**Relatório Trabalho MLOps - Everton Vaszelewski**

## Repositório do GitHub:
https://github.com/Vaszelewski/Trabalho_ECD15_MLOps

## Escolha do dataset e problema abordado
Loan Default Prediction Dataset
https://www.kaggle.com/datasets/nikhil1e9/loan-default

O Loan Default Prediction Dataset é um dataset disponível no Kaggle (link acima) que contém informações sobre empréstimos concedidos para clientes, incluindo informações demográficas, financeiras, pessoais (como idade, escolaridade) e histórico de crédito. O problema abordado foi desenvolver e identificar quais clientes têm maior probabilidade de inadimplência, ou seja, de não conseguirem pagar o empréstimo.

## Metodologia e ferramentas utilizadas
Metodologia:
A metodologia foi de dois modelos, Random Forest Classifier e XGBoost, desenvolver um modelo cada capaz de identificar quais clientes têm maior probabilidade de inadimplência.

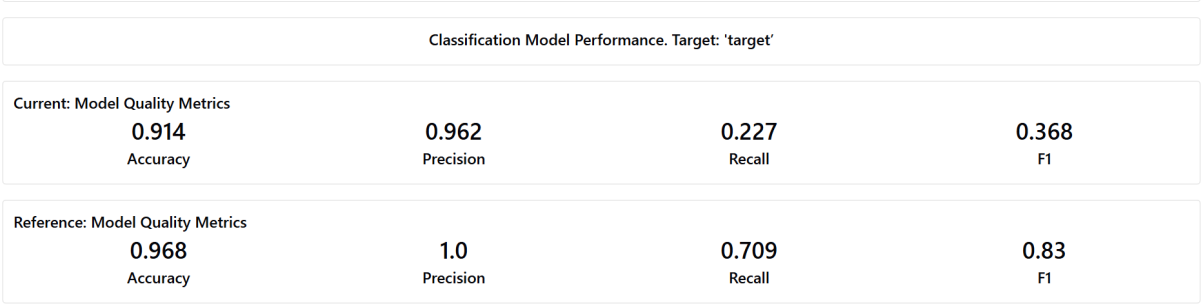
Ferramentas:
- Linguagem: Python 3.10
- MLflow (para rastreamento e versionamento de modelos)
- Evidently AI (para monitoramento de drift)
- FastAPI/Flask (para disponibilização do modelo via API, (ex. usando Mlflow)
- GitHub (para controle de versão)

## Resultados e métricas dos modelos

Para análise do problema proposto, foram utilizados dois modelos:
**RandomForestGridSearch:** que se trata da junção de dois conceitos de machine learning:
- Random Forest: algoritmo de aprendizado supervisionado baseado em múltiplas árvores de decisão.
- Grid Search: método de otimização que testa diferentes combinações de hiperparâmetros para encontrar a melhor configuração para um modelo.
  Para o meu projeto, os parâmetros utilizados foram:
  - n_estimators: Número de árvores na floresta.
  - max_depth: Profundidade máxima das árvores.
  - min_samples_split: Número mínimo de amostras necessárias para dividir um nó.

O melhor modelo com métrica F1 gerado foi a Versão 8 com F1 = 0.102.
Durante as execuções e o versionamento, a média de acurácia ficou entre 0.86 e 0.88, a Precision entre 0.60 e 0.74 e Recall entre 0.23 e 0.53
Neste modelo ocorreu drift em 3 colunas e 0.167 (Share of Drifted Columns)

- Prediction
- Months Employed
- LoanTerm



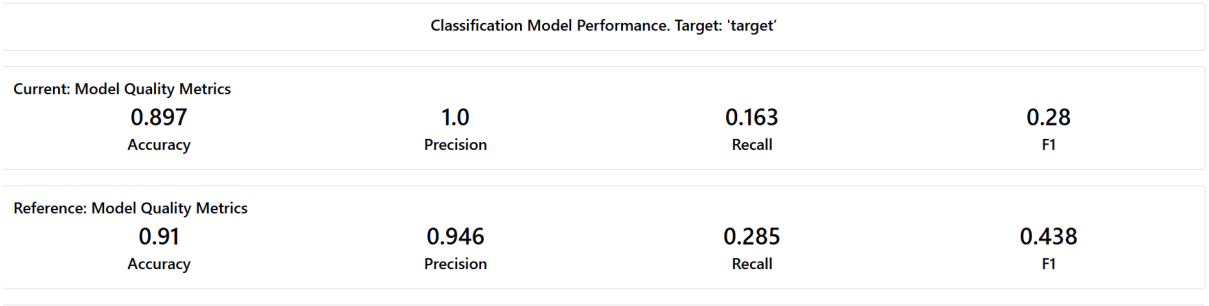

Quando comparamos as métricas de qualidade, é notável que houve uma queda de 5 a 6% entre a Referência e o Atual, mostrando que o modelo perdeu sua qualidade com as execuções. Inclusive sua precisão saiu de 100% para 96%.

**XGBoost:** treina o melhor modelo encontrado e registra os melhores parâmetros no MLflow. Ele também cria árvores sequencialmente, porém a cada nova árvore tenta corrigir os erros da anterior usando gradiente descendente, que é um método de otimização que ajusta os parâmetros do modelo para errar o mínim possível.
Para o meu projeto, os parâmetros utilizados foram:

- n_estimators: Número de árvores na floresta.
- max_depth: Profundidade máxima das árvores.
- learning_rate: velocidade de aprendizado.

O melhor modelo com métrica F1 foi a Versão 6 com F1 = 0.139.
Durante as execuções e o versionamento, a média de acurácia ficou entre 0.84 e 0.86, a Precision entre 0.0 e 0.74, demonstrando uma volatilidade muito grande, e e Recall entre 0.0 e 0.79
Neste modelo ocorreu drift em 3 colunas e 0.167 (Share of Drifted Columns)

- Prediction
- Months Employed
- LoanTerm




Quando comparamos as métricas de qualidade, a queda na Acurácia foi pouca, cerca de menos de 1% entre a Referência e o Atual, mostrando que o modelo se manteve estável, e a Precisão melhorou muito, chegando a 100%, mostrando resultados com tendências positivas.

## Fluxo completo do pipeline

**1 - Exploração e Pré-processamento dos Dados**

- Análise exploratória e tratamento de valores ausentes.

Normalização/Padronização dos dados.

**2 - Treinamento e Avaliação do Modelo**

- Implementação dos modelos Random Forest Classifier e XGBoost e comparação de métricas.
- Utilização do MLflow para rastrear experimentos.

**3 - Versionamento e Armazenamento do Modelo**

- Registro do modelo no MLflow Model Registry.

**4 - Implantação do Modelo**

- Construção de uma API com FastAPI ou Flask para servir previsões (mlflow).
- Deploy local.

**5 - Monitoramento e Re-treinamento**

- Implementação de monitoramento de drift de dados com Evidently AI.
- Definição de uma estratégia para re-treinamento automático do modelo.

**6 - Conteinerização e Documentação**

- Instruções de execução/documentação do pipeline no repositório.
- Não houve conteinerização.

## Considerações Finais

Os resultados do Random Forest Classiffier e do XGBoost podem ser comparados visto que são modelos baseado em árvores e podem ser aplicados para o mesmo tipo de problema, e no caso, classificação.

Quando comparados os valores de acurácia, precisão e recall, é observado que XGBoost teve um desempenho superior ao Random Forest, já que os valores de XGBoost melhoraram quando o drift foi aplicado, ao contrário do Random Forest.
A métrica F1 também foi superior em XGBoost, mostrando que o modelo se comportou melhor com os dados.

Como tem muitos registros no dataset, o Random Forest não foi muito rápido e eficaz, se mostrando inferior no geral.