AP2 - Projeto de Aprendizagem 1: Regressão com Redes Neurais

Equipe: José Pedro, Lauro Aguiar, Miguel, Rafael Alves

A base de dados utilizada foi a "Auction Verification", disponível no repositório UCI Machine Learning Repository. O trabalho foi realizado utilizando a linguagem Python, no ambiente Colab.

Referência

Ordoni,Elaheh, Bach,Jakob, Fleck,Ann-Katrin, and Bach,Jakob. (2022). Auction Verification. UCI Machine Learning Repository. https://doi.org/10.24432/C52K6N.

Introdução

Neste projeto de aprendizagem, será realizada a análise de dados e a construção de um modelo de regressão utilizando redes neurais, especificamente o Perceptron Multicamadas (MLPRegressor), aplicados ao dataset "Auction Verification". Este trabalho é parte da disciplina de Inteligência de Mercado, do curso de Ciências Econômicas, sob a orientação do professor Christiano Alves Farias, com a data de entrega estabelecida para 31/05/2024 e um valor total de 10 pontos em 40pts da AC.

Objetivo do Projeto

O objetivo deste projeto é explorar e analisar o dataset "Auction Verification", que foi criado como parte de um estudo científico para investigar a possibilidade de substituir a verificação custosa de modelos de processos complexos (como leilões simultâneos de múltiplas rodadas para a venda de espectros de frequência) por previsões dos resultados. Através de diversas etapas, desde a preparação dos dados até a interpretação do modelo, buscamos entender e prever o comportamento dos leilões de forma eficiente.

Descrição da Base de Dados

O dataset "Auction Verification" contém instâncias que representam execuções de verificação de leilões. Cada instância verifica se um determinado preço é possível para um determinado produto e, em alguns casos, se um determinado licitante pode ganhar o produto por aquele preço. É importante ressaltar que o dataset não possui valores ausentes, o que facilita o processo de análise e modelagem.

1. Preparação dos dados

Estudo das Variáveis

O dataset "Auction Verification" possui as seguintes variáveis, conforme descrito na tabela "Variable Table" no site do banco de dados.

Nome da Variável	Função	Tipo	Descrição	Unidades	Valores Faltantes
process.b1.capacity	Feature	Inteiro	Capacidade (número máximo de produtos para ganhar) do Licitante 1.	-	não
process.b2.capacity	Feature	Inteiro	Capacidade (número máximo de produtos para ganhar) do Licitante 2.	-	não
process.b3.capacity	Feature	Inteiro	Capacidade (número máximo de produtos para ganhar) do Licitante 3.	-	não
process.b4.capacity	Feature	Inteiro	Capacidade (número máximo de produtos para ganhar) do Licitante 4.	-	não
property.price	Feature	Inteiro	Preço atualmente verificado.	-	não
property.product	Feature	Inteiro	Produto atualmente verificado.	-	não
property.winner	Feature	Inteiro	Licitante atualmente verificado como vencedor do produto (0 se apenas preço verificado).	-	não
verification.result	Target	Categórico	Resultado da verificação binária - é possível verificar o resultado?	-	não
verification.time	Target	Contínuo	Tempo de execução do procedimento de verificação.	-	não

1.1 Coleta dos dados

```
# Install the ucimlrepo package
!pip install ucimlrepo

# Import the dataset and required libraries
from ucimlrepo import fetch_ucirepo
 import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Fetch dataset
auction_verification = fetch_ucirepo(id=713)

# Data (as pandas dataframes)
X = auction_verification.data.features
y = auction_verification.data.targets

```

```
12. # Convert to DataFrame for ease of use
13. X_df = pd.DataFrame(X)
14. y_df = pd.DataFrame(y)
15.
16. # Check for missing values
17. missing_values_X = X_df.isnull().sum()
18. missing_values_y = y_df.isnull().sum()
19.
20. # Basic information about the dataset
21. info_X = X_df.info()
22. info_y = y_df.info()
23.
24. # Descriptive statistics for features
25. desc_X = X_df.describe()
26.
27. # Descriptive statistics for targets
28. desc_y = y_df.describe()
```

```
RangeIndex: 2043 entries, 0 to 2042
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   process.b1.capacity  2043 non-null   int64
 1   process.b2.capacity  2043 non-null   int64
 2   process.b3.capacity  2043 non-null   int64
 3   process.b4.capacity  2043 non-null   int64
 4   property.price       2043 non-null   int64
 5   property.product     2043 non-null   int64
 6   property.winner      2043 non-null   int64
```

Integridade dos Dados: Não há valores faltantes no conjunto de dados, o que facilita a análise e o uso para treinamento de modelos de machine learning sem a necessidade de tratamento de valores nulos.

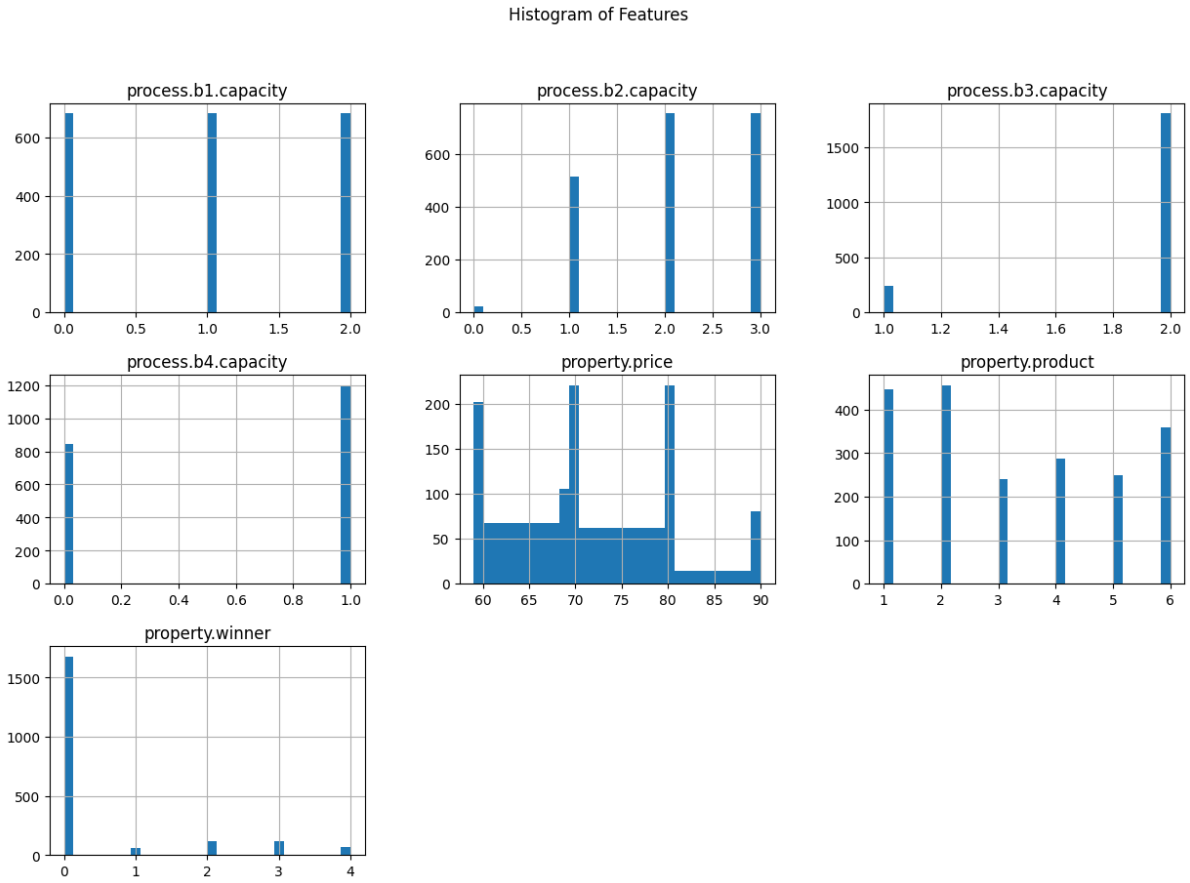
Estrutura Consistente: O tipo de dado consistente (int64) para todas as colunas simplifica a análise e a manipulação dos dados.

1.2. Análise exploratória dos dados

```
# Visualizations
# Histogram for each feature
X_df.hist(bins=30, figsize=(15, 10))
plt.suptitle('Histogram of Features')
plt.show()

# Box plots for each feature
X_df.plot(kind='box', subplots=True, layout=(3, 3), figsize=(15, 10), sharex=False,
sharey=False)
```

```
plt.suptitle('Box Plot of Features')
plt.show()

# Correlation matrix
corr_matrix = X_df.corr()

# Heatmap of the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix of Features')
plt.show()

# Scatter plot for pairs of features
sns.pairplot(X_df)
plt.suptitle('Pairplot of Features')
plt.show()

(missing_values_X, missing_values_y, desc_X, desc_y)
```

1.2.1 Histograma com principais variáveis



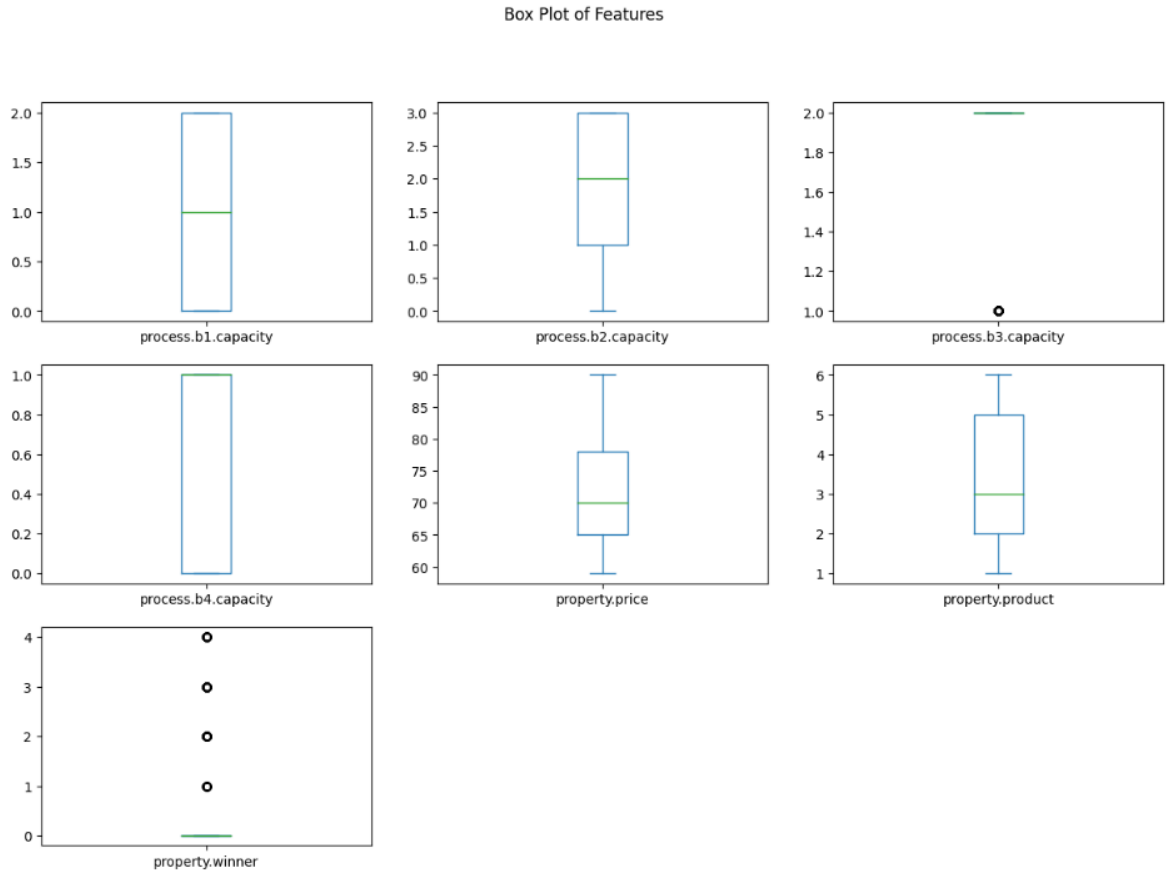
As características process.b1.capacity, process.b2.capacity, process.b3.capacity, e process.b4.capacity são discretas, com valores específicos que dominam a distribuição.

property.price tem uma distribuição multimodal, indicando que há várias faixas de preços preferenciais.

property.product é uniformemente distribuído entre 1 e 6, indicando que não há um valor que domine a distribuição.

property.winner é fortemente concentrado em 0, com alguns valores menores distribuídos entre 1 e 4.

1.2.2 Box Plot



process.b1.capacity: O box plot confirma que os valores predominantes são 0, 1 e 2, com uma distribuição uniforme.

process.b2.capacity: O box plot mostra uma distribuição uniforme similar ao **histograma.**

process.b3.capacity: O box plot mostra que a maioria dos valores são 2, confirmando o pico visto no histograma. O outlier em 1 também está presente no box plot.

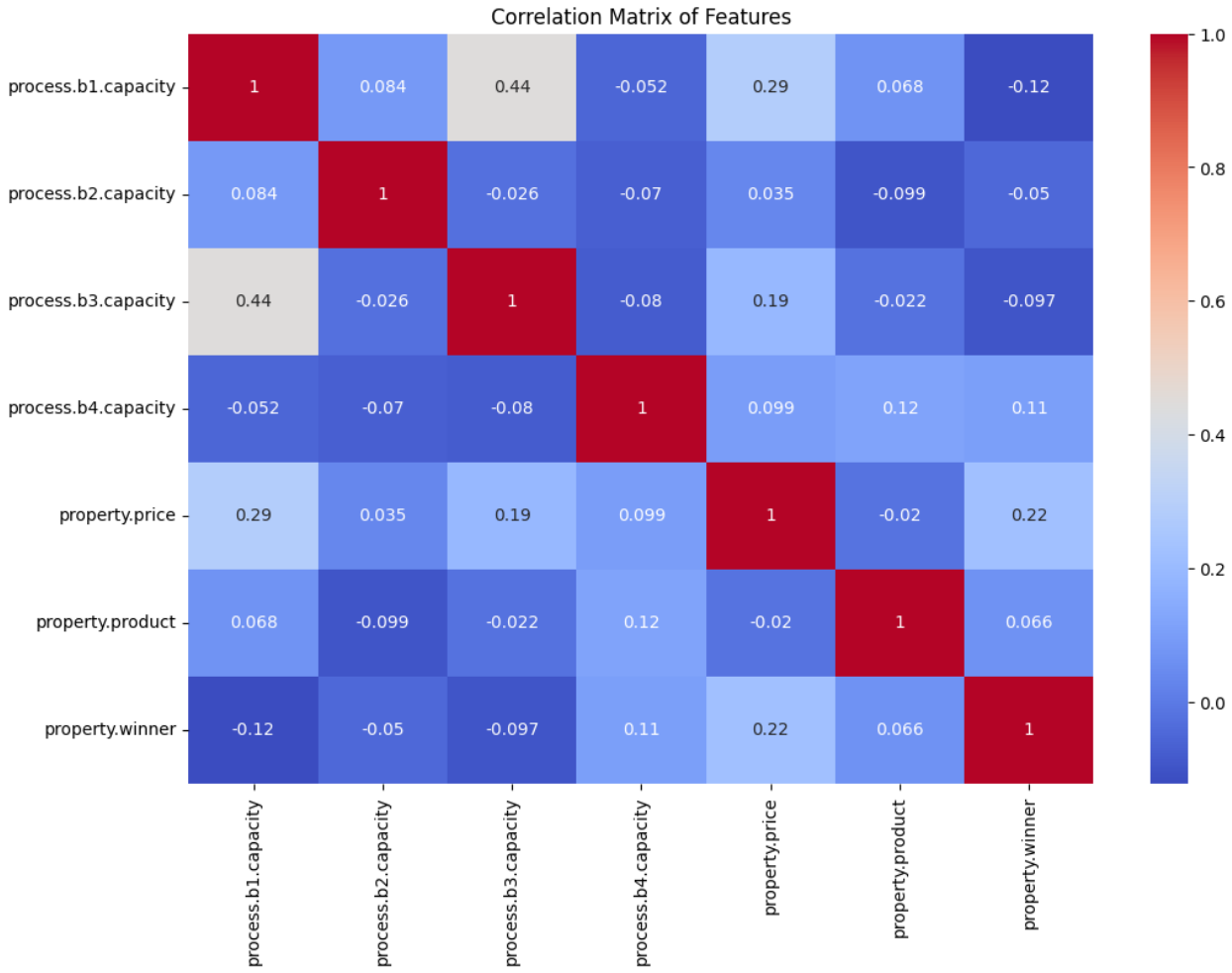
process.b4.capacity: O box plot confirma que a maioria dos valores são 1, como visto no histograma.

property.price: O box plot mostra a faixa de valores de 60 a 90, com uma mediana em 70, refletindo a distribuição vista no histograma.

property.product: O box plot confirma a distribuição relativamente uniforme dos valores 1 a 6.

property.winner: O box plot mostra que a maioria dos valores é 0, com outliers em 1, 2, 3 e 4, refletindo a distribuição vista no histograma.

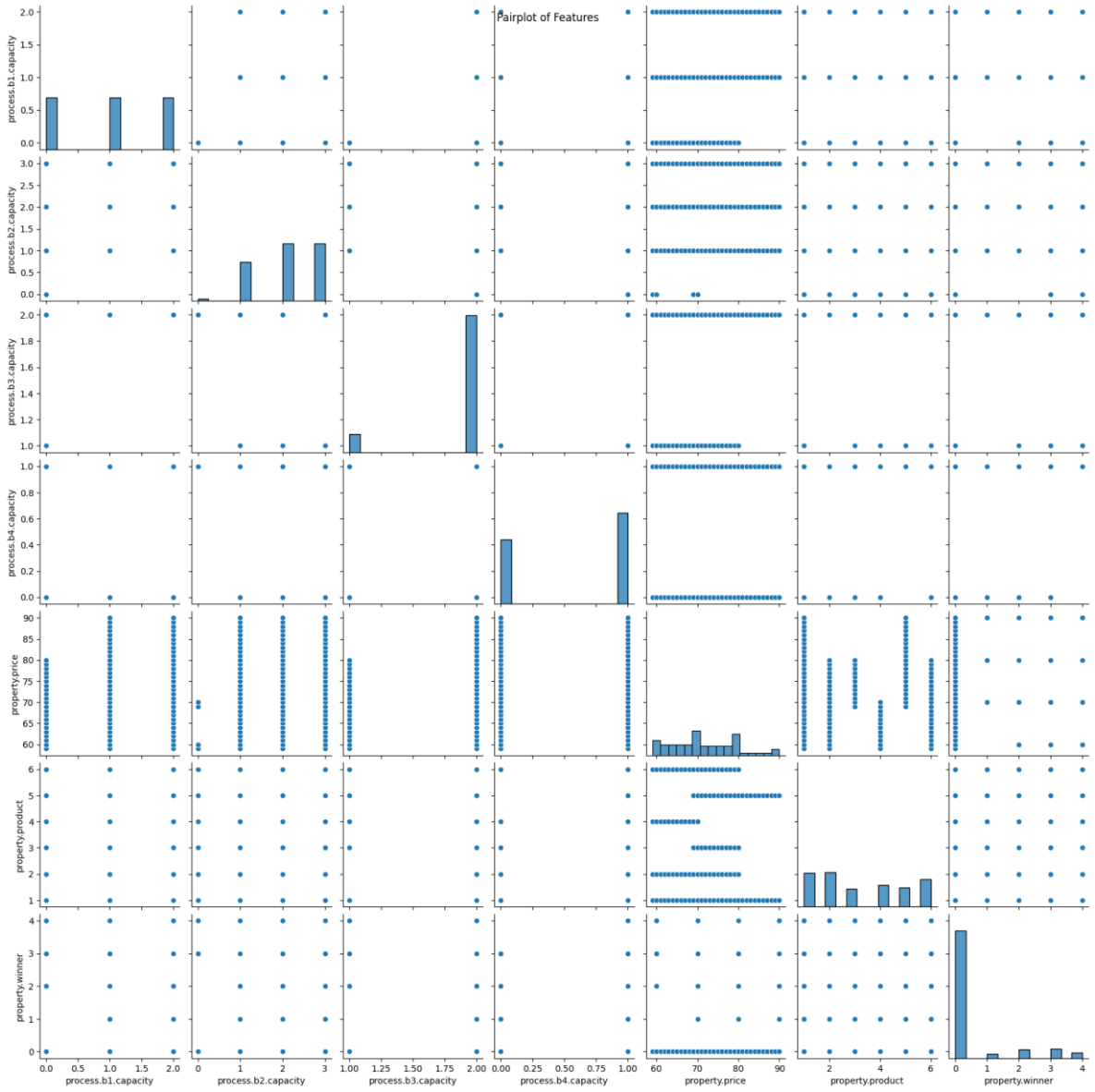
1.2.3 Matriz de Correlação



A variável `process.b1.capacity` tem uma correlação positiva moderada com `process.b3.capacity` (0.44) e uma correlação positiva fraca com `property.price` (0.29). Com as outras variáveis, as correlações são fracas. Já `process.b2.capacity` não apresenta correlações fortes com nenhuma outra variável, sendo todas as correlações fracas e próximas de zero, indicando pouca ou nenhuma relação linear com as outras variáveis. A `process.b3.capacity`, além da correlação positiva moderada com `process.b1.capacity` (0.44), não possui outras correlações significativas, mostrando apenas correlações fracas com as demais variáveis. A `process.b4.capacity` apresenta correlações fracas com todas as outras variáveis, todas próximas de zero.

A variável `property.price` tem uma correlação positiva moderada com `property.winner` (0.22), enquanto as correlações com as outras variáveis são fracas. A `property.product` não possui correlações significativas com nenhuma outra variável, todas sendo fracas e próximas de zero. Por fim, a `property.winner` apresenta uma correlação positiva moderada com `property.price` (0.22) e correlações fracas com as outras variáveis.

As conclusões gerais indicam que as correlações mais fortes encontradas são entre `process.b1.capacity` e `process.b3.capacity` (0.44) e entre `property.price` e `property.winner` (0.22). A maioria das correlações são fracas ou inexistentes, sugerindo que as variáveis não têm relações lineares fortes entre si. A relação positiva moderada entre `property.price` e `property.winner` sugere que, à medida que o preço aumenta, a probabilidade de um vencedor verificado também aumenta, embora essa relação não seja muito forte.



O pairplot confirma muitas das observações feitas anteriormente com base nos histogramas e na matriz de correlação:

process.b1.capacity e process.b3.capacity têm uma correlação moderada, visível nos agrupamentos.

property.price tem uma leve correlação com property.winner, indicando que os vencedores são mais prováveis em preços mais altos.

A maioria das outras relações são fracas, com dispersões amplas e sem padrões claros.

1.3. Pré-processamento dos dados

```
from sklearn.preprocessing import StandardScaler

# Normalizing the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_df)

# Convert to DataFrame
X_scaled_df = pd.DataFrame(X_scaled, columns=X_df.columns)
```

1.4. Divisão dos dados em treinamento, validação e teste

```
from sklearn.model_selection import train_test_split

# Split data
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled_df, y_df, test_size=0.3,
random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42)

# Check proportions
print(f'Train set: {len(X_train)} samples')
print(f'Validation set: {len(X_val)} samples')
print(f'Test set: {len(X_test)} samples')
```

Train set: 1430 samples

Validation set: 306 samples

Test set: 307 samples

Os dados serão divididos em conjuntos de treino (70%), validação (15%) e teste (15%). A escolha dessa proporção é baseada em práticas comuns que equilibram a quantidade de dados para treinamento e validação/teste.

2. Seleção e treinamento do modelo

2.1. Escolha da técnica de regressão pelo método Rede Neural Perceptron Multicamadas (MLPRegressor).

```
from sklearn.neural_network import MLPRegressor

# Initialize model
mlp = MLPRegressor(hidden_layer_sizes=(100,), max_iter=1000,
random_state=42)
```

2.2. Seleção das variáveis preditoras

2.3. Treinamento do modelo

2.4. Avaliação do modelo com o conjunto de validação

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Selecionar variáveis com base na análise exploratória e métodos
de seleção
selected_features = [
    'process.b1.capacity',
    'process.b2.capacity',
    'process.b3.capacity',
    'process.b4.capacity',
    'property.price',
    'property.product',
    'property.winner'
]

# Supomos que X e y já foram definidos previamente conforme as
instruções anteriores
# Por exemplo:
# X, y = auction_verification.data.features,
auction_verification.data.targets

# Selecionar apenas as variáveis indicadas
X_selected = X[selected_features]

# Normalização
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_selected)

# Divisão dos dados
```

```
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y,
test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)

# Criação e treinamento do modelo
mlp = MLPRegressor(hidden_layer_sizes=(100,), max_iter=500,
random_state=42)
mlp.fit(X_train, y_train)

# Avaliação no conjunto de validação
y_val_pred = mlp.predict(X_val)
print(f"Validação MSE: {mean_squared_error(y_val, y_val_pred)}")
print(f"Validação R^2: {r2_score(y_val, y_val_pred)}")
```

Validação MSE: 45701101.56123714

Validação R^2: -0.007393380963516105

/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't converged yet.

warnings.warn(

Os resultados obtidos indicam que o modelo não está performando bem. O valor negativo de R^2 sugere que o modelo está performando pior do que uma linha base que simplesmente prevê a média do alvo para todos os exemplos.

```
# Importar as bibliotecas necessárias
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor

# Supondo que X_df e y_df já tenham sido definidos como DataFrames
anteriormente

# Verificar a forma de y
print(y_df.shape)

# Se y_df tiver mais de uma coluna, precisamos selecionar apenas
uma para a regressão
# Supondo que y_df tenha apenas uma coluna chamada 'target_column'
if y_df.shape[1] == 1:
    y = y_df.values.ravel()
```

```
else:
    # Selecionando a primeira coluna para o exemplo
    y = y_df.iloc[:, 0].values

# Escalar os dados de X
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_df)

# Definindo o modelo de MLP (Multi-layer Perceptron) para regressão
mlp = MLPRegressor(random_state=1, max_iter=300)

# Realizar validação cruzada com 5 folds usando R^2 como métrica de
avaliação
cv_scores = cross_val_score(mlp, X_scaled, y, cv=5, scoring='r2')

# Calcular a média dos coeficientes de determinação (R^2)
mean_r2 = np.mean(cv_scores)

# Exibir o resultado
print(f"Média do coeficiente de determinação R^2 da validação
cruzada: {mean_r2:.2f}")
```

Média do coeficiente de determinação R^2 da validação cruzada: 0.29

Desempenho do Modelo: A média do coeficiente de determinação R^2 da validação cruzada é 0.29. Em termos gerais, um R^2 de 0.29 indica que o modelo, neste caso um regressor de perceptron multicamadas (MLP), explica aproximadamente 29% da variabilidade dos dados em torno da média. Este não é um valor muito alto para R^2, sugerindo que o modelo pode não estar muito bem ajustado aos dados ou que os dados não são facilmente previsíveis usando este modelo.

Abaixo, segue o coeficiente de determinação (R^2) especificado por cada *fold.*

```
# Importar as bibliotecas necessárias
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor

# Supondo que X_df e y_df já tenham sido definidos como DataFrames
anteriormente

# Verificar a forma de y
print(y_df.shape)

# Se y_df tiver mais de uma coluna, precisamos selecionar apenas
uma para a regressão
# Supondo que y_df tenha apenas uma coluna chamada 'target_column'
if y_df.shape[1] == 1:
    y = y_df.values.ravel()
```

```
else:
    # Selecionando a primeira coluna para o exemplo
    y = y_df.iloc[:, 0].values

# Escalar os dados de X
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_df)

# Definindo o modelo de MLP (Multi-layer Perceptron) para regressão
mlp = MLPRegressor(random_state=1, max_iter=300)

# Realizar validação cruzada com 5 folds usando R^2 como métrica de
avaliação
cv_scores = cross_val_score(mlp, X_scaled, y, cv=5, scoring='r2')

# Exibir os coeficientes de determinação R^2 para cada fold da
validação cruzada
for i, score in enumerate(cv_scores, start=1):
    print(f"Coeficiente de determinação R^2 do fold {i}:
{score:.2f}")
```

Coeficiente de determinação R^2 do fold 1: 0.26

Coeficiente de determinação R^2 do fold 2: 0.30

Coeficiente de determinação R^2 do fold 3: 0.28

Coeficiente de determinação R^2 do fold 4: 0.35

Coeficiente de determinação R^2 do fold 5: 0.25

3. Ajuste e validação do modelo

3.1. Ajuste dos hiperparâmetros do modelo

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor

# Escalonando os dados
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Criação do modelo e ajuste dos hiperparâmetros com um espaço de
busca reduzido
mlp = MLPRegressor(random_state=42)
```

```
param_grid = {
    'hidden_layer_sizes': [(100,), (50, 50)],  # Reduzido para duas
configurações
    'activation': ['relu'],  # Manter apenas a função de ativação
mais comum
    'solver': ['adam'],  # Utilizar apenas um otimizador
    'alpha': [0.0001, 0.001],  # Menos opções para a regularização
L2
    'learning_rate': ['constant'],  # Uma opção de taxa de
aprendizado
    'max_iter': [500]  # Reduzir o número de iterações máximas
}

grid_search = GridSearchCV(mlp, param_grid, cv=5,
scoring='neg_mean_squared_error', verbose=1)
grid_search.fit(X_train_scaled, y_train)

# Melhores hiperparâmetros
print("Melhores Hiperparâmetros:", grid_search.best_params_)

# Avaliação no conjunto de validação com o melhor modelo
best_mlp = grid_search.best_estimator_
y_val_pred = best_mlp.predict(X_val_scaled)
print(f"Validação MSE: {mean_squared_error(y_val, y_val_pred)}")
print(f"Validação R^2: {r2_score(y_val, y_val_pred)}")

# Avaliação no conjunto de teste
y_test_pred = best_mlp.predict(X_test_scaled)
print(f"Teste MSE: {mean_squared_error(y_test, y_test_pred)}")
print(f"Teste R^2: {r2_score(y_test, y_test_pred)}")
```

Fitting 5 folds for each of 4 candidates, totalling 20 fits

Melhores Hiperparâmetros: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'constant', 'max_iter': 500, 'solver': 'adam'}

Validação MSE: 26946327.4039434

Validação R^2: -0.9753288666468064

Teste MSE: 33296569.56452253

Teste R^2: -0.4886563641457833

A fim de buscar uma melhor performance do modelo foi realizado os seguintes ajustes, conforme código abaixo

Correção do Shape do Target (y):

Certificamos que y estava no formato correto, convertendo-o para uma matriz unidimensional (ravel()). Isso garante que y seja consistente durante a divisão dos dados e o treinamento do modelo.

Normalização dos Dados:

Usamos StandardScaler para normalizar os dados de entrada (X). Isso é importante para algoritmos como o MLPRegressor, que são sensíveis à escala dos dados. A normalização ajuda a acelerar a convergência e a melhorar o desempenho do modelo.

Divisão Adequada dos Dados:

Dividimos os dados em conjuntos de treino, validação e teste, garantindo que as proporções fossem mantidas corretamente. Isso ajuda a validar o modelo de forma mais robusta e a evitar problemas de overfitting e underfitting. Busca de Hiperparâmetros com Grid Search.

Utilizamos GridSearchCV para buscar os melhores hiperparâmetros para o MLPRegressor. A busca incluiu variações no tamanho das camadas escondidas, funções de ativação, solver, regularização (alpha) e taxa de aprendizado. Testamos diferentes combinações de hiperparâmetros para encontrar a configuração que maximiza o desempenho do modelo.

Aumento do Número de Iterações:

Aumentamos o número máximo de iterações para 1000. Isso permite que o modelo tenha mais tempo para convergir e encontrar uma solução melhor durante o treinamento.

```
import pandas as pd
from ucimlrepo import fetch_ucirepo
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Fetch dataset
auction_verification = fetch_ucirepo(id=713)

# Data (as pandas dataframes)
X = auction_verification.data.features
y = auction_verification.data.targets

# Convert to DataFrame for ease of use
X_df = pd.DataFrame(X)
```

```
y_df = pd.DataFrame(y)

# Check for missing values
print(X_df.isnull().sum())
print(y_df.isnull().sum())

# Information about the dataset
print(X_df.info())
print(y_df.info())

# Descriptive statistics
print(X_df.describe())
print(y_df.describe())

# Selected features based on exploratory analysis
selected_features = [
    'process.b1.capacity',
    'process.b2.capacity',
    'process.b3.capacity',
    'process.b4.capacity',
    'property.price',
    'property.product',
    'property.winner'
]

# Select only the chosen features
X_selected = X_df[selected_features]

# Normalize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_selected)

# Ensure y is a single column and convert to 1D array if needed
if y_df.shape[1] > 1:
    y_df = y_df.iloc[:, 0]
y = y_df.values.ravel()

# Split the data ensuring y has the correct shape
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y,
test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)

# Verify the sizes of the datasets after splitting
print(f'Tamanho de X_train: {X_train.shape}')
print(f'Tamanho de y_train: {y_train.shape}')
print(f'Tamanho de X_val: {X_val.shape}')
print(f'Tamanho de y_val: {y_val.shape}')
print(f'Tamanho de X_test: {X_test.shape}')
```

```
print(f'Tamanho de y_test: {y_test.shape}')

# Define and train the model
mlp = MLPRegressor(random_state=42, max_iter=1000)
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 100)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam', 'lbfgs'],
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate': ['constant', 'adaptive']
}

grid_search = GridSearchCV(mlp, param_grid, cv=5,
scoring='neg_mean_squared_error', verbose=1)
grid_search.fit(X_train, y_train)

# Best hyperparameters
print("Melhores Hiperparâmetros:", grid_search.best_params_)

# Evaluation on the validation set with the best model
best_mlp = grid_search.best_estimator_
y_val_pred = best_mlp.predict(X_val)
print(f"Validação MSE: {mean_squared_error(y_val, y_val_pred)}")
print(f"Validação R^2: {r2_score(y_val, y_val_pred)}")

# Evaluation on the test set
y_test_pred = best_mlp.predict(X_test)
print(f"Teste MSE: {mean_squared_error(y_test, y_test_pred)}")
print(f"Teste R^2: {r2_score(y_test, y_test_pred)}")
```

Melhores Hiperparâmetros: {'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (100, 100), 'learning_rate': 'constant', 'solver': 'lbfgs'}

Validação MSE: 0.019618467759117394

Validação R^2: 0.7776560362321872

Teste MSE: 0.03163608789420352

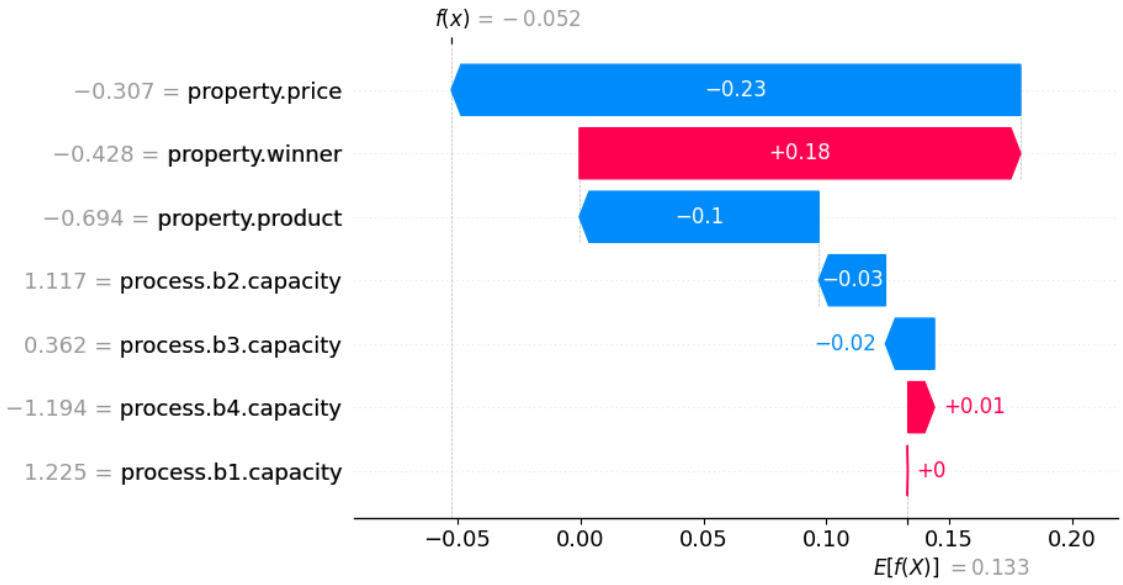
Teste R^2: 0.7633221637285216

Os valores de R^2 próximos entre o conjunto de validação e o conjunto de teste indicam que o modelo está bem ajustado e generaliza bem para dados não vistos. O R^2 em torno de 0.76-0.78 é um bom indicador de desempenho, mostrando que o modelo explica uma grande parte da variância na variável alvo.

4. Interpretando modelo tipo “Caixa preta”

4.1 Use a biblioteca SHAP com ferramenta para interpretação do seu modelo.

a) Construa um gráfico de “cascata” (waterfall). Interprete os resultados.



O gráfico de cascata (waterfall) mostra a decomposição da previsão do modelo para uma instância específica. No gráfico, cada barra representa a contribuição de uma característica individual para a previsão final do modelo. As barras podem ser positivas (em vermelho) ou negativas (em azul), indicando o impacto positivo ou negativo na previsão.

Valores das Características**:** Os valores das características para a instância específica estão listados ao lado esquerdo de cada barra.

Barras de Contribuição**:** Cada barra representa o valor SHAP de uma característica, mostrando como essa característica impactou a previsão. Barras em vermelho indicam um impacto positivo (aumentando a previsão), enquanto barras em azul indicam um impacto negativo (diminuindo a previsão).

Predição do Modelo (f(x))**:** A previsão do modelo para essa instância é mostrada no topo do gráfico (f(x) = -0.052). Este é o valor final predito pelo modelo após considerar todas as contribuições das características.

Valor Esperado (E[f(X)])**:** O valor esperado da previsão do modelo (média de todas as previsões) está mostrado no eixo X (E[f(X)] = 0.133).

Interpretação Específica do Gráfico Waterfall:

property.price: Tem um valor de -0.307 e contribui com -0.23 para a previsão final. Isso significa que o preço da propriedade está diminuindo a previsão em 0.23 unidades.

property.winner: Tem um valor de -0.428 e contribui com +0.18 para a previsão final. Isso significa que a variável winner está aumentando a previsão em 0.18 unidades.

property.product: Tem um valor de -0.694 e contribui com -0.1 para a previsão final. Isso indica uma redução na previsão devido a essa característica.

process.b2.capacity: Tem um valor de 1.117 e contribui com -0.03 para a previsão final, indicando um pequeno impacto negativo.

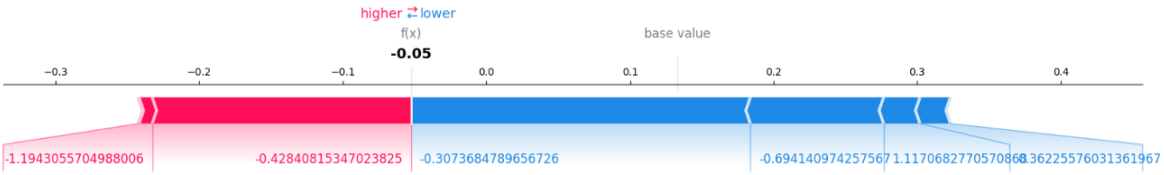
process.b3.capacity: Tem um valor de 0.362 e contribui com -0.02 para a previsão final, indicando outro pequeno impacto negativo.

process.b4.capacity: Tem um valor de -1.194 e contribui com +0.01 para a previsão final, indicando um pequeno impacto positivo.

process.b1.capacity: Tem um valor de 1.225 e praticamente não impacta a previsão final.

A soma dessas contribuições, junto com o valor esperado, resulta na previsão final do modelo para essa instância específica.

b) Construa um gráfico de "força" (force). Interprete os resultados.



O gráfico de força (force plot) é uma visualização que mostra a contribuição de cada característica para a previsão do modelo para uma instância específica. Ele é similar ao gráfico de cascata, mas apresenta as contribuições de uma maneira linear e acumulativa, o que pode facilitar a interpretação do impacto total de cada característica.

Componentes do Gráfico:

Base Value: O valor esperado do modelo para todas as instâncias, também conhecido como valor médio da previsão. No gráfico, este valor é mostrado como uma linha vertical rotulada como "base value".

f(x): A previsão do modelo para a instância específica. Este valor é mostrado no centro do gráfico, e no caso deste gráfico, é -0.05.

Características e Contribuições: Cada característica é representada por uma faixa colorida que se estende para a esquerda (vermelho, indicando redução) ou para a direita (azul, indicando aumento) do valor base. A largura de cada faixa representa a magnitude da contribuição da característica.

Interpretação Específica:

property.price: Tem uma contribuição de -0.307, o que reduz a previsão em 0.307 unidades.

property.winner: Contribui com -0.428, também reduzindo a previsão.

property.product: Contribui com -0.694, causando uma redução adicional.

process.b2.capacity: Contribui com +1.117, aumentando a previsão.

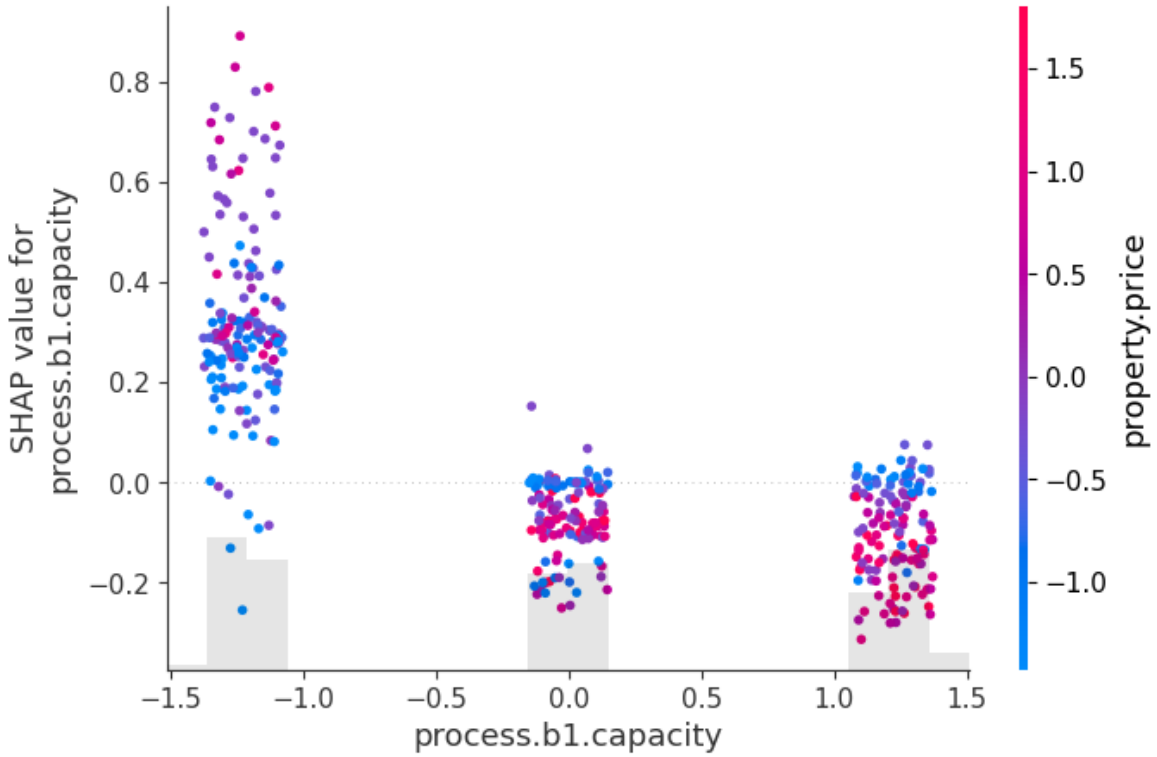
process.b3.capacity: Contribui com +0.362, também aumentando a previsão.

process.b4.capacity: Contribui com -1.194, causando uma redução.

process.b1.capacity: Contribui com +1.225, aumentando a previsão.

Este gráfico é útil para entender como cada característica individualmente impacta a previsão do modelo e ajuda a identificar quais características estão puxando a previsão para cima ou para baixo.

c) Construa um gráfico de "disperção" (scatter). Interprete os resultados.



O gráfico de dispersão (scatter plot) dos valores SHAP ajuda a visualizar a relação entre uma característica específica e seu impacto nas previsões do modelo. No gráfico, os

pontos representam instâncias do conjunto de dados, com a posição no eixo horizontal representando o valor da característica e a posição no eixo vertical representando o valor SHAP (contribuição) dessa característica para a previsão.

Componentes do Gráfico:

Eixo X (process.b1.capacity): Representa os valores da característica process.b1.capacity para as instâncias no conjunto de dados.

Eixo Y (SHAP value for process.b1.capacity): Representa os valores SHAP, ou seja, a contribuição de process.b1.capacity para a previsão do modelo.

Coloração dos Pontos: A cor dos pontos representa os valores de outra característica, neste caso, property.price. A escala de cores vai do azul (valores baixos) ao vermelho (valores altos).

Interpretação Específica:

1. Relação entre process.b1.capacity e sua Contribuição (SHAP Value):

Para valores negativos de process.b1.capacity (aproximadamente -1.5 a -1), os valores SHAP são predominantemente positivos, indicando que esses valores aumentam a previsão do modelo.

Para valores próximos a zero, os valores SHAP estão concentrados em torno de zero, indicando pouco ou nenhum impacto na previsão.

Para valores positivos de process.b1.capacity (aproximadamente 0.5 a 1.5), os valores SHAP são novamente positivos, sugerindo um impacto positivo na previsão.

2. Influência de property.price:

Os pontos coloridos em vermelho (valores altos de property.price) estão concentrados nos valores positivos de process.b1.capacity, indicando uma possível correlação entre essas duas características.

Os pontos coloridos em azul (valores baixos de property.price) estão mais dispersos, sugerindo menor correlação com process.b1.capacity.

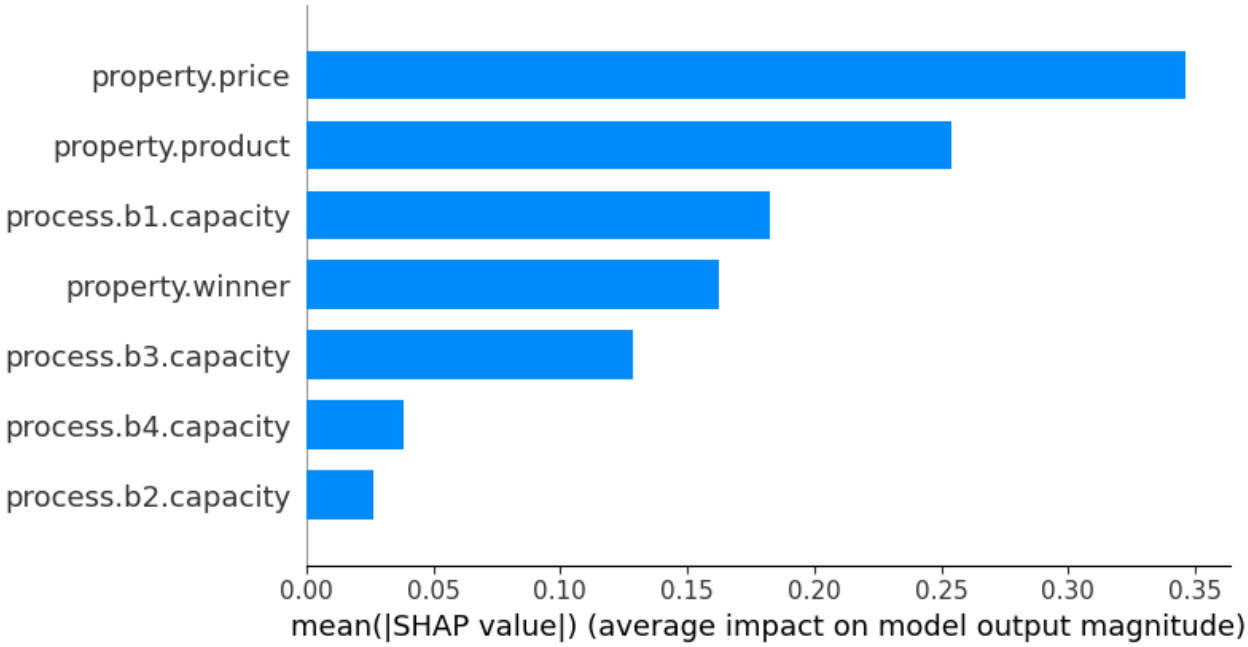
Conclusões

A característica process.b1.capacity tende a ter um impacto positivo na previsão do modelo quando seus valores são significativamente diferentes de zero (tanto positivos quanto negativos).

Os valores altos de property.price estão associados a valores positivos de process.b1.capacity, indicando que essas duas características podem estar correlacionadas e influenciam conjuntamente as previsões do modelo.

Valores próximos a zero de process.b1.capacity têm pouco impacto nas previsões, conforme indicado pelos valores SHAP próximos a zero.

d) Construa um gráfico de "barras" (bar). Interprete os resultados.



Eixo Y (Características): Lista as características do conjunto de dados que foram usadas pelo modelo para fazer previsões.

Eixo X (mean(|SHAP value|)): Representa o valor médio absoluto dos valores SHAP para cada característica, indicando a magnitude média do impacto de cada característica na previsão do modelo.

Interpretação Específica:

property.price: É a característica mais importante, com um impacto médio de aproximadamente 0.35 na previsão do modelo. Isso indica que o preço da propriedade tem a maior influência nas previsões.

property.product: Tem um impacto significativo, com um valor médio de aproximadamente 0.25. Isso sugere que o tipo de produto da propriedade também é um fator importante nas previsões.

process.b1.capacity: Com um impacto médio de aproximadamente 0.20, esta característica também contribui significativamente para as previsões do modelo.

property.winner: Apresenta um impacto médio de aproximadamente 0.15, indicando que esta característica também é relevante para as previsões.

process.b3.capacity: Tem um impacto médio menor, mas ainda significativo, de aproximadamente 0.10.

process.b4.capacity e process.b2.capacity: Têm os menores impactos médios, com valores próximos a 0.05. Embora tenham impacto menor em comparação com outras características, ainda são relevantes para o modelo.

Conclusões

As características property.price, property.product e process.b1.capacity são as mais influentes nas previsões do modelo, indicando que variações nesses valores têm um grande impacto nos resultados preditivos.

Características como property.winner e process.b3.capacity também são importantes, mas com impacto menor em comparação com as principais características.

Características como process.b4.capacity e process.b2.capacity têm um impacto menor, mas ainda contribuem para as previsões do modelo.

Este gráfico é útil para identificar quais características o modelo considera mais importantes e como elas influenciam as previsões. Isso pode ajudar a focar em áreas específicas para melhorar o desempenho do modelo ou para entender melhor o fenômeno sendo modelado.