Conjuntos Regressão

November 11, 2025

[5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
```

[6]:
```
df = pd.read_csv("Salary_Data.csv")
df.head()
```

[6]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     float64
dtypes: float64(2)
memory usage: 612.0 bytes
```

[8]:
```
# Selecionar a variável independente (anos de experiência)
X = df[["YearsExperience"]]  # Variável independente

# Selecionar a variável dependente (salário)
```

```
y = df["Salary"]  # Variável dependente

# Criar um gráfico de dispersão (scatter plot)
plt.scatter(X, y)  # Criar gráfico de dispersão

# Adicionar título ao gráfico
plt.title("Salario vs Anos de Experiencia")  # Título do gráfico

# Adicionar rótulo ao eixo X
plt.xlabel("Anos de experiencia")  # Rótulo do eixo X

# Adicionar rótulo ao eixo Y
plt.ylabel("Salario")  # Rótulo do eixo Y

# Mostrar o gráfico
plt.show()  # Exibir o gráfico
```

[9]:
```
# Inicializar os escaladores
x_scaler = StandardScaler()
y_scaler = StandardScaler()
```

```
# Normalizar X e y
X_scaled = x_scaler.fit_transform(np.array(X).reshape(-1, 1))
y_scaled = y_scaler.fit_transform(np.array(y).reshape(-1, 1))

# Definir o tamanho do teste para 20% dos dados de treinamento
x_train, x_test, y_train, y_test = train_test_split(X_scaled, y_scaled,␣
 ↪test_size=0.2, random_state=6)
```

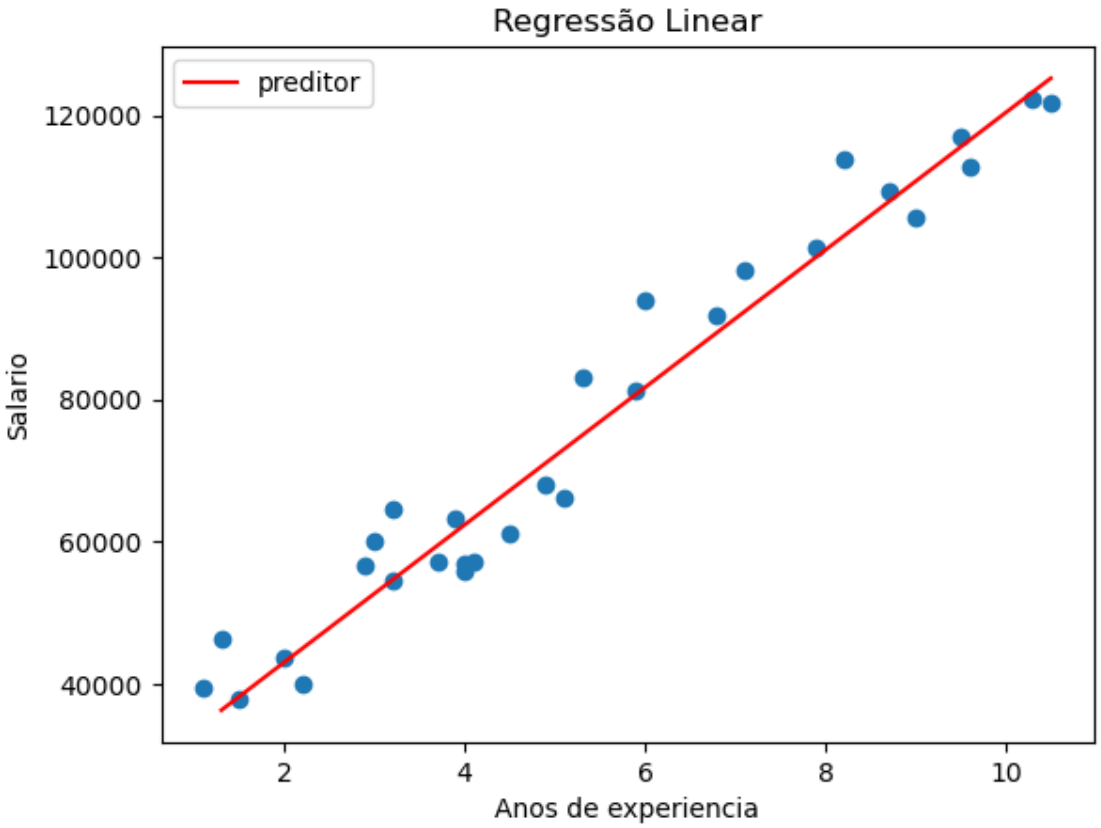
[10]:
```
# Instantiate linear regression model
lin_reg = LinearRegression()

lin_reg.fit(x_train,y_train)
```

[10]:
```
LinearRegression()
```

[11]:
```
# Fazer previsões no conjunto de teste usando o modelo de regressão linear
y_pred = lin_reg.predict(x_test)

# Calcular e imprimir o erro quadrático médio (RMSE)
print("RMSE: ", np.sqrt(mean_squared_error(y_test, y_pred)))

# Plotar a linha de previsão da regressão linear sobre os dados
# Criar um domínio de valores para o eixo X
x_domain = np.linspace(min(x_train), max(x_train), 100)

# Reverter a normalização das previsões e dos valores de X
y_pred_rescaled = y_scaler.inverse_transform(lin_reg.predict(x_domain))
x_rescaled = x_scaler.inverse_transform(x_domain)

# Criar uma nova figura para o gráfico
plt.figure()

# Criar um gráfico de dispersão dos dados originais
plt.scatter(X, y)

# Plotar a linha de previsão da regressão linear
plt.plot(x_rescaled, y_pred_rescaled, color="red", label='preditor')

# Adicionar rótulo ao eixo X
plt.xlabel("Anos de experiencia")

# Adicionar rótulo ao eixo Y
plt.ylabel("Salario")

# Adicionar título ao gráfico
plt.title("Regressão Linear")
```

```
# Adicionar legenda ao gráfico
plt.legend()

# Mostrar o gráfico
plt.show()
```

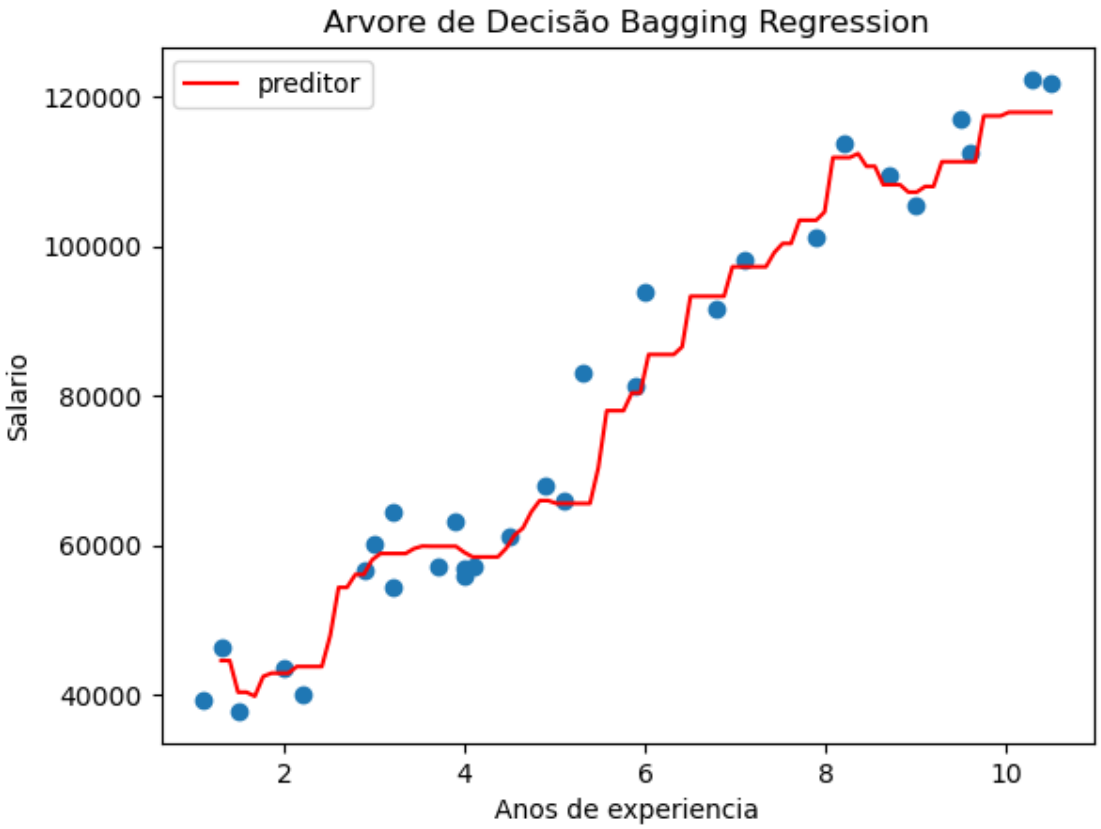
```
RMSE:  0.267786079046986
```



[12]:
```
# Importar a classe BaggingRegressor do sklearn
from sklearn.ensemble import BaggingRegressor

# Instanciar o modelo de regressão de árvore de decisão para usar como modelo␣
↪base
d_tree = DecisionTreeRegressor(max_depth=4)

# Instanciar o modelo BaggingRegressor com uma árvore de decisão como modelo␣
↪base
bag_reg = BaggingRegressor(estimator=d_tree)
```

```
# Treinar o modelo BaggingRegressor nos dados de treinamento
bag_reg.fit(x_train, y_train[:, 0])

# Fazer previsões nos dados de teste
y_pred = bag_reg.predict(x_test)

# Calcular e imprimir o erro quadrático médio (RMSE)
print("RMSE: ", np.sqrt(mean_squared_error(y_test, y_pred)))

# Plotar a linha de previsão da regressão bagging sobre os dados
# Criar um domínio de valores para o eixo X
x_domain = np.linspace(min(x_train), max(x_train), 100)

# Reverter a normalização das previsões e dos valores de X
y_pred_rescaled = y_scaler.inverse_transform(bag_reg.predict(x_domain).
 ↪reshape(-1, 1))
x_rescaled = x_scaler.inverse_transform(x_domain)

# Criar uma nova figura para o gráfico
plt.figure()

# Criar um gráfico de dispersão dos dados originais
plt.scatter(X, y)

# Plotar a linha de previsão da regressão bagging
plt.plot(x_rescaled, y_pred_rescaled, color="red", label='preditor')

# Adicionar rótulo ao eixo X
plt.xlabel("Anos de experiencia")

# Adicionar rótulo ao eixo Y
plt.ylabel("Salario")

# Adicionar título ao gráfico
plt.title("Arvore de Decisão Bagging Regression")

# Adicionar legenda ao gráfico
plt.legend()

# Mostrar o gráfico
plt.show()
```

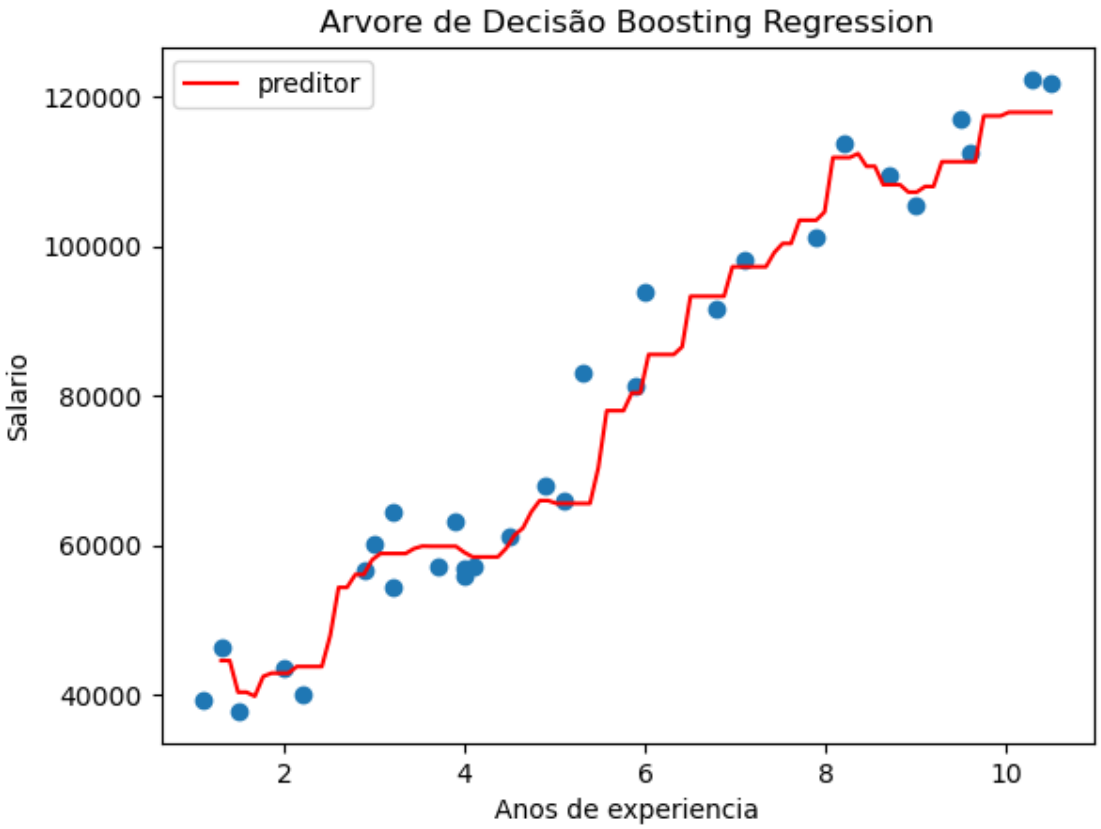
```
RMSE:  0.3143127101083037
```



[13]:
```
# Importar a classe AdaBoostRegressor do sklearn
from sklearn.ensemble import AdaBoostRegressor

# Instanciar o modelo de regressão de árvore de decisão para usar como modelo␣
 ↪base
d_tree = DecisionTreeRegressor(max_depth=3)

# Instanciar o modelo AdaBoostRegressor com uma árvore de decisão como modelo␣
 ↪base
bst_reg = AdaBoostRegressor(estimator=d_tree)

# Treinar o modelo AdaBoostRegressor nos dados de treinamento
bst_reg.fit(x_train, y_train[:, 0])

# Fazer previsões nos dados de teste
y_pred = bst_reg.predict(x_test)

# Calcular e imprimir o erro quadrático médio (RMSE)
print("RMSE: ", np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
# Plotar a linha de previsão da regressão boosting sobre os dados
# Criar um domínio de valores para o eixo X
x_domain = np.linspace(min(x_train), max(x_train), 100)

# Reverter a normalização das previsões e dos valores de X
y_pred_rescaled = y_scaler.inverse_transform(bag_reg.predict(x_domain).
 ↪reshape(-1, 1))
x_rescaled = x_scaler.inverse_transform(x_domain)

# Criar uma nova figura para o gráfico
plt.figure()

# Criar um gráfico de dispersão dos dados originais
plt.scatter(X, y)

# Plotar a linha de previsão da regressão boosting
plt.plot(x_rescaled, y_pred_rescaled, color="red", label='preditor')

# Adicionar rótulo ao eixo X
plt.xlabel("Anos de experiencia")

# Adicionar rótulo ao eixo Y
plt.ylabel("Salario")

# Adicionar título ao gráfico
plt.title("Arvore de Decisão Boosting Regression")

# Adicionar legenda ao gráfico
plt.legend()

# Mostrar o gráfico
plt.show()
```

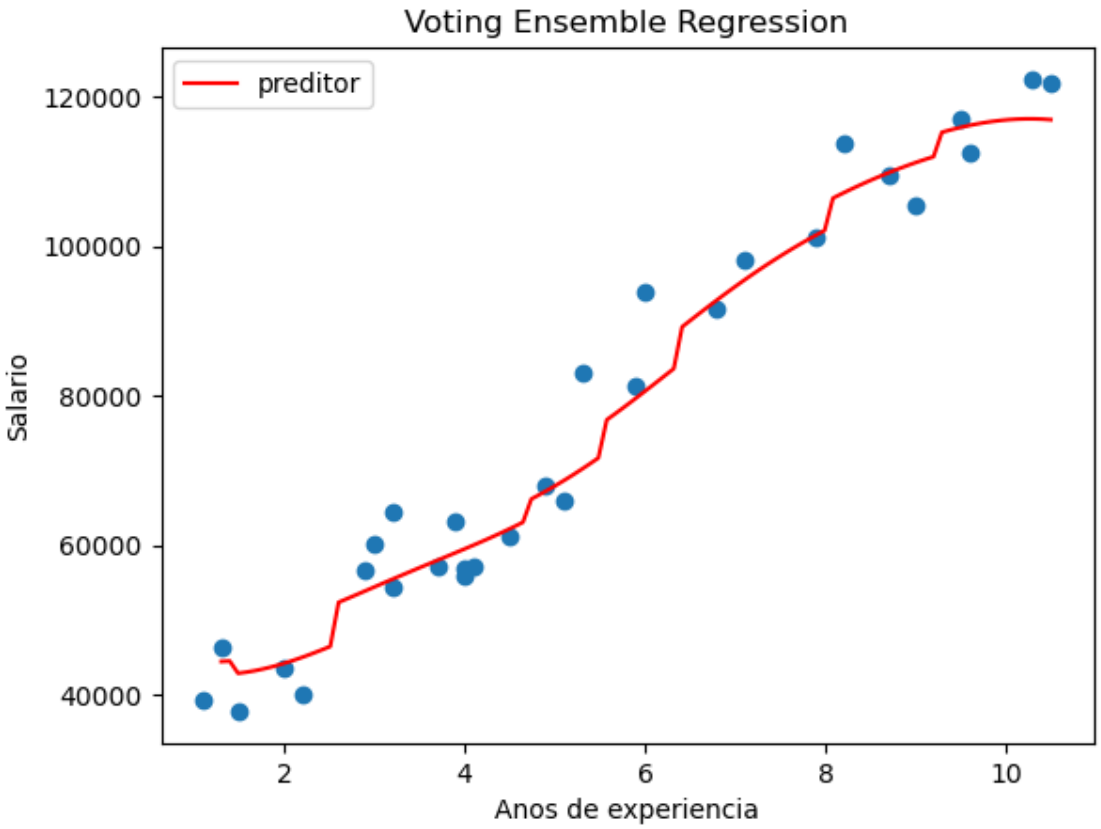
```
RMSE:  0.3282822314058341
```



[16]:
```python
# Importar a classe VotingRegressor do sklearn
from sklearn.ensemble import VotingRegressor

# Instanciar o modelo SVR
sv_reg = SVR()

# Definir os modelos base
models = [("LR", lin_reg), ("DT", d_tree), ("SVR", sv_reg)]

# Especificar pesos para a média ponderada dos modelos
model_weightings = np.array([0.1, 0.3, 0.6])

# Instanciar o modelo VotingRegressor com os modelos base e os pesos
v_reg = VotingRegressor(estimators=models, weights=model_weightings)

# Treinar o modelo VotingRegressor nos dados de treinamento
v_reg.fit(x_train, y_train[:, 0])

# Fazer previsões nos dados de teste
y_pred = v_reg.predict(x_test)
```

```
# Calcular e imprimir o erro quadrático médio (RMSE)
print("RMSE: ", np.sqrt(mean_squared_error(y_test, y_pred)))

# Plotar a linha de previsão da regressão por votação sobre os dados
# Criar um domínio de valores para o eixo X
x_domain = np.linspace(min(x_train), max(x_train), 100)

# Reverter a normalização das previsões e dos valores de X
y_pred_rescaled = y_scaler.inverse_transform(v_reg.predict(x_domain).
 ↪reshape(-1, 1))
x_rescaled = x_scaler.inverse_transform(x_domain)

# Criar uma nova figura para o gráfico
plt.figure()

# Criar um gráfico de dispersão dos dados originais
plt.scatter(X, y)

# Plotar a linha de previsão da regressão por votação
plt.plot(x_rescaled, y_pred_rescaled, color="red", label='preditor')

# Adicionar rótulo ao eixo X
plt.xlabel("Anos de experiencia")

# Adicionar rótulo ao eixo Y
plt.ylabel("Salario")

# Adicionar título ao gráfico
plt.title("Voting Ensemble Regression")

# Adicionar legenda ao gráfico
plt.legend()

# Mostrar o gráfico
plt.show()
```

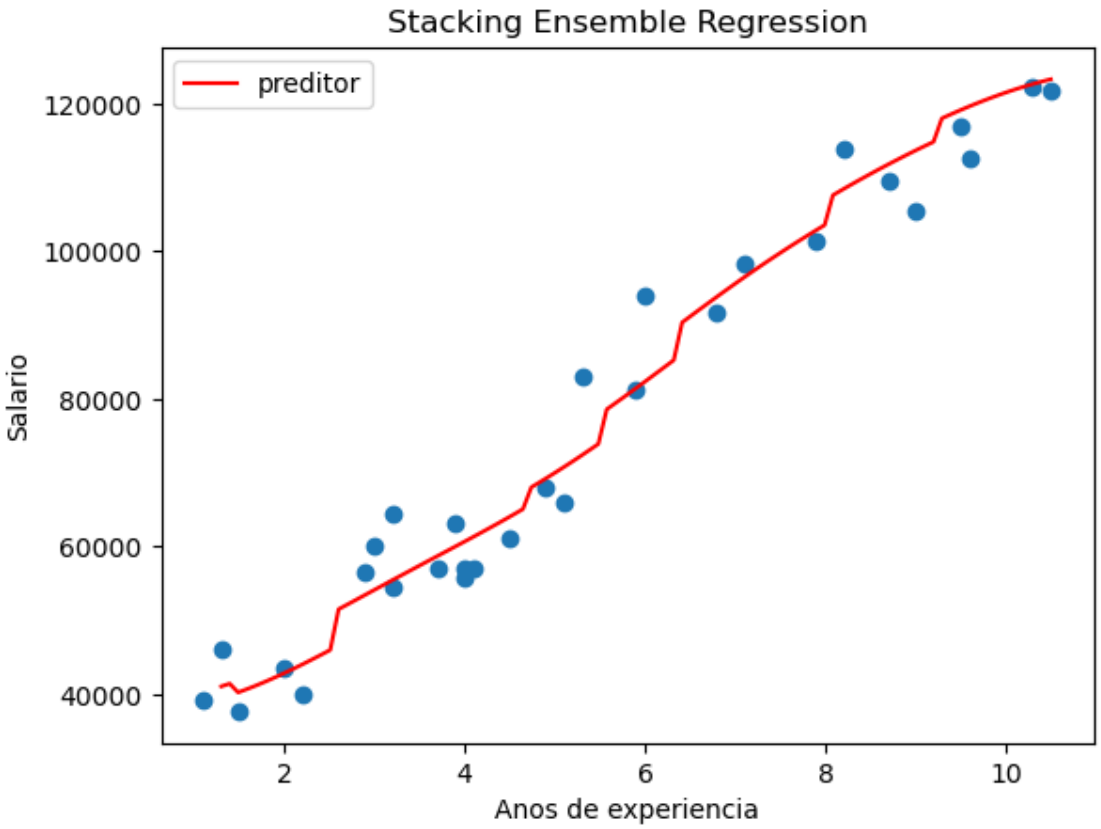
```
RMSE:  0.31939669309902596
```



[15]:
```
from sklearn.ensemble import StackingRegressor

# Para maior clareza, declaramos nossa lista de modelos novamente aqui
models = [("LR", lin_reg), ("DT", d_tree), ("SVR", sv_reg)]

# Em vez de escolher ponderações de modelo, agora declaramos o modelo␣
 ↪meta-learner para nosso conjunto de empilhamento
meta_learner_reg = LinearRegression()

# Inicializar o Stacking Regressor com os modelos base e o meta-learner
s_reg = StackingRegressor(estimators=models, final_estimator=meta_learner_reg)

# Treinar o modelo Stacking Regressor nos dados de treinamento
s_reg.fit(x_train, y_train[:, 0])

# Fazer previsões nos dados de teste
y_pred = s_reg.predict(x_test)

# Calcular e imprimir o erro quadrático médio (RMSE)
print("RMSE: ", np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
# Plotar a linha de previsão da regressão de empilhamento sobre os dados
# Criar um domínio de valores para o eixo X
x_domain = np.linspace(min(x_train), max(x_train), 100)

# Reverter a normalização das previsões e dos valores de X
y_pred_rescaled = y_scaler.inverse_transform(s_reg.predict(x_domain).
 ↪reshape(-1, 1))
x_rescaled = x_scaler.inverse_transform(x_domain)

# Criar uma nova figura para o gráfico
plt.figure()

# Criar um gráfico de dispersão dos dados originais
plt.scatter(X, y)

# Plotar a linha de previsão da regressão de empilhamento
plt.plot(x_rescaled, y_pred_rescaled, color="red", label='preditor')

# Adicionar rótulo ao eixo X
plt.xlabel("Anos de experiencia")

# Adicionar rótulo ao eixo Y
plt.ylabel("Salario")

# Adicionar título ao gráfico
plt.title("Stacking Ensemble Regression")

# Adicionar legenda ao gráfico
plt.legend()

# Mostrar o gráfico
plt.show()
```

```
RMSE:  0.2768616507476488
```



0.1 Vantagens dos Modelos Ensemble

Os modelos de conjunto têm várias vantagens sobre os modelos individuais, incluindo:

- Precisão Aprimorada: Modelos de conjunto podem melhorar a precisão das previsões em comparação a um único modelo. Ao combinar vários modelos, o conjunto pode alavancar os pontos fortes de cada modelo individual enquanto atenua suas fraquezas, resultando em um modelo mais preciso e robusto.
- Overfitting Reduzido: Modelos ensemble podem ajudar a reduzir o risco de overfitting, que é um problema comum em machine learning, onde um modelo aprende os dados de treinamento muito bem e falha em generalizar para novos dados. Ao combinar vários modelos, o ensemble pode reduzir a variância nas previsões, tornando o modelo mais robusto e menos propenso a overfitting.
- Estabilidade Aumentada: Modelos ensemble tendem a ser mais estáveis e confiáveis do que modelos únicos, pois são menos sensíveis a flutuações nos dados de treinamento ou parâmetros do modelo. Isso ocorre porque o ensemble agrega as previsões de vários modelos, o que pode suavizar qualquer ruído ou erro nas previsões individuais.
- Flexibilidade: Modelos ensemble podem ser usados com uma ampla gama de algoritmos e técnicas, e podem ser adaptados a problemas e tipos de dados específicos. Essa flexibilidade permite que modelos ensemble sejam aplicados em uma ampla gama de domínios e aplicações.

- Fácil de Implementar: Modelos ensemble são relativamente fáceis de implementar, especialmente com a disponibilidade de bibliotecas populares de machine learning que fornecem suporte integrado para métodos ensemble. Isso torna fácil para os profissionais incorporar métodos ensemble em seus fluxos de trabalho e melhorar o desempenho de seus modelos.

No geral, os modelos ensemble são uma ferramenta poderosa para melhorar a precisão e a robustez dos modelos de machine learning. Eles são amplamente usados em muitos domínios e aplicações, e se tornaram uma técnica básica na caixa de ferramentas de machine learning.

0.2 Desvantagens dos Métodos Ensemble

- Complexidade: Os métodos de conjunto podem ser mais complexos do que os modelos individuais, o que pode torná-los mais difíceis de entender e interpretar.
- Tempo de Computação: Construir um conjunto de modelos pode ser computacionalmente caro, especialmente para grandes conjuntos de dados e modelos complexos.
- Seleção de Aprendizes Fracas: O desempenho dos métodos de conjunto depende muito da escolha de aprendizes fracos. Se os aprendizes fracos não forem diversos o suficiente ou não forem apropriados para o problema, o método de conjunto pode não ter um bom desempenho.