Exercício Prático 01 _ Projeto de Aprendizado de Máquina de Ponta a Ponta

October 29, 2025

Usaremos o dataset **Ames Housing** para construir um modelo de regressão.

Objetivo: Prever o preço final de venda (`SalePrice`) de cada casa, com base em suas características.

0.1 1. Setup Inicial

Primeiro, vamos importar todas as bibliotecas que precisaremos ao longo do exercício.

[71]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

sns.set(style="whitegrid")
```

0.2 2. Obtenção dos Dados

Vamos carregar o dataset Ames Housing diretamente do OpenML.

[72]:
```
housing = fetch_openml(name="house_prices", as_frame=True)
df = housing.frame
print("Dados carregados com sucesso!")
```

```
Dados carregados com sucesso!
```

0.3 3. Exploração dos Dados

0.3.1 Exercício 1: Explorar o conteúdo do DataFrame

Use os métodos do Pandas para visualizar as primeiras linhas, obter um resumo das colunas (tipos de dados e valores nulos) e gerar estatísticas descritivas.

[73]:
```
print("As 5 primeiras linhas (head):")
print(df.head()) # SEU CÓDIGO AQUI

print("\nResumo dos dados (info):")
df.info() # SEU CÓDIGO AQUI

print("\nEstatísticas Descritivas (describe):")
print(df.describe()) # SEU CÓDIGO AQUI

# Categorias de cada atributo categórico
print("\nCategorias de cada atributo categórico:")
for col in df.select_dtypes(include=['object', 'category']):
    print(f"{col}: {df[col].unique()}")
```

```
As 5 primeiras linhas (head):
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0   1          60       RL         65.0     8450   Pave   NaN      Reg
1   2          20       RL         80.0     9600   Pave   NaN      Reg
2   3          60       RL         68.0    11250   Pave   NaN      IR1
3   4          70       RL         60.0     9550   Pave   NaN      IR1
4   5          60       RL         84.0    14260   Pave   NaN      IR1

  LandContour Utilities  … PoolArea PoolQC Fence MiscFeature MiscVal MoSold  \
0         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
1         Lvl    AllPub  …        0    NaN   NaN         NaN       0      5
2         Lvl    AllPub  …        0    NaN   NaN         NaN       0      9
3         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
4         Lvl    AllPub  …        0    NaN   NaN         NaN       0     12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000

[5 rows x 81 columns]

Resumo dos dados (info):
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
```

```
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     1452 non-null   object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
 29  Foundation     1460 non-null   object
 30  BsmtQual       1423 non-null   object
 31  BsmtCond       1423 non-null   object
 32  BsmtExposure   1422 non-null   object
 33  BsmtFinType1   1423 non-null   object
 34  BsmtFinSF1     1460 non-null   int64
 35  BsmtFinType2   1422 non-null   object
 36  BsmtFinSF2     1460 non-null   int64
 37  BsmtUnfSF      1460 non-null   int64
 38  TotalBsmtSF    1460 non-null   int64
 39  Heating        1460 non-null   object
 40  HeatingQC      1460 non-null   object
 41  CentralAir     1460 non-null   object
 42  Electrical     1459 non-null   object
 43  1stFlrSF       1460 non-null   int64
 44  2ndFlrSF       1460 non-null   int64
 45  LowQualFinSF   1460 non-null   int64
```

```
 46  GrLivArea      1460 non-null   int64
 47  BsmtFullBath   1460 non-null   int64
 48  BsmtHalfBath   1460 non-null   int64
 49  FullBath       1460 non-null   int64
 50  HalfBath       1460 non-null   int64
 51  BedroomAbvGr   1460 non-null   int64
 52  KitchenAbvGr   1460 non-null   int64
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB

Estatísticas Descritivas (describe):
                Id   MSSubClass  LotFrontage        LotArea  OverallQual  \
count  1460.000000  1460.000000  1201.000000    1460.000000  1460.000000
mean    730.500000    56.897260    70.049958   10516.828082     6.099315
std     421.610009    42.300571    24.284752    9981.264932     1.382997
min       1.000000    20.000000    21.000000    1300.000000     1.000000
25%     365.750000    20.000000    59.000000    7553.500000     5.000000
50%     730.500000    50.000000    69.000000    9478.500000     6.000000
75%    1095.250000    70.000000    80.000000   11601.500000     7.000000
max    1460.000000   190.000000   313.000000  215245.000000    10.000000
```

```
       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea   BsmtFinSF1  …  \
count  1460.000000  1460.000000   1460.000000  1452.000000  1460.000000  …
mean      5.575342  1971.267808   1984.865753   103.685262   443.639726  …
std       1.112799    30.202904     20.645407   181.066207   456.098091  …
min       1.000000  1872.000000   1950.000000     0.000000     0.000000  …
25%       5.000000  1954.000000   1967.000000     0.000000     0.000000  …
50%       5.000000  1973.000000   1994.000000     0.000000   383.500000  …
75%       6.000000  2000.000000   2004.000000   166.000000   712.250000  …
max       9.000000  2010.000000   2010.000000  1600.000000  5644.000000  …

        WoodDeckSF  OpenPorchSF  EnclosedPorch    3SsnPorch  ScreenPorch  \
count  1460.000000  1460.000000    1460.000000  1460.000000  1460.000000
mean     94.244521    46.660274      21.954110     3.409589    15.060959
std     125.338794    66.256028      61.119149    29.317331    55.757415
min       0.000000     0.000000       0.000000     0.000000     0.000000
25%       0.000000     0.000000       0.000000     0.000000     0.000000
50%       0.000000    25.000000       0.000000     0.000000     0.000000
75%     168.000000    68.000000       0.000000     0.000000     0.000000
max     857.000000   547.000000     552.000000   508.000000   480.000000

          PoolArea       MiscVal       MoSold       YrSold      SalePrice
count  1460.000000   1460.000000  1460.000000  1460.000000    1460.000000
mean      2.758904     43.489041     6.321918  2007.815753  180921.195890
std      40.177307    496.123024     2.703626     1.328095   79442.502883
min       0.000000      0.000000     1.000000  2006.000000   34900.000000
25%       0.000000      0.000000     5.000000  2007.000000  129975.000000
50%       0.000000      0.000000     6.000000  2008.000000  163000.000000
75%       0.000000      0.000000     8.000000  2009.000000  214000.000000
max     738.000000  15500.000000    12.000000  2010.000000  755000.000000

[8 rows x 38 columns]

Categorias de cada atributo categórico:
MSZoning: ['RL' 'RM' "'C (all)'" 'FV' 'RH']
Street: ['Pave' 'Grvl']
Alley: [nan 'Grvl' 'Pave']
LotShape: ['Reg' 'IR1' 'IR2' 'IR3']
LandContour: ['Lvl' 'Bnk' 'Low' 'HLS']
Utilities: ['AllPub' 'NoSeWa']
LotConfig: ['Inside' 'FR2' 'Corner' 'CulDSac' 'FR3']
LandSlope: ['Gtl' 'Mod' 'Sev']
Neighborhood: ['CollgCr' 'Veenker' 'Crawfor' 'NoRidge' 'Mitchel' 'Somerst'
'NWAmes'
 'OldTown' 'BrkSide' 'Sawyer' 'NridgHt' 'NAmes' 'SawyerW' 'IDOTRR'
 'MeadowV' 'Edwards' 'Timber' 'Gilbert' 'StoneBr' 'ClearCr' 'NPkVill'
 'Blmngtn' 'BrDale' 'SWISU' 'Blueste']
Condition1: ['Norm' 'Feedr' 'PosN' 'Artery' 'RRAe' 'RRNn' 'RRAn' 'PosA' 'RRNe']
```

```
Condition2: ['Norm' 'Artery' 'RRNn' 'Feedr' 'PosN' 'PosA' 'RRAn' 'RRAe']
BldgType: ['1Fam' '2fmCon' 'Duplex' 'TwnhsE' 'Twnhs']
HouseStyle: ['2Story' '1Story' '1.5Fin' '1.5Unf' 'SFoyer' 'SLvl' '2.5Unf'
'2.5Fin']
RoofStyle: ['Gable' 'Hip' 'Gambrel' 'Mansard' 'Flat' 'Shed']
RoofMatl: ['CompShg' 'WdShngl' 'Metal' 'WdShake' 'Membran' 'Tar&Grv' 'Roll'
 'ClyTile']
Exterior1st: ['VinylSd' 'MetalSd' "'Wd Sdng'" 'HdBoard' 'BrkFace' 'WdShing'
'CemntBd'
 'Plywood' 'AsbShng' 'Stucco' 'BrkComm' 'AsphShn' 'Stone' 'ImStucc'
 'CBlock']
Exterior2nd: ['VinylSd' 'MetalSd' "'Wd Shng'" 'HdBoard' 'Plywood' "'Wd Sdng'"
'CmentBd'
 'BrkFace' 'Stucco' 'AsbShng' "'Brk Cmn'" 'ImStucc' 'AsphShn' 'Stone'
 'Other' 'CBlock']
MasVnrType: ['BrkFace' 'None' 'Stone' 'BrkCmn' nan]
ExterQual: ['Gd' 'TA' 'Ex' 'Fa']
ExterCond: ['TA' 'Gd' 'Fa' 'Po' 'Ex']
Foundation: ['PConc' 'CBlock' 'BrkTil' 'Wood' 'Slab' 'Stone']
BsmtQual: ['Gd' 'TA' 'Ex' nan 'Fa']
BsmtCond: ['TA' 'Gd' nan 'Fa' 'Po']
BsmtExposure: ['No' 'Gd' 'Mn' 'Av' nan]
BsmtFinType1: ['GLQ' 'ALQ' 'Unf' 'Rec' 'BLQ' nan 'LwQ']
BsmtFinType2: ['Unf' 'BLQ' nan 'ALQ' 'Rec' 'LwQ' 'GLQ']
Heating: ['GasA' 'GasW' 'Grav' 'Wall' 'OthW' 'Floor']
HeatingQC: ['Ex' 'Gd' 'TA' 'Fa' 'Po']
CentralAir: ['Y' 'N']
Electrical: ['SBrkr' 'FuseF' 'FuseA' 'FuseP' 'Mix' nan]
KitchenQual: ['Gd' 'TA' 'Ex' 'Fa']
Functional: ['Typ' 'Min1' 'Maj1' 'Min2' 'Mod' 'Maj2' 'Sev']
FireplaceQu: [nan 'TA' 'Gd' 'Fa' 'Ex' 'Po']
GarageType: ['Attchd' 'Detchd' 'BuiltIn' 'CarPort' nan 'Basment' '2Types']
GarageFinish: ['RFn' 'Unf' 'Fin' nan]
GarageQual: ['TA' 'Fa' 'Gd' nan 'Ex' 'Po']
GarageCond: ['TA' 'Fa' nan 'Gd' 'Po' 'Ex']
PavedDrive: ['Y' 'N' 'P']
PoolQC: [nan 'Ex' 'Fa' 'Gd']
Fence: [nan 'MnPrv' 'GdWo' 'GdPrv' 'MnWw']
MiscFeature: [nan 'Shed' 'Gar2' 'Othr' 'TenC']
SaleType: ['WD' 'New' 'COD' 'ConLD' 'ConLI' 'CWD' 'ConLw' 'Con' 'Oth']
SaleCondition: ['Normal' 'Abnorml' 'Partial' 'AdjLand' 'Alloca' 'Family']
```

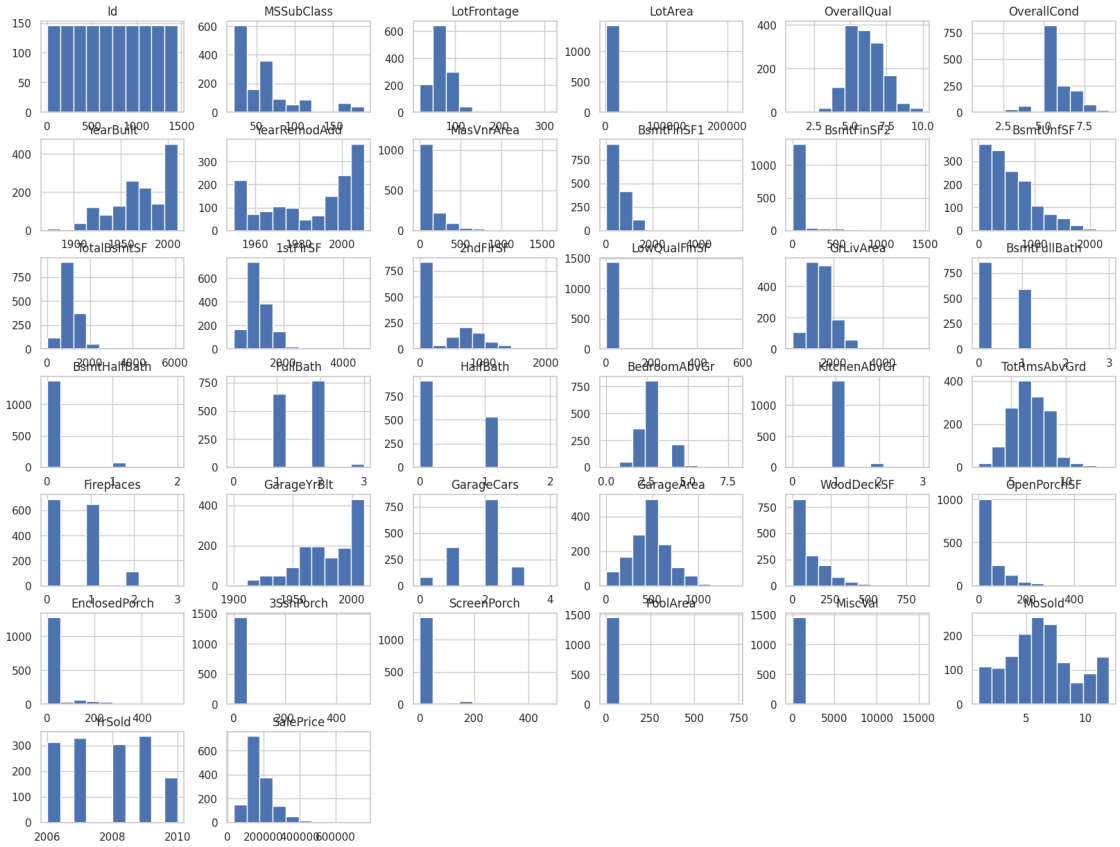
0.3.2 Exercício 2: Gerar os histogramas

Visualize a distribuição dos atributos numéricos. Use o parâmetro `figsize=(20,15)` do método `hist` para melhorar a visualização. É possível identificar fortes assimetrias e valores truncados?

[74]:
```
print("\nHistogramas dos atributos numéricos:")
df.hist(figsize=(20, 15))
```

```
Histogramas dos atributos numéricos:
```

[74]:
```
array([[<Axes: title={'center': 'Id'}>,
        <Axes: title={'center': 'MSSubClass'}>,
        <Axes: title={'center': 'LotFrontage'}>,
        <Axes: title={'center': 'LotArea'}>,
        <Axes: title={'center': 'OverallQual'}>,
        <Axes: title={'center': 'OverallCond'}>],
       [<Axes: title={'center': 'YearBuilt'}>,
        <Axes: title={'center': 'YearRemodAdd'}>,
        <Axes: title={'center': 'MasVnrArea'}>,
        <Axes: title={'center': 'BsmtFinSF1'}>,
        <Axes: title={'center': 'BsmtFinSF2'}>,
        <Axes: title={'center': 'BsmtUnfSF'}>],
       [<Axes: title={'center': 'TotalBsmtSF'}>,
        <Axes: title={'center': '1stFlrSF'}>,
        <Axes: title={'center': '2ndFlrSF'}>,
        <Axes: title={'center': 'LowQualFinSF'}>,
        <Axes: title={'center': 'GrLivArea'}>,
        <Axes: title={'center': 'BsmtFullBath'}>],
       [<Axes: title={'center': 'BsmtHalfBath'}>,
        <Axes: title={'center': 'FullBath'}>,
        <Axes: title={'center': 'HalfBath'}>,
        <Axes: title={'center': 'BedroomAbvGr'}>,
        <Axes: title={'center': 'KitchenAbvGr'}>,
        <Axes: title={'center': 'TotRmsAbvGrd'}>],
       [<Axes: title={'center': 'Fireplaces'}>,
        <Axes: title={'center': 'GarageYrBlt'}>,
        <Axes: title={'center': 'GarageCars'}>,
        <Axes: title={'center': 'GarageArea'}>,
        <Axes: title={'center': 'WoodDeckSF'}>,
        <Axes: title={'center': 'OpenPorchSF'}>],
       [<Axes: title={'center': 'EnclosedPorch'}>,
        <Axes: title={'center': '3SsnPorch'}>,
        <Axes: title={'center': 'ScreenPorch'}>,
        <Axes: title={'center': 'PoolArea'}>,
        <Axes: title={'center': 'MiscVal'}>,
        <Axes: title={'center': 'MoSold'}>],
       [<Axes: title={'center': 'YrSold'}>,
        <Axes: title={'center': 'SalePrice'}>, <Axes: >, <Axes: >,
        <Axes: >, <Axes: >]], dtype=object)
```



0.3.3 Exercício 3: Listar os atributos mais correlacionados com o target

Calcule a correlação de Pearson de todos os atributos em relação ao atributo alvo (`SalePrice`) e exiba os atributos em ordem decrescente de correlação. Lembre de usar o parâmetro `numeric_only=True` no método `corr`, para evitar o erro provocado ao tentar calcular correlação envolvendo atributos categóricos.

[75]:
```
# SEU CÓDIGO AQUI
target_name = 'SalePrice' # Define o nome do atributo alvo

print(f"\nCorrelação de Pearson com o target ({target_name}):")

# 1. Calcular a matriz de correlação (apenas para atributos numéricos)
# Usa o parâmetro numeric_only=True conforme solicitado
correlation_matrix = df.corr(numeric_only=True)

# 2. Selecionar a coluna de correlação com o atributo alvo (SalePrice)
target_correlation = correlation_matrix[target_name]

# 3. Filtrar o próprio alvo (que tem correlação 1.0) e obter a ordem decrescente
# O .sort_values(ascending=False) coloca o maior valor de correlação no topo
```

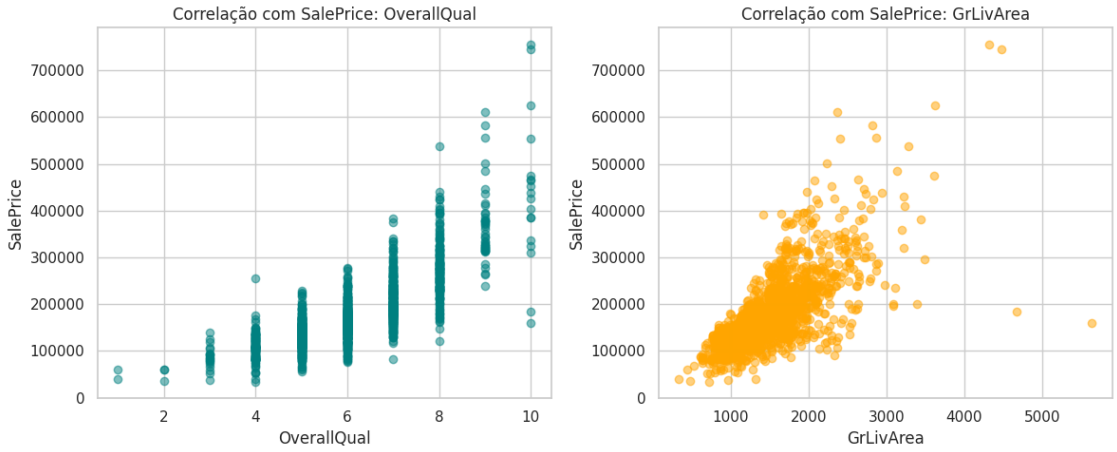
```
sorted_correlations = target_correlation.drop(target_name).
  ↪sort_values(ascending=False)

# 4. Exibir o resultado
print(sorted_correlations)# SEU CÓDIGO AQUI
```

```
Correlação de Pearson com o target (SalePrice):
OverallQual      0.790982
GrLivArea        0.708624
GarageCars       0.640409
GarageArea       0.623431
TotalBsmtSF      0.613581
1stFlrSF         0.605852
FullBath         0.560664
TotRmsAbvGrd     0.533723
YearBuilt        0.522897
YearRemodAdd     0.507101
GarageYrBlt      0.486362
MasVnrArea       0.477493
Fireplaces       0.466929
BsmtFinSF1       0.386420
LotFrontage      0.351799
WoodDeckSF       0.324413
2ndFlrSF         0.319334
OpenPorchSF      0.315856
HalfBath         0.284108
LotArea          0.263843
BsmtFullBath     0.227122
BsmtUnfSF        0.214479
BedroomAbvGr     0.168213
ScreenPorch      0.111447
PoolArea         0.092404
MoSold           0.046432
3SsnPorch        0.044584
BsmtFinSF2      -0.011378
BsmtHalfBath    -0.016844
MiscVal         -0.021190
Id              -0.021917
LowQualFinSF    -0.025606
YrSold          -0.028923
OverallCond     -0.077856
MSSubClass      -0.084284
EnclosedPorch   -0.128578
KitchenAbvGr    -0.135907
Name: SalePrice, dtype: float64
```

Faça o scatter plot do atributo de maior correlação (em módulo) e do atributo `SalePrice`. Faça

também um scatter plot para o segundo atributo de maior correlação (em módulo) e o atributo `SalePrice`. É possível observar uma relação aproximadamente linear?

[76]:
```
import matplotlib.pyplot as plt

# 5. Selecionar os dois atributos com maior correlação (em módulo)
top1_feature = sorted_correlations.index[0]
top2_feature = sorted_correlations.index[1]

# 6. Gerar os scatter plots
plt.figure(figsize=(12, 5))

# Scatter plot do atributo mais correlacionado
plt.subplot(1, 2, 1)
plt.scatter(df[top1_feature], df[target_name], alpha=0.5, color='teal')
plt.xlabel(top1_feature)
plt.ylabel(target_name)
plt.title(f"Correlação com {target_name}: {top1_feature}")

# Scatter plot do segundo atributo mais correlacionado
plt.subplot(1, 2, 2)
plt.scatter(df[top2_feature], df[target_name], alpha=0.5, color='orange')
plt.xlabel(top2_feature)
plt.ylabel(target_name)
plt.title(f"Correlação com {target_name}: {top2_feature}")

plt.tight_layout()
plt.show()
```



0.4 4. Preparação dos Dados

0.4.1 Exercício 4: Separar os conjuntos de treino e teste

Para melhorar a eficiência dos experimentos, vamos usar apenas os atributos `OverallQual`, `GrLivArea`, `Neighborhood` e `GarageCars` (código fornecido abaixo).

Para garantir que a distribuição de preços seja semelhante nos conjuntos de treino e teste, vamos criar uma categoria de preços para usar na amostragem estratificada (código fornecido abaixo).

Em seguida, faça a divisão das instâncias em `train_set` e `test_set` usando a função `train_test_split`, deixando 20% das instâncias no conjunto de teste, e usando `random_state=42`.

Vamos chamar as features do conjunto de treino de `housing_features`, e seus rótulos de `housing_labels` (código fornecido abaixo).

[77]:
```
# Seleciona apenas os atributos relevantes
df = df[["OverallQual", "GrLivArea", "Neighborhood", "GarageCars",␣
 ↪"SalePrice"]].copy()

# Cria a coluna de categoria de preço para estratificação
df["price_cat"] = pd.cut(df["SalePrice"],
                         bins=[0., 100000, 150000, 200000, 300000, np.inf],
                         labels=[1, 2, 3, 4, 5])


train_set, test_set = train_test_split(df, test_size=0.2, random_state=42,␣
 ↪stratify=df["price_cat"])

# Remove a coluna auxiliar
for set_ in (train_set, test_set):
    set_.drop("price_cat", axis=1, inplace=True)

# Define features e rótulos
housing_features = train_set.drop("SalePrice", axis=1)
housing_labels = train_set["SalePrice"].copy()
```

0.4.2 Exercício 5: Criar um Pipeline de Pré-processamento Completo

Crie um `ColumnTransformer` (chamado `preprocessor`) que: - Preencha os valores faltantes dos atributos numéricos com a mediana e os padronize. - Preencha os valores faltantes dos atributos categóricos com o valor mais frequente (`SimpleImputer(strategy='most_frequent')`) e aplique One-Hot encoding.

Em seguida, aplicamos esta transformação nas features (`housing_features`), e armazenamos o resultado em `housing_prepared` (código fornecido abaixo).

[78]:
```
# Identifica os tipos de atributos
num_attribs = ["OverallQual", "GrLivArea", "GarageCars"]
cat_attribs = ["Neighborhood"]
```

```
# Pipeline para atributos numéricos
num_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

# Pipeline para atributos categóricos
cat_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])

# ColumnTransformer combinando os dois pipelines
preprocessor = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", cat_pipeline, cat_attribs)
])

# Aplica a transformação
housing_prepared = preprocessor.fit_transform(housing_features)
```

0.5 5. Seleção e Treinamento de Modelos

0.5.1 Exercício 6: Treinar e avaliar modelos de base

Treine os modelos `LinearRegression`, `DecisionTreeRegressor` e `RandomForestRegressor` com parâmetros default e `random_state=42` e imprima o RMSE de cada um no conjunto de treino.

[79]:
```
# Treinamento dos modelos
lin_reg = LinearRegression()
tree_reg = DecisionTreeRegressor(random_state=42)
forest_reg = RandomForestRegressor(random_state=42)

lin_reg.fit(housing_prepared, housing_labels)
tree_reg.fit(housing_prepared, housing_labels)
forest_reg.fit(housing_prepared, housing_labels)

# Previsões no conjunto de treino
lin_preds = lin_reg.predict(housing_prepared)
tree_preds = tree_reg.predict(housing_prepared)
forest_preds = forest_reg.predict(housing_prepared)

# Cálculo do RMSE
lin_rmse = np.sqrt(mean_squared_error(housing_labels, lin_preds))
tree_rmse = np.sqrt(mean_squared_error(housing_labels, tree_preds))
forest_rmse = np.sqrt(mean_squared_error(housing_labels, forest_preds))
```

```
# Exibição dos resultados
print(f"Linear Regression RMSE: {lin_rmse:.2f}")
print(f"Decision Tree RMSE: {tree_rmse:.2f}")
print(f"Random Forest RMSE: {forest_rmse:.2f}")
```

```
Linear Regression RMSE: 35254.91
Decision Tree RMSE: 2465.67
Random Forest RMSE: 13196.16
```

0.5.2 Exercício 7: Validação cruzada para o RandomForestRegressor

Avalie o `RandomForestRegressor` (com parâmetros default, e `random_state=42`) usando validação cruzada com 10 folds para ter uma estimativa mais robusta de seu desempenho. Qual a média e desvio padrão do RMSE?

[80]:
```
# Modelo com parâmetros padrão
forest_reg = RandomForestRegressor(random_state=42)

# Validação cruzada com scoring negativo do MSE
scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)

# Converte para RMSE
rmse_scores = np.sqrt(-scores)

# Exibe os resultados
print("RMSE por fold:", rmse_scores)
print(f"Média do RMSE: {rmse_scores.mean():.2f}")
print(f"Desvio padrão do RMSE: {rmse_scores.std():.2f}")
```

```
RMSE por fold: [35480.79669454 31793.8360906  34002.17498559 26642.53198639
 31021.77910471 36676.36066141 32218.38285276 40534.07772505
 46470.13928305 33203.69695214]
Média do RMSE: 34804.38
Desvio padrão do RMSE: 5222.66
```

0.6 6. Ajuste Fino e Avaliação Final

0.6.1 Exercício 8: Otimização com GridSearchCV

Use `GridSearchCV` para encontrar os melhores hiperparâmetros para o `RandomForestRegressor`. Teste os valores 20, 30 e 50 para o parâmetro `n_estimators`. Use 5 folds na validação cruzada.

Qual o melhor RMSE encontrado? Para qual valor do parâmetro `n_estimators`?

[81]:
```
# Define o modelo base
forest_reg = RandomForestRegressor(random_state=42)

# Define os hiperparâmetros a testar
```

```
param_grid = {
    "n_estimators": [20, 30, 50]
}

# Configura o GridSearchCV
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring="neg_mean_squared_error",
                           return_train_score=True)

# Executa a busca
grid_search.fit(housing_prepared, housing_labels)

# Extrai o melhor modelo e RMSE
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_
best_rmse = np.sqrt(-grid_search.best_score_)

# Exibe os resultados
print(f"Melhor RMSE: {best_rmse:.2f}")
print(f"Melhor valor de n_estimators: {best_params['n_estimators']}")
```

```
Melhor RMSE: 35381.42
Melhor valor de n_estimators: 30
```

0.6.2 Exercício 9: Avaliar o melhor modelo no conjunto de teste

Finalmente, avalie no conjunto de teste o desempenho do melhor modelo obtido na otimização de hiparparâmetros. Lembre de aplicar o pré-processamento nos atributos do conjunto de teste.

Qual RMSE obtido?

[82]:
```
# Aplica o pré-processamento nas features do conjunto de teste
X_test_prepared = preprocessor.transform(test_set.drop("SalePrice", axis=1))
y_test = test_set["SalePrice"].copy()

# Faz previsões com o melhor modelo encontrado no GridSearchCV
final_predictions = best_model.predict(X_test_prepared)

# Calcula o RMSE no conjunto de teste
final_rmse = np.sqrt(mean_squared_error(y_test, final_predictions))

# Exibe o resultado
print(f"RMSE no conjunto de teste: {final_rmse:.2f}")
```

```
RMSE no conjunto de teste: 33369.64
```

Exercício Prático 02_ Classificação

October 29, 2025

1 Exercício Prático: Classificação com Múltiplas Métricas (Capítulo 3)

Este notebook contém um exercício prático para aplicar os conceitos de classificação, avaliação de modelos e análise de erros abordados em aula. Utilizaremos o dataset "Wine", um conjunto de dados público e pequeno com múltiplas classes.

[1]:
```
# Execute esta célula para configurar o ambiente do notebook
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪cross_val_predict
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDClassifier
from sklearn.dummy import DummyClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,␣
 ↪precision_score, recall_score, f1_score, precision_recall_curve, roc_curve,␣
 ↪roc_auc_score

# --- Carregando o Dataset ---
# O dataset "Wine" possui 178 amostras de vinhos,
# cada uma com 13 características químicas.
# O objetivo é classificar os vinhos em uma de 3 classes (cultivares).
wine = load_wine()
X = wine.data
y = wine.target

# Separação dos conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Normalizando os dados para melhor desempenho do SGDClassifier
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
print("Dataset 'Wine' carregado e pronto.")
print(f"Características do treino: {X_train_scaled.shape}")
print(f"Rótulos do treino: {y_train.shape}")
```

```
Dataset 'Wine' carregado e pronto.
Características do treino: (142, 13)
Rótulos do treino: (142,)
```

1.1 Parte 1: Classificação Binária

Nesta seção, vamos adaptar o problema para uma tarefa de classificação binária, onde o objetivo será apenas identificar se um vinho pertence à classe 0 ou não.

1.1.1 1.1 - Adaptar o Dataset para 2 Classes

Sua tarefa: Crie as variáveis `y_train_0` e `y_test_0`. Elas devem conter `True` para as amostras que são da classe 0 e `False` para as outras. Dica: Use uma comparação booleana com `y_train` e `y_test`.

[2]:
```
# ASSUMINDO QUE y_train e y_test JÁ FORAM DEFINIDOS
# E contêm os rótulos originais das classes (e.g., 0, 1, 2, ...)

# SEU CÓDIGO AQUI
y_train_0 = (y_train == 0)
y_test_0 = (y_test == 0)

print("Primeiros 5 labels do y_train_0:", y_train_0[:5])
```

```
Primeiros 5 labels do y_train_0: [False False False False  True]
```

1.1.2 1.2 - Treinar um Classificador Binário e Fazer uma Predição

Sua tarefa: 1. Crie e treine uma instância do `SGDClassifier`. Use `random_state=42` para reprodutibilidade. 2. Use o classificador treinado para prever a classe da primeira instância do conjunto de treino (`X_train_scaled[0]`). 3. Imprima a classe prevista e a classe real.

[3]:
```
from sklearn.linear_model import SGDClassifier

# 1. Criar e treinar o classificador
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train_scaled, y_train)

# 2. Fazer a predição da primeira instância
predicao = sgd_clf.predict([X_train_scaled[0]])

# 3. Imprimir a classe prevista e a classe real
print("Classe prevista:", predicao[0])
print("Classe real:", y_train[0])
```

```
Classe prevista: 2
Classe real: 2
```

1.2 Parte 2: Avaliação de Desempenho (Binário)

1.2.1 2.1 - Medir Acurácia com Validação Cruzada

Sua tarefa: Use `cross_val_score` para avaliar seu `sgd_clf` com 3 folds (`cv=3`) e a métrica de acurácia. Imprima as pontuações de cada fold.

[4]:
```
from sklearn.model_selection import cross_val_score

# Avaliar o classificador com validação cruzada
acuracia_scores = cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3,␣
 ↪scoring='accuracy')

# Imprimir as pontuações de cada fold
print("Acurácia em cada fold:", acuracia_scores)
```

```
Acurácia em cada fold: [0.95833333 0.95744681 0.95744681]
```

1.2.2 2.2 - Baseline com `DummyClassifier`

Sua tarefa: Faça o mesmo que no passo anterior, mas usando um `DummyClassifier` com a estratégia "most_frequent". Isso nos dará uma baseline para comparar. O SGD é melhor que o baseline?

[5]:
```
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import cross_val_score

# Criar o classificador Dummy
dummy_clf = DummyClassifier(strategy="most_frequent")

# Avaliar com validação cruzada
acuracia_dummy = cross_val_score(dummy_clf, X_train_scaled, y_train, cv=3,␣
 ↪scoring='accuracy')

# Imprimir os resultados
print("Acurácia (Dummy) em cada fold:", acuracia_dummy)
```

```
Acurácia (Dummy) em cada fold: [0.39583333 0.40425532 0.40425532]
```

1.2.3 2.3 - Matriz de Confusão, Precisão, Recall e F1-Score do SGDClassifier (binário)

Sua tarefa: 1. Obtenha as previsões para todo o conjunto de treino usando `cross_val_predict`. 2. Calcule e exiba a matriz de confusão. 3. Calcule e imprima a precisão, o recall e o F1-score.

[6]:
```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix, precision_score, recall_score,␣
 ↪f1_score
```

```
# 1. Obter previsões com validação cruzada
y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)

# 2. Matriz de confusão
matriz_confusao = confusion_matrix(y_train, y_train_pred)
print("Matriz de Confusão:\n", matriz_confusao)

# 3. Métricas para problema multiclasse
precisao = precision_score(y_train, y_train_pred, average='weighted')
recall = recall_score(y_train, y_train_pred, average='weighted')
f1 = f1_score(y_train, y_train_pred, average='weighted')

print("Precisão (weighted):", precisao)
print("Recall (weighted):", recall)
print("F1-Score (weighted):", f1)
```

```
Matriz de Confusão:
 [[44  1  0]
 [ 1 53  3]
 [ 0  1 39]]
Precisão (weighted): 0.9582403511980977
Recall (weighted): 0.9577464788732394
F1-Score (weighted): 0.9577127398537567
```

1.3 Parte 3: Curvas de Avaliação (Binário)

1.3.1 3.1 - Obter Scores de Decisão

Sua tarefa: Use `cross_val_predict` novamente, mas desta vez para obter os *scores de decisão* (`decision_function`) em vez das previsões de classe.

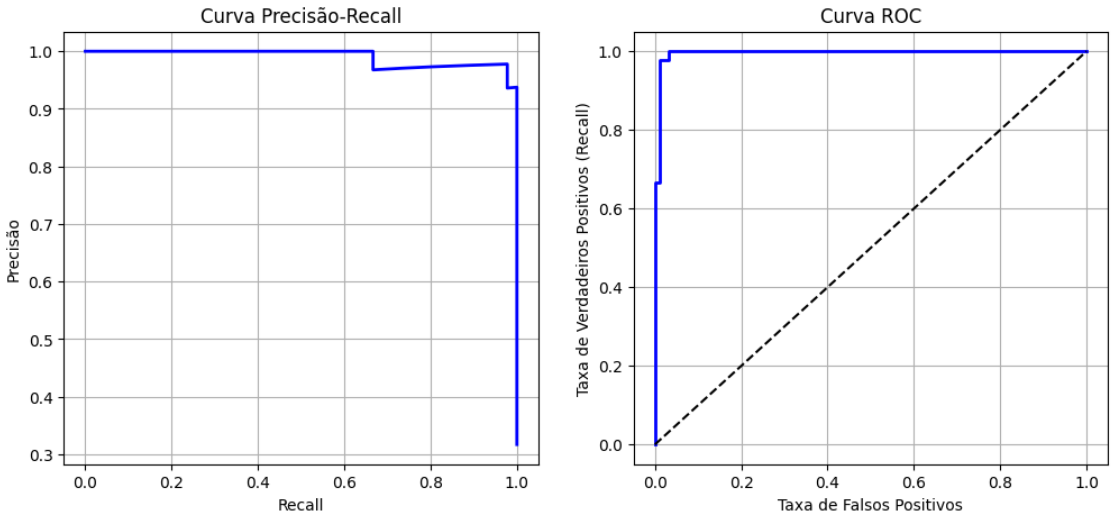
[7]:
```
from sklearn.model_selection import cross_val_predict

# Obter scores de decisão com validação cruzada
y_scores = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3,␣
 ↪method='decision_function')

# Exibir os primeiros 5 scores
print("Scores de decisão (primeiros 5):", y_scores[:5])
```

```
Scores de decisão (primeiros 5): [[ -65.6785192  -185.76708802   98.87998731]
 [-144.45010382  -23.84815242   54.19254207]
 [ -66.99672564   62.36970305 -161.67296785]
 [-101.73649127 -144.45876737   88.67256483]
 [  29.56970163   -4.297556   -130.12312568]]
```

1.3.2 3.2 - Plotar Curvas de Precisão-Recall e ROC

Sua tarefa: 1. Use a função `precision_recall_curve` para obter as precisões, recalls e limiares. 2. Plote a curva de Precisão vs. Recall. 3. Use a função `roc_curve` para obter a taxa de falsos positivos (fpr) e a taxa de verdadeiros positivos (tpr). 4. Plote a curva ROC. 5. Calcule e imprima a Área Sob a Curva ROC (AUC).

[8]:
```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import precision_recall_curve, roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Criar vetor binário para a classe 0
y_train_0 = (y_train == 0)

# Obter scores de decisão para todas as classes (matriz)
y_scores_all = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3,␣
 ↪method='decision_function')

# Extrair os scores da classe 0 (coluna correspondente)
y_scores_0 = y_scores_all[:, 0]

# Curva Precisão-Recall
precisions, recalls, thresholds_pr = precision_recall_curve(y_train_0,␣
 ↪y_scores_0)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(recalls, precisions, "b-", linewidth=2)
plt.xlabel("Recall")
plt.ylabel("Precisão")
plt.title("Curva Precisão-Recall")
plt.grid(True)

# Curva ROC
fpr, tpr, thresholds_roc = roc_curve(y_train_0, y_scores_0)

plt.subplot(1, 2, 2)
plt.plot(fpr, tpr, "b-", linewidth=2, label="SGD")
plt.plot([0, 1], [0, 1], 'k--')  # Linha de referência
plt.xlabel("Taxa de Falsos Positivos")
plt.ylabel("Taxa de Verdadeiros Positivos (Recall)")
plt.title("Curva ROC")
plt.grid(True)
plt.show()

# Área Sob a Curva ROC (AUC)
roc_auc = roc_auc_score(y_train_0, y_scores_0)
print(f"Área Sob a Curva ROC (AUC): {roc_auc:.4f}")
```




```
Área Sob a Curva ROC (AUC): 0.9961
```

1.4 Parte 4: Classificação Multiclasse e Análise de Erros

1.4.1 4.1 - Treinar o Classificador Multiclasse

Sua tarefa: 1. Treine uma nova instância do `SGDClassifier` (ou use a antiga), mas desta vez usando o `y_train` original, que contém todas as 3 classes. 2. Faça uma predição para a primeira instância do conjunto de treino e imprima a classe prevista e a real.

[9]:
```python
from sklearn.linear_model import SGDClassifier

# 1. Treinar o classificador com todas as 3 classes
sgd_clf_multiclasse = SGDClassifier(random_state=42)
sgd_clf_multiclasse.fit(X_train_scaled, y_train)

# 2. Fazer a predição da primeira instância
predicao_multiclasse = sgd_clf_multiclasse.predict([X_train_scaled[0]])

# Imprimir a classe prevista e a classe real
print("Classe prevista:", predicao_multiclasse[0])
print("Classe real:", y_train[0])
```

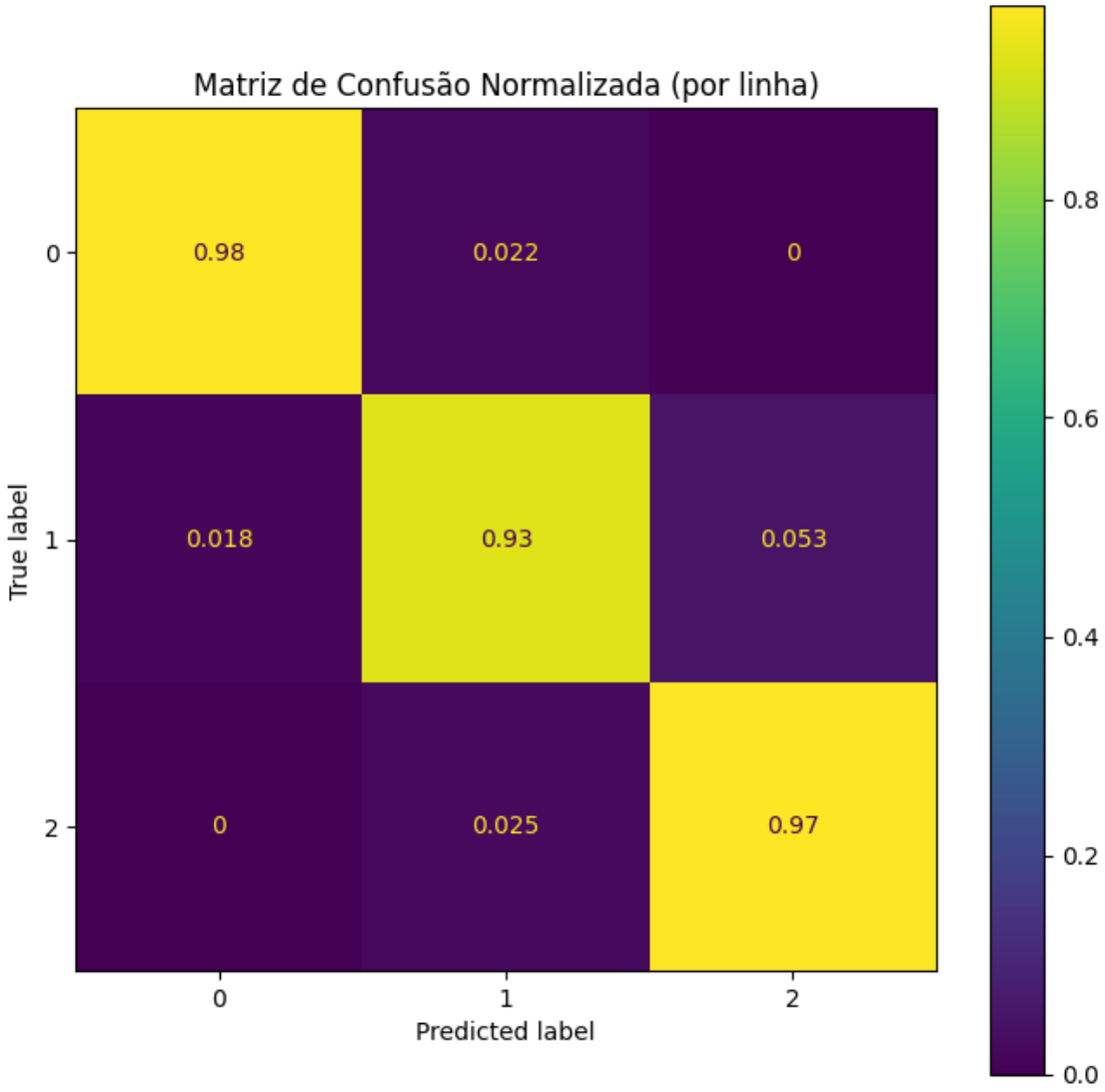
```
Classe prevista: 2
Classe real: 2
```

1.4.2 4.2 - Análise de Erros com Matriz de Confusão

Sua tarefa: 1. Use `cross_val_predict` para obter as previsões no conjunto de treino para o modelo multiclasse. 2. Plote a matriz de confusão normalizada (use

`ConfusionMatrixDisplay.from_predictions` com `normalize='true'`). 3. Com base na matriz, identifique quais classes o modelo mais confunde.

[10]:
```
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.model_selection import cross_val_predict
import matplotlib.pyplot as plt

# 1. Obter previsões com validação cruzada
y_train_pred_multi = cross_val_predict(sgd_clf_multiclasse, X_train_scaled,␣
 ↪y_train, cv=3)

# 2. Plotar matriz de confusão normalizada
fig, ax = plt.subplots(figsize=(8, 8))
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred_multi,␣
 ↪normalize='true', ax=ax)

plt.title("Matriz de Confusão Normalizada (por linha)")
plt.show()
```


Matriz de Confusão Normalizada (por linha)
0.98
0.022
0
0.018
0.93
0.053
0
0.025
0.97
True label
Predicted label
0
1
2
0.0
0.2
0.4
0.6
0.8

Exercício prático 03_ Regressão

October 29, 2025

1 Exercício de Laboratório: Regressão Linear com a Equação Normal

Objetivo: aplicar o método da Equação Normal para encontrar os parâmetros ótimos de um modelo de Regressão Linear e, em seguida, verificar os resultados usando a biblioteca Scikit-Learn.

Você irá: 1. Gerar dados sintéticos com uma tendência linear decrescente. 2. Usar a Equação Normal e o NumPy para encontrar os melhores parâmetros para o seu modelo de Regressão Linear. 3. Fazer previsões e plotar a reta de regressão. 4. Verificar seus resultados usando a classe `LinearRegression` do Scikit-Learn.

1.0.1 Passo 1: Importar Bibliotecas

A célula de código abaixo importa todas as bibliotecas que usaremos neste exercício: `NumPy` para cálculos numéricos, `Matplotlib` para plotagem e `Scikit-Learn` para o modelo de Regressão Linear.

[38]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

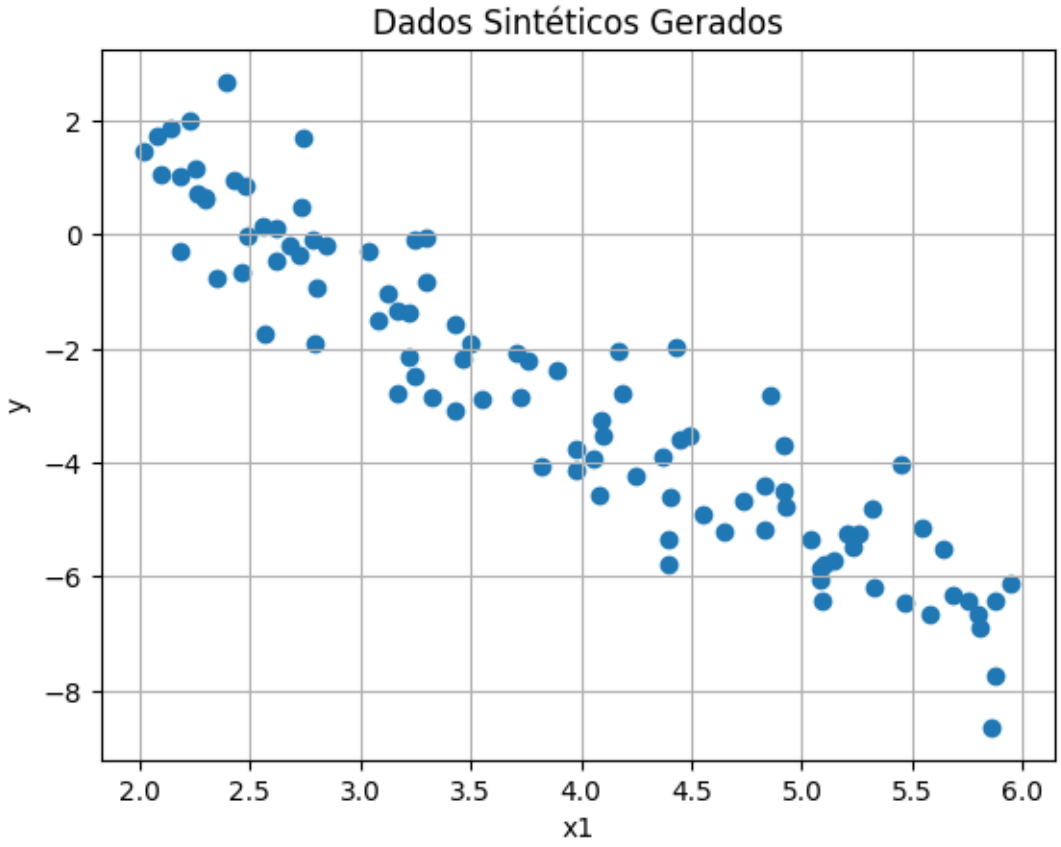
1.0.2 Passo 2: Gerar Dados Sintéticos

No livro, os dados foram gerados com a equação `y = 4 + 3x + ruído`. Para este exercício, vamos usar uma equação diferente para criar dados com uma correlação negativa (uma reta decrescente). Além disso, os valores do atributo vão de 2 até 6.

Sua Tarefa: Complete a célula de código a seguir para gerar 100 pontos de dados usando a equação **`y = 5 - 2x + ruído Gaussiano`**.

[39]:
```
import numpy as np
import matplotlib.pyplot as plt

# Para garantir que seja possível reproduzir os resultados
np.random.seed(42)

# Gere 100 instâncias
m = 100

# ############################################################ #
```

```
# Gera vetor de características X com valores entre 2 e 6
X = 2 + 4 * np.random.rand(m)

# Gera ruído Gaussiano
noise = np.random.randn(m)

# Gera vetor de rótulos y usando a equação y = 5 - 2x + ruído
y = 5 - 2 * X + noise
# ################################################################ #

# Vamos plotar os dados para visualizá-los
plt.scatter(X, y)
plt.xlabel("x1")
plt.ylabel("y")
plt.title("Dados Sintéticos Gerados")
plt.grid(True)
plt.show()
```



1.0.3 Passo 3: Encontrar Parâmetros com a Equação Normal

Agora, você irá calcular os parâmetros ótimos ($\hat{\theta}$) para o seu modelo. Conforme visto na aula, para encontrar o valor de θ que minimiza a função de custo, existe uma solução de forma fechada conhecida como a Equação Normal:

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

Sua Tarefa: 1. Adicione o termo de viés ($x_0 = 1$) a cada instância no seu conjunto de dados `X`. Isso é necessário para que a Equação Normal calcule o intercepto (θ_0). 2. Implemente a Equação Normal usando as operações de álgebra linear do NumPy para encontrar `theta_best`.

[40]:
```
import numpy as np

# 1. Adiciona o termo de viés (x0 = 1) a cada instância
X_b = np.c_[np.ones((m, 1)), X.reshape(-1, 1)]  # X.reshape para garantir␣
 ↪formato coluna

# 2. Aplica a Equação Normal
theta_best = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y.reshape(-1, 1)

# Exibe os parâmetros encontrados
print("Parâmetros encontrados com a Equação Normal:")
print(f"Theta 0 (Intercepto): {theta_best[0][0]:.2f}")
print(f"Theta 1 (Coeficiente): {theta_best[1][0]:.2f}")
```

```
Parâmetros encontrados com a Equação Normal:
Theta 0 (Intercepto): 5.44
Theta 1 (Coeficiente): -2.11
```

1.0.4 Passo 4: Fazer Previsões e Plotar a Reta de Regressão

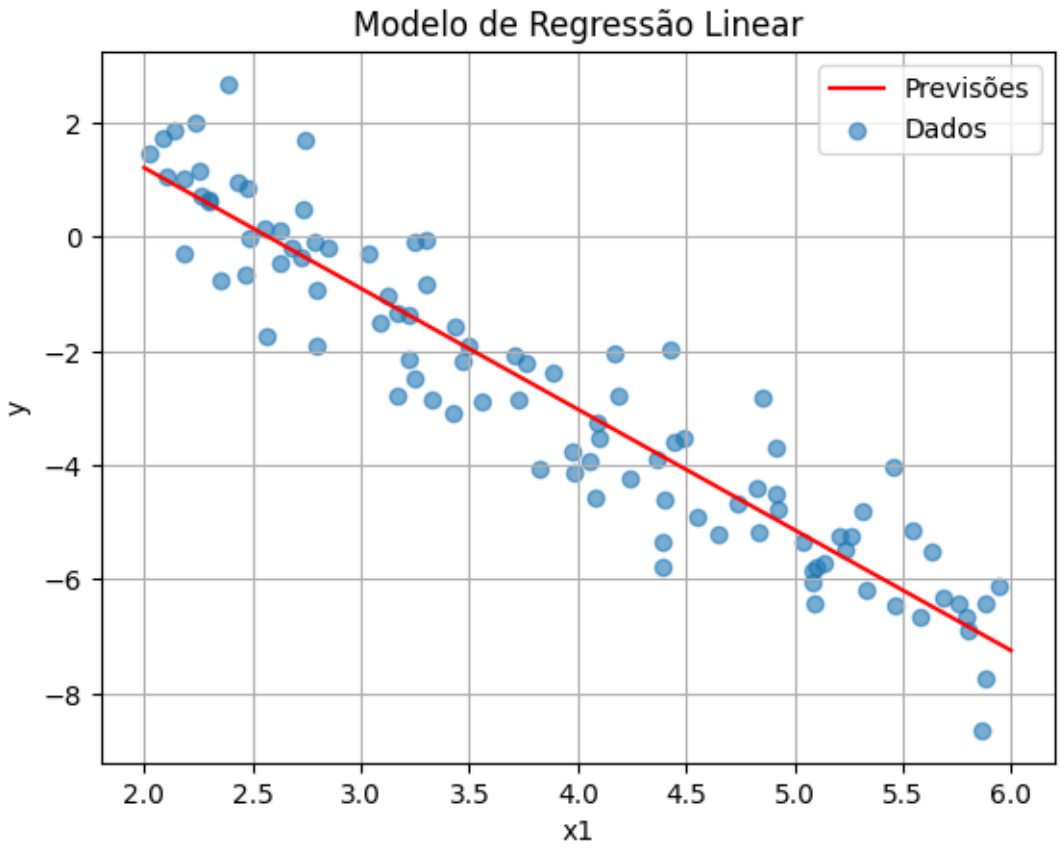
Com os parâmetros em `theta_best`, seu modelo está treinado! Agora você pode usá-lo para fazer previsões em novos dados.

Sua Tarefa: 1. Crie um novo conjunto de dados `X_new` para prever os valores de y quando x=2 e x=6. 2. Adicione o termo de viés a `X_new`. 3. Calcule as previsões `y_predict` usando `theta_best`. 4. Plote a reta de regressão resultante sobre os dados originais para visualizar o ajuste do modelo.

[41]:
```
# 1. Crie X_new para fazer previsões
X_new = np.array([[2], [6]])

# 2. Adicione o termo de viés a X_new para criar X_new_b
X_new_b = np.c_[np.ones((2, 1)), X_new]

# 3. Calcule as previsões
y_predict = X_new_b @ theta_best

# Exiba as previsões
```

```
print("Previsões para X_new:")
print(y_predict)

# Plote a reta de regressão
plt.plot(X_new, y_predict, "r-", label="Previsões")  # reta de regressão
plt.scatter(X, y, alpha=0.6, label="Dados")          # dados originais
plt.xlabel("x1")
plt.ylabel("y")
plt.title("Modelo de Regressão Linear")
plt.legend()
plt.grid(True)
plt.show()
```

```
Previsões para X_new:
[[ 1.21509616]
 [-7.24467707]]
```



1.0.5 Passo 5: Verificar com Scikit-Learn

Realizar a Regressão Linear usando o Scikit-Learn é muito mais direto. A classe `LinearRegression` lida com o termo de viés automaticamente e usa um método de otimização computacionalmente mais eficiente (SVD) nos bastidores.

Sua Tarefa: 1. Crie e treine uma instância da classe `LinearRegression`. 2. Imprima o `intercept_` (intercepto) e o `coef_` (coeficiente) do modelo treinado. Os valores devem ser praticamente idênticos aos que você calculou com a Equação Normal.

[42]:
```
from sklearn.linear_model import LinearRegression

# 1. Crie o modelo de Regressão Linear
lin_reg = LinearRegression()

# 2. Treine o modelo com os dados originais (X precisa estar em formato 2D)
lin_reg.fit(X.reshape(-1, 1), y)

# 3. Faça previsões para X_new
y_predict = lin_reg.predict(X_new)

# Exiba os parâmetros encontrados pelo Scikit-Learn
print("Parâmetros encontrados com Scikit-Learn:")
print(f"Intercepto: {lin_reg.intercept_:.2f}")
print(f"Coeficiente: {lin_reg.coef_[0]:.2f}")

# Exiba as previsões
print("Previsões para X_new com Scikit-Learn:")
print(y_predict)
```

```
Parâmetros encontrados com Scikit-Learn:
Intercepto: 5.44
Coeficiente: -2.11
Previsões para X_new com Scikit-Learn:
[ 1.21509616 -7.24467707]
```

Conclusão

Se tudo correu bem, os parâmetros que você calculou manualmente usando a Equação Normal devem corresponder exatamente aos parâmetros encontrados pela biblioteca Scikit-Learn. Isso demonstra como o Scikit-Learn simplifica o processo (e torna-o mais eficiente pelo uso da pseudo-inversa), mas entender a matemática subjacente, como a Equação Normal, é fundamental para fazer boas escolher no projeto de sistemas de aprendizado de máquina.

Exercício prático 04_ Gradiente Descendente

October 29, 2025

1 Exercício de Programação: Gradiente Descendente

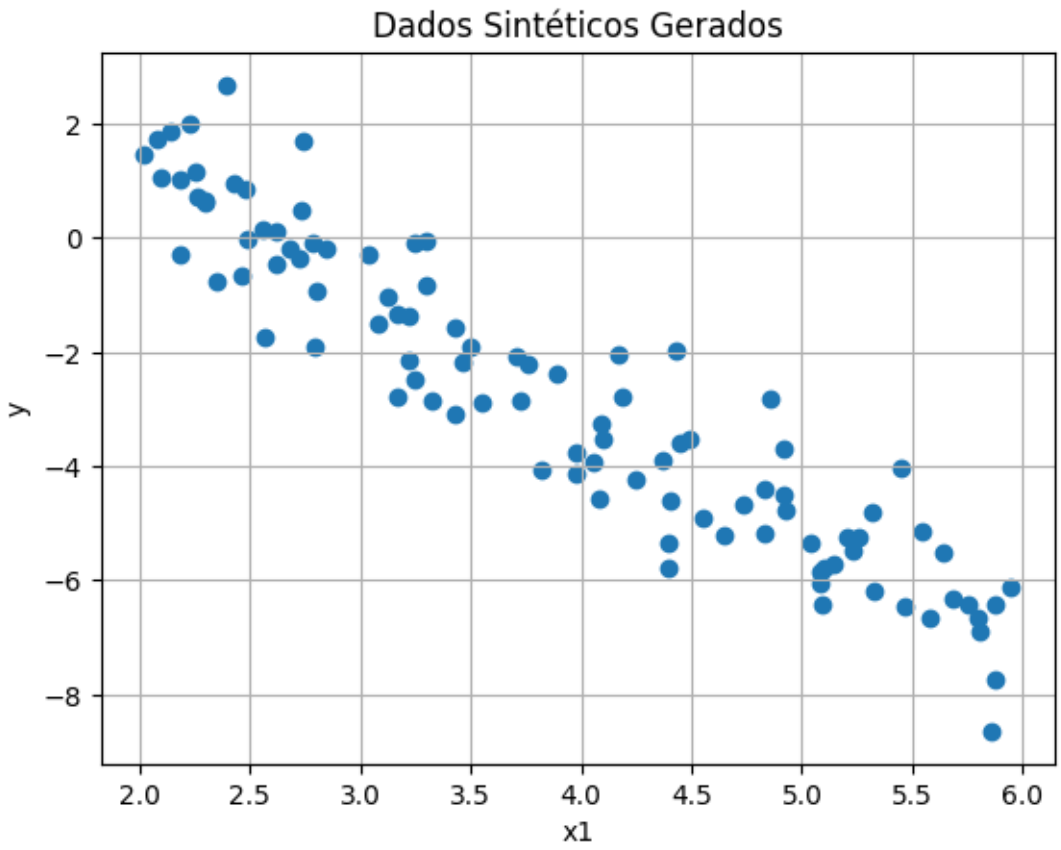
Neste notebook, você implementará e explorará os algoritmos de Gradiente Descendente em Lote (Batch) e Estocástico (Stochastic) para treinar um modelo de Regressão Linear.

1.1 Parte 1: Gerar Dados Sintéticos

Aqui você pode aproveitar a geração de dados sintéticos que utilizou no exercício anterior (Regressão Linear).

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# Para garantir que seja possível reproduzir os resultados
np.random.seed(42)

# Gere 100 instâncias
m = 100

# ################################################################ #
X = 2 + 4 * np.random.rand(m, 1)
ruido = np.random.randn(m, 1)
y = 5 - 2 * X + ruido


# ################################################################ #

# Vamos plotar os dados para visualizá-los
plt.scatter(X, y)
plt.xlabel("x1")
plt.ylabel("y")
plt.title("Dados Sintéticos Gerados")
plt.grid(True)
plt.show()
```



1.2 Parte 2: Gradiente Descendente em Lote (Batch GD)

O Gradiente Descendente em Lote calcula os gradientes com base em todo o conjunto de treinamento a cada passo. A fórmula para o vetor de gradiente da função de custo MSE é:

$$\nabla_\theta \text{MSE}(\theta) = \frac{2}{m} X^T (X\theta - y)$$

E o passo de atualização dos pesos é:

$$\theta^{(\text{próximo passo})} = \theta - \eta \nabla_\theta \text{MSE}(\theta)$$

Sua tarefa: Adicione o termo de viés e implemente o Batch GD.

[3]:
```
eta = 0.05
n_epochs = 50
t0, t1 = 5, 50

def learning_schedule(t):
    return t0 / (t + t1)
```

```
np.random.seed(42)
theta = np.random.randn(2, 1)  # inicialização aleatória

m = len(X)
X_b = np.c_[np.ones((m, 1)), X]  # adiciona o termo de viés

for epoch in range(n_epochs):
    for iteration in range(m):
        random_index = np.random.randint(m)
        xi = X_b[random_index : random_index + 1]
        yi = y[random_index : random_index + 1]
        gradients = 2 * xi.T @ (xi @ theta - yi)
        eta = learning_schedule(epoch * m + iteration)
        theta = theta - eta * gradients

print(f"Theta final (solução): {theta.ravel()}")
```

```
Theta final (solução): [21.35344567 -5.96750236]
```

1.2.1 Explorando a Taxa de Aprendizagem (eta)

Agora, vamos observar como os parâmetros `theta` mudam após 200 iterações com diferentes taxas de aprendizagem. Isso nos ajuda a entender a importância desse hiperparâmetro. O que podemos concluir?

[4]:
```
learning_rates = [0.01, 0.05, 0.10]

m = len(X)
X_b = np.c_[np.ones((m, 1)), X]  # adiciona o termo de viés

for eta in learning_rates:
    np.random.seed(42)
    theta = np.random.randn(2, 1)
    for iteration in range(200):
        gradients = 2 / m * X_b.T @ (X_b @ theta - y)
        theta = theta - eta * gradients

    print(f"Para eta = {eta}, theta após 200 iterações: {theta.ravel()}")
```

```
Para eta = 0.01, theta após 200 iterações: [ 1.7314504  -1.23520356]
Para eta = 0.05, theta após 200 iterações: [ 4.42960373 -1.87439894]
Para eta = 0.1, theta após 200 iterações: [1.03483014e+78 4.36819555e+78]
```

1.3 Parte 3: Gradiente Descendente Estocástico (Stochastic GD)

O Gradiente Descendente Estocástico acelera o processo calculando os gradientes com base em uma única instância aleatória a cada passo. Devido à sua natureza aleatória, é comum usar um **agendamento de aprendizagem** para diminuir gradualmente a taxa de aprendizagem.

Neste exercício, usaremos a função de agendamento: `learning_rate = 5 / (t + 500)`, onde `t` é o número da iteração.

Sua tarefa: Complete o código abaixo para implementar o Stochastic GD. Compare com o resultado do Batch GD. Note que executamos apenas 100 épocas.

[5]:
```
n_epochs = 100
t0, t1 = 5, 500  # hiperparâmetros do agendamento de aprendizagem

def learning_schedule(t):
    return t0 / (t + t1)

np.random.seed(42)
theta_solution_sgd = np.random.randn(2, 1)  # inicialização aleatória

m = len(X)
X_b = np.c_[np.ones((m, 1)), X]  # adiciona o termo de viés

t = 0  # contador de iterações

for epoch in range(n_epochs):
    for i in range(m):
        random_index = np.random.randint(m)
        xi = X_b[random_index:random_index + 1]
        yi = y[random_index:random_index + 1]
        gradients = 2 * xi.T @ (xi @ theta_solution_sgd - yi)
        eta = learning_schedule(t)
        theta_solution_sgd = theta_solution_sgd - eta * gradients
        t += 1

print(f"Theta final (solução): {theta_solution_sgd.ravel()}")
```

```
Theta final (solução): [ 4.9710209  -2.00493894]
```

1.4 Parte 4: Stochastic GD com Scikit-Learn

O Scikit-Learn oferece a classe `SGDRegressor`, que implementa o Gradiente Descendente Estocástico para regressão.

Sua tarefa: Use a classe `SGDRegressor` para treinar um modelo nos mesmos dados sintéticos. Em seguida, imprima o intercepto (`intercept_`) e o coeficiente (`coef_`) encontrados. Use 0.05 para a taxa de aprendizagem inicial (eta0), 100 épocas e random_state=42.

[6]:
```
from sklearn.linear_model import SGDRegressor

# ############################################################## #
sgd_reg = SGDRegressor(max_iter=100, eta0=0.05, random_state=42,␣
 ↪learning_rate="constant")
# ############################################################## #
```

```
# O método fit espera um array 1D para y, por isso usamos .ravel()
sgd_reg.fit(X, y.ravel())

print(f"Intercepto do Scikit-learn (solução): {sgd_reg.intercept_}")
print(f"Coeficiente do Scikit-learn (solução): {sgd_reg.coef_}")
```

```
Intercepto do Scikit-learn (solução): [5.45017526]
Coeficiente do Scikit-learn (solução): [-2.22372277]
```

1.4.1 Conclusão

Compare os valores de `theta` que você encontrou com a implementação manual (tanto Batch quanto Stochastic) e com a implementação do Scikit-Learn. Eles devem ser muito próximos!

Os parâmetros da função original eram intercepto = 5 e coeficiente = -2. Nossos resultados devem estar próximos a esses valores, com pequenas variações devido ao ruído que adicionamos aos dados.

Exercício prático 06_ Modelos Lineares Regularizados

October 29, 2025

1 Exercício de Programação: Modelos Lineares Regularizados

Objetivo: Este exercício tem como objetivo aplicar e comparar diferentes técnicas de regularização para modelos de regressão linear. Você irá explorar como as penalidades ℓ_1 (Lasso), ℓ_2 (Ridge) e Elastic Net afetam o desempenho do modelo e os coeficientes das features. Além disso, você implementará a regularização por Early Stopping.

Dataset: Usaremos o dataset `California Housing`, disponível na biblioteca Scikit-learn. É um conjunto de dados pequeno e adequado para problemas de regressão, onde o objetivo é prever a mediana do preço das casas nos distritos da Califórnia com base em 8 variáveis explicativas.

1.1 1. Configuração Inicial e Carregamento dos Dados

Primeiro, vamos importar as bibliotecas necessárias e carregar o dataset. Em seguida, dividiremos os dados em três conjuntos: treinamento, validação e teste. - **Treinamento:** Usado para treinar os modelos. - **Validação:** Usado para ajustar os hiperparâmetros (como o `alpha` de regularização). - **Teste:** Usado para a avaliação final do melhor modelo, simulando dados nunca vistos.

[29]:
```
# Importação das bibliotecas
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDRegressor, ElasticNet
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.datasets import fetch_california_housing

# Carregando o dataset
housing = fetch_california_housing()

X, y = housing.data, housing.target
feature_names = housing.feature_names

# Dividindo os dados em treino+validação (80%) e teste (20%)
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.
 ↪2, random_state=42)

# Dividindo o conjunto de treino+validação em treino (75%) e validação (25%)
# Isso resulta em 60% treino, 20% validação, 20% teste do total original
```

```
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,␣
 ↪test_size=0.25, random_state=42)

# Aplicando StandardScaler globalmente
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
X_train_val_scaled = scaler.fit_transform(X_train_val)

print("Dimensões do conjunto de treino:", X_train_scaled.shape)
print("Dimensões do conjunto de validação:", X_val_scaled.shape)
print("Dimensões do conjunto de teste:", X_test_scaled.shape)
```

```
Dimensões do conjunto de treino: (12384, 8)
Dimensões do conjunto de validação: (4128, 8)
Dimensões do conjunto de teste: (4128, 8)
```

1.2 2. Modelos Regularizados: Ridge, Lasso e Elastic Net

Nesta seção, você irá treinar três tipos de modelos regularizados, variando o hiperparâmetro de regularização `alpha`. Para cada modelo, você deve: 1. Criar um loop para iterar sobre uma lista de valores de `alpha`. 2. Dentro do loop, criar um `Pipeline` que primeiro aplica `StandardScaler` (para normalizar os dados) e depois treina o modelo de regressão. 3. Treinar o pipeline com os dados de **treinamento**. 4. Fazer previsões nos dados de **validação**. 5. Calcular o Erro Quadrático Médio (MSE) e armazená-lo. 6. Após o loop, plotar o MSE de validação em função do `alpha` para encontrar o melhor valor.

1.2.1 2.1 Regressão Ridge (SGD com penalidade ℓ_2)

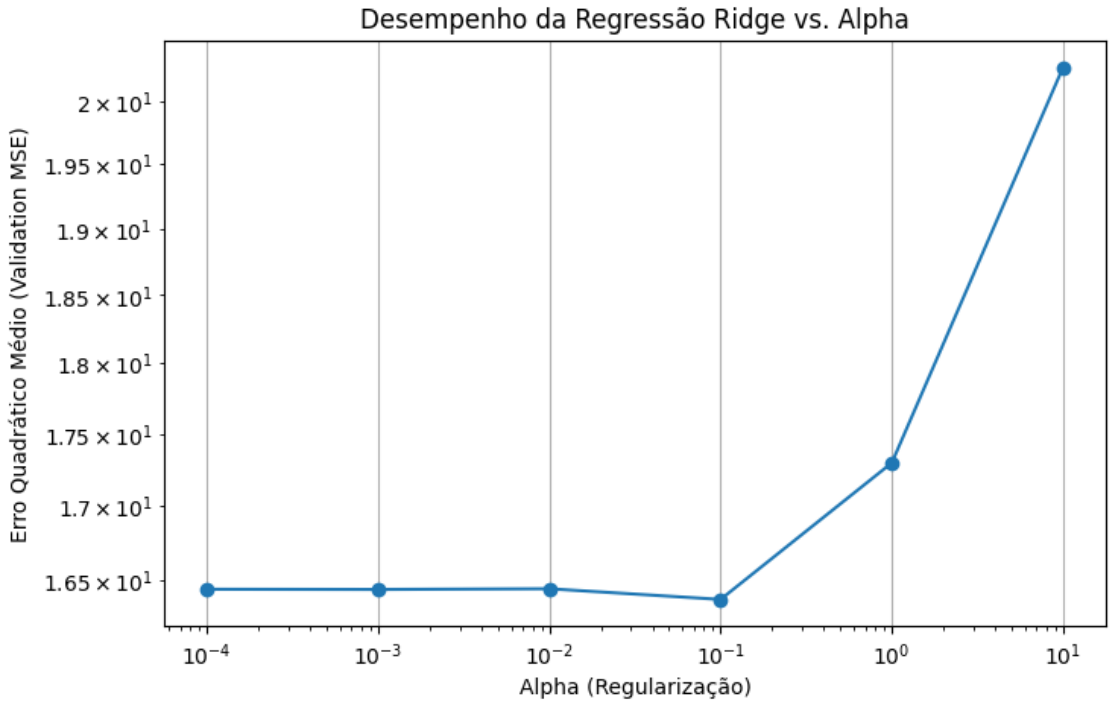
A Regressão Ridge adiciona uma penalidade ℓ_2 (soma dos quadrados dos coeficientes) à função de custo. Use o `SGDRegressor` com `penalty='l2'`.

[30]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# Gerando dados de exemplo (caso ainda não tenha)
np.random.seed(42)
X = np.random.uniform(-5, 5, 100).reshape(-1, 1)
y = 0.5 * X**2 + X + 2 + np.random.randn(100, 1) * 2  # y com shape (100, 1)

# Separando em treino e validação
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Corrigindo o formato de y para 1D
y_train = y_train.ravel()
y_val = y_val.ravel()

# Lista de alphas
alphas = [0.0001, 0.001, 0.01, 0.1, 1, 10]
ridge_val_mses = []
best_ridge_mse = float('inf')
best_ridge_alpha = None

print("Avaliando a Regressão Ridge (SGD com L2)...")
for alpha in alphas:
    # Criando pipeline com normalização e regressão Ridge via SGD
    ridge_pipeline = Pipeline([
        ("scaler", StandardScaler()),
        ("ridge_reg", SGDRegressor(penalty='l2', alpha=alpha, max_iter=1000,␣
 ↪tol=1e-3, random_state=42))
    ])

    # Treinando o modelo
    ridge_pipeline.fit(X_train, y_train)

    # Fazendo previsões
    y_pred = ridge_pipeline.predict(X_val)

    # Calculando MSE
    mse = mean_squared_error(y_val, y_pred)

    ridge_val_mses.append(mse)
    print(f"  Alpha: {alpha:<6} -> Validation MSE: {mse:.2f}")

    if mse < best_ridge_mse:
        best_ridge_mse = mse
        best_ridge_alpha = alpha

print(f"\nMelhor alpha para Ridge: {best_ridge_alpha} com MSE de validação:␣
 ↪{best_ridge_mse:.2f}")

# Plotando os resultados
plt.figure(figsize=(8, 5))
plt.plot(alphas, ridge_val_mses, marker='o')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Alpha (Regularização)')
```

```
plt.ylabel('Erro Quadrático Médio (Validation MSE)')
plt.title('Desempenho da Regressão Ridge vs. Alpha')
plt.grid(True)
plt.show()
```

```
Avaliando a Regressão Ridge (SGD com L2)…
  Alpha: 0.0001 -> Validation MSE: 16.44
  Alpha: 0.001  -> Validation MSE: 16.44
  Alpha: 0.01   -> Validation MSE: 16.45
  Alpha: 0.1    -> Validation MSE: 16.38
  Alpha: 1      -> Validation MSE: 17.30
  Alpha: 10     -> Validation MSE: 20.27

Melhor alpha para Ridge: 0.1 com MSE de validação: 16.38
```



1.2.2 2.2 Regressão Lasso (SGD com penalidade ℓ_1)

A Regressão Lasso usa uma penalidade ℓ_1 (soma dos valores absolutos dos coeficientes), que tem a propriedade de zerar os coeficientes de features menos importantes, realizando uma seleção automática de features.

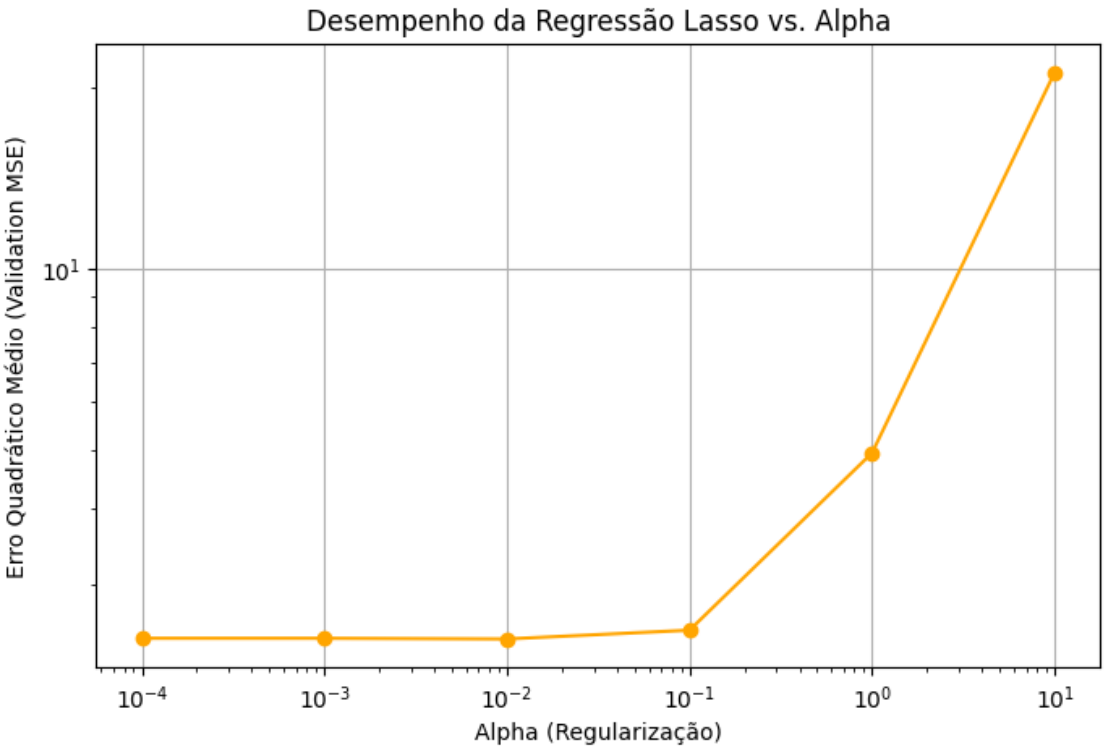
Use o `SGDRegressor` com `penalty='l1'`.

[31]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```python
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# Gerando dados de exemplo
np.random.seed(42)
X = np.random.uniform(-5, 5, 100).reshape(-1, 1)
y = 0.5 * X**2 + X + 2 + np.random.randn(100, 1) * 2

# Criando features polinomiais para testar a seleção do Lasso
poly = PolynomialFeatures(degree=10, include_bias=False)
X_poly = poly.fit_transform(X)
feature_names = poly.get_feature_names_out(["x"])

# Separando em treino e validação
X_train, X_val, y_train, y_val = train_test_split(X_poly, y, test_size=0.2,␣
 ↪random_state=42)
y_train = y_train.ravel()
y_val = y_val.ravel()

# Lista de alphas
alphas = [0.0001, 0.001, 0.01, 0.1, 1, 10]
lasso_val_mses = []
best_lasso_mse = float('inf')
best_lasso_alpha = None
best_lasso_model = None

print("Avaliando a Regressão Lasso (SGD com L1)...")
for alpha in alphas:
    # Pipeline com normalização e regressão Lasso via SGD
    lasso_model = Pipeline([
        ("scaler", StandardScaler()),
        ("lasso_reg", SGDRegressor(penalty='l1', alpha=alpha, max_iter=1000,␣
 ↪tol=1e-3, random_state=42))
    ])

    # Treinando o modelo
    lasso_model.fit(X_train, y_train)

    # Previsões
    y_pred = lasso_model.predict(X_val)

    # MSE
    mse = mean_squared_error(y_val, y_pred)
    lasso_val_mses.append(mse)
```

```
    print(f"  Alpha: {alpha:<6} -> Validation MSE: {mse:.2f}")
    if mse < best_lasso_mse:
        best_lasso_mse = mse
        best_lasso_alpha = alpha
        best_lasso_model = lasso_model

print(f"\nMelhor alpha para Lasso: {best_lasso_alpha} com MSE de validação:␣
 ↪{best_lasso_mse:.2f}")

# Plotando os resultados
plt.figure(figsize=(8, 5))
plt.plot(alphas, lasso_val_mses, marker='o', color='orange')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Alpha (Regularização)')
plt.ylabel('Erro Quadrático Médio (Validation MSE)')
plt.title('Desempenho da Regressão Lasso vs. Alpha')
plt.grid(True)
plt.show()

# Verificando a seleção de features
print("\nCoeficientes do melhor modelo Lasso:")
lasso_coeffs = best_lasso_model.named_steps["lasso_reg"].coef_
for feature, coef in zip(feature_names, lasso_coeffs):
    print(f"  {feature:>5}: {coef:.2f}")
print(f"\nLasso zerou {np.sum(lasso_coeffs == 0)} de {len(lasso_coeffs)}␣
 ↪features.")
```

```
Avaliando a Regressão Lasso (SGD com L1)…
  Alpha: 0.0001 -> Validation MSE: 2.44
  Alpha: 0.001  -> Validation MSE: 2.44
  Alpha: 0.01   -> Validation MSE: 2.43
  Alpha: 0.1    -> Validation MSE: 2.51
  Alpha: 1      -> Validation MSE: 4.95
  Alpha: 10     -> Validation MSE: 21.14

Melhor alpha para Lasso: 0.01 com MSE de validação: 2.43
```




```
Coeficientes do melhor modelo Lasso:
      x: 2.48
    x^2: 3.39
    x^3: 0.61
    x^4: 1.22
    x^5: 0.00
    x^6: 0.12
    x^7: -0.20
    x^8: -0.22
    x^9: -0.29
   x^10: -0.55

Lasso zerou 1 de 10 features.
```

1.2.3 2.3 Elastic Net

Elastic Net é um meio-termo entre Ridge e Lasso, combinando ambas as penalidades. O hiperparâmetro `l1_ratio` controla a mistura. Para este exercício, vamos fixar `l1_ratio=0.5` e variar `alpha`.

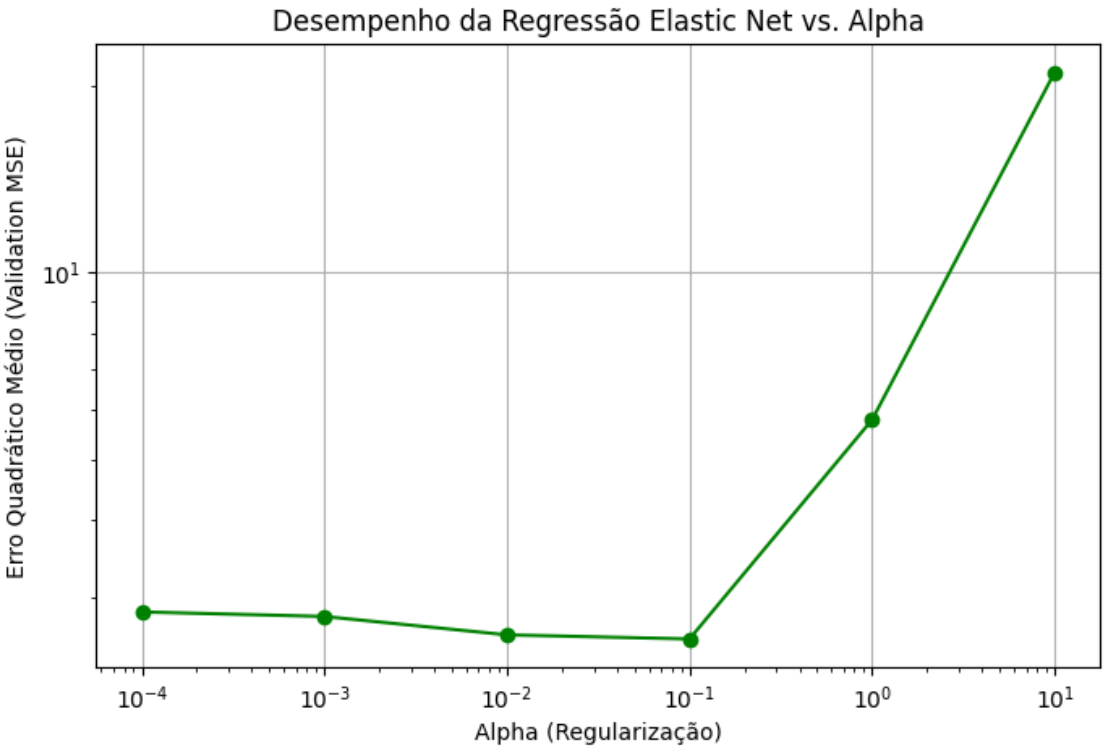
Use o modelo `ElasticNet`.

[32]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import ElasticNet
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# Gerando dados de exemplo
np.random.seed(42)
X = np.random.uniform(-5, 5, 100).reshape(-1, 1)
y = 0.5 * X**2 + X + 2 + np.random.randn(100, 1) * 2

# Criando features polinomiais para testar regularização
poly = PolynomialFeatures(degree=10, include_bias=False)
X_poly = poly.fit_transform(X)

# Separando em treino e validação
X_train, X_val, y_train, y_val = train_test_split(X_poly, y, test_size=0.2,␣
 ↪random_state=42)
y_train = y_train.ravel()
y_val = y_val.ravel()

# Lista de alphas
alphas = [0.0001, 0.001, 0.01, 0.1, 1, 10]
elastic_val_mses = []
best_elastic_mse = float('inf')
best_elastic_alpha = None

print("Avaliando a Regressão Elastic Net...")
for alpha in alphas:
    # Pipeline com normalização e ElasticNet
    elastic_model = Pipeline([
        ("scaler", StandardScaler()),
        ("elastic_reg", ElasticNet(alpha=alpha, l1_ratio=0.5, max_iter=1000,␣
 ↪random_state=42))
    ])

    # Treinando o modelo
    elastic_model.fit(X_train, y_train)

    # Previsões
    y_pred = elastic_model.predict(X_val)

    # MSE
    mse = mean_squared_error(y_val, y_pred)
    elastic_val_mses.append(mse)
```

```
    print(f"  Alpha: {alpha:<6} -> Validation MSE: {mse:.2f}")
    if mse < best_elastic_mse:
        best_elastic_mse = mse
        best_elastic_alpha = alpha

print(f"\nMelhor alpha para Elastic Net: {best_elastic_alpha} com MSE de␣
 ↪validação: {best_elastic_mse:.2f}")

# Plotando os resultados
plt.figure(figsize=(8, 5))
plt.plot(alphas, elastic_val_mses, marker='o', color='green')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Alpha (Regularização)')
plt.ylabel('Erro Quadrático Médio (Validation MSE)')
plt.title('Desempenho da Regressão Elastic Net vs. Alpha')
plt.grid(True)
plt.show()
```

```
Avaliando a Regressão Elastic Net…
  Alpha: 0.0001 -> Validation MSE: 2.83
  Alpha: 0.001  -> Validation MSE: 2.79
  Alpha: 0.01   -> Validation MSE: 2.60
  Alpha: 0.1    -> Validation MSE: 2.56
  Alpha: 1      -> Validation MSE: 5.79
  Alpha: 10     -> Validation MSE: 21.06

Melhor alpha para Elastic Net: 0.1 com MSE de validação: 2.56
```

```
/home/tailan/Documentos/6 Semestre/Aprendizado de Maquina/resolucao dos
colabs/venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.041e+02, tolerance: 2.047e-01
  model = cd_fast.enet_coordinate_descent(
/home/tailan/Documentos/6 Semestre/Aprendizado de Maquina/resolucao dos
colabs/venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.051e+01, tolerance: 2.047e-01
  model = cd_fast.enet_coordinate_descent(
```



1.3 3. Regularização por Early Stopping

Early stopping é uma forma diferente de regularização. Em vez de adicionar um termo de penalidade, ela interrompe o treinamento assim que o erro no conjunto de validação para de diminuir (ou começa a aumentar), evitando o overfitting.

Para isso, use `SGDRegressor` sem penalidade (`penalty=None`), mas com os parâmetros de early stopping ativados.

[33]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# Gerando dados de exemplo
np.random.seed(42)
X = np.random.uniform(-5, 5, 100).reshape(-1, 1)
y = 0.5 * X**2 + X + 2 + np.random.randn(100, 1) * 2

# Criando features polinomiais para aumentar a complexidade
```

```
poly = PolynomialFeatures(degree=10, include_bias=False)
X_poly = poly.fit_transform(X)

# Separando em treino e validação
X_train, X_val, y_train, y_val = train_test_split(X_poly, y, test_size=0.2,␣
 ↪random_state=42)
y_train = y_train.ravel()
y_val = y_val.ravel()

print("Avaliando SGD com Early Stopping...")

# ====================== Early Stopping ======================
early_stopping_model = Pipeline([
    ("scaler", StandardScaler()),
    ("sgd", SGDRegressor(
        penalty=None,
        early_stopping=True,
        n_iter_no_change=1,
        validation_fraction=0.5,
        max_iter=1000,
        tol=1e-3,
        random_state=42
    ))
])

# Treinando o modelo
early_stopping_model.fit(X_train, y_train)

# Previsões
y_pred = early_stopping_model.predict(X_val)

# MSE
early_stopping_mse = mean_squared_error(y_val, y_pred)
# ============================================================

print(f"MSE de validação do modelo com Early Stopping: {early_stopping_mse:.
 ↪2f}")
```

```
Avaliando SGD com Early Stopping…
MSE de validação do modelo com Early Stopping: 3.66
```

1.4 4. Avaliação Final no Conjunto de Teste

Agora que você avaliou todos os modelos no conjunto de validação, é hora de escolher o melhor e avaliá-lo no conjunto de teste. O melhor modelo é aquele que obteve o menor MSE de validação.

1. Compare os melhores MSEs de Ridge, Lasso, Elastic Net e Early Stopping.
2. Identifique (manualmente) o melhor modelo e seu melhor hiperparâmetro alpha (se aplicável).

3. **Importante:** Re-treine este modelo final usando o conjunto de **treinamento + validação** (`X_train_val`, `y_train_val`) para aproveitar o máximo de dados possível.
4. Faça a previsão final no conjunto de **teste** (`X_test`) e calcule o MSE.

[34]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# Supondo que X_poly e y já estejam definidos
# Separando os dados em treino+validação e teste
X_train_val, X_test, y_train_val, y_test = train_test_split(X_poly, y,␣
 ↪test_size=0.2, random_state=42)
y_train_val = y_train_val.ravel()
y_test = y_test.ravel()

# Exibindo os melhores MSEs
print(f"Melhor MSE (Ridge):       {best_ridge_mse:.2f}␣
 ↪(alpha={best_ridge_alpha})")
print(f"Melhor MSE (Lasso):       {best_lasso_mse:.2f}␣
 ↪(alpha={best_lasso_alpha})")
print(f"Melhor MSE (Elastic Net): {best_elastic_mse:.2f}␣
 ↪(alpha={best_elastic_alpha})")
print(f"Melhor MSE (Early Stop):  {early_stopping_mse:.2f}")

# ====================== Avaliação Final ======================
# Supondo que Elastic Net foi o melhor modelo
final_model = Pipeline([
    ("scaler", StandardScaler()),
    ("elastic_reg", ElasticNet(alpha=best_elastic_alpha, l1_ratio=0.5,␣
 ↪max_iter=1000, random_state=42))
])

# Re-treinando com treino + validação
final_model.fit(X_train_val, y_train_val)

# Previsão no conjunto de teste
y_test_pred = final_model.predict(X_test)

# MSE final
final_mse = mean_squared_error(y_test, y_test_pred)
# ============================================================

print(f"\nMSE final no conjunto de teste do melhor modelo: {final_mse:.2f}")
```

```
Melhor MSE (Ridge):       16.38 (alpha=0.1)
```

```
Melhor MSE (Lasso):       2.43 (alpha=0.01)
Melhor MSE (Elastic Net): 2.56 (alpha=0.1)
Melhor MSE (Early Stop):  3.66

MSE final no conjunto de teste do melhor modelo: 2.56
```

1.5 Conclusão

Neste exercício, você: - Implementou e comparou Regressão Ridge, Lasso e Elastic Net. - Usou um conjunto de validação para encontrar o melhor hiperparâmetro `alpha` para cada modelo. - Observou como a Regressão Lasso zerou alguns coeficientes, realizando seleção de features. - Implementou a regularização por Early Stopping como uma alternativa às penalidades ℓ_1/ℓ_2. - Selecionou o melhor modelo com base no desempenho de validação e o avaliou em um conjunto de teste separado para obter uma estimativa imparcial de seu desempenho em dados novos.

Exercício prático 07_ Regressão Logística

October 29, 2025

1 Exercício de Programação: Regressão Logística e Softmax com MNIST

Neste exercício, você aplicará os conceitos de Regressão Logística para classificação binária e Regressão Softmax para classificação multiclasse. Usaremos o famoso dataset MNIST, que consiste em imagens de dígitos manuscritos.

Objetivos: 1. Treinar um classificador binário para identificar se um dígito é '5' ou 'não-5'. 2. Treinar um classificador multiclasse para identificar os dígitos de 0 a 9. 3. Avaliar a performance de ambos os modelos.

1.1 1. Preparação do Ambiente e Carregamento dos Dados

Primeiro, vamos importar as bibliotecas necessárias e carregar o dataset MNIST.

Também vamos pré-processar os dados: - Dividir em conjuntos de treino e teste. - Escalar os valores dos pixels para melhorar a performance do gradiente descendente. - Remodelar as imagens de 28x28 para vetores de 784 dimensões, que é o formato esperado por um classificador como `LogisticRegression`.

[24]:
```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# Carregar o dataset MNIST
digits = load_digits()
X, y = digits.data, digits.target

# Dividir em conjuntos de treino e teste
X_train, X_test = X[:1600], X[1600:]
y_train, y_test = y[:1600], y[1600:]

# Escalar os pixels
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Remodelar as imagens para vetores 1D (28*28 = 784)
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)

print(f"Formato dos dados de treino: {X_train_flat.shape}")
print(f"Formato dos dados de teste: {X_test_flat.shape}")
```

```
Formato dos dados de treino: (1600, 64)
Formato dos dados de teste: (197, 64)
```

1.2 2. Treinando um Classificador Binário (5 ou não-5)

Agora, vamos criar um classificador para uma tarefa binária: detectar se um dígito é o número 5 ou não. Para isso, precisamos ajustar nossos rótulos (`y_train` e `y_test`).

[25]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split

# Carregar o dataset MNIST
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
X, y = mnist["data"], mnist["target"].astype(int)

# Dividir em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Criar os rótulos binários: True para 5, False para outros
y_train_5 = (y_train == 5)
y_test_5 = (y_test == 5)

# Inicializar e treinar o modelo de Regressão Logística
log_reg = LogisticRegression(max_iter=10)
log_reg.fit(X_train, y_train_5)

print(" Modelo de Regressão Logística treinado!")
```

```
 Modelo de Regressão Logística treinado!
/home/tailan/Documentos/6 Semestre/Aprendizado de Maquina/resolucao dos
colabs/venv/lib/python3.12/site-packages/sklearn/linear_model/_logistic.py:473:
ConvergenceWarning: lbfgs failed to converge after 10 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

1.2.1 Avaliação do Classificador Binário

Com o modelo treinado, vamos avaliar sua acurácia no conjunto de teste.

[26]:
```
from sklearn.metrics import accuracy_score

y_pred_5 = log_reg.predict(X_test)

# Ou usar o método .score() diretamente
acc_score = log_reg.score(X_test, y_test_5)
print(f"Acurácia usando .score(): {acc_score:.4f}")
```

```
Acurácia usando .score(): 0.9629
```

1.3 3. Treinando um Classificador Multiclasse (Regressão Softmax)

O `LogisticRegression` do Scikit-Learn automaticamente lida com a classificação multiclasse usando a estratégia "um-contra-o-resto" (OvR) por padrão. Para usar a Regressão Softmax (também chamada de Regressão Logística Multinomial), podemos definir o argumento `multi_class='multinomial'`.

Vamos treinar um novo modelo para classificar todos os 10 dígitos (0 a 9).

[27]:
```
# SEU CÓDIGO AQUI: Treine o modelo softmax usando o conjunto de treinamento com␣
↪todas as classes (X_train_flat, y_train).

# Inicializar o modelo de Regressão Softmax
# Usamos max_iter=1000 para garantir a convergência.

# ----------------
from sklearn.linear_model import LogisticRegression

# Inicializar o modelo de Regressão Softmax
softmax_reg = LogisticRegression(multi_class='multinomial', solver='lbfgs',␣
↪max_iter=10)

# Treinar o modelo com todas as classes
softmax_reg.fit(X_train, y_train)

print("Modelo de Regressão Softmax treinado!")
```

```
/home/tailan/Documentos/6 Semestre/Aprendizado de Maquina/resolucao dos
colabs/venv/lib/python3.12/site-packages/sklearn/linear_model/_logistic.py:1272:
FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed
in 1.8. From then on, it will always use 'multinomial'. Leave it to its default
```

```
value to avoid this warning.
  warnings.warn(

Modelo de Regressão Softmax treinado!

/home/tailan/Documentos/6 Semestre/Aprendizado de Maquina/resolucao dos
colabs/venv/lib/python3.12/site-packages/sklearn/linear_model/_logistic.py:473:
ConvergenceWarning: lbfgs failed to converge after 10 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

1.3.1 Avaliação do Classificador Multiclasse

Por fim, avaliamos o modelo multiclasse no conjunto de teste.

[28]:
```python
# Calcular e imprimir a acurácia no conjunto de teste
accuracy = softmax_reg.score(X_test, y_test)
print(f"Acurácia do modelo multiclasse: {accuracy:.4f}")
```

```
Acurácia do modelo multiclasse: 0.8826
```

1.4 Conclusão

Parabéns! Você implementou e avaliou com sucesso dois tipos de classificadores lineares: 1. Um classificador de Regressão Logística para uma tarefa binária. 2. Um classificador de Regressão Softmax para uma tarefa multiclasse.

Você deve notar que, embora simples, esses modelos já alcançam uma acurácia razoavelmente alta para o reconhecimento de dígitos.

Exercício prático 08_ SVM Linear e Não Linear

October 29, 2025

1 Exercício de Programação: Classificação com SVM Linear e Não-Linear

Neste exercício, você aplicará os conceitos de Máquinas de Vetores de Suporte (SVM) para classificação linear e não-linear. O objetivo é visualizar o impacto de diferentes hiperparâmetros (`C`, `kernel`, `degree`, `coef0`, `gamma`) nos limites de decisão do modelo.

Usaremos uma versão modificada do dataset de habitação da Califórnia para criar um problema de classificação binária.

Objetivos: 1. Treinar e visualizar classificadores `LinearSVC` com diferentes valores de regularização `C`. 2. Treinar e visualizar classificadores `SVC` com kernel polinomial, explorando o hiperparâmetro `coef0`. 3. Treinar e visualizar classificadores `SVC` com kernel RBF, explorando os hiperparâmetros `gamma` e `C`.

1.1 1. Preparação do Ambiente e dos Dados

Primeiro, importamos as bibliotecas necessárias. Em seguida, carregamos o dataset de habitação da Califórnia, mas o adaptamos para uma tarefa de classificação: - Usaremos apenas duas features: Renda Média (`MedInc`) e Idade Média da Casa (`HouseAge`). - Criaremos um target binário: `y = 1` se o Valor Médio da Casa (`MedHouseVal`) for maior ou igual a 1.5 (US$150.000), e `y = 0` caso contrário.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC, LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score


# Carregar o dataset
housing = fetch_california_housing()
X = housing.data[:, [0, 5]]  # MedInc e HouseAge
y = (housing.target >= 1.5).astype(np.float64)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Para tornar a visualização mais clara, vamos usar apenas uma amostra dos dados
np.random.seed(42)
sample_idx = np.random.permutation(len(X_train))[:100]
X_train = X_train[sample_idx]
y_train = y_train[sample_idx]

print(f"Formato de X: {X_train.shape}")
print(f"Formato de y: {y_train.shape}")
```

```
Formato de X: (100, 2)
Formato de y: (100,)
```

1.2 2. Escalando as Features

SVMs são sensíveis à escala das features. Portanto, é crucial escalá-las usando `StandardScaler`.

[2]:
```
scaler = StandardScaler()

# SEU CÓDIGO AQUI: Use o scaler para ajustar e transformar o conjunto de dados␣
 ↪X.
X_train_scaled = ...
X_test_scaled = ...
# ----------------

print("Features escaladas!")
```

```
Features escaladas!
```

1.3 3. Classificação com SVM Linear (`LinearSVC`)

Vamos treinar dois classificadores `LinearSVC`, um com `C=1` (mais regularização, margem mais larga) e outro com `C=100` (menos regularização, margem mais estreita). Lembre-se que `LinearSVC` maximiza a margem entre as classes.

[3]:
```
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
import numpy as np

# Suponha que X e y já estejam definidos com os dados
# Exemplo fictício para contexto:
# X = ... (matriz de atributos)
# y = ... (rótulos binários: 0 ou 1)
```

```
# Dividir em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Escalar os dados
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Hiperparâmetros
C1, C2 = 1, 100

# Treinar os dois modelos LinearSVC com os dados escalados
lin_svc_c1 = LinearSVC(C=C1)
lin_svc_c100 = LinearSVC(C=C2)

lin_svc_c1.fit(X_train_scaled, y_train)
lin_svc_c100.fit(X_train_scaled, y_train)

print("Modelos Lineares treinados!")

# Prever no conjunto de teste
y_pred_c1 = lin_svc_c1.predict(X_test_scaled)
y_pred_c100 = lin_svc_c100.predict(X_test_scaled)

# Calcular F1 Score
f1_c1 = f1_score(y_test, y_pred_c1)
f1_c100 = f1_score(y_test, y_pred_c100)

print(f"F1 Score (C={C1}): {f1_c1}")
print(f"F1 Score (C={C2}): {f1_c100}")
```

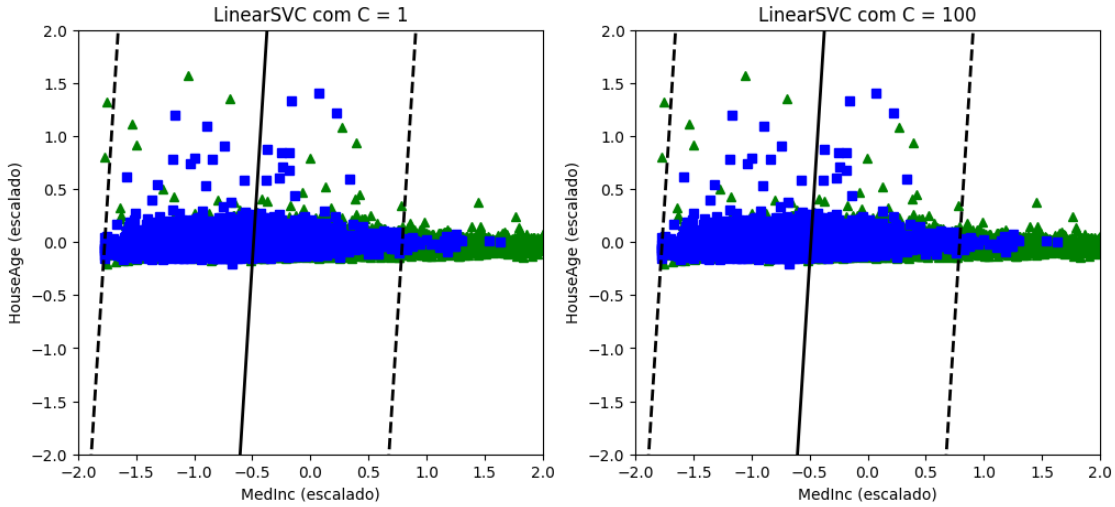
```
Modelos Lineares treinados!
F1 Score (C=1): 0.8082940622054665
F1 Score (C=100): 0.8082940622054665
```

1.3.1 Plotando os Limites de Decisão do SVM Linear

A função `plot_svc_decision_boundary` abaixo nos ajuda a visualizar a rua (margem) criada pelos modelos. Sua tarefa é chamar essa função para plotar os resultados dos dois modelos que você treinou.

[4]:
```
# Função auxiliar para plotar o limite de decisão e as margens
def plot_svc_decision_boundary(svm_clf, xmin, xmax):
    w = svm_clf.coef_[0]
    b = svm_clf.intercept_[0]

    x0 = np.linspace(xmin, xmax, 200)
```

```
    decision_boundary = -w[0] / w[1] * x0 - b / w[1]

    margin = 1 / w[1]
    gutter_up = decision_boundary + margin
    gutter_down = decision_boundary - margin

    plt.plot(x0, decision_boundary, "k-", linewidth=2)
    plt.plot(x0, gutter_up, "k--", linewidth=2)
    plt.plot(x0, gutter_down, "k--", linewidth=2)

# Configuração da figura
plt.figure(figsize=(12, 5))

# Plot para C=1
plt.subplot(121)
plt.plot(X_train_scaled[:, 0][y_train==1], X_train_scaled[:, 1][y_train==1],␣
 ↪"g^", label="y=1")
plt.plot(X_train_scaled[:, 0][y_train==0], X_train_scaled[:, 1][y_train==0],␣
 ↪"bs", label="y=0")
plt.xlabel("MedInc (escalado)")
plt.ylabel("HouseAge (escalado)")
plt.title(f"LinearSVC com C = {C1}")
plt.axis([-2, 2, -2, 2])
plot_svc_decision_boundary(lin_svc_c1, -2, 2)

# Plot para C=100
plt.subplot(122)
plt.plot(X_train_scaled[:, 0][y_train==1], X_train_scaled[:, 1][y_train==1],␣
 ↪"g^")
plt.plot(X_train_scaled[:, 0][y_train==0], X_train_scaled[:, 1][y_train==0],␣
 ↪"bs")
plt.xlabel("MedInc (escalado)")
plt.ylabel("HouseAge (escalado)")
plt.title(f"LinearSVC com C = {C2}")
plt.axis([-2, 2, -2, 2])
plot_svc_decision_boundary(lin_svc_c100, -2, 2)

plt.show()
```



1.4 4. Classificação com SVM Não-Linear (Kernel Polinomial)

Para dados que não são linearmente separáveis, podemos usar o truque do kernel. Vamos treinar um `SVC` com um kernel polinomial de grau 3. O hiperparâmetro `coef0` controla o quanto o modelo é influenciado por polinômios de alto grau versus baixo grau. Vamos comparar `coef0=1` e `coef0=100`.

[5]:
```
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score

# Modelos com kernel polinomial
poly_svc_c1 = Pipeline([
    ("scaler", StandardScaler()),
    ("svm_clf", SVC(kernel="poly", degree=3, coef0=1, C=5))
])

poly_svc_c100 = Pipeline([
    ("scaler", StandardScaler()),
    ("svm_clf", SVC(kernel="poly", degree=3, coef0=10, C=5))
])

# Treino
poly_svc_c1.fit(X_train, y_train)
poly_svc_c100.fit(X_train, y_train)

print("Modelos com kernel polinomial treinados!")

svm_clfs = [poly_svc_c1, poly_svc_c100]
hyperparams = [(1, 5), (100, 5)]
```

```
for (C, coef0), clf in zip(hyperparams, svm_clfs):
    print(f"F1 Score (C={C}, coef0={coef0}): {f1_score(y_test, clf.
 ↪predict(X_test))}")
```

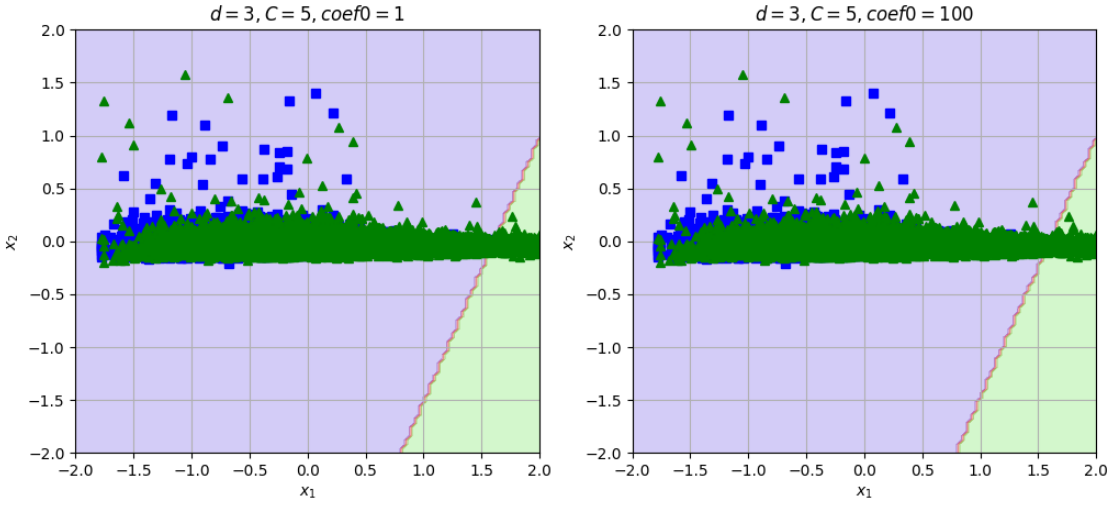
```
Modelos com kernel polinomial treinados!
F1 Score (C=1, coef0=5): 0.8184794470716624
F1 Score (C=100, coef0=5): 0.8184133915574964
```

1.4.1 Plotando os Limites de Decisão do SVM Polinomial

Como os limites não são mais lineares, precisamos de uma nova função para plotar as regiões de decisão.

[6]:
```
# Função auxiliar para plotar os limites de modelos não-lineares
def plot_predictions(clf, axes):
    x0s = np.linspace(axes[0], axes[1], 100)
    x1s = np.linspace(axes[2], axes[3], 100)
    x0, x1 = np.meshgrid(x0s, x1s)
    X = np.c_[x0.ravel(), x1.ravel()]
    y_pred = clf.predict(X).reshape(x0.shape)
    plt.contourf(x0, x1, y_pred, cmap=plt.cm.brg, alpha=0.2)

def plot_dataset(X, y, axes):
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "bs")
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "g^")
    plt.axis(axes)
    plt.grid(True, which='both')
    plt.xlabel(r"$x_1$")
    plt.ylabel(r"$x_2$")

# Configuração da figura
plt.figure(figsize=(12, 5))

# Plot para coef0=1
plt.subplot(121)
plot_dataset(X_train_scaled, y_train, [-2, 2, -2, 2])
plt.title(r"$d=3, C=5, coef0=1$")

# SEU CÓDIGO AQUI: Chame a função para plotar o limite de decisão do primeiro␣
 ↪modelo polinomial.

plot_predictions(poly_svc_c1, [-2, 2, -2, 2])
# Plot para coef0=100
plt.subplot(122)
plot_dataset(X_train_scaled, y_train, [-2, 2, -2, 2])
plt.title(r"$d=3, C=5, coef0=100$")
```

```
# SEU CÓDIGO AQUI: Chame a função para plotar o limite de decisão do segundo␣
 ↪modelo polinomial.

plot_predictions(poly_svc_c100, [-2, 2, -2, 2])
plt.show()
```



1.5 5. Classificação com SVM Não-Linear (Kernel RBF)

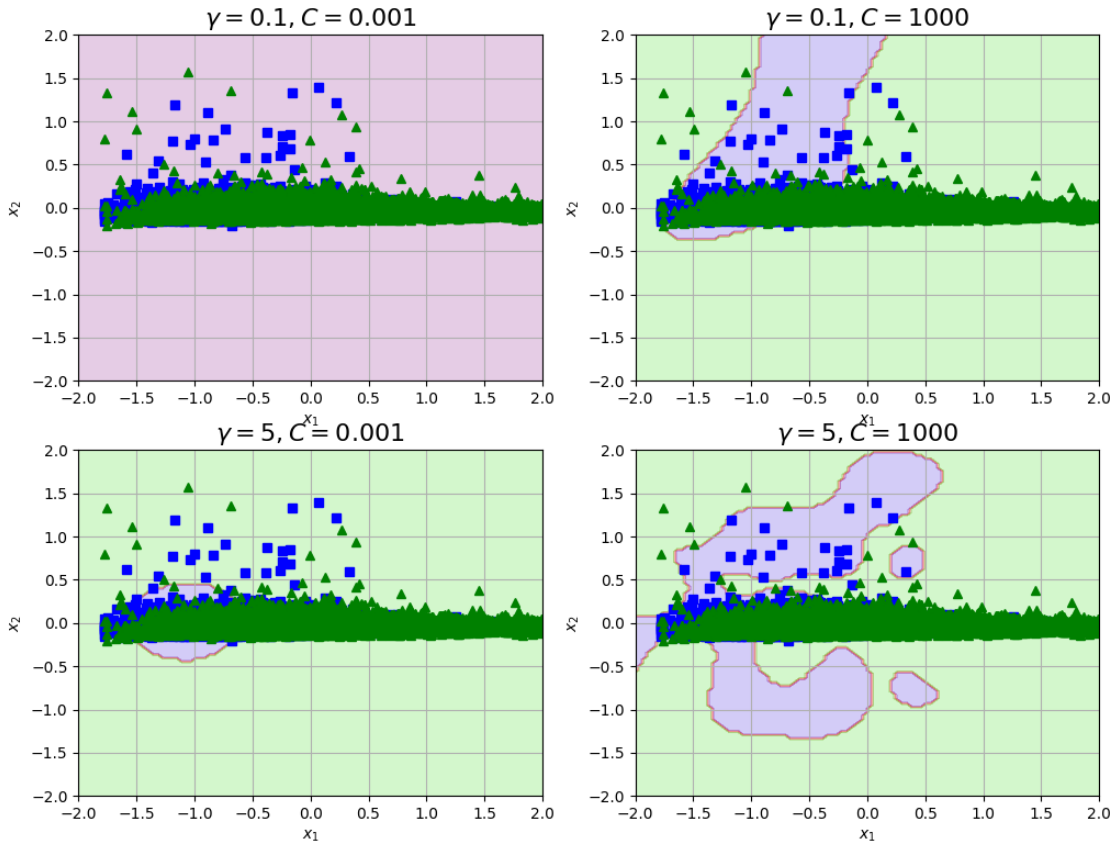
O kernel RBF (Função de Base Radial) é muito poderoso. Seus principais hiperparâmetros são `gamma` e `C`. - `gamma` atua como um hiperparâmetro de regularização: aumentá-lo torna o limite de decisão mais irregular (aumenta a variância, diminui o viés), enquanto diminuí-lo o torna mais suave. - `C` funciona como nos modelos lineares: um `C` grande leva a menos regularização, enquanto um `C` pequeno leva a mais regularização.

Vamos treinar e visualizar quatro modelos com diferentes combinações desses hiperparâmetros.

[7]:
```
from sklearn.svm import SVC

gamma1, gamma2 = 0.1, 5
C1, C2 = 0.001, 1000
hyperparams = (gamma1, C1), (gamma1, C2), (gamma2, C1), (gamma2, C2)

svm_clfs = []
for gamma, C in hyperparams:
    # Modelo
    rbf_kernel_svm_clf = SVC(kernel='rbf', gamma=gamma, C=C)

    # Treino
    rbf_kernel_svm_clf.fit(X_train_scaled, y_train)
```

```
    svm_clfs.append(rbf_kernel_svm_clf)

print("Modelos RBF treinados!")

plt.figure(figsize=(12, 9))

for i, svm_clf in enumerate(svm_clfs):
    y_pred = svm_clf.predict(X_test_scaled)
    print(f"F1 Score (gamma={hyperparams[i][0]}, C={hyperparams[i][1]}):␣
 ↪{f1_score(y_test, y_pred)}")

    plt.subplot(221 + i)
    plot_dataset(X_train_scaled, y_train, [-2, 2, -2, 2])
    gamma, C = hyperparams[i]
    plt.title(r"$\gamma = {}, C = {}$".format(gamma, C), fontsize=16)

    # Chame a função para plotar o limite de decisão para o modelo atual
    plot_predictions(svm_clf, [-2, 2, -2, 2])

plt.show()
```

```
Modelos RBF treinados!
F1 Score (gamma=0.1, C=0.001): 0.7779751332149201
F1 Score (gamma=0.1, C=1000): 0.8162062615101289
F1 Score (gamma=5, C=0.001): 0.8202468252548739
F1 Score (gamma=5, C=1000): 0.8173652694610778
```


$\gamma = 0.1, C = 0.001$
$\gamma = 0.1, C = 1000$
$\gamma = 5, C = 0.001$
$\gamma = 5, C = 1000$
x_1
x_2

Exercício de Árvore de Decisão: Classificação de Vinhos

Objetivo: Neste exercício, você irá treinar, visualizar e avaliar um classificador de Árvore de Decisão usando o dataset 'wine' do Scikit-Learn. Você focará nos conceitos introduzidos no Capítulo 6, até a seção "Gini Impurity or Entropy?", incluindo a divisão treino/teste, treinamento, visualização da árvore, plotagem dos limites de decisão e avaliação de desempenho usando validação cruzada e métricas de classificação.

1. Configuração e Importações

Primeiro, vamos importar as bibliotecas necessárias.

Obs.: a instalação do pacote graphviz pode levar mais de 10 minutos!

In [100…
```
#!conda install -y -c conda-forge graphviz python-graphviz
```

In [101…
```
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
import pandas as pd

from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Para plotar figuras bonitas
%matplotlib inline
import matplotlib as mpl
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Onde salvar as figuras
IMAGES_PATH = Path() / "images" / "decision_trees"
IMAGES_PATH.mkdir(parents=True, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=3
    path = IMAGES_PATH / f"{fig_id}.{fig_extension}"
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

# Função auxiliar para plotar limites de decisão
def plot_decision_boundary(clf, X, y, axes=[0, 7.5, 0, 3], iris=True, leg
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = mpl.colors.ListedColormap(['#fafab0','#9898ff','#a0faa0
```

```
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)
    if not iris:
        custom_cmap2 = mpl.colors.ListedColormap(['#7d7d58','#4c4c7f','#5
        plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    if plot_training:
        plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", label="Classe 0")
        plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", label="Classe 1")
        plt.plot(X[:, 0][y==2], X[:, 1][y==2], "g^", label="Classe 2")
        plt.axis(axes)
    if iris:
        plt.xlabel("Comprimento da pétala (cm)", fontsize=14)
        plt.ylabel("Largura da pétala (cm)", fontsize=14)
    else:
        plt.xlabel(r"$x_1$", fontsize=18)
        plt.ylabel(r"$x_2$", fontsize=18, rotation=0)
    if legend:
        plt.legend(loc="lower right", fontsize=14)
```

2. Carregar e Preparar os Dados

Vamos carregar o dataset `wine` e usar apenas duas características para facilitar a visualização: 'alcohol' e 'malic_acid'.

In [102…

```
wine = load_wine(as_frame=True)
print(wine.DESCR)
wine.data.info()
wine.data.head()
```

```
.. _wine_dataset:

Wine recognition dataset
------------------------

**Data Set Characteristics:**

:Number of Instances: 178
:Number of Attributes: 13 numeric, predictive attributes and the class
:Attribute Information:
    - Alcohol
    - Malic acid
    - Ash
    - Alcalinity of ash
    - Magnesium
    - Total phenols
    - Flavanoids
    - Nonflavanoid phenols
    - Proanthocyanins
    - Color intensity
    - Hue
    - OD280/OD315 of diluted wines
    - Proline
    - class:
        - class_0
        - class_1
        - class_2

:Summary Statistics:

============================= ==== ===== ======= =====
                               Min   Max   Mean     SD
============================= ==== ===== ======= =====
Alcohol:                     11.0  14.8    13.0   0.8
Malic Acid:                  0.74  5.80    2.34  1.12
Ash:                         1.36  3.23    2.36  0.27
Alcalinity of Ash:           10.6  30.0    19.5   3.3
Magnesium:                   70.0 162.0    99.7  14.3
Total Phenols:               0.98  3.88    2.29  0.63
Flavanoids:                  0.34  5.08    2.03  1.00
Nonflavanoid Phenols:        0.13  0.66    0.36  0.12
Proanthocyanins:             0.41  3.58    1.59  0.57
Colour Intensity:             1.3  13.0     5.1   2.3
Hue:                         0.48  1.71    0.96  0.23
OD280/OD315 of diluted wines: 1.27  4.00    2.61  0.71
Proline:                      278  1680     746   315
============================= ==== ===== ======= =====

:Missing Attribute Values: None
:Class Distribution: class_0 (59), class_1 (71), class_2 (48)
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.
https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

The data is the results of a chemical analysis of wines grown in the same
region in Italy by three different cultivators. There are thirteen differe
nt
```

```
measurements taken for different constituents found in the three types of
wine.

Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlatio
n.
Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository
[https://archive.ics.uci.edu/ml]. Irvine, CA: University of California,
School of Information and Computer Science.

.. dropdown:: References

    (1) S. Aeberhard, D. Coomans and O. de Vel,
    Comparison of Classifiers in High Dimensional Settings,
    Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
    Mathematics and Statistics, James Cook University of North Queensland.
    (Also submitted to Technometrics).

    The data was used with many others for comparing various
    classifiers. The classes are separable, though only RDA
    has achieved 100% correct classification.
    (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
    (All results using the leave-one-out technique)

    (2) S. Aeberhard, D. Coomans and O. de Vel,
    "THE CLASSIFICATION PERFORMANCE OF RDA"
    Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
    Mathematics and Statistics, James Cook University of North Queensland.
    (Also submitted to Journal of Chemometrics).

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 13 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
dtypes: float64(13)
memory usage: 18.2 KB
```

Out[102…

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	n
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	

In [103…
```
# Selecionar apenas as features 'alcohol' e 'malic_acid'
X = wine.data[["alcohol", "malic_acid"]].values
y = wine.target.values

print("Shape de X:", X.shape)
print("Shape de y:", y.shape)
```

```
Shape de X: (178, 2)
Shape de y: (178,)
```

3. Dividir os Dados em Conjuntos de Treino e Teste

Sua tarefa: Use `train_test_split` para dividir os dados `X` e `y` em conjuntos de treinamento e teste. Use 20% dos dados para teste e defina `random_state=42` para reprodutibilidade.

In [104…
```
# <<< SEU CÓDIGO AQUI >>>
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
# <<< FIM DO SEU CÓDIGO >>>

print("Tamanho do treino:", len(X_train))
print("Tamanho do teste:", len(X_test))
```

```
Tamanho do treino: 142
Tamanho do teste: 36
```

4. Treinar um Classificador de Árvore de Decisão

Sua tarefa: Treine um `DecisionTreeClassifier` com `max_depth=2` e `random_state=42` no conjunto de treinamento.

In [105…
```
tree_clf_depth2 = None

# <<< SEU CÓDIGO AQUI >>>
tree_clf_depth2 = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf_depth2.fit(X_train, y_train)
# <<< FIM DO SEU CÓDIGO >>>

print("Modelo treinado com max_depth=2.")
```

```
Modelo treinado com max_depth=2.
```

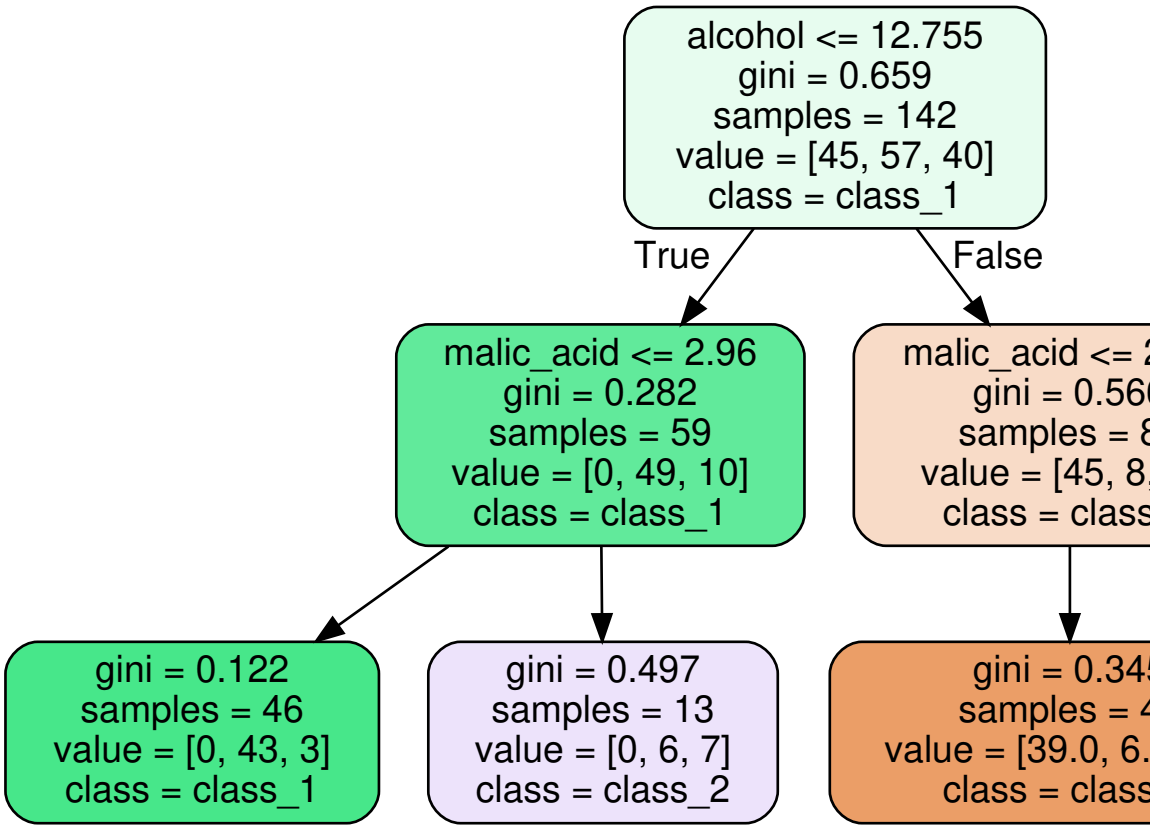
5. Visualizar a Árvore de Decisão

Usamos `export_graphviz` para gerar um arquivo `.dot` representando a árvore treinada. Incluímos os nomes das características ('alcohol', 'malic_acid') e os nomes das classes (do dataset `wine`). Em seguida, exibimos a árvore usando `graphviz.Source`.

In [106…

```
from graphviz import Source

dot_file_path = IMAGES_PATH / "wine_tree_depth2.dot"

export_graphviz(
    tree_clf_depth2,
    out_file=str(dot_file_path),
    feature_names=["alcohol", "malic_acid"],
    class_names=wine.target_names,
    rounded=True,
    filled=True
)

# Exibir a árvore
source = Source.from_file(dot_file_path)
source
```

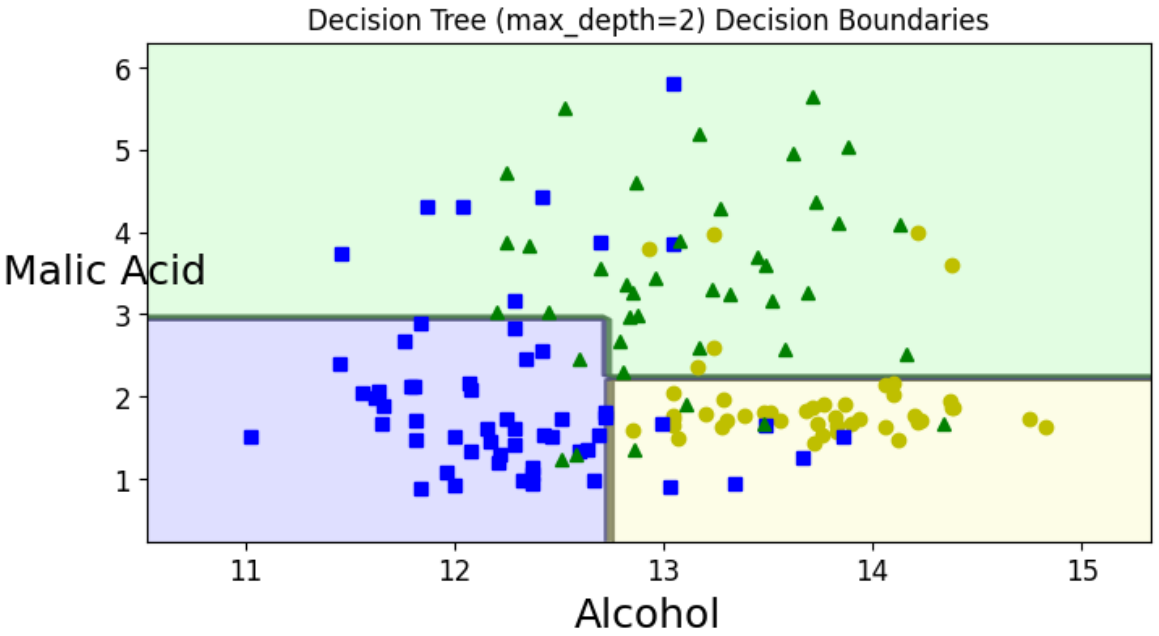
Out[106…



6. Plotar os Limites de Decisão

Usamos a função auxiliar `plot_decision_boundary` para visualizar os limites de decisão da árvore treinada (`tree_clf_depth2`) no conjunto de treinamento (`X_train` , `y_train`). Ajustamos os eixos para uma boa visualização.

In [107…
```
plt.figure(figsize=(8, 4))

axes = [X[:, 0].min()-0.5, X[:, 0].max()+0.5, X[:, 1].min()-0.5, X[:, 1].
plot_decision_boundary(tree_clf_depth2, X_train, y_train, axes=axes, iris
plt.xlabel("Alcohol")
plt.ylabel("Malic Acid")
plt.title("Decision Tree (max_depth=2) Decision Boundaries")

plt.show()
```



7. Comparar Profundidades Usando Validação Cruzada

Sua tarefa: Use validação cruzada (com `cross_val_score`, `cv=5` e `scoring='accuracy'`) no conjunto de **treinamento** para comparar o desempenho médio de árvores com `max_depth=2` e `max_depth=3`. Qual profundidade parece ser melhor?

In [108…
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
import numpy as np

tree_clf_depth3 = None
scores_depth2 = None
scores_depth3 = None

# Crie o classificador com max_depth=3
tree_clf_depth3 = DecisionTreeClassifier(max_depth=3, random_state=42)

# Calcule os scores de validação cruzada para max_depth=2
tree_clf_depth2 = DecisionTreeClassifier(max_depth=2, random_state=42)
scores_depth2 = cross_val_score(tree_clf_depth2, X_train, y_train, cv=5,

# Calcule os scores de validação cruzada para max_depth=3
scores_depth3 = cross_val_score(tree_clf_depth3, X_train, y_train, cv=5,
```

```
print(f"Acurácia média (max_depth=2): {np.mean(scores_depth2):.4f}")
print(f"Acurácia média (max_depth=3): {np.mean(scores_depth3):.4f}")

# <<< SEU CÓDIGO AQUI para imprimir qual profundidade é melhor e por quê
if np.mean(scores_depth3) > np.mean(scores_depth2):
    print("max_depth=3 teve melhor desempenho, indicando que uma árvore m
else:
    print("max_depth=2 teve desempenho igual ou superior, sugerindo que u
# <<< FIM DO SEU CÓDIGO >>>
```

```
Acurácia média (max_depth=2): 0.7468
Acurácia média (max_depth=3): 0.7746
max_depth=3 teve melhor desempenho, indicando que uma árvore mais profunda
capturou melhor os padrões dos dados.
```

8. Avaliar o Melhor Modelo no Conjunto de Teste

Sua tarefa:

1. Selecione o melhor modelo com base nos resultados da validação cruzada.
2. **Retreine** o melhor modelo usando **todo** o conjunto de treinamento (`X_train` , `y_train`).
3. Faça previsões no conjunto de teste (`X_test`).
4. Calcule e imprima a acurácia, precisão, recall e F1-score no conjunto de teste. Use `average='weighted'` para as métricas multiclasse.

In [109…

```
best_tree_clf = None
y_pred_test = None
accuracy = None
precision = None
recall = None
f1 = None

# <<< SEU CÓDIGO AQUI >>>
# 1 & 2: Selecionar e retreinar o melhor modelo (max_depth=3)
best_tree_clf = DecisionTreeClassifier(max_depth=3, random_state=42)
best_tree_clf.fit(X_train, y_train)
...

# 3: Fazer previsões no conjunto de teste
y_pred_test = best_tree_clf.predict(X_test)

# 4: Calcular métricas
accuracy = accuracy_score(y_test, y_pred_test)
precision = precision_score(y_test,  y_pred_test, average='weighted')
recall = recall_score(y_test, y_pred_test, average='weighted')
f1 = f1_score(y_test, y_pred_test, average='weighted')
# <<< FIM DO SEU CÓDIGO >>>

print(f"Métricas do melhor modelo (max_depth=3) no conjunto de teste:")
print(f"  Acurácia: {accuracy:.4f}")
print(f"  Precisão: {precision:.4f}")
print(f"  Recall:   {recall:.4f}")
print(f"  F1-Score: {f1:.4f}")
```

```
Métricas do melhor modelo (max_depth=3) no conjunto de teste:
  Acurácia: 0.8889
  Precisão: 0.9116
  Recall:   0.8889
  F1-Score: 0.8905
```

Conclusão

Você treinou com sucesso um classificador de Árvore de Decisão, visualizou sua estrutura e limites de decisão, usou validação cruzada para escolher a melhor profundidade e avaliou o modelo final no conjunto de teste.

Exercício prático 10_ Árvores de decisão - regularização e regressão

October 29, 2025

1 Exercício de Árvore de Decisão: Regressão de Preços de Imóveis na Califórnia

Objetivo: Neste exercício, você irá treinar, avaliar e regularizar um regressor de Árvore de Decisão usando o dataset 'California housing'. Você aplicará conceitos do Capítulo 6, focando em hiperparâmetros de regularização, regressão com árvores, avaliação e o efeito do pré-processamento (PCA).

Tópicos Cobertos: - Treinamento de `DecisionTreeRegressor` - Avaliação com Validação Cruzada (RMSE) - Hiperparâmetros de Regularização (`min_samples_leaf`) - Efeito do Pré-processamento (`StandardScaler`, `PCA`) - Avaliação Final no Conjunto de Teste

1.1 1. Configuração e Importações

Importar as bibliotecas necessárias.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline

# Para plotar figuras bonitas
%matplotlib inline
import matplotlib as mpl
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Definir estado aleatório para reprodutibilidade
RANDOM_STATE = 42
```

1.2 2. Carregar os Dados

Carregar o dataset California housing.

[2]:
```
housing = fetch_california_housing()
X = housing.data
y = housing.target

print("Shape de X:", X.shape)
print("Shape de y:", y.shape)
```

```
Shape de X: (20640, 8)
Shape de y: (20640,)
```

1.3 3. Dividir os Dados em Conjuntos de Treino e Teste

Sua tarefa: Use `train_test_split` para dividir os dados `X` e `y` em conjuntos de treinamento e teste. Use 20% dos dados para teste e defina `random_state=RANDOM_STATE`.

[3]:
```
X_train, X_test, y_train, y_test = None, None, None, None # <<< SUBSTITUA None␣
↪PELO SEU CÓDIGO >>>

# <<< SEU CÓDIGO AQUI >>>
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
# <<< FIM DO SEU CÓDIGO >>>

print("Tamanho do treino:", len(X_train))
print("Tamanho do teste:", len(X_test))
```

```
Tamanho do treino: 16512
Tamanho do teste: 4128
```

1.4 4. Treinar e Avaliar um Modelo Base (Sem Pré-processamento)

Sua tarefa: 1. Crie um `DecisionTreeRegressor` com `random_state=RANDOM_STATE` (parâmetros default). 2. Avalie-o usando validação cruzada (`cross_val_score`) com 5 folds (`cv=5`). Use a métrica 'neg_mean_squared_error'. 3. Calcule e imprima o RMSE médio a partir dos scores da validação cruzada (lembre-se que os scores são negativos e são MSE, não RMSE).

[4]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
import numpy as np

tree_reg_base = None
base_scores = None
base_rmse = None

# Gerar os dados quadráticos com ruído
```

```
np.random.seed(42)
X_quad = np.random.rand(200, 1) - 0.5
y_quad = X_quad ** 2 + 0.025 * np.random.randn(200, 1)

# 1. Criar o regressor
tree_reg_base = DecisionTreeRegressor(random_state=42)

# 2. Avaliar com validação cruzada (MSE negativo)
base_scores = cross_val_score(tree_reg_base, X_quad, y_quad, cv=5,␣
 ↪scoring='neg_mean_squared_error')

# 3. Calcular RMSE médio
base_rmse = np.sqrt(-base_scores.mean())

print(f"RMSE Médio (Modelo Base): {base_rmse:.4f}")
```

```
RMSE Médio (Modelo Base): 0.0328
```

1.5 5. Aplicar Pré-processamento (StandardScaler + PCA) e Avaliar

Árvores de decisão não exigem escalonamento, mas são sensíveis à rotação dos dados. Vamos ver se aplicar `StandardScaler` seguido por `PCA` melhora o desempenho.

Sua tarefa: 1. Crie um `Pipeline` que primeiro aplica `StandardScaler` e depois `PCA` (mantenha 95% da variância, `n_components=0.95`, e defina `random_state=RANDOM_STATE` no PCA). 2. Ajuste (fit) o pipeline aos dados de **treinamento** (`X_train`). 3. Transforme os dados de treinamento usando o pipeline ajustado. 4. Crie um `DecisionTreeRegressor` (com `random_state=RANDOM_STATE`). 5. Avalie este regressor nos dados de treinamento **pré-processados** usando validação cruzada (`cross_val_score`, `cv=5`, `scoring='neg_mean_squared_error'`). 6. Calcule e imprima o RMSE médio. 7. Compare com o RMSE do modelo base e comente se o pré-processamento ajudou.

[5]:
```
preprocessing_pipeline = None
X_train_prepared = None
tree_reg_pca = None
pca_scores = None
pca_rmse = None

# <<< SEU CÓDIGO AQUI >>>
# 1. Criar o pipeline de pré-processamento
preprocessing_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.95, random_state=42))
])

# 2. Ajustar o pipeline aos dados de treinamento
preprocessing_pipeline.fit(X_train)
```

```
# 3. Transformar os dados de treinamento
X_train_prepared = preprocessing_pipeline.transform(X_train)

# 4. Criar o regressor
tree_reg_pca = DecisionTreeRegressor(random_state=42)

# 5. Avaliar nos dados pré-processados
pca_scores = cross_val_score(tree_reg_pca, X_train_prepared, y_train, cv=5,␣
 ↪scoring='neg_mean_squared_error')

# 6. Calcular RMSE médio
pca_rmse = np.sqrt(-pca_scores.mean())
# <<< FIM DO SEU CÓDIGO >>>

print(f"RMSE Médio (com PCA): {pca_rmse:.4f}")

if pca_rmse < base_rmse:
    print("\nO pré-processamento com StandardScaler e PCA melhorou ligeiramente␣
 ↪o RMSE médio.")
else:
    print("\nO pré-processamento com StandardScaler e PCA NÃO melhorou (ou␣
 ↪piorou) o RMSE médio.")
print("Continuaremos usando os dados pré-processados para os próximos passos.")
```

```
RMSE Médio (com PCA): 0.9586

O pré-processamento com StandardScaler e PCA NÃO melhorou (ou piorou) o RMSE
médio.
Continuaremos usando os dados pré-processados para os próximos passos.
```

1.6 6. Ajustar `min_samples_leaf` Usando Validação Cruzada

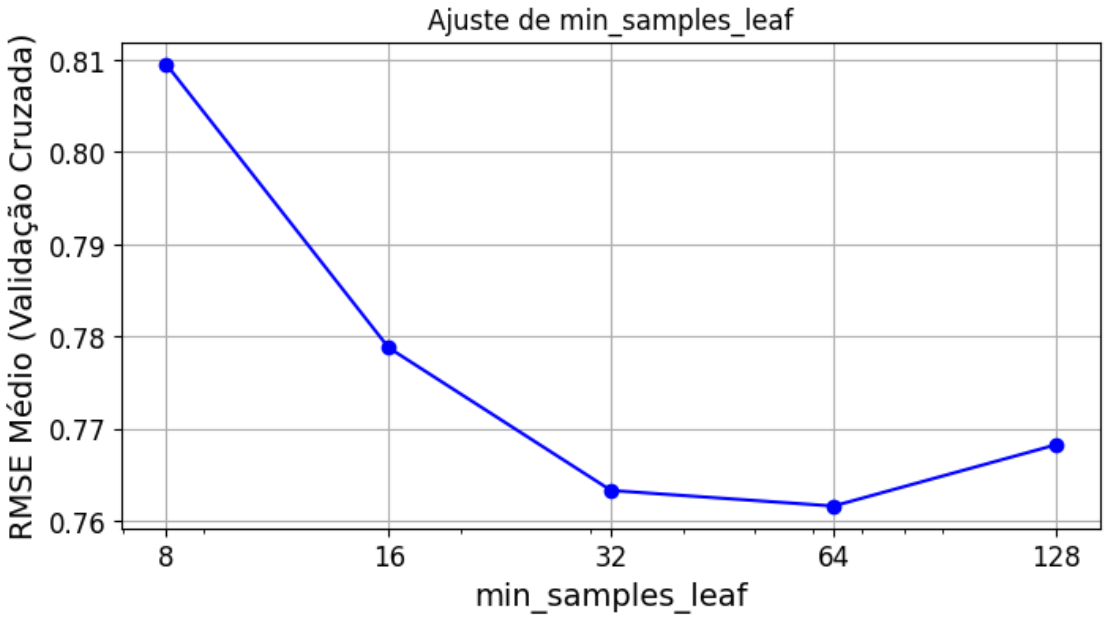
Agora vamos encontrar o melhor valor para o hiperparâmetro de regularização `min_samples_leaf`.

Sua tarefa: 1. Defina uma lista de valores para testar: `min_samples_leaf_values = [2, 4, 8, 16, 32]`. 2. Para cada valor na lista: a. Crie um `DecisionTreeRegressor` com o `min_samples_leaf` atual e `random_state=RANDOM_STATE`. b. Avalie-o nos dados de treinamento **pré-processados** (`X_train_prepared`) usando `cross_val_score` (`cv=5`, `scoring='neg_mean_squared_error'`). c. Calcule o RMSE médio e armazene-o. 3. Encontre o valor de `min_samples_leaf` que resultou no menor RMSE médio. 4. Imprima o melhor valor de `min_samples_leaf` e o RMSE correspondente.

[6]:
```
min_samples_leaf_values = [8, 16, 32, 64, 128]
rmse_scores = []
best_min_samples_leaf = None
best_rmse = float('inf')

for min_samples in min_samples_leaf_values:
```

```
# <<< SEU CÓDIGO AQUI >>>
    # 2a. Criar o regressor
    reg = DecisionTreeRegressor(min_samples_leaf=min_samples, random_state=42)

    # 2b. Avaliar com validação cruzada
    scores = cross_val_score(reg, X_train_prepared, y_train, cv=5,␣
 ↪scoring='neg_mean_squared_error')

    # 2c. Calcular e armazenar RMSE médio
    current_rmse = np.sqrt(-scores.mean())
# <<< FIM DO SEU CÓDIGO >>>

    rmse_scores.append(current_rmse)
    print(f"min_samples_leaf={min_samples}, RMSE Médio={current_rmse:.4f}")

    # 3. Encontrar o melhor valor
    if current_rmse < best_rmse:
        best_rmse = current_rmse
        best_min_samples_leaf = min_samples

# 4. Imprimir o melhor resultado
print(f"\nMelhor min_samples_leaf: {best_min_samples_leaf}")
print(f"Melhor RMSE Médio de Validação Cruzada: {best_rmse:.4f}")

# Plotar os resultados
plt.figure(figsize=(8, 4))
plt.plot(min_samples_leaf_values, rmse_scores, "bo-")
plt.xscale('log')
plt.xticks(min_samples_leaf_values, min_samples_leaf_values)
plt.xlabel("min_samples_leaf")
plt.ylabel("RMSE Médio (Validação Cruzada)")
plt.title("Ajuste de min_samples_leaf")
plt.grid(True)
plt.show()
```

```
min_samples_leaf=8, RMSE Médio=0.8095
min_samples_leaf=16, RMSE Médio=0.7788
min_samples_leaf=32, RMSE Médio=0.7633
min_samples_leaf=64, RMSE Médio=0.7616
min_samples_leaf=128, RMSE Médio=0.7683

Melhor min_samples_leaf: 64
Melhor RMSE Médio de Validação Cruzada: 0.7616
```



1.7 7. Treinar o Modelo Final e Avaliar no Conjunto de Teste

Finalmente, treine o modelo com o melhor hiperparâmetro encontrado e avalie-o no conjunto de teste pré-processado.

Sua tarefa: 1. Crie o regressor final (`DecisionTreeRegressor`) usando o `best_min_samples_leaf` encontrado e `random_state=RANDOM_STATE`. 2. Treine este modelo final no conjunto de treinamento **pré-processado completo** (`X_train_prepared`, `y_train`). 3. Transforme o conjunto de teste (`X_test`) usando o `preprocessing_pipeline` **ajustado**. 4. Faça previsões no conjunto de teste pré-processado. 5. Calcule e imprima o RMSE no conjunto de teste.

[ ]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import numpy as np

final_tree_reg = None
X_test_prepared = None
y_pred_test = None
final_rmse = None

# <<< SEU CÓDIGO AQUI >>>
# 1. Criar o regressor final
final_tree_reg = DecisionTreeRegressor(min_samples_leaf=best_min_samples_leaf,␣
↪random_state=42)

# 2. Treinar no conjunto de treino pré-processado completo
final_tree_reg.fit(X_train_prepared, y_train)
```

```python
# 3. Transformar o conjunto de teste
X_test_prepared = preprocessing_pipeline.transform(X_test)

# 4. Fazer previsões
y_pred_test = final_tree_reg.predict(X_test_prepared)

# 5. Calcular RMSE final
final_rmse = np.sqrt(mean_squared_error(y_test, y_pred_test))
# <<< FIM DO SEU CÓDIGO >>>

print(f"RMSE Final no Conjunto de Teste: {final_rmse:.4f}")
```

```
RMSE Final no Conjunto de Teste: 0.7624
```

1.8 Conclusão

Você treinou e avaliou um regressor de Árvore de Decisão, observou o efeito (potencialmente pequeno ou até negativo) do PCA neste caso, ajustou um hiperparâmetro de regularização (`min_samples_leaf`) e avaliou o desempenho final do modelo no conjunto de teste.