

Kobe Bryant Shot Selection ML

Este é um projeto de Engenharia de Machine Learning e tem o objetivo de utilizar a base de dados kobe-bryant-shot-selection disponível no site Kaggle, URL: <https://www.kaggle.com/c/kobe-bryant-shot-selection/data> (<https://www.kaggle.com/c/kobe-bryant-shot-selection/data>). Essa base de dados trás informações como circunstâncias e localização, entre outras, dos arremessos realizados pelo astro da NBA Kobe Bryant durante sua carreira. A intenção é determinar através dos algoritmos de machine learning de foi convertida a cesta, variável alvo shot_made_flag.

1. Repositório e Template

Este projeto possui o seguinte repositório de dados URL: <https://github.com/eriktavares/KobeBryantShotSelectionML> (<https://github.com/eriktavares/KobeBryantShotSelectionML>). As estruturas de diretórios de arquivos foram baseadas no padrão Framework TDSP da Microsoft, e foi baixo o template pela URL <https://github.com/Azure/Azure-TDSP-ProjectTemplate> (<https://github.com/Azure/Azure-TDSP-ProjectTemplate>). Somente a pasta Simple_Data foi renomeada para Data, por conta da descrição que foi solicitado no enunciado da atividade (moodle). O arquivo de dados foi renomeado para

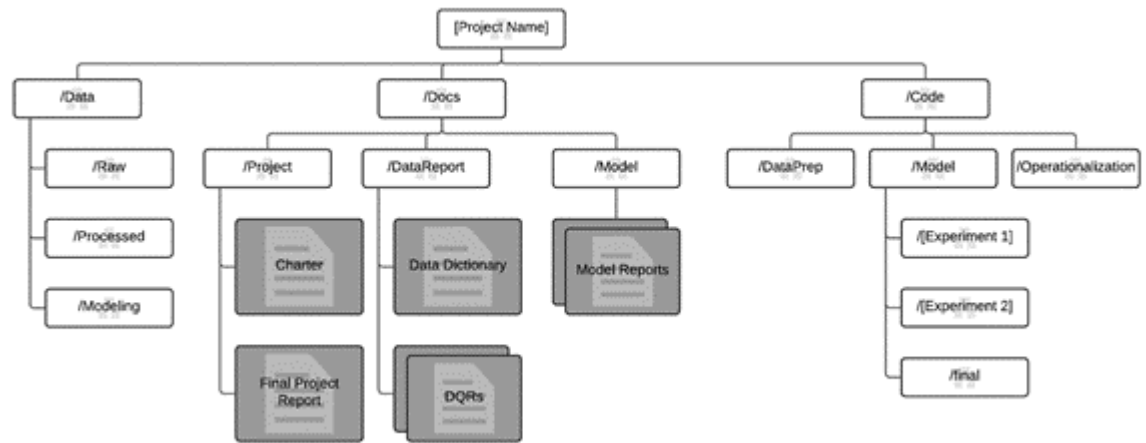
```
In [5]: import os
import warnings
import sys

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import tree, preprocessing, metrics, model_selection
import mlflow
import mlflow.sklearn

import logging

logging.basicConfig(level=logging.WARN)
logger = logging.getLogger(__name__)
```

Estrutura dos diretórios dentro do Repositório



2. Diagrama MLOps

Para início do projeto de Machine Learning é preciso pensar nos processos necessários para a execução. Dessa forma, é preciso haver inicialmente um Entendimento de Negócio, esta etapa é uma visão das atividades macro que envolve o projeto. Esse entendimento da origem a outras 3 atividades principais.

O Planejamento, onde será feito o diagrama pipeline do projeto, entre outros documentos, cronogramas, ferramentas de gestão e acompanhamento. Estrutura de repositórios de código, dados, etc. Ferramentas utilizadas para o desenvolvimento, frameworks, plataformas e etc.

Exploração de dados. Os projetos de Machine Learning são baseados em informações, então, dessa forma dependendo do tamanho do projeto, pode ou não ser contínua e deve fazer interface com planejamento e entendimento do negócio. E por último abastecer uma base de dados com informações. No caso desse trabalho específico, essa base é coletada do site kaggle.

Desenvolvimento do Experimento. Com os dados em mãos, é iniciado o processo experimental onde é realizada a modelagem. Esta etapa começa com a validação dos dados, pode ser feita por exemplo utilizando o Pycaret Setup. A preparação dos dados, onde neste trabalho são removidos os dados nulos, normalização, entre outros processos de tratamento. Treino e avaliação, caso os resultados não sejam conforme esperado, processo de melhoria e otimização, e nova avaliação. Posteriormente o registro, versionamento e deployment do modelo. Claro que podem ocorrer versionamentos durante qualquer etapa desenvolvimento. Após o deployment o modelo entra em operação, e pode ser como uma API, código, serviço, entre outros. A operação é monitorada e pode gerar novos gatilhos de desenvolvimento e melhorias, retreinamentos, etc.

em Python que automatiza fluxos de trabalho de aprendizado de máquina (<https://pycaret.gitbook.io/docs/> (<https://pycaret.gitbook.io/docs/>)). O Pycaret possui funções para os processos de preparação dos dados, Treinamentos de modelos, ajuste de hiperparâmetros, análise e interpretação, seleção de modelos e gestão de experimeto. Dessa forma, utilizar essas funções já desenvolvidas e testadas gera automação do processo de modelagem que será feito a seguir. Durante o rastreamento dos experimentos serão utilizadas as funções como a Setup()

Comparado com outras bibliotecas de aprendizado de máquina de código aberto, o PyCaret é uma biblioteca alternativa de baixo código que pode ser usada para substituir centenas de linhas de código por apenas algumas linhas. Isso torna os experimentos exponencialmente rápidos e eficientes. O PyCaret é essencialmente um wrapper Python em torno de várias bibliotecas e estruturas de aprendizado de máquina, como scikit-learn, XGBoost, LightGBM, CatBoost, spaCy, Optuna, Hyperopt, Ray e mais alguns. (<https://pycaret.gitbook.io/docs/> (<https://pycaret.gitbook.io/docs/>))

O Pycare possui diversas funções, abaixo esta sendo exemplificado algumas funções que em relação as etapas definidas no processo de experiemntação e desenvolvimento de modelos de machine learning.

Rastreio de Experimentos Setup() Essa função inicializa o experimento no PyCaret e prepara o pipeline de transformação com base em todos os parâmetros passados na função. A função de configuração deve ser chamada antes de executar qualquer outra função. Requer apenas dois parâmetros: dados e destino. Todos os outros parâmetros são opcionais (<https://pycaret.gitbook.io/docs/get-started/functions/initialize#setting-up-environment> (<https://pycaret.gitbook.io/docs/get-started/functions/initialize#setting-up-environment>)).

Treinamento:

compare_model()

Essa função treina e avalia o desempenho de todos os estimadores disponíveis na biblioteca de modelos usando validação cruzada. Dessa forma o processo de comparação dentre modelos para escolha do melhor podelo fica automatizada e pratica.

create_model() Essa função treina e avalia o desempenho de um determinado estimador usando validação cruzada. Facilita o treinamento e a busca utilizando a validação cruzada com avaliação de desempenho.

Monitoramento

plot_model() Esta função analisa o desempenho de um modelo treinado no conjunto hold-out.

Atualização calibrate_model() optimize_threshold

tune_model() Esta função ajusta os hiperparâmetros do modelo

Provisionamento(Deployment)

Funções como save_model()

Essa função salva o pipeline de transformação e um objeto de modelo treinado no diretório de trabalho atual como um arquivo pickle para uso posterior.

`deploy_model()`

Essa função implanta todo o pipeline de ML na nuvem.

<https://pycaret.gitbook.io/docs/get-started/functions/train> (<https://pycaret.gitbook.io/docs/get-started/functions/train>)

MLFLOW

Para realizar o gerenciamento do ciclo de vida deste projeto de machine learning, será utilizado o MLFLOW. Conforme a descrição do site "O MLflow é uma plataforma de código aberto para gerenciar o ciclo de vida do ML, incluindo experimentação, reprodutibilidade, implantação e um registro de modelo central. Atualmente, o MLflow oferece quatro componentes: " <https://mlflow.org/> (<https://mlflow.org/>)

- MLflow Tracking

Gravar e consultar experimentos: código, dados, configuração e resultados.

Na etapa de preparação dos dados, são utilizadas algumas funções como `log_param` e `log_metric` para gerar o log de parametros, seleção e features e de metricas, que são os tamanhos das bases, dados nulos e etc.

Também as metricas e parametros nos processos de treino e teste dos modelos.

No monitoramento da saúde do modelo, durante a operação, será comparado resultados dos experiementos que geraram o registro do modelo, com os de operação para identificação se a performance do modelo esta se mantendo.

- MLflow Projects

Empacote o código de ciência de dados em um formato para reproduzir execuções em qualquer plataforma.

Permite o monitoramento do modelo e revalidações em outros ambientes.

- MLflow Models

Implanta modelos de aprendizado de máquina em diversos ambientes de atendimento.

Esses pacotes vão ajudar criar um servidor de aplicação do modelo para requisições Http, via JSON, por exemplo, entre outros formatos possíveis.

- Model Registry

Armazena e gerencie modelos em um repositório central

Esses pacotes serão utilizados para colocar o modelo em Staging\Produção

Set up do MLFlow Server, executado no notebook `MFLOWSetup`

Streamlit

O Streamlit é uma biblioteca Python de código aberto que facilita a criação e o compartilhamento de aplicativos da Web personalizados e bonitos para aprendizado de máquina e ciência de dados. Em apenas alguns minutos, você pode criar e implantar aplicativos de dados poderosos

<https://docs.streamlit.io/> (<https://docs.streamlit.io/>)

Todas as etapas do processo podem ser disponibilizados como indicadores visuais no streamlit, através principalmente da criação de graficos interativos.

Experiment Tracking. Pode ser utilizados vizualizações dessa etapa, por exemplo, informações sobre os dados, e sobre os tratamentos utilizados.

Treino e Teste. Pode ser disponibilizado graficos e para monitoramento dessa etapa, até mesmo aproveitamento os proprios artefatos gerados pelo MLFlow.

Monitoramento da saúde. Esse passo será implementado realizando a comparação de metricas do registro com de operação com visualização dentro do Streamlit

Atualização do modelo. Pode ser comparado com os processos de registros anteriores e gerar visualizações no Streamlit..

e No Deployment podem ser gerados visualizações de metricas relacionadas as versões entre outras.

Sklearn

No projeto será utilizada a biblioteca SKlearn que possui diversas funções de código de machine learning prontas para a utilização. Essas funções são utilizadas também dentro do Pycaret, conforme consta na propria descrição do pycaret. Também existem outras bibliotecas que além do sklearn.

Experiment Tracking por exemplo, funções de tratamento de dados e metricas do sklearn serão utilizadas em conjunto com pycaret e mlflow. Treinamento. As funções utilizadas pelo pycaret serão as do sklearn para treino e teste, como Regressão Logistica e Arvore de decisão.

Monitoramento da Saúde do Modelo. Funções de metricas do sklearn principalmente, no nosso caso, principalmente o Log Loss e o F1, mas diversas outras serão registradas nos MFlow.

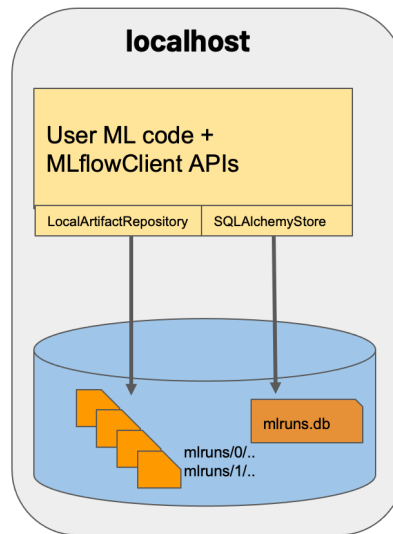
Atualização do Modelo, Entra novamente funções de treino teste e metricas utilizadas dentro do pycaret, por exemplo. Deployment. Os algoritmos da bibllioteca treinados e prontos para uso nas diversas formas de aplicação possíveis.

5. Artefatos

Aqui está sendo definido o experimento para log dentro do MLFlow, os dados do experimento serão armazenados no banco mlruns.db, e será utilizado o SQLite como banco de dados. O experimento foi definido como 'Kobe_Bryant_Shot_Experiment', o banco esta hospedado na pasta

Code, conforme os exemplos vistos. Os artefatos estão definidos para o diretório ./mlruns a partir da pasta Code.

Esta arquitetura esta representada no Cenário 02 da documentação do MLFlow <https://www.mlflow.org/docs/latest/tracking.html#scenario-2-mlflow-on-localhost-with-sqlite> (<https://www.mlflow.org/docs/latest/tracking.html#scenario-2-mlflow-on-localhost-with-sqlite>). Onde os artefatos são armazenados na pasta mlruns e as entidades no SQL Lite



Scenario 2: MLflow on the localhost with backend store as an SQLAlchemy compatible database type: **SQLite**

```
In [6]: #!/mlflow server --backend-store-uri sqlite:///mlruns.db --default-artifact-root .
```

Caso fosse definido um diretório para o armazenamento do banco e artefatos a partir da pasta dos notebooks, será preciso incluir `sqlite:///mydirectory/mlruns.db` no comando. Existem diversos outros cenários que podem ser utilizados na documentação do MLFlow.

PreparacaoDados:

São gerados alguns artefatos, a base `../Data/Processed/data_filtered.parquet`, `../Data/Operalization/base_train_test.parquet` e `'../Data/Operalization/base_operation.parquet'`. Ambas descritas na etapa abaixo. O objetivo é guardar os dados após a realização da seleção de features, e do tratamento de dados nulos por exemplo. Esses arquivos estão disponíveis para serem lidos nessas pastas por exemplo, pelo streamlit ou por outras ferramentas, e no decorrer dos experimentos podem ser lidos, não ficando salvos apenas em tempo de execução do kernel do jupyter.

Treinamento:

Durante o treinamento são gerados um artefato `Transformation Pipeline.pkl` contendo as informações da execução. e são logados parâmetros resultantes da análise feita e tags.

Durante a criação do modelo são gerados artefatos MLmodel, condayaml, model.pkl, requirements.txt, além de imagens dos plot das métricas (extra). Esses artefatos servem para caso preciso realizar a utilização do modelo, por exemplo como Python. Também está incluído um dataset './Data/Operalization/base_operation_processed.parquet' após a o processamento dos resultados com a coluna dos valores que foram preditos para facilitar os cálculos de métricas no Streamlit

```
In [7]: # Para usar o sqlite como repositório

mlflow.set_tracking_uri("sqlite:///mlruns.db")

experiment_name = 'Kobe_Bryant_Shot_Experiment'
experiment = mlflow.get_experiment_by_name(experiment_name)
if experiment is None:
    experiment_id = mlflow.create_experiment(experiment_name)
    experiment = mlflow.get_experiment(experiment_id)
experiment_id = experiment.experiment_id
```

6. PreparacaoDados

A preparação dos dados é um passo importante no processo de Auto ML, nesta etapa será carregado o tratado os dados para os processos

seguintes. Os dados são carregados do arquivo './Data/kobe_dataset.csv' que veio do site kaggle, o tamanho inicial foi registrado com o nome Tamanho/Linhas - Base Entrada. A variável alvo 'shot_made_flag' está com dados faltantes, a quantidade de linhas foi registrado como Quantidade de shot_made_flag Faltante

O tamanho resultante da remoção dos dados faltantes foi registrado como Tamanho/Linhas - Base sem dados faltantes. Posteriormente foi filtrada para somente os dados com arremessos de 2 pontos 2PT Field Goal e salvo em './Data/Processed/data_filtered.parquet'

Essa base foi separada em treino/teste 80% e 20% para operação. Registrados os tamanhos no MLFlow como Tamanho/Linhas - Base Treino/Teste e Tamanho/Linhas - Base Operação, respectivamente. Salvos em './Data/Operalization/base_train_test.parquet' e './Data/Operalization/base_operation.parquet'

Os dados com arremessos de 3 pontos 3PT Field Goal, registrado o tamanho Tamanho/Linhas - Base Novidade e armazenado em './Data/Operalization/base_novelty.parquet'.

Name Value Quantidade de shot_made_flag Faltante 5000

Tamanho/Linhas - Base Entrada 30697

Tamanho/Linhas - Base Novidade 5412

Tamanho/Linhas - Base Operação 4057

Tamanho/Linhas - Base Treino/Teste 16228

Tamanho/Linhas - Base sem dados faltantes 25697

Essa separação dos dados treino/teste foi feita utilizando `shuffle=True` para que seja feito de forma aleatória, e o parâmetro `stratify array` como default, garantido que seja aleatória e estratificada. Aleatória que os dados serão misturados, e o estratificado garante a proporcionalidade das amostras. Essa técnica evita que os dados sejam divididos de forma a não expressar a real existência da informação. Por exemplo, se todos os dados de cesta convertidos estivessem no início do dataset ou os erros no final, uma divisão mantendo essa ordenação, iria disponibilizar para o modelo, informação agrupada com uma tendência predominando, diferente dos dados totais. Dessa forma a modelagem ficaria prejudicada, assim como a validação com os dados de teste. Assim, a aleatoriedade e a manutenção das proporcionalidades na divisão dos dados, garante que o modelo está recebendo a informação coerente a totalidade dos dados.

```

In [8]: # COLOCAR RUN DE LEITURA DE DADOS
# PARAMETROS: top_features,
# METRICS: SHAPE de cada base de dados
# ARTIFACTS: nenhum

import warnings
warnings.filterwarnings('ignore')
top_features = ['lat', 'lon', 'minutes_remaining' , 'period', 'playoffs', 'shot_d
target_col = 'shot_made_flag'
target_col_label = 'shot_made_label'

with mlflow.start_run(experiment_id=experiment_id, run_name = 'PreparacaoDados',

    #Leitura de dados
    path_kb_data_input= '../Data/kobe_dataset.csv'
    df_kb_all = pd.read_csv(path_kb_data_input)
    mlflow.log_metric("Tamanho/Linhas - Base Entrada", df_kb_all.shape[0])

    #Descrição Variável alvo
    mapa ={0 : 'Errou', 1 : 'Cesta'}
    df_kb_all['shot_made_label'] = pd.DataFrame(df_kb_all [target_col].map(mapa))
    df_kb_all[[target_col, target_col_label]]

    #Remoção de dados Faltantes na Shot_made_Flag
    mlflow.log_metric("Quantidade de {} Faltante".format(target_col), df_kb_all[
    df_kb = df_kb_all[df_kb_all['shot_made_flag'].notnull()].reset_index()
    df_kb[target_col] = df_kb[target_col].astype(int)
    mlflow.log_metric("Tamanho/Linhas - Base sem dados faltantes", df_kb.shape[0]

    #Seleção de Features
    df_kb_tf = df_kb [top_features + ['shot_type', target_col]].copy()
    mlflow.log_param("top_features", top_features)

    #Filtro 2PT Field Goal

    df_kb_2PT = df_kb_tf[df_kb_tf['shot_type'] == '2PT Field Goal'].copy().drop(
    df_kb_2PT.to_parquet('../Data/Processed/data_filtered.parquet')

    # Separação da base com 80%/20% test_size=0.2
    #stratifyarray-like, default=None If not None, data is split in a stratified
    #shuffle = True
    df_kb_tt, df_kb_operation, ytrain, ytest = model_selection.train_test_split(c
                                                                    df_kb
                                                                    test_
                                                                    shuff

    mlflow.log_param("Percentual Operação", '0.2')
    df_kb_tt[target_col] = ytrain
    df_kb_operation[target_col] = ytest

    mlflow.log_metric("Tamanho/Linhas - Base Treino/Teste", df_kb_tt.shape[0])
    mlflow.log_metric("Tamanho/Linhas - Base Operação", df_kb_operation.shape[0])

```

```

#Base 3PT Field Goal
df_kb_novelty = df_kb[df_kb['shot_type'] == '3PT Field Goal'].copy().drop('shot_type', axis=1)
mlflow.log_metric("Tamanho/Linhas - Base Novidade", df_kb_novelty.shape[0])

#Envio datasets para "/Data/operalization/base_{train/test}.parquet"
df_kb_tt.to_parquet('../Data/Operalization/base_train_test.parquet')
df_kb_operation.to_parquet('../Data/Operalization/base_operation.parquet')
df_kb_novelty.to_parquet('../Data/Operalization/base_novelty.parquet')

#Label_map = df_wine[['target', 'target_label']].drop_duplicates()
#drop_cols = ['target_label']
#df_wine.drop(drop_cols, axis=1, inplace=True)
#print(df_kb.shape)

#df_kb.head()
#df_kb.keys()

mlflow.end_run()

```

7. Treinamento

Essa função inicializa o experimento no PyCaret e cria o pipeline de transformação com base em todos os parâmetros passados na função. A função de configuração deve ser chamada antes de executar qualquer outra função. São necessários dois parâmetros obrigatórios: data e destino. Todos os outros parâmetros são opcionais. <https://pycaret.gitbook.io/docs/get-started/functions/initialize#setting-up-environment> (<https://pycaret.gitbook.io/docs/get-started/functions/initialize#setting-up-environment>).

Neste caso os parâmetros obrigatórios são df_kb_tt (base de dados) e o nome da coluna da variável alvo.

Os parâmetros para gerar os logs do experimento no MLFLOW.

- log_experiment = True,
- experiment_name = experiment_name,
- log_plots = True

As métricas default do PyCaret são: 'Accuracy', 'AUC', 'Recall', 'Precision', 'F1', 'Kappa', 'MCC'. Porém será adicionado também a Métrica Perda de Log

Perda de log, também conhecida como perda logística ou perda de entropia cruzada.

Esta é a função de perda usada na regressão logística (multinomial) e em suas extensões, como redes neurais, definida como a probabilidade logarítmica negativa de um modelo logístico que retorna probabilidades y_pred para seus dados de treinamento y_true. A perda de log é definida apenas para dois ou mais rótulos Adicionando Metric Loss Log. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html)

7a - Regressão Logística

```

In [9]: #import pycaret.classification as pc
# COLOCAR RUN DE TREINAMENTO DE MODELOS
# PARAMETROS: fold_strategy, fold, model_name, registered_model_name, cross_validation
# METRICS: auto sklearn
# ARTIFACTS: plots
# add Log Loss metric in pycaret
import pycaret.classification as pc
from sklearn.metrics import log_loss

registered_model_name = 'modelo_regressão_kb'
model_name = 'lr'
probability_threshold = 0.5
cross_validation = True
fold_strategy = 'stratifiedkfold',
fold = 10
with mlflow.start_run(experiment_id=experiment_id, run_name = 'Treinamento', nest
    # train/test
    s = pc.setup(data = df_kb_tt,
                  target = target_col,
                  train_size=0.7,
                  silent = True,
                  fold_strategy = 'stratifiedkfold',
                  fold = fold,
                  log_experiment = True,
                  experiment_name = experiment_name,
                  log_plots = True
                )
    pc.add_metric('logloss', 'LogLoss', log_loss, greater_is_better=False)
    bestmodel = pc.create_model(model_name,
                                cross_validation = cross_validation,
                                probability_threshold=probability_threshold)

    # Log do run, e nao do modelo respectivo
    classification_plots = [ 'auc','pr','confusion_matrix',
                             #'error', 'class_report',
                             'threshold', 'f1', 'logloss',
                             'learning','vc','feature',
                           ]
    for plot_type in classification_plots:
        print('=> Aplicando plot ', plot_type)
        try:
            artifact = pc.plot_model(bestmodel, plot=plot_type, save=True, use_tr
            mlflow.log_artifact(artifact)
        except:
            print('=> Nao possivel plotar: ', plot_type )
            continue

    #pc.save_model(bestmodel, f'./{registered_model_name}')
    # Carrega novamente o pipeline + bestmodel
    #model_pipe = pc.load_model(f'./{registered_model_name}')

mlflow.end_run()

```

INFO:logs:Saving 'Feature Importance.png'
INFO:logs:Visual Rendered Successfully

INFO:logs:plot_model() succesfully complete
d.....



```
In [10]: pc.get_metrics()
```

Out[10]:

	Name	Display Name	Score Function	S
ID				
acc	Accuracy	Accuracy	<function accuracy_score at 0x0000022712FD68C8>	acc
auc	AUC	AUC	<function roc_auc_score at 0x0000022712FD1268>	make_scorer(roc_auc_score, needs_proba=True)
recall	Recall	Recall	<pycaret.internal.metrics.BinaryMulticlassScorer at 0x0000022712FE3778>	make_scorer(recall_score, average=weighted)
precision	Precision	Prec.	<pycaret.internal.metrics.BinaryMulticlassScorer at 0x0000022712FE3778>	make_scorer(precision_score, average=weighted)
f1	F1	F1	<pycaret.internal.metrics.BinaryMulticlassScorer at 0x0000022712FE3778>	make_scorer(f1_score, average=weighted)
kappa	Kappa	Kappa	<function cohen_kappa_score at 0x0000022712FD68C8>	make_scorer(cohen_kappa_score)
mcc	MCC	MCC	<function matthews_corrcoef at 0x0000022712FE3778>	make_scorer(matthews_corrcoef)
logloss	LogLoss	LogLoss	<function log_loss at 0x0000022712FE8378>	make_scorer(log_loss, greater_is_better=False)

```
In [11]: pred_holdout = pc.predict_model(bestmodel)
```

```
INFO:logs:Initializing predict_model()
INFO:logs:predict_model(drift_kwargs=None, display=None, ml_usecase=MLUsecase.C
LASSIFICATION, verbose=True, round=4, raw_score=False, drift_report=False, enco
ded_labels=False, probability_threshold=None, estimator=CustomProbabilityThresh
oldClassifier(C=1.0, class_weight=None,
                                classifier=LogisticRegression(C=1.0,
                                                                class_weight
=None,
                                                                dual=False,
                                                                fit_intercep
t=True,
                                                                intercept_sc
aling=1,
                                                                l1_ratio=Non
e,
                                                                max_iter=100
0,
                                                                multi_class
='auto',
                                                                n_jobs=None,
                                                                penalty='l
2',
                                                                random_state
=4678,
                                                                solver='lbfg
s',
                                                                tol=0.0001,
                                                                verbose=0,
                                                                warm_start=F
alse),
                                dual=False, fit_intercept=True,
                                intercept_scaling=1, l1_ratio=None,
                                max_iter=1000, multi_class='auto',
                                n_jobs=None, penalty='l2',
                                probability_threshold=0.5,
                                random_state=4678, solver='lbfgs',
                                tol=0.0001, verbose=0, warm_start=False))
INFO:logs:Checking exceptions
INFO:logs:Preloading libraries
INFO:logs:Preparing display monitor
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss
0	Logistic Regression	0.5701	0.5859	0.4945	0.5387	0.5156	0.1305	0.1309	14.8471

7C - Arvore de Decisão

```

In [12]: import pycaret.classification as pc
# COLOCAR RUN DE TREINAMENTO DE MODELOS
# PARAMETROS: fold_strategy, fold, model_name, registered_model_name, cross_validation
# METRICS: auto sklearn
# ARTIFACTS: plots
# add Log Loss metric in pycaret

registered_model_name = 'modelo_arvore_kb'
model_name = 'dt'
probability_threshold = 0.5
cross_validation = True
fold_strategy = 'stratifiedkfold',
fold = 10
with mlflow.start_run(experiment_id=experiment_id, run_name = 'Treinamento', nest
    # train/test

    #pc.add_metric('logloss', 'LogLoss', log_loss, greater_is_better=False)
    bestmodel = pc.create_model(model_name,
                                cross_validation = cross_validation,
                                probability_threshold=probability_threshold)

    # Log do run, e nao do modelo respectivo
    #classification_plots = [ 'f1','logloss']
    # for plot_type in classification_plots:
    #     print('=> Aplicando plot ', plot_type)
    #     try:
    #         artifact = pc.plot_model(bestmodel, plot=plot_type, save=True, use_t
    #         mlflow.log_artifact(artifact)
    #     except:
    #         print('=> Nao possivel plotar: ', plot_type )
    #         continue

    #pc.save_model(bestmodel, f'./{registered_model_name}')
    # Carrega novamente o pipeline + bestmodel
    #model_pipe = pc.load_model(f'./{registered_model_name}')

mlflow.end_run()

```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss
Fold								
0	0.5326	0.5102	0.6062	0.5154	0.5571	0.0691	0.0701	16.1447
1	0.5546	0.5335	0.6171	0.5354	0.5734	0.1123	0.1135	15.3846
2	0.5423	0.5156	0.5935	0.5249	0.5571	0.0872	0.0879	15.8102
3	0.5079	0.4807	0.5917	0.4939	0.5384	0.0206	0.0210	16.9960
4	0.5423	0.5248	0.5826	0.5254	0.5525	0.0866	0.0871	15.8102
5	0.5211	0.4963	0.5644	0.5057	0.5334	0.0446	0.0449	16.5399

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss
Fold								
6	0.5625	0.5380	0.6298	0.5422	0.5827	0.1283	0.1299	15.1109
7	0.5335	0.5061	0.5662	0.5174	0.5407	0.0686	0.0689	16.1142
8	0.5484	0.5197	0.6123	0.5306	0.5685	0.0999	0.1010	15.5974
9	0.5427	0.5229	0.5808	0.5263	0.5522	0.0874	0.0878	15.7937
Mean	0.5388	0.5148	0.5944	0.5217	0.5556	0.0804	0.0812	15.9302
Std	0.0151	0.0164	0.0206	0.0135	0.0150	0.0299	0.0303	0.5210

```

INFO:logs:create_model_container: 2
INFO:logs:master_model_container: 2
INFO:logs:display_container: 4
INFO:logs:CustomProbabilityThresholdClassifier(ccp_alpha=0.0, class_weight=None,
                                              classifier=DecisionTreeClassifier(ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini',
                                              max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=4678,
                                              splitter='best'),
                                              criterion='gini', max_depth=None,
                                              max_features=None, max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1, min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              probability_threshold=0.5,
                                              random_state=4678, splitter='best')
INFO:logs:create_model() successfully completed.....

```

7C. Escolha Livre

Uma forma de realização de uma escolha para um algoritmo seria utilizar o função `compare_models` do `Pycaret`. O `sort` define o parâmetro de ordenação, nesse caso foi utilizado o `Log Loss`. Para a escolha do melhor modelo, pode ser utilizada a função `compare_models`, e neste caso o melhor resultado foi o `Gradient Boosting Classifier` e `Ada Boost Classifier`. Então seria escolhido o `Gradient Boosting Classifier`. Se for utilizado uma outra metrica para o `sort`, outro modelo pode ser selecionado como melhor resultado, ou até mesmo em outra simulação.

```
In [27]: with mlflow.start_run(experiment_id=experiment_id, run_name = 'Compare', nested=True):
          best_model = pc.compare_models(n_select = 1, sort='logloss')
          mlflow.autolog()
          mlflow.end_run()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss	TT (Sec)
ada	Ada Boost Classifier	0.5873	0.5957	0.3713	0.6263	0.4658	0.1640	0.1792	14.2547	0.1280
gbc	Gradient Boosting Classifier	0.5863	0.5947	0.3892	0.6169	0.4771	0.1630	0.1750	14.2881	0.3690
lda	Linear Discriminant Analysis	0.5764	0.5957	0.5044	0.5720	0.5359	0.1491	0.1502	14.6317	0.0310
ridge	Ridge Classifier	0.5762	0.0000	0.5043	0.5718	0.5358	0.1488	0.1498	14.6378	0.0200
lr	Logistic Regression	0.5754	0.5961	0.5005	0.5714	0.5334	0.1470	0.1482	14.6652	0.8260
lightgbm	Light Gradient Boosting Machine	0.5648	0.5842	0.4759	0.5608	0.5147	0.1251	0.1265	15.0301	0.0640
rf	Random Forest Classifier	0.5532	0.5604	0.5456	0.5391	0.5422	0.1060	0.1060	15.4315	0.4300
et	Extra Trees Classifier	0.5473	0.5499	0.5632	0.5317	0.5469	0.0954	0.0956	15.6353	0.4970
knn	K Neighbors Classifier	0.5419	0.5487	0.5086	0.5290	0.5185	0.0819	0.0820	15.8238	0.1270
nb	Naive Bayes	0.5365	0.5831	0.7159	0.5200	0.5978	0.0828	0.0831	16.0092	0.0160
dt	Decision Tree Classifier	0.5351	0.5148	0.5812	0.5188	0.5481	0.0726	0.0731	16.0579	0.0360
svm	SVM - Linear Kernel	0.5305	0.0000	0.5218	0.5090	0.4251	0.0601	0.0776	16.2159	0.0920
dummy	Dummy Classifier	0.5148	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	16.7570	0.0180
qda	Quadratic Discriminant Analysis	0.4981	0.4498	0.5713	0.4839	0.5065	0.0004	0.0024	17.3351	0.0230

```
INFO:logs:create_model_container: 43
INFO:logs:master_model_container: 43
INFO:logs:display_container: 21
INFO:logs:AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                             n_estimators=50, random_state=4678)
INFO:logs:compare_models() succesfully complete
d.....
```

2022/04/21 11:39:27 INFO mlflow.tracking.fluent: Autologging successfully enabled for sklearn.
2022/04/21 11:39:27 INFO mlflow.tracking.fluent: Autologging successfully enabled for lightgbm.

In [28]: best_model

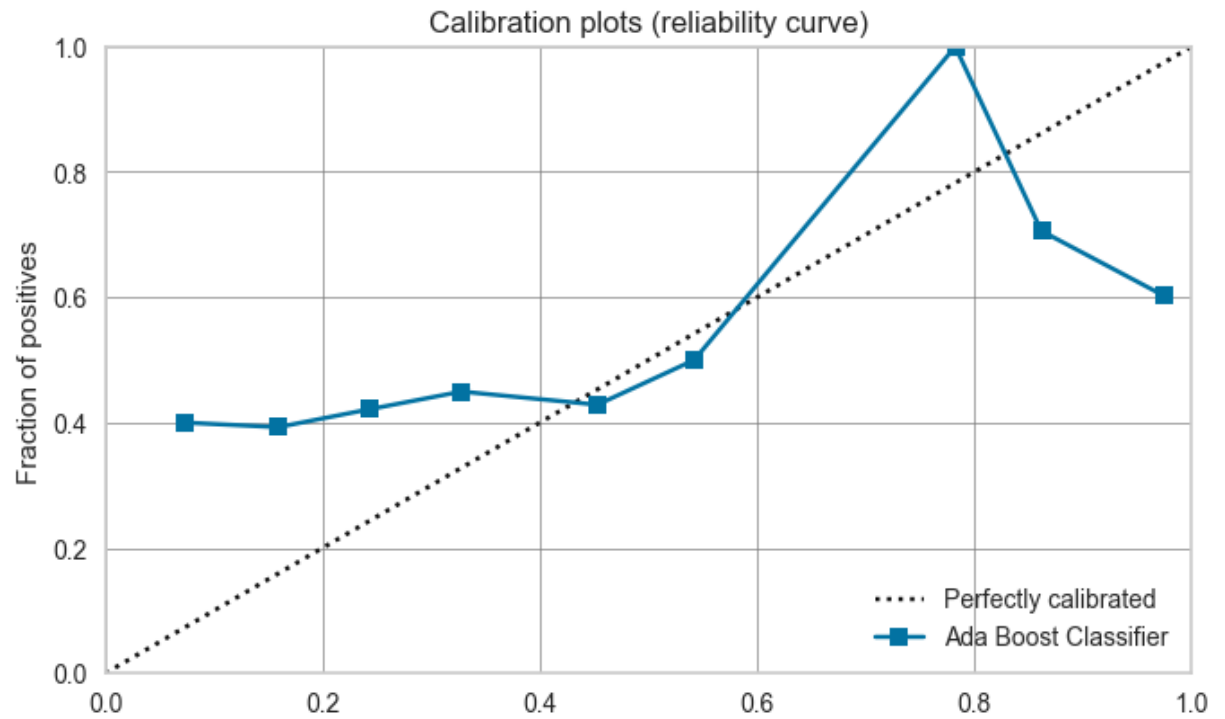
Out[28]: AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0, n_estimators=50, random_state=4678)

```
In [29]: with mlflow.start_run(experiment_id=experiment_id, run_name = 'Tune', nested=True):
         tuned_model = pc.tune_model(best_model,
                                   optimize = 'logloss',
                                   search_library = 'scikit-learn',
                                   search_algorithm = 'random',
                                   n_iter = 4)
         mlflow.end_run()
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss
Fold								
0	0.5854	0.5779	0.3448	0.6333	0.4465	0.1589	0.1778	14.3203
1	0.5924	0.5966	0.3539	0.6457	0.4572	0.1733	0.1934	14.0771
2	0.5915	0.5938	0.3775	0.6322	0.4727	0.1727	0.1880	14.1075
3	0.5871	0.5938	0.3593	0.6306	0.4578	0.1631	0.1800	14.2595
4	0.5915	0.6101	0.3412	0.6505	0.4476	0.1709	0.1934	14.1075
5	0.5783	0.5883	0.3503	0.6146	0.4462	0.1453	0.1603	14.5635
6	0.5995	0.6077	0.3612	0.6589	0.4666	0.1876	0.2094	13.8338
7	0.5898	0.5896	0.3503	0.6412	0.4531	0.1679	0.1876	14.1683
8	0.6083	0.6148	0.3822	0.6698	0.4867	0.2066	0.2279	13.5298
9	0.6035	0.6190	0.3757	0.6613	0.4792	0.1966	0.2172	13.6939
Mean	0.5927	0.5992	0.3596	0.6438	0.4614	0.1743	0.1935	14.0661
Std	0.0084	0.0125	0.0136	0.0159	0.0136	0.0173	0.0190	0.2904

```
INFO:logs:create_model_container: 44
INFO:logs:master_model_container: 44
INFO:logs:display_container: 22
INFO:logs:AdaBoostClassifier(algorithm='SAMME', base_estimator=None, learning_rate=0.4,
                             n_estimators=280, random_state=4678)
INFO:logs:tune_model() succesfully complete
d.....
```

```
In [30]: with mlflow.start_run(experiment_id=experiment_id, run_name = 'Evaluate Tuned', r
        calibrated_model = pc.calibrate_model(tuned_model, method='sigmoid', calibrat
        pc.plot_model(calibrated_model, plot='calibration')
mlflow.end_run()
```



INFO:logs:Visual Rendered Successfully
INFO:logs:plot_model() succesfully complete
d.....

```
In [31]: pc.optimize_threshold(calibrated_model, optimize = 'logloss');
```

```
INFO:logs:Ada Boost Classifier Imported succesfully
INFO:logs:Starting cross validation
INFO:logs:Cross validating with StratifiedKFold(n_splits=10, random_state=None, shuffle=False), n_jobs=-1
INFO:logs:Calculating mean and std
INFO:logs:Creating metrics dataframe
INFO:logs:Finalizing model
INFO:logs:Uploading results into container
INFO:logs:Uploading model into container now
INFO:logs:create_model_container: 51
INFO:logs:master_model_container: 51
INFO:logs:display_container: 29
INFO:logs:CustomProbabilityThresholdClassifier(base_estimator=AdaBoostClassifier(algorithm='SAMME',
base_estimator=None,
learning_rate=0.4,
n_estimators=280,
```

```
In [32]: pred_holdout = pc.predict_model(calibrated_model)
```

```
INFO:logs:Initializing predict_model()
INFO:logs:predict_model(drift_kwargs=None, display=None, ml_usecase=MLUsecase.CLASSIFICATION, verbose=True, round=4, raw_score=False, drift_report=False, encoded_labels=False, probability_threshold=None, estimator=CalibratedClassifierCV(base_estimator=AdaBoostClassifier(algorithm='SAMME',
base_estimator=None,
learning_rate=0.4,
n_estimators=280,
random_state=4678),
cv=5, method='sigmoid'))
INFO:logs:Checking exceptions
INFO:logs:Preloading libraries
INFO:logs:Preparing display monitor
```

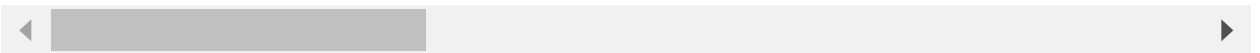
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss
0	Ada Boost Classifier	0.5886	0.5871	0.3418	0.5969	0.4347	0.1474	0.1616	14.2086

In [33]: pred_holdout

Out[33]:

	lat	lon	shot_distance	minutes_remaining_0	minutes_remaining_1	minutes_re
0	33.950298	-118.159798	14.0	0.0	0.0	
1	34.027302	-118.420799	15.0	0.0	0.0	
2	34.044300	-118.269798	0.0	0.0	0.0	
3	33.981300	-118.369797	11.0	0.0	0.0	
4	33.947300	-118.284798	9.0	0.0	0.0	
...
4864	33.964298	-118.268799	8.0	1.0	0.0	
4865	33.886299	-118.129799	21.0	1.0	0.0	
4866	33.903301	-118.381798	18.0	0.0	1.0	
4867	33.921299	-118.392799	17.0	0.0	0.0	
4868	33.896301	-118.192802	16.0	0.0	0.0	

4869 rows × 26 columns



```
In [34]: def eval_metrics(actual, pred):  
    return ({'Prec.': metrics.precision_score(actual, pred),  
            'Recall': metrics.recall_score(actual, pred),  
            'F1': metrics.f1_score(actual, pred),  
            'LogLoss': metrics.log_loss(actual, pred),  
            'AUC': metrics.roc_auc_score(actual, pred),  
            'Accuracy': metrics.accuracy_score(actual, pred),  
            'Kappa': metrics.cohen_kappa_score(actual, pred),  
            'MCC': metrics.matthews_corrcoef(actual, pred)})
```

Otimização dos hiperparâmetros

8 Registro do Modelo

```

In [35]: from mlflow.tracking import MlflowClient
import mlflow
import warnings
warnings.filterwarnings('ignore')
#mlflow.set_registry_uri("sqlite:///mlruns.db")
#from mlflow.models.signature import infer_signature
from sklearn import tree, preprocessing, metrics, model_selection
#from mlflow.models.signature import ModelSignature
mode_registre=calibrated_model
model_version = -1
registered_model_name = 'Modelo Kobe Bryant'

with mlflow.start_run(experiment_id=experiment_id, run_name = 'RegistroModelo', r
    pred_holdout = pc.predict_model(calibrated_model)
    mr=metrics.precision_score(pred_holdout[target_col], pred_holdout['Label'])
    # Test set
    #pred_holdout = pc.predict_model(model_to_registre)
    #pr = metrics.precision_score(pred_holdout[target_col], pred_holdout['Label'])
    #if pr > min_precision:
        # print(f'=> Aceito o modelo com precisão {pr} (min: {min_precision})')
        # Pycaret exporta junto o pipeline de preprocessamento
    pc.save_model(mode_registre, f'./{registered_model_name}')
        # Carrega novamente o pipeline + bestmodel
    model_pipe = pc.load_model(f'./{registered_model_name}')
        # Assinatura do Modelo Inferida pelo MLFlow
    model_features = list(df_kb_tt.drop(target_col, axis=1).columns)
        #inf_signature = infer_signature(DataBin[model_features], model_pipe.prec
        # Exemplo de entrada para o MLmodel
        #input_example = {x: DataBin[x].values[:nexamples] for x in model_feature
        # Log do pipeline de modelagem do sklearn e registrar como uma nova versao
    mlflow.sklearn.log_model(
        sk_model=model_pipe,
        artifact_path="sklearn-model",
        registered_model_name=registered_model_name,
            #signature = inf_signature,
            #input_example = input_example
        )
        # Criacao do cliente do servico MLFlow e atualizacao versao modelo
    client = MlflowClient()
    if model_version == -1:
        model_version = client.get_latest_versions(registered_model_name)[-1].ver
        # Registrar o modelo como staging
    client.transition_model_version_stage(
        name=registered_model_name,
        version=model_version, # Verificar com usuario qual versao
        stage="Staging"
    )
    result= eval_metrics(pred_holdout[target_col].values, pred_holdout['Label'].v
    result_title=''
    result_value=''
    for metric in result.keys():
        mlflow.log_metric(metric, result[metric])
        print('{:<8}\t{:<0.2f}'.format(metric, result[metric]))

    mlflow.log_metric('Version', model_version)
#else:

```



```
#print(f'=> Rejeitado o modelo com precisão {pr} (min: {min_precision})')
```

```
mlflow.end_run()
```

```
INFO:logs:Initializing predict_model()
INFO:logs:predict_model(drift_kwargs=None, display=None, ml_usecase=MLUsecase.C
LASSIFICATION, verbose=True, round=4, raw_score=False, drift_report=False, enco
ded_labels=False, probability_threshold=None, estimator=CalibratedClassifierCV
(base_estimator=AdaBoostClassifier(algorithm='SAMME',
                                     base_estimator=None,
                                     learning_rate=0.4,
                                     n_estimators=280,
                                     random_state=4678),
                                     cv=5, method='sigmoid'))
INFO:logs:Checking exceptions
INFO:logs:Preloading libraries
INFO:logs:Preparing display monitor
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	LogLoss
0	Ada Boost Classifier	0.5886	0.5871	0.3418	0.5969	0.4347	0.1474	0.1616	14.2086

```
INFO:logs:Initializing save_model()
INFO:logs:save_model(kwargs={}, verbose=True, prep_pipe=Pipeline(memory=None,
steps=[('dtypes',
        DataTypes_Auto_infer(categorical_features=[],
                               display_types=False, features_todrop=[],
                               id_columns=[],
                               ml_usecase='classification',
                               numerical_features=[],
                               target='shot_made_flag',
                               time_features=[])),
        ('imputer',
         Simple_Imputer(categorical_strategy='not_available',
                        fill_value_categorical=None,
                        fill_value_numerical=None,
                        nume...
        ('scaling', 'passthrough'), ('P_transform', 'passthrough'),
        ('binn', 'passthrough'), ('rem_outliers', 'passthrough'),
        ('cluster_all', 'passthrough'),
        ('dummy', Dummify(target='shot_made_flag')),
        ('fix_perfect', Remove_100(target='shot_made_flag')),
        ('clean_names', Clean_Colum_Names()),
        ('feature_select', 'passthrough'), ('fix_multi', 'passthroug
h')),
        ('dfs', 'passthrough'), ('pca', 'passthrough')],
        verbose=False), model_name=./Modelo Kobe Bryant, model=CalibratedClass
ifierCV(base_estimator=AdaBoostClassifier(algorithm='SAMME',
                                             base_estimator=None,
                                             learning_rate=0.4,
                                             n_estimators=280,
                                             random_state=4678),
                                             cv=5, method='sigmoid'))
INFO:logs:Adding model into prep_pipe
INFO:logs:./Modelo Kobe Bryant.pkl saved in current working directory
INFO:logs:Pipeline(memory=None,
```

```

steps=[('dtypes',
        DataTypes_Auto_infer(categorical_features=[],
                              display_types=False, features_todrop=[],
                              id_columns=[],
                              ml_usecase='classification',
                              numerical_features=[],
                              target='shot_made_flag',
                              time_features=[])),
        ('imputer',
         Simple_Imputer(categorical_strategy='not_available',
                        fill_value_categorical=None,
                        fill_value_numerical=None,
                        nume...
        ('fix_perfect', Remove_100(target='shot_made_flag')),
        ('clean_names', Clean_Column_Names()),
        ('feature_select', 'passthrough'), ('fix_multi', 'passthroug
h'),
        ('dfs', 'passthrough'), ('pca', 'passthrough'),
        ['trained_model',
         CalibratedClassifierCV(base_estimator=AdaBoostClassifier(algor
ithm='SAMME',
                                                                    base_
estimator=None,
                                                                    learn
ing_rate=0.4,
                                                                    n_est
imators=280,
                                                                    rando
m_state=4678),
                                                                    cv=5, method='sigmoid')]],
        verbose=False)

```

INFO:logs:save_model() successfully complete

d.....

INFO:logs:Initializing load_model()

INFO:logs:load_model(verbose=True, authentication=None, platform=None, model_name=./Modelo Kobe Bryant)

Transformation Pipeline and Model Successfully Saved

Transformation Pipeline and Model Successfully Loaded

Registered model 'Modelo Kobe Bryant' already exists. Creating a new version of this model...

2022/04/21 12:08:41 INFO mlflow.tracking._model_registry.client: Waiting up to 300 seconds for model version to finish creation. Model name: Modelo Kobe Bryant, version 11

Prec.	0.60
Recall	0.34
F1	0.43
LogLoss	14.21
AUC	0.57
Accuracy	0.59
Kappa	0.15
MCC	0.16

Created version '11' of model 'Modelo Kobe Bryant'.

Ativando o serviço Server para o modelo Modelo Kobe Bryant em Staging, execução em outro

```
In [22]: #import os
#os.environ['MLFLOW_TRACKING_URI'] = 'sqlite:///mlruns.db'

#!mlflow models serve -m "models:/modelo_cancer/Staging" --no-conda -p 5001
```

8.1 Revalidação

Para a revalidação será feito utilizando os dados com 3PT Field Goal, que são diferentes porque o acerremeço de 3 pontos é mais distânte do de 2 pontos. Dessa forma representa um novo conjunto de dados com características, digamos que não esperadas pelo modelo, que foi treinado com dados de arremessos de 2pts.

Abaixo esta uma função para calculo das principais metricas e retorno em formato dicionário.

O Serviço vai enviar uma request http para o serviço da API que realiza a predição e retorna os valores preditos em um JSON que é convertido para DataFrame e então são calculadas as metricas. Todas as metricas são então salvas como log metric no MLFlow

```
In [36]: import pandas as pd
import requests
from sklearn.metrics import log_loss
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import auc

#Configuração do request
host = 'localhost'
port = '5001'
url = f'http://{host}:{port}/invocations'
headers = {'Content-Type': 'application/json',}

with mlflow.start_run(experiment_id=experiment_id, run_name = 'RevalidaçãoModelo')
    #Dados para revalidação
    df_kb_op=pd.read_parquet('../Data/Operalization/base_operation.parquet')
    http_data = df_kb_op.drop(target_col,axis=1).to_json(orient='split')
    r = requests.post(url=url, headers=headers, data=http_data)
    df_kb_op.loc[:, 'operation_label'] = pd.read_json(r.text).values[:,0]
    df_kb_op.to_parquet('../Data/Operalization/base_operation_processed.parquet')
    #ll = log_loss(df_kb_op[target_col], df_kb_op['operation_label'])
    #f1 = f1_score(df_kb_op[target_col], df_kb_op['operation_label'])
    #acc= accuracy_score(df_kb_op[target_col], df_kb_op['operation_label'])
    #auc=auc(df_kb_op[target_col], df_kb_op['operation_label'])
    result= eval_metrics(df_kb_op[target_col], df_kb_op['operation_label'])
    result_title=''
    result_value=''
    for metric in result.keys():
        mlflow.log_metric(metric, result[metric])
        print('{:<8}\t{:<0.2f}'.format(metric, result[metric]))

    mlflow.log_metric('Version', model_version)
mlflow.end_run()
```

Prec.	0.63
Recall	0.39
F1	0.48
LogLoss	13.74
AUC	0.59
Accuracy	0.60
Kappa	0.19
MCC	0.20

In []:

Comparação

8.a Aderência com Novo Conjunto de Dados

```
In [37]: df_ex = mlflow.search_runs([experiment_id], order_by=["metrics.m DESC"])
df_ex_fh = df_ex[df_ex['status'] == 'FINISHED'].copy()
df_ex_fh_rv = df_ex_fh[df_ex['tags.mlflow.runName'] == 'RevalidaçãoModelo'].copy()
df_ex_fh_rg = df_ex_fh[df_ex['tags.mlflow.runName'] == 'Gradient Boosting Classifier'].copy()
metrics_select = ['tags.mlflow.runName', 'metrics.LogLoss', 'metrics.F1', 'metrics.Accuracy', 'metrics.Prec', 'metrics.Recall']
df_ex_fh_rv_fl = df_ex_fh_rv[metrics_select].copy()
df_ex_fh_rg_fl = df_ex_fh_rg[metrics_select].copy()
#print(df_ex_fh_rv_fl.keys())
df_rs = pd.concat([pd.DataFrame(df_ex_fh_rv_fl.iloc[:1]), df_ex_fh_rg_fl.iloc[:1]]).reset_index(drop=True)
df_rs.to_parquet('../Data/Operalization/results/results01.parquet')
df_rs
```

Out[37]:

	tags.mlflow.runName	metrics.LogLoss	metrics.F1	metrics.Accuracy	metrics.Prec.	metrics.Recall
0	RevalidaçãoModelo	13.740678	0.478008	0.602169	0.630009	0.385009
62	Gradient Boosting Classifier	14.005300	0.461300	0.594500	0.626600	0.365200

8b Monitoramento do Modelo

Na comparação entre os resultados obtidos no experimento de Ada Boost Classifier que foi o melhor modelo escolhido pelo Pycaret e o RelalidaçãoModelo que foi realizado com os dados de arremessos de 3 pontos. As metricas ficaram muito próximas, significa que não houve perca de performace, com resultados até mesmo com metricas acima dos resultados coletados pelo algoritmo de melhor resultado. O correto seria com esses dados novos, realizar uma nova amostra de dados de forma aleatoria e estratificada, unindo os arremessos de 2 e de 3 para gerar uma melhora nos resultados de operação, com intuito de manter as matricas que perderam caíram mais como a Prec. e o Recall e o F1 mais alinhados com os resultados do desenvolvimento.

8c Estrategias Reativa e Preditiva

O monitoramnto do modelo pode ser feito com a variável resposta, como no caso acima, onde foi comparada a operação com os dados do desenvolvimento para gerar indicadores, e os indicadores servem para descrever como está a saúde do modelo. Ou no caso de não existir a variável resposta, deve ser utilizada uma forma de gerar a variável, seja com equipe de especialista na área em questão, ou utilizando outros algoritmos por exemplo. Mas nesse caso específico a variável resposta existe porque cada arremesso gera uma cesta ou erro. Num caso por exemplo de tratamento de uma doença, ou ainda não se saiba, pode ser coletada amostras tem realização de exames onde e apoio de especialistas, onde se possa gerar os resultados, e daqui extrair as metricas.

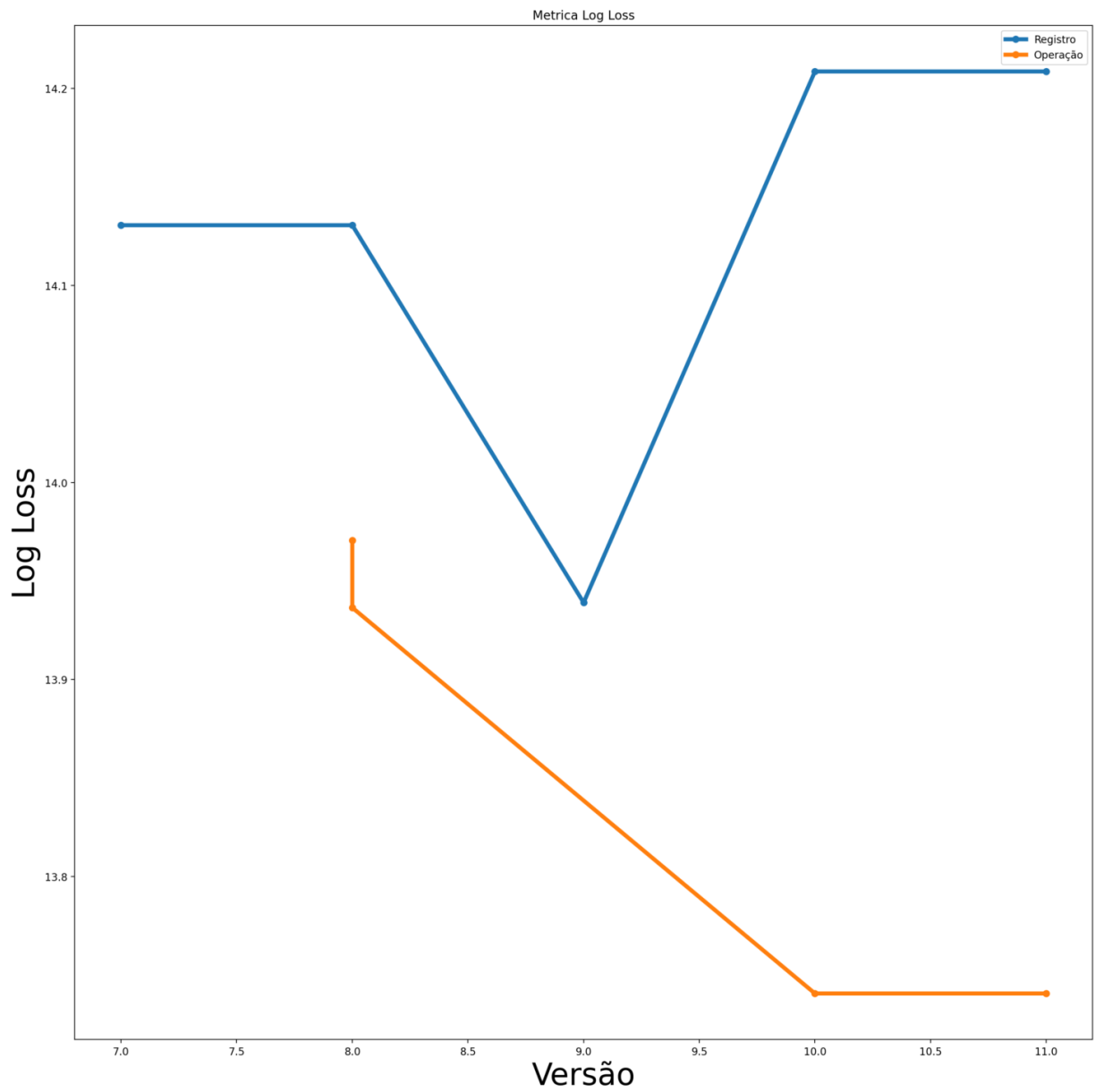
9 Streamlit

Para o acompanhamento da operação do modelo, está sendo utilizado visualizações graficas no streamlit. Os graficos então sendo construídos utilizando informações do MLFlow do experimento e também dos artefatos gerados no experimento. E o streamlit fazendo um papel de front-end para exibição.

A navegação está no slidebar no lado esquerdo da tela, uma selectbox com opções para selecionar escolhe a visualização, Inicial, Versionamento, Operação. No Versionamento, os gráficos que comparam dada uma versão de Staging, a operação e o registro, ou seja, as metricas do teste com algoritmo durante o registro para Staging, e o algoritmo durante a operação simulada em Staging. Caso houver duas simulações por exemplo, com a mesma versão (operação) há dois ou mais pontos para com as metricas. A opção visualização escolhe o gráfico, como LogLoss ou F1-Score.

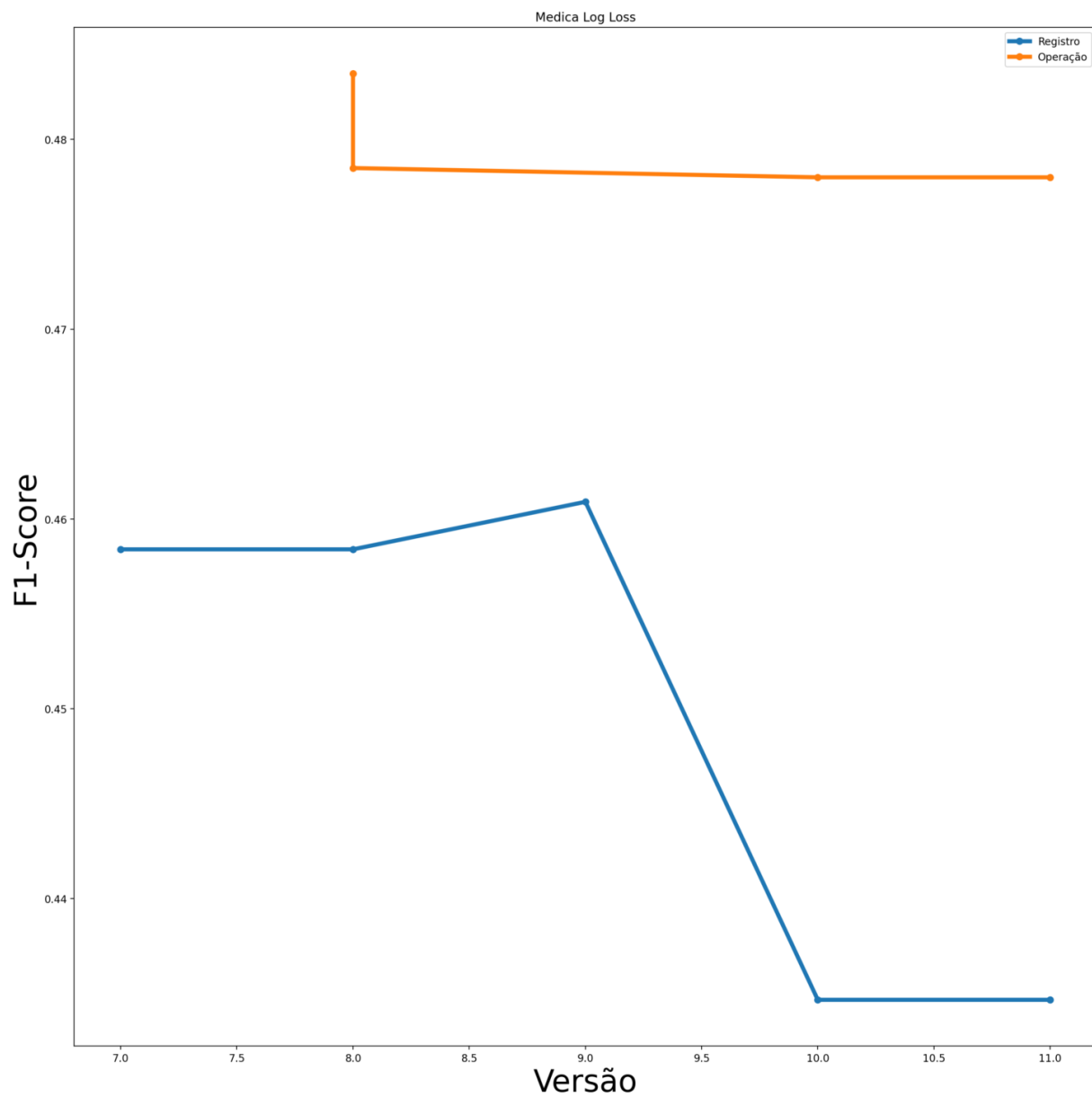
Versionamento

Um Gráfico mostra a versão do modelo como eixo X e a métrica como eixo Y, Uma linha é o registro e outra a simulação da operação

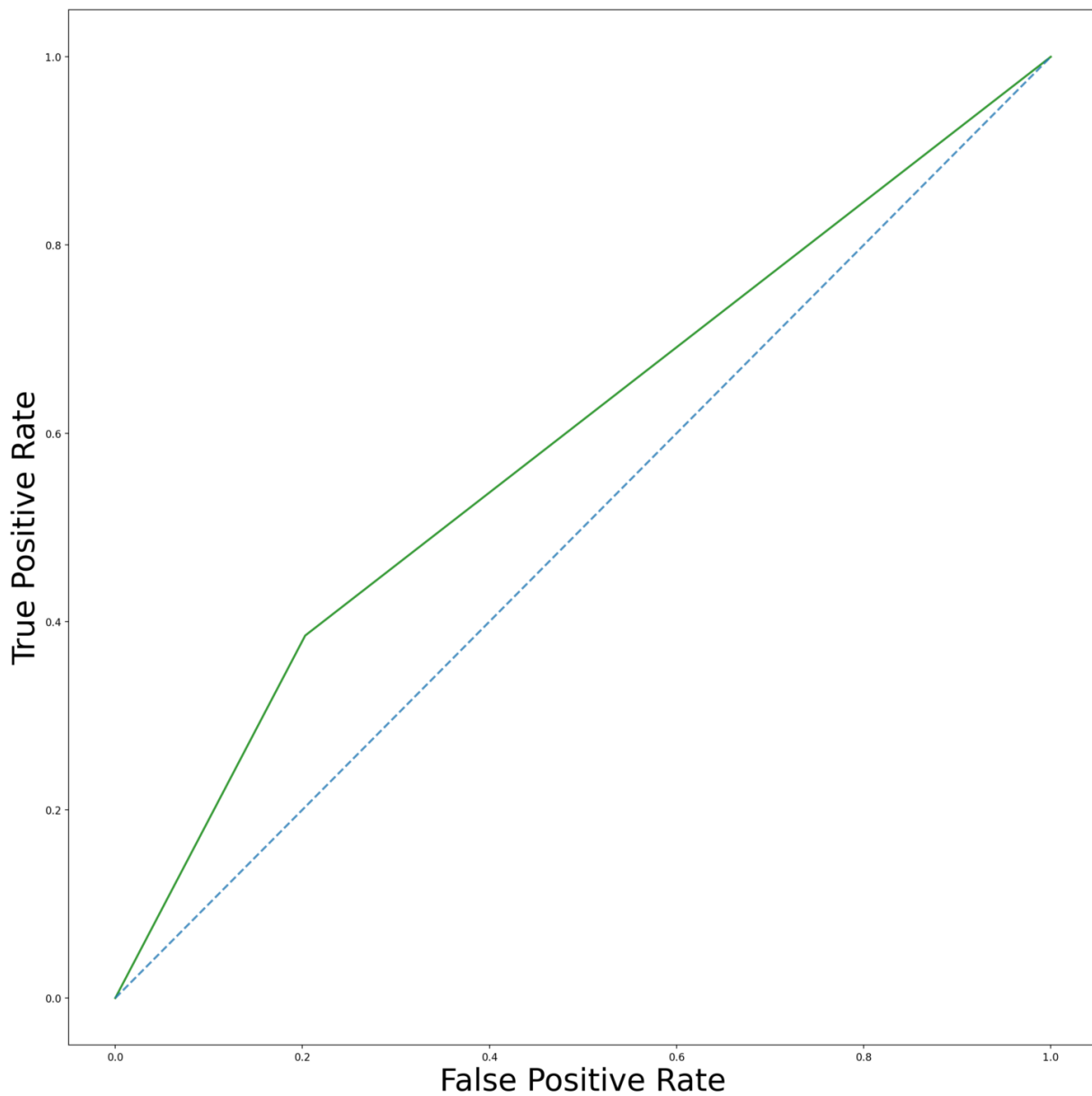


E que mostra que os resultados de operação foram melhores que os de treino e test em todas as versões

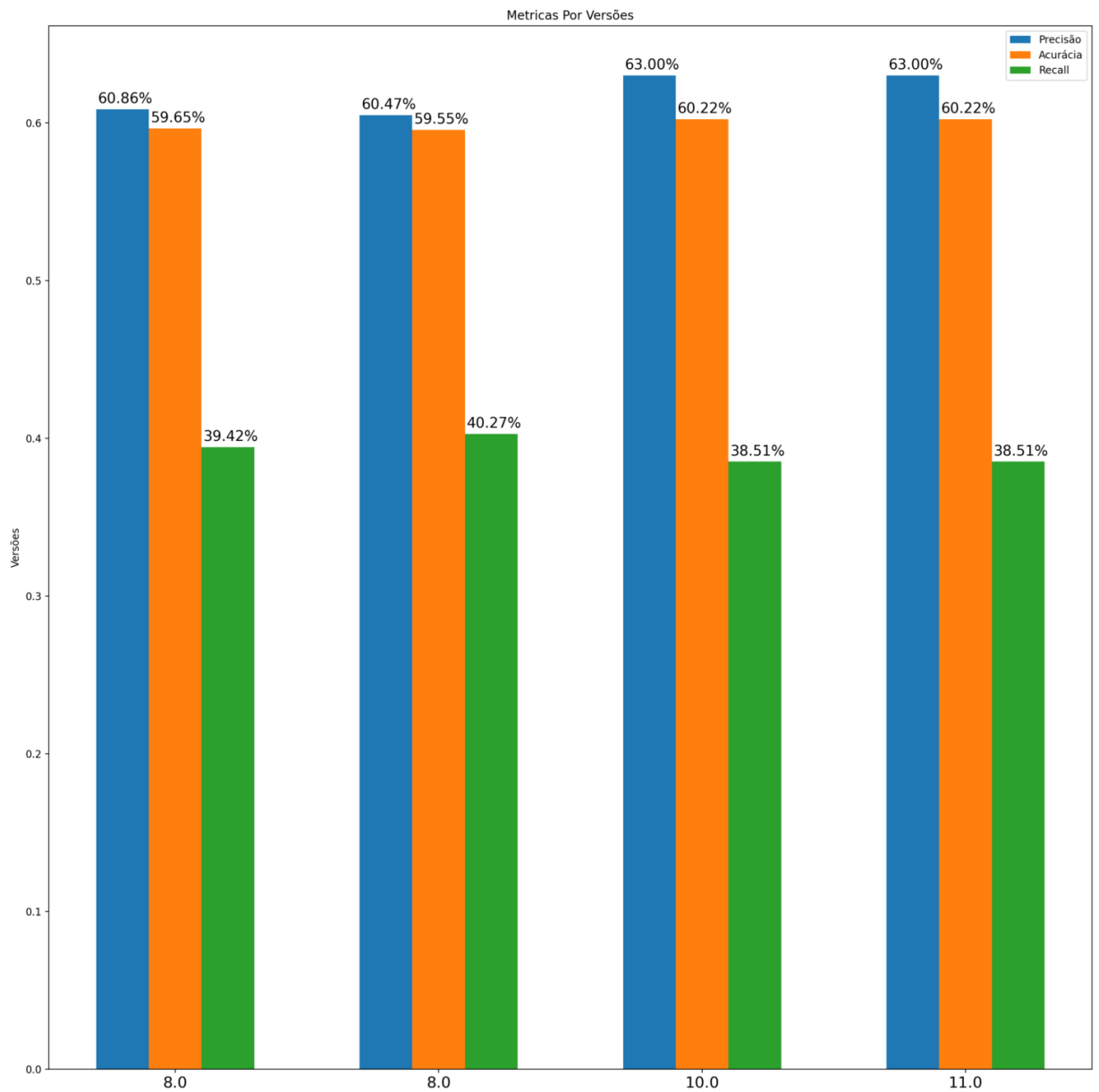
Da mesma forma o gráfico de F1-Score



In []: Outros graficos só de operação como curva_roc



E um com a Precisão, Acurácia e Recall nas validações da operação por versão do modelo registrada no MLFlow. Na versão 8, houveram duas operações de revalidação, por isso ela aparece duas vezes.



No fim os resultados das mettricas não foram bons, possivelmente, pode ser fruto da seleção de mettricas inicial que foi realizada. Pode ser que que uma boa opção seria avaliar as melhores mettricas antes da seleção. Porém o objetivo principal do trabalho é demonstrar o processo de Auto ML com a utilização das ferramentas Pycaret, MLFlow, Streamlit, Sklearn, principalmente. Principalmente por que também há, Jupyter notebook, Ambiente Anaconda, entre outros.

Este processo é ciclico, então, haveria novos preprocessamentos, com novo treino e teste, registro e versão e monitoramento ou simulação da operação. Os gráficos teriam a versão 12 incluída, e assim por diante.

In []:

