Universidade Federal do Rio Grande do Sul

Professor: Nielsen Rechia

MLOPS: Diabetes Detection Project

Aluno: Nícolas Cendron Fernandes

Porto Alegre 21 de Abril de 2025

1. Introdução

O presente trabalho tem como objetivo aplicar práticas MLOPs para construir um pipeline completo de Machine Learning, desde a obtenção e preparação dos dados até o deploy do modelo e seu monitoramento em produção.

Como primeira experiência do autor com práticas MLOPs foi construída uma versão que visa ser o mais simples e escalável possível para que fosse simples pivotar caso necessário. Cada uma das etapas do pipeline foi construída de forma independente, a fim de cada módulo ser passível ser substituido com o minimo impacto.

A principal parametrização existente é a escolha do modelo a ser executado, tomando o cuidado para que o pipeline pudesse ser facilmente convertido rapidamente para diferentes datasets apenas com mudanças de Strings e Exploração/pré-processamento.

2. Escolha do Dataset e Definição do Problema

Inicialmente foi escolhido um dataset de detecção de Spam, mas após a etapa de treinamento o autor não estava conectado ao dataset. Em conversa com familiares, um familiar próximo comentou que estava em investigação de diabetes, uma doença crônica com milhões de vítimas no mundo todo, o que o motivou a pivotar de dataset, e testar o quão fácil estava adaptar a arquitetura.

Por isso, para este projeto foi escolhido o Dataset “Diabetes Health Indicators” baseado na pesquisa Behavioral Risk Factor Surveillance System (BRFSS) realizado por telefone em 2015 pelo Centro de Controle de Doenças (CDC) dos Estados Unidos. Sendo realizada anuallmente desde 2014 essa pesquisa busca auxiliar no combate de uma das doenças crônicas com maiores vitimas no mundo.

Para a realização deste trabalho foi eleito o recorte diabetes_012 que é uma releitura dos dados originais, que passou por pré-processamento, contando com 253.680 respostas, cada um com 21 features relacionadas a hábitos e perfil dos pacientes (ex.: idade, IMC, histórico de hipertensão, colesterol, atividade física), podendo ser classificados em 3 possíveis grupos: 0 - Sem Diabetes (ou apenas durante a gravidez), 1 pré-diabetes e 2 para Diabetes. O dataset possui um forte desbalanço de classe, com a maior parte dos dados para pacientes saudáveis.

Para fins deste trabalho foram descartados os registros classificado como pré-diabetes, transformando o problema em classificação binária entre 0 - Sem Diabetes e 1- Com Diabetes.

3. Metodologia e Ferramentas Utilizadas

Para este projeto foi escolhida a linguagem Python em sua versão 3.10 para fins de compatibilidade entre as bibliotecas, sendo utilizada a ferramenta pyenv-win para gerenciamento de ambientes python.

Foi utilizado GitHub para controle de versão e documentação.

Para rastreamento de experimentos foi utilizado MLFlow rodando localmente, com monitoramento de drift através da biblioteca Evidently AI.

Por ultimo foi utilizado Docker local e FastAPI para deploy e uso para teste de inferência.

4. Desenvolvimento do Pipeline MLOPs

4.1 Exploração e Pré-processamento de dados

Em um primeiro momento, foi salvo o dataset original em CSV, e rodado um script o qual gerava dois novos dataset “1” e “2”, o primeiro contendo os primeiros 70% de registros e o segundo 30%. Isso foi feito para depois ser possível simular um pipeline com 2 momentos de treinamento, um inicial e outro ao chegar novos dados.

Em um primeiro momento foi realizada uma análise completa do CSV original, a fim de verificar o desbalanceamento de classes, descarte da classe de pré-diabetes.

Não foram identificados dados faltantes, mas igual foi implementado tratamento. Foram testadas alguns tratamentos de Features como remoção de outliers e combinação de features, controlados por flags FIX_AGE e FIX_BMI, ambos mantidos desativados na versão final.

Não foi executado scaling das variáveis por não serem necessários para Random Forests e XGBoost.

Como não era o foco principal do trabalho, foi executado apenas o básico.

4.2 Treinamento e Avaliação de Modelos

Conforme requisito, foram implementados dois de algoritmos de classificação: Random Forest e XGBoost. Em determinado momento foi implementada Regressão Logistica mas se mostrou pouco promissor e foi substituido pelo XGBoost.

Para lidar com o desbalanceamento de classes foi aplicada a técnica SMOTE

Foi utilizado MLFlow para rastrear experimentos, parâmetros e métricas (AUC, Acurácia, recall, F1 e precisão) além dos artefatos

Foram comparados os modelos com pela métrica AUC, visando sempre manter em produção o com maior AUC. As demais métricas foram utilizadas para auxiliar na escolha dos melhores parâmetros para os modelos.

4.3 Versionamento e Armazenamento de Modelos

A cada execução do pipeline era treinado o modelo para cada conjunto de dados, registrando suas métricas e artefatos no MLflow Model Registry, e caso apresenta-se melhor AUC que o atual em produção ele era automaticamente eleito como novo modelo de produção.

Essa abordagem é ingênua mas visa transitar rapidamente entre os diversos estados do pipeline várias vezes por ciclo de desenvolvimento. Em um projeto real seria importante testes, analise de outras métricas e demais cuidados para tomar uma decisão informada sobre a real viabilidade do novo modelo.

4.4 Implantação do Modelo via API e Conteinerização

O autor teve um pouco de dificuldade em conectar os artefatos que estavam no disco local da máquina, acessiveis através do MLflow Model Registry e o Docker, onde o caminho apresentado pelo MLFlow era o caminho absoluto.

Para fins deste trabalho foi eleita uma abordagem ingenua, onde foi criado um script fetch_production_model.py que obtem o modelo da trilha de produção, salva no disco com a biblioteca joblib, garantindo que estará disponivel na root do container do docker, em detrimento de uma abordagem cloud que seria utilizada em projetos maiores.

Através do docker, foi carregado um container local rodando uma API do FastAPI, que em seu startup carrega o modelo de produção utilizando joblib expõe uma rota *predict* que recebe um JSON com as features nos nomes e formatos esperados e retorna se foi detectado risco de diabetes ou não.

4.5 Monitoramento e Estratégia de Re-treinamento

Implementação de monitoramento de drift de dados com Evidently AI com a possibilidade de gerar relatórios a cada novo conjunto de dados. No dataset de Spam foi detectado drift, mas no caso do dataset de Diabetes o Drift não foi detectado, o que levou o autor a criar uma flag para forçar o reconhecimento de Drift para fins didáticos.

A estratégia de re-treinamento automático ficou definido quando for detectado drift entre os novos dados e o dataset utilizado anteriormente. Da forma que foi pensado, a cada novo pacote de dados, este é somado aos dados conhecidos e os modelos são treinandos utilizando todos os dados conhecidos.

Como o fim era didático a cada execução de pipeline.py é como se o treinamento começasse do zero, mas em um projeto real existiria um script

agendado para realizar esse controle, e não necessariamente seriam usados todo universo de dados.

4.6 Conteinerização e Documentação

5. Resultados e Métricas

Neste trabalho foi dada maior enfase na compreensão do modelo MLOps através do MLFlow do que as métricas obtidas através dos modelos, uma vez que detalhes de pré-processamento e treinamento foram explorados em disciplinas anteriores, porém é sempre importante olharmos os números.

Trago os resultados obtidos após o ciclo de treinamento, simulando a detecção de drift nos dados, o que implica no treinamento com todos os dados.

Através da análise de resultados é possivel perceber baixa precisão e acurácia, e AUC moderado-baixo, implicando um resultado baixo para produção mas superior ao acaso, indicando potencial de as features serem relevantes ao problema.

- **Random Forest:**
  - AUC = 0.7314
  - F1 = 0.4403
  - Precisão = 0.3049
  - Acurácia = 0.6895
- **XGBoost:**
  - AUC = 0.7401
  - F1 = 0.4416
  - Recall = 0.8355
  - Precisão = 0.3001
  - Acurácia = 0.6741
- Modelo final selecionado: **XGBoost** (maior AUC) seguindo a abordagem definida anteriormente.

6. Fluxo Completo do Pipeline

- Ingestão e pré-processamento de dados
- Treinamento e experimentação (MLFlow)
- Registro dos modelos
- Seleção de modelo para prdodução
- Deploy via API (FastAPI + Docker)
- Recebimento de novos dados
    - Verificação de Drift
        - Caso detectado Drift, é realizado mais um ciclo da pipeline
        - Caso não seja detectado, mantem-se o modelo atual.

7. Considerações Finais

Como resultado principal da execução deste trabalho trago o aprendizado de como levar todos os conhecimentos adquiridos nas disciplinas anteriores, geralmente em google colabs, para o mundo real. Como partir dos dados brutos até um modelo em produção para mim foi uma experiência indescritível, e por mais que eu ainda esteja engatinhando na parte de MLOps, sinto após a execução deste trabalho uma integração muito maior de todos os passos, sua importância e suas peculiaridades e desafios.

Mesmo que a pipeline que construí não seja *production ready*, sinto que foi um importante passo na minha carreira e desenvolvimento, e por isso agradeço ao professor e monitores.

8. Referências

MLflow Documentation: https://mlflow.org

Evidently AI Documentation: https://evidently.ai

Kaggle Dataset: Diabetes Health Indicators (https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset)

FastAPI Documentation: https://fastapi.tiangolo.com

Pyenv-win: https://pyenv-win.github.io/pyenv-win/