Projeto Final de Curso - Eng. ML

Elisandro Murliky

https://github.com/elisandrom/infnet_ml

Engenharia de Machine Learning [24E1_3]

Agenda do Trabalho

O aluno deve preencher essa apresentação com os resultados da sua implementação do modelo. Os códigos devem ser disponibilizados em repositório próprio, público, para inspeção.

Essa apresentação é padronizada para que os alunos possam incluir os seus resultados, com figuras, tabelas e descrições sobre o projeto de curso. Os resultados aqui descritos serão confrontados com os códigos disponibilizados.

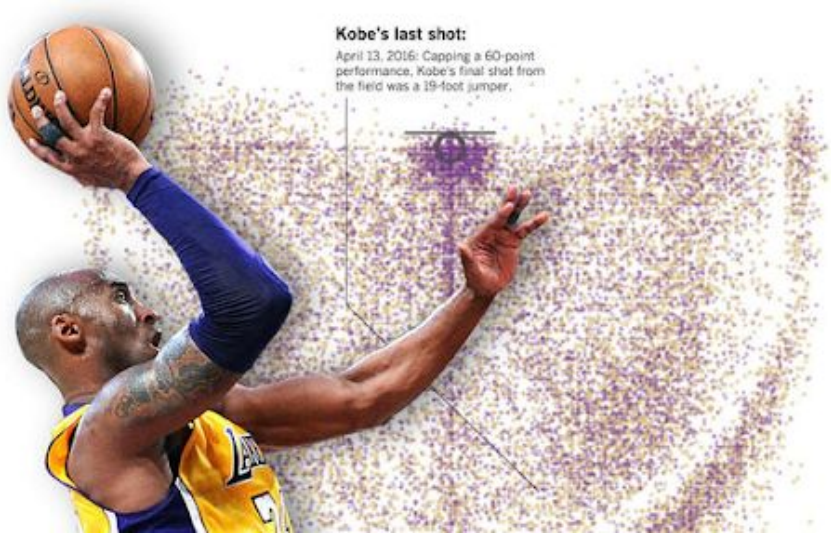
Roteiro

- Objetivo da modelagem
- Arquitetura da solução
  - Diagrama
  - Bibliotecas
  - Artefatos e Métricas
- Pipeline de processamento dos dados
  - Descrição dos dados
  - Análise Exploratória
  - Seleção base de teste
- Pipeline de Treinamento do Modelo
  - Validação Cruzada
  - Regressão Logística
  - Árvore de Decisão
  - Seleção, finalização e registro
- Aplicação do Modelo
  - Model as a Service localmente
  - Interface para aplicação na base de produção
  - Monitoramento do modelo

Objetivo da modelagem

Em homenagem ao jogador da NBA Kobe Bryant (falecido em 2020), foram disponibilizados os dados de 20 anos de arremessos, bem sucedidos ou não, e informações correlacionadas.

O objetivo desse estudo é aplicar técnicas de inteligência artificial para prever se um arremesso será convertido em pontos ou não.

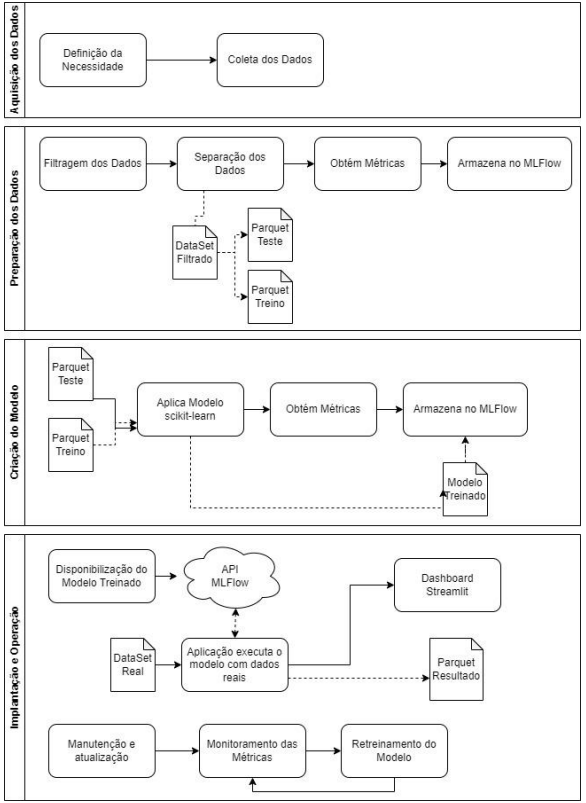


Arquitetura da Solução

Arquitetura da Solução

Diagrama

- **Aquisição dos Dados**
  - Definição da Necessidade
  - Coleta dos Dados
- **Preparação dos Dados**
  - Filtragem dos Dados
  - Separação dos Dados
  - Obtém Métricas
  - Armazena no MLFlow
- **Criação do Modelo**
  - Aplica o Modelo Scikit-Learn
  - Obtém Métricas
  - Armazena no MLFlow
- **Implantação e Operação**
  - Disponibilização do Modelo Treinado - API MLFlow
  - Execução do Dataset de PRD no Modelo Treinado
  - Armazenamento dos Resultados
  - Disponibilização de Dashboard no Streamlit
  - Monitoramento de Métricas
  - Retreinamento do Modelo



Arquitetura da Solução

Bibliotecas

Em suma, o MLFlow garante organização e acompanhamento, o PyCaret e o Scikit-Learn oferecem expertise no treinamento, e o Streamlit é a interface para a interação com o usuário.

- **Rastreamento de experimentos**
  - MLFlow possibilita a rastreabilidade de experimentos, o que permite aos desenvolvedores validarem as métricas, parâmetros e artefatos de cada experimento. Isto facilita a comparação de diferentes modelos e configurações, bem como o monitoramento das alterações de desempenho no ciclo de vida.

- **Funções de treinamento**
  - PyCaret e Scikit-Learn possuem várias funções de treinamento para machine learning, sendo elas: classificação, regressão e clusterização. Em especial o PyCaret que automatiza o processo de treinamento, pré-processamento, seleção dos hiperparâmetros de uma maneira muito simples e prática.

Arquitetura da Solução

- **Monitoramento da saúde do modelo**
  - Neste item se destaca o MLFlow que permite monitorar a saúde do modelo e registrar métricas relacionadas ao desempenho. Este mecanismo facilita a identificação de problemas de desempenho caso seja identificada uma queda de performance, garantindo que o modelo ainda seja relevante, possuindo um papel de gestor do ciclo de vida do projeto.

- **Atualização de modelo**
  - O MLFlow permite o gerenciamento de versões dos modelos e a aplicação de atualizações, permitindo aos desenvolvedores implantar novas versões e comparar o desempenho das mesmas.

- **Provisionamento (Deployment)**
  - Neste cenário, o Streamlit é o elo entre o usuário e o modelo, permite a criação de aplicativos web interativos para modelos já treinados de machine learning. Garante que os desenvolvedores criem interfaces de usuário amigáveis para seus modelos e as implantem facilmente em servidores ou plataformas de nuvem. Além disso, o MLFlow também oferece funcionalidades para implantar modelos em produção em diferentes plataformas como AWS, Azure e Kubernetes, facilitando o provisionamento e o monitoramento do modelo durante sua vida.

Arquitetura da Solução

Artefatos e Métricas

- **Aquisição dos Dados**
  - Dado a necessidade, é gerado o arquivo original que possui todos os arremessos do Kobe Bryant, nessa etapa sendo representado pela já separação entregue:
    - "Data/Raw/dataset_kobe_dev.parquet" = Representa os dados que devem ser treinados por modelos.
    - "Data/Raw/dataset_kobe_prd.parquet" = Representa os dados que devem ser processados no modelo final que tenha o melhor resultado ou o de classificação como solicitado no trabalho.

- **Preparação dos Dados**
  - Responsável por esse processo "Code/PreparacaoDados.py"
    - Geração do "Data/Processed/data_filtered.parquet" = Dataset filtrado e pré-processado com as regras exigidas.
    - Geração do "Data/Processed/base_train.parquet" = Dataset de treino para ser utilizado pelo modelo.
    - Geração do "Data/Processed/base_test.parquet" = Dataset de teste para ser utilizado pelo modelo.
    - Armazena no MLFlow na run "PreparacaoDados" as métricas como "base_treino_tamanho" e "base_teste_tamanho", juntamento com o parâmetro que informa o tamanho utilizado para base de teste denominado: "teste_percentual" .

- **Criação do Modelo**
  - Responsável por esse processo "Code/Treinamento.py"
    - Para cada modelo testado:
      - Armazena no MLFlow na run "Treinamento" as métricas do modelo em questão, sendo elas "log_loss" e "f1_score", a tag que informa o algoritmo utilizado "model" e por último o modelo treinado.
      - Geração do "Data/Modeling/rl_model_treined.pkl" e "Data/Modeling/dt_model_treined.pkl" que representam os arquivos pickle dos modelos de Regressão Logística e Árvore de Decisão.

- **Implantação e Operação**
  - Responsável por esse processo "Code/Aplicacao.py"
    - Disponibiliza o modelo da árvore de decisão através da API do MLFLow, sendo acessível pelo endpoint "/invocations".
    - Após a execução dos dados reais no modelo, é armazenado o artefato do resultado do processamento na run "PipelineAplicacao".
  - Responsável por esse processo "Code/Dashboard.py"
    - Disponibiliza um Dashboard no Streamlit para monitoramento da saúde e resultados do modelo.

Processamento de Dados

Pipeline de processamento dos dados

Descrição dos dados

Este dataset possui os dados disponibilizados de 20 anos de arremessos, bem sucedidos ou não, e informações correlacionadas do jogador Kobe Bryant durante sua carreira em vida na NBA.

Sobre o DataSet de Treinamento

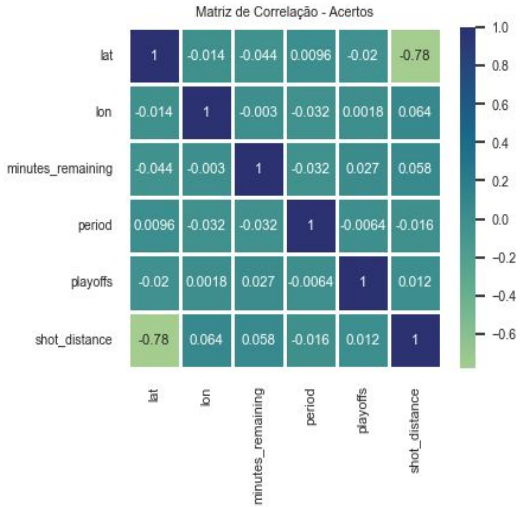
- **Total de Variáveis:** 25
- **Total de Registros Originais:** 24.271
- **Registros com "shot_made_flag" Não Preenchido:** 3.986
  - Estes registros foram desconsiderados no treinamento
- **Variáveis Utilizadas no Treinamento:** 6
  - lat: Tipo Contínua
  - lon: Tipo Contínua
  - minutes_remaining: Tipo Discreta
  - period: Tipo Discreta
  - playoffs: Tipo Categórica
  - shot_distance: Tipo Contínua

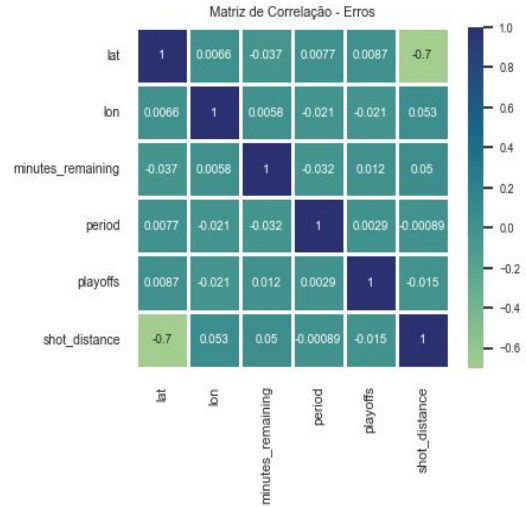
Pipeline de processamento dos dados

Análise Exploratória

Utilizando como referências as variáveis: “lat, lon, minutes_remaining, period, playoffs, shot_distance”, conforme demostrado no gráfico ao lado de Matriz de Correlação de Acertos e Erros, podemos identificar uma correlação entre a posição do jogador dentro da quadra, apontado por “lat e lon” e a distância do arremesso “shot_distance”, essas influenciando diretamente no sucesso ou não do arremesso, neste cenário, não considerando o tempo de partida já jogado ou em que fase da carreira ele se encontra.

Nas variáveis, “minutes_remaining e period” elas impactam o resultado, onde com o passar do tempo, a chance do jogador errar o arremesso aumentam.





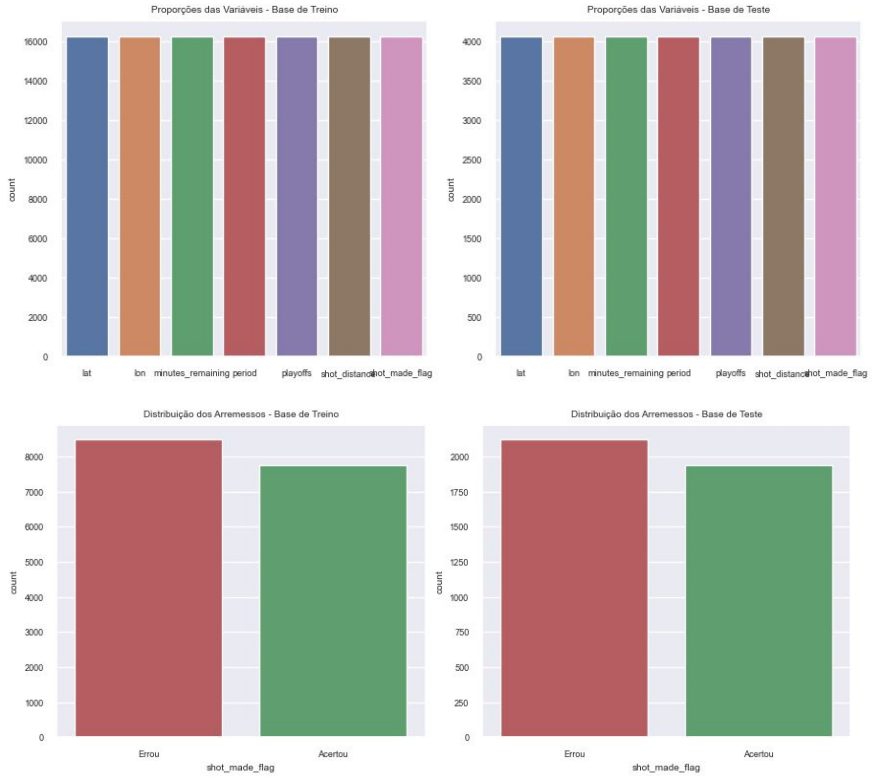
Pipeline de processamento dos dados

Seleção base de teste

Na divisão de treino e teste foi utilizado o critério padrão, sendo 80% treino e 20% teste, com estratificação para garantir que a proporção de cada classe seja preservada tanto no conjunto de treino quanto no conjunto de teste.

- **Total de Registro para Treino:** 16.228
- **Total de Registro para Teste:** 4.057

Nesta quantidade de registros, podemos variar de 20 a 30% na base de teste para validar a performance dos modelos.



Treinamento do Modelo

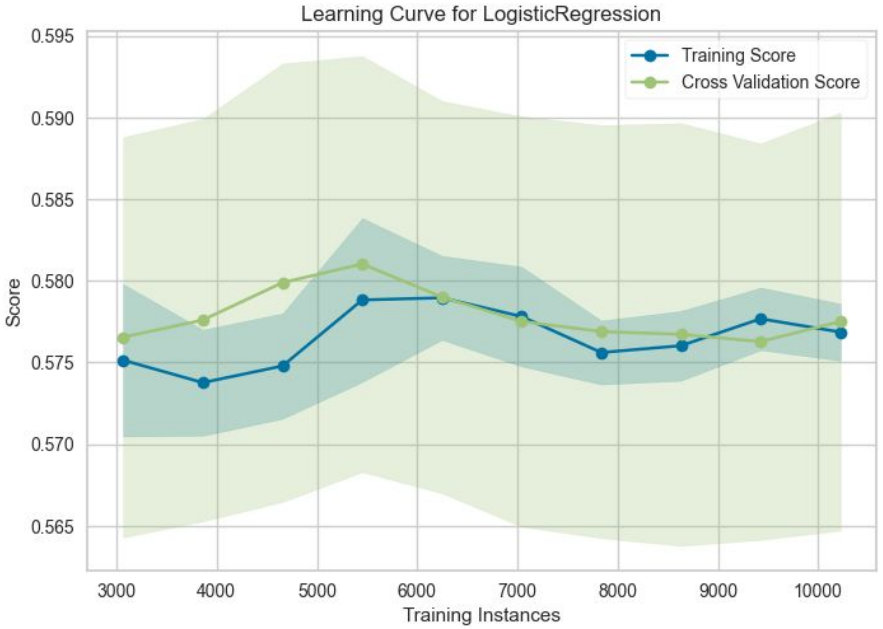
Pipeline de Treinamento do Modelo

Regressão Logística - Validação Cruzada

A validação cruzada é útil para comparar diferentes modelos garantindo que a análise não seja tendenciosa com relação aos dados de treinamento e de teste utilizados, ele fornece a estimativa geral do desempenho do modelo reduzindo o viés e a variância da avaliação do modelo.

Curva de Aprendizado: Mostra se o desempenho do modelo melhora à medida que o tamanho do conjunto de treinamento aumenta, ajudando a demonstrar se o modelo está sofrendo viés ou variância.

Curva de Validação: Ajuda a encontrar o melhor valor dos hiperparâmetros para o modelo, sendo ela o parâmetro C da regressão logística, variando o valor de C e analisando a métrica de desempenho, pode-se identificar o melhor valor para o parâmetro.



Pipeline de Processamento dos dados

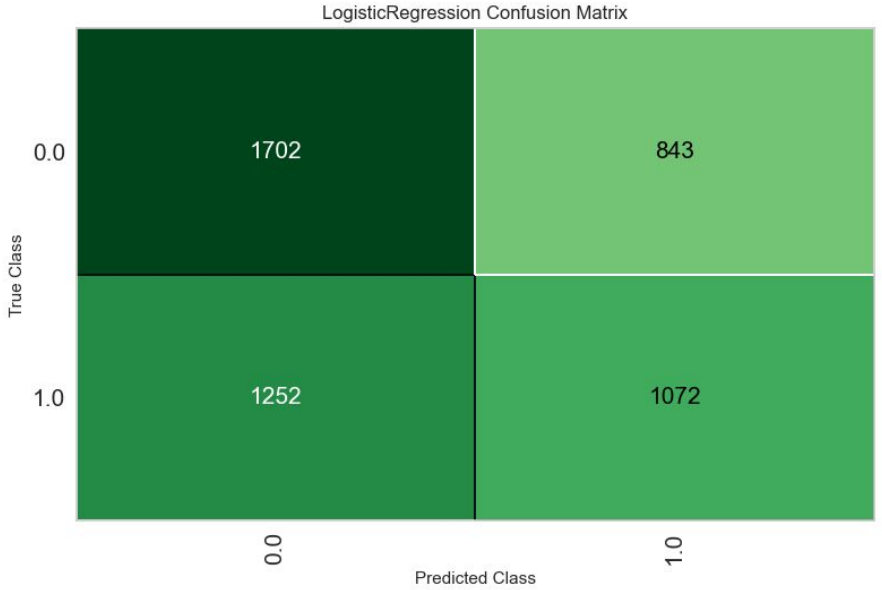
Regressão Logística - Classificação

Métricas:

- **Accuracy:** 0.5823
- **Recall:** 0.4891
- **Prec:** 0.5732
- **F1:** 0.5278

Interpretação:

- **Verdadeiros Positivos (1:1):** Corretamente classificados como positivos.
- **Verdadeiros Negativos (1:0):** Corretamente classificados como negativos.
- **Falsos Positivos (0:1):** Incorretamente classificados como positivos (geralmente erros graves).
- **Falsos Negativos (0:0):** Incorretamente classificados como negativos (geralmente erros menos graves).



Pipeline de Treinamento do Modelo

Árvore de Decisão - Validação Cruzada

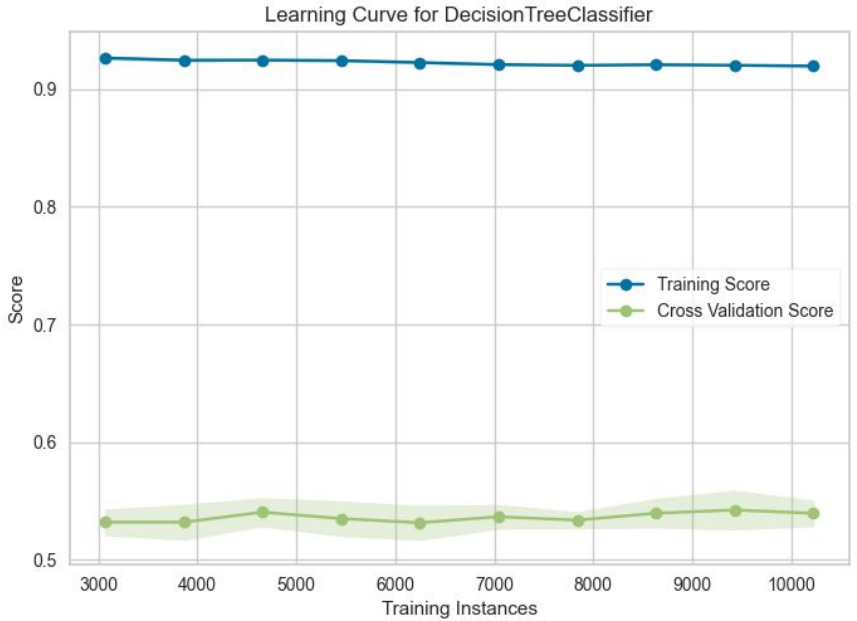
Interpretação da Curva de Aprendizado

Envolve em analisar como o desempenho do modelo muda à medida que o tamanho dos dados de treinamento aumenta.

Score = Erro médio de validação cruzada para cada iteração

Training Instances = Número de Iterações

O ideal é encontrar um modelo que a curva de validação seja baixa e a curva de treinamento seja próxima a ela. Nas análises é possível encontrar aspectos como a curva de aprendizado pode indicar se o modelo está sofrendo de viés elevado (underfitting) ou variância elevada (overfitting), a curva de aprendizado pode ajudar a determinar também o tamanho ideal dos dados para treinamento e assim obter um melhor desempenho do modelo.



Pipeline de Treinamento do Modelo

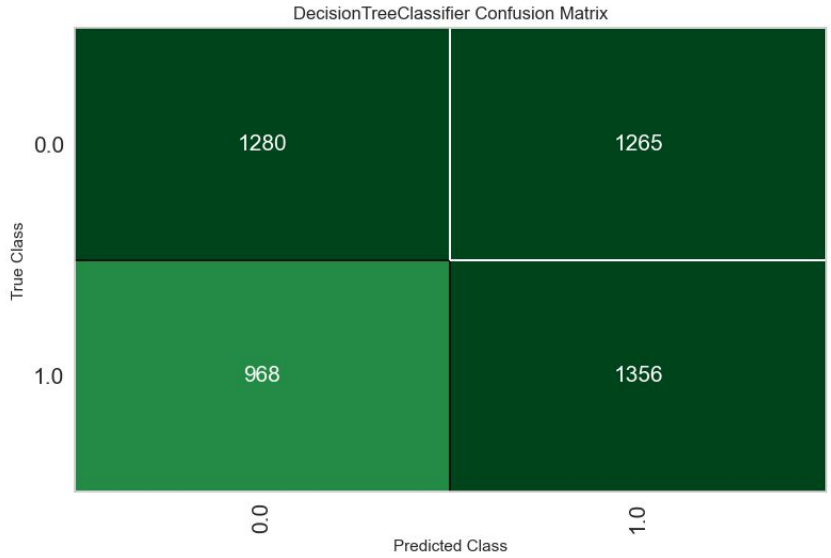
Árvore de Decisão - Classificação

Métricas:

- **Accuracy:** 0.5398
- **Recall:** 0.5800
- **Prec:** 0.5159
- **F1:** 0.5460

Interpretação:

- **Verdadeiros Positivos (1:1):** Corretamente classificados como positivos.
- **Verdadeiros Negativos (1:0):** Corretamente classificados como negativos.
- **Falsos Positivos (0:1):** Incorretamente classificados como positivos (geralmente erros graves).
- **Falsos Negativos (0:0):** Incorretamente classificados como negativos (geralmente erros menos graves)

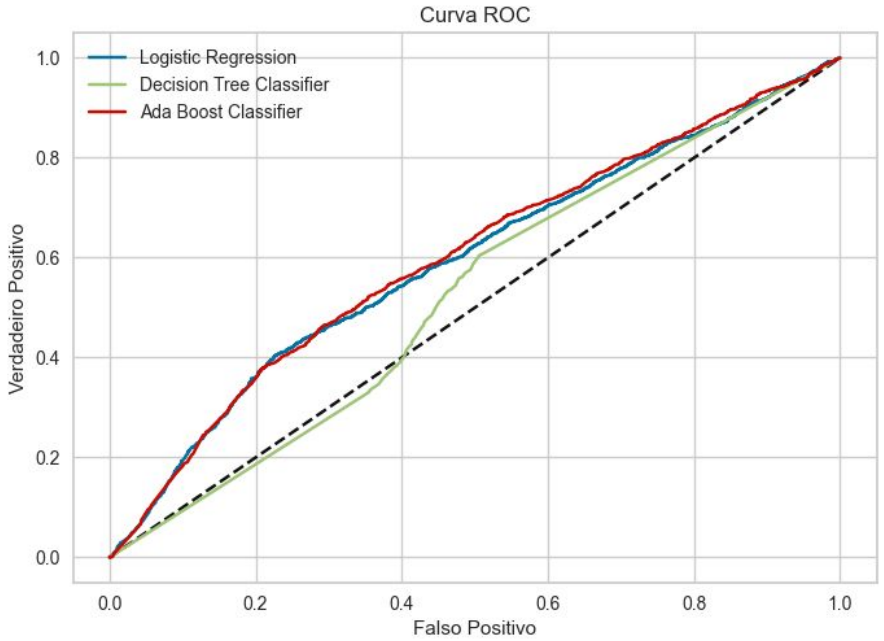


Pipeline de Treinamento do Modelo

Seleção, finalização e registro

No gráfico da Curva ROC apresentado ao lado, possuem dados dos modelos: Regressão Logística, Árvore de Decisão e Ada Boost.

Podemos verificar que o modelo “Ada Boost Classifier” é o mais confiável para prever os resultados dos arremessos.



Pipeline de processamento dos dados

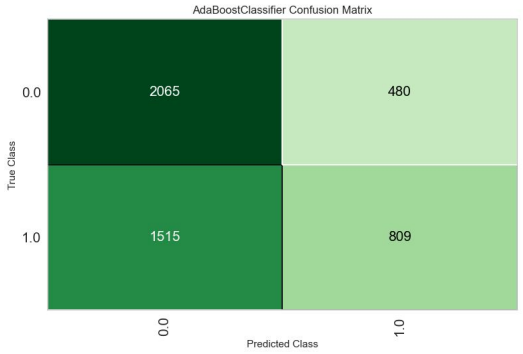
Regressão Logística - Validação Cruzada e Classificação

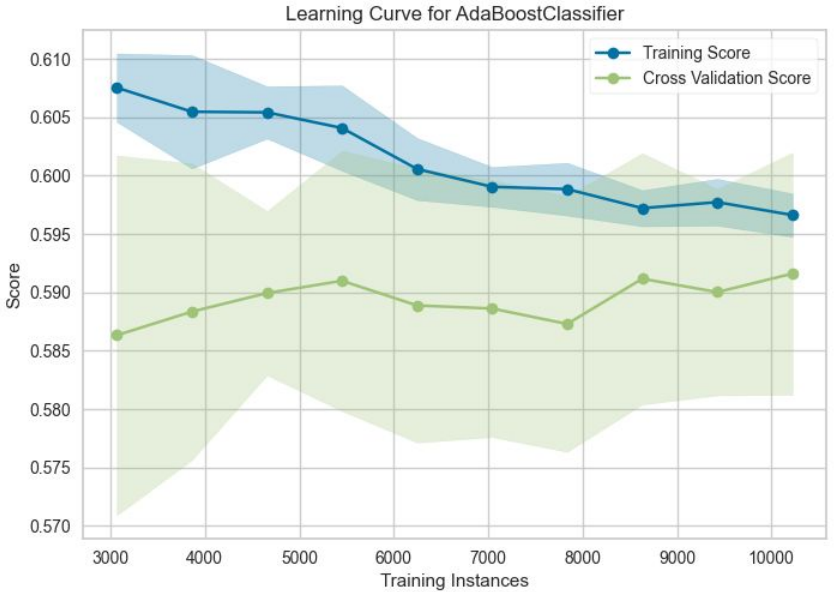
Para o melhor algoritmo: ***Ada Boost Classifier***

compare_models(n_select = 1, sort='Accuracy')

Métricas:

- **Accuracy:** 0.5915
- **Recall:** 0.3731
- **Prec:** 0.6203
- **F1:** 0.4657





Aplicação do Modelo

Pipeline de Aplicação do Modelo

Estratégia reativa:

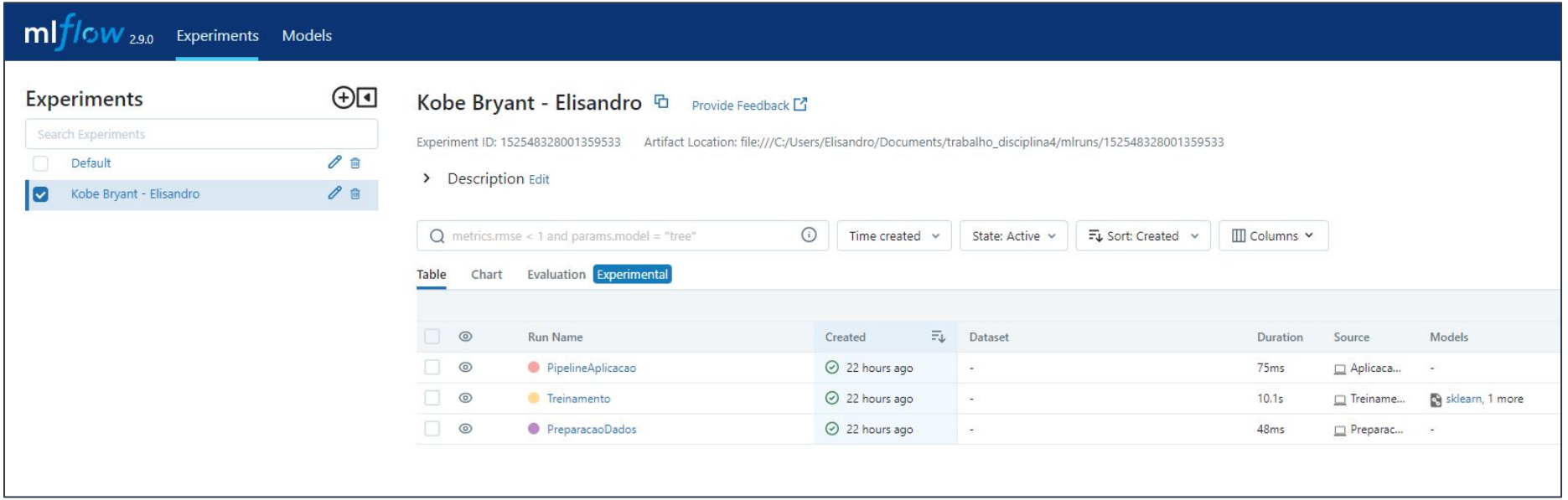
A estratégia reativa ocorre quando é identificado uma queda de performance, por exemplo: Aumento na taxa de erro, mudança dos dados, etc, neste cenário é necessário o retreino do modelo.

Estratégia preditiva:

A estratégia preditiva ocorre antes da identificação de queda de performance, desta forma o modelo é retreinado com novos dados sempre que possível, aumentando assim a sua taxa de acertos.

MLFlow

Estes são os runs que foram gerados na execução do projeto



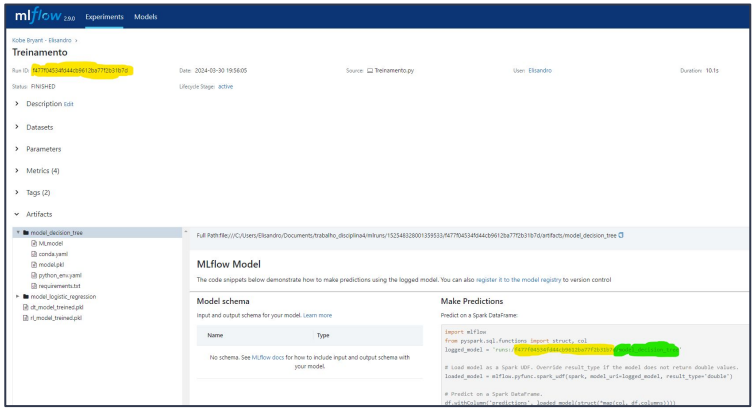
Modelo Servido em API

Configurando o modelo disponível em API foi utilizado o próprio recurso do MLFlow

Exemplo:

mlflow models serve -m "runs:/f477f04534fd44cb9612ba77f2b31b7d/model_decision_tree" --no-conda -p 8081

- “f477f04534fd44cb9612ba77f2b31b7d” representa o run id de onde o modelo foi salvo treinado no MLFlow
- “model_decision_tree” representa o nome do arquivo do modelo salvo

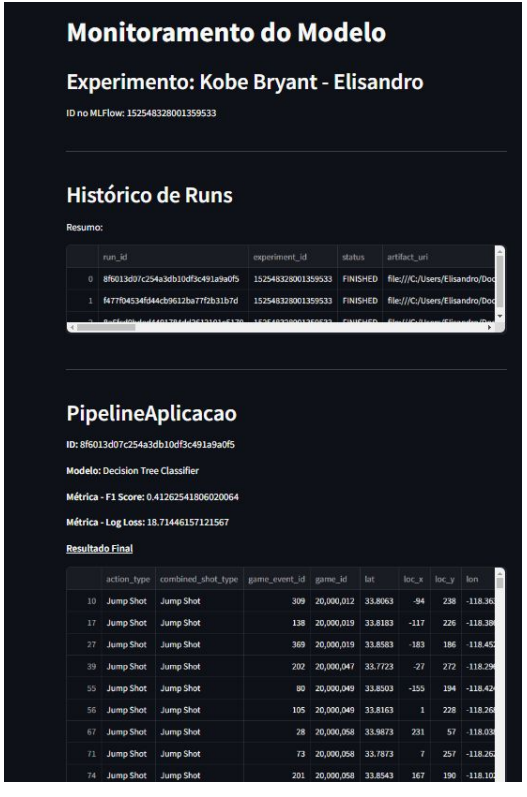


Para realizar uma solicitação na API via Python, foi utilizado o trecho abaixo:

```
#Faz a requisição localmente para a API do MLFlow
responseMLFlow = requests.post(
    'http://127.0.0.1:8081/invocations',
    headers={'Content-Type':'application/json'},
    json={
        "dataframe_split":
        {
            "columns": df.columns.tolist(),
            "data": df.values.tolist()
        }
    }
)
```

Dashboard

Para o desenvolvimento do dashboard foi utilizado o Streamlit que é o elo entre o usuário e o modelo, permite a criação de aplicativos web interativos para modelos já treinados de machine learning. Garante que os desenvolvedores criem interfaces de usuário amigáveis para seus modelos e as implantem facilmente em servidores ou plataformas de nuvem. Além disso, o MLFlow também oferece funcionalidades para implantar modelos em produção em diferentes plataformas como AWS, Azure e Kubernetes, facilitando o provisionamento e o monitoramento do modelo durante sua vida.

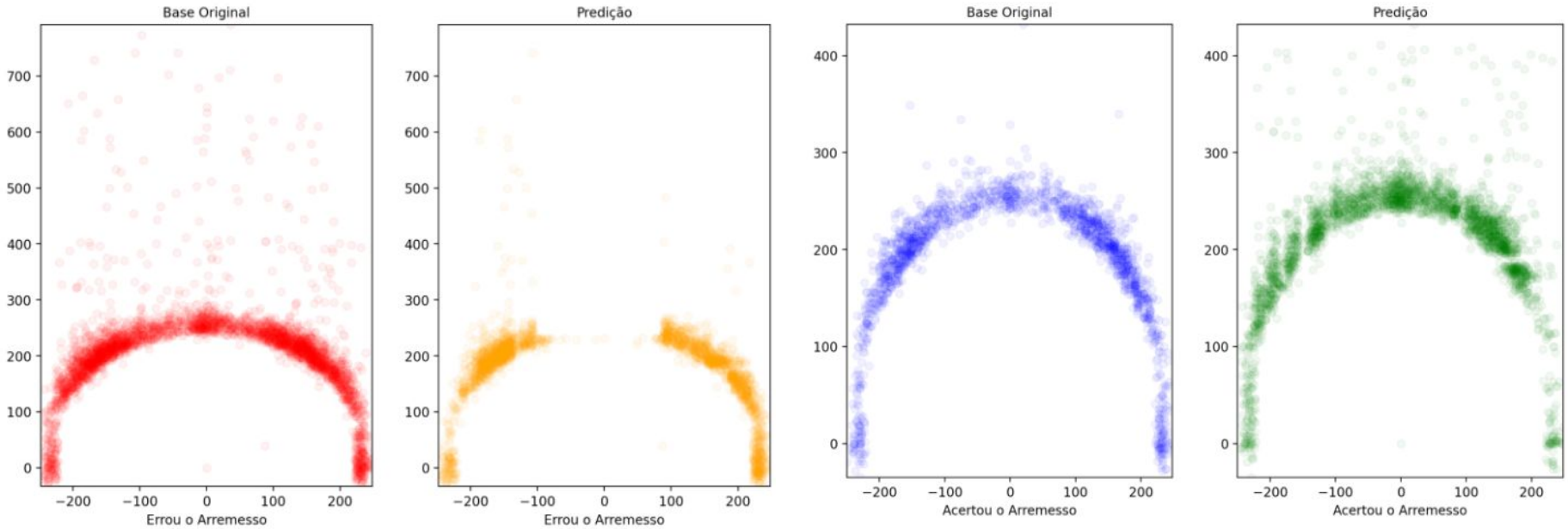


Resultado do Modelo Treinado

O modelo não é aderente a nova base, o modelo foi treinado usando uma base de dados que possui somente arremessos de 2 pontos, na base final, todos os dados da base são de arremessos de 3 pontos, portanto, o modelo erra nas predições dos arremessos já que a distância da cesta para o arremesso muda consideravelmente, uma forma de visualizar a mudança repentina da métrica é comparar o run de “Treinamento” com log loss de 16.107234203240136 contra o “PipelineAplicacao” com log loss de 18.71446157121567, ficando claro que com os dados de 3 pontos ficou pior no modelo.

Resultado do Modelo Treinado

Demonstração dos Arremessos - Árvore de Decisão



Total de Registros Processados da Base de Produção: 5.412

Respostas das Questões

Respostas

Todas as respostas das questões do trabalho se encontram no GitHub, diretório “Docs”, arquivo “RespostasQuestoes.md”

Link do Repositório no GitHub:

https://github.com/elisandrom/infnet_ml

Obrigado!