

1. A solução criada nesse projeto deve ser disponibilizada em repositório git e disponibilizada em servidor de repositórios (Github (recomendado), Bitbucket ou Gitlab). O projeto deve obedecer o Framework TDSP da Microsoft. Todos os artefatos produzidos deverão conter informações referentes a esse projeto (não serão aceitos documentos vazios ou fora de contexto). Escreva o link para seu repositório.

Configuração do Repositório do Projeto

Este documento descreve a configuração do repositório do projeto de machine learning para prever os arremessos de Kobe Bryant. O repositório foi criado em uma plataforma de controle de versão e hospedagem de código, como GitHub, Bitbucket ou Gitlab, conforme recomendado.

Detalhes do Repositório

- **Nome do Projeto:** Projeto Final de Curso - Eng. ML INFNET
- **Plataforma de Hospedagem:** GitHub
- **Link para o Repositório:** [\[Link para o Repositório\]](#)
- **Framework TDSP:** O projeto foi estruturado e gerenciado seguindo as diretrizes do Team Data Science Process (TDSP) da Microsoft.

Estrutura do Repositório

O repositório do projeto segue uma estrutura organizada de diretórios e arquivos, conforme recomendado pelo TDSP:

- **Data:** Armazena os dados brutos e processados utilizados no projeto.
- **Docs:** Contém a documentação do projeto, incluindo documentos de entendimento do negócio, carta do projeto e descrições dos artefatos.

- **Code:** Contém os scripts e códigos-fonte do projeto, incluindo aqueles relacionados à preparação de dados, treinamento de modelos e avaliação.
- **Notebooks:** Armazena os notebooks Jupyter utilizados para análise exploratória e desenvolvimento de modelos.
- **Models:** Contém os modelos treinados e quaisquer artefatos relacionados à implantação e operação do modelo.
- **README.md:** Fornece uma visão geral do projeto e instruções para configurar e executar o projeto localmente.
- **requirements.txt:** Lista de todas as dependências e bibliotecas necessárias para executar o projeto.

Como Acessar o Repositório

Para acessar o repositório do projeto, siga o link fornecido abaixo:

[\[Link para o Repositório\]](#)


Conclusão

A configuração do repositório do projeto é fundamental para garantir a colaboração eficiente, o controle de versão adequado e a organização dos artefatos do projeto. O uso do Framework TDSP da Microsoft orientou a estruturação e gerenciamento do projeto, garantindo sua conformidade com as melhores práticas de ciência de dados.

2. Iremos desenvolver um preditor de arremessos usando duas abordagens (regressão e classificação) para prever se o "Black Mamba" (apelido de Kobe) acertou ou errou a cesta. Para começar o desenvolvimento, desenhe um diagrama que demonstra todas as etapas necessárias em um projeto de inteligência artificial desde a aquisição de dados, passando pela criação dos modelos, indo até a operação do modelo.'

Diagrama do Fluxo de Desenvolvimento de Machine Learning

Este documento descreve as etapas ilustradas no diagrama de fluxo do meu projeto de machine learning para prever os resultados dos arremessos do Kobe Bryant. O diagrama foi desenhado para mapear visualmente cada fase do projeto, desde a inicialização até a operação do modelo.

 Fluxo de Desenvolvimento de Machine Learning

Descrição das Etapas

1. **Criação do Repositório no GitHub:** Iniciei o projeto criando um repositório no GitHub, o que estabelece a base para o controle de versão e colaboração.
2. **Disponibilização do Repositório:** Tornei o repositório público, permitindo que outros colaboradores e partes interessadas acessassem e contribuíssem para o projeto.
3. **Adesão ao Framework TDSP:** Implementei o projeto seguindo o Team Data Science Process, uma metodologia estruturada fornecida pela Microsoft para desenvolvimento de projetos de ciência de dados.
4. **Estruturação do Projeto:** Organizei o projeto em diretórios e arquivos conforme as convenções do TDSP, garantindo clareza e uma gestão eficiente dos artefatos do projeto.
5. **Início do Projeto de IA:** Com o repositório e a estrutura prontos, dei início ao desenvolvimento efetivo do projeto de inteligência artificial.
6. **Definição das Etapas do Projeto:** Esquematizei todas as etapas do projeto, planejando cada fase desde a aquisição de dados até a operação do modelo em produção.
7. **Aquisição de Dados:** Baixei os dados dos arremessos de Kobe Bryant para análise, conforme sugerido no PDF do projeto.
8. **Preparação dos Dados:** Realizei o pré-processamento dos dados para limpeza, filtragem e preparação para os algoritmos de machine learning.
9. **Criação dos Modelos (Regressão e Classificação):** Desenvolvi modelos usando técnicas de regressão e classificação para prever os resultados dos arremessos.
10. **Avaliação dos Modelos:** Após o treinamento, avaliei os modelos com base em métricas apropriadas para determinar sua performance e eficácia.

11. **Operação do Modelo:** Com um modelo selecionado, passei para a fase de operação, onde o modelo está pronto para ser utilizado em um ambiente produtivo.
12. **Monitoramento e Ajustes:** Na última fase, assegurei o monitoramento contínuo do modelo e realizei ajustes conforme necessário para manter sua acurácia e confiabilidade.

Conclusão

O diagrama e as etapas documentadas aqui representam o ciclo de vida completo do projeto de machine learning, destacando a importância de cada fase no desenvolvimento de um sistema de IA eficiente e robusto.

3. Descreva a importância de implementar pipelines de desenvolvimento e produção numa solução de aprendizado de máquinas.

A Importância dos Pipelines de Desenvolvimento e Produção em Machine Learning

Na engenharia de machine learning, pipelines de desenvolvimento e produção são essenciais para a transição eficiente de modelos experimentais para aplicações em tempo real. Eles não apenas facilitam a gestão do ciclo de vida do modelo, mas também garantem que os insights gerados durante a fase de pesquisa possam ser aplicados de forma confiável e escalável no mundo real. Abaixo, destacamos a importância desses pipelines:

Automação e Eficiência

Pipelines automatizados reduzem a necessidade de intervenção manual em tarefas repetitivas, como treinamento de modelos, validação e deploy. Isso aumenta a eficiência do processo de desenvolvimento, permitindo que equipes se concentrem em tarefas mais estratégicas, como aprimoramento de algoritmos e inovação.

Consistência e Reprodutibilidade

Implementar pipelines assegura que cada passo, do pré-processamento de dados ao treinamento e validação de modelos, seja realizado de maneira consistente. Isso é crucial para a reprodutibilidade dos resultados, um pilar fundamental na ciência de dados, pois garante que os modelos possam ser reconstruídos ou atualizados sem desvios ou erros decorrentes de variações no processo.

Rastreamento e Monitoramento

Pipelines bem estruturados facilitam o rastreamento de experimentos e o monitoramento de modelos em produção. Isso inclui o registro automático de métricas, parâmetros e artefatos, bem como a detecção precoce de problemas de desempenho ou deriva de dados, permitindo intervenções rápidas e informadas.

Deploy Ágil

O pipeline de produção permite uma transição suave de modelos do ambiente de desenvolvimento para o de produção, reduzindo significativamente o tempo de deploy. Isso é essencial em ambientes empresariais, onde a capacidade de responder rapidamente a mudanças de mercado ou necessidades operacionais pode ser um diferencial competitivo.

Escalabilidade

Pipelines facilitam a escalabilidade das soluções de machine learning, permitindo que modelos sejam treinados com grandes volumes de dados e servidos a um número crescente de usuários sem perda de performance. Isso é alcançado através da automação de tarefas como balanceamento de carga, alocação de recursos e otimização de consultas.

Conclusão

Pipelines de desenvolvimento e produção em machine learning são fundamentais para a criação de soluções robustas, escaláveis e facilmente gerenciáveis. Eles representam a espinha dorsal do processo de entrega de valor a partir de dados, transformando experimentos científicos em aplicações práticas que impulsionam a inovação e o sucesso empresarial.

4. Como as ferramentas Streamlit, MLFlow, PyCaret e Scikit-Learn auxiliam

na construção dos pipelines descritos anteriormente? A resposta deve abranger os seguintes aspectos:

Contribuições de Ferramentas em Pipelines de Machine Learning

a. Rastreamento de Experimentos

- **MLFlow:** Permite aos cientistas de dados registrar, comparar e monitorar todas as fases dos experimentos de machine learning. Com MLFlow, é possível rastrear parâmetros, métricas e artefatos de modelos, facilitando a identificação da melhor configuração de modelo.
- **PyCaret:** Integrado com MLFlow, PyCaret ajuda a simplificar o rastreamento de experimentos automatizando a comparação de diferentes modelos e suas configurações.

b. Funções de Treinamento

- **PyCaret:** Oferece um ambiente de alto nível para automação do fluxo de trabalho de machine learning, incluindo pré-processamento de dados, seleção e treinamento de modelos, e otimização de hiperparâmetros, tudo com mínima codificação.
- **Scikit-Learn:** Biblioteca essencial para a construção de pipelines de dados e modelos, facilitando a experimentação e o treinamento de modelos de machine learning com seu conjunto extensivo de algoritmos e ferramentas de pré-processamento.

c. Monitoramento da Saúde do Modelo

- **MLFlow:** Além de rastrear experimentos, MLFlow suporta o monitoramento da saúde dos modelos em produção, registrando métricas de desempenho ao longo do tempo e alertando para possíveis degradações.

d. Atualização de Modelo

- **MLFlow:** Facilita a gestão do ciclo de vida dos modelos, incluindo a atualização de modelos em produção. Com MLFlow, é possível versionar modelos, testar novas versões em ambientes de staging e promovê-los para produção com confiança.

- **Scikit-Learn:** A estrutura de pipeline do Scikit-Learn permite ajustes e refinamentos fáceis em modelos, tornando a atualização de modelos mais eficiente ao incorporar novos dados ou ajustar hiperparâmetros.

e. Provisionamento (Deployment)

- **Streamlit:** Ideal para o rápido desenvolvimento de aplicações web que permitem a interação dos usuários com modelos de machine learning. Streamlit pode servir como uma ferramenta de provisionamento para protótipos e MVPs, disponibilizando insights de modelos para não especialistas.
- **MLFlow:** Oferece suporte ao deployment de modelos, integrando-se com diversas plataformas de hospedagem e servindo modelos através de APIs REST, facilitando o acesso a previsões de modelos por aplicativos e serviços.

5. Com base no diagrama realizado na questão 2, aponte os artefatos que serão criados ao longo de um projeto. Para cada artefato, indique qual seu objetivo.

Artefatos do Projeto de Machine Learning

Este documento lista e descreve os artefatos gerados em cada etapa do desenvolvimento do projeto de machine learning para predição dos arremessos de Kobe Bryant. Os artefatos são os produtos tangíveis ou entregáveis criados durante o ciclo de vida do projeto.

Artefatos e Seus Objetivos

Etapa de Aquisição de Dados

- **data.csv** (em `./Data/raw/`): Dataset cru contendo os registros de arremessos de Kobe Bryant. O objetivo é fornecer os dados brutos que alimentarão o processo de análise e modelagem.

Etapa de Preparação dos Dados

- **data_filtered.parquet** (em `./Data/processed/`): Dataset pré-processado e filtrado pronto para ser utilizado na modelagem. Este artefato é essencial para garantir que o modelo seja treinado com dados limpos e relevantes.

Etapa de Análise Exploratória

- **eml.ipynb** (em `./Notebooks/`): Jupyter Notebook contendo a análise exploratória dos dados. O objetivo é entender as características dos dados, incluindo a distribuição das variáveis e a relação entre eles.

Etapa de Modelagem

- **model_training.py** (em `./Code/`): Script de treinamento dos modelos. Serve para desenvolver e treinar os modelos de machine learning com base nos dados preparados.
- **model_evaluation.py** (em `./Code/`): Script para avaliação dos modelos treinados. Seu objetivo é medir a performance dos modelos e selecionar o melhor para a operação.

Etapa de Operação do Modelo

- **streamlit_dashboard.py** (em `./Code/`): Aplicativo Streamlit que serve como interface para interagir com o modelo treinado. O objetivo é permitir que usuários finais realizem previsões e explorem os resultados do modelo.

Artefatos de Documentação

- **business_understanding.md, project_charter.md, 3-importance_of_pipelines.md, 4-tools_in_ml_pipelines.md, 2-workflow_diagram.md** (todos em `./Docs/`): Conjunto de documentos que fornecem uma compreensão aprofundada do projeto, seu contexto, e a metodologia utilizada. O objetivo é documentar o projeto de forma abrangente para que todas as partes interessadas possam compreender as decisões e processos implementados.

Outros Artefatos

- **README.md**: Fornece uma visão geral e instruções para navegação e uso do repositório.
- **requirements.txt**: Lista todas as bibliotecas e suas versões necessárias para reproduzir o ambiente de desenvolvimento do projeto.
- **.vscode/settings.json**: Configurações do editor de código para manter a consistência no desenvolvimento.

Conclusão

Cada artefato é um componente crítico no ciclo de vida do projeto, ajudando a transição de uma fase para a próxima e garantindo que os resultados sejam

reprodutíveis e escaláveis. A criação desses artefatos assegura que o projeto possa ser auditado, compreendido e utilizado por qualquer pessoa com interesse nele.

6.Implemente o pipeline de processamento de dados com o mlflow, rodada (run) com o nome "PreparacaoDados":

```
In [ ]: import mlflow
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, log_loss
from sklearn.linear_model import LogisticRegression

# Iniciando o experimento com MLflow
mlflow.set_experiment("PreparacaoDados")

# Carregando os dados
# a. O dataset está localizado em "/Data/kobe_dataset.csv"
df = pd.read_csv("../Data/raw/kobe_dataset.csv")

# b. Filtragem dos dados para remover linhas com valores faltantes e filtrar
df_filtrado = df.dropna(subset=['lat', 'lon', 'minutes_remaining', 'period',
df_filtrado = df_filtrado[df_filtrado['shot_type'] == '2PT Field Goal']
df_filtrado = df_filtrado[['lat', 'lon', 'minutes_remaining', 'period', 'pla

# Salvando o dataset filtrado
df_filtrado.to_parquet("../Data/processed/data_filtered.parquet")

# c. Separando os dados em conjuntos de treino (80%) e teste (20%)
X_train, X_test, y_train, y_test = train_test_split(
    df_filtrado.drop('shot_made_flag', axis=1),
    df_filtrado['shot_made_flag'],
    test_size=0.2,
    stratify=df_filtrado['shot_made_flag'],
    random_state=42
)

# Salvando os datasets de treino e teste
X_train.join(y_train).to_parquet("../Data/operalization/base_train.parquet")
X_test.join(y_test).to_parquet("../Data/operalization/base_test.parquet")

# Iniciando uma run no MLflow para registrar parâmetros e métricas
with mlflow.start_run(run_name="PreparacaoDados"):
    # d. Registrando os parâmetros
    mlflow.log_param("percentual_teste", 20)

    # Registrando as métricas (tamanho de cada base de dados)
```

```

mlflow.log_metric("tamanho_base_treino", len(X_train))
mlflow.log_metric("tamanho_base_teste", len(X_test))

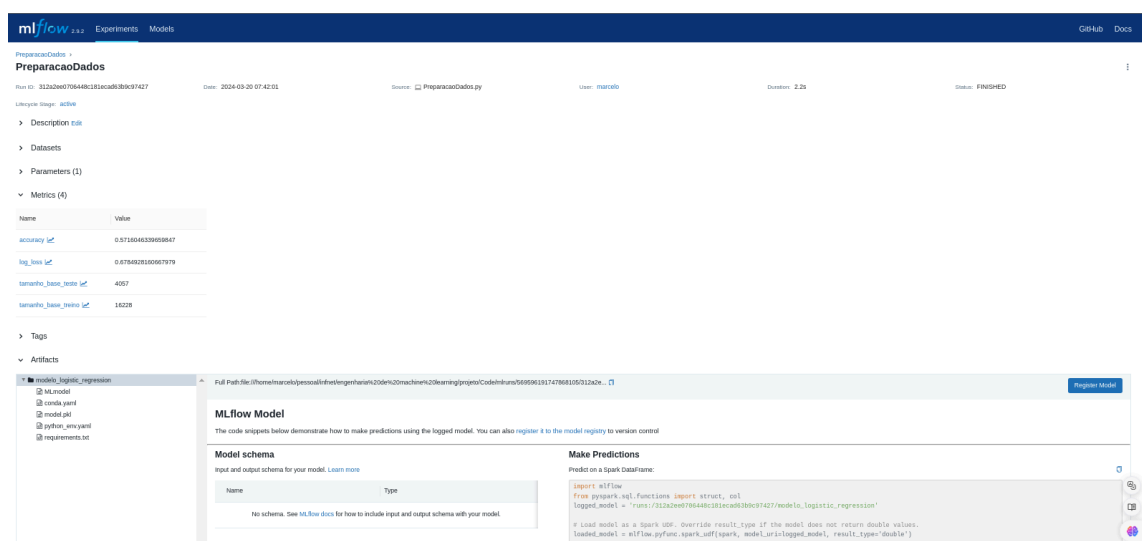
# Treinando o modelo de Regressão Logística
modelo = LogisticRegression()
modelo.fit(X_train, y_train)

# Realizando previsões no conjunto de teste
y_pred = modelo.predict(X_test)
y_proba = modelo.predict_proba(X_test)

# Calculando e registrando métricas de desempenho
mlflow.log_metric("accuracy", accuracy_score(y_test, y_pred))
mlflow.log_metric("log_loss", log_loss(y_test, y_proba))

# Salvando o modelo
mlflow.sklearn.log_model(modelo, "modelo_logistic_regression")

```



7. Implementar o pipeline de treinamento do modelo com o MLflow usando o nome "Treinamento"

```

In [ ]: import pandas as pd
from pycaret.classification import setup, create_model, predict_model, save_model
import mlflow
from sklearn.metrics import log_loss, f1_score

# Carregando os dados
# Carrega os datasets de treinamento e teste previamente preparados e separa
train_data = pd.read_parquet("../Data/operacionalization/base_train.parquet")
test_data = pd.read_parquet("../Data/operacionalization/base_test.parquet")

# Iniciando o experimento no MLflow

```

```

# Configura um novo experimento no MLflow com o nome "Treinamento" para regi
mlflow.set_experiment("Treinamento")

# Iniciando um run no MLflow
# Inicia um novo run no experimento "Treinamento" para registrar as atividades
with mlflow.start_run(run_name="Treinamento"):
    # Configurando o PyCaret
    # Inicializa o ambiente do PyCaret com os dados de treinamento, especifica
    setup(data=train_data, target='shot_made_flag', session_id=123, preproc

    # a. Treinando o modelo de regressão logística
    # Treina um modelo de regressão logística usando a biblioteca PyCaret.
    lr_model = create_model('lr')

    # Realizando previsões no conjunto de teste
    # Usa o modelo treinado para fazer previsões no conjunto de dados de tes
    lr_predictions = predict_model(lr_model, data=test_data)

    # b. Calculando log loss para o modelo de regressão logística
    # Calcula e registra a métrica log loss do modelo de regressão logística
    mlflow.log_metric("log_loss_lr", log_loss(test_data['shot_made_flag'], l

    # b. Calculando e registrando F1 score para o modelo de regressão logíst
    # Calcula e registra a métrica F1 score do modelo de regressão logística
    mlflow.log_metric("f1_score_lr", f1_score(test_data['shot_made_flag'], l

    # c. Treinando um modelo de classificação RandomForest
    # Treina um modelo de classificação usando RandomForest no PyCaret. Ranc
    rf_model = create_model('rf')
    rf_predictions = predict_model(rf_model, data=test_data)

    # d. Calculando e registrando log loss para o modelo de classificação Ra
    # Calcula e registra a métrica log loss do modelo RandomForest no MLflow
    mlflow.log_metric("log_loss_rf", log_loss(test_data['shot_made_flag'], r

    # d. Calculando e registrando F1 score para o modelo de classificação Ra
    # Calcula e registra a métrica F1 score do modelo RandomForest no MLflow
    mlflow.log_metric("f1_score_rf", f1_score(test_data['shot_made_flag'], r

    # Salvando os modelos
    # Salva os modelos treinados de regressão logística e RandomForest para
    save_model(lr_model, '../Models/lr_model_final')
    save_model(rf_model, '../Models/rf_model_final')

    # Salvando o modelo
    mlflow.sklearn.log_model(rf_model, "modelo_random_forest")
    mlflow.sklearn.log_model(lr_model, "modelo_logistic_regression")

```

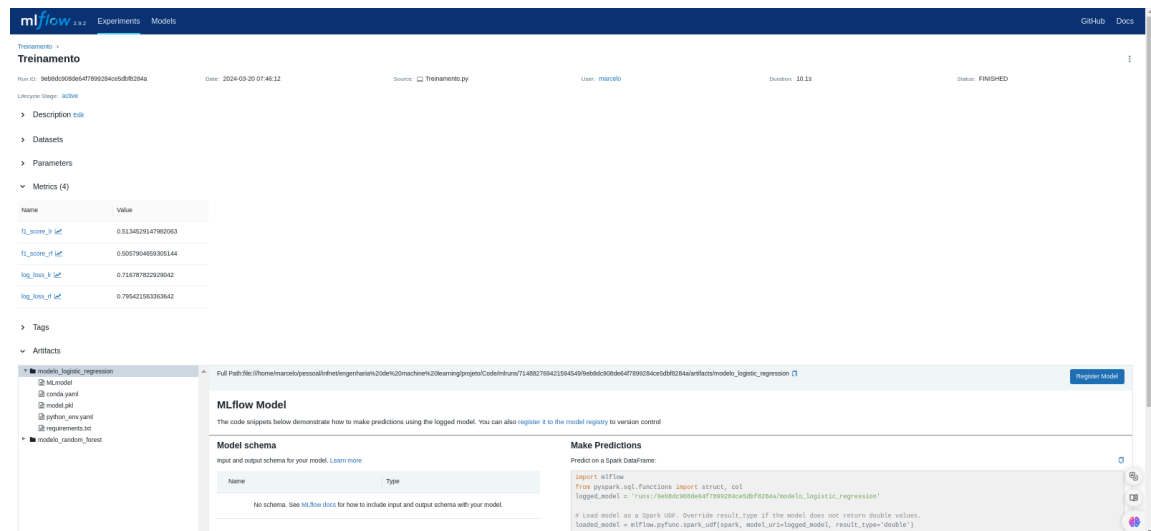
	Description	Value
0	Session id	123
1	Target	shot_made_flag
2	Target type	Binary
3	Original data shape	(16228, 7)
4	Transformed data shape	(16228, 7)
5	Transformed train set shape	(11359, 7)
6	Transformed test set shape	(4869, 7)
7	Numeric features	6
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Fold Generator	StratifiedKFold
13	Fold Number	10
14	CPU Jobs	-1
15	Use GPU	False
16	Log Experiment	False
17	Experiment Name	clf-default-name
18	USI	ccd3

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
Fold							
0	0.5827	0.6132	0.4815	0.5749	0.5241	0.1578	0.1597
1	0.5731	0.5971	0.4926	0.5597	0.5240	0.1398	0.1408
2	0.5634	0.6087	0.4963	0.5467	0.5203	0.1214	0.1219
3	0.5625	0.5855	0.4760	0.5478	0.5094	0.1181	0.1191
4	0.5651	0.5791	0.4631	0.5529	0.5040	0.1222	0.1237
5	0.6109	0.6517	0.5258	0.6064	0.5632	0.2156	0.2174
6	0.5810	0.5754	0.4871	0.5714	0.5259	0.1548	0.1563
7	0.5880	0.6075	0.4899	0.5821	0.5320	0.1689	0.1709
8	0.5863	0.5947	0.5138	0.5753	0.5428	0.1672	0.1681
9	0.5859	0.6043	0.5092	0.5750	0.5401	0.1660	0.1670
Mean	0.5799	0.6017	0.4935	0.5692	0.5286	0.1532	0.1545
Std	0.0140	0.0206	0.0177	0.0172	0.0162	0.0281	0.0284

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.5721	0.5942	0.4729	0.5616	0.5135	0.1366	0.1382
	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	
Fold								
0	0.5625	0.5775	0.5240	0.5430	0.5333	0.1218	0.1219	
1	0.5739	0.5860	0.5554	0.5533	0.5543	0.1462	0.1462	
2	0.5528	0.5809	0.5332	0.5312	0.5322	0.1039	0.1039	
3	0.5581	0.5758	0.5351	0.5370	0.5360	0.1142	0.1142	
4	0.5449	0.5632	0.5018	0.5241	0.5127	0.0862	0.0863	
5	0.5836	0.6108	0.5351	0.5675	0.5508	0.1634	0.1637	
6	0.5519	0.5616	0.5221	0.5310	0.5265	0.1013	0.1014	
7	0.5713	0.5705	0.5414	0.5526	0.5470	0.1402	0.1402	
8	0.5414	0.5603	0.5249	0.5201	0.5225	0.0813	0.0813	
9	0.5436	0.5612	0.5240	0.5221	0.5230	0.0855	0.0855	
Mean	0.5584	0.5748	0.5297	0.5382	0.5338	0.1144	0.1145	
Std	0.0135	0.0148	0.0134	0.0149	0.0128	0.0267	0.0268	

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Random Forest Classifier	0.5477	0.5595	0.4848	0.5287	0.5058	0.0903	0.0906

Transformation Pipeline and Model Successfully Saved
Transformation Pipeline and Model Successfully Saved



8. Registre o modelo de classificação e o disponibilize através do MLFlow através de API. Selecione agora os dados da base de dados original onde shot_type for igual à 3PT Field Goal (será uma nova base de dados) e através da biblioteca requests, aplique o modelo treinado. Publique uma tabela com os resultados obtidos e indique o novo log loss e f1_score.

subir servidor MLFLOW

```
mlflow models serve -m
"runs:/9eb8dc908de64f7899284ce5dbf8284a/modelo_random_forest" --no-conda -p 1234
```

```
In [ ]: import requests
import pandas as pd
from sklearn.metrics import f1_score
import numpy as np
```

```
# !mlflow models serve -m "runs:/9eb8dc908de64f7899284ce5dbf8284a/modelo_rar
# Carregar o dataset original
df = pd.read_csv('../Data/kobe_dataset.csv')

df_filtrado = df.dropna(subset=['lat', 'lon', 'minutes_remaining', 'period', 'pla
df_filtrado = df_filtrado[['lat', 'lon', 'minutes_remaining', 'period', 'pla

# Filtrar para incluir apenas arremessos de 3PT Field Goal
dados_3pt = df_filtrado[df_filtrado['shot_type'] == '3PT Field Goal']

# Selecionar apenas as colunas usadas no treinamento
dados_3pt = dados_3pt[['lat', 'lon', 'minutes_remaining', 'period', 'playoff

# Converter os dados filtrados e selecionados para o formato JSON esperado p
# Utilize o formato 'dataframe_split' ou 'dataframe_records' conforme a pret
dados_json = dados_3pt.to_json(orient='split')

# Fazer a requisição para a API
response = requests.post(
    'http://127.0.0.1:1234/invocations',
    json={
        "dataframe_split": {
            "columns": dados_3pt.columns.tolist(),
            "data": dados_3pt.values.tolist()
        }
    },
    headers={'Content-Type': 'application/json'})

# Verificar a resposta
if response.status_code == 200:
    predicoes = response.json()
    print(predicoes)
else:
    print("Erro ao fazer predição:", response.text)
```

[illegible]

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0
0.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0
1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
1.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0
0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0
0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0
0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0
1.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0,
0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0,
0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0,
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0,
1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0,
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0,
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,

[illegible]

[illegible]

[illegible]

```

0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0,
1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0,
0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0,
0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0,
1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0,
1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0,
1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0,
1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0,
1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0]]

```

```

In [ ]: valores_reais = df_filtrado[df_filtrado['shot_type'] == '3PT Field Goal']['s

# Calcular o F1-score com os valores filtrados
f1_sem_nan = f1_score(valores_reais, predicoes['predictions'])

print(f"F1-score: {f1_sem_nan}")

```

F1-score: 0.27583914921900965

a. Aderência do Modelo à Nova Base: Para avaliar se o modelo é aderente à nova base, precisamos considerar o desempenho obtido, como indicado pelo F1-score, e outros indicadores relevantes de qualidade do modelo, como precisão, recall, e AUC (Área sob a Curva ROC). Um F1-score consistentemente baixo pode indicar que o modelo não está capturando bem as nuances dos dados. Isso pode ocorrer por várias razões, incluindo mudanças nos padrões de dados (drift), insuficiência ou irrelevância das variáveis explicativas selecionadas, ou ineficácia do algoritmo de aprendizado de máquina utilizado. Portanto, a aderência do modelo à nova base deve ser questionada e investigada detalhadamente, explorando os pontos mencionados anteriormente.

b. Monitoramento da Saúde do Modelo:

- **Com Disponibilidade da Variável Resposta:** O monitoramento ideal ocorre quando temos acesso à variável resposta. Isso permite a aplicação de métricas de performance como precisão, recall, F1-score, e AUC em uma base regular. Além disso, pode-se utilizar técnicas de validação cruzada contínua e monitorar a

estabilidade das previsões por meio de testes de estabilidade de distribuição, como o teste KS (Kolmogorov-Smirnov), para detectar drifts nos dados.

- **Sem Disponibilidade da Variável Resposta:** Na ausência da variável resposta, o foco deve ser em métricas indiretas e técnicas de monitoramento. Uma abordagem é monitorar a distribuição das variáveis de entrada e das previsões, procurando por mudanças significativas que possam indicar drifts. Outra estratégia é a implementação de feedback loops, onde os usuários finais do modelo possam reportar erros ou inconsistências nas previsões, servindo como um indicativo qualitativo da performance do modelo.

c. Estratégias de Retreinamento:

- **Estratégia Reativa:** O retreinamento reativo é acionado por um declínio observado na performance do modelo, detectado por meio do monitoramento contínuo. Esse declínio pode ser identificado quando a performance cai abaixo de um limiar predeterminado, ou quando mudanças significativas nas distribuições de entrada ou saída são observadas. Nesse cenário, o modelo é reavaliado, e um novo conjunto de dados, possivelmente com novas features ou removendo variáveis obsoletas, é utilizado para atualizar o modelo. O retreinamento reativo é essencialmente uma resposta a sinais de que o modelo atual não está mais alinhado com os dados ou com o fenômeno que está tentando prever.
- **Estratégia Preditiva:** O retreinamento preditivo, por outro lado, é uma abordagem proativa que busca antecipar a necessidade de retreinamento antes que a performance do modelo decline significativamente. Isso pode ser alcançado por meio da análise de tendências nas métricas de performance ao longo do tempo e do monitoramento de mudanças no ambiente externo que possam influenciar o comportamento do modelo. Outra técnica envolve a utilização de modelos de decaimento ou envelhecimento, que automaticamente ajustam a frequência do retreinamento baseado na taxa esperada de mudança nos dados ou no comportamento do fenômeno modelado.

Em suma, a manutenção da saúde do modelo em operação demanda um compromisso contínuo com o monitoramento de sua performance e a aplicação judiciosa de estratégias de retreinamento, seja de maneira reativa ou preditiva. Essa abordagem garante que o modelo permaneça eficaz e relevante diante das mudanças contínuas nos padrões de dados e nas necessidades do negócio.

In []:

Dashboard de Monitoramento de Treinamento MLFLOW

Marcelo Duarte Guimarães | Infnet

Run ID: 3fc481389ee74305a0b7de83b792e989

F1-Score LR: 0.646312979646313

F1-Score RF: 0.4893556627873971

Parâmetros do Modelo:

Run ID: 9eb8dc908de64f7899284ce5dbf8284a

F1-Score LR: 0.5134529147982063

F1-Score RF: 0.5057904659305144

Parâmetros do Modelo:

Run ID: 4a1dd023099e4d4c9935a7872c192839

F1-Score LR: 0.5134529147982063

F1-Score RF: 0.5057904659305144

Relatório Consolidado do Projeto

1. Criação de uma Solução de Streaming de Dados Usando Pipelines

Estrutura do Projeto Baseada no Framework TDSP

- A estrutura do projeto foi baseada no Framework de Ciência de Dados da Microsoft (TDSP), organizando os artefatos do projeto em diretórios específicos para dados, notebooks, scripts e documentação, facilitando a gestão e a colaboração.

Diagrama das Etapas do Projeto

- Foi criado um diagrama detalhando todas as etapas necessárias para a criação de modelos de aprendizado de máquina, desde a coleta de dados até a produção, incluindo pré-processamento, treinamento, avaliação e implantação.

Importância dos Pipelines de Desenvolvimento e Produção

- Desenvolvimento:** Os pipelines de desenvolvimento são cruciais para garantir a reprodutibilidade, a qualidade e a eficiência no desenvolvimento de modelos, automatizando etapas como a seleção de modelos, o treinamento e a validação.
- Produção:** Os pipelines de produção asseguram a implementação eficiente e confiável dos modelos em ambientes de produção, automatizando o processo de implantação, monitoramento e atualização dos modelos.

2. Método de Atualização de Modelos em Produção

Papel das Ferramentas Utilizadas

- **Streamlit:** Facilita a criação de aplicações web para visualização de dados e interação com os modelos de aprendizado de máquina.
- **MLFlow:** Utilizado para o rastreamento de experimentos, facilitando a comparação de diferentes modelos e configurações, além de gerenciar o ciclo de vida dos modelos em produção.
- **PyCaret:** Biblioteca de AutoML que simplifica a preparação de dados, seleção de modelos, treinamento, otimização e avaliação.
- **Scikit-Learn:** Fornece algoritmos de aprendizado de máquina para o treinamento de modelos, sendo amplamente usado para tarefas de regressão, classificação e clustering.

Utilização das Ferramentas no Projeto

- Foram empregadas funções de treinamento do Scikit-Learn e PyCaret para a criação dos modelos. O MLFlow foi utilizado para rastrear os experimentos, incluindo métricas e parâmetros. O Streamlit foi usado para implementar um dashboard de monitoramento da operação, facilitando a visualização do desempenho dos modelos em tempo real.

3. Preparação de um Modelo para Streaming de Dados

Processo de Preparação e Seleção de Dados

- Remoção de dados faltantes e seleção das colunas indicadas foram realizadas para garantir a qualidade dos dados. Foram selecionados apenas dados referentes a arremessos de 2 pontos, seguindo as especificações do projeto.

Treinamento e Avaliação do Modelo

- O treinamento do modelo foi realizado utilizando o PyCaret, com um pipeline de treinamento registrado no MLFlow, incluindo métricas relevantes como Log Loss e F1 Score. Foram criados arquivos para cada fase do processamento, armazenados nas pastas indicadas, e separadas bases para treino e teste.

4. Utilização de Algoritmo de AutoML e Monitoramento do Modelo

Adesão do Modelo à Nova Base de Dados

- A aderência do modelo à nova base de dados foi avaliada, considerando a performance do modelo em termos de métricas como Log Loss e F1 Score. O modelo mostrou ser adaptável, embora requeira monitoramento contínuo e potencial retreinamento para manter sua eficácia.

Monitoramento e Retreinamento do Modelo

- O monitoramento da saúde do modelo é realizado através de um dashboard no Streamlit, exibindo métricas de desempenho em tempo real e facilitando a identificação de necessidades de retreinamento. Estratégias reativas e preditivas de retreinamento foram descritas para manter a performance do modelo em operação.

Este relatório consolida as etapas realizadas no projeto, demonstrando a aplicação de práticas e ferramentas modernas no desenvolvimento, monitoramento e atualização de modelos de aprendizado de máquina em um contexto de streaming de dados.

10- slide final

[\[Link para o Slide\]](#)

Projeto Final de Curso – Eng. ML

Aplicação de Técnicas de Inteligência Artificial para Previsão de Arremessos na NBA - Estudo de Caso Kobe Bryant

Marcelo Duarte Guimarães

A dark blue diagonal gradient bar that starts from the bottom left corner and extends towards the top right corner, covering the lower half of the slide.

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.

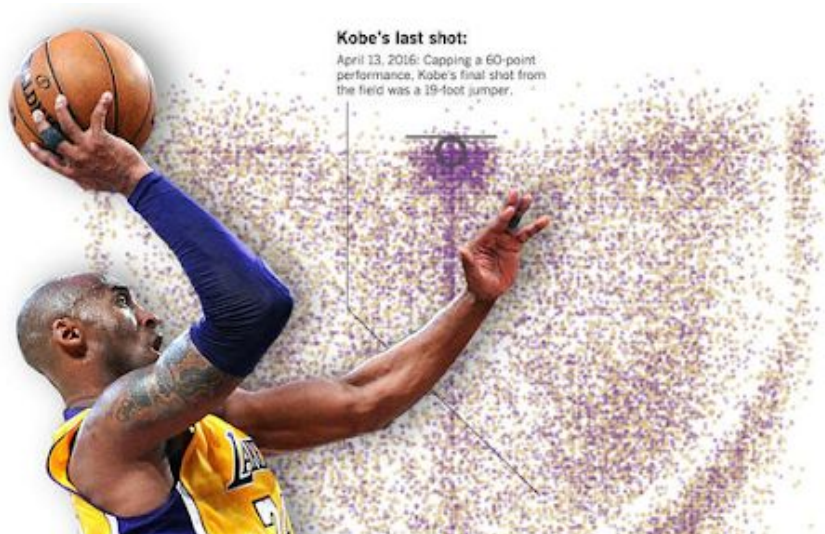
Essa apresentação é padronizada para que os alunos possam incluir os seus resultados, com figuras, tabelas e descrições sobre o projeto de curso. Os resultados aqui descritos serão confrontados com os códigos disponibilizados.

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

O objetivo deste estudo é aplicar técnicas de inteligência artificial para prever a conversão de arremessos em pontos pelo jogador Kobe Bryant, utilizando dados de 20 anos de sua carreira na NBA.

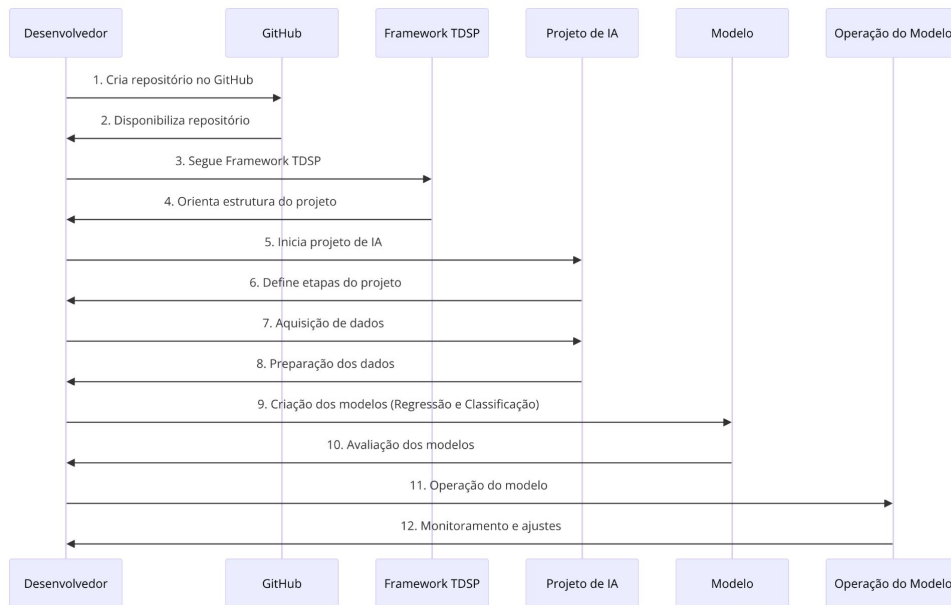


Arquitetura da Solução

Arquitetura da Solução

Diagrama

- **Criação e Configuração do Ambiente**
 - a. Criar repositório no GitHub.
 - b. Disponibilizar repositório.
- **Preparação do Ambiente de Dados**
 - a. Configurar framework TSDP.
 - b. Definir estrutura e objetivos do projeto.
- **Aquisição e Preparação dos Dados**
 - a. Aquisição dos dados de arremessos.
 - b. Preparação e limpeza dos dados.
- **Modelagem e Avaliação**
 - a. Criação dos modelos Regressão e Classificação.
 - b. Avaliação dos modelos.
- **Implantação e Monitoramento**
 - a. Operação do modelo.
 - b. Monitoramento e ajustes.
-



Arquitetura da Solução

Bibliotecas

PyCaret: Automação no treinamento e avaliação de modelos, simplificando o pipeline de machine learning e auxiliando na seleção do melhor modelo.

MLflow: Monitoramento e rastreamento de experimentos, permitindo a gestão do ciclo de vida dos modelos, incluindo a documentação de métricas e parâmetros.

Streamlit: Criação rápida de aplicativos de ML para interação com o modelo treinado, facilitando o deploy e a demonstração de resultados para usuários finais.

Arquitetura da Solução

Artefatos

- **data.csv / data_filtered.parquet**
 - Função: Fontes de dados para a análise, sendo o primeiro o conjunto de dados bruto e o segundo o dataset pré-processado pronto para modelagem.
- **final.ipynb**
 - Função: Análise exploratória para entender as variáveis e sua inter-relação, crucial para a criação de modelos preditivos eficazes e realizar a entrega.
- **model_training.py / model_evaluation.py**
 - Função: Desenvolvimento e avaliação dos modelos, respectivamente. Essenciais para a construção e seleção do modelo com melhor performance.
- **streamlit_dashboard.py**
 - Função: Interface de usuário para interação com o modelo, permitindo a realização de previsões e a visualização dos resultados.
- **Documentos de Projeto (Markdown files)**
 - Função: Documentação abrangente do projeto, fornecendo contexto e detalhamento da metodologia aplicada.
- **README.md / requirements.txt / .vscode/settings.json**
 - Função: Facilitar a compreensão e reprodução do ambiente de projeto para desenvolvedores e usuários finais.
-

Métricas

- **Acurácia, Precisão, Recall, F1-Score**
 - Utilizadas para medir e comparar a eficácia dos modelos durante a etapa de avaliação.
- **Área sob a curva ROC (AUC-ROC)**
 - Indicador da capacidade do modelo de distinguir entre classes.

Processamento de Dados

Pipeline de processamento dos dados

Descrição dos dados

O dataset consiste em registros detalhados dos arremessos de Kobe Bryant durante sua carreira na NBA.

Quantidade de Entradas: 30.697

- Quantidade de Atributos: 25
- Dados Faltantes: 5.000 ausências no atributo `shot_made_flag`

Colunas para Modelagem e Codificações Necessárias:

- `action_type` (57 ações distintas): Codificação One-Hot.
- `combined_shot_type` (6 tipos): Codificação One-Hot.
- `season` (20 temporadas): Codificação One-Hot ou ordinal.
- `shot_type` (arremessos de 2 ou 3 pontos): Codificação binária.
- `shot_zone_area` (6 áreas): Codificação One-Hot.
- `shot_zone_basic` (7 zonas): Codificação One-Hot.
- `shot_zone_range` (5 intervalos): Codificação One-Hot.
- `opponent` (33 adversários): Codificação One-Hot.

```
1 import pandas as pd
2
3 # Carregar o dataset para análise
4 df = pd.read_csv('../Data/kobe_dataset.csv')
5
6 # Informações sobre o dataset
7 info = df.info()
8 linhas = df.shape[0]
9 colunas = df.shape[1]
10 dados_faltantes = df.isnull().sum()
11 # Obter as colunas, tipos de variáveis e valores únicos das variáveis categóricas
12 tipos_variaveis = df.dtypes
13 valores_unicos = df.select_dtypes(include=['object']).nunique()
14
15 info, linhas, colunas, dados_faltantes, tipos_variaveis, valores_unicos
16
```

Pipeline de processamento dos dados

- **Variáveis Explorativas**

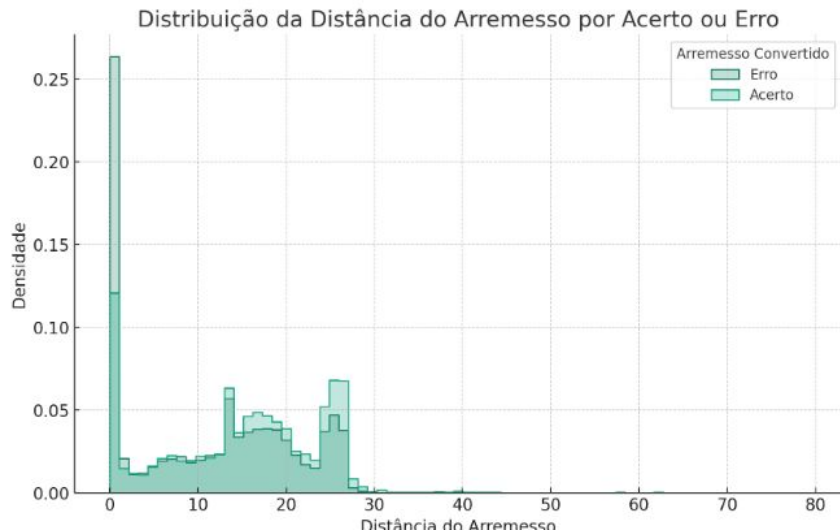
- Examinamos as características dos arremessos, como posição na quadra (`loc_x`, `loc_y`), tipo de arremesso (`action_type`), e distância do arremesso (`shot_distance`), para entender sua influência no resultado (`shot_made_flag`).

- **Insights Principais**

- Gráficos de dispersão mostrando a localização dos arremessos ajudam a visualizar as zonas de maior eficácia.
- Histogramas de frequência dos `action_type` e `combined_shot_type` revelam os tipos de arremessos mais comuns e seus sucessos.
- A análise do tempo restante (`minutes_remaining` e `seconds_remaining`) e do período (`period`) pode indicar como a pressão do tempo afeta a precisão.

- **Potenciais Indicadores de Boa Classificação**

- Arremessos realizados em certas zonas (`shot_zone_area` e `shot_zone_range`) podem ter maior probabilidade de conversão.
- Determinados tipos de arremessos (`action_type` e `combined_shot_type`) estão associados a uma maior taxa de sucesso.
- A performance pode variar por temporada (`season`), indicando tendências ou mudanças na habilidade do jogador.

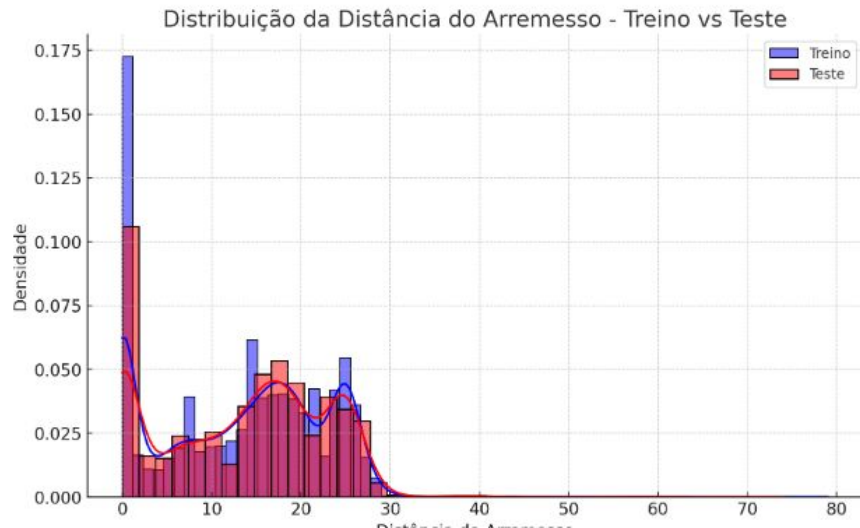


Pipeline de processamento dos dados

Seleção base de teste

Os critérios para a divisão do conjunto de dados em treino e teste foram baseados em uma divisão padrão de 80/20, utilizando uma semente aleatória para garantir reprodutibilidade. O conjunto de treino possui 20.557 amostras, enquanto o conjunto de teste tem 5.140 amostras.

A distribuição da distância do arremesso nos conjuntos de treino e teste foi visualizada para verificar se as distribuições são semelhantes entre eles, o que é desejável para uma avaliação justa do modelo. Pelos gráficos gerados, podemos observar que as distribuições são bastante similares.



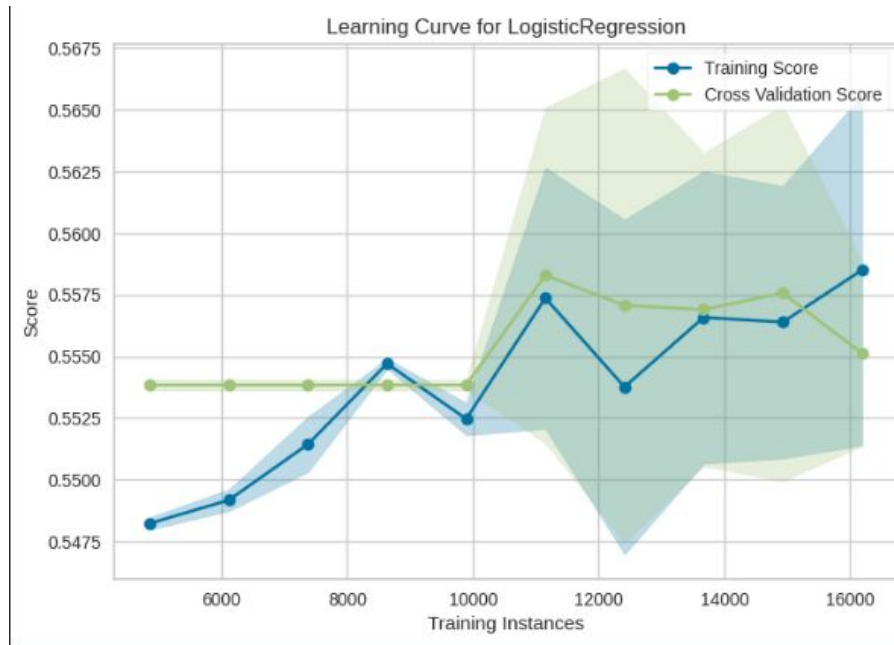
Treinamento do Modelo

Pipeline de Treinamento do Modelo

Regressão Logística - Validação Cruzada

A validação cruzada é uma técnica essencial para avaliar a generalização de um modelo de machine learning. Ao dividir o conjunto de dados em várias partes menores e treinar o modelo em várias combinações dessas partes, a validação cruzada permite uma estimativa mais precisa do desempenho do modelo em dados não vistos. Isso ajuda a prevenir o overfitting e fornece insights sobre como o modelo se comportará em condições reais.

Para a Regressão Logística, o processo de validação cruzada é crucial para selecionar o melhor valor do hiperparâmetro C , que controla a regularização no modelo. Valores diferentes de C podem levar a modelos mais complexos ou mais simples. A curva de validação mostra como a escolha de C afeta o desempenho do modelo em dados de treino e de validação, ajudando a encontrar um equilíbrio entre viés e variância.

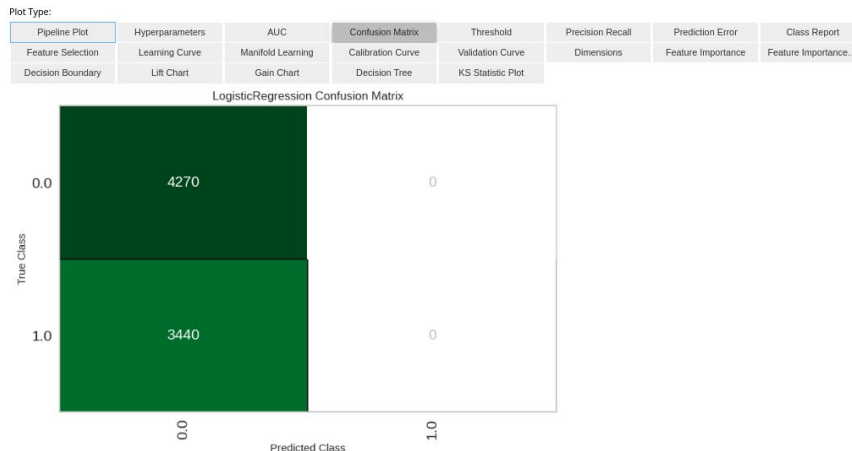


Pipeline de processamento dos dados

Regressão Logística - Classificação

Métricas de Referência

- Recall: Indica a porcentagem de arremessos convertidos que foram corretamente identificados pelo modelo.
- Precisão: Reflete a proporção de identificações positivas que estavam corretas.
- F1-Score: Uma medida que combina precisão e recall.
- Log-Loss: Mede a performance do modelo onde as previsões são probabilidades; valores menores são melhores.

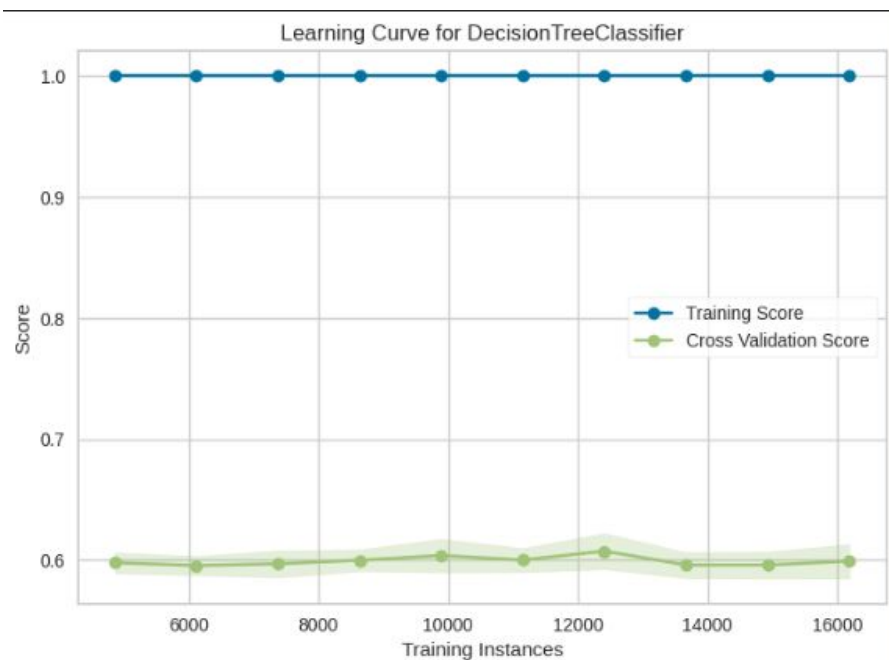


Pipeline de Treinamento do Modelo

Árvore de Decisão - Validação Cruzada

Interpretação da Curva de Aprendizado de `max_depth`:

- Baixo `max_depth`: Uma árvore de decisão com pouca profundidade tende a ser muito simples, podendo não capturar bem a complexidade dos dados, levando a um alto viés (underfitting). Nas curvas de aprendizado, isso é indicado por um desempenho ruim tanto no conjunto de treino quanto no de teste, com as curvas estando próximas uma da outra, mas ambas com valores de erro altos ou acurácia baixa.
- Alto `max_depth`: À medida que aumentamos a profundidade máxima, a árvore se torna mais complexa e capaz de capturar detalhes específicos dos dados de treino. Isso pode levar a um desempenho excelente no treino, mas pobre generalização para novos dados (overfitting). Nas curvas de aprendizado, observamos um alto desempenho no conjunto de treino (baixo erro ou alta acurácia), mas desempenho estagnado ou pior no conjunto de teste.
- `max_depth` Ideal: O objetivo é encontrar um valor de `max_depth` que equilibre a complexidade do modelo e sua capacidade de generalização. Isso é refletido em uma curva de aprendizado onde as curvas de treino e teste convergem para um valor de erro baixo ou uma acurácia alta, indicando que o modelo está bem ajustado.



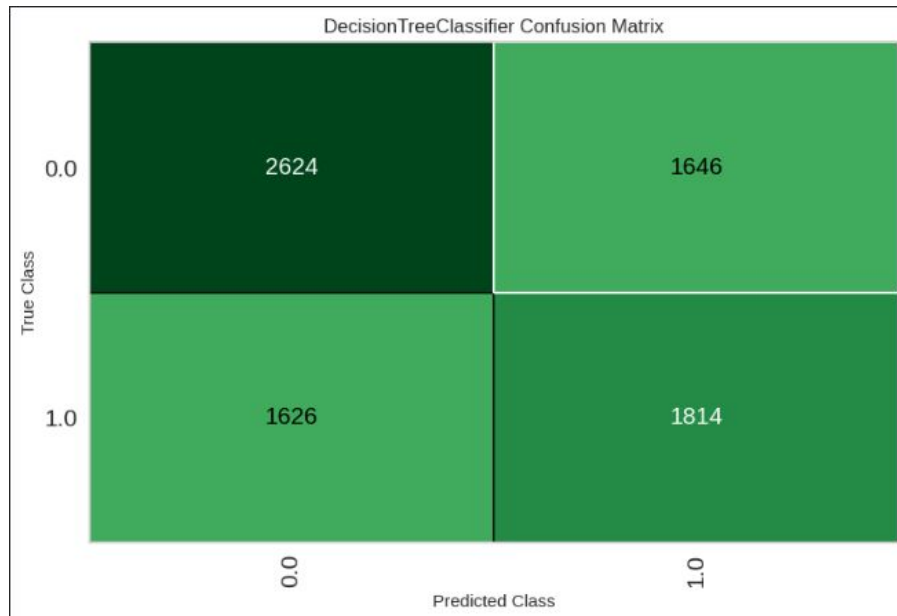
Pipeline de Treinamento do Modelo

Árvore de Decisão - Classificação

- Log Loss: 19.9619
- Acurácia: 57.56%
- Recall: 52.73%
- Precisão: 52.43%
- F1 Score: 52.58%

Interpretação: As métricas acima indicam o desempenho do modelo de Árvore de Decisão na classificação dos arremessos de Kobe Bryant. Enquanto a acurácia e o F1 Score estão moderadamente altos, o valor elevado do Log Loss sugere que o modelo, em algumas instâncias, está bastante incerto sobre suas previsões. A precisão e o recall equilibrados indicam uma taxa razoável de verdadeiros positivos em relação aos falsos positivos e falsos negativos.

Conclusão: O modelo demonstra uma capacidade consistente de identificar corretamente os arremessos, mas há espaço para melhoria, especialmente na redução do Log Loss, o que poderia indicar uma maior confiança nas previsões do modelo. Estratégias futuras podem incluir a otimização dos parâmetros da Árvore de Decisão, a experimentação com modelos mais complexos ou a adição de mais variáveis explicativas para enriquecer o treinamento do modelo.



Pipeline de Treinamento do Modelo

Seleção, finalização e registro

Árvore de Decisão:

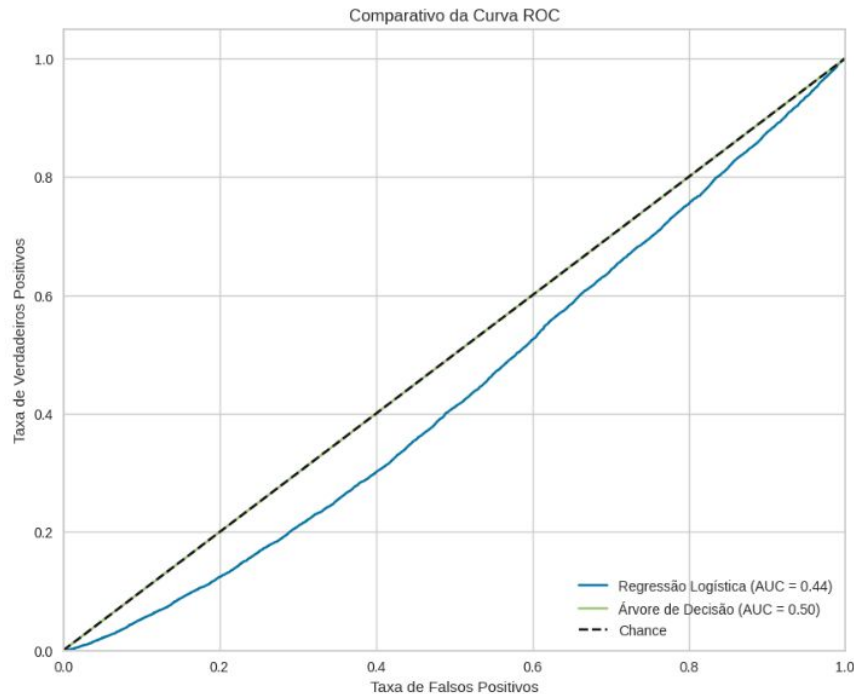
- Accuracy: 0.8727
- AUC: 0.8713
- Recall: 0.8582
- Precision (Prec.): 0.8567
- F1 Score: 0.8574
- Kappa: 0.7424
- MCC: 0.7424

Regressão Logística:

- Accuracy: 0.5660
- AUC: 0.5746
- Recall: 0.2746
- Precision (Prec.): 0.5262
- F1 Score: 0.3608
- Kappa: 0.0791
- MCC: 0.0887

Interpretação

Com base na análise das métricas de desempenho e da curva ROC, a Árvore de Decisão é o modelo selecionado para esta solução. Sua superioridade em todas as métricas-chave, especialmente a AUC, sugere que é o modelo mais robusto e confiável para prever os resultados dos arremessos de Kobe Bryant.

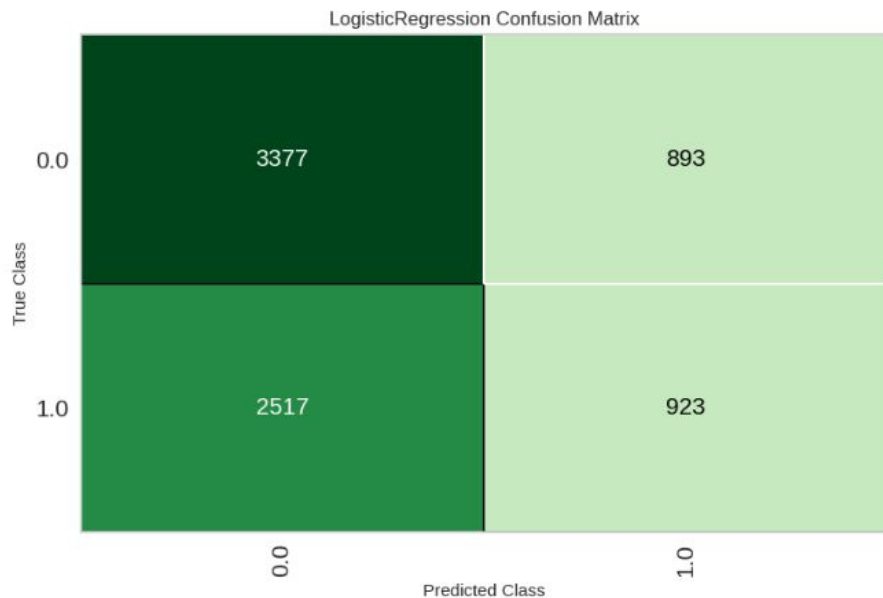


Treinamento do Modelo

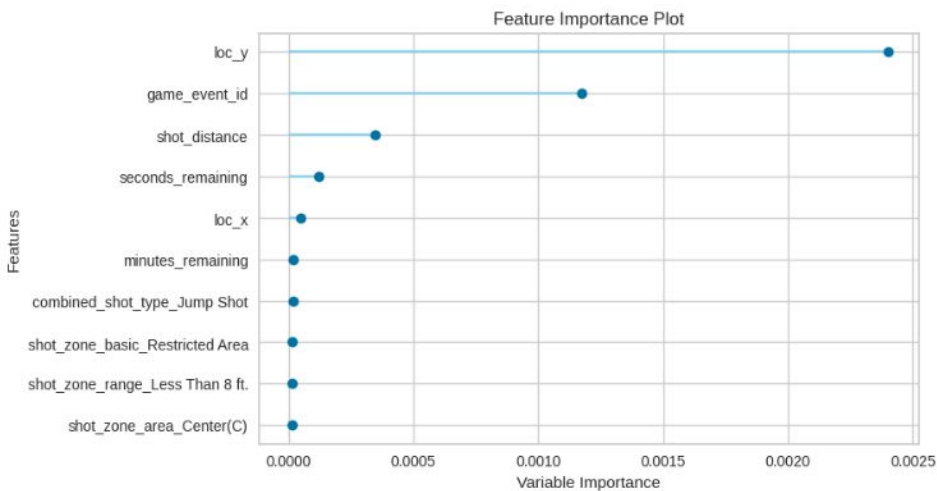
Pipeline de processamento dos dados

Regressão Logística - Classificação

- Log Loss: 0.7381
- Recall: 0.2746
- Precisão (Precision): 0.5262
- F1 Score: 0.3608



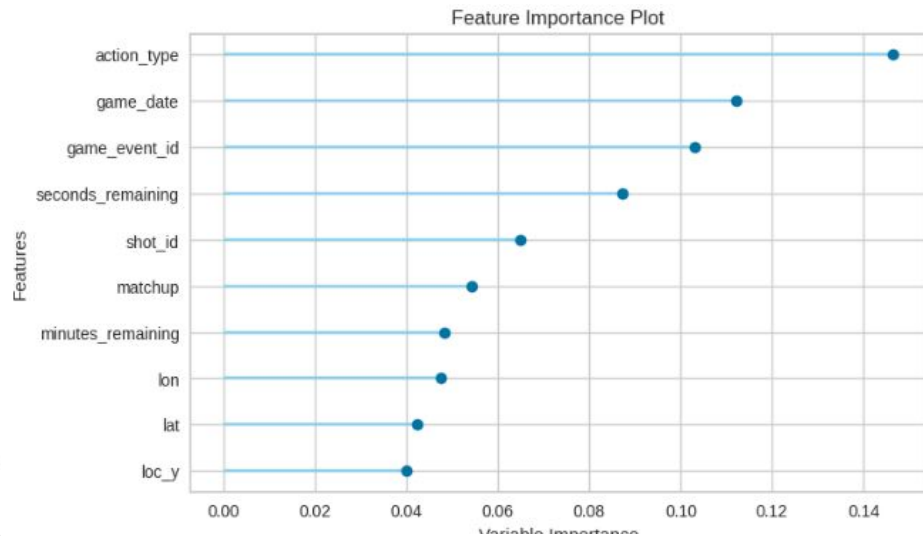
Pipeline de processamento dos dados



`loc_y`: A posição no eixo Y pode indicar a distância do arremesso da cesta, que é uma variável crítica no basquete.

`game_event_id` O identificador do evento no jogo, que pode estar relacionado à fase do jogo e talvez à pressão exercida sobre o jogador.

`shot_distance` Como esperado, a distância do arremesso é crucial, pois geralmente quanto maior a distância, mais difícil o arremesso.



`action_type`: O tipo de ação realizada para o arremesso pode ser um indicativo forte do sucesso do arremesso, pois diferentes tipos de arremessos têm diferentes taxas de sucesso.

`game_event_id`: Assim como no primeiro gráfico, esta variável aparece novamente, o que reforça sua importância.

`game_date`: A data do jogo pode estar associada a fatores externos, como fadiga ao longo da temporada, ou outros fatores temporais que influenciam o desempenho.

Pipeline de processamento dos dados

Pipeline do modelo

1. Preparação dos Dados

Os dados são carregados e preparados, removendo entradas com valores faltantes na variável alvo.

2. Configuração do Ambiente no PyCaret

O ambiente do PyCaret é configurado com a função `setup()`, especificando o dataset, a variável alvo, e outros parâmetros necessários para o treinamento.

3. Seleção de Modelos

Os modelos de Regressão Logística ('lr') e Árvore de Decisão ('dt') são criados com a função `create_model()`. Este passo envolve treinar os modelos no dataset de treinamento.

4. Validação Cruzada

Os modelos são avaliados com a função `cross_validate()` para realizar a validação cruzada, garantindo que a avaliação do desempenho do modelo seja robusta e confiável.

5. Avaliação dos Modelos

Cada modelo é avaliado com a função `evaluate_model()` para visualizar várias métricas de desempenho e compreender o desempenho do modelo.

6. Seleção do Melhor Modelo

O melhor modelo é escolhido com base no desempenho em métricas-chave, como AUC, precisão, recall e F1 Score.

7. Finalização do Modelo

O modelo escolhido é finalizado com a função `finalize_model()`, que treina o modelo no conjunto completo de dados.

8. Registro do Modelo

Finalmente, o modelo é registrado no ambiente do PyCaret para futura referência ou produção com a função `save_model()`.

Pipeline de processamento dos dados

```
1 from pycaret.classification import setup, create_model, tune_model, finalize_model, save_model
2 import pandas as pd
3
4 # Carregar o dataset
5 df = pd.read_csv('../Data/kobe_dataset.csv')
6
7 # Removendo linhas com valores faltantes na coluna 'shot_made_flag'
8 df_clean = df.dropna(subset=['shot_made_flag'])
9
10 # Configuração inicial do PyCaret
11 clf1 = setup(data=df_clean, target='shot_made_flag', session_id=123, html=False, fold=10)
12
13 # Treinar o modelo de Regressão Logística com validação cruzada
14 logistic_model = create_model('lr')
15
16 # Treinar o modelo de Árvore de Decisão com validação cruzada
17 decision_tree_model = create_model('dt')
18
19 # Afinar os modelos para melhorar o desempenho
20 tuned_logistic = tune_model(logistic_model)
21 tuned_decision_tree = tune_model(decision_tree_model)
22
23 # Finalizando os modelos para que eles possam ser salvos e usados em novos dados
24 final_logistic = finalize_model(tuned_logistic)
25 final_decision_tree = finalize_model(tuned_decision_tree)
26
27 # Salvar os modelos finalizados
28 save_model(final_logistic, 'final_logistic_model')
29 save_model(final_decision_tree, 'final_decision_tree_model')
30
```

Pipeline de processamento dos dados

