Drzewa Decyzyjne

Igor Wojnicki

April 4, 2024

Plan prezentacji

Drzewa Decyzyjne

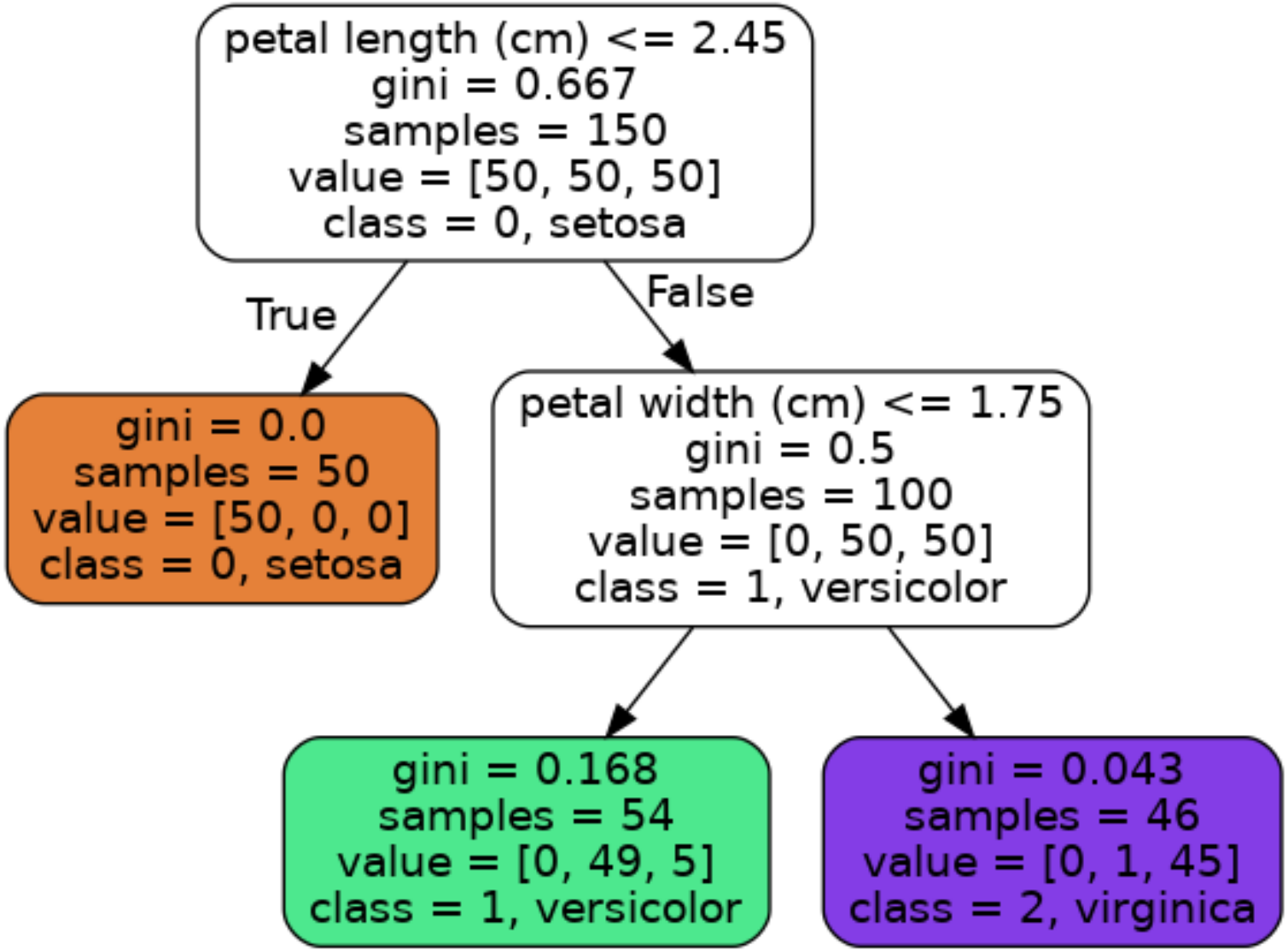
Drzewa Decyzyjne, uczenie

- klasyfikacja i regresja
- Nie wymagają skalowania cech!

```
1  from sklearn.datasets import load_iris
2  from sklearn.tree import DecisionTreeClassifier
3  iris = load_iris()
4  X = iris.data[:, 2:] # długość i szerokość płatka
5  y = iris.target
6  print(iris.target_names,"\n",y)
7  tree_clf = DecisionTreeClassifier(max_depth=2,
8                                    random_state=42)
9  tree_clf.fit(X, y)
```

```
['setosa' 'versicolor' 'virginica']
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Rezultat procesu uczenia



Czy jest to model parametryczny?

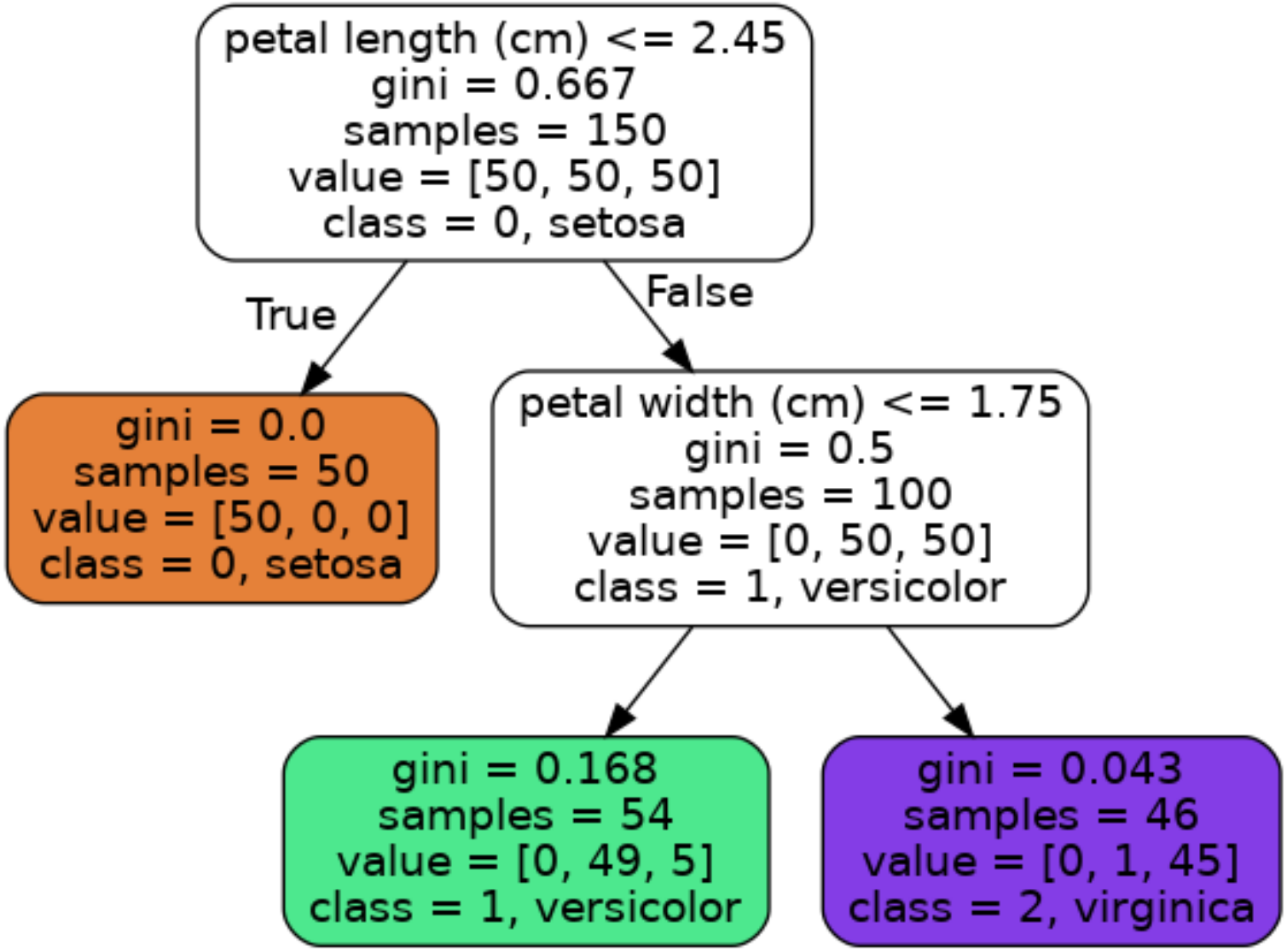
Co to jest drzewo decyzyjne?

```
1  from sklearn.tree import export_graphviz
2  f = "iris_tree.dot"
3  export_graphviz(
4          tree_clf,
5          out_file=f
6          feature_names=iris.feature_names[2:],
7          class_names=[str(num)+", "+name
8                       for num,name in
9                       zip(set(iris.target),
10                          iris.target_names)],
11         rounded=True,
12         filled=True
13     )
14 print(f)
```

```
iris_tree.dot
```

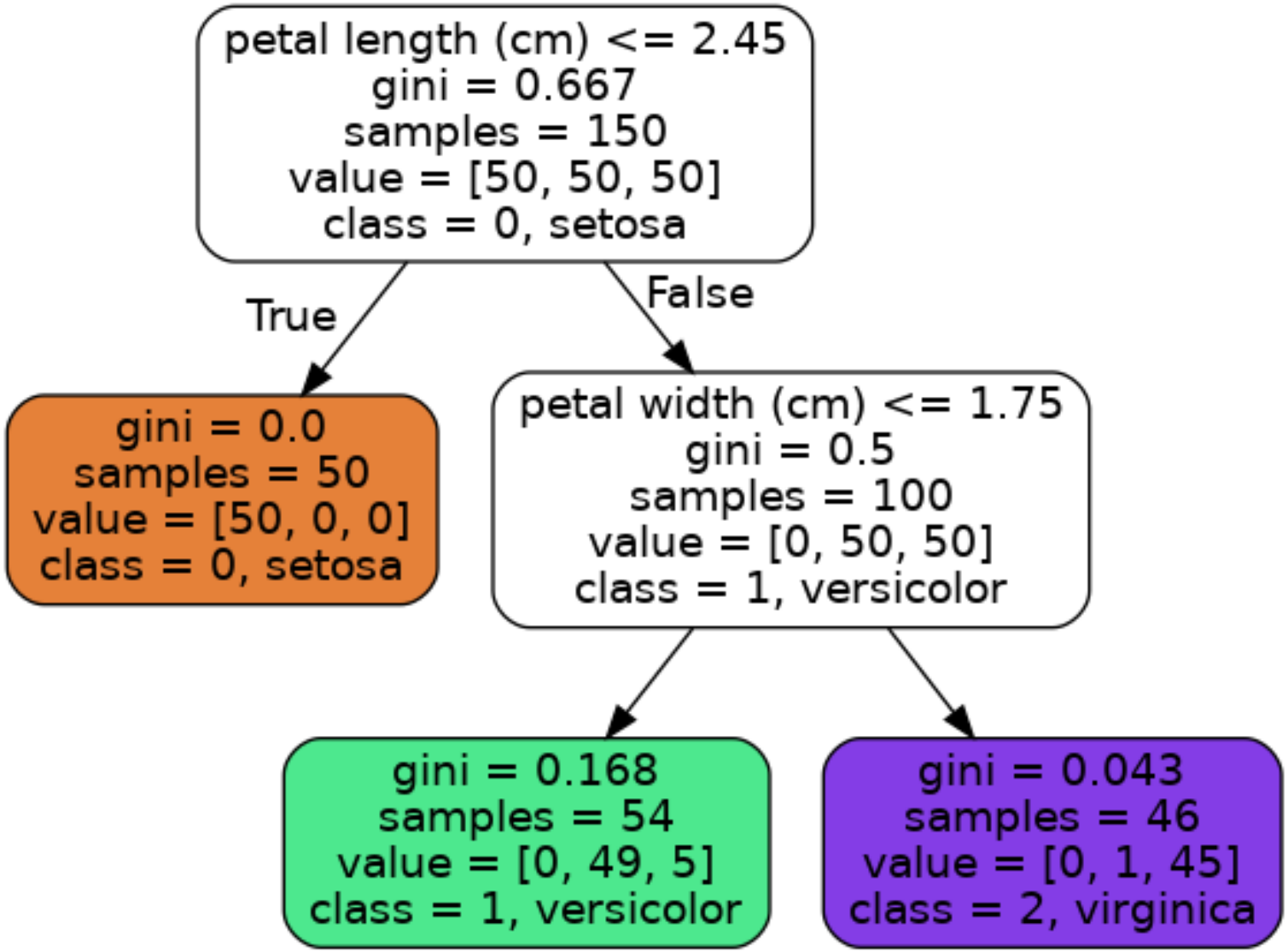
Pokaż drzewo!

```
dot -Tpng iris_tree.dot -o iris_tree.png
```



Pokaż drzewo, Python!

```
1  import graphviz
2  print(graphviz.render('dot', 'png', f))
```



Pokaż drzewo, Jupyter!

```
import graphviz
graph = graphviz.Source.from_file(f)
graph
```

```
str_dot = export_graphviz(
    tree_clf,
    out_file=None,
    feature_names=iris.feature_names[2:],
    class_names=[str(num)+", "+name
                 for num,name in
                 zip(set(iris.target),
                     iris.target_names)],
    rounded=True,
    filled=True)

graph = graphviz.Source(str_dot)
graph
```

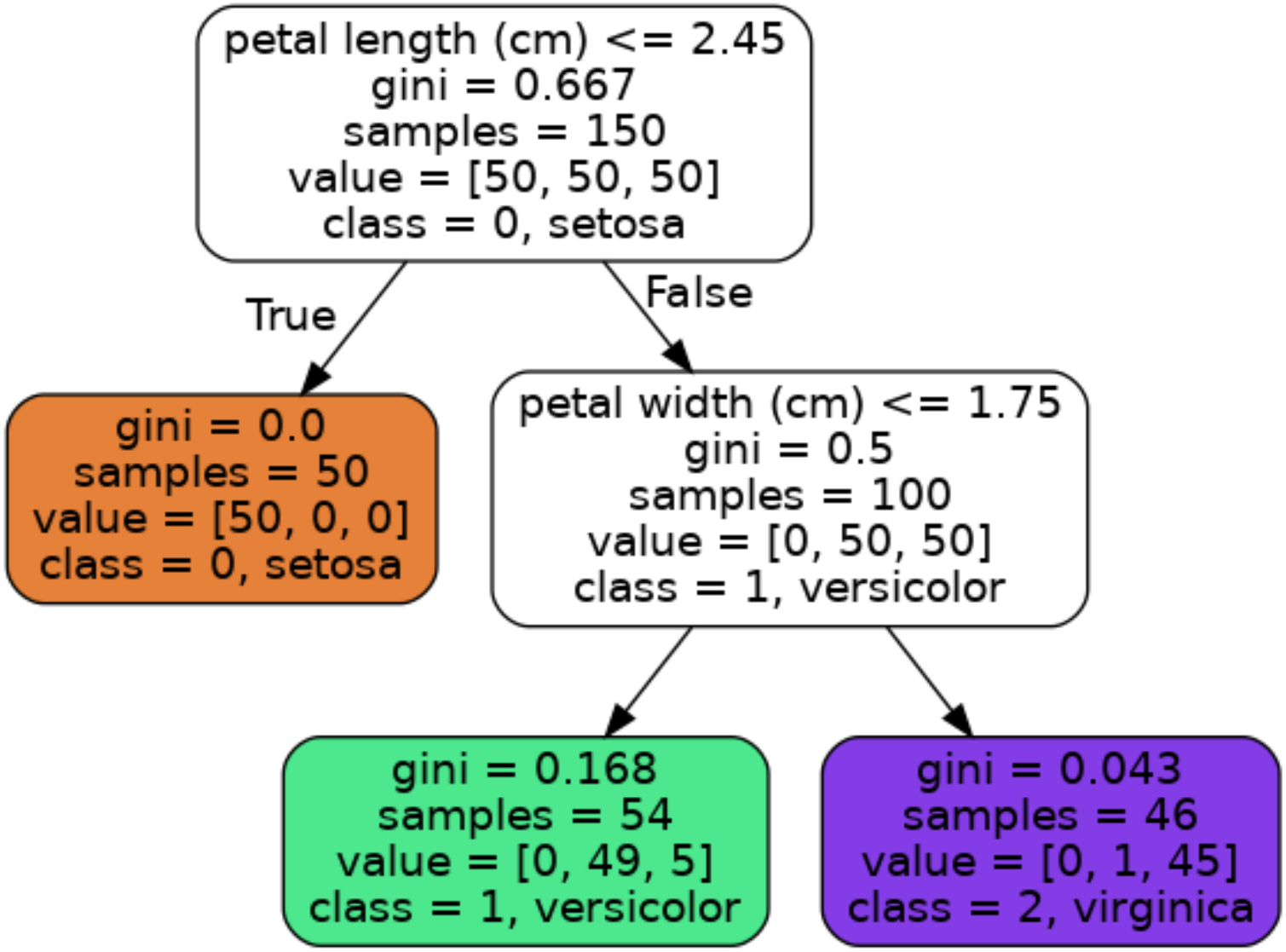
Drzewa decyzyjne, predykcja

```
1 print(tree_clf.classes_)
2 print(tree_clf.predict([[5.4,1.7]]))
3 print(tree_clf.predict_proba([[5.4,1.7]]))
```

```
[0 1 2]
[1]
[[0.         0.90740741 0.09259259]]
```

Prawdopodobieństwo: 49/54=0.907407407407

Reszta informacji



- *samples* – ile instancji ze zbioru uczącego
- *value* – ile instancji poszczególnych klas

Zanieczyszczenie Giniego

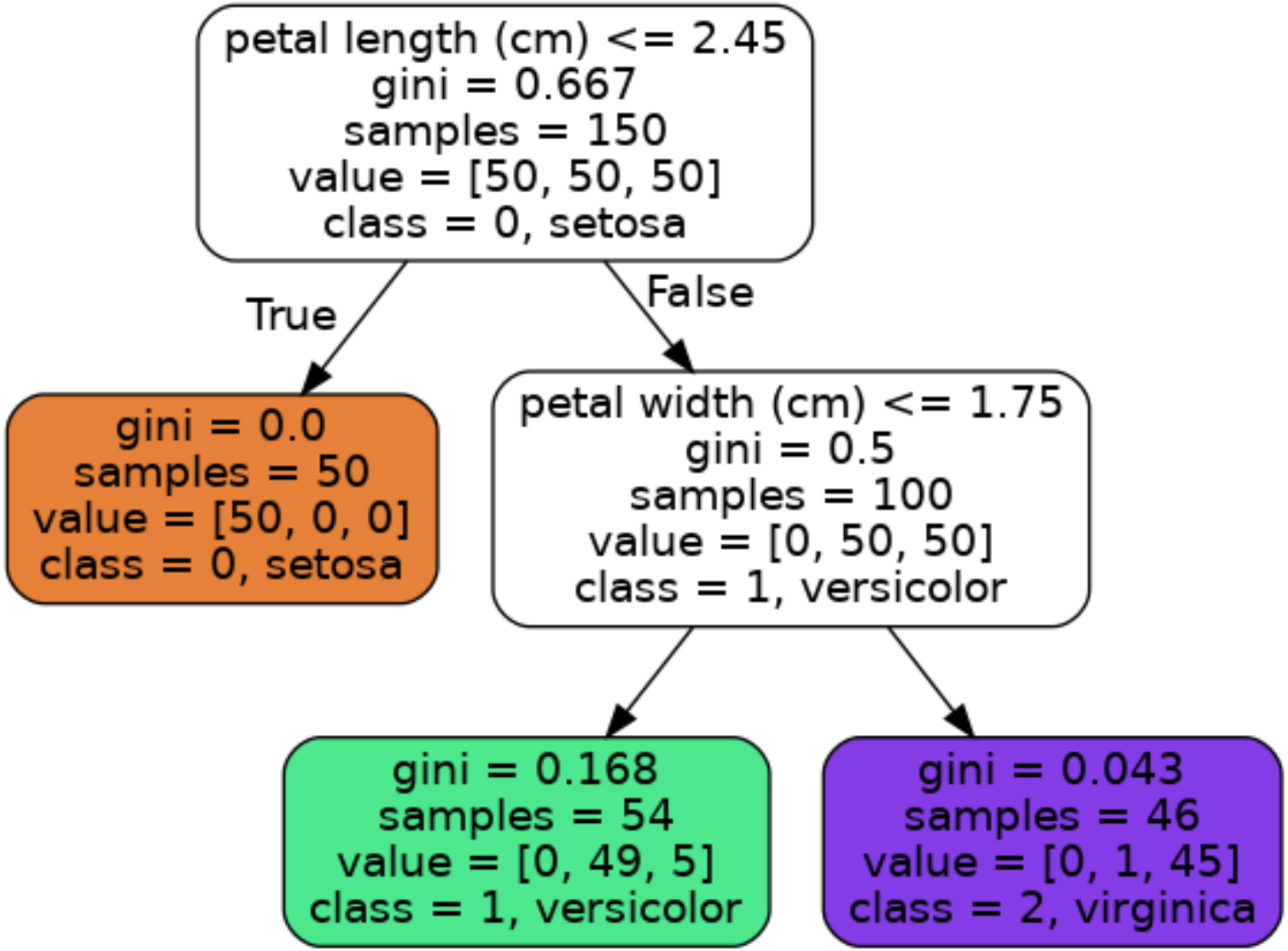
- *gini* – zanieczyszczenie, ile instancji należy do klasy (=0 wszystkie)

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

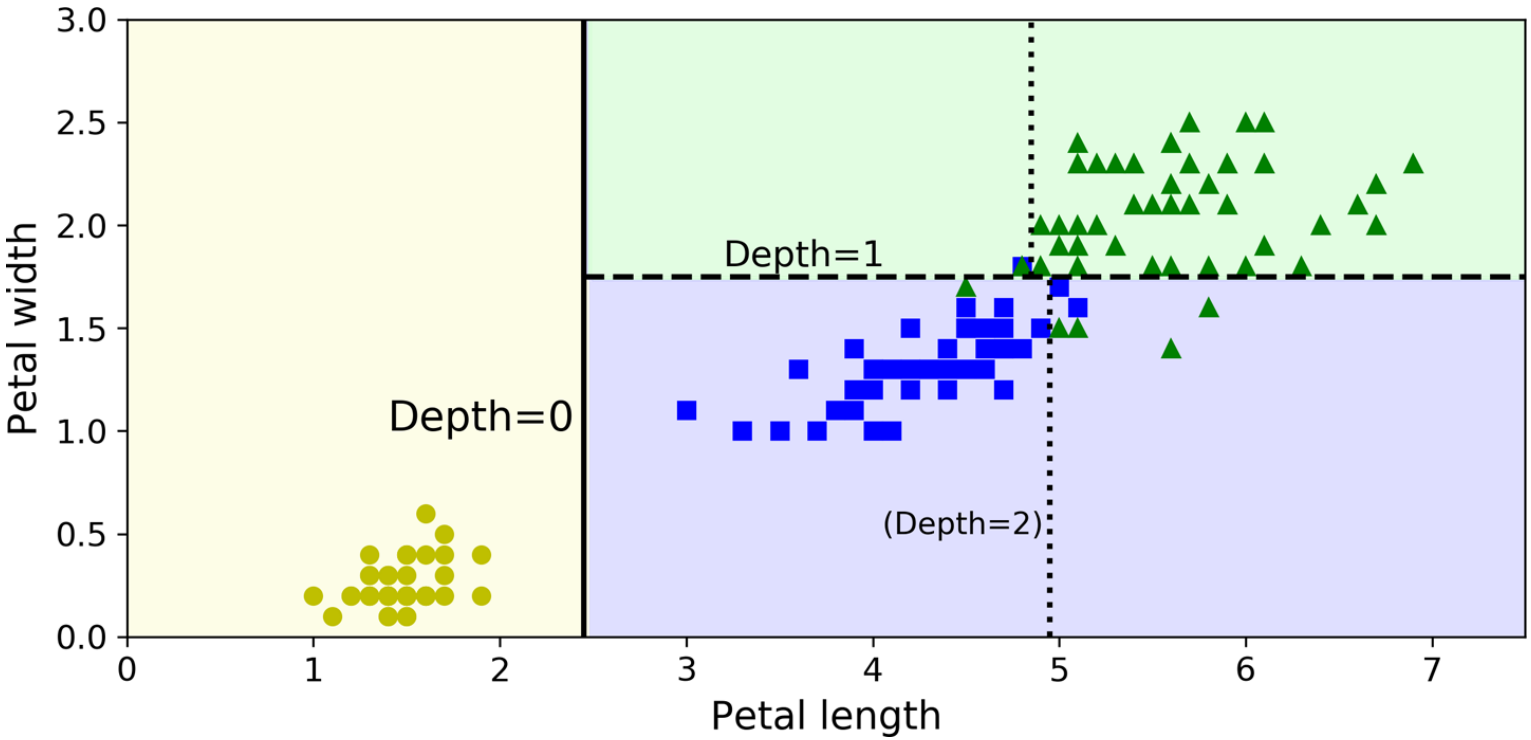
- $p_{i,k}$ iloraz liczby instancji klasy k w stosunku do liczby instancji w węźle i

Zanieczyszczenie

$$1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168$$
$$1 - (0/46)^2 - (1/46)^2 - (45/46)^2 \approx 0.043$$



Granice podziału drzewa



Drzewa decyzyjne, regresja, dane

```
1  import numpy as np
2  np.random.seed(42)
3  m = 200
4  X = np.random.rand(m, 1)
5  y = 4 * (X - 0.5) ** 2
6  y = y + np.random.randn(m, 1) / 10
```

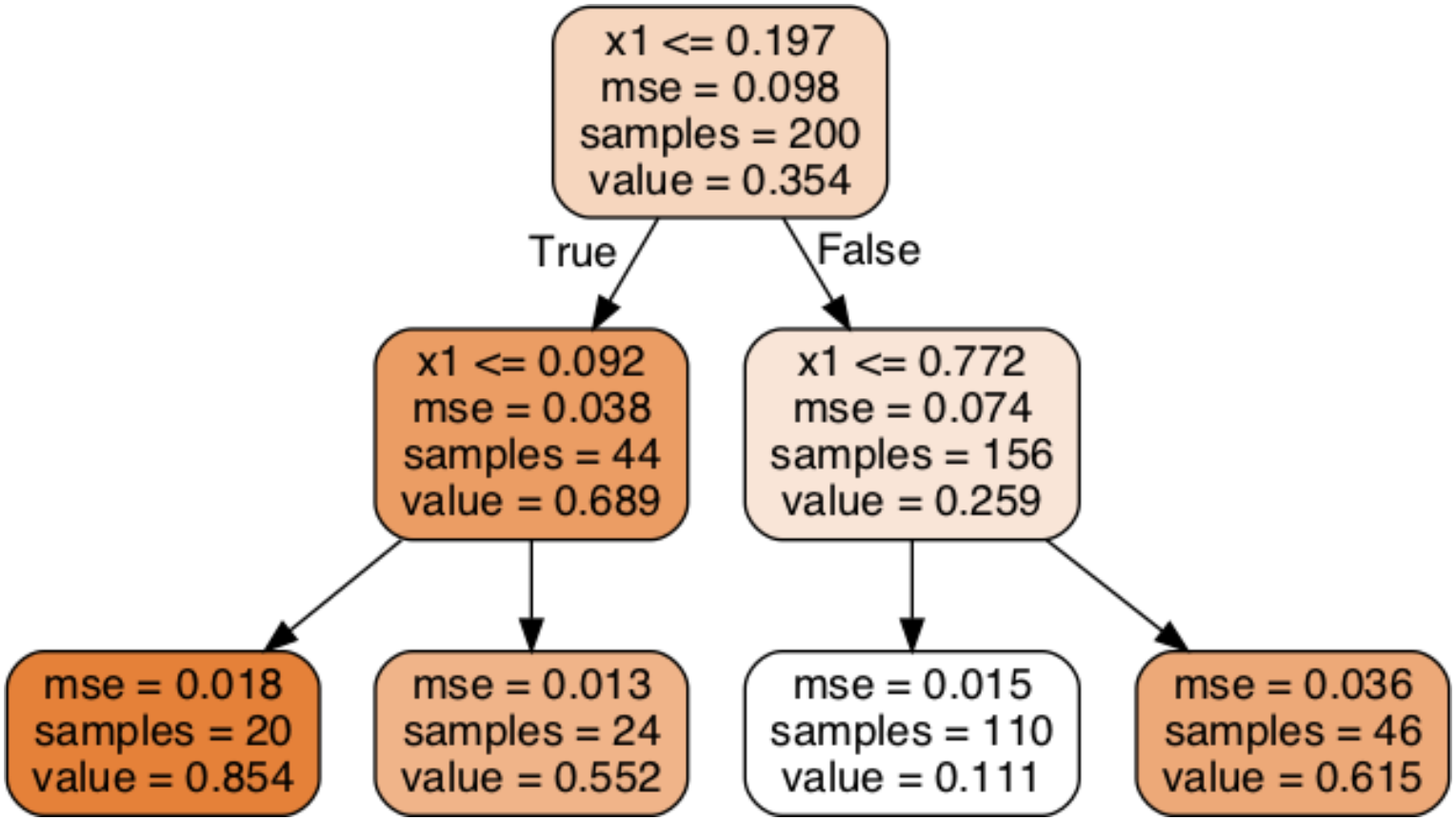
Drzewa decyzyjne, regresja, uczenie i predykcja

```
1  from sklearn.tree import DecisionTreeRegressor
2
3  tree_reg = DecisionTreeRegressor(max_depth=2,
4                                   random_state=42)
5  tree_reg.fit(X, y)
```

Drzewa decyzyjne, regresja, predykcja

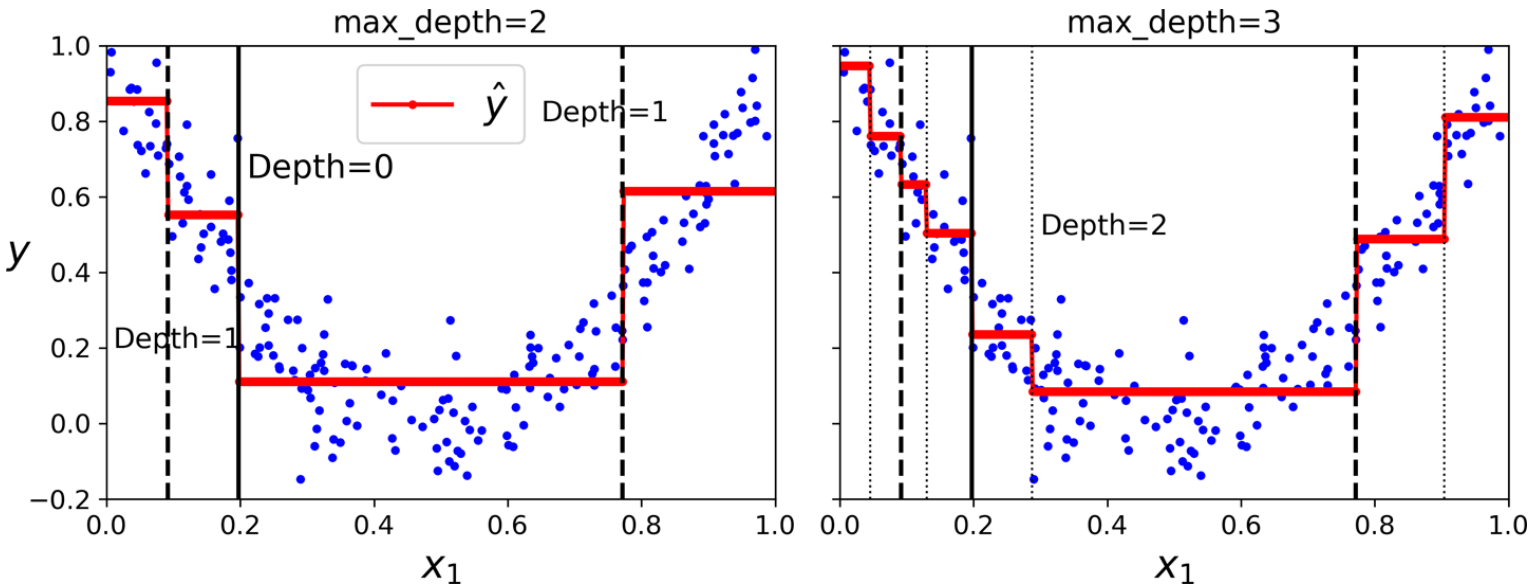
```
1  print(tree_reg.predict([[0.6]]))
```

```
[0.11063973]
```

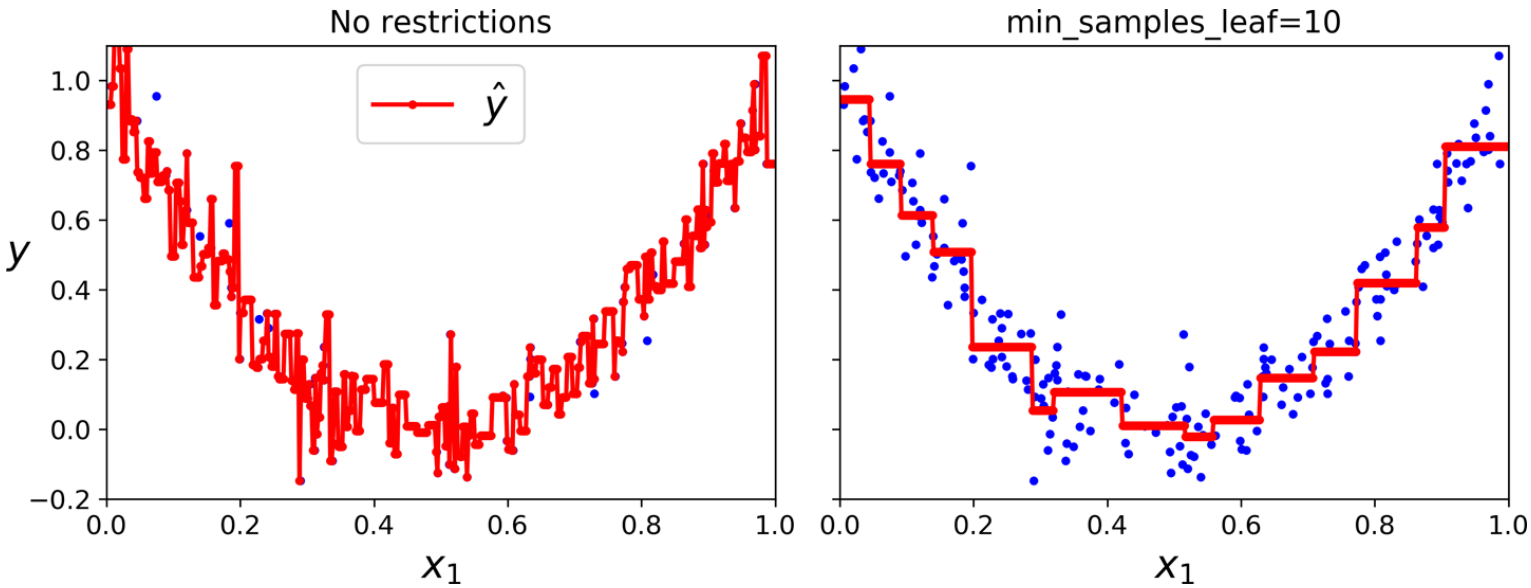


wartość = średnia z wszystkich instancji w węźle

Drzewa decyzyjne, regresja, drzewo, głębokość?



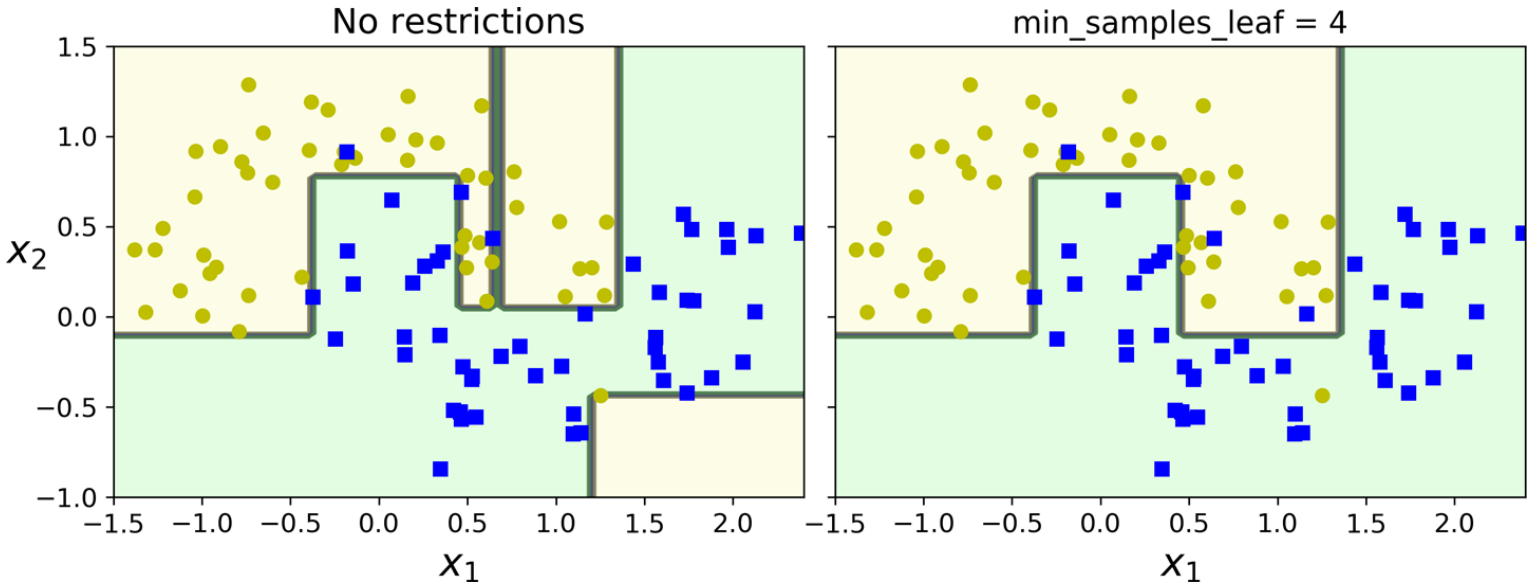
Drzewa decyzyjne, regresja, hiperparametr



Drzewa decyzyjne, inne hiperparametry

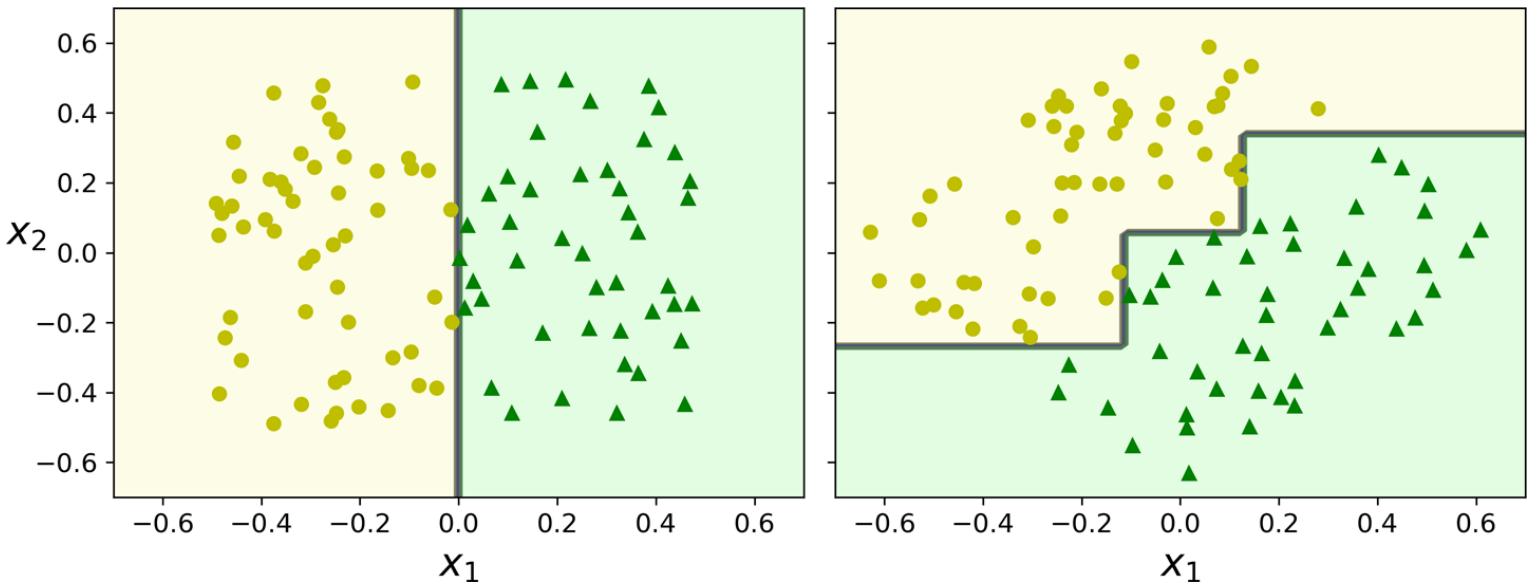
- Drzewa decyzyjne nie zakładają rodzaju modelu (np. że jest liniowy).
- Skłonność do overfitting.
- Jest to model *nieparametryczny* tzn. ilość parametrów nie jest znana przed rozpoczęciem proceu uczenia; w przeciwieństwie do modelu np. liniowego, gdzie ilość parametrów jest znana (współczynniki równania liniowego).
- *max_depth* – im mniejsza tym bardziej regularny model, mniejsze ryzyko overfitting.
- *min_samples_split* – minimalna liczba instancji, przy których można podzielić węzeł.
- *min_samples_leaf* – minimalna liczba instancji dla liścia.
- *min_weight_fraction_leaf*, *max_leaf_nodes*, *max_features*.
- Zwiększanie *min_ **, zmniejszanie *max_ ** -> regularyzacja.

Drzewa decyzyjne, przykładowa regularyzacja



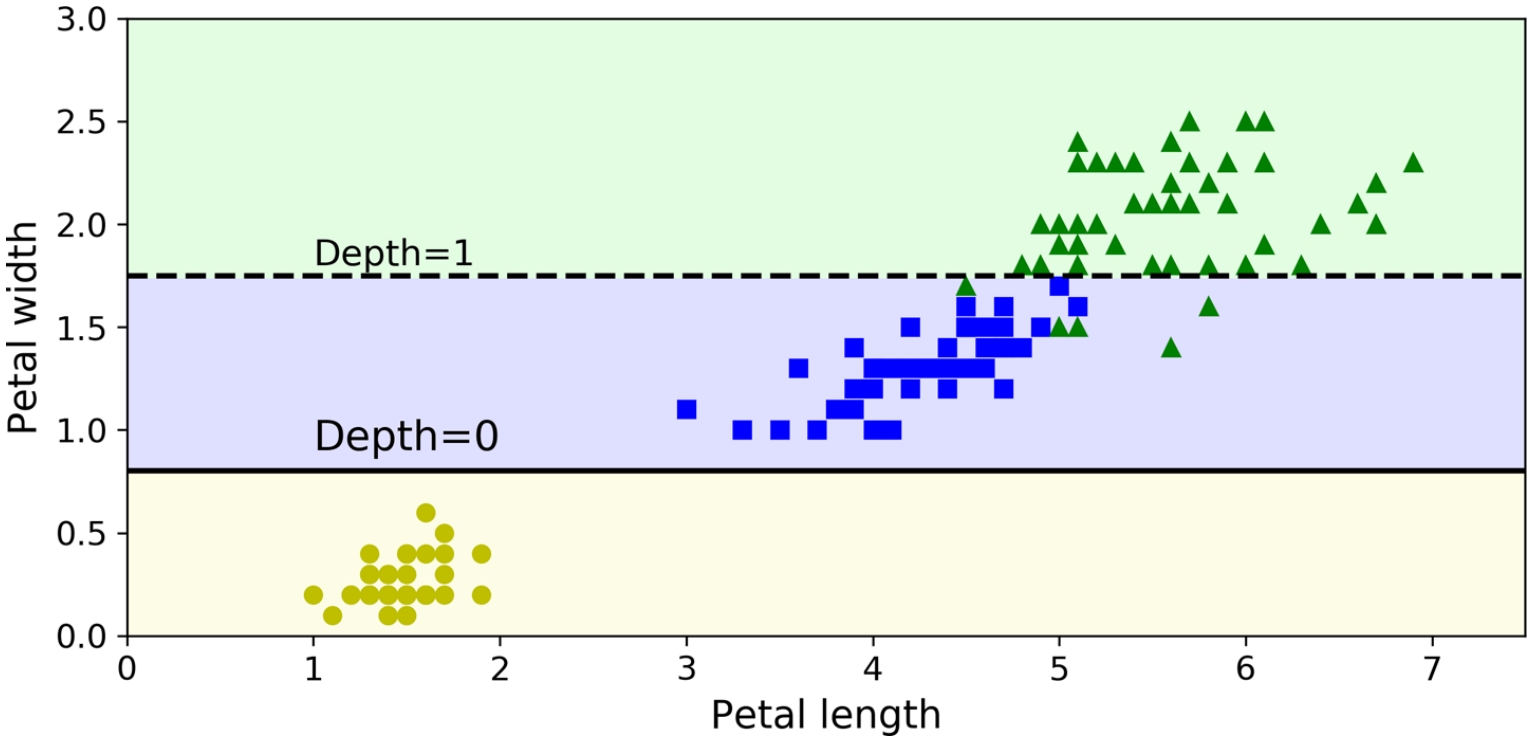
Drzewa decyzyjne, niestabilność

"Obrót" zbioru uczącego?



Drzewa decyzyjne, niestabilność

Czułość na zmiany w zbiorze uczącym: np. usunięcie *Iris versicolor* dł. 4.8, szer. 1.8.



Algorytm CART

- Classification and Regression Tree.
- Tworzy drzewo binarne.
- Algortym zachłanny.
- Rozwiązanie nie jest optymalne, ale wystarczająco dobre.

Algorytm CART, szczegóły

1. Podziel zbiór uczący na dwa, ze względu na pojedyczą cechę k i próg t_k,
   - tj. znajdź taką parę k i t_k, dla której w/w zbiory będą najczystsze (normalizowane licznością zbioru).
2. Wykonaj rekurencyjnie dalsze podziały na uzyskanych zbiorach, aż do osiągnięcia maksymalnej głębokości lub jeżeli nie będzie można znaleźć najczystszych zbiorów.
   Minimalizowana funkcja celu (kosztu):

$$J(k, t_k) = \frac{m_{lewa}}{m} G_{lewa} + \frac{m_{prawa}}{m} G_{prawa}$$

gdzie:

- $G_{lewa/prawa}$ – nieczystość lewego/prawego zbioru,
- $m_{lewa/prawa}$ – liczba instancji lewego/prawego zbioru,
- m – liczba wszystkich instancji.

Algorytm CART, złożoność

- uczenia: $O(n * m * log_2(m))$, n – ilość cech, m – ilość instancji
- predykcji: $O(log_2(m))$, m – ilość węzłów drzewa.

Inne algorytmy

- ID3: Iterative Dichotomizer 3.
  - Algorytm zachłanny.
  - Rozwiązanie nie jest optymalne, ale wystarczająco dobre.
  - Podział bazujący na entropii (niepewności związanej ze zmienną losową).
  - klasa `DecisionTreeClassifier`, parametr: `criterion='entropy'`
- C4.5
  - Podział bazujący na zysku informacyjnym (Information Gain).
  - Ulepszona wersja ID3.

Regresja, drzewa vs. knn, przykład I

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  np.random.seed(42)
5  m = 10
6  X = np.random.rand(m, 1)
7  y = 4 * (X - 0.5) ** 2
8  y = y + np.random.randn(m, 1) / 10
9
10 from sklearn.tree import DecisionTreeRegressor
11 tree_reg = DecisionTreeRegressor()
12 tree_reg.fit(X, y)
13
14 import sklearn.neighbors
15 knn_reg = sklearn.neighbors.KNeighborsRegressor()
16 knn_reg.fit(X, y)
```

Regresja, drzewa vs. knn, przykład II

```
17
18  plt.clf()
19  plt.scatter(X, y, c="blue")
20  X_new1 = np.arange(0, 1, 0.001).reshape(-1,1)
21  plt.plot(X_new1, tree_reg.predict(X_new1), c="red")
22  plt.plot(X_new1, knn_reg.predict(X_new1), c="green")
23  f = "drzewa-knn.png"
24  plt.savefig(f)
25  print(f)
```

Regresja, drzewa vs. knn, przykład III