Árvores de Decisão

Regularização e Regressão

Slides extraídos do livro "*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*" de Aurélien Géron

Sumário

Sumário

Hiperparâmetros de Regularização

- Árvores de Decisão são **modelos não paramétricos**, então:
  - Não há um número fixo de parâmetros a ajustar no treino.
  - Fazem poucas suposições sobre os dados.
  - Se não forem restringidas, a estrutura da árvore se adaptará demais aos dados (**overfitting**).
- Para evitar o overfitting, precisamos restringir a Árvore de Decisão durante o treinamento (**regularização**).
- O hiperparâmetro de regularização mais comum é restringir a profundidade máxima da árvore, controlado pelo hiperparâmetro `max_depth`. Reduzi-lo regularizará o modelo.

Outros Hiperparâmetros de Regularização

A classe `DecisionTreeClassifier` possui outros parâmetros que restringem a forma da Árvore de Decisão:

- `max_features`: Número máximo de features que são avaliadas para divisão em cada nó.
- `max_leaf_nodes`: Número máximo de nós folha.
- `min_samples_split`: Número mínimo de amostras que um nó deve ter para que possa ser dividido.
- `min_samples_leaf`: Número mínimo de amostras que um nó folha deve ter para ser criado.
- `min_weight_fraction_leaf`: O mesmo que `min_samples_leaf`, mas expresso como uma fração do número total de instâncias ponderadas.

Regra Geral

Aumentar os hiperparâmetros `min_*` ou reduzir os hiperparâmetros `max_*` regularizará o modelo.

Exemplo de Regularização

```
from sklearn.datasets import make_moons

X_moons, y_moons = make_moons(n_samples=150, noise=0.2, random_state=42)

tree_clf1 = DecisionTreeClassifier(random_state=42)
tree_clf2 = DecisionTreeClassifier(min_samples_leaf=5, random_state=42)
tree_clf1.fit(X_moons, y_moons)
tree_clf2.fit(X_moons, y_moons)
```

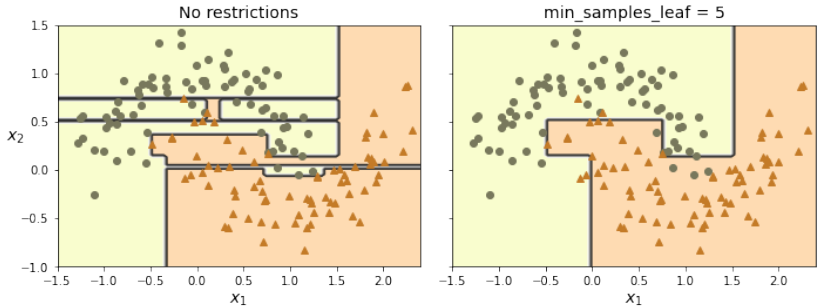

Figure: Limites de decisão de árvore não regularizada (esquerda) e regularizada (direita).

Verificando o Desempenho

O modelo não regularizado à esquerda está claramente com overfitting. O modelo regularizado à direita provavelmente generalizará melhor. Podemos verificar isso avaliando ambas as árvores em um conjunto de teste.

```
>>> X_moons_test, y_moons_test = make_moons(n_samples=1000, noise=0.2, random_state=43)
...
>>> tree_clf1.score(X_moons_test, y_moons_test)
0.898
>>> tree_clf2.score(X_moons_test, y_moons_test)
0.92
```

De fato, a segunda árvore tem uma acurácia melhor no conjunto de teste.

Sumário

Árvores de Decisão para Regressão

As Árvores de Decisão também são capazes de realizar tarefas de regressão.
Vamos construir uma árvore de regressão usando a classe `DecisionTreeRegressor` do Scikit-Learn, treinando-a em um dataset quadrático ruidoso com `max_depth=2`.

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor

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)

tree_reg = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg.fit(X_quad, y_quad)
```

Árvores de Decisão para Regressão

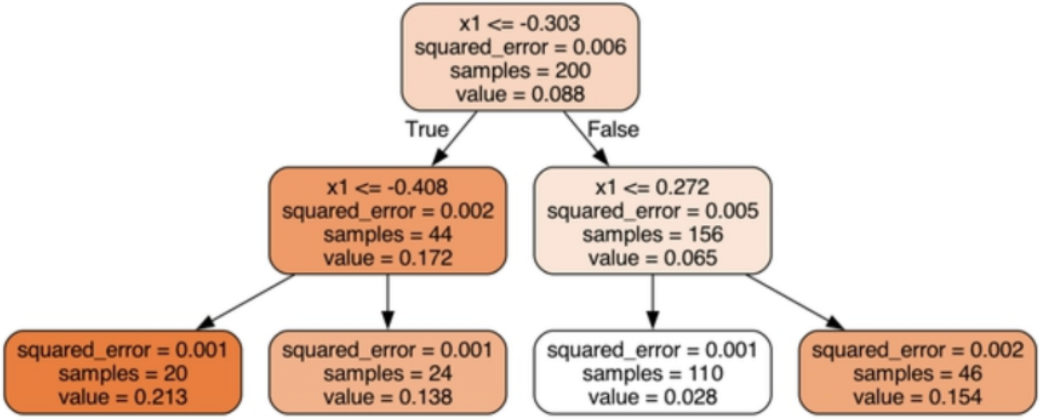

Figure: Uma Árvore de Decisão para regressão.

Previsões da Árvore de Regressão

- Na regressão prevemos um valor, ao invés de uma classe.
- Ex.: para nova instância com $x_1 = 0.2$, a árvore percorre os nós e chega em nó folha que prevê o valor 0.028.
- Esta previsão é o **valor médio do alvo** das 110 instâncias de treinamento associadas a este nó folha.

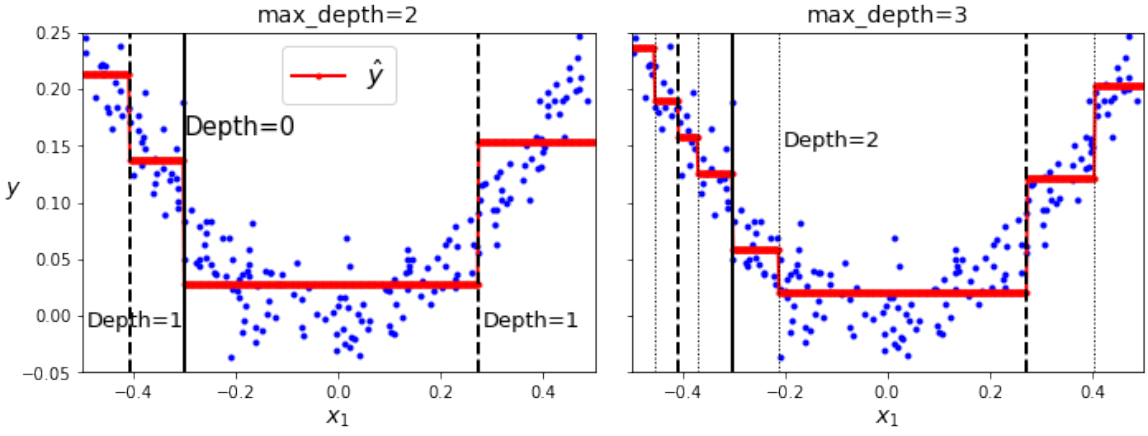

Figure: Previsões de dois modelos de Árvore de Decisão para regressão.

Algoritmo CART para Regressão

O algoritmo CART funciona de forma semelhante à classificação, mas, em vez de tentar dividir o conjunto de treinamento de uma forma que minimize a impureza, ele agora tenta dividir o conjunto de treinamento de uma forma que minimize o **MSE (Erro Quadrático Médio)**.

Equação 6-4. Função de custo CART para regressão

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}}$$

onde

$$\text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{\text{node}} - y^{(i)})^2 / m_{\text{node}}$$

$$\hat{y}_{\text{node}} = \sum_{i \in \text{node}} y^{(i)} / m_{\text{node}}$$

Algoritmo CART para Regressão

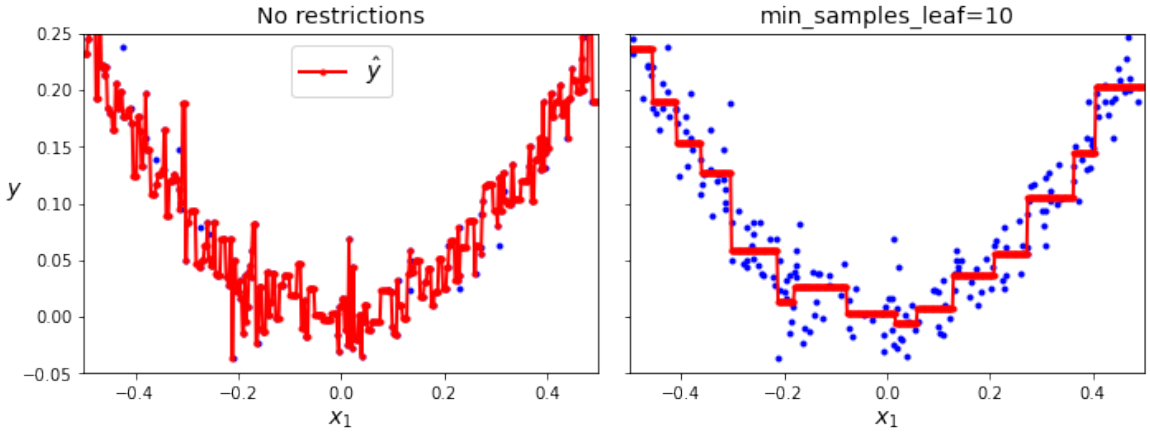

Figure: Previsões de uma árvore de regressão não regularizada (esquerda) e uma regularizada (direita).

Sumário

Sensibilidade à Orientação dos Eixos

- Árvores de Decisão produzem limites de decisão ortogonais (divisões são perpendiculares a um eixo). Isto as torna sensíveis a rotações dos dados.

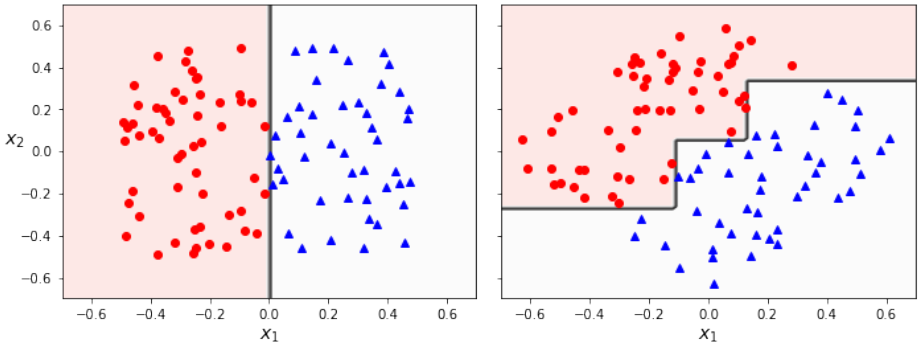

Figure: Sensibilidade à rotação do conjunto de treinamento.

- À esquerda, uma Árvore de Decisão pode dividir facilmente o dataset.
- À direita, após o dataset ser rotacionado em 45°, o limite de decisão parece desnecessariamente complicado.
- Embora ambas se ajustem perfeitamente aos dados, é muito provável que o da direita não generalize bem.

Mitigando a Sensibilidade à Rotação

Uma maneira de limitar este problema é escalar os dados e aplicar **Análise de Componentes Principais (PCA)**.

- O PCA rotaciona os dados de modo a reduzir a correlação entre as features, geralmente ajudando as árvores.

```
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

pca_pipeline = make_pipeline(StandardScaler(), PCA())
X_iris_rotated = pca_pipeline.fit_transform(X_iris)
tree_clf_pca = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf_pca.fit(X_iris_rotated, y_iris)
```

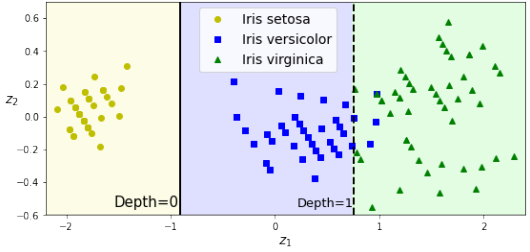

Figure: Limites de decisão em dados escalados e rotacionados por PCA.

Alta Variância

- O principal problema com as Árvores de Decisão é que elas têm uma **variância bastante alta**.
- Ou seja, pequenas mudanças nos hiperparâmetros ou nos dados podem produzir modelos muito diferentes.
- Como o algoritmo de treino no Sklearn é estocástico, repetir o treino com os mesmos dados pode produzir um modelo muito diferente (a menos que você defina o hiperparâmetro `random_state`).

Solução

Felizmente, ao fazer a média das previsões de muitas árvores, é possível reduzir significativamente a variância. Tal conjunto de árvores é chamado de **Random Forest**, um dos tipos de modelos mais poderosos disponíveis hoje.

Fim