presente na RPC. Neste cenário, enseja-se a necessidade de escolha de componentes que suportem um incremento do escopo da requisição (payload) sem perda de performance.
- **C3**. A comunicação entre os componentes 3 e 4 é feita por um pacote da linguagem de programação em questão que se conecte ao quarto componente (JDBC, SQLAlchemy e etc). Desta forma, estabelece-se a conexão e invoca-se uma operação de inserção em SQL para armazenamento de um registro.
- **C4**. A comunicação entre os componentes 4 e 5 é feita de forma semelhante à C3. A maioria dos visualizadores de dados utilizam um ORM (por exemplo, SQLAlchemy) que se conecta aos principais bancos de dados da indústria. Neste cenário, enseja-se a escolha de um componente 4 escalável que suporte picos de cargas de trabalho de ingestão e consumo de dados concorrentes.

4. Caso de uso

Para explorar a arquitetura de referência, a figura 2 representa uma instanciação visando à análise dos dados pelo humano, complementando a automação do ambiente de ML. Os artefatos exemplares podem ser avaliados no repositório do GitHub[1]. Os respectivos componentes escolhidos são:

Figura 2. Arquitetura específica do experimento

1. **Colab**. Representa o Google Colab. O uso básico é gratuito. No experimento, foi utilizado um algoritmo de rede neural convolucional com TensorFlow (Keras)[2].

[1]https://github.com/dfilipeaugusto/sample_sbbd_2023_artifacts. Acesso em 14/08/2023.
[2]https://keras.io/examples/vision/mnist_convnet/. Acesso em 24/06/2023.

A utilização do Colab em sua versão básica cumpre seu papel no caso de uso apresentado, mas não suporta um ambiente de ML produtivo em escala.

2. **AWS API Gateway**. Serviço gerenciado de API da Amazon Web Services (AWS). Devido à extensa rede global, a principal vantagem é o alcance em escala com uma baixa latência para as requisições efetuadas.
3. **AWS Lambda**. Serviço gerenciado de computação sem servidor para execução de código Python. Reconhecido por atender sistemas financeiros transacionais, o serviço permite atender cargas de trabalho imprevisíveis com baixo tempo de resposta e sem gerenciamento de infraestrutura.
4. **Snowflake**. A escolha deste componente reside sobre a forma de processamento de consultas. O Snowflake utiliza clusters de computação MPP independentes - sem compartilhamento de recursos computacionais - para processamento de consultas. Isso garante uma escalabilidade horizontal do componente 4.
5. **Preset**. Serviço gerenciado do Apache Superset com uso gratuito para até 5 usuários. A camada de visualização é uma das aplicações suportadas pela arquitetura de referência, por isso foi escolhido o Superset, que é notório por ser uma grande ferramenta de exploração e visualização de dados em contexto comercial.

É importante salientar que, com exceção do primeiro componente, os demais têm alta maturidade no mercado de computação comercial. Além disso, todas as formas de comunicação utilizam padrões bem estabelecidos (canônicos), que favorecem a substituição individual dos componentes. Assim, a figura 2 representa apenas uma sugestão de viabilidade.

Sob o aspecto de DevOps, é esperado salientar que o experimento não seguiu de acordo com o seu conjunto de práticas e ferramentas. Entretanto, a escolha dos componentes em questão seguiu critérios de maturidade, ou seja, a arquitetura ensejada pode atender prontamente demandas como controle de acesso baseado em função (RBAC), Infraestrutura como Código (IaC), controle de versionamento (Git) e etc.

O experimento permitiu acompanhar métricas em tempo de execução. O quarto componente armazenou os *logs* em um esquema flexível (tipo semiestruturado). A figura 3 mostra dois gráficos - alimentados por consultas SQL geradas dinamicamente pelo quinto componente - que podem ser enriquecidos com quaisquer outros metadados para além de acurácia e perda, bem como outros tipos de gráficos.

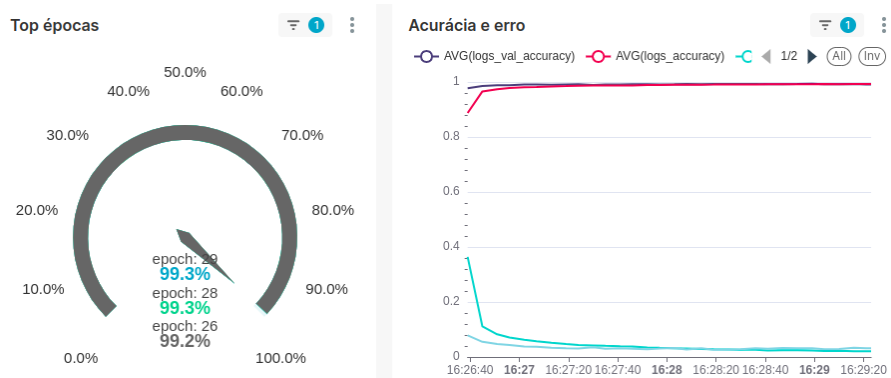

Figura 3. Gráficos dos tipos medidor e linha com acurácia e perda

5. Considerações finais

A crescente demanda por dados na indústria para a tomada de decisão também contempla algoritmos de ML. Em escala comercial, treinamentos de ML são processos custosos operacional e financeiramente. O experimento de instanciação mostrou a factibilidade de um serviço modular, interoperável e com governança de dados, sendo atingidos, respectivamente, pela capacidade de personalização e expansão de funcionalidades no projeto de ML, pela comunicação canônica entre os componentes e pela maturidade dos componentes tecnológicos frente ao leque disponível igualmente maduro. Ao seguir essa referência, caminha-se na direção de uma padronização na invocação de recursos analíticos de apoio ao humano de forma complementar aos ambientes produtivos de base de ML.

Agradecimentos

Trabalho realizado com o apoio das agências de fomento CNPq e FAPERJ.

Referências

De Bie, T., De Raedt, L., Hernández-Orallo, J., Hoos, H. H., Smyth, P., and Williams, C. K. (2022). Automating data science. *Communications of the ACM*, 65(3):76–87. ACM New York, NY, USA.

Gil, Y., Honaker, J., Gupta, S., Ma, Y., D'Orazio, V., Garijo, D., Gadewar, S., Yang, Q., and Jahanshad, N. (2019). Towards human-guided machine learning. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pages 614–624.

Kumar, A., McCann, R., Naughton, J., and Patel, J. M. (2016). Model selection management systems: The next frontier of advanced analytics. *ACM SIGMOD Record*, 44(4):17–22. ACM New York, NY, USA.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444. Publisher: Nature Publishing Group UK London.

Lee, D. and Macke, S. (2020). A Human-in-the-loop Perspective on AutoML: Milestones and the Road Ahead. *IEEE Data Engineering Bulletin*. National Science Foundation. NSF-PAR ID: 10161752.

Pang, B., Nijkamp, E., and Wu, Y. N. (2020). Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics*, 45(2):227–248. SAGE Publications, Los Angeles, CA.

Schelter, S., Boese, J.-H., Kirschnick, J., Klein, T., and Seufert, S. (2017). Automatically tracking metadata and provenance of machine learning experiments. In *NeurIPS 2017*, pages 27–29.

Schlegel, M. and Sattler, K.-U. (2023). Management of Machine Learning Lifecycle Artifacts: A Survey. *SIGMOD Record*, 51(4).

Spinner, T., Schlegel, U., Schäfer, H., and El-Assady, M. (2020). explAIner: A visual analytics framework for interactive and explainable machine learning. *IEEE transactions on visualization and computer graphics*, 26(1):1064–1074. Publisher: IEEE.

Victorino, M. and Bräscher, M. (2009). Organização da informação e do conhecimento, engenharia de software e arquitetura orientada a serviços: uma abordagem holística para o desenvolvimento de sistemas de informação computadorizados. *Revista de Ciência da Informação*, 10(3).

Wang, J., Liu, S., and Zhang, W. (2023). Visual Analytics For Machine Learning: A Data Perspective Survey. *arXiv e-prints*, page arXiv:2307.07712.

Yuan, J., Chen, C., Yang, W., Liu, M., Xia, J., and Liu, S. (2021). A survey of visual analytics techniques for machine learning. *Computational Visual Media*, 7(1):3–36. Publisher: Springer.