به نام خدا
خلاصه فصل ۶ کتاب
HANDS-ON MACHINE LEARNING

DECISION TREES

- داخل این فصل با decision trees آشنا میشیم که توانایی اینو دارن هم classification هم regression رو انجام بدن. توانایی خیلی خوبی برای وفق دادن خودشون با ساختار دیتاهای پیچیده دارن. همچنین پایه مفهوم random forests هستن که داخل فصل بعدی میخونیم.

- اول بیایم همون دیتاست iris رو برداریم و یه مدل decision tree رو روش train کنیم.

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris(as_frame=True)
X_iris = iris.data[["petal length (cm)", "petal width (cm)"]].values
y_iris = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X_iris, y_iris)
```

CONTINUE

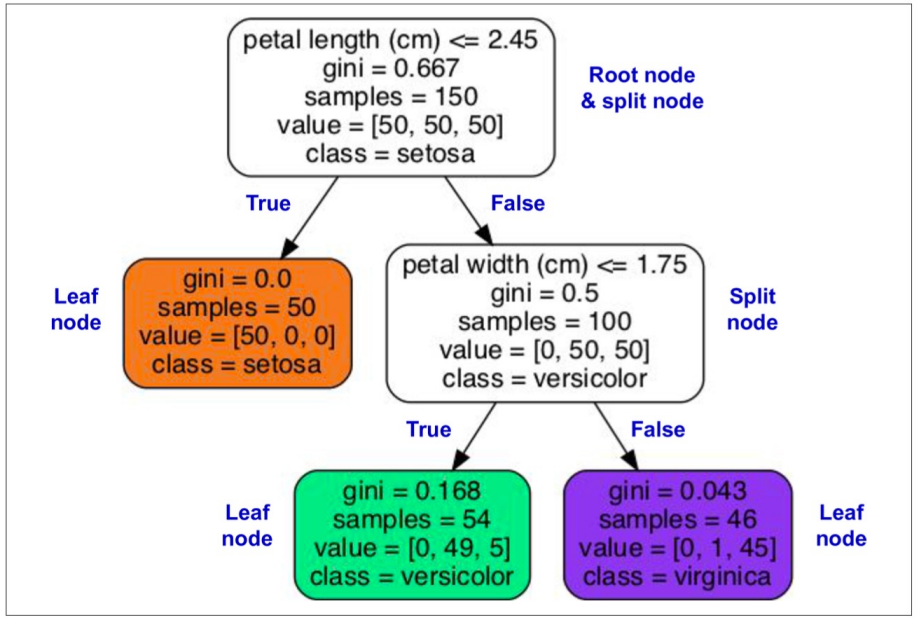
- میتونیم مدلی که train شده رو visualize کنیم با استفاده از export_graphviz که میاد به عنوان خروجی یه گراف بهمون میده.

```
from sklearn.tree import export_graphviz

export_graphviz(
        tree_clf,
        out_file="iris_tree.dot",
        feature_names=["petal length (cm)", "petal width (cm)"],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )
```

- نحوه فراخونی گراف تشکیل شده:

```
from graphviz import Source

Source.from_file("iris_tree.dot")
```

- شکل درختمون این شکلی میشه:



MAKE PREDICTIONS

- اما بیایم ببینیم با این درخت چه شکلی میشه پیش‌بینی کرد. فرض کنید که میخوایم یه گل رو برحسب ویژگی petalsش برسی کنیم. از ریشه درخت شروع میکنیم و داخل این نود میپرسه که ایا طول petal کوچیکتر از ۲.۴۵ سانتی متر هست یا نه اگه بود، میریم به بچه سمت چپش، این نود که برگ درخت هستش دیگه سوالی نمیپرسه و به کلاس متعلق به این نود یعنی setosa میرسیم و این شکلی پیش بینی میکنه.
- یه خوبی decision trees اینه که نیاز به feature scaling ندارن. پس نیاز به استاندارد کردن متغیر ها نداریم.
- یه attribute داخل نود ها بود به اسم samples که نشون میداد چند تا نمونه تا اونجا اومده بودن برای تصمیم گیری.
- یه attribute دیگه value هستش که نشون میده از هر کلاس چند تا نمونه رسیده اینجا.

CONTINUE

- در نهایت یه attribute دیگه داخل نود ها هست به اسم gini. اگه یه نود خالص باشه gini مساوی صفر داره که یعنی همه نمونه هایی که رسیدن به اون نود متعلق به یک کلاس خاص هستن. برای نود سبز رنگ داخل درخت قبلی که نشون دادیم این شکلی حساب میشه:

Equation 6-1. Gini impurity

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

In this equation:

- G_i is the Gini impurity of the i^{th} node.
- $p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node.

CONTINUE

- اگه بخوایم نشون بدیم چطوری تصمیم گیری میشه برای predict کردن شکل زیر هم خیلی خوبه. یه خط عمودی داره که مرزبندی میکنه برای نود ریشه و برای depth = 1 که دو تا بچه نود ریشه هستن خط افقی کشیده و بر مبنا اون کلاس بندی داره میکنه. اگه max depth رو بیشتر میذاشتیم اون دو تا خط نقطه چین رو میذاشت.

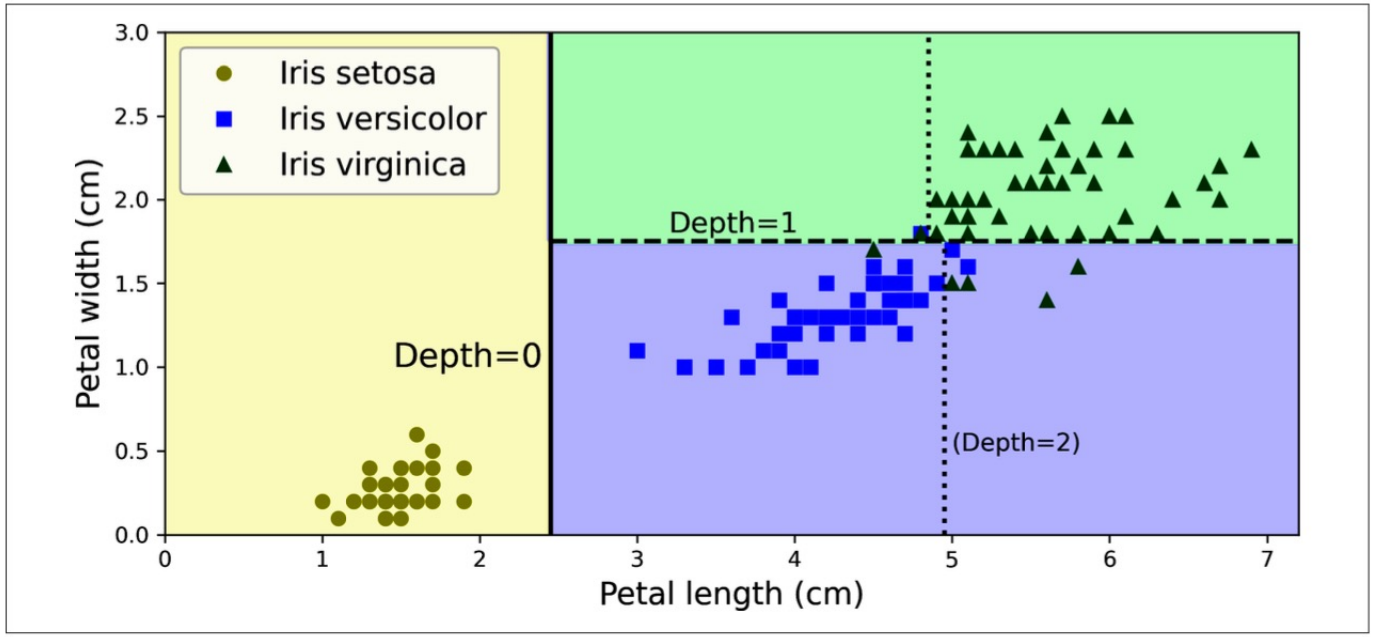


Figure 6-2. Decision tree decision boundaries

ESTIMATING CLASS PROBABILITES

- کاری که انجام میده این درخت وقتی یه نمونه جدید میاد اینه میره تا جایی که به یه نود برگ برسه که متعلق به اون کلاسه و برای اینکه درصد احتمالی بده بهمون که چند درصد احتمال داره متعلق به این کلاس هستش رو حساب میکنه و چون بیشترین درصد رو داره میگه که به اون کلاس تعلق داره.

suppose you have found a flower whose petals are 5 cm long and 1.5 cm wide. The corresponding leaf node is the depth-2 left node, so the decision tree outputs the following probabilities: 0% for *Iris setosa* (0/54), 90.7% for *Iris versicolor* (49/54), and 9.3% for *Iris virginica* (5/54). And if you ask it to predict the class, it outputs *Iris versicolor* (class 1) because it has the highest probability. Let's check this:

```
>>> tree_clf.predict_proba([[5, 1.5]]).round(3)
array([[0.   , 0.907, 0.093]])
>>> tree_clf.predict([[5, 1.5]])
array([1])
```

THE CART TRAINING ALGORITHM

- داخل scikit-learn میاد از الگوریتم classification and regression tree (cart) استفاده میکنه تا decision tree رو train میکنه. این الگوریتم میاد اول training set رو به دو بخش جدا میکنه. این جدا سازی بر اساس یه feature مثل k و یه Tk treshold هست مثل اینکه بگیم خط pental length = 2.45cm یه نمونه میشه برای treshold. اما چطوری این دو تا مولفه رو پیدا میکنه؟ در اصل دنبال این دو مورده به شرطی که خالص ترین زیر مجموعه هارو برامون بسازه. که البته وزن این دو تا زیر مجموعه هم مهمه که زیاد باشه. با همه تفاسیر بالا میشه یه همچین cost functionی تعریف کرد:

Equation 6-2. CART cost function for classification

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

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

CONTINUE

- وقتی که یه مجموعه رو بتونه با موفقیت به دو تا زیرمجموعه جدا کنه اون موقع میتونه همین کار رو به طور بازگشتی برای زیر مجموعه های پایینی انجام بده تا برسه به maximum depth. همینجور که میبینیم این الگوریتم یه الگوریتم greedy هستش و به این معنا هستش که چک نمیکنه ایا یه جداسازی تو این منطقه باعث میشه که بهترین جداسازی هم تو عمق پایین تر رخ بده.
- برای اینکه بهترین جداسازی که منجر به نتایج بهتر هم تو پایین بشه اون موقع مسئله اوردر بالاتری نسبت به حالت قبل پیدا میکنه.

Unfortunately, finding the optimal tree is known to be an *NP-complete* problem.[1] It requires $O(\exp(m))$ time, making the problem intractable even for small training sets. This is why we must settle for a "reasonably good" solution when training decision trees.

COMPUTATIONAL COMPLEXITY

- به طور کلی یک درخت باینری تو کانسپت ما بالانس هست پس صرفا برای اینکه بتونیم یه predict انجام بدیم نیازه که عمق درخت رو پیمایش کنیم تا به یه برگ برسیم. که همچین کاری از اوردر لگاریتم ۲ هستش.

requires going through roughly $O(\log_2(m))$ nodes, where $\log_2(m)$ is the *binary logarithm* of m, equal to $\log(m) / \log(2)$. Since each node only requires checking the value of one feature, the overall prediction complexity is $O(\log_2(m))$, independent of the number of features. So predictions are very fast, even when dealing with large training sets.

- همچنین اوردر training این شکلیه که همه فیچر ها روی هر نود برسی میشه تا بشه یه decision boundary رسم کنیم و همچنین همه سمپل ها برسی میشه.

The training algorithm compares all features (or less if `max_features` is set) on all samples at each node. Comparing all features on all samples at each node results in a training complexity of $O(n \times m \log_2(m))$.

GINI IMPURITY OR ENTROPY

- یه مقیاس دیگه به جز gini impurity میتونیم تعریف کنیم که این شکلی باشه بیایم از entropy استفاده کنیم. Entropy مساوی صفر میشه اگه داخل یه نود تمام شی های متعلق به یک کلاس باشن و به همچین شکلی حساب میشه.

is zero when it contains instances of only one class. Equation 6-3 shows the definition of the entropy of the i^{th} node. For example, the depth-2 left node in Figure 6-1 has an entropy equal to $-(49/54)\log_2(49/54) - (5/54)\log_2(5/54) \approx 0.445$.

Equation 6-3. Entropy

$$H_i = -\sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log_2(p_{i,k})$$

- تفاوت خیلی خاصی هم با هم ندارن صرفا gini یه ذره سریعتره اما اگه بخوایم خیلی درخت بالانسی داشته باشیم entropy استفاده میکنیم.

REGULARIZATION HYPERPARAMETERS

- اگه خودمون یه سری hyperparameter برای درختمون نذاریم به طور کلی خیلی آزاده و میتونه خودشو با training set های مختلف به طور خوبی adapt کنه که طبیعتا ریسک overfit زیاد میشه. برای اینکه ریسک overfit کم بشه نیاز داریم که یه سری آزادی های درخت رو زمان train کم کنیم. که به این کار regularization گفته میشه. یه مثال همون max_depthی بود که قبلا استفاده کردیم اینجا هم یه سری دیگه میگیم:

`max_features`
Maximum number of features that are evaluated for splitting at each node

`max_leaf_nodes`
Maximum number of leaf nodes

`min_samples_split`
Minimum number of samples a node must have before it can be split

`min_samples_leaf`
Minimum number of samples a leaf node must have to be created

`min_weight_fraction_leaf`
Same as `min_samples_leaf` but expressed as a fraction of the total number of weighted instances

CONTINUE

- یه مثال اگه بخوایم ببینیم این شکلی میتونیم اعمال کنیم محدودیت هارو و نتیجه کار هم ببینیم

```
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)
```

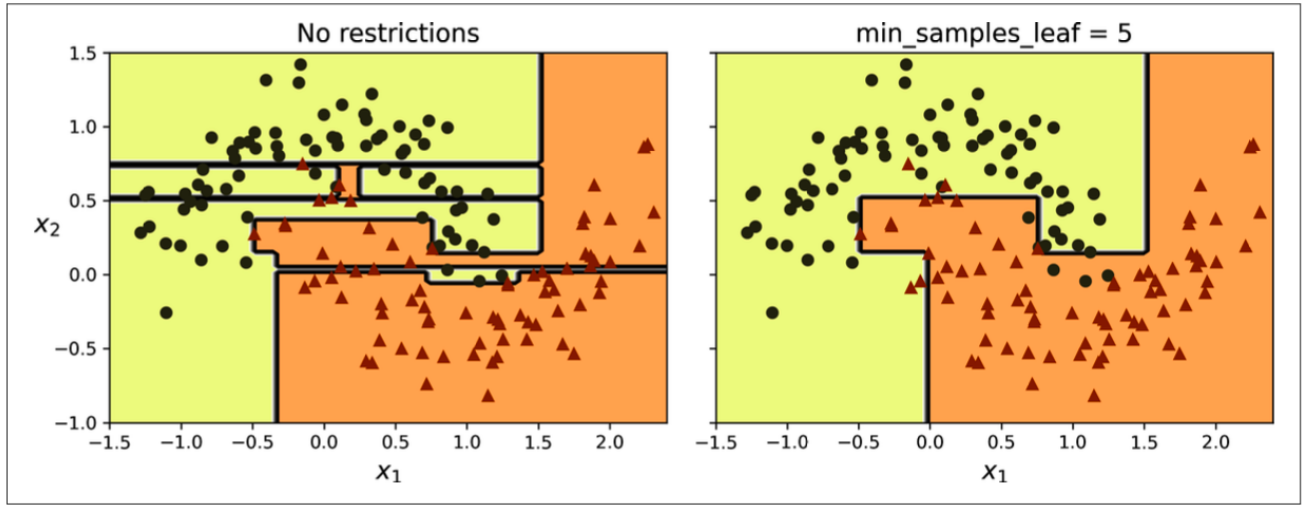


Figure 6-3. Decision boundaries of an unregularized tree (left) and a regularized tree (right)

REGRESSION

- Decision tree فقط برای تسک های classification استفاده نمیشن و توانایی اینو دارن برای regression هم استفاده بشن. بیایم یه درخت رو برای یه مدل درجه ۲ برسی کنیم.

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

np.random.seed(42)
X_quad = np.random.rand(200, 1) - 0.5  # a single random input feature
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)
```

The resulting tree is represented in Figure 6-4.

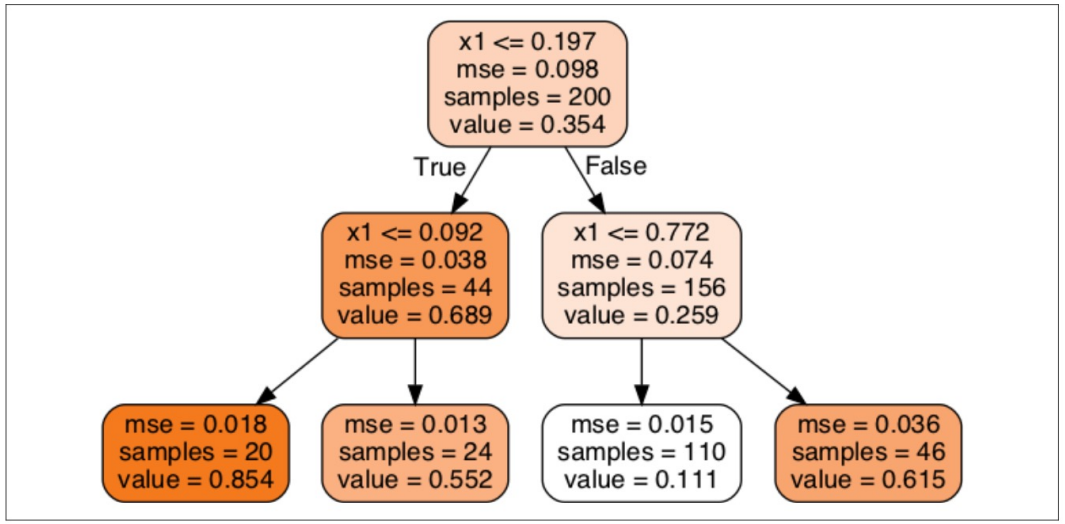


Figure 6-4. A decision tree for regression

CONTINUE

- روش پیش‌بینی اینم شبیه به classification عه. یعنی این شکلی که مثلا برای x=0.2 میاد میبینه که کوچیکتر از 0.197 هست یا نه میره راست تا برسه به نود سفید رنگ و مقدار 0.111 رو خروجی میده. این عدد هم در واقع میانگینی از نمونه هایی که متعلق به این نود بودن بدست اومده. اینم یه خروجی دیگه که چطوری درخت میاد پیش بینی انجام میده.

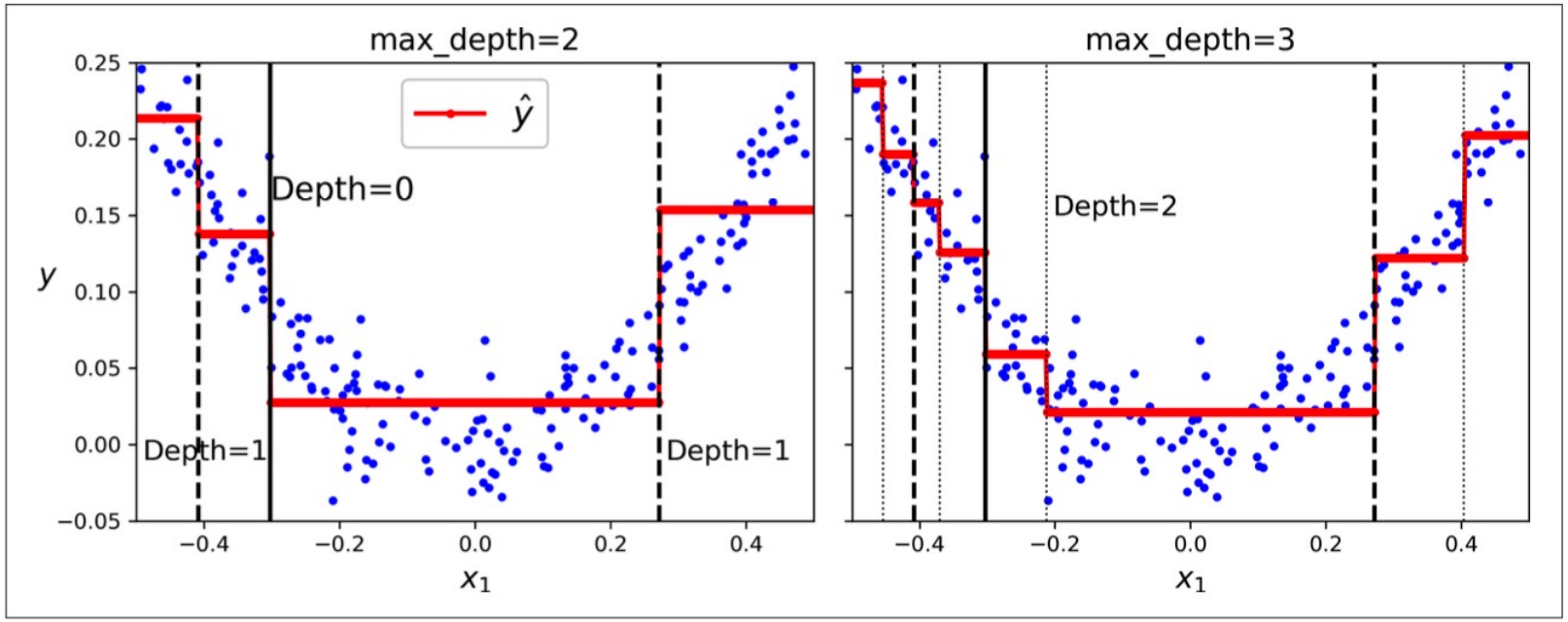


Figure 6-5. Predictions of two decision tree regression models

COST FUNCITON OF REGRESSION

- اینجا هم همون cart الگوریتم به کار برده میشه صرفا اینجا به جای اینکه بیایم دنبال minimize کردن impurity هستیم دنبال minimize کردن MSE هستیم.

Equation 6-4. CART cost function for regression

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \dfrac{\sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2}{m_{\text{node}}} \\ \hat{y}_{\text{node}} = \dfrac{\sum_{i \in \text{node}} y^{(i)}}{m_{\text{node}}} \end{cases}$$

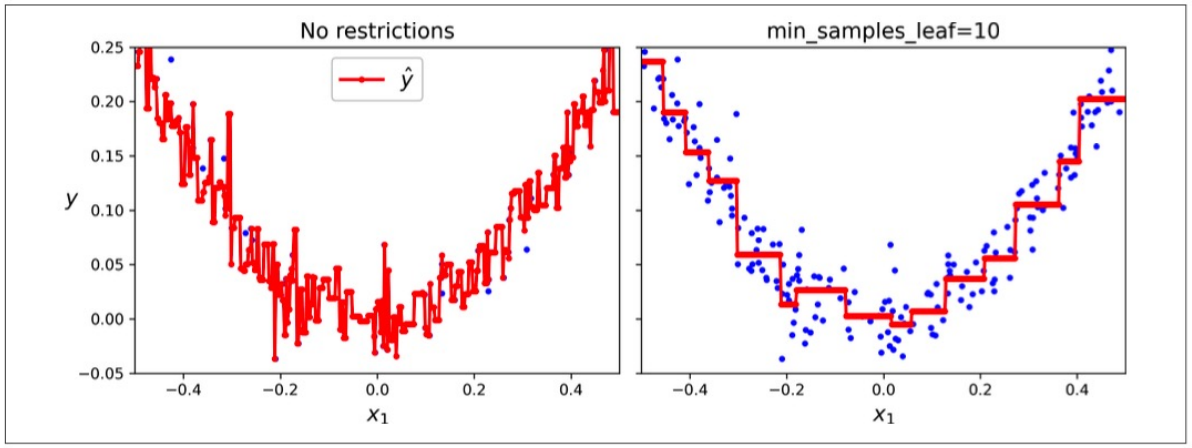


Figure 6-6. Predictions of an unregularized regression tree (left) and a regularized tree (right)

برای اینکه overfit نکنه دوباره باید یه سری regularization انجام بدیم که میتونیم از min_sample_leaf استفاده کنیم که این شکلی میشه:

SENSITIVITY TO AXIS ORIENTATION

- درختمون خیلی ارتباط با نحوه جایگیری دیتاها در راستای محور های مختصات داره. یعنی چی؟ تو شکل زیر اگه دیتاهای سمت چپ رو ۴۵ درجه rotation بدیم باید درخت سمت راست رو درست کنیم تا بتونه جداسازی رو انجام بده. چرا؟ چون decision boundaries عمودیه نیاز داره همچین خط هایی بکشه.

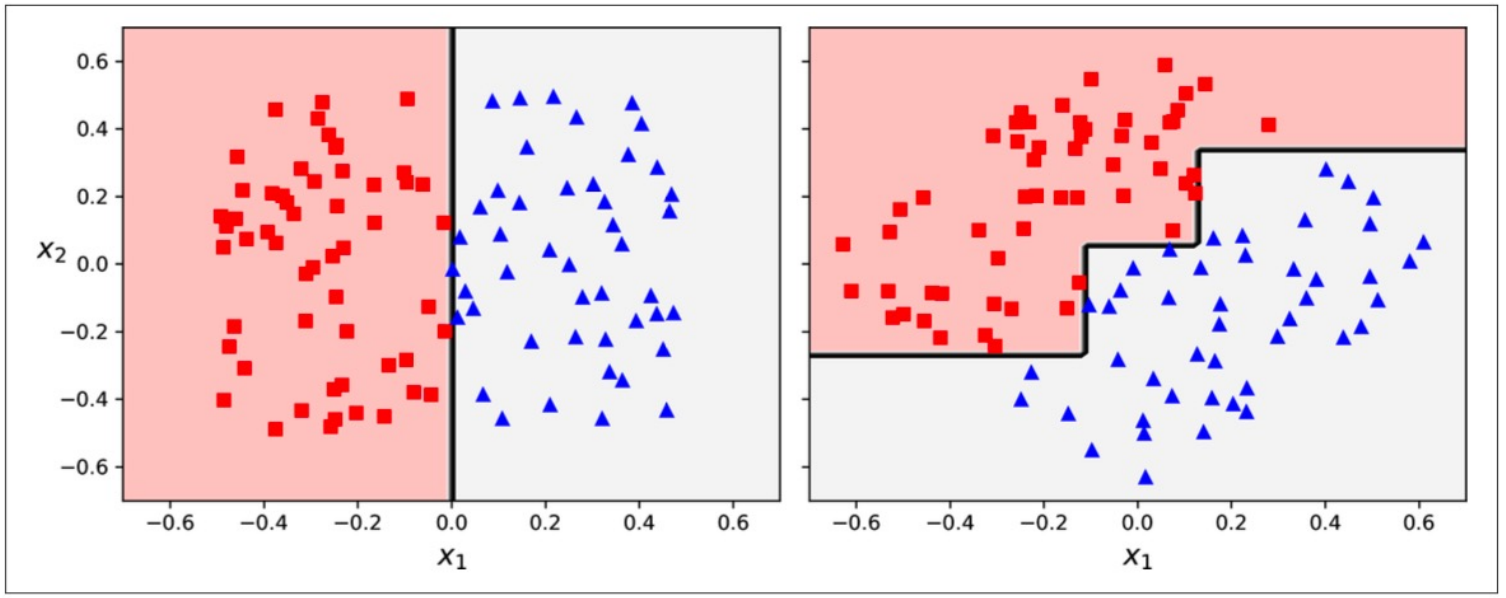


Figure 6-7. Sensitivity to training set rotation

CONTINUE

- صرفا باید دقت کرد میشه با استفاده از pca میتونیم دیتاهامون رو بچرخونیم روی محور ها به شکلی که correlation بین فیچرهامون کم بشه. که در اکثر مواقع این کار باعث میشه کار برای درختمون آسون تر بشه. بیایم ببینیم اگه همچین کاری کنیم چه شکلی میشه درختمون.

```
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)
```

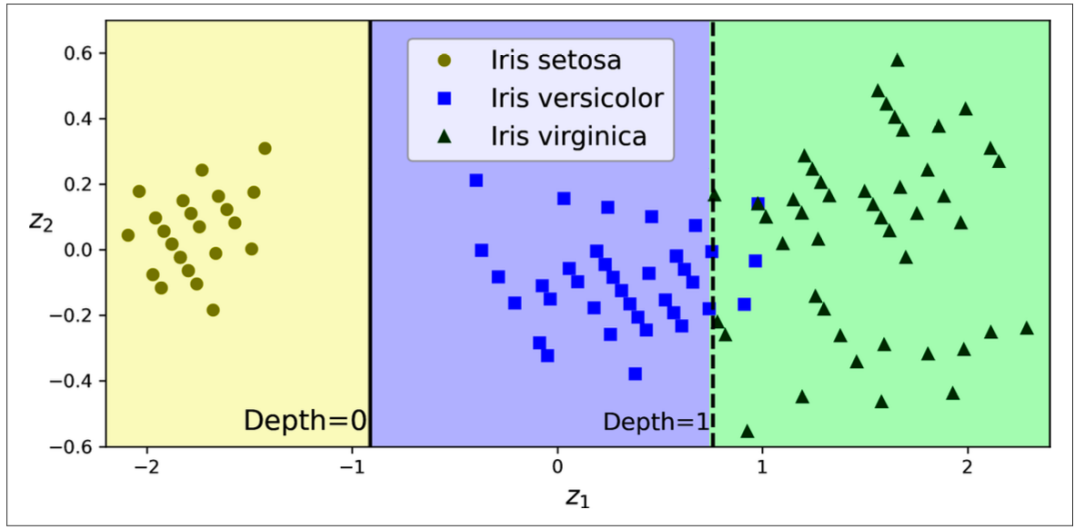


Figure 6-8. A tree's decision boundaries on the scaled and PCA-rotated iris dataset

- کاری که کرده اینه اومده یه متغیر z1 که ترکیب خطی از ۲ متغیر طول و عرض گلبرگ گل ها بوده ساخته با استفاده از PCA و طبق این متغیر خط میکشه.

DECISION TREES HAVE A HIGH VARIANCE

- در حقیقت یه مشکل اصلی که وجود داره این درخت ها high variance هستن. به این معنی که یه تغییر کوچیک داخل hyperparameter یا data باعث میشه مدل خیلی متفاوتی ساخته بشه. به این معنی که اگه روی یه دیتاست بیایم دوباره مدل رو train کنیم یه چیز دیگه نتیجه میگیریم. بخاطر اینکه کلا چون داخل فرایند یادگیری یه سری فیچر رندوم انتخاب میکنه ممکنه یه مدل دیگه بهمون برگردونه مگه اینکه بیایم مثلا random_state رو یه عدد فیکس بذاریم. اگه همچین کاری نکنیم رو مدل صفحه قبل همچین مدلی میگیریم.

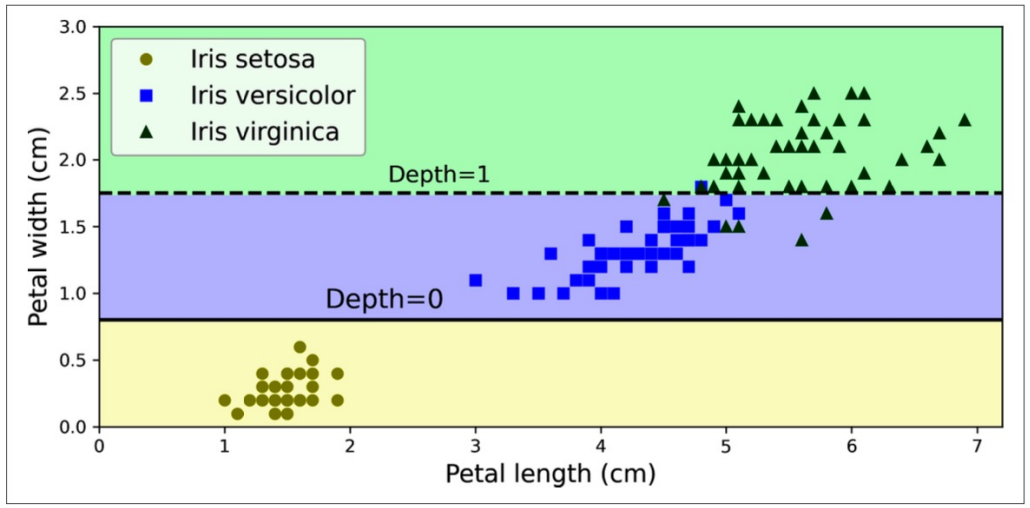


Figure 6-9. Retraining the same model on the same data may produce a very different model

- تو فصل بعد با استفاده از ensembling که در واقع استفاده از random forest هست این مشکل رو حل میکنیم که تو فصل بعد برسی میکنیم.