Introdução ao Machine Learning Supervisionado

Marcos Eustorgio Filho

Resumo

Neste material são comparadas técnicas de Machine Learning supervisionado em relação ao desempenho em situações de classificação e regressão. O objetivo neste projeto é prever o valor de uma (ou mais) variável de saída (variável resposta ou target) baseado em uma quantidade observável de variáveis de entrada (covariáveis ou features). O conhecimento das observações na variável resposta funciona como um "supervisionador".

Para os contextos de classificação e regressão foram utilizados, respectivamente, os dois conjuntos de dados **Banknote Authentication** e **Boston**:

- **Banknote Authentication**: Apresenta informações sobre 1372 notas genuínas e falsificadas, em que os métodos trataram de classificar as notas de acordo com as *features*.
- **Boston**: Conjunto de dados também conhecido como o conjunto de dados Boston Housing, que contém valores de habitação e outras informações sobre os subúrbios de Boston, Massachusetts, EUA (n = 506), em que o objetivo é prever o valor médio das casas.

Informações adicionais:

Buscando obter-se boas previsões a partir do equilíbrio (trade-off) entre vício e variância, em ambos contextos foi utilizado o procedimento de *hold-out* dividindo randomicamente as bases de dados nas proporções de 70% para treinamento e 30% para teste, para que se possa validar os modelos. Para fins de replicabilidade foi escolhido o número 321 como seed, antes da divisão de ambos conjuntos de dados. Os dados utilizados nestes exemplos estão disponíveis no seguinte **repositório github**.

Nomenclaturas utilizadas

É crucial para profissionais das áreas de estatística/data analytics e data science/machine learning entenderem as diferenças e semelhanças nas nomenclaturas utilizadas em seus campos. Embora ambos os campos compartilhem muitos conceitos e técnicas, as palavras usadas para descrevê-los podem variar, levando a possíveis confusões e mal-entendidos.

Por exemplo, em Aprendizado de Máquina, termos como "rede" e "grafo" são frequentemente utilizados para descrever estruturas de dados complexas, enquanto em Estatística, "modelo" é mais comumente associado a uma representação matemática de um processo ou fenômeno. Além disso, em Aprendizado de Máquina, "pesos" são atribuídos às conexões entre unidades em uma rede neural, enquanto em Estatística, "parâmetros" são valores desconhecidos que precisam ser estimados a partir dos dados.

Portanto, é fundamental para profissionais de ambas as áreas estarem cientes dessas diferenças de nomenclatura para evitar mal-entendidos e garantir uma comunicação clara e eficaz. Ao reconhecer que termos aparentemente semelhantes podem ter significados distintos em diferentes contextos, os profissionais podem colaborar de forma mais eficiente e aproveitar ao máximo as ferramentas e técnicas disponíveis em ambas as áreas.

A seguir é apresentada a Tabela 1, contendo as correspondências entre os principais termos utilizados:

Tabela 1: Tabela com comparativo de nomenclaturas utilizadas

Data Science / Aprendizado de Máquina	**Estatística / Data Analytics**
Rede, Grafo (*network, graph*)	Modelo (*model*)
Pesos (*weights*)	Parâmetros (*parameters*)
Atributos/Características (*features/inputs*)	Covariáveis (*covariates*)
Saída (*target/output*)	Variável Resposta (*response variable*)
Aprendizado (*learning*)	Ajuste (*fitting*)
Generalização (*generalization*)	Avaliação em amostra teste (*performance test set*)
Aprendizado supervisionado (*supervised learning*)	Regressão, classificação (*regression, classification*)
Aprendizado não-supervisionado (*unsupervised learning*)	Estimação de densidade, agrupamento (*density estimation, clustering*)

Metodologia

Para realizar tal analise foram considerados os modelos listados a seguir, de acordo com a plausibilidade de aplicação a estrutura dos dados. A avaliação da adequabilidade de cada um dos modelos é feita utilizando métricas de ajuste apropriadas. Referências técnicas sobre estes modelos podem ser encontradas nos trabalhos de JAMES, WITTEN, HASTIE & TIBSHIRANI (2021) e IZBICK, SANTOS (2020). Além disto, referências sobre a implementação dos métodos e métricas abordados podem ser encontradas em *Scikit-Learn 3.3 - Metrics and scoring: quantifying the quality of predictions*.

Modelos de classificação abordados

- **Modelo logístico**: é um modelo de regressão usado para modelar a chance de uma determinada classe ou evento ocorrer, com base em variáveis independentes. É comumente usado em problemas de classificação binária, onde o objetivo é prever se um exemplo pertence a uma das duas classes possíveis.
- **KNN – Classificação**: é um método de aprendizado supervisionado usado para classificação e regressão. O algoritmo identifica os K pontos de dados mais próximos do ponto a ser classificado com base na medida de distância escolhida. Para classificação, o algoritmo atribui ao ponto a classe mais frequente entre seus vizinhos mais próximos (ou seja, a classe predominante entre os K vizinhos).
- **Gaussian Naive Bayes (GNB)**: é um algoritmo de classificação probabilístico que se baseia no teorema de Bayes. O modelo Naive Bayes Gaussiano é uma variação do algoritmo Naive Bayes que assume que as características são continuamente distribuídas de acordo com uma distribuição normal (Gaussiana). Este modelo é comumente usado em problemas de classificação onde as características são contínuas e assumem uma distribuição gaussiana.
- **Árvore de decisão**: modela a relação entre uma variável de saída (ou resposta) e um conjunto de variáveis de entrada dividindo iterativamente o conjunto de dados em subconjuntos menores com base nas características que melhor separam os valores da variável de saída. Ela procura a melhor maneira de dividir os dados para que as classes sejam o mais homogêneas possível dentro de cada subconjunto.

Modelos de regressão abordados

- **Regressão linear múltipla**: A regressão linear é um método estatístico que é usado para modelar a relação entre uma variável dependente (ou resposta) e uma ou mais variáveis independentes (ou preditoras) de forma linear. Uma vez que os coeficientes são estimados, o modelo pode ser usado para prever o impacto na média da variável dependente de acordo com os valores das variáveis independentes. É o modelo de regressão mais popular e possui vantagens como: fácil implementação e verificação de adequabilidade, além da clara interpretabilidade dos coeficientes.
- **KNN – Regressão**: o KNN para regressão calcula a média (ou mediana) dos valores alvo dos K vizinhos mais próximos. O valor médio (ou mediano) calculado é então atribuído como a previsão do valor alvo para o ponto de dados em questão.

- **Árvore de regressão**: modela a relação entre uma variável de saída (ou resposta) e um conjunto de variáveis de entrada dividindo iterativamente o conjunto de dados em subconjuntos menores, escolhendo a característica e o valor de corte que resultam na maior redução na variabilidade da variável de saída nos subconjuntos resultantes.

Métricas para classificação

Como temos exemplos em que os desfechos de classificação são binários, um procedimento tradicional para avaliar a capacidade preditiva dos métodos de classificação binária é construir uma matriz de confusão, em que:

- **VP** representa o número de verdadeiros positivos;
- **FP** representa o número de falsos positivos;
- **FN** representa o número de falsos negativos;
- **VN** representa o número de verdadeiros negativos.

Real/ Predito	$\hat{Y}$=0	$\hat{Y}$=1
Y=0	VN	FP
Y=1	FN	VP

Assim, a partir desses números podem ser definidas algumas métricas de ajuste para modelos de classificação.

Acurácia (ACC)

A acurácia é a fração de predições corretas (ou acertos) de um modelo, para a classificação de indivíduos das classes 0 e 1.

$$ACC = \frac{VP + VN}{VP + VN + FP + FN}$$

Apesar da ACC ser ainda a medida mais utilizada para a comparação entre classificadores, ela não é adequada quando existem classes desbalanceadas, ou seja, quando existe uma frequência de observações muito alta em uma classe ou categoria da variável resposta, enquanto que as demais classes apresentam frequência baixa. Exemplo: 97% dos pacientes de um estudo apresentaram determinado sintoma, enquanto que 3% não apresentaram.

Coeficiente de Correlação de Matthews (MCC)

Medida usualmente utilizada para interpretar a classificação geral do modelo, com interpretação é semelhante ao coeficiente de correlação de Pearson, coeficiente estatístico que varia no intervalo de [-1;1] e serve para qantificar a magnetude de uma relação linear entre duas variáveis numéricas. Desta forma temos a seguinte interpretação para os valores do MCC: classificação perfeita (+1); quando igual a 0, a classificação é completamente aleatória; e classificação completamente inversa (-1).

$$MCC = \frac{VP \times VN - FP \times FN}{\sqrt{(VP + FP)(VP + FN)(VN + FP)(VN + FN)}}$$

Sensibilidade ou *Recall* (SEN)

É a fração dos indivíduos que o modelo classificou corretamente para a classe 1 (classe de interesse ou de ocorrência de evento), dentre todos os indivíduos pertencentes à classe 1. A SEN também é conhecida como recall (ou revocação).

$$SEN = P(\hat{Y} = 1|Y = 1) = \frac{VP}{VP + FN}$$

Valor Preditivo Positivo ou *Precision* (VPP)

É a fração dos indivíduos pertencentes à classe 1, dentre todos os indivíduos que o modelo classificou para a classe 1. O VPP também é conhecido como precision (ou precisão).

$$VPP = P(Y = 1|\hat{Y} = 1) = \frac{VP}{VP + FP}$$

F1 Score (F1)

Essa medida considera VPP e SEN (a média harmônica delas), e pode ser usada para avaliar o desempenho geral do classificador, em vez de ter que avaliar os trade-offs entre VPP e SEN. O F1 Score varia de 0 a 1, onde 1 indica um modelo perfeito que alcança tanto alta precisão quanto alto recall. Modelos com valor abaixo de 0.5 podem ser considerados ruins.

$$F_1 = 2 \times \frac{VPP \times SEN}{VPP + SEN}$$

Também é uma métrica útil para avaliar o desempenho de modelos de classificação, especialmente em situações de desbalanceamento de classes, onde a precisão e o recall podem dar uma visão limitada do desempenho do modelo. Ao lidar com classificação multiclasse, existem variações para o cálculo do escore F1, gerando escores médios (macro, ponderado, micro) que podem ser utilizados de acordo com o problema de classificação.

Brier Score (BS)

Originalmente proposto por Brier (1950), serve para medir a precisão das previsões probabilísticas.

$$BS = \frac{1}{n}\sum_{t=1}^{n}(p_t - o_t)^2,$$

em que, n é o número de instâncias, p_t é a probabilidade prevista da t-ésima instância pertencer à classe 1, e o_t é 1 se a classe atual y_t é igual a 1, e 0 caso contrário. $BS \in [0, 1]$, sendo que o mínimo que se espera de um bom modelo é que $BS < 0.25$.

Métricas para regressão

Coeficiente de determinação R^2

Também conhecido como coeficiente de determinação, esta métrica representa o percentual da variância dos dados que é explicada pelo modelo. Os valores possíveis variam de 0 a 1, podendo também ser expressos em termos percentuais.Quanto maior é o valor de R^2, maior é o percentual da variância da variável dependente (target) explicada através das covariáveis (features).

$$R^2 = 1 - \frac{SQ_{res}}{SQ_{tot}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

em que n é o número de instâncias, y_i é o valor real da i-ésima instância, $\hat{y}_i$ é o valor predito para i-ésima instância e $\bar{y}$ é a média da variável dependente. Embora este coeficiente seja amplamente utilizado para comparação entre modelos, a interpretação usual desta métrica só em válida no contexto de modelos de regressão linear.

R^2 ajustado

Uma outra questão envolvendo o coeficiente de determinação é que o aumento da complexidade do modelo pode inflacionar o valor de R^2, mesmo que as novas *features* incluidas sejam pouco explicativas em relação a variável dependente.

$$\bar{R}^2 = \frac{n-1}{n-(k+1)}(1 - R^2),$$

em que, k representa o número de *features* Essa formulação para o coeficiente de determinação penaliza a inclusão de *features* pouco explicativas.

Erro Médio Absoluto

O erro médio absoluto (MAE) mensura a diferença absoluta média entre o valor real y_i e o predito $\hat{y}_i$. Seu valor significa o quanto, em média, o valor predito se afasta do valor verdadeiro, de foma positiva ou negativa.

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Valores de MAE mais baixos indicam que as previsões do modelo estão mais próximas dos valores reais, enquanto valores mais altos indicam maior desvio entre as previsões e os valores reais. O MAE é robusto a *outliers*, pois não penaliza erros grandes de forma mais significativa, como o RMSE.

Erro Percentual Absoluto Médio

O erro percentual absoluto médio (MAPE) calcula a média dos erros percentuais absolutos entre as previsões e os valores reais, expressos comumente como uma porcentagem do valor real.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{max(\epsilon, |y_i|)}$$

como esta métrica é interpretada de forma percentual, o seu resultado significa que o modelo faz previsões de forma que, em média, a diferença entre o valor real y_i e o predito $\hat{y}_i$ equivale a $MAPE\%$ do valor verdadeiro. Para evitar devisões por 0 é adotado um valor muito próximo de zero porém não nulo para a constante $\epsilon \approx 2.221e^{-16}$ que é o valor adotado pela biblioteca scikit-learn do Python.

Raiz do Erro Quadrático Médio

A raiz do erro quadrático médio(RMSE) é uma medida de avaliação comumente usada para avaliar a precisão de modelos de previsão, especialmente em problemas de regressão. Ela mede a média da diferença entre as previsões do modelo e os valores reais, elevada ao quadrado, e então tira a raiz quadrada desse valor para retornar à escala original dos dados.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}$$

Valores de RMSE mais baixos indicam que as previsões do modelo estão mais próximas dos valores reais, enquanto valores mais altos indicam maior desvio entre as previsões e os valores reais. O RMSE é sensível a outliers, ou seja, valores extremos podem ter um impacto significativo no resultado.

Resultados

Os resultados apresentados a seguir foram desenvolvidos utilizando linguagem computacional Python, com auxilio das bibliotecas `Pandas`, `Seaborn` e `Scikit-Learn`, e estão disponíveis no **repositório github** em formato notebook markdown e HTML.

Inicialmente vamos carregar os pacotes necessários para poder realizar a importação e manipulação dos dados.

```
#Importando pacotes necessarios
import pandas as pd
import numpy as np
import sklearn as skl
import seaborn as sns
import matplotlib.pyplot as plt
```

O pacote `Pandas` é responsável pela implementação dos métodos e funções necessárias para leitura e manipulação das bases de dados em Python. O pacote `numpy` é utilizado para operações matemáticas, possuindo diversas funções programadas para estas operações. Os pacotes `matplotlib` e `seaborn` são úteis para criação de gráficos e figuras. Já o `sklearn`, conhecido como Scikit-Learn, é o pacote em Python que apresenta funções para as implementações para diversos modelos de *Machine Learning*, além das respectivas métricas de avaliação.

Dados Banknote Authentication

Análise descritiva e exploratória

Inicialmente vamos realizar a importação dos dados para dar inicio a análise descritiva e exploratória.

```
#Importando dados
Banknote=pd.read_csv('data_banknote_authentication.txt', sep=',',header=None)

#Renomeando colunas
Banknote.columns=['X1', 'X2', 'X3', 'X4', 'Y']

#Visualização das linhas iniciais
Banknote.head()
```

```
##         X1      X2      X3       X4  Y
## 0  3.62160  8.6661 -2.8073 -0.44699  0
## 1  4.54590  8.1674 -2.4586 -1.46210  0
## 2  3.86600 -2.6383  1.9242  0.10645  0
## 3  3.45660  9.5228 -4.0112 -3.59440  0
## 4  0.32924 -4.4552  4.5718 -0.98880  0
```

Esta é uma base de dados em que as *features* são numéricas (contínuas), enquanto o desfecho assume apenas dois valores, 0 ou 1. Como tem-se *features* numéricas é possível obter a tabela de medidas resumo para as mesmas, através da combinação dos métodos `.loc[]`, utilizado para selecionar apenas as features e `.describe()` para obter as medidas resumo das colunas selecionadas.

```
#Medidas resumo para as features
Banknote.loc[:,'X1':'X4'].describe().T
```

```
##       count      mean       std      min       25%      50%       75%      max
## X1  1372.0  0.433735  2.842763  -7.0421 -1.773000  0.49618  2.821475   6.8248
## X2  1372.0  1.922353  5.869047 -13.7731 -1.708200  2.31965  6.814625  12.9516
## X3  1372.0  1.397627  4.310030  -5.2861 -1.574975  0.61663  3.179250  17.9274
## X4  1372.0 -1.191657  2.101013  -8.5482 -2.413450 -0.58665  0.394810   2.4495
```

A partir dos resultados provenientes do `.describe()`, que são as medidas resumo, pode-se perceber que as *features* `X4` e `X1` apresentam as menores médias se comparadas com as demais. Em termos de variabilidade `X2` e `X3` apresentaram os maiores valores para o desvio padrão e amplitude. Comparando os valores de média e mediana é possível perceber a ocorrência de assimetria na distribuição amostral das *features* `X3` e `X4`.

Após análise das features podemos também contabilizar o percentual de observações em cada categoria do desfecho (*target*) combinando os métodos `.value_conts()` para contabilizar a frequência de cada categoria em `Y` e o método `.shape[]` que ao ser aplicado com valor zero retorna o número de linhas da base de dados.

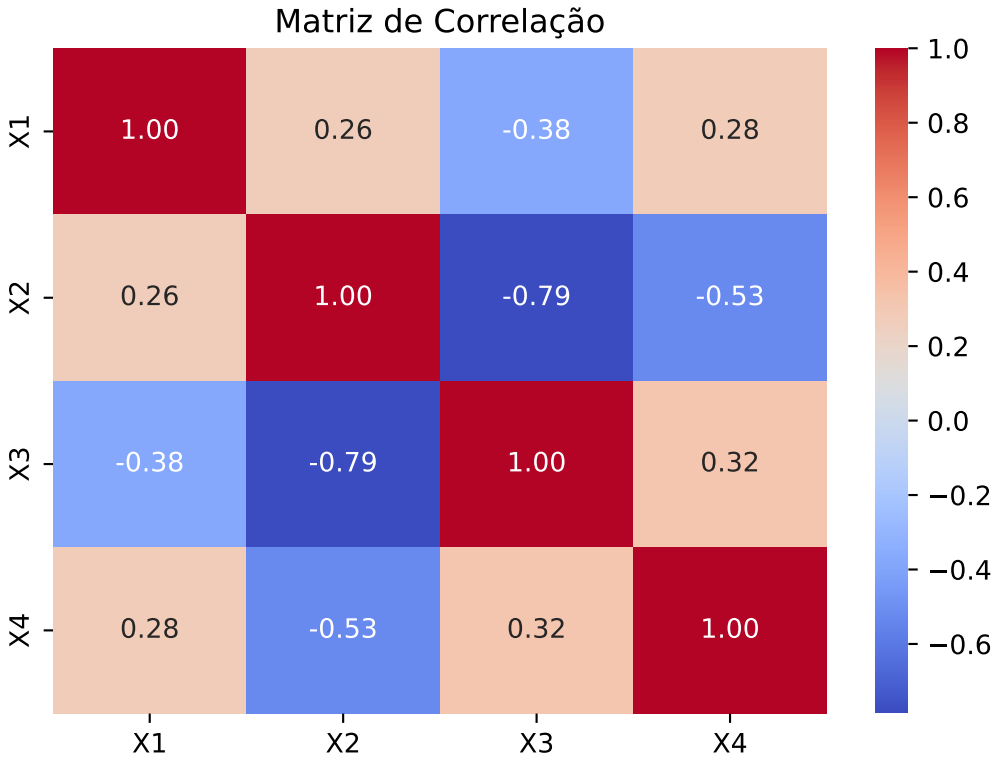
```
#Obtendo proporcao de valores para cada categoria do target
(Banknote['Y'].value_counts()/Banknote.shape[0]).round(2)*100
```

```
## Y
## 0    56.0
## 1    44.0
## Name: count, dtype: float64
```

A categoria 0 apresenta percentual de 56% enquanto a categoria 1 tem 44%. Esta é uma situação onde temos classes balanceadas, o que é um ponto positivo em problema de classificação, pois alguns modelos tendem a apresentar problemas mediante a ocorrência de classes desbalanceadas.

A seguir também é interessante estudar se, e como, as *features* estão correlacionadas umas as outras, visto que, uma forte correlação entre as *features* gera um problema conhecido como multicolinearidade, que é prejudicial para o ajuste de alguns modelos, como a regressão logística e o modelo naive bayes gaussiano.

```
#Gráfico da matriz de correlação
sns.heatmap(Banknote.loc[:,'X1':'X4'].corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Matriz de Correlação')
plt.show()
```



Com excessão do par `X2` e `X3` que apresentou forte correlação negativa, as demais *features* apresentaram correlações baixas ou moderadas entre si considerando todo conjunto de dados. Contudo também pode ser de interesse investigar se estas correlações mudam se comparadas entre as observações de cada classe.

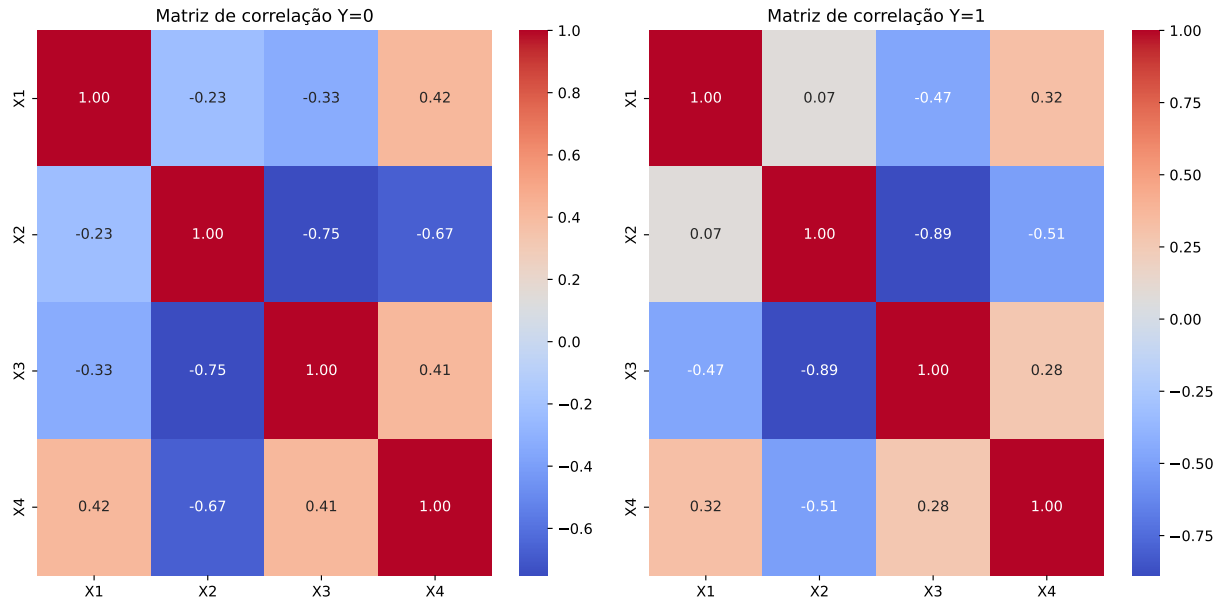
```
#Investigando correlação em cada grupo de classificação
matriz_correlacao_0=Banknote.loc[Banknote['Y']==0].loc[:,'X1':'X4'].corr()
matriz_correlacao_1=Banknote.loc[Banknote['Y']==1].loc[:,'X1':'X4'].corr()


# Criando a figura e os subplots
fig, axs = plt.subplots(1, 2, figsize=(12, 6))

# Plotando o primeiro gráfico no primeiro subplot
sns.heatmap(matriz_correlacao_0, annot=True, cmap='coolwarm', fmt='.2f',ax=axs[0])
axs[0].set_title('Matriz de correlação Y=0')

# Plotando o segundo gráfico no segundo subplot
sns.heatmap(matriz_correlacao_1, annot=True, cmap='coolwarm', fmt='.2f',ax=axs[1])
axs[1].set_title('Matriz de correlação Y=1')

#Evitando sobreposição e printando
plt.tight_layout()
plt.show()
```

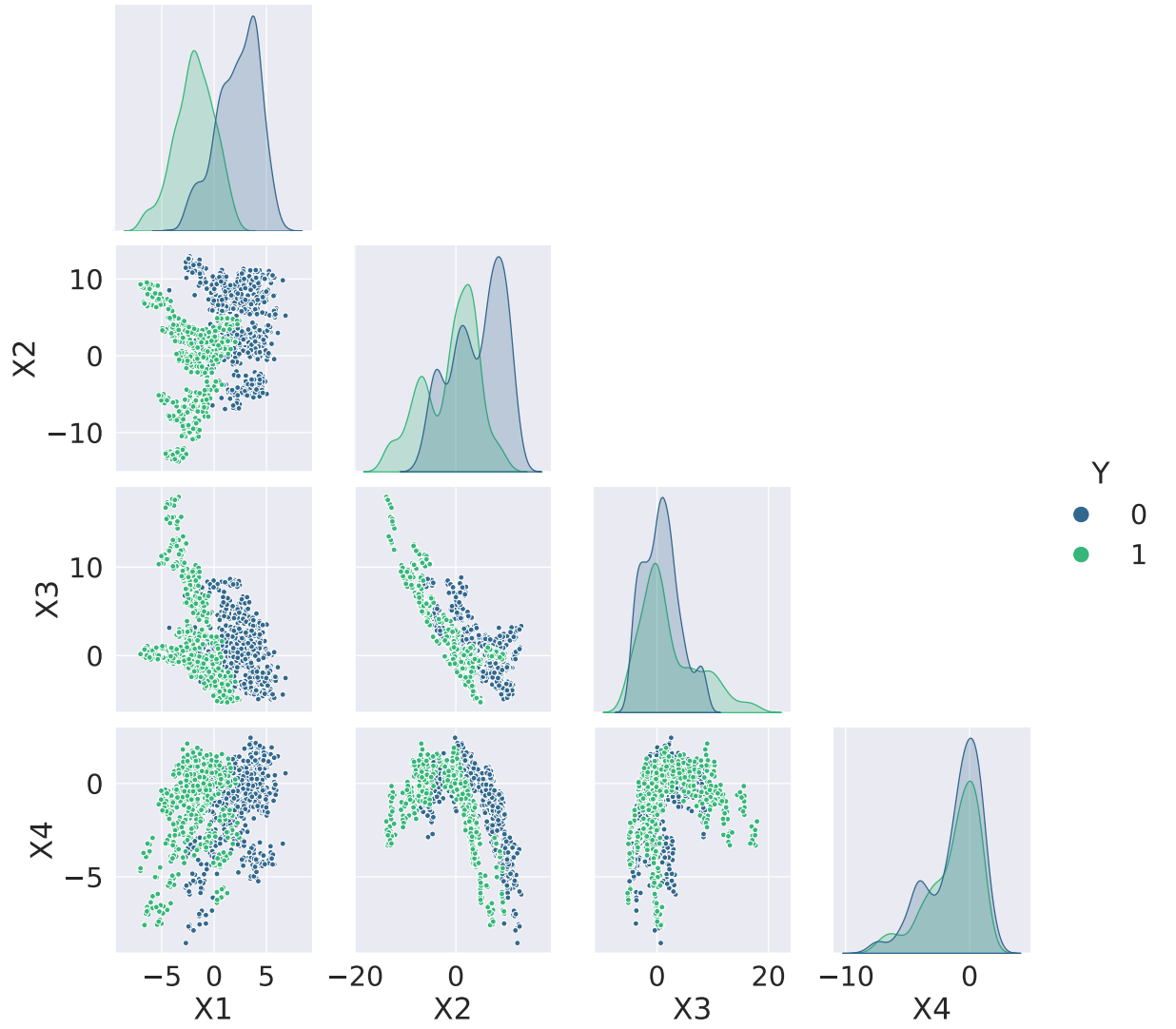


Ao dividir a matriz de correlação por classe de `Y` pode-se perceber que as *features* `X2` e `X4` apresentam forte correlação negativa na classe 0, enquanto que o mesmo não ocorre na classe 1. Já `X2` e `X3` continuam apresentando o mesmo tipo de correlação independente da classe. As demais features apresentam correlações baixas entre si em ambas as classes.

Neste problema de classificação temos *features* numéricas é possível plotar um painel contendo gráficos de dispersão, de acordo com as classes, e histogramas. Assim é possível visualizar como as observações estão relacionadas e podem ser separadas de acordo com `Y`.

```
# Setting the font size
sns.set(font_scale=2)
plt.rc('legend',markerscale=3.0)

# Pairplot usando seaborn
pairplot = ( sns.pairplot(Banknote, hue='Y', palette='viridis', markers=['o', 'o'],
plot_kws={'s': 20},diag_kind='kde', corner=True, height=3) )
plt.show(pairplot)
```



Ao avaliar o painel de diagramas de dispersão e histogramas, levando em consideração a classificação da nota, podemos constatar a evidência de assimetria na distribuição das *features* `X3` e `X4`, além de perceber que os pontos de cores diferentes apresentam menor separação linear nos diagramas de dispersão entre `X4` e demais *features*, o que pode ser uma evidência de que `X4` pode atrapalhar o desempenho de alguns classificadores, sobretudo os classificadores lineares.

A partir do painel de gráficos, podemos analisar a distribuição das *features* de acordo com boxplots em cada classe, assim podemos estudar características que auxiliem os classificadores, além de aumentar o conhecimento sobre as *features*.

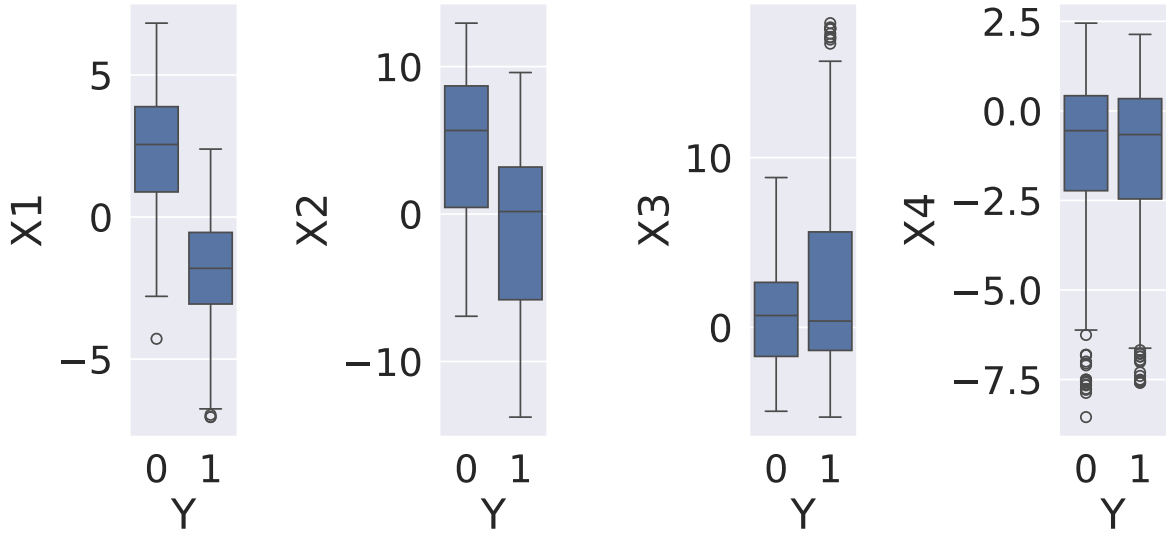
```
#Agora vamos plotar os boxplots para cada valor único de Y

# Criando um subplot com 1 linha e 4 colunas
fig, axs = plt.subplots(1, 4, figsize=(10, 5))

# Criando um boxplot para cada variável X, separados por valor de Y
```

```
for i, col in enumerate(['X1', 'X2', 'X3', 'X4']):
    sns.boxplot(x='Y', y=col, data=Banknote, ax=axs[i])
    plt.tight_layout()

# Exibindo o gráfico
plt.show()
```



Analisando o boxplot de `X1` pode-se perceber que há evidências de que as observações da classe `Y=1` apresentam valores menores que as da outra classe, embora sejam notados alguns valores discrepantes. A distribuição de `X2` apresenta valores um pouco menores na classe 1, porém não tão distintos se comparados com os valores das observações da mesma *feature* na classe 0, e em ambos boxplots é possível perceber a ocorrência de assimetria negativa na distribuição de `X2`. A distribução de `X3` em relação as classes de `Y` não apresenta tanta diferença, embora a distribuição dos valores de `X3` na classe 1 apresentem valores um pouco maiores que da classe 0, tenham assimetria positiva e alguns *outliers*. A distribuição dos valores da *feature* `X4` tem forma semelhante entre as classes, apresentando assimetria negativa e *outliers* em ambas, não apresentando diferença perceptível dentre as classes.

Processo de divisão dos dados

Para que seja possível treinar e avaliar as métricas de ajuste e capacidade preditiva dos modelos, os conjuntos os dados Banknote foram segmentados na proporção de 70% para treinamento e 30% para teste, no processo que é conhecido como *hold-out*. Desta forma é possível avaliar o quão bem os modelos se saem na predição a partir de novas informações. Para divisão considerou-se seed 321, e a partir dela foram sorteadas linhas dos dados Banknote para compor o conjunto de dados de treinamento.

O método `train_test_split` foi importado do pacote Sciki-Learn para que fosse possível dividir os dados nos conjuntos de treino e teste. Além disto, do mesmo pacote também foram importadas algumas funções para calculo das métricas de ajuste do modelo que foram discutivas na metodologia deste relatório.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import (confusion_matrix, accuracy_score,matthews_corrcoef,
                             precision_score, recall_score, f1_score,brier_score_loss)

# Seu conjunto de dados
X = Banknote.loc[:,'X1':'X4']
y = Banknote['Y']


# Divisão em treino e teste com uma semente (seed)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=321)
```

Modelo logístico

O modelo logístico é um dos mais classicos modelos para desfecho (*target*) do tipo categórico. Ele modela o logarítmo natural da chance de ocorrência do evento, para o caso de resposta binária. Além de poder também ser utilizado para classificação ele tem a vantagem da interpretabilidade em relação aos outros modelos de classificação comumente utilizados. Uma outra característica deste modelo é que se as classes forem linearmente separáveis é esperado que a regressão logística apresente um excelente desempenho.

Para ajuste deste modelo será importada a função `LogisticRegression` do pacote Scikit-Learn.

```
from sklearn.linear_model import LogisticRegression

# Modelo de regressão logística
model = LogisticRegression()
reg_log = model.fit(X_train, y_train)

# Coeficientes do modelo
#print("Coeficientes:", reg_log.coef_)
#print("Intercept:", reg_log.intercept_)

# Desempenho preditivo (teste)
y_pred = reg_log.predict(X_test)
```

Após ajuste do modelo e armazenamento das classes preditas no objeto `y_pred` vamos obter a matriz de confusão, onde é possível ter uma ideia de como o modelo esta se saindo.

```
# Matriz de confusão
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

```
## [[237   4]
##  [  3 168]]
```

```
# Métricas de comparação
VN, FP, FN, VP = conf_matrix.ravel()
print({'VN': VN, 'FP': FP, 'FN': FN, 'VP':VP})
```

```
## {'VN': 237, 'FP': 4, 'FN': 3, 'VP': 168}
```

A partir desta matriz temos a informação de que o modelo após ser treinado conseguiu predizer corretamente `VN=237` observações do grupo 0, também conseguiu predizer `VP=168` informações corretamente do grupo 1, e errou a classificação para 7 (`4FP + 3FN`) obserações.

Com as funções importadas do pacote Scikit-Learn vamos calcular e armazenar as estatísticas de ajuste para o modelo.

```
ACC = accuracy_score(y_test, y_pred)
MCC = matthews_corrcoef(y_test, y_pred)
SEN = recall_score(y_test, y_pred)
VPP = precision_score(y_test, y_pred)
F1 = f1_score(y_test, y_pred)
BS = brier_score_loss(y_test, reg_log.predict_proba(X_test)[:, 1])

# Dicionario com métricas de comparacao para modelo logistico
rlog = {'ACC': ACC, 'MCC': MCC, 'SEN': SEN, 'VPP':VPP, 'F1': F1, 'BS': BS}
rlog = {key: round(value, 2) for key, value in rlog.items()}
print(rlog)
```

```
## {'ACC': 0.98, 'MCC': 0.97, 'SEN': 0.98, 'VPP': 0.98, 'F1': 0.98, 'BS': 0.01}
```

```
#Apagando objetos
del y_pred, conf_matrix, model
```

KNN - Classificação

Para ajuste do modelo KNN para classificação serão importadas as funções `KNeighborsClassifier` e `StandardScaler`. Esta última é responsável pelo processo de padronização das *features* que é recomendado ao utilizar este tipo de modelo uma vez que, características com diferentes escalas podem ter influências desproporcionais nos cálculos de distância no k-NN, além contribuir para melhora da eficiência computacional, especialmente em conjuntos de dados com muitas características, porque reduz a escala dos dados e pode levar a cálculos mais rápidos.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

# Padronizar as features (opcional, mas recomendado para o KNN)
scaler = StandardScaler()
X_train_sc = scaler.fit_transform(X_train)
X_test_sc = scaler.transform(X_test)
```

Note que, inicialmente as *features* para os dados de treino são padronizados e os parâmetros resultantes desta padronização são utilizados também para padronizar o conjunto de teste. Ao usar o `StandardScaler`, é calculada a média e o desvio padrão de cada coluna nos dados de treinamento separadamente e, em seguida, aplica a transformação de subtração da média e divisão pelo desvio padrão a cada coluna individualmente.

Segundo Sckit-Learn (data) sempre deve-se dividir os dados em subconjuntos de treinamento e teste primeiro, especialmente antes de qualquer etapa de pré-processamento. Nunca inclua dados de teste ao usar os métodos `fit` e `fit_transform`. Usar todos os dados, por exemplo, `fit(X)`, pode resultar em pontuações excessivamente otimistas. Por outro lado, o método `transform` deve ser usado em ambos os subconjuntos de treinamento e teste, pois o mesmo pré-processamento deve ser aplicado a todos os dados. Isso pode ser alcançado usando `fit_transform` no subconjunto de treinamento e `transform` no subconjunto de teste.

```
# Criar e treinar o modelo KNN
k = 10  # Número de vizinhos (Escolha inicial arbitraria)
model = KNeighborsClassifier(n_neighbors=k)
knn_c=model.fit(X_train_sc, y_train)
```

```
# Fazer previsões no conjunto de teste
y_pred = knn_c.predict(X_test_sc)

# Matriz de confusão
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

```
## [[240   1]
##  [  0 171]]
```

```
# Métricas de comparação
VN, FP, FN, VP = conf_matrix.ravel()
print({'VN': VN, 'FP': FP, 'FN': FN, 'VP':VP})
```

```
## {'VN': 240, 'FP': 1, 'FN': 0, 'VP': 171}
```

Após ajuste do modelo e armazenamento das classes preditas no objeto `y_pred` vamos obter a matriz de confusão, onde é possível ter uma ideia de como o modelo esta se saindo.

A partir desta matriz temos a informação de que o modelo após ser treinado conseguiu predizer corretamente `VN=240` observações do grupo 0, também conseguiu predizer `VP=171` informações corretamente do grupo 1, e errou apenas em uma observação `FP=1`.

É importante salientar que a escolha de $k = 10$ foi arbitrária apenas para fins ilustrativos, contudo, é de costume ao utilizar este tipo de modelo realizar o procedimento de validação cruzada *k-fold*. Neste método, os dados são divididos em k partes iguais (chamadas de *folds*), e o modelo é treinado k vezes, cada vez usando $k-1$ *folds* como conjunto de treinamento e 1 *fold* como conjunto de teste. Isso significa que cada fold é usado como conjunto de teste exatamente uma vez. Por fim é selecionado o valor de k que apresenta o melhor desempenho médio de acordo com a métrica escolhida.

```
# Métricas de comparação
ACC = accuracy_score(y_test, y_pred)
MCC = matthews_corrcoef(y_test, y_pred)
SEN = recall_score(y_test, y_pred)
VPP = precision_score(y_test, y_pred)
F1 = f1_score(y_test, y_pred)
BS = brier_score_loss(y_test, knn_c.predict_proba(X_test_sc)[:, 1])


# Dicionario com métricas de comparacao para modelo KNN - Classificacao
knnc = {'ACC': ACC, 'MCC': MCC, 'SEN': SEN, 'VPP':VPP, 'F1': F1, 'BS': BS}
knnc = {key: round(value, 2) for key, value in knnc.items()}
print(knnc)
```

```
## {'ACC': 1.0, 'MCC': 1.0, 'SEN': 1.0, 'VPP': 0.99, 'F1': 1.0, 'BS': 0.0}
```

Como este modelo apresentou apenas um erro de classificação considerando os dados de teste, ele apresenta ótimas métrica de ajuste.

Gaussian Naive Bayes (NB)

O modelo Gaussian Naive Bayes é uma técnica de classificação baseada no Teorema de Bayes, que assume que as features são independentes e seguem uma distribuição gaussiana (normal). Ele é especialmente útil para problemas de classificação com *features* numéricas, como é o caso dos dados Banknote. Contudo, para

estes dados não há evidências para corroborar os pressupostos de independência e normalidade esperados, o que pode atrabalhar o desempenho do método.

```
from sklearn.naive_bayes import GaussianNB

# Criar e treinar o modelo Gaussian Naive Bayes
model = GaussianNB()
gnbayes=model.fit(X_train, y_train)

# Fazer previsões no conjunto de teste
y_pred = gnbayes.predict(X_test)
```

Após ajuste do modelo podemos analisar a matriz de confusão associada ao mesmo.

```
# Matriz de confusão
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

```
## [[209  32]
##  [ 32 139]]
```

```
# Métricas de comparação
VN, FP, FN, VP = conf_matrix.ravel()
print({'VN': VN, 'FP': FP, 'FN': FN, 'VP':VP})
```

```
## {'VN': 209, 'FP': 32, 'FN': 32, 'VP': 139}
```

A partir desta matriz temos a informação de que o modelo após ser treinado conseguiu predizer corretamente `VN=209` observações do grupo 0, também conseguiu predizer `VP=139` informações corretamente do grupo 1, e errou a classificação para 64 (`32FP + 32FN`) obserações. A quantidade de observações preditas erroneamente neste método pode estar asociada a duas questões:

- Não verificação do pressuposto de normalidade das *features*;
- Não verificação do pressuposto de independência entre as *features*;

Desta forma é esperado que este método tenha um desempenho prejudicado devido estas questões.

Posteriormente são calculadas e armazenadas as métricas de ajuste para este modelo:

```
# Métricas de comparação
#VN, FP, FN, VP = conf_matrix.ravel()
ACC = accuracy_score(y_test, y_pred)
MCC = matthews_corrcoef(y_test, y_pred)
SEN = recall_score(y_test, y_pred)
VPP = precision_score(y_test, y_pred)
F1 = f1_score(y_test, y_pred)
BS = brier_score_loss(y_test, gnbayes.predict_proba(X_test)[:, 1])


# Dicionario com métricas de comparacao para modelo GNB - Classificacao
gnb = {'ACC': ACC, 'MCC': MCC, 'SEN': SEN, 'VPP':VPP, 'F1': F1, 'BS': BS}
gnb = {key: round(value, 2) for key, value in gnb.items()}
print(gnb)
```

```
## {'ACC': 0.84, 'MCC': 0.68, 'SEN': 0.81, 'VPP': 0.81, 'F1': 0.81, 'BS': 0.1}
```

Árvore de Decisão

Árvores de decisão são modelos de aprendizado de máquina que são populares por sua interpretabilidade e facilidade de visualização, pois podem ser representadas graficamente como uma árvore.

```
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Criar e treinar o modelo de árvore de decisão
model = DecisionTreeClassifier(criterion='entropy')
DTClass=model.fit(X_train, y_train)

# Fazer previsões no conjunto de teste
y_pred = DTClass.predict(X_test)
```

Após treino e cálculo das observações preditas, podemos obter a matriz de confusão para este modelo e armazenar as métricas de ajuste.

```
# Matriz de confusão
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

```
## [[239   2]
##  [  3 168]]
```

```

# Métricas de comparação
#VN, FP, FN, VP = conf_matrix.ravel()
ACC = accuracy_score(y_test, y_pred)
MCC = matthews_corrcoef(y_test, y_pred)
SEN = recall_score(y_test, y_pred)
VPP = precision_score(y_test, y_pred)
F1 = f1_score(y_test, y_pred)
BS = brier_score_loss(y_test, DTClass.predict_proba(X_test)[:, 1])
```

Uma grande vantagem deste tipo de modelo é poder visualizar graficamente a árvore de decisão, tendo uma informação adicional e de fácil interpretabilidade sobre o modelo.

```
# Plotar a árvore de decisão
plt.figure(figsize=(20, 20))  # Definir o tamanho da figura
plot_tree(DTClass, filled=True, feature_names=['X1', 'X2', 'X3', 'X4'],
class_names=['0', '1'], rounded=True, fontsize=15)
plt.show()
```

Para análise e melhor entendimento do gráfico da árvore de decisão podemos considerar os seguintes itens:

- ***Pergunta/Feature***: A primeira linha do quadro indica qual feature está sendo avaliada no nó.
- ***Entropia (Entropy)***: A entropia é uma medida da impureza do nó. Quanto maior a entropia, maior a mistura de classes no nó. O objetivo da árvore de decisão é reduzir a entropia em cada divisão, tornando os nós resultantes mais puros.
- ***Número de amostras (Samples)***: Indica quantas amostras estão sendo consideradas no nó.
- ***Valor (Value)***: Mostra a distribuição das classes no nó.
- ***Classe predominante (Class)***: Indica a classe predominante no nó.

- ***Ganho de Informação (Information Gain)***: O ganho de informação é a medida de quanto a divisão do nó reduz a entropia.

Cada nó da árvore de decisão representa uma decisão sobre como dividir os dados com base em uma feature específica, com o objetivo de criar subgrupos mais homogêneos em termos de classe. Desta forma é possível visualizar todas as decisões que foram tomadas até que se obtivessem os grupos mais homogêneos possíveis considerando os dados.

```
# Dicionario com métricas de comparacao para modelo Arvore de Decisao - Classificacao
dtc = {'ACC': ACC, 'MCC': MCC, 'SEN': SEN, 'VPP':VPP, 'F1': F1, 'BS': BS}
dtc = {key: round(value, 2) for key, value in dtc.items()}
print(dtc)
```

```
## {'ACC': 0.99, 'MCC': 0.97, 'SEN': 0.98, 'VPP': 0.99, 'F1': 0.99, 'BS': 0.01}
```

É importante ressaltar que, ao utilizar este tipo de modelo pode também ser realizado o processo de calibração. Este processo geralmente se refere à otimização de seus hiperparâmetros, que são configurações que controlam o processo de construção da árvore. Alguns dos hiperparâmetros mais importantes são a profundidade máxima da árvore (***max_depth***), número mínimo de amostras necessárias para dividir um nó interno (***min_samples_split***), número mínimo de amostras necessárias em uma folha (***min_samples_leaf***).

Desta forma é possível encontrar um conjunto de hiperparâmetros que aumentem o desempenho do modelo de classificação para um conjunto de dados. Mais detalhes sobre este tipo de modelo podem ser encontrados no tópico Decision Trees da documentação do Scikit-Learn.

Comparativo entre modelos de classificação

Após ajuste de todos os modelos propostos para tarefa de classificação é possível criar uma tabela contendo as métricas de ajuste dos modelos com a seguinte sintaxe:

```
# Criar um DataFrame a partir dos dicionários e adicionar uma coluna com o nome do dicionário
comp_model_class = pd.DataFrame([rlog, knnc, gnb, dtc])
comp_model_class['models'] = ['Reg. Logistic', 'KNN Class', 'Gaussian NB', 'DecisionTree Class']

# Exibir o DataFrame
print(comp_model_class)
```

```
##     ACC   MCC   SEN   VPP    F1    BS              models
## 0  0.98  0.97  0.98  0.98  0.98  0.01       Reg. Logistic
## 1  1.00  1.00  1.00  0.99  1.00  0.00           KNN Class
## 2  0.84  0.68  0.81  0.81  0.81  0.10         Gaussian NB
## 3  0.99  0.97  0.98  0.99  0.99  0.01  DecisionTree Class
```

Utilizando ordenação das linhas de forma decrescente para as colunas ACC até F1 e crescente para BS, podemos chegar aos modelos que tiveram melhor desempenho para estes dados.

```
# Ordenar o DataFrame pelos valores das colunas 'ACC' a 'F1' em ordem decrescente e pela
#coluna 'BS' em ordem crescente:

sorted_df = comp_model_class.sort_values(by=['ACC', 'MCC', 'SEN', 'VPP', 'F1', 'BS'],
ascending=[False, False, False, False, False, True])
```

```
# Imprimir o DataFrame ordenado
print(sorted_df)
```

```
##     ACC   MCC   SEN   VPP    F1    BS              models
## 1  1.00  1.00  1.00  0.99  1.00  0.00          KNN Class
## 3  0.99  0.97  0.98  0.99  0.99  0.01  DecisionTree Class
## 0  0.98  0.97  0.98  0.98  0.98  0.01       Reg. Logistic
## 2  0.84  0.68  0.81  0.81  0.81  0.10         Gaussian NB
```

A partir da tabela é possível perceber que o modelo que apresentou melhor desempenho de classificação para os dados Banknote foi o KNN - Classificação, seguido dos modelos de Árvore de Decisão e Regressão Logística. Contudo a escolha do melhor modelo neste caso pode ser realizada levando em consideração também o quesito de interpretabilidade, uma vez que, o modelo logístico apresenta uma vantagem pois permite a interpretação dos seus coeficientes, em termos da razão de chance de ocorrência do evento, de acordo com o valor de uma ou mais *features*. Assim, caso o foco seja apenas classificação o modelo de KNN - Classificação apresenta o melhor resultado, todavia se também há interesse em interpretabilidade pode-se optar pelo modelo logístico.

Caso a escolha do modelo tenha sido o modelo de Regressão Logística, podemos realizar o ajuste considerando todos os dados, e assim realizar a interpretação dos coeficientes:

```
from sklearn.linear_model import LogisticRegression

# Modelo de regressão logística
model = LogisticRegression()
reg_log = model.fit(X, y)

# Coeficientes do modelo
coeficientes = reg_log.coef_[0]
intercepto = reg_log.intercept_
odds_ratio = np.exp(coeficientes)

print(intercepto)
```

```
## [3.73908395]
```

```
# Criar um dicionário com os dados e rótulos
df_dict = {'Coeficientes': coeficientes, 'Odds Ratio': odds_ratio}

# Criar um DataFrame a partir do dicionário
tabela_coeficientes = pd.DataFrame(df_dict, index=X.columns.tolist())


# Imprimindo a tabela de coeficientes
print(tabela_coeficientes)
```

```
##     Coeficientes  Odds Ratio
## X1     -3.368258    0.034450
## X2     -1.887516    0.151448
## X3     -2.307540    0.099506
## X4     -0.088031    0.915733
```

Ou seja, o aumento de uma unidade da *feature* `X1` ocasiona um decréscimo de 96.55% na chance da nota ser falsificada. De forma análoga, acréscimos de uma unidade nas *features* `X2`, `X3` e `X4` ocasionam decréscimos de 84.86%, 90% e 8.5%, respectivamente, na chance da nota ser falsificada. A Equação (1) apresenta a

formulação do modelo logístico, e como os coeficientes estimados são utilizados para predição da probabilidade de uma nova nota ser falsificada $\hat{p}_i$.

$$P[y_i = 1] = \hat{p}_i = \frac{e^{3.74-3.37X_1-1.89X_2-2.31X_3-0.09X_4}}{1 + e^{3.74-3.37X_1-1.89X_2-2.31X_3-0.09X_4}} \tag{1}$$

Dados Boston

O conjunto de dados Boston, também conhecido como Boston Housing Dataset, é um conjunto de dados clássico frequentemente utilizado em problemas de regressão. Ele contém informações sobre habitação e características dos subúrbios de Boston, Massachusetts, EUA. O conjunto de dados foi coletado em 1978 e consiste em 506 observações, cada uma com 14 atributos:

Tabela 3: Variáveis disponíveis nos dados Boston

CRIM: Taxa de criminalidade per capita por cidade.	**DIS**: Distâncias ponderadas para cinco centros de emprego em Boston.
ZN: Proporção de terrenos residenciais com lotes com mais de 25.000 pés quadrados.	**RAD**: Índice de acessibilidade às rodovias radiais.
INDUS: Proporção de acres de negócios não varejistas por cidade.	**TAX**: Taxa de imposto sobre a propriedade de valor total por $10.000.
CHAS: Variável indicadora, relacionada ao rio Charles (= 1 se o trecho limita o rio; 0 caso contrário).	**PTRATIO**: Proporção aluno-professor por cidade.
NOX: Concentração de óxidos nítricos (partes por 10 milhões).	**Black**: $1000(B_k - 0.63)^2$, em que B_k é a proporção de pessoas negras por cidade.
RM: Número médio de quartos por habitação.	**LSTAT**: Porcentagem de status inferior da população.
AGE: Proporção de unidades ocupadas pelo proprietário construídas antes de 1940.	**MEDV**: Valor médio das casas ocupadas pelos proprietários, em milhares de dólares.

```
#Importando dados Boston
Boston=pd.read_excel('Boston_Housing.xlsx')

# Definir a configuração para mostrar todas as colunas
pd.set_option('display.max_columns', None)

#Visualização da estrutura dos dados
print(Boston.head())
```

```
##       crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
## 0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
## 1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
## 2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
## 3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
## 4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7
##
##     black  lstat  medv
## 0  396.90   4.98  24.0
## 1  396.90   9.14  21.6
## 2  392.83   4.03  34.7
```

```
## 3  394.63   2.94  33.4
## 4  396.90   5.33  36.2
```

O objetivo deste conjunto de dados é prever o valor médio das casas (**MEDV**) com base nas outras características. É um conjunto de dados comumente utilizado para práticas de regressão, modelagem preditiva e aprendizado de máquina em geral.

Análise descritiva e exploratória

Inicialmente vamos visualizar a tabela de medidas resumo para as *features* e a variável dependente (*target*), de modo a conhecer melhor as características dos dados.

```
#Medidas resumo para variaveis dos dados Boston
Boston.describe().T
```

```
##          count        mean         std       min         25%        50%  \
## crim     506.0    3.613524    8.601545    0.00632    0.082045    0.25651
## zn       506.0   11.363636   23.322453    0.00000    0.000000    0.00000
## indus    506.0   11.136779    6.860353    0.46000    5.190000    9.69000
## chas     506.0    0.069170    0.253994    0.00000    0.000000    0.00000
## nox      506.0    0.554695    0.115878    0.38500    0.449000    0.53800
## rm       506.0    6.284634    0.702617    3.56100    5.885500    6.20850
## age      506.0   68.574901   28.148861    2.90000   45.025000   77.50000
## dis      506.0    3.795043    2.105710    1.12960    2.100175    3.20745
## rad      506.0    9.549407    8.707259    1.00000    4.000000    5.00000
## tax      506.0  408.237154  168.537116  187.00000  279.000000  330.00000
## ptratio  506.0   18.455534    2.164946   12.60000   17.400000   19.05000
## black    506.0  356.674032   91.294864    0.32000  375.377500  391.44000
## lstat    506.0   12.653063    7.141062    1.73000    6.950000   11.36000
## medv     506.0   22.532806    9.197104    5.00000   17.025000   21.20000
##
##                 75%       max
## crim       3.677083   88.9762
## zn        12.500000  100.0000
## indus     18.100000   27.7400
## chas       0.000000    1.0000
## nox        0.624000    0.8710
## rm         6.623500    8.7800
## age       94.075000  100.0000
## dis        5.188425   12.1265
## rad       24.000000   24.0000
## tax      666.000000  711.0000
## ptratio   20.200000   22.0000
## black    396.225000  396.9000
## lstat     16.955000   37.9700
## medv      25.000000   50.0000
```

Analisando as medidas resumo para todas as variáveis dos dados Boston é possível perceber algumas características como a ausencia de informações faltantes na base de dados, assim como as diferenças entre as escalas de mensuração das variáveis, que pode ser notada ao analisar a amplitude, média e variabilidade das variáveis, com as *features* `tax`, `black` e `age` apresentando valores distintos de média e variabilidade se comparadas com as demais.

Em seguida podemos também visualizar a tabela de proporção para variável `chas`, que é a única *feature* categórica nos dados Boston. Dos 502 registros, apenas 35 relataram que o trecho limita o rio Charles.

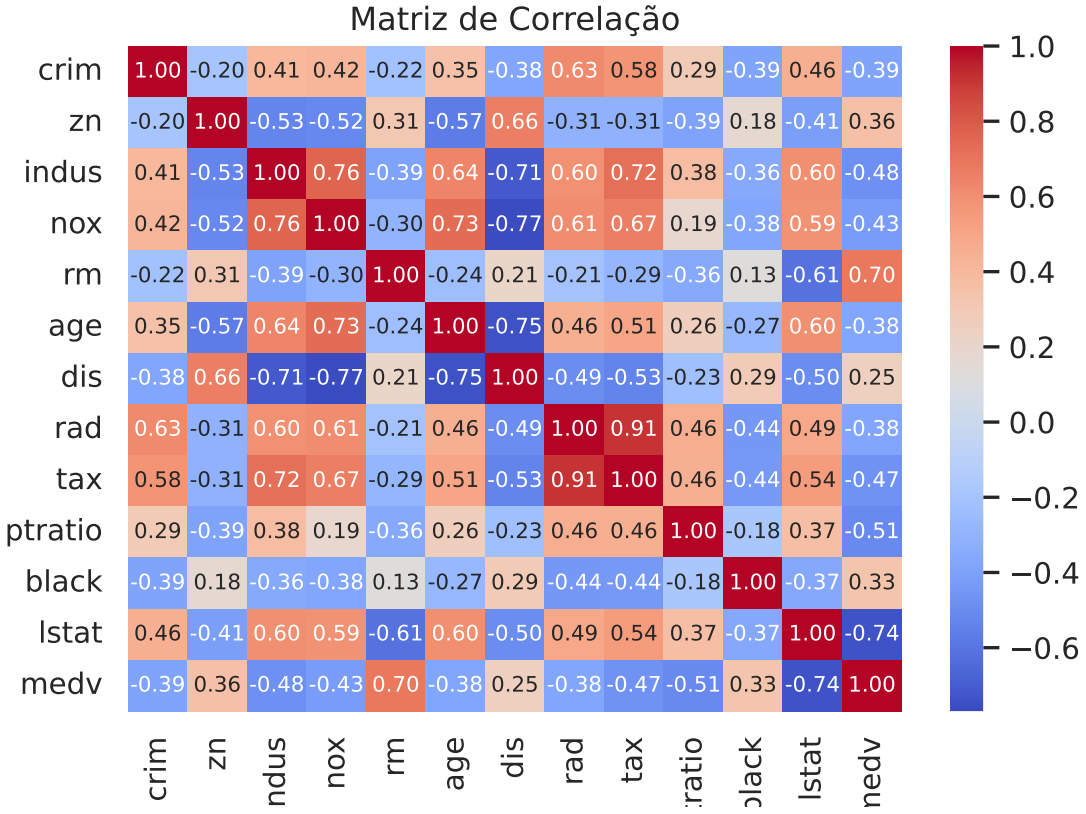
```
#Proporção de observações nas categorias da feature chas
Boston['chas'].value_counts()
```

```
## chas
## 0    471
## 1     35
## Name: count, dtype: int64
```

Como temos um número grande de *features*, uma forma interessante de visualizar as estruturas de correlação dos dados é apresentar a matriz de correlação linear, juntamente do gráfico de calor para a mesma. Este gráfico possibilita mensurar os valores para correlação linear de Pearson entre pares de variáveis.

```
sns.set(font_scale=1.0)

#Gráfico da matriz de correlação
pairplot=(sns.heatmap(Boston.drop(columns=['chas']).corr(), annot=True,
cmap='coolwarm', fmt='.2f',annot_kws={"size": 8}))
plt.title('Matriz de Correlação')
plt.show(pairplot)
```



Analisando o gráfico para esta matriz percebemos a ocorrência de correlações lineares fortes ($|cor| > 0.7$) entre os pares de *features*: `corr(nox,indius)=0.76`, `cor(age,nox)=0.73` ,`cor(dis,indius)=-0.71`, `cor(dis,nox)=-0.77`, `cor(dis,age)=-0.75` e `cor(tax,indius)=0.72`, `cor(tax,rad)=0.91`.

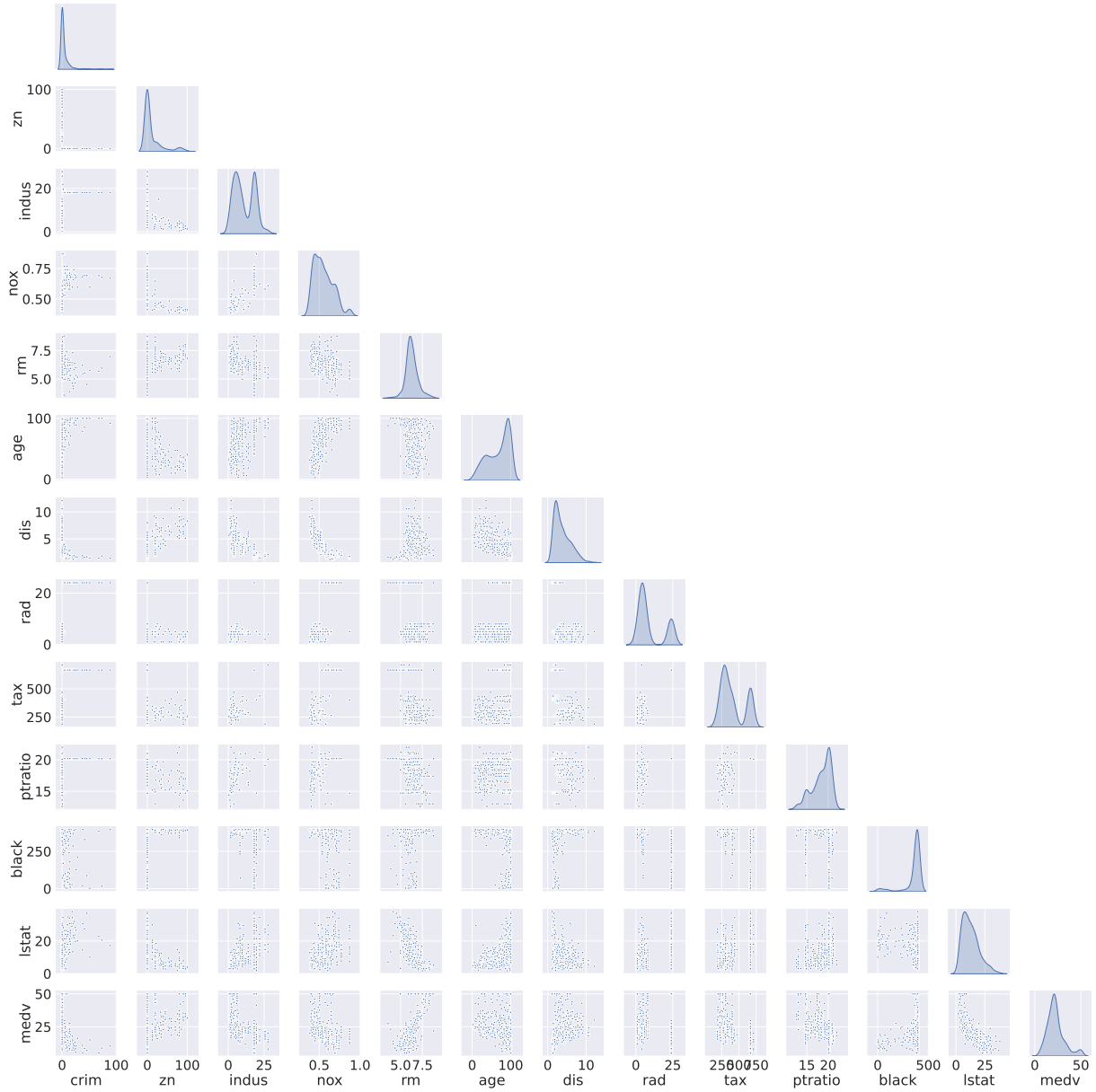
Analisando as correlações lineares entre *features* e a variável *target* temos correlações fortes nos seguintes pares de variáveis: `cor(rm,medv)=0.70` e `cor(lstat,medv)=-0.74`

Contudo o valor numérico para o coeficiente de correlação linear de Pearson não significa que haja de fato uma correlação linear entre as variávies, e para confirmar este fato é importante analisar o diagrama de dispersão entre todas as variáveis numéricas estudadas.

```
#Gráfico de dispersão entre variaveis da base de dados
# A variavel 'chas' é categorica e por isso nao sera considerada neste grafico

# Setting the font size by 3
sns.set(font_scale=3)

pairplot=(sns.pairplot(Boston.drop(columns=['chas']), markers=['o', 'o'],
plot_kws={'s': 30},diag_kind='kde', corner=True, height=3))
plt.show(pairplot)
```



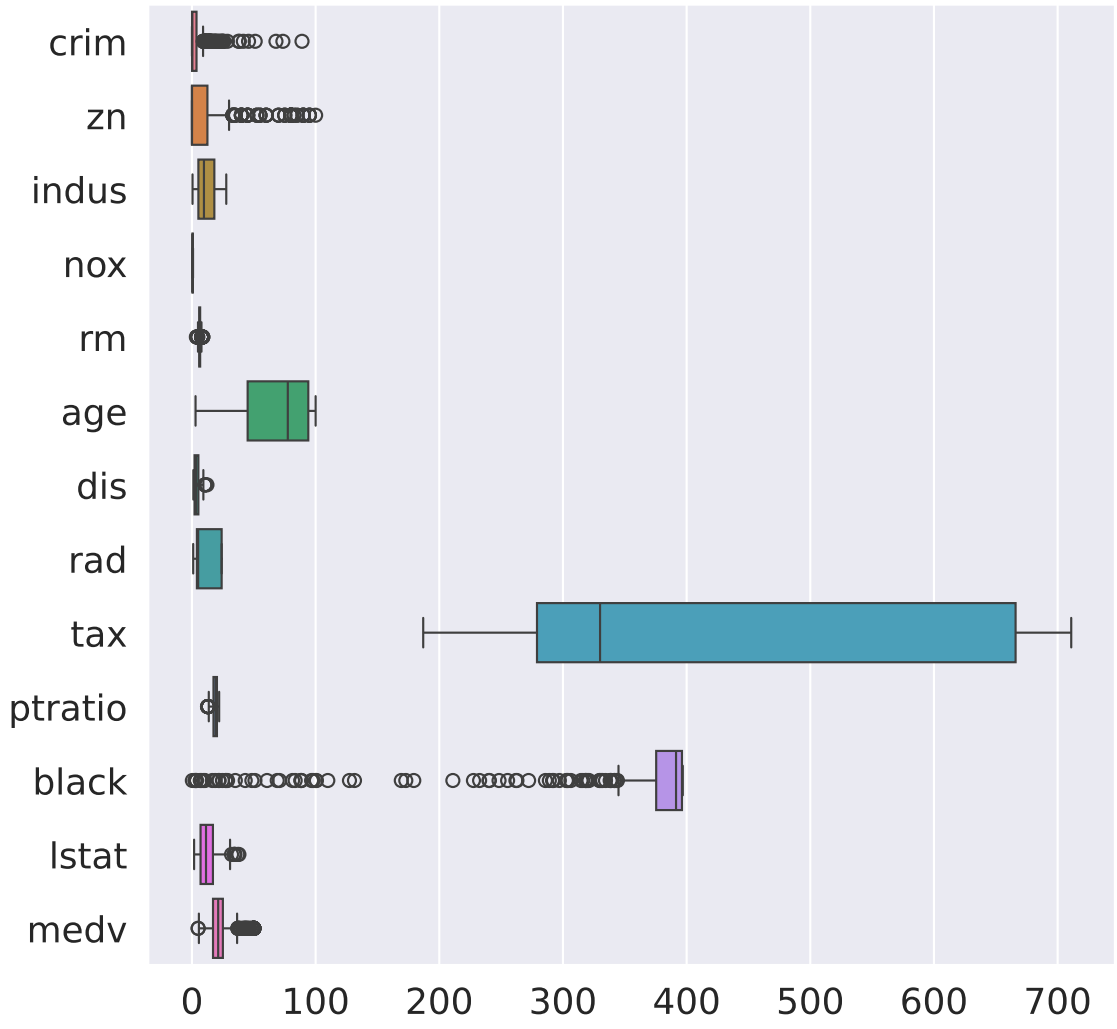
Ao analisar graficamente os diagramas de dispersão para os pares de *features* que apresentaram valor

para correlação linear alta percebe-se que apenas as *features* `(nox,indius)`, `(dis,indius)`, `(dis,nox)` e `(dis,age)` apresentam de fato correlação linear entre si. Já para os casos entre *feature* e *target* temos evidências de que `(rm,medv)` e `(lstat,medv)` apresentam de fato relação linear entre si. Este tipo de verificação é muito importante pelos seguintes motivos:

- Verificar se o valor do coeficiente de correlação linear de Pearson calculado é válido;
- Investigar como e com que intensidade as *features* estão relacionadas entre si e com a variável *target*;
- Obter maior conhecimento sobre as *features*, afim de evitar problemas relacionados a multicolinearidade.

Em seguida podemos visualizar numa figura os boxplots para *features* dos dados Boston:

```
# Plotar os boxplots
plt.figure(figsize=(8, 8))
sns.set(font_scale=1.5)
sns.boxplot(data=Boston.drop(columns=['chas']), orient='h')
plt.show()
```



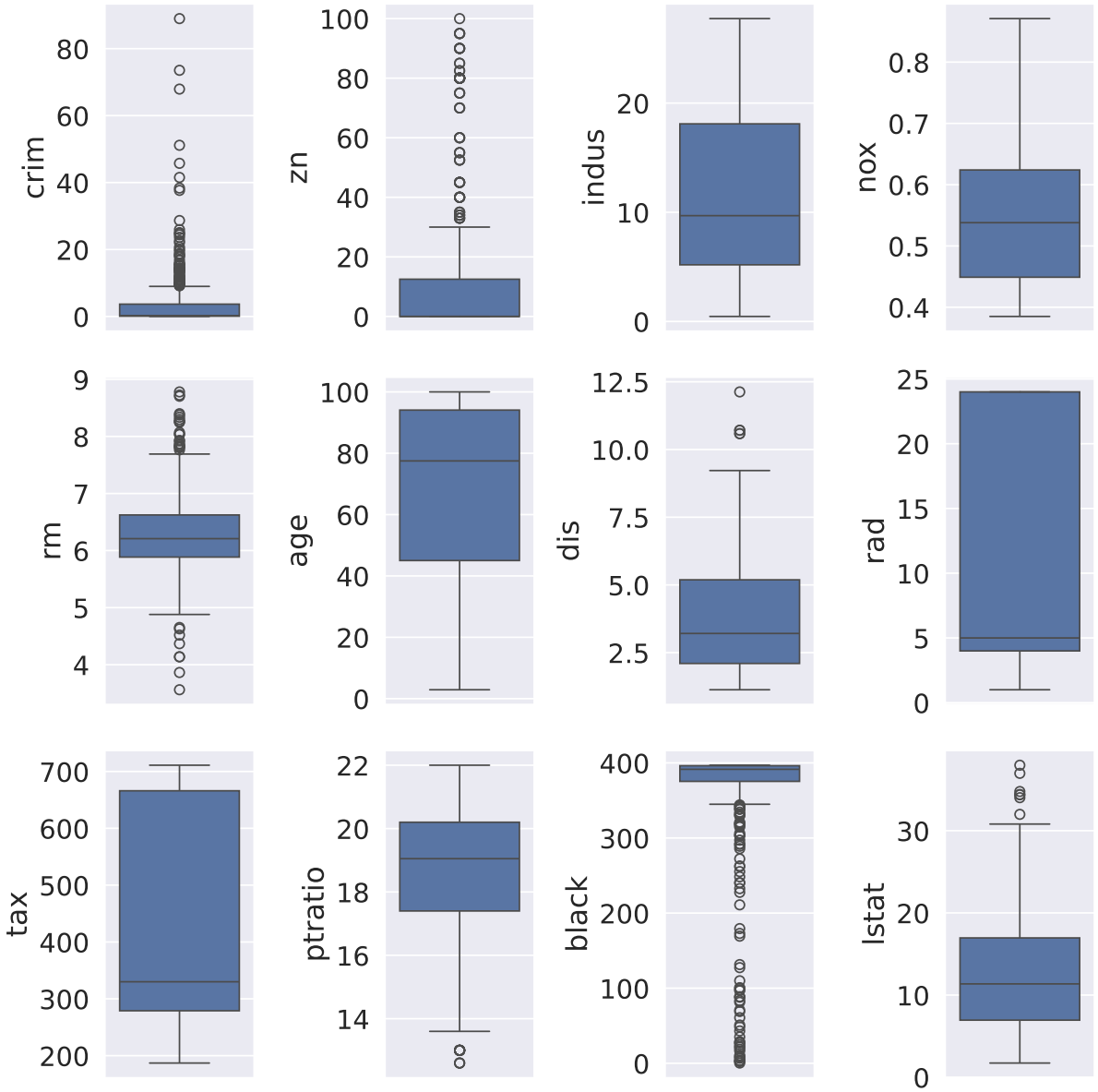
Como já havia sido notado a partir das medidas resumo, as diferentes escalas de mensuração das *features* podem prejudicar a análise. Desta forma é recomendado que os boxplots sejam visualizados separadamente:

```
boston_df=Boston.drop(columns=['chas','medv'])

# Criando uma figura e os subplots
fig, axs = plt.subplots(3, 4, figsize=(10, 10))

# Iterando sobre as variáveis e criando um boxplot para cada uma delas
for i, colu in enumerate(boston_df.columns):
    row = i // 4
    col = i % 4
    sns.boxplot(y=boston_df[colu], ax=axs[row, col])
```

```
# Ajustando o layout e mostrando os gráficos
plt.tight_layout()
plt.show()
```



Analisando a figura acima é possivel perceber que as *features* `crim` e `black` possuem distribuição bastante assimétrica, com assimetrias positiva e negativa respectivamente, além de um número elevado de *outliers*. Outras *features* que também apresentam assimetria positiva são `zn`, `rad` e `tax`, ou seja, indica que há uma concentração de dados nas extremidades menores da escala, com alguns valores extremamente altos que aumentam a média.

Como a escala dos dados, assimetria e número de outliers varia bastante entre as features o procedimento

de padronização destas variáveis se faz recomendado, evitando que features com escalas muito diferentes influenciem o modelo de maneira desproporcional. Desta forma espera-se realizar um melhor ajuste dos modelos propostos a este conjunto de dados.

Processo de divisão dos dados

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error

# Seu conjunto de dados
X = Boston.drop(columns=['medv'])
y = Boston['medv']


# Divisão em treino e teste com uma semente (seed)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=321)
```

Após o processo de divisão dos dados em teste e treinamento, as *features* presentes nos conjuntos de teste e treinamento serão padronizadas, de forma diminuir as discrepancias causadas pela diferenças de escala. Para realizar a padronização das *features* pode-se utilizar a seguinte sintaxe:

```
from sklearn.preprocessing import StandardScaler

# Padronizar as features ( recomendado para estes dados)
scaler = StandardScaler()
X_train_sc = scaler.fit_transform(X_train)
X_test_sc = scaler.transform(X_test)
```

Regressão linear múltipla

```
from sklearn.linear_model import LinearRegression

# Atribuindo modelo de regressão linear múltipla a um objeto
lm = LinearRegression()

# Treinando o modelo com dados de treino
lm.fit(X_train, y_train)
```

```
## LinearRegression()
```

```
# Calcular previsões utilizando dados de teste
y_pred = lm.predict(X_test)

# Calcular métricas de ajuste
r2 = r2_score(y_test, y_pred)
adj_r2 = 1 - (1 - r2) * (len(y_test) - 1) / (len(y_test) - X_test.shape[1] - 1)
mae = mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

Agora imprimindo resultados para as estatísticas de ajuste:

```
# Dicionario contendo métricas de comparacao para modelo de Reg. Linear multipla
reg_lin = {'R2': r2, 'R2a': adj_r2, 'MAE': mae, 'MAPE':mape, 'RMSE': rmse}
reg_lin = {key: round(value, 2) for key, value in reg_lin.items()}
print(reg_lin)
```

```
## {'R2': 0.73, 'R2a': 0.71, 'MAE': 3.33, 'MAPE': 0.17, 'RMSE': 4.64}
```

KNN - Regressão

```
from sklearn.neighbors import KNeighborsRegressor

# Atribuindo modelo de KNN para Regressão (k=10 arbitrário) a um objeto
knn_reg = KNeighborsRegressor(n_neighbors=10)

# Treinando o modelo com dados de treino
knn_reg.fit(X_train, y_train)
```

```
## KNeighborsRegressor(n_neighbors=10)
```

```
# Calcular previsões utilizando dados de teste
y_pred = knn_reg.predict(X_test)

# Calcular métricas de ajuste
r2 = r2_score(y_test, y_pred)
adj_r2 = 1 - (1 - r2) * (len(y_test) - 1) / (len(y_test) - X_test.shape[1] - 1)
mae = mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

Agora imprimindo resultados para as estatísticas de ajuste:

```
# Dicionario contendo métricas de comparacao para modelo de KNN Regressão
KNN_Reg = {'R2': r2, 'R2a': adj_r2, 'MAE': mae, 'MAPE':mape, 'RMSE': rmse}
KNN_Reg = {key: round(value, 2) for key, value in KNN_Reg.items()}
print(KNN_Reg)
```

```
## {'R2': 0.49, 'R2a': 0.44, 'MAE': 4.52, 'MAPE': 0.2, 'RMSE': 6.42}
```

Árvore de regressão

```
from sklearn.tree import DecisionTreeRegressor

# Atribuindo modelo de Árvore de Regressão a um objeto
arvre_reg = DecisionTreeRegressor()

# Treinando o modelo com dados de treino
arvre_reg.fit(X_train, y_train)
```

```
## DecisionTreeRegressor()
```

```
# Calcular previsões utilizando dados de teste
y_pred = arvre_reg.predict(X_test)
```

```
# Calcular métricas de ajuste
r2 = r2_score(y_test, y_pred)
adj_r2 = 1 - (1 - r2) * (len(y_test) - 1) / (len(y_test) - X_test.shape[1] - 1)
mae = mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

Agora imprimindo resultados para as estatísticas de ajuste:

```
# Dicionario contendo métricas de comparacao para modelo de Árvore de Regressão
Arvre_Reg = {'R2': r2, 'R2a': adj_r2, 'MAE': mae, 'MAPE':mape, 'RMSE': rmse}
Arvre_Reg = {key: round(value, 2) for key, value in Arvre_Reg.items()}
print(Arvre_Reg)
```

```
## {'R2': 0.7, 'R2a': 0.67, 'MAE': 3.3, 'MAPE': 0.17, 'RMSE': 4.92}
```

Comparativo entre modelos

Após ajuste de todos os modelos propostos para tarefa de classificação é possível criar uma tabela contendo as métricas de ajuste dos modelos com a seguinte sintaxe:

```
# Criar um DataFrame a partir dos dicionários e adicionar uma coluna com o nome do dicionário
comp_model_reg = pd.DataFrame([reg_lin, KNN_Reg, Arvre_Reg])
comp_model_reg['models'] = ['Reg. Linear', 'KNN Regressão', 'Árvore de Regressão']

# Exibir o DataFrame
print(comp_model_reg)
```

```
##      R2   R2a   MAE  MAPE  RMSE              models
## 0  0.73  0.71  3.33  0.17  4.64         Reg. Linear
## 1  0.49  0.44  4.52  0.20  6.42       KNN Regressão
## 2  0.70  0.67  3.30  0.17  4.92  Árvore de Regressão
```

Utilizando ordenação das linhas de forma decrescente para as colunas MAE até RMSE e crescente para R^2 e R^2_a, podemos visualizar aos modelos que tiveram melhor desempenho para estes dados de forma ordenada.

```
# Ordenar o DataFrame pelos valores das colunas 'ACC' a 'F1' em ordem decrescente e pela
#coluna 'BS' em ordem crescente:

sorted_df = comp_model_reg.sort_values(by=['R2', 'R2a', 'MAE', 'MAPE', 'RMSE'],
ascending=[False, False, True, True, True])

# Imprimir o DataFrame ordenado
print(sorted_df)
```

```
##      R2   R2a   MAE  MAPE  RMSE              models
## 0  0.73  0.71  3.33  0.17  4.64         Reg. Linear
## 2  0.70  0.67  3.30  0.17  4.92  Árvore de Regressão
## 1  0.49  0.44  4.52  0.20  6.42       KNN Regressão
```

A partir da tabela é possível perceber que o modelo que apresentou melhor desempenho de classificação para os dados Boston foi a Árvore de Regressão, seguido dos modelos de Regressão Linear e KNN para Regressão. Contudo a escolha do melhor modelo neste caso também pode ser realizada levando em consideração o quesito

de interpretabilidade, uma vez que, o modelo linear apresenta uma vantagem em relação a árvore de regressão pois permite a interpretação mais clara dos seus coeficientes. Assim, caso o foco seja apenas em realizar predição de valores, o modelo de Árvore de Regressão apresentou as melhores métricas, todavia, se também há interesse em interpretabilidade pode-se optar pelo modelo de regressão linear.

Caso a escolha do modelo tenha sido o modelo de Regressão Linear, podemos realizar o ajuste considerando todos os dados, e assim realizar a interpretação dos coeficientes:

```
#Ajustando modelo de regressão linear aos dados completos
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

# Padronizando as features
scaler = StandardScaler()
X_sc = scaler.fit_transform(X)

# Atribuindo modelo de regressão linear múltipla a um objeto
lm_all = LinearRegression()

# Treinando o modelo com dados de treino
lm_all.fit(X_sc, y)
```

```
## LinearRegression()
```

```
# Obtendo os coeficientes do modelo
coeficientes = lm_all.coef_
intercepto = lm_all.intercept_


# Criando a tabela de coeficientes
tabela_coeficientes = pd.DataFrame(coeficientes, columns=['Coeficiente'], index=X.columns.tolist())
tabela_coeficientes.loc['Intercepto'] = intercepto


# Imprimindo a tabela de coeficientes
print(tabela_coeficientes)
```

```
##             Coeficiente
## crim          -0.928146
## zn             1.081569
## indus          0.140900
## chas           0.681740
## nox           -2.056718
## rm             2.674230
## age            0.019466
## dis           -3.104044
## rad            2.662218
## tax           -2.076782
## ptratio       -2.060607
## black          0.849268
## lstat         -3.743627
## Intercepto    22.532806
```

Quando as features são padronizadas, os coeficientes do modelo ainda podem ser interpretados como em

modelos com features não padronizadas, mas com algumas diferenças importantes devido à padronização. Aqui estão algumas considerações:

1. **Interpretação do coeficiente**: Um coeficiente positivo significa que um aumento de uma unidade na feature está associado a um aumento de (β) unidades na média da variável resposta, e para um coeficiente negativo representa um decréscimo. No entanto, como as features foram padronizadas, uma unidade de mudança corresponde a uma mudança de um desvio padrão da feature padronizada.
2. **Comparação entre features**: A magnitude dos coeficientes ainda pode ser usada para comparar a importância relativa das features no modelo. Coeficientes maiores em magnitude indicam uma maior influência da feature na variável resposta, considerando a escala padronizada.
3. **Intercepto**: O intercepto também pode ser interpretado da mesma maneira, representando o valor médio da variável resposta quando todas as features padronizadas são iguais a zero (ou ao seu valor médio padronizado).

Portanto, embora os coeficientes sejam interpretados em termos de desvios padrão das features padronizadas, a interpretação geral ainda é semelhante à interpretação em modelos com features não padronizadas.

$$\begin{aligned} \hat{MEDV}_i = 22.53 &- 0.93 \times \text{crim} + 1.08 \times \text{zn} + 0.14 \times \text{indus} + 0.68 \times \text{chas} - 2.06 \times \text{nox} + 2.67 \times \text{rm} \\ &+ 0.02 \times \text{age} - 3.10 \times \text{dis} + 2.66 \times \text{rad} - 2.07 \times \text{tax} - 2.06 \times \text{pratio} + 0.84 \times \text{Black} - 3.74 \times \text{lstat} \end{aligned} \tag{2}$$

A partir dos coeficientes do modelo linear, ou também da Equação (2), podemos chegar a algumas interpretações:

As *features* `lstat` (Porcentagem de status inferior da população), `dis` (Distâncias ponderadas para cinco centros de emprego em Boston), `rm` (Número médio de quartos por habitação) e `rad` (Índice de acessibilidade às rodovias radiais) são as que apresentam o maior influência na variável resposta `medv` (Valor médio das casas ocupadas pelos proprietários, em milhares de dólares).

Temos ainda que o valor médio das casas ocupadas pelos proprietários, em milhares de dólares, quando todas as features padronizadas são iguais a zero é igual a 22.53.

Um coeficiente de -3,10 para a feature `dis` indica que, para cada aumento de um desvio padrão nas distâncias ponderadas para cinco centros de emprego em Boston, o valor médio das casas ocupadas pelos proprietários, em milhares de dólares decresce em 3,10 desvios padrão, em média.

Conclusão

Para os dados Banknote, o modelo com melhores métricas de ajuste foi o KNN - classificação, enquanto que para os dados Boston o melhor modelo foi o de Árvore de regressão. Contudo essa escolha foi baseada puramente nas métricas de ajuste para cada situação, de acordo também com os conjuntos de treino e teste utilizados que podem variar de acordo com a *seed* escolhida. Porém, também foram apresentados coeficientes que permitem a interpretação do ajuste utilizando modelos de regressão adequados para cada cenário. Esses resultados destacam a importância de considerar não apenas as métricas de ajuste, mas também a interpretação dos coeficientes, ao escolher o melhor modelo para um determinado conjunto de dados.

Considerações finais

Durante o processo de ajuste dos modelos em cada um dos conjuntos de dados apresentado é importante atentar-se a alguns pontos:

- Em alguns casos foi utilizado o procedimento de padronização das *features*;
- A alteração da seed ocasionaria em diferentes amostras de treino e teste, o que poderia afetar o resultado final;
- Embora as métricas ofereçam um bom suporte para escolha de um modelo, esta deve ser realizada também levando em consideração demais critérios como a interpretabilidade;
- Alguns modelos como Árvores de decisão/Classificação e também KNN necessitam de parâmetros adicionais (hiperparâmetros), e a escolha de diferentes hiperparâmetros pode influenciar no ajuste destes modelos, contudo, para fins ilustrativos foram setados alguns valores pré estabelecidos.

O estudo comparativo entre modelos neste notbook aborda um framework para que seja possível elencar um modelo candidato para situações envolvendo classificação e regressão. Entretanto, melhores resultados podem ser obtidos por meio da realização de procedimentos de validação cruzada, *K-fold* repetido juntamente com *Hold-out*, visando a calibração ou *tunning* de parâmetros ou hiperparâmetros de alguns modelos, porém estes são tópicos mais avançados que serão posteriormente discutidos nos próximos materiais.

Referências

- BRIER, Glenn W. Verification of forecasts expressed in terms of probability. Monthly weather review, v. 78, n. 1, p. 1-3, 1950.
- Common pitfalls and recommended practices. 10. Disponível em: https://scikit-learn.org/stable/common_pitfalls.html#:~:text=Always%20split%20the%20data%20into. Acesso em: 7 mar. 2024.
- Decision Trees. 1.10. Disponível em: https://scikit-learn.org/stable/modules/tree.html#multi-output-problems. Acesso em: 7 mar. 2024.
- IZBICK, R; SANTOS, T.M. (2020). Aprendizado de Máquina: Uma Abordagem Estatística. ISBN 978-65-00-02410-4. Disponível em: http://www.rizbicki.ufscar.br/AME.pdf
- JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. (2021). An Introduction to Statistical Learning. 2nd Ed. New York: Springer. Disponível em: https://web.stanford.edu/~hastie/ISLR2/ISLRv2_website.pdf
- Metrics and scoring: quantifying the quality of predictions. 3.3 — scikit-learn 0.22.1 documentation. Disponível em: https://scikit-learn.org/stable/modules/model_evaluation.html. Acesso em: 7 mar. 2024.