# HOML - Chapter 6 - Decision Trees

## A taxa de Gini:

A taxa de Gini mede o quão preciso uma Árvore de Decisão é. Existe uma taxa de Gini para cada folha.

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$$

1 – (0/54)^2 – (49/54)^2 – (5/54)^2 ≈ 0.168

## Probabilidades:

Árvores de Decisão podem ser utilizadas para calcular probabilidades, no exemplo anterior, seria 0/54 para a c1, 49/54 para c2 e 5/54 para c3

Esse cálculo apresenta uma falha: independente da localização do ponto ao qual desejamos calcular a probabilidade, essa probabilidade permanecerá a mesma.

## The CART Training Algorithm

CART - Classification and Regression Tree

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

$$\text{where} \begin{cases} G_{\text{left/right}} \text{ measures the impurity of the left/right subset,} \\ m_{\text{left/right}} \text{ is the number of instances in the left/right subset.} \end{cases}$$

A DT tentará minimizar essa função de custo

## Complexidade

It is really fast compared to other algorithms

Prediction:

$$O(\log_2(m))$$

Training.

$$O(n \times m \log_2(m)).$$

Where m is the number of instances and n is the number of features

## Regularização

De maneira geral, as árvores de decisão são bastante flexíveis, elas podem se moldar muito bem aos dados, o que infelizmente pode acabar gerando problemas de overfitting, assim muitas vezes, é necessário que haja uma regularização.

Para regulalizarmos as DT, podemos diminuir o max depth (que por padrão é infinito)

Outros parâmetros que podemos utilizar para regularizar:

- min_samples_split (the minimum number of samples a node must have before it can be split),
- min_samples_leaf (the minimum number of samples a leaf node must have),
- min_weight_fraction_leaf (same as min_samples_leaf but expressed as a fraction of the total number of weighted instances),
- max_leaf_nodes (the maximum number of leaf nodes),
- and max_features (the maximum number of features that are evaluated for splitting at each node).
- Increasing min_* hyperparameters or reducing max_* hyperparameters will regularize the model.

## Regression:

DT também podem funcionar com regressão.

Divide em folhas e tira a média de cada folha para retornar o resultado

Neste caso também pode dar overfitting, e temos de regularizar de alguma maneira.

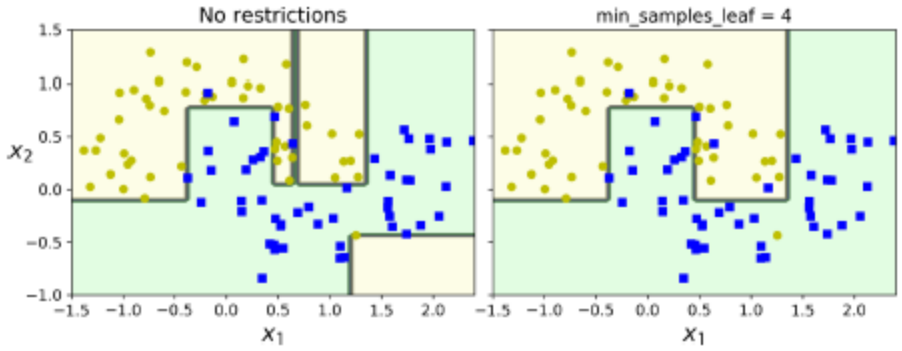


*Figure 6-3. Regularization using min_samples_leaf*

## Instabilidade:

"Decision Trees love orthogonal decision boundaries (all splits are perpendicular to an axis), which

makes them sensitive to training set rotation"

Um outro problema é que ele é extremamente sensível, as vezes pequenas mudanças podem acarretar em diferenças gigantescas no modelo.

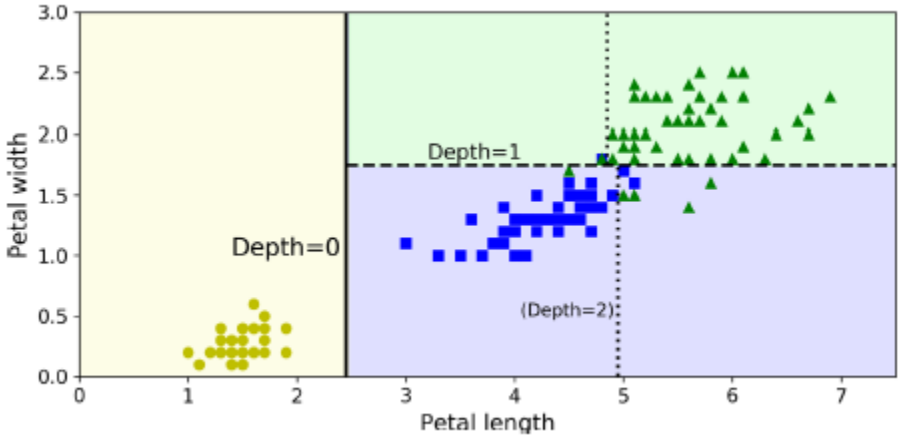


*Figure 6-2. Decision Tree decision boundaries*

Retiramos apenas uma instancia e olha o tanto que mudou!

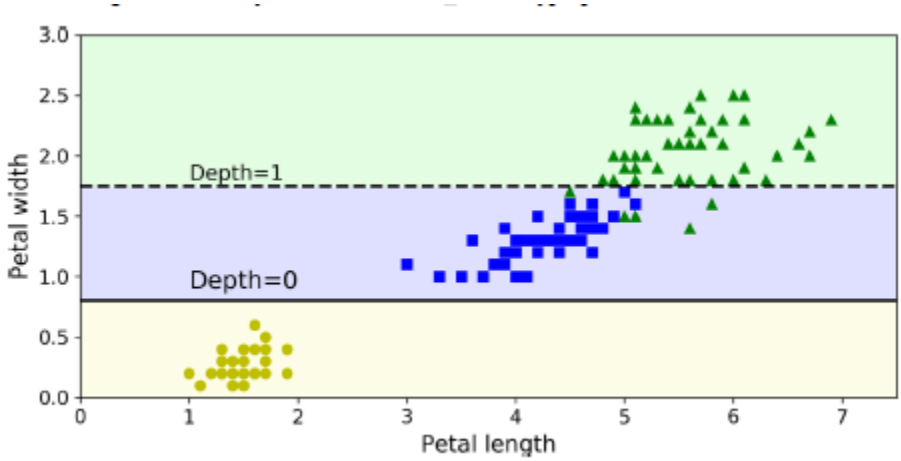


*Figure 6-8. Sensitivity to training set details*

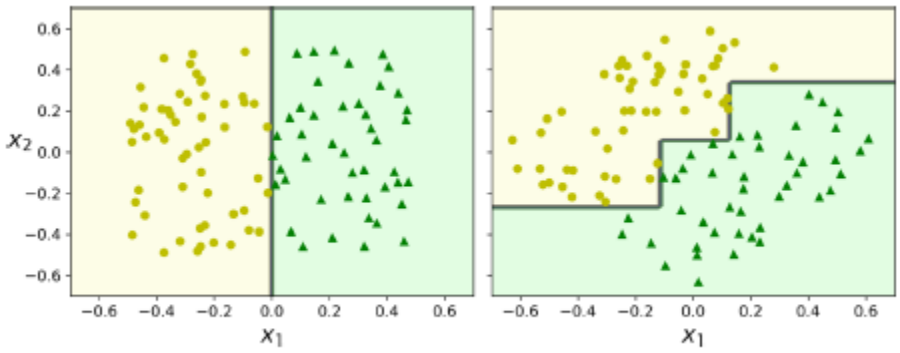


*Figure 6-7. Sensitivity to training set rotation*

# Exercises

- **What is the approximate depth of a Decision Tree trained (without restrictions) on a
  training set with one million instances?**

If we let our Decision Tree be trained without any restrictions, we will get a node for every instance and since the number of nodes equals 2 to the potency of the depth of the tree, the depth of the tree will be 20

$log(1000)/log(2)$ = 20

- **Is a node's Gini impurity generally lower or greater than its parent's? Is it generally
  lower/greater, or always lower/greater?**

Well, let's check it out.

We already know that it tends to go lower, else it would not make any sense, the lower the better, and a decision tree with a high Gini impurity would be useless.

Is there a situation when the Gini stays the same?

Let's suppose we have 100 instances, and the Gini impurity is 1- (50/100)² - (50/100)² = 0.5

Now let's suppose that the Gini impurity in the next node is 1 - (25/50)² - (25/50)² = 0.5. It has stayed the same. So we can already discard the "always" options.

But is there a situation that it can increase?

Yes there is, if we go from 98/2 (extremely low gini) and in the next node we go to 20/78, we have increased the Gini impurity

So it is generally lower.

- **If a Decision Tree is overfitting the training set, is it a good idea to try decreasing
  max_depth ?**

Yes it is. A major problem with unrestrained decision trees is that it will most probably overfit, so in order to make it more generalized we need to constrain it. One of the possible ways to do it is by decreasing its max depth

- **If a Decision Tree is underfitting the training set, is it a good idea to try scaling the input features?**

No, Decision Trees are not sensitive to scaling.

- **If it takes one hour to train a Decision Tree on a training set containing 1 million instances, roughly how much time will it take to train another Decision Tree on a training set containing 10 million instances?**

The training complexity equals $n * m * \log_2 m$

$1h = n * 10^6 * \log_2 10^6 \rightarrow x = n * 10^7 * \log_2(10^6 * 10) \rightarrow$

$x = n * 10^7 * (\log_2 10^6 + \log_2 10)$

$x = 10h + 1,666h = 11,7h$

- **If your training set contains 100,000 instances, will setting presort=True speed up training?**

No, because it will take a long time to sort, sorting will only be useful to small datasets (with only a few thousand or less instances)

- **Train and fine-tune a Decision Tree for the moons dataset by following these steps:a. Use make_moons(n_samples=10000, noise=0.4) to generate a moons dataset.**
  **b. Use train_test_split() to split the dataset into a training set and a test set.**
  **c. Use grid search with cross-validation (with the help of the GridSearchCV class) to find good hyperparameter values for a DecisionTreeClassifier . Hint: try various values for max_leaf_nodes .**
  **d. Train it on the full training set using these hyperparameters, and measure your model's performance on the test set. You should get roughly 85% to 87% accuracy.**

- **Grow a forest by following these steps:**
  **a. Continuing the previous exercise, generate 1,000 subsets of the training**

**set,**
**each containing 100 instances selected randomly. Hint: you can use Scikit-Learn's ShuffleSplit class for this.**
**b. Train one Decision Tree on each subset, using the best hyperparameter values found in the previous exercise. Evaluate these 1,000 Decision Trees on the test set. Since they were trained on smaller sets, these Decision Trees will likely perform worse than the first Decision Tree, achieving only about 80% accuracy.**
**c. Now comes the magic. For each test set instance, generate the predictions**
**of the 1,000 Decision Trees, and keep only the most frequent prediction (you can use SciPy's mode() function for this). This approach gives you majority-vote predictions over the test set.**
**d. Evaluate these predictions on the test set: you should obtain a slightly higher accuracy than your first model (about 0.5 to 1.5% higher). Congratulations, you have trained a Random Forest classifier!**