Relatório Final das Etapas
CRISP-DM

Shill Bidding Analysis

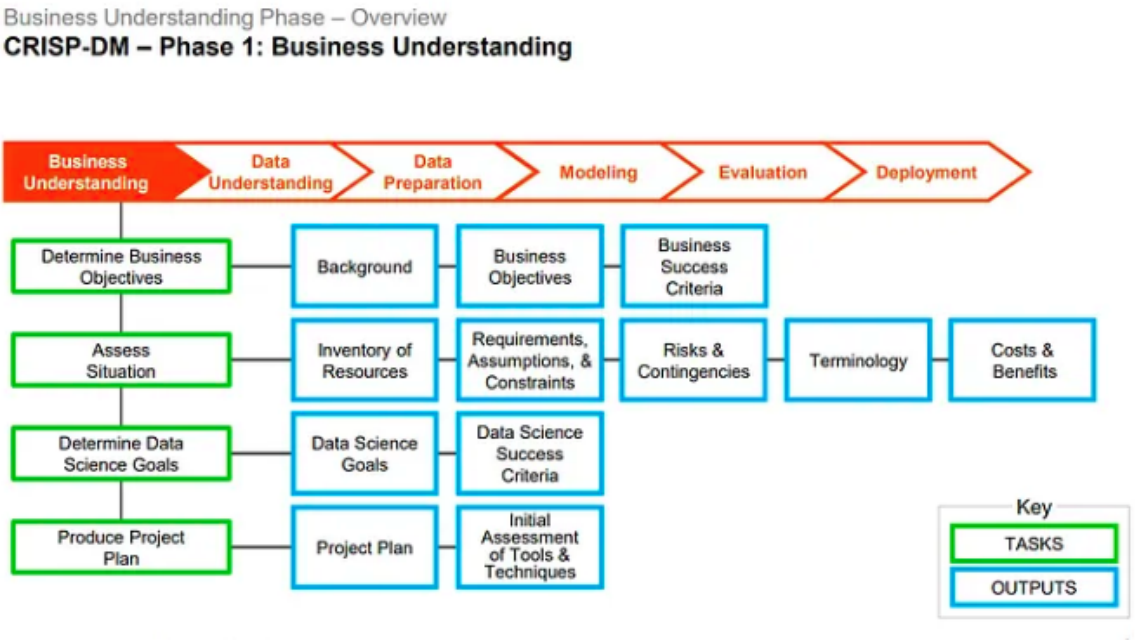
Alunos:
André Luiz Pereira da Silva (alps2@cin.ufpe.br)
Victoria Célia César Ferreira (vccf@cin.ufpe.br)

Repositório:
https://github.com/alpsilva/ShillBiddingAnalysis

Dataset:
https://archive.ics.uci.edu/ml/datasets/Shill+Bidding+Dataset

Sumário:

1. Business understanding



1.1 Business Objectives

O negócio lida com leilões online. Existe um problema sério na plataforma, que é o de "Arrematantes Cúmplices" (Shill Bidders, em inglês). Basicamente, são contas associadas ao leiloeiro que aplicam lances nos itens apenas para inflar seu preço, sem intenção de comprá-lo. A prática é tida como desleal e ilegal na plataforma, pois interfere na venda de itens por preços justos, podendo lesar os usuários comuns.
O objetivo da plataforma é identificar essa prática para poder banir as contas com atividade suspeita, sem correr o risco de acabar banindo usuários legítimos.
Para isso, um modelo precisaria ter uma alta precisão de verdadeiros positivos, e uma baixa (preferencialmente zero) ocorrência de falsos positivos.

1.2 Situation

Para implementação de uma possível solução, seria necessário um conjunto de dados contendo informações sobre lances. Dessa forma, analisando os lances, é possível identificar características que identificam atividades suspeitas.
A plataforma forneceu conjunto de registros históricos contendo características diversas e corretamente rotulados como "atividade suspeita" e "atividade normal". Os atributos deste dataset são detalhados abaixo:
- Record ID: Unique identifier of a record in the dataset.
- Auction ID: Unique identifier of an auction.
- Bidder ID: Unique identifier of a bidder.

- Bidder Tendency: A shill bidder participates exclusively in auctions of few sellers rather than a diversified lot. This is a collusive act involving the fraudulent seller and an accomplice.
- Bidding Ratio: A shill bidder participates more frequently to raise the auction price and attract higher bids from legitimate participants.
- Successive Outbidding: A shill bidder successively outbids himself even though he is the current winner to increase the price gradually with small consecutive increments.
- Last Bidding: A shill bidder becomes inactive at the last stage of the auction (more than 90\% of the auction duration) to avoid winning the auction.
- Auction Bids: Auctions with SB activities tend to have a much higher number of bids than the average of bids in concurrent auctions.
- Auction Starting Price: a shill bidder usually offers a small starting price to attract legitimate bidders into the auction.
- Early Bidding: A shill bidder tends to bid pretty early in the auction (less than 25\% of the auction duration) to get the attention of auction users.
- Winning Ratio: A shill bidder competes in many auctions but hardly wins any auctions.
- Auction Duration: How long an auction lasted.
- Class: 0 for normal behaviour bidding; 1 for otherwise.

Nada específico a declarar para riscos e contingências. Os dados precisam ser corretamente armazenados de acordo com a LGPD. O modelo em si precisará de uma boa taxa de precisão para evitar identificações errôneas.
O custo da solução estará principalmente atrelado ao seu desenvolvimento e manutenção. Uma vez definido o melhor modelo com os melhores parâmetros para a tarefa, é simples a sua inclusão no workflow da plataforma. Mas estudos sequentes são necessários para garantir que essa classificação sempre reflita dados e tendências atualizadas e realistas. Uma pipeline deverá ser construída e entregue para a equipe de infraestrutura in-house da plataforma para a análise e treinamento de modelos futuros, com datasets atualizados.

1.3 Data Science Goals

- Preparação e transformação dos dados, com normalização de dados constantes e técnicas como one-hot-encoding para dados discretos.
- Criar uma pipeline de treinamento, para testar com diversos modelos e hiper parâmetros.
- Otimização de hiperparâmetros.
- Critério de sucesso: Acima de 70% de precisão em verdadeiros positivos e minimizar os falsos negativos e falsos positivos o máximo possível.

1.4 Project Plan

- Análise inicial dos dados, para definir características que podem ser exploradas ou problemas a serem resolvidos.
- Pipeline de treinamento. Aplicará todas as etapas de preparação e transformação dos dados, incluindo a separação entre conjuntos de treinamento, validação e teste, e deve funcionar com qualquer modelo.

Dessa forma, será simples replicar o pipeline para cada modelo treinado.

- Treinamento e otimização dos seguintes modelos: K-NN, LVQ, Árvore de decisão, SVM, Random Forest, Rede Neural MLP, Comitê de Redes Neurais Artificiais e um comitê heterogêneo. A otimização será feita por meio de randomized search, para evitar tempos de processamento muito longos na fase inicial. Para Cross validation, será usado um fold de tamanho 10.
- Para cada modelo treinado, apresentar a acurácia média (e desvio padrão), matriz de confusão e os gráficos de caixa (boxplot).
- Para cada modelo, analisar os resultados com relatório de classificação e apresentar testes de significância estatística.
- Treinar, separadamente dos passos acima, uma árvore de decisão no intuito de identificar quais atributos ela considera mais relevantes (primeiros utilizados na construção da árvore). Sugerir decisões baseadas nesse conhecimento.

2. Data Understanding

2.1 Perfil dos dados

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6321 entries, 0 to 6320
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Record_ID               6321 non-null   object
 1   Auction_ID              6321 non-null   object
 2   Bidder_ID               6321 non-null   object
 3   Bidder_Tendency         6321 non-null   float64
 4   Bidding_Ratio           6321 non-null   float64
 5   Successive_Outbidding   6321 non-null   object
 6   Last_Bidding            6321 non-null   float64
 7   Auction_Bids            6321 non-null   float64
 8   Starting_Price_Average  6321 non-null   float64
 9   Early_Bidding           6321 non-null   float64
 10  Winning_Ratio           6321 non-null   float64
 11  Auction_Duration        6321 non-null   int64
 12  Class                   6321 non-null   int64
dtypes: float64(7), int64(2), object(4)
memory usage: 642.1+ KB
```

Descrição dos atributos presentes no dataset ShillBidding.

- Colunas: 13.
- Registros: 6321.
- Existem 3 atributos que poderiam ser considerados categóricos (Record_ID, Auction_ID, Bidder_ID), mas como são apenas dados de identificação do registro, não serão utilizados para a classificação.
- O Successive_Outbidding, embora não esteja declarado explicitamente como um dado categórico e não haja legenda para os valores que pode adotar, na prática age como um atributo categórico por só adotar 3 possíveis valores.
- Todos os demais atributos são numéricos.

2.2 Existência de valores nulos

Como podemos ver na consulta abaixo, não há dados nulos.

```
df.isnull().sum()
```

```
Record_ID                0
Auction_ID               0
Bidder_ID                0
Bidder_Tendency          0
Bidding_Ratio            0
Successive_Outbidding    0
Last_Bidding             0
Auction_Bids             0
Starting_Price_Average   0
Early_Bidding            0
Winning_Ratio            0
Auction_Duration         0
Class                    0
dtype: int64
```

2.3 Checando registros duplicados

Como podemos ver na consulta abaixo, não há dados duplicados.

```
if len(df[df.duplicated()]) > 0:
    print("No. of duplicated entries: ", len(df[df.duplicated()]))
    print(df[df.duplicated(keep=False)].sort_values(by=list(df.columns)).head())
else:
    print("No duplicated entries found")
```

[6]

```
No duplicated entries found
```

2.4 Distribuição de dados numéricos

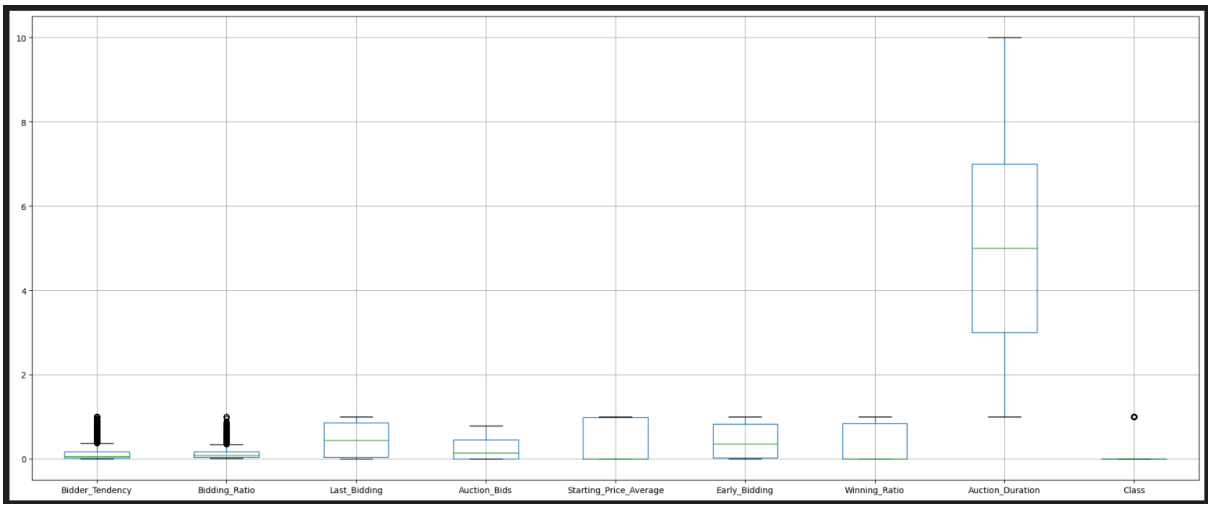
```
df.describe()
```

	Bidder_Tendency	Bidding_Ratio	Last_Bidding	Auction_Bids	Starting_Price_Average	Early_Bidding	Winning_Ratio	Auction_Duration	Class
count	6321.000000	6321.000000	6321.000000	6321.000000	6321.000000	6321.000000	6321.000000	6321.000000	6321.000000
mean	0.142541	0.127670	0.463119	0.231606	0.472821	0.430683	0.367731	4.615093	0.106787
std	0.197084	0.131530	0.380097	0.255252	0.489912	0.380785	0.436573	2.466629	0.308867
min	0.000000	0.011765	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000
25%	0.027027	0.043478	0.047928	0.000000	0.000000	0.026620	0.000000	3.000000	0.000000
50%	0.062500	0.083333	0.440937	0.142857	0.000000	0.360104	0.000000	5.000000	0.000000
75%	0.166667	0.166667	0.860363	0.454545	0.993593	0.826761	0.851852	7.000000	0.000000
max	1.000000	1.000000	0.999900	0.788235	0.999935	0.999900	1.000000	10.000000	1.000000

Descrição dos atributos numéricos presentes no dataset ShillBidding.

- Podemos ver que não há nenhum valor negativo nos dados numéricos, o que é algo positivo pois nada na descrição dos dados dá a entender que existiriam. Caso houvesse algum registro com valores negativos, seria um bom indicador de sujeira nos dados.

- Em alguns dados, principalmente Bidder_Tendency, Bidding_Ration e Auction_Bids, o valor máximo está muito distante do 75% percentil, indicando a possível existência de outliers. Essa possibilidade será averiguada nos gráficos boxplot.

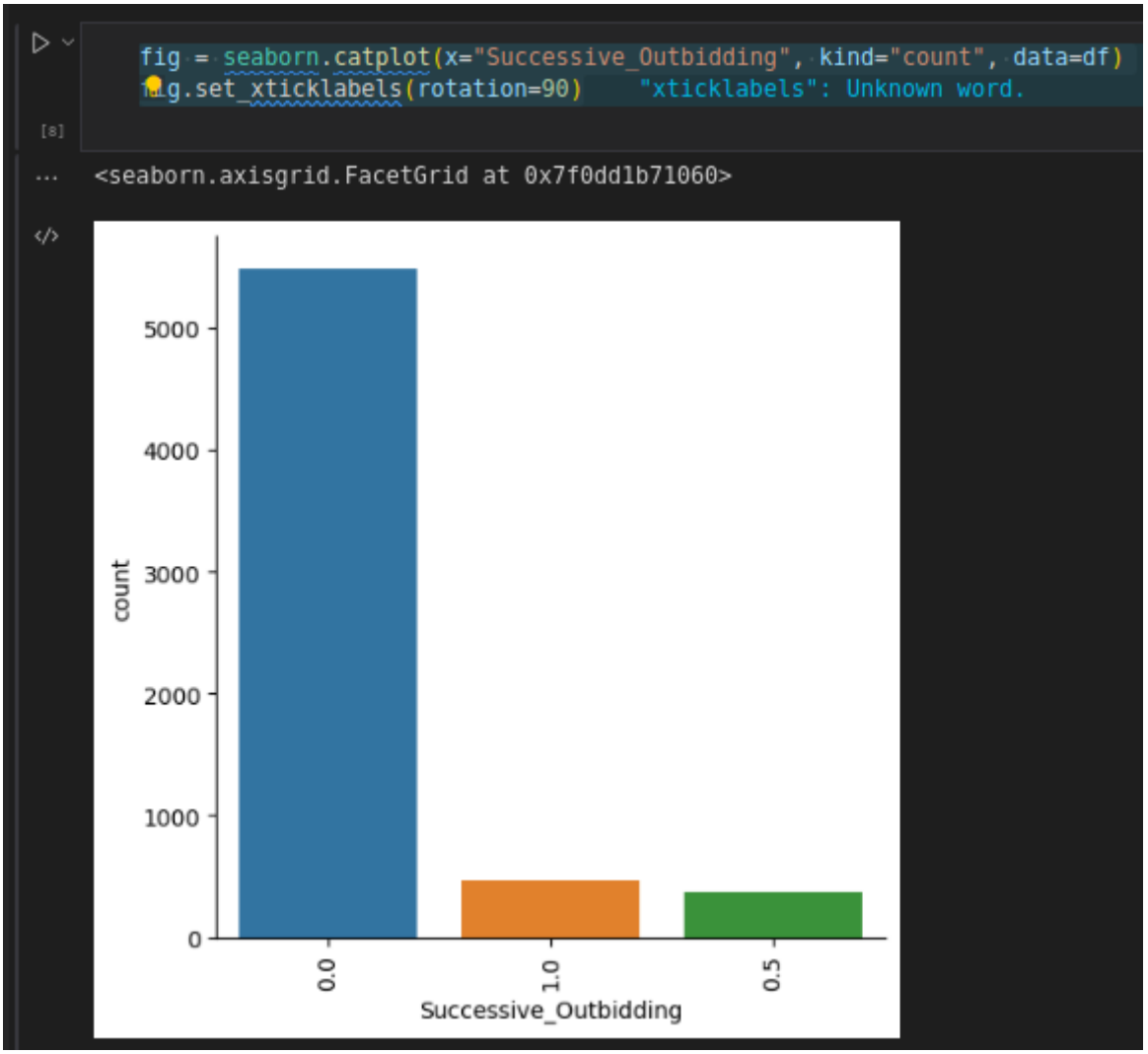


Gráficos boxplot de todos os atributos numéricos presentes no dataset ShillBidding.

- De fato, Bidder_Tendency e Bidding_Ratio possuem outliers. Porém, os 2 são atributos muito relevantes pois o Bidder Tendency indica a participação em auctions com poucos compradores, e o Bidding Ratio indica participação frequente pra aumentar o preço, ambos grandes indicativos de fraude. Como temos poucos registros positivos de comportamento fraudulento, podemos entender que esses indicadores aparentam ser outliers pois definem muito bem comportamento fraudulento, e naturalmente ocorrem menos vezes nesses valores altos. Concluindo, não devemos remover esses valores, pois na prática não são outliers. Mais detalhes sobre a distribuição das classes de registros mais adiante.
- Os outliers em Class serão desconsiderados pois essa é a label do problema, e consiste de 0's e 1's. Porém, é um bom indicativo de que talvez o dataset esteja desbalanceado. Essa possibilidade será averiguada logo a seguir.

2.5 Distribuição de dados categóricos

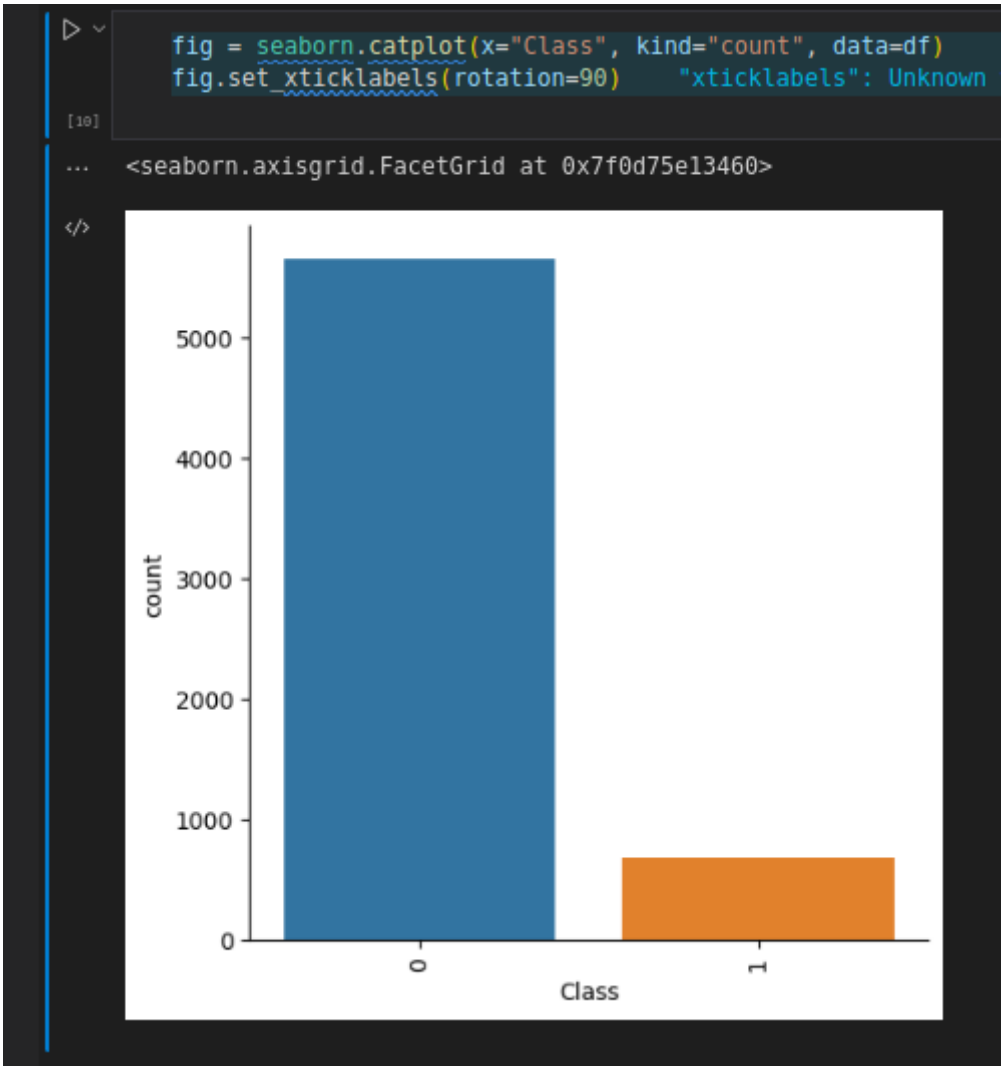
Como só temos 1 atributo categórico, analisamos apenas ele:



Distribuição do atributo “Successive_Outbidding“.

Levando em consideração o significado de Successive_Outbidding, que é configurado por uma pessoa continuar dando lances maiores mesmo quando está vencendo um leilão, podemos entender que valores maiores que zero estão fortemente atrelados ao comportamento fraudulento. A frequência maior do valor 0 provavelmente está ligada ao fato do dataset ter mais registros de comportamento normal do que de comportamento fraudulento (mais informações adiante).

2.6 Distribuição das classes



Distribuição dos rótulos presentes no dataset ShillBidding.

Como podemos ver na distribuição, temos bem mais registros cuja classe é 0, do que registros cuja classe é 1.

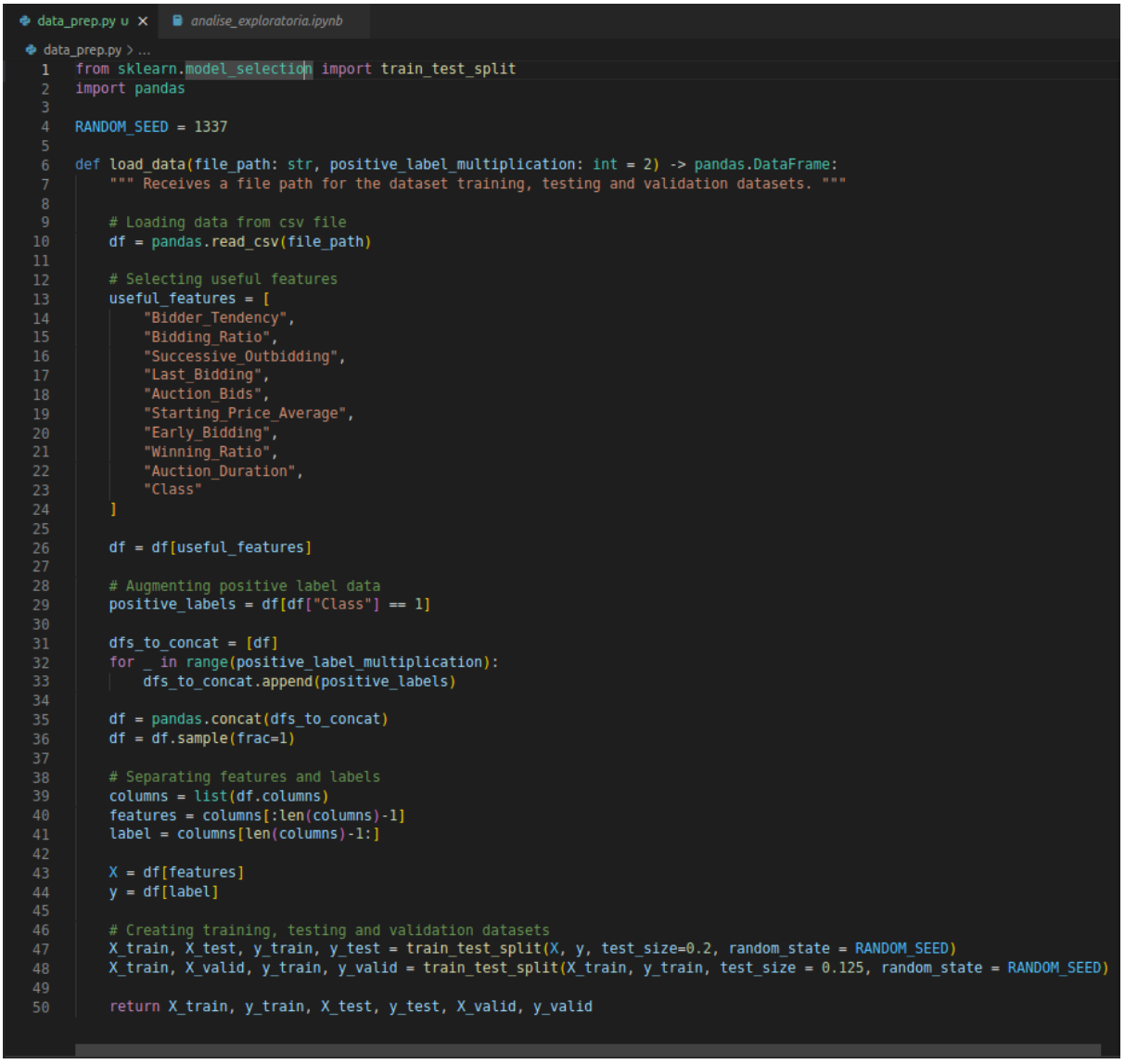
Isso é um problema sério durante o treinamento de modelos, pois pode enviesá-lo a assumir classes negativas com mais frequência. Como já temos poucos registros para trabalhar, cortar registros da classe mais representada pode ser muito prejudicial.

Assim, alguma técnica de Data Augmentation precisará ser utilizada para incrementar o número de exemplos da classe menos representada.

Uma possível solução seria simplesmente replicar o número de registros até que as classes estivessem balanceadas, mas com isso corremos um sério risco de overfitting para esses dados registrados.

3. Data Preparation

Para o preparo dos dados, foi criada uma função "load_data", como pode ser visto a seguir:



```python
from sklearn.model_selection import train_test_split
import pandas

RANDOM_SEED = 1337

def load_data(file_path: str, positive_label_multiplication: int = 2) -> pandas.DataFrame:
    """ Receives a file path for the dataset training, testing and validation datasets. """

    # Loading data from csv file
    df = pandas.read_csv(file_path)

    # Selecting useful features
    useful_features = [
        "Bidder_Tendency",
        "Bidding_Ratio",
        "Successive_Outbidding",
        "Last_Bidding",
        "Auction_Bids",
        "Starting_Price_Average",
        "Early_Bidding",
        "Winning_Ratio",
        "Auction_Duration",
        "Class"
    ]

    df = df[useful_features]

    # Augmenting positive label data
    positive_labels = df[df["Class"] == 1]

    dfs_to_concat = [df]
    for _ in range(positive_label_multiplication):
        dfs_to_concat.append(positive_labels)

    df = pandas.concat(dfs_to_concat)
    df = df.sample(frac=1)

    # Separating features and labels
    columns = list(df.columns)
    features = columns[:len(columns)-1]
    label = columns[len(columns)-1:]

    X = df[features]
    y = df[label]

    # Creating training, testing and validation datasets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = RANDOM_SEED)
    X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size = 0.125, random_state = RANDOM_SEED)

    return X_train, y_train, X_test, y_test, X_valid, y_valid
```

Durante o EDA, checamos o dataset para verificar casos específicos. Embora o dataset já esteja em ótimo estado (sem valores nulos ou duplicados, valores numéricos consistentes e sem outliers), ainda é preciso tomar algumas medidas para que os dados possam ser consumidos por um classificador. Cada passo da função será explicado neste relatório.

3.1 Feature selection

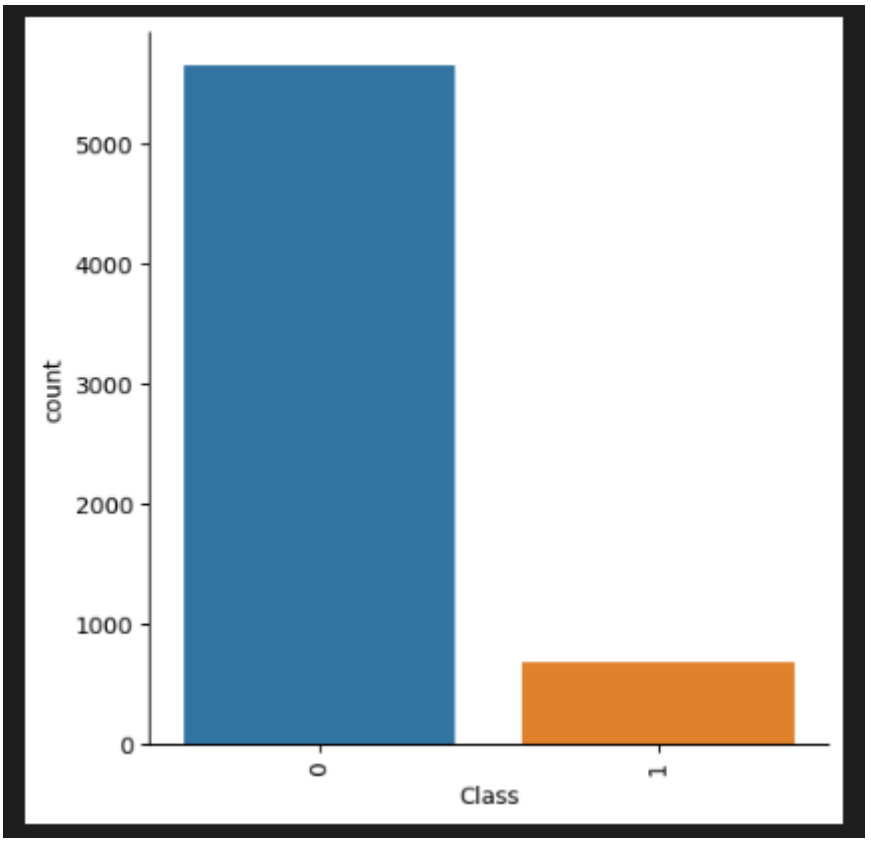
A primeira etapa da função é uma seleção de features úteis. Foi determinado que os atributos Record_ID, Auction_ID e Bidder_ID representam identificadores dos registros e não seriam relevantes para a classificação, dessa forma mantendo apenas os atributos:

- Bidder_Tendency
- Bidding_Ratio
- Successive_Outbidding
- Last_Bidding
- Auction_Bids
- Starting_Price_Average
- Early_Bidding
- Winning_Ratio
- Auction_Duration
- Class

Embora o Bidder_ID pudesse ser uma informação útil (já que identifica uma conta, e assumindo que uma conta que já foi marcada com comportamento fraudulento anteriormente, provavelmente vai infringir as regras da plataforma novamente), ainda assim foi cortado, pois criar uma conta nova na plataforma é simples. Dessa forma, indivíduos desejando burlar as regras sempre teriam Bidder_ID's novos quando necessário.

3.2 Data Augmentation

Como visto no relatório de análise exploratória, o dataset está desbalanceado, com muito mais labels negativas que positivas.



Distribuição dos rótulos no dataset ShillBidding

Isso pode ser muito ruim para o treinamento do modelo, que pode acabar enviesado a classificar sempre de forma negativa, já que, estatisticamente falando, esse rótulo ocorre mais vezes.

Para sanar isso, incluímos a possibilidade de multiplicar os registros de labels positivas presentes no dataset, por meio do atributo "positive_label_multiplication". Caso o valor passado seja 0, não será aplicado data augmentation no dataset. Por padrão, multiplicamos esses rótulos 3 vezes.
Com isso, passamos um sério risco de gerarmos modelos sofrendo de overfitting, já que o treinamento estaria muito contido no universo inicial de dados com classificação positiva. Contudo, como vimos no relatório, a maior parte dos atributos numéricos possui uma discrepância de valor muito grande entre rótulos positivos e negativos, nos levando a crer que vale a pena o risco.

3.3 Training, testing and validation sets

Por último, separamos o dataset completo em 3 conjuntos menores, de treinamento, teste e validação (seguindo a escala "70% - 20% - 10%", respectivamente). Utilizamos uma semente aleatória fixa, para garantir reprodutibilidade dos resultados.

4 Modelling

4.1 Função de Otimização

Definimos duas funções de otimização: a grid search e a randomized. Ambas serão usadas para determinar a melhor combinação de parâmetros para cada modelo.
Uma função search é definida, que recebe uma instância de modelo a ser "variado", um dicionário de parâmetros e suas variações e uma String indicando se é grid ou randomized.
A gridsearch testa exaustivamente todas as configurações possíveis, o que garante o melhor resultado porém pode ser muito custosa em relação ao tempo.
A randomized não testa todas as configurações possíveis, não garante que seu resultado é o melhor, mas tem bons resultados, chegando perto do melhor. Pode ser vantajoso usar randomized porque ela é muito menos custosa em relação ao tempo. Se estabelece o número máximo de candidatos que serão testados. Empregamos o randomized porque alguns modelos como o random forest consumiram muito tempo.
Antes do randomized:

```
def search(model, model_parameters):
    """ """
    model_pipeline = Pipeline([
        ('clf', model)
    ])

    grid_search = GridSearchCV(
        model_pipeline,
        model_parameters,
        n_jobs=N_JOBS,
        cv=FOLDS,
        verbose=1
    )

    grid_search.fit(X_train, y_train.values.ravel())

    print(f"Best score after optimization: {grid_search.best_score_}")
    print("Best params:")
    for key, value in grid_search.best_params_.items():
        print(f"{key}: {value}")

    return grid_search
```

Depois com o randomized:

```python
def search(model, model_parameters, search_method: str = "grid", randomized_max_candidates: int = 5000):
    """ """
    model_pipeline = Pipeline([
        ('clf', model)
    ])

    if search_method == "grid":
        search = GridSearchCV(
            model_pipeline,
            model_parameters,
            n_jobs=N_JOBS,
            cv=FOLDS,
            verbose=1
        )
    else:
        search = RandomizedSearchCV(
            model_pipeline,
            model_parameters,
            n_jobs=N_JOBS,
            cv=FOLDS,
            verbose=1,
            n_iter=randomized_max_candidates,
            random_state=RANDOM_SEED
        )

    search.fit(X_train, y_train.values.ravel())

    print(f"Best score after optimization: {search.best_score_}")
    print("Best params:")
    for key, value in search.best_params_.items():
        print(f"{key}: {value}")

    return search
```

4.2 Variando K-NN

Parâmetros e ranges escolhidos para a variação:

1. n_neighbors[4,10]: número de vizinhos.
2. weights["uniform", "distance"]: função de peso usada para a previsão. No uniform todos os pontos tem o mesmo peso e no distance pontos mais próximos terão mais influência.
3. algorithm["auto", "ball_tree", "kd_tree", "brute"]: algoritmo para calcular os vizinhos mais próximos.
4. leaf_size[20,40]: parâmetro usado no BallTree ou KDTree. O default é 30, então decidimos um range intermediário.
5. metric ["cityblock","cosine","euclidean","haversine","nan_euclidean"]: métrica usada para calcular a distância.
6. P[1,3]: paramêtro usado na métrica de Minkowski. Se p = 1, vai usar a manhattan_distance e se p=2 a euclidean_distance. Qualquer outro valor vai usar a minkowski_distance.
   Antes de usar o metric:

```
Fitting 10 folds for each of 1920 candidates, totalling 19200 fits
Best score after optimization: 0.9988826815642458
Best params:
clf__algorithm: auto
clf__leaf_size: 20
clf__n_neighbors: 4
clf__p: 1
clf__weights: distance
```

Após usar o metric:

```
Best score after optimization: 0.9996275605214151
Best params:
clf__algorithm: auto
clf__leaf_size: 20
clf__metric: cityblock
clf__n_neighbors: 8
clf__p: 1
clf__weights: distance
```

4.3 Variando Decision Tree

Parâmetros e ranges escolhidos para a variação:

1. criterion ["gini", "entropy", "log_loss"]: Algoritmo para medição de qualidade de divisão de nós. Servem para determinar quais as melhores features para estarem mais perto do topo da árvore (ou seja, que apresentam maior ganho de informação).
2. splitter["best", "random"]: como os dados vão ser divididos a cada nó. Quando o splitter ocorre os dados são divididos em 2 ou mais subconjuntos e 1 nó para cada subconjunto é criado. A divisão dos dados procura melhor separar as diferentes classes.
3. max_features [0.2,0.4,0.6,0.8, None, "sqrt", "log2"]: máximo número de features/atributos considerados para o melhor splitter possível a cada nó da árvore. Usar muitas features resulta em overfitting e usar poucas em underfitting. Se não especificado o número máximo de features, o modelo irá treinar com todas as features.
4. max_depth[3, 50]: Máxima profundidade da árvore. Na construção de uma árvore de decisão o algoritmo recursivamente divide os dados em subconjuntos baseado nas features/atributos, até parar. Uma das formas de fazer o algoritmo parar é setando um valor máximo da profundidade da árvore, que é o max_depth. Se a árvore for muito profunda resulta em overfitting, se for pouco profunda resulta em underfitting.

```
Fitting 10 folds for each of 1974 candidates, totalling 19740 fits
Best score after optimization: 0.9990689013035382
Best params:
clf__criterion: gini
clf__max_depth: 31
clf__max_features: None
clf__splitter: best
```

Foram adicionados:

5. min_samples_split [2, 10]: o número mínimo de amostras necessárias para dividir um nó interno. Se o número de amostras é menor que o min_samples_split o nó não se dividirá mais e se tornará 1 folha. Pode ajudar a prevenir o overfitting pois força o algoritmo a considerar apenas features com um número suficiente de amostras para fazer 1 divisão.
6. min_samples_leaf [1,20] o número mínimo de amostras em uma folha. Se o número de amostras em uma folha é menor que o min_samples_leaf o nó se juntará com o seu nó vizinho, ou com nó pai se for o único filho. O processo de junção continua até todas as folhas terem no mínimo o número de amostras em min_samples_leaf. Um valor muito alto para min_samples_leaf pode levar a underfitting e um valor muito baixo pode levar a overfitting. O objetivo é assegurar que cada folha tem um número suficiente de amostras para fazer 1 previsão confiável.

```
Fitting 10 folds for each of 300048 candidates, totalling 3000480 fits
Best score after optimization: 0.999440993357236
Best params:
clf__criterion: gini
clf__max_depth: 24
clf__max_features: 0.8
clf__min_samples_leaf: 1
clf__min_samples_split: 4
clf__splitter: best
```

4.4 Variando SVM

Parâmetros e ranges escolhidos para a variação:

1. kernel["linear", "poly", "rbf", "sigmoid"]: O kernel é uma função matemática para transformar dos dados do input do seu espaço de features original em uma espaço de maior dimensão, para facilitar a separação de classes usando um classificador linear. A função de kernel recebe 2 vetores como input e retorna um valor escalar que mede a distância entre eles no espaço transformado. Essa medida de semelhança/distância é usada para encontrar o hiperplano que separa as classes diferentes no espaço transformado, com a maior distância possível entre os 2 pontos mais próximos de cada classe diferente.

Linear: o espaço original;
Polinomial: uma função polinomial sobre os dados de entradas;
Gaussiana: uma função gaussiana é usada para medir a similaridade entre os vetores de entrada;
Sigmóide: uma função sigmóide sobre os dados de entradas.

2. gamma["scale", "auto"]: o kernel coefficient (coeficiente do kernel). Controla a forma e largura da função de kernel. Define o range de influência de 1 único exemplo de treinamento na fronteira de decisão. Se o gamma é menor, a fronteira é mais simples e a influência de cada exemplo do treinamento é maior. Se gamma é maior, a fronteira de decisão é mais complexa e a influência de cada exemplo de treinamento é menor.
3. C[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]: o regularization parameter (parâmetro de regularização). O C controla o trade-off entre maximizar a margem e minimizar o erro de classificação. O C determina a penalidade por classificar de forma errada os exemplos de treinamento. Se C é pequeno, a penalidade é menor e a margem maior, se C é maior a penalidade é maior e a margem menor. Um C muito grande resulta em overfitting, um C muito pequeno resulta em underfitting. O valor de C é determinado com o valor de outros hiperparâmetros como o gamma, para encontrar a combinação que leve a melhor performance. O C foi entre 0.1 e 1.0 pois foram vistos exemplos de outros modelos sendo treinados e esses eram os valores vistos.
4. range (1,4): grau da função polinomial

```
Fitting 10 folds for each of 240 candidates, totalling 2400 fits
Best score after optimization: 0.9936681814391729
Best params:
clf__C: 1.0
clf__degree: 1
clf__gamma: auto
clf__kernel: rbf
```

Foram adicionados:

5. class_weight[0: 0.2, 1: 0.8] é usado quando há mais exemplos de 1 classe que de outra. Quando isso acontece o modelo pode se enviesar para a classe majoritária e ter uma performance pior na classe minoritária. Quando fizemos a análise exploratória de dados notamos que esse é o caso para o nosso problema. O class_weight atribui 1 peso para cada classe baseado na frequência que ela aparece nos dados. O SVM usa esse peso para ajustar a penalidade dada a cada classe no treinamento. Quanto maior o peso, maior a penalidade.
6. decision_function_shape["ovr","ovo"]: determina a forma da função de decisão. Em problemas de classificação binária, a função de decisão

retorna um score (pontuação) para cada amostra indicando qual classe é a classe prevista. Se o score é >=0 é da classe positiva, se o score é <0 é da classe negativa. Em problemas com múltiplas classes, a função de decisão retorna um score para cada classe, e a classe prevista é a com o maior score. decision_function_shape controla o cálculo do score para problemas com múltiplas classes:
"ovr" (one-vs-rest) existe um classificador SVM binário para cada classe, e as amostras de cada classe são consideradas da classe positiva e as outras amostras da classe negativa. O score para 1 amostra é o maior score dentre todos classificadores binários.
"ovo" (one-vs-one) existe um classificador SVM binário para cada par de classes, 1 das classe é considerada positiva e a outra negativa. O score para 1 amostra é a soma dos scores de todos os classificadores binários. A classe prevista é a com o maior score acumulado.

```
Fitting 10 folds for each of 960 candidates, totalling 9600 fits
Best score after optimization: 0.9934819616998805
Best params:
clf__C: 1.0
clf__class_weight: {0: 2, 1: 8}
clf__decision_function_shape: ovr
clf__degree: 1
clf__gamma: auto
clf__kernel: rbf
```

4.5 Variando Random Forest

Parâmetros e ranges escolhidos para a variação:

1. n_estimators [50:150]: Número de árvores de decisão presentes na floresta.
2. criterion ["gini", "entropy", "log_loss"]: Algoritmo para medição de qualidade de divisão de nós. Servem para determinar quais as melhores features para estarem mais perto do topo do árvore (ou seja, que apresentam maior ganho de informação).
3. max_features ["sqrt", "log2", None]: Número de atributos a serem considerados na hora de dividir.
4. max_depth [3:50]: Profundida máxima da árvore. O próprio SKLearn recomenda 3 como mínimo. O máximo depende muito das situações, mas via de regra, uma árvore mais profunda representa um classificador com risco de overfitting. Manter uma profundidade balanceada pode garantir um modelo mais genérico que consegue lidar com registros novos (diferentes dos que foram usado para o treino) sem muitos problemas.
5. min_samples_split [2..20]: Número mínimo de amostras necessárias para dividir um nó interno (que não são folhas).

6. min_samples_leaf [1..20]: Número mínimo de amostras necessárias para considerarmos um nó como folha. Aumentar esse número produz árvores mais generalistas, com profundidades menores.

Da entrega I para a II, foram adicionados os parâmetros min_samples_split e min_samples_leaf. Originalmente essa busca foi feita usando gridsearch, o que resultou num tempo de execução de mais de 10 horas! Mas com o random search, limitamos o número de candidatos para 5000, economizando muito tempo e chegando num resultado semelhante.

Resultado original:

```
Fitting 10 folds for each of 49350 candidates, totalling 493500 fits
Best score after optimization: 0.9986964618249534
Best params:
clf__criterion: entropy
clf__max_depth: 8
clf__max_features: 0.8
clf__n_estimators: 106
```

Resultado usando randomized e mais parâmetros:

```
Best score after optimization: 0.9986964618249535
Best params:
clf__n_estimators: 178
clf__min_samples_split: 4
clf__min_samples_leaf: 1
clf__max_features: None
clf__max_depth: 44
clf__criterion: gini
```

4.6 Variando Rede Neural MLP

Parâmetros e ranges escolhidos para a variação:

1. hidden_layer_sizes [(100,), (50, 50,), (33, 33, 34,), (25, 25, 25, 25,)]: Quantidade de camadas internas do perceptron (e o número de neurons em cada uma)
2. Activation ["identity", "logistic", "tanh", "relu"]: Função de ativação para os neurons.
3. Solver ["lbfgs", "sgd"]: De acordo com o sklearn, o solver "adam" funciona melhor com datasets maiores (na casa das dezenas de milhares). Como nosso dataset tem uma escala menor, seu uso não é recomendado, pois as alternativas convergem mais rapidamente e performam melhor.
4. Alpha: [0.0001, 0.0002, 0.0003, ..., 0.0009]: Termo de Regularização L2.
5. Learning rate ["constant", "invscaling", "adaptative"]: Taxa de aprendizado que define as atualizações de peso.
6. max_iter [50..1000]: Número de iterações a ocorrem internamente durante o treinamento. Maiores números significam mais tempo de processamento, e possivelmente mais precisão, já que o modelo terá mais tentativas para convergir numa solução.

Da entrega I para a II, foram adicionados os parâmetros Alpha, learning rate e max_iter.
O learning rate, em especial, foi adcionado por ser considerado um dos parâmetros mais importantes de uma MLP, pois define a taxa de aprendizado, ou seja, o quanto o modelo é impactado em cada atualização recorrente dos pesos internos. Um valor muito baixo pode resultar em treinamentos longos e pouco conclusivos (falhas de convergência), enquanto um valor muito alto pode resultar em convergência demasiadamente alta, em um conjunto de pesos que não é ótimo.
O learning rate pode assumir os seguintes valores:

- Constant: Valor constante durante todo o processo (0.001, por padrão).
- Invscaling: Gradualmente diminui o learning rate a medida que o aprendizado acontece (a cada iteração).
- Adaptative: Mantém o learning rate constante até que 2 iterações seguidas falhem em aumentar o score de validação interno. Nesse caso, diminui o learning rate por 5.

Essas adições fizeram o tempo de busca aumentar bastante, então optamos por utilizar o randomized search aqui.

Resultado original:

```
Best score after optimization: 0.9983240223463687
Best params:
clf__activation: logistic
clf__hidden_layer_sizes: (100,)
clf__solver: lbfgs
/home/alps2/.local/lib/python3.10/site-packages/sk
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Resultado usando randomized e mais parâmetros:

```
Best score after optimization: 0.998510242085661
Best params:
clf__solver: lbfgs
clf__max_iter: 200
clf__learning_rate: invscaling
clf__hidden_layer_sizes: (100,)
clf__alpha: 0.0006
clf__activation: relu
```

4.7 Variando Comitê de redes neurais Artificiais

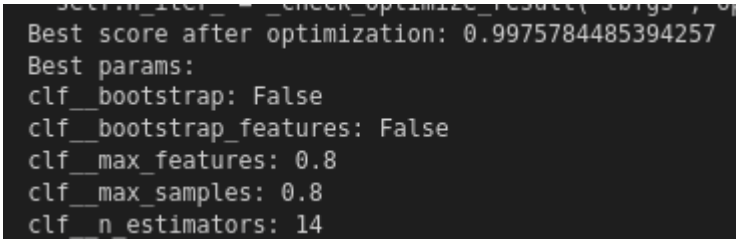
Como o algoritmo Bagging é um ensemble e pode receber vários classificadores, decidimos instanciar o MLP com o resultado de sua otimização, para garantir maior qualidade do comitê.
Parâmetros e ranges escolhidos para a variação:

1. n_estimators [5:15]: Número de classificadores presentes no comitê.
2. max_samples [0.2:1]: Número de registros a serem extraídos de X na hora de treinar cada classificador do comitê.

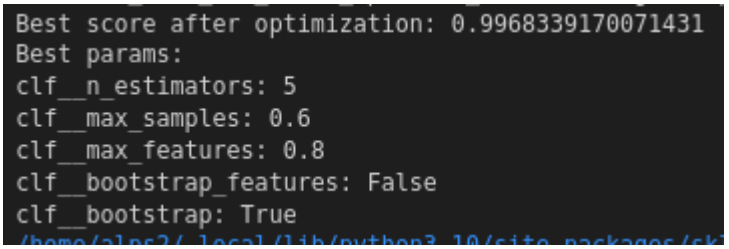
3. max_features [0.2:1]: Número de atributos a serem considerados na hora de treinar cada classificador do comitê.
4. bootstrap [True, False].
5. bootstrap_features [True, False].

Da entrega I para a II, foram adicionados os parâmetros bootstrap e bootstrap_features, que controlam amostras dos conjuntos de treinos e de suas features, para treinamento das instâncias dos modelos que compõe o comitê.
O bootstrap_features, em especial, é interessante pois teoricamente faria com que os diferentes modelos foquem em features diferentes, possibilitando que as diferentes instâncias tenham "especialidades" levemente diferentes entre si.
Também optamos por utilizar o randomized search aqui, para poupar tempo.

Resultado original:



Resultado usando randomized e mais parâmetros:



4.8 Variando Comitê Heterogêneo

Como o algoritmo VotingClassifier é um ensemble e pode receber vários classificadores diferentes, decidimos instanciá-los já com os melhores parâmetros provenientes de asuas otimizações. Os classificadores escolhidos para o VotingClassifier foram:

- MLP
- Random Forest
- SVM
- KNN

Parâmetros e ranges escolhidos para a variação:

1. voting ["hard", "soft"]: Regra para definir vencedor da votação (maioria simples com hard, probabilidade de classes com soft).
2. weights [None, [2.5, 2, 1.5, 1]]: pesos atribuídos para cada classificador. No caso de None, o peso é uniforme. Caso contrário, os pesos passados

são atribuídos. Declaramos os classificadores em ordem do mais preciso para o menos preciso, garantindo que os melhores classificadores influenciam mais na decisão.

```
Best score after optimization: 0.9988826815642458
Best params:
clf__voting: hard
clf__weights: None
```

5. Evaluation

5.1 Acurácia média, matriz de confusão e relatório de classificação

5.1.1 Matriz de confusão

A matriz de confusão é uma tabela que resume a performance de um modelo de classificação comparando as classes previstas com as classes reais de um dataset. Mostra a acurácia e padrões de classificação errados.
Uma matriz de confusão é estruturada da seguinte maneira:

1. Verdadeiro Positivo: é da classe positiva e foi previsto como da classe positiva;
2. Verdadeiro Negativo: é da classe negativa e foi previsto como da classe negativa;
3. Falso positivo: é da classe negativa e foi previsto como da classe positiva;
4. Falso negativo: é da classe positiva e foi previsto como da classe negativa.

Numa matriz de decisão queremos aumentar o número de verdadeiros positivos e verdadeiros negativos (classificações corretas) e diminuir o número de falsos positivos e falsos negativos (classificações erradas). As linhas são as classes reais e as colunas as classes previstas.

Podemos notar que nossos modelos têm uma alta porcentagem de verdadeiros (as classes reais e previstas são as mesmas) e uma baixa (em alguns casos até nula) porcentagem de falsos (as classes reais e previstas são diferentes) .

5.1.2 Relatório de classificação

O classification_report é uma função para avaliar a performance de um modelo de classificação
As métricas para a classificação (representadas pelas colunas) são:

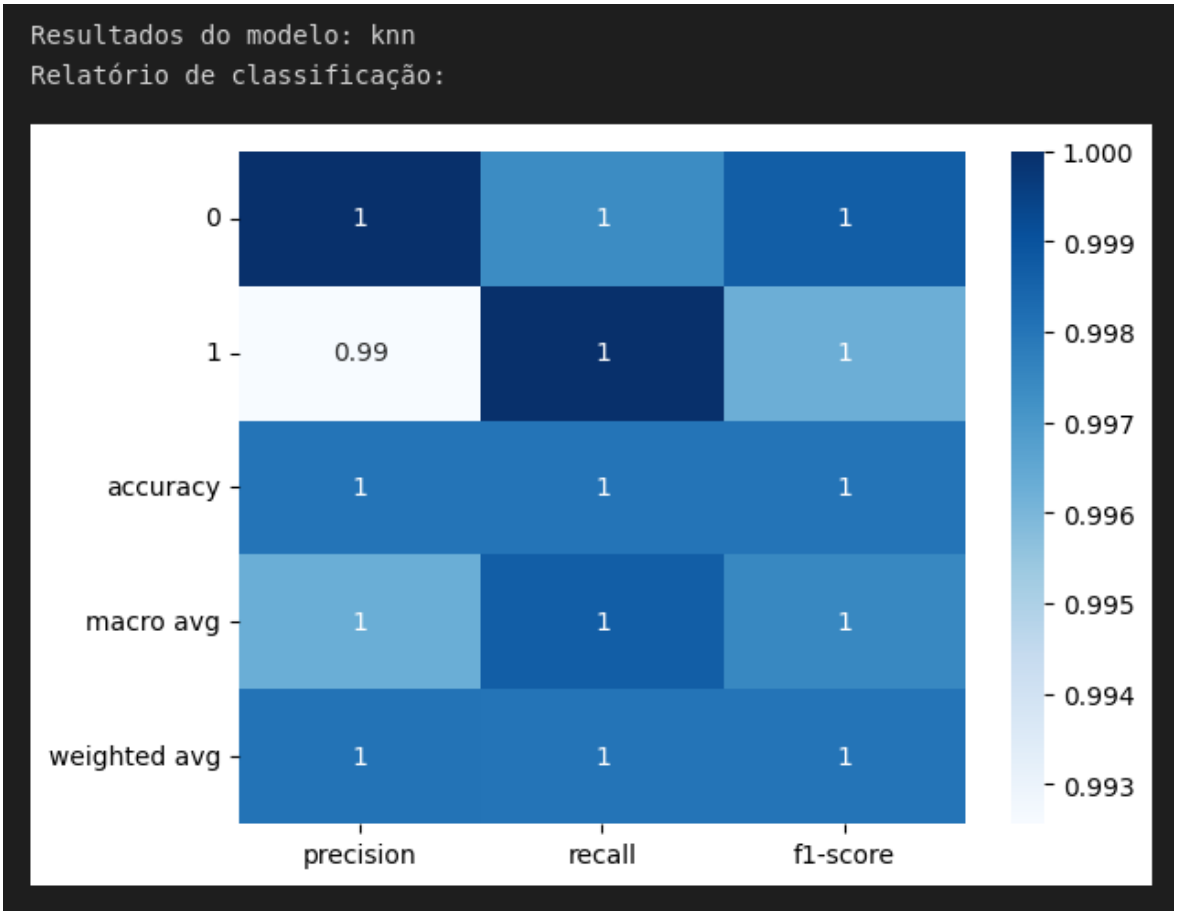
1. precision: a porcentagem de verdadeiros positivos dentre todos os previstos como positivos (verdadeiros positivos + falsos positivos);
2. recall: a porcentagem de verdadeiros positivos dentre todos os realmente positivos (verdadeiros positivos + falsos negativos). Varia entre 0 e 1, sendo 1 a melhor performance;
3. F1-score: média harmônica do precision e do recall. Varia entre 0 e 1, sendo 1 a melhor performance;
4. support: o número de amostras de cada classe no dataset.

O classification_report calcula essas métricas para cada classe do dataset e também a média de todas as classes. Também mostra a média com pesos (weighted average) das métricas, considerando o support (número de instâncias) de cada classe.
As linhas são:

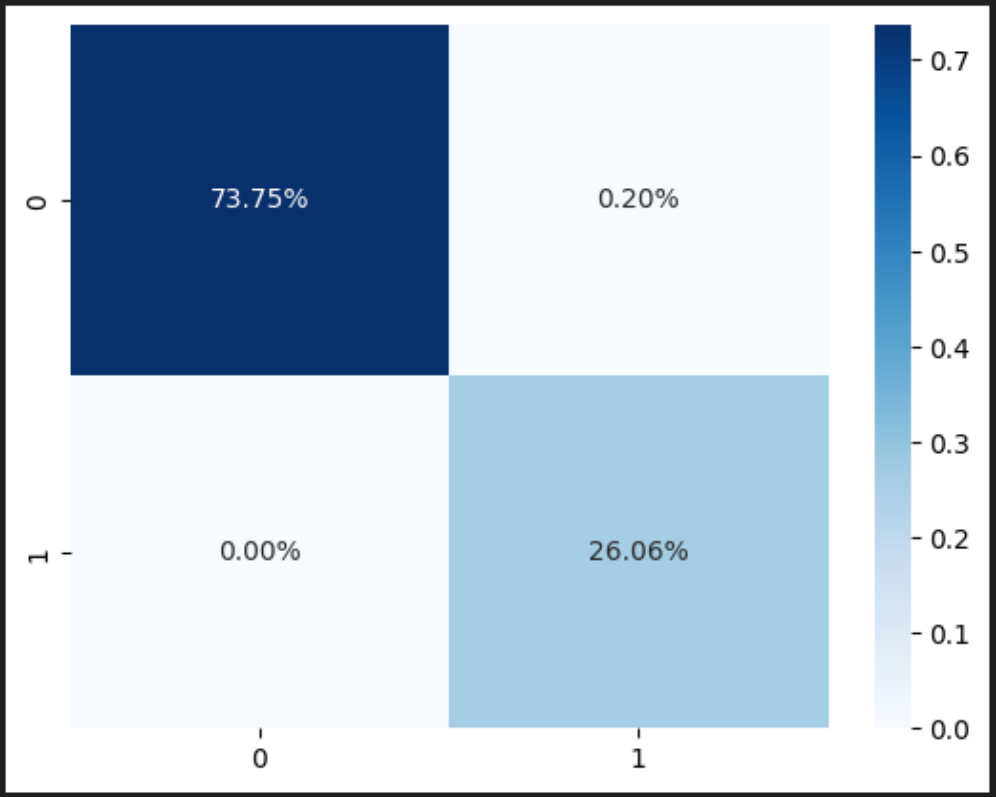
1. 0 e 1 são os labels das classes;

2. accuracy: a porcentagem entre o número total de classificações corretas (verdadeiros positivos e verdadeiros negativos) e o número total de instâncias do dataset;
3. macro avg (macro-average): a média das métricas (precision, recall, F1-score) para cada classe sem considerar 1 das classes pode ter muito mais exemplos que a outra;
4. weighted avg (weighted average): a média das métricas para cada classe, considerando o número de instâncias (support) de cada classe. O peso é proporcional ao número de instâncias de cada classe. Dessa forma, a classe com mais exemplos têm um maior peso.

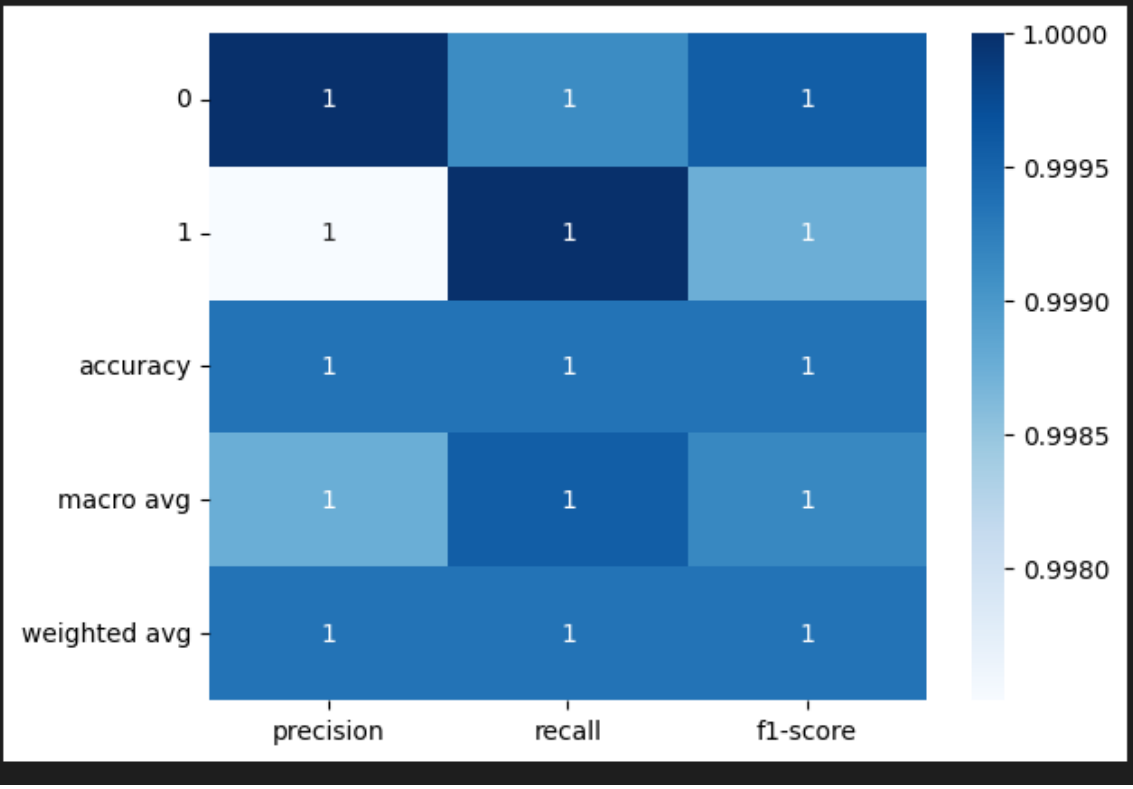
Notamos como a performance dos nossos modelos foram muito boas: 0.99 e 1.



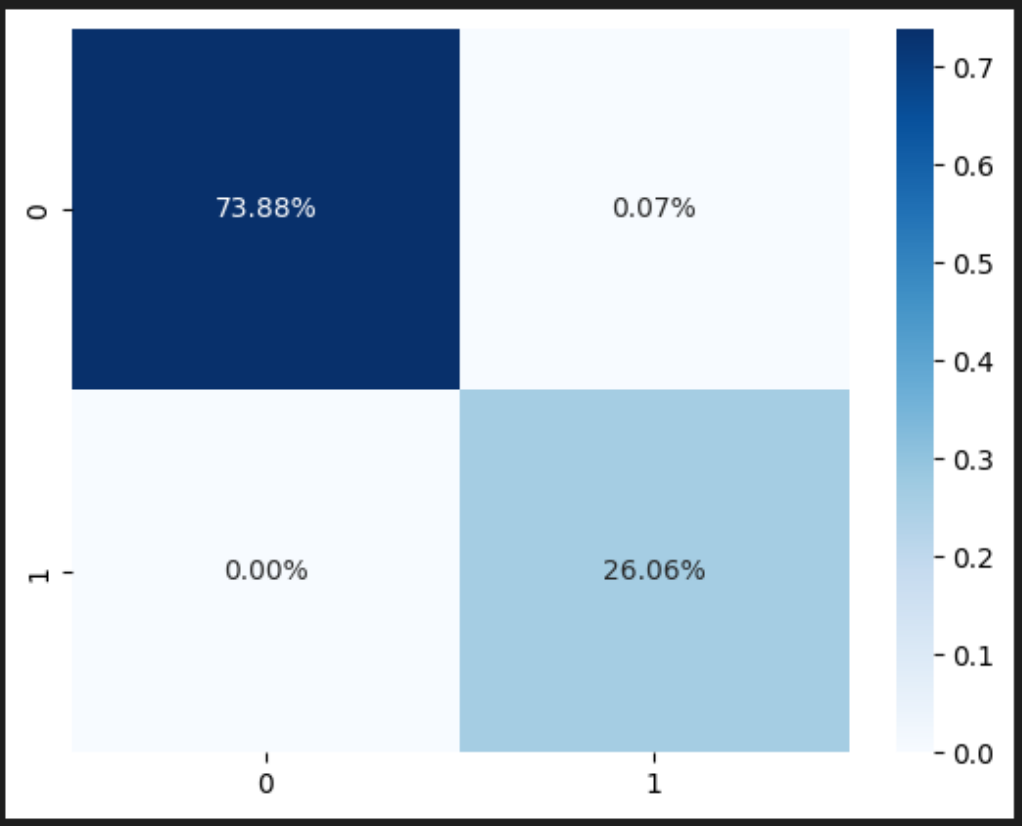
Matriz de confusão:



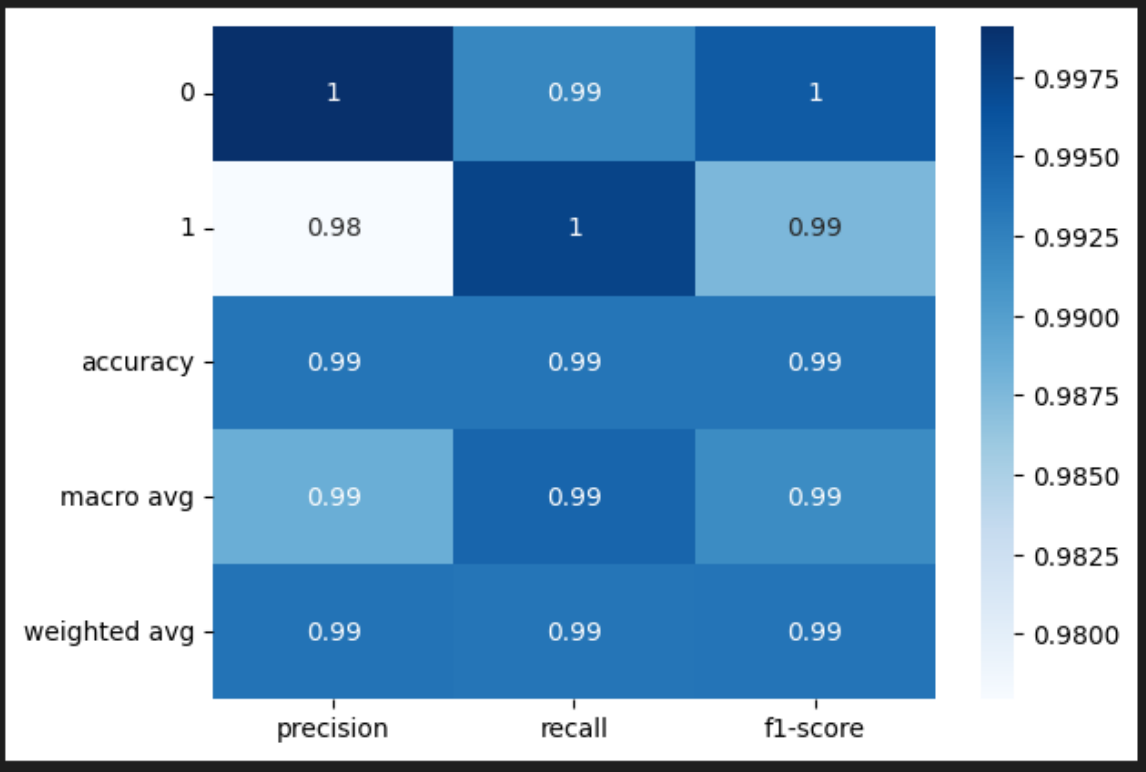
Resultados do modelo: decision tree
Relatório de classificação:



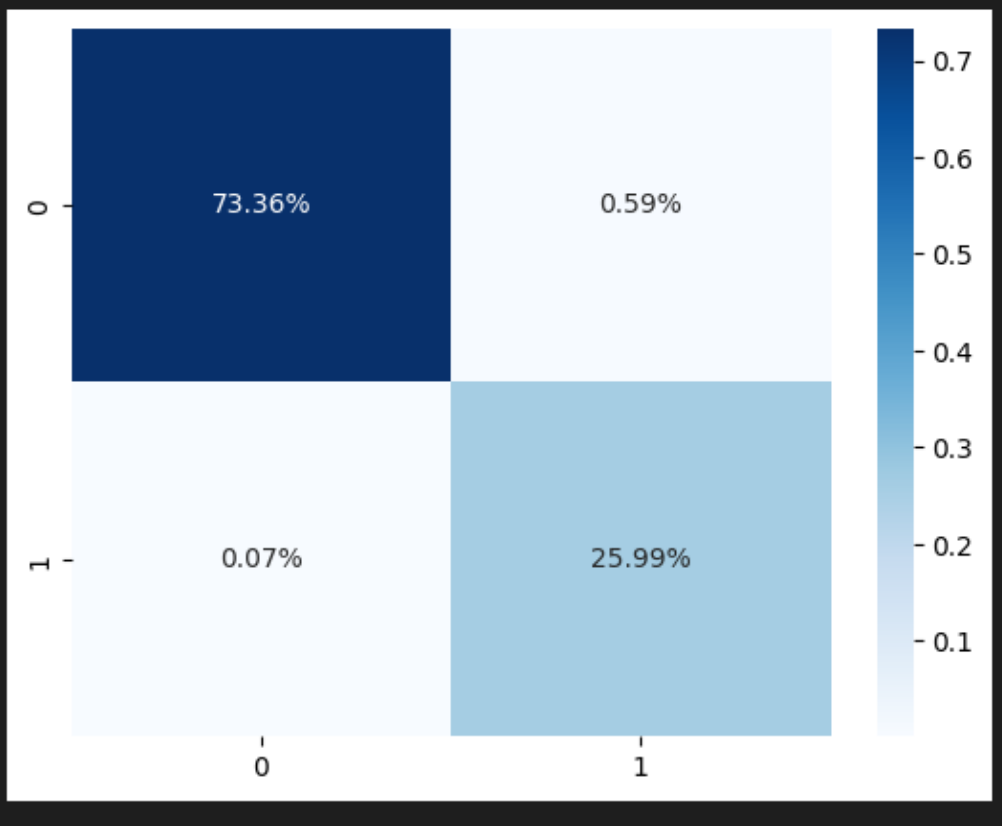
Matriz de confusão:



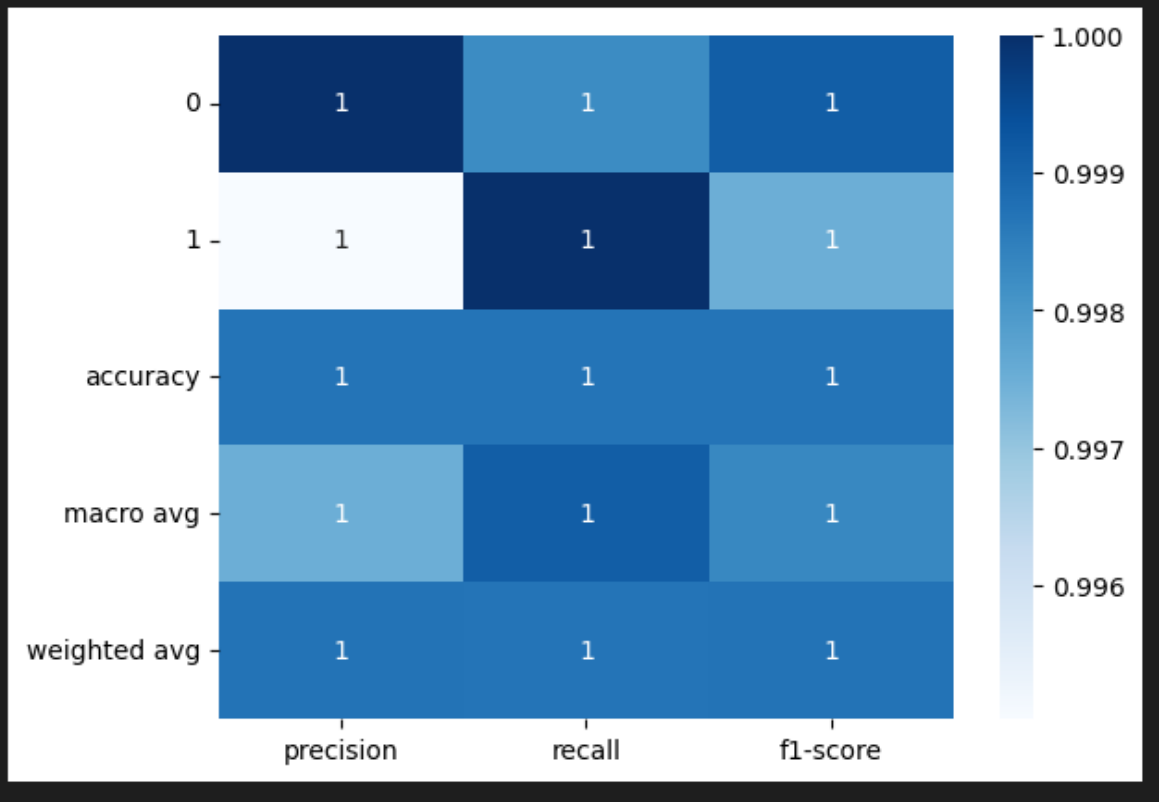
Resultados do modelo: svm
Relatório de classificação:



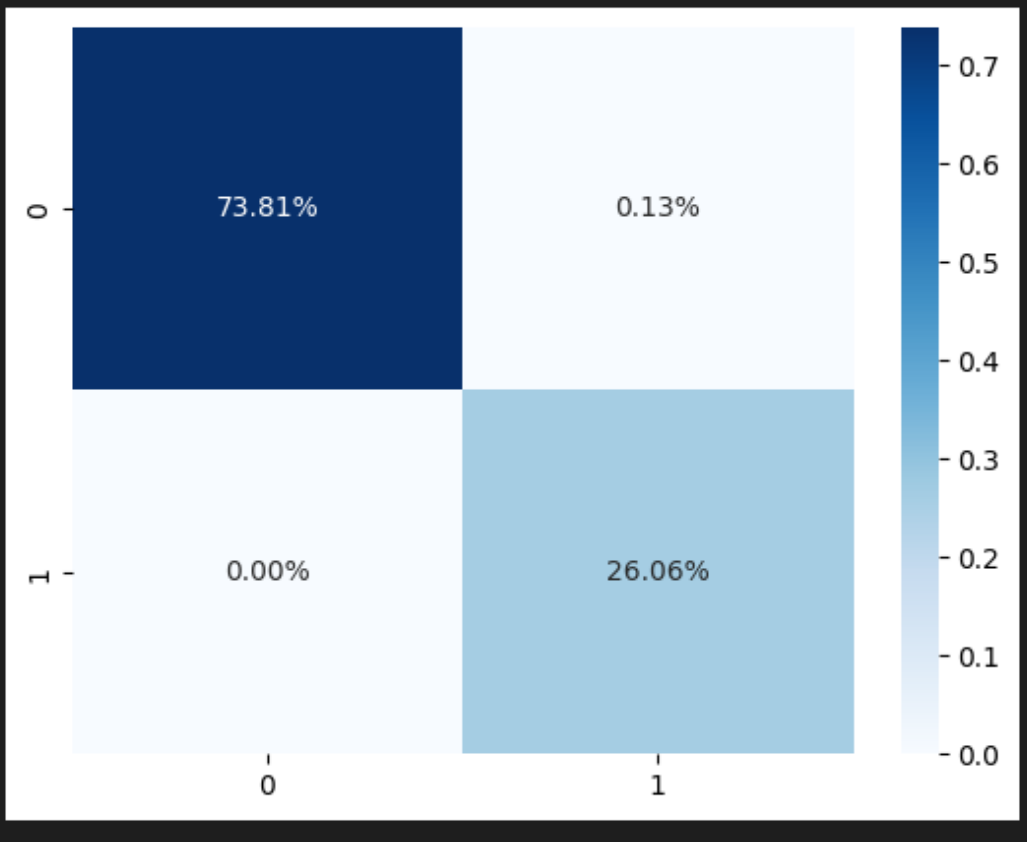
Matriz de confusão:



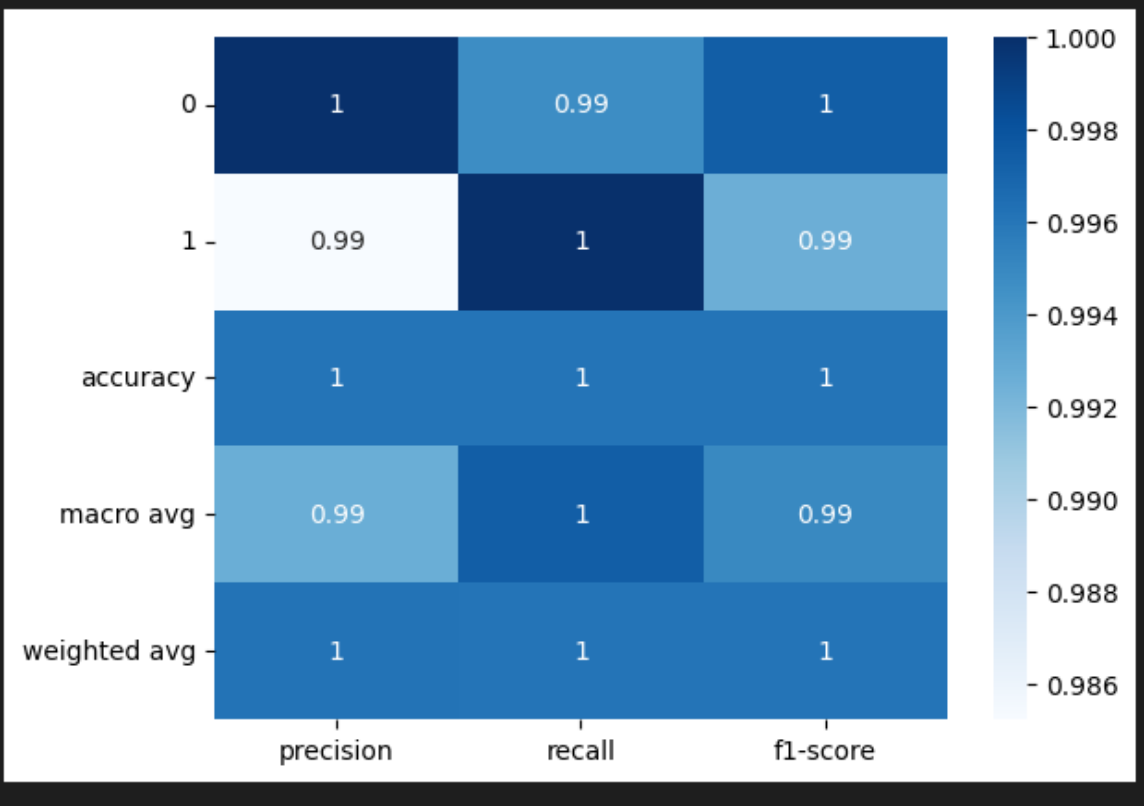
Resultados do modelo: random forest
Relatório de classificação:



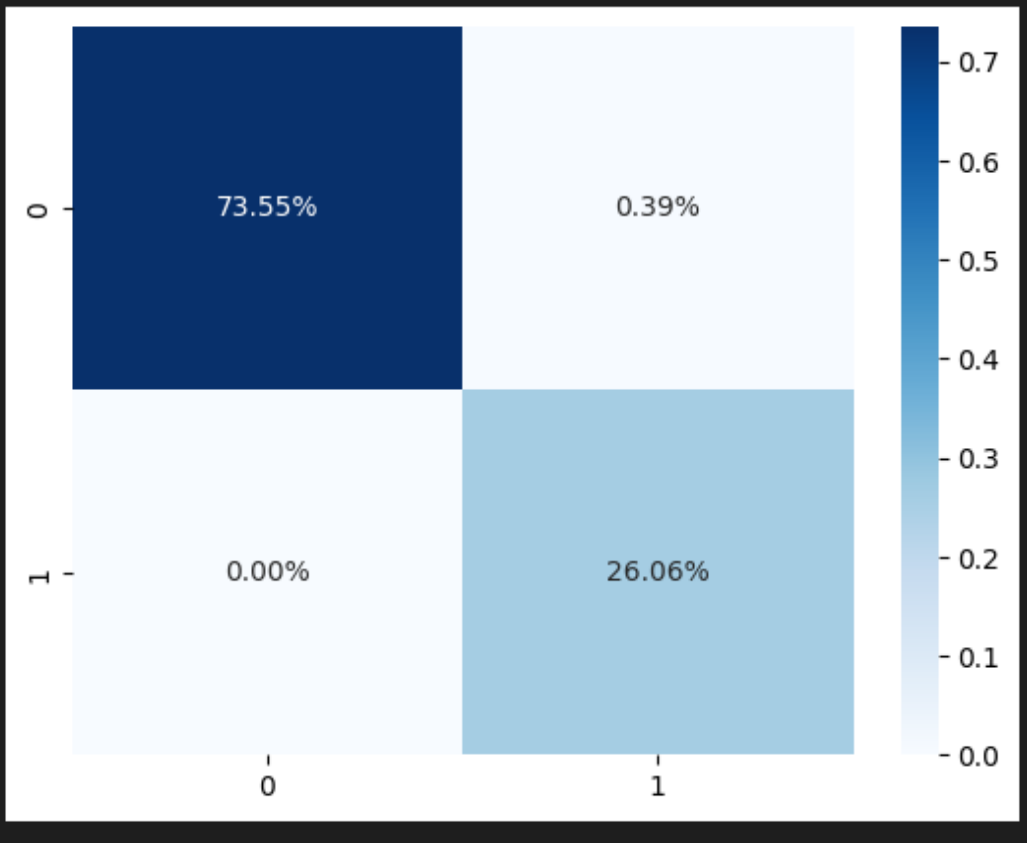
Matriz de confusão:



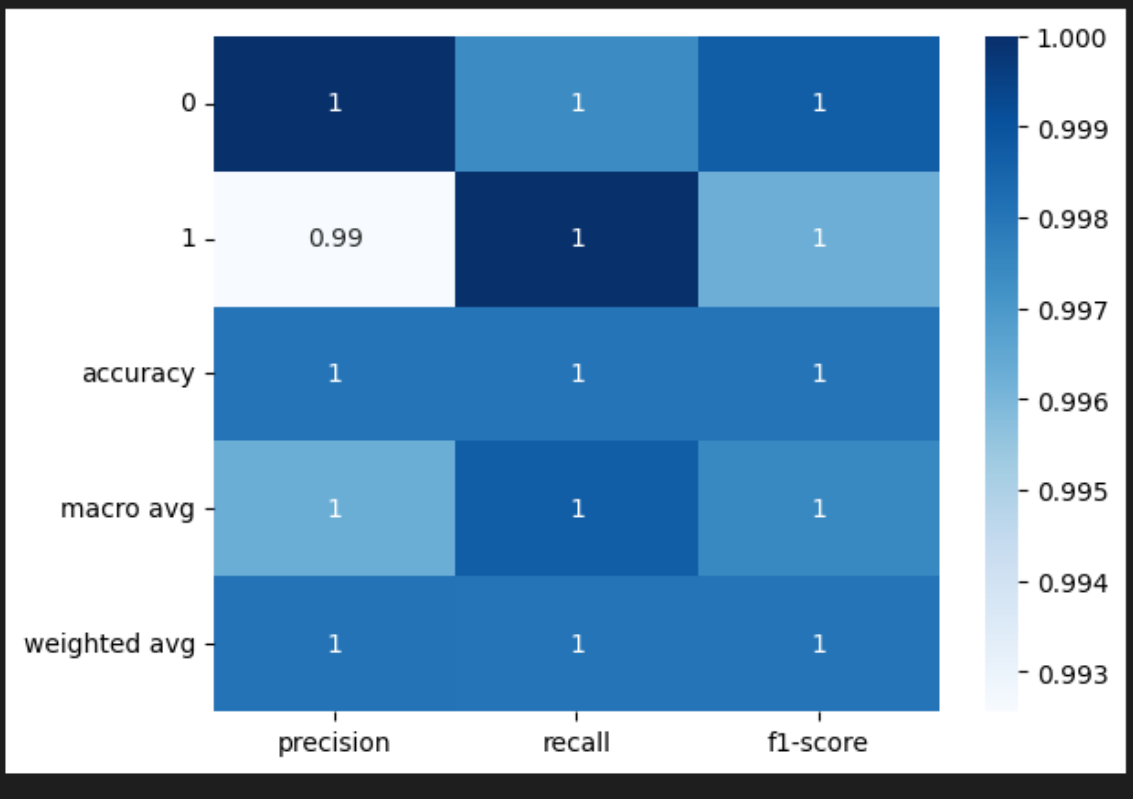
Resultados do modelo: mlp
Relatório de classificação:



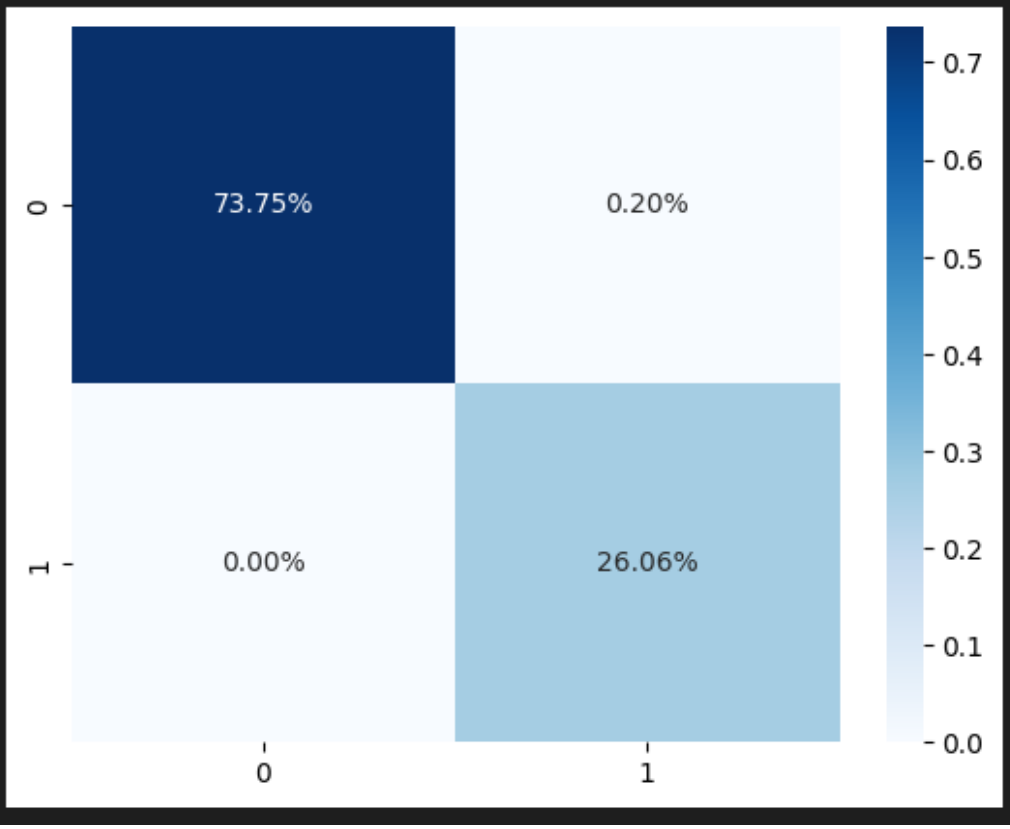
Matriz de confusão:



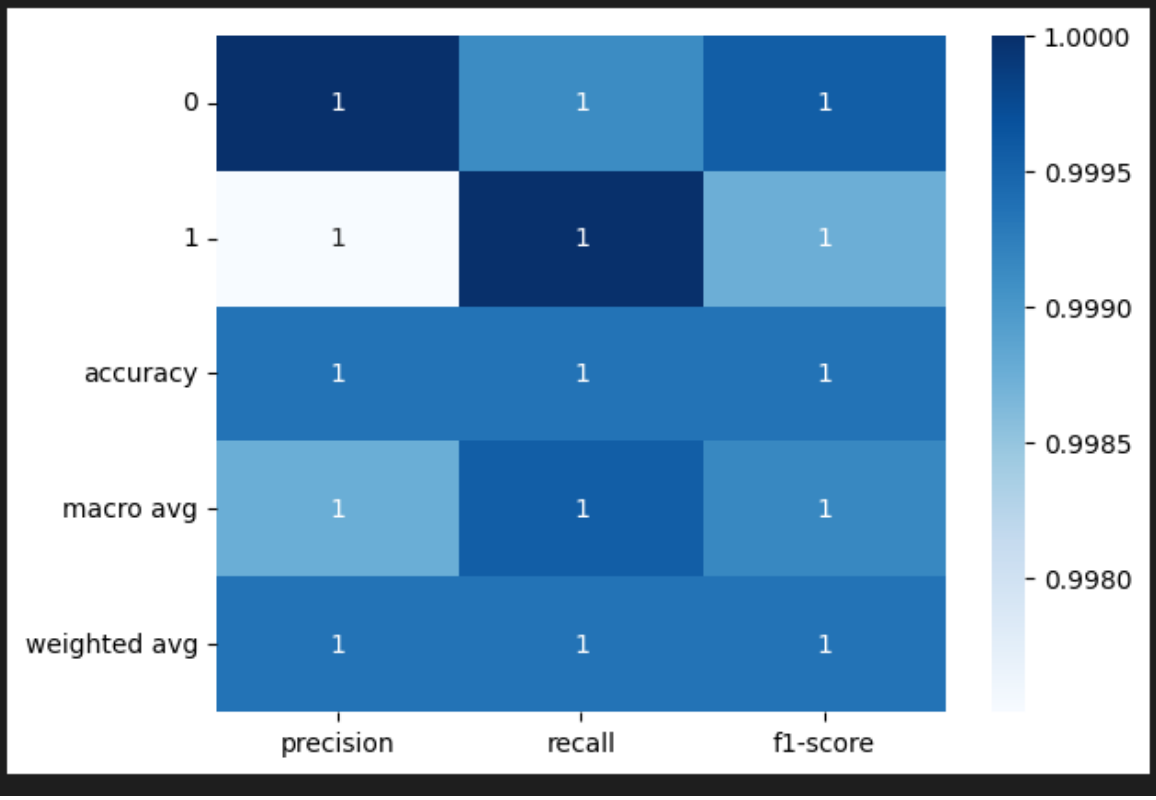
Resultados do modelo: ensemble mlp (bagging)
Relatório de classificação:



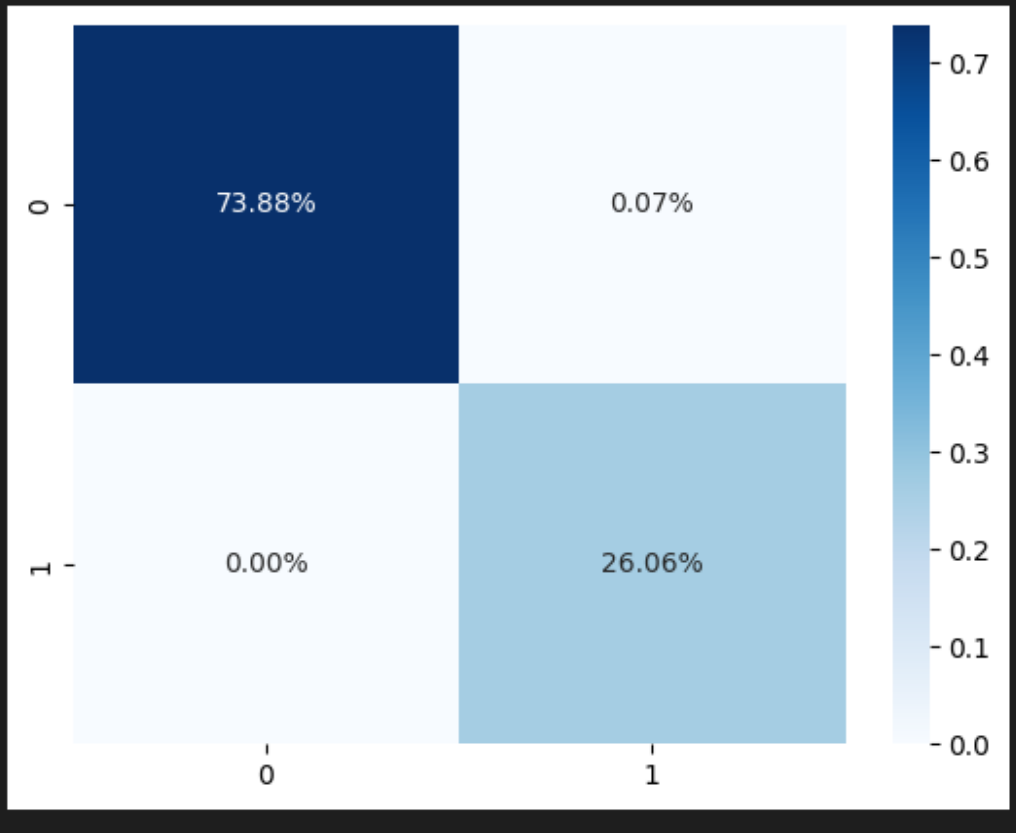
Matriz de confusão:



Resultados do modelo: heterogeneous ensemble (voting)
Relatório de classificação:



Matriz de confusão:



5.2 Testes de significância estatística

Para os testes de significância estatística, selecionamos um dos modelos mais promissores, o decision tree, e realizamos teste de significância pareados entre o os resultados do decision tree e os dos demais modelos. Especificamente, utilizamos k-fold com 10 folds para gerar uma lista de 10 precisões para cada um dos modelos e analisamos se há alguma diferença estatística entre esses resultados, quando pareados. Como hipótese nula, utilizamos "Não há diferença significativa.", e "Há diferença significativa." como hipótese alternativa, além de um alpha de 0.05.

```
chosen_model = "decision tree"
chosen_model_cv_score = cv_scores[chosen_model]

h0 = "Não há diferença significativa."
h1 = "Há diferença significativa."

alpha = 0.05

for model_name, cv_score in cv_scores.items():
    if model_name != chosen_model:
        stat_study_conclusion = significance_level(chosen_model_cv_score, cv_score, h0, h1, a=alpha)
        print(f"Resultado para análise de significância entre {chosen_model} e {model_name}:")
        print(stat_study_conclusion)
        print()
```

✓ 0.0s

```
Resultado para análise de significância entre decision tree e knn:
Há diferença significativa.

Resultado para análise de significância entre decision tree e svm:
Há diferença significativa.

Resultado para análise de significância entre decision tree e random forest:
Há diferença significativa.

Resultado para análise de significância entre decision tree e mlp:
Há diferença significativa.

Resultado para análise de significância entre decision tree e ensemble mlp (bagging):
Há diferença significativa.

Resultado para análise de significância entre decision tree e heterogeneous ensemble (voting):
Há diferença significativa.
```

Resultado da execução dos testes de significância.

Como pode ser observado nas análises de classificação e matriz de confusão, a decision tree acabou com um resultado particularmente bom.

Com a adição dos testes de significância, podemos afirmar que não há diferenças significativas entre ela e os demais modelos, inclusive os que performaram de forma pior.
Aliamos isso ao fato de Decision Tree ser um modelo simples e de treinamento ágil, além de bastante explicável para alguém não técnico para concluir que:

Esse é o melhor modelo para o contexto apresentado.

5.3 Checando relevância dos atributos da Árvore de decisão

Utilizamos os atributos presentes numa decision tree treinada (feature_names_in_ e feature_importances_), para determinar quais features são mais relevantes para a decision tree, na hora de classificar.

```
features = decision_tree.feature_names_in_
importances = decision_tree.feature_importances_

feature_importances = []
for i, feature in enumerate(features):
    feature_importances.append({
        "feature": feature,
        "importance": round(importances[i], 3)
    })

feature_importances = sorted(feature_importances, key=lambda x: x["importance"], reverse=True)
feature_importances = feature_importances[:15]

print("Feature Importances:")
seaborn.set(rc={'figure.figsize':(20,10)})
seaborn.barplot(data=pandas.DataFrame(feature_importances), x="feature", y="importance")
```

Trecho de código que gerou o gráfico de relevância dos atributos

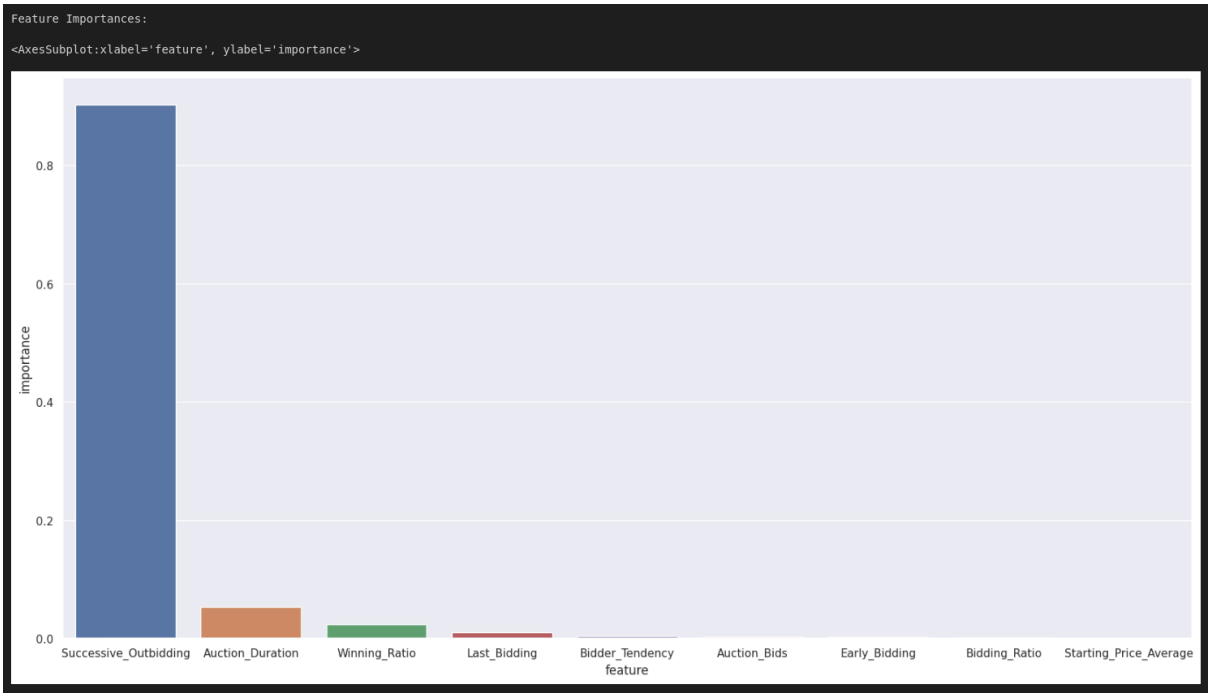


Gráfico de relevância dos atributos

É importante ressaltar que esses atributos, por si mesmos, não indicam exatamente se a presença ou falta deles é um ótimo indicativo do comportamento fraudulento que desejamos detectar, mas sim que a decision tree consegue ganhar muita informação ao dividir os registros com base no valor do atributo.

Dito isso, se somarmos o conhecimento obtido nessa análise com o contexto do nosso problema, podemos afirmar com bastante certeza que o atributo mais relevante para a árvore, Successive_Outbidding, é um indicador excelente de comportamento suspeito. O Successive_Outbidding indica o comportamento muito específico de um usuário que JÁ ESTÁ VENCENDO um leilão, mas CONTINUA A DAR LANCES, CADA VEZ MAIORES. É um comportamento ilógico que dificilmente partiria de um usuário legítimo, pois faria com que precisasse pagar mais sem necessidade.

Esse comportamento é exemplar de usuários falsos pertencentes ao dono do item em leilão, ou ao menos relacionados, que fazem isso apenas para inflacionar o valor do item, sem interesse em comprá-lo.
Sendo assim, após analisar o gráfico, as sugestões mais relevante para darmos seriam:

1. Imediatamente cancelar qualquer lance que se enquadre no padrão de Successive_Outbidding e bloquear o usuário responsável, mantendo um canal de comunicação aberto para os casos raros de usuários legítimos que por acaso fizeram isso, sem intenções maliciosas terem a opção de contestar o bloqueio.
2. Criar um alerta automático para a presença dos comportamentos descritos como sendo os 2º, 3º e 4º mais relevantes, que repassa o caso para as IA's desenvolvidas. Embora esses atributos (Auction Duration, Winning_Ration e Last_Bidding) sejam bem menos relevantes que o Successive_Outbidding para a classificação, até por representarem uma range de valores e não exatamente uma categoria, registros com esses valores próximos dos valores médios de comportamento fraudulentos são bastante suspeitos.

6. Deployment

O modelo selecionado pode ser facilmente integrado internamente dentro da plataforma de leilões para análise em tempo real de comportamentos suspeitos. Contudo, todo modelo de IA só é tão bom quanto os dados que usa para o seu treinamento, então é necessário um plano de monitoramento de performance, para verificar se o modelo tem algum decaimento de precisão com o passar do tempo, e re-treinamentos periódicos com datasets mais completos e atualizados, que vão acumulando com o tempo, no decorrer do funcionamento da aplicação.

6.1 Final Project Review

O projeto foi muito interessante para colocarmos em prática tudo que aprendemos na disciplina. Em especial, o conhecimento acerca de comparação entre modelos foi algo que nunca tínhamos visto antes, e não tínhamos noção da profundidade que essa decisão acarretava. Durante o desenvolvimento do projeto, sentimos dificuldades na seleção de parâmetros para a variação paramétrica, pelo volume alto de opções disponíveis, e por causa do tempo necessário para executar a otimização de alguns modelos. Dito isso, por causa dessa dificuldade, tivemos a chance de conhecer o RandomizedSearch, que produz resultados quase tão bons quanto a busca exaustiva do GridSearch, mas com apenas uma fração do tempo.

Também é válido destacar o conhecimento formal adquirido em relação ás etapas do CRISP-DM, que é um ótimo padrão de projeto a ser seguido, bastante objetivo e produtivo, e que, infelizmente, ainda não tínhamos tido contato no âmbito da academia. Acredito que o CRISP-DM deveria ser introduzido mais cedo na grade curricular acadêmica, em outras disciplinas até mesmo obrigatória, e não “apenas“ numa eletiva, pois é um conhecimento particularmente relevante para quem deseja seguir alguma carreira na área de dados (seja engenharia, análise ou ciência propriamente dita).