Decision Trees

Decision Trees are versatile ML algorithms, Can perform both classification and regression tasks, and even multioutput tasks.

Training and visualizing a Decision Tree

- Train a Decision tree on the iris dataset

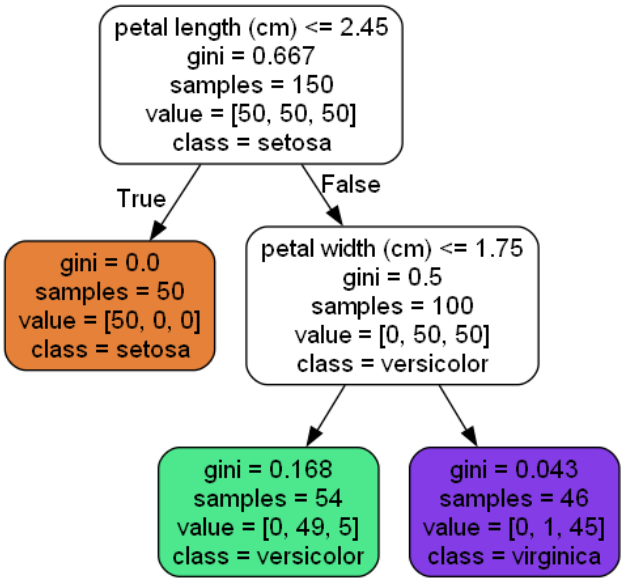
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

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)
```

- Visualizing the trained decision tree, using export_graphviz() which output iris_tree.dot

```
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
        )
```

- Then you can use graphviz.Source.from_file() to load and display the file in a Jupyter notebook:

```
from graphviz import Source

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



Making Predictions

- One of many qualities of Decision Trees that they require very little data preparation, they don't require feature scaling or centering at all
- In fig 6-1, you start at the *root node*(depth 0) asks petal length <= 2.45
- If it is yes, move down to the root's left child node(depth 1, left), Called a *Leaf node, doesn't ask any questions*
- The root's right child node(depth1, right), which is *not a leaf node, it asks a question*
- A node's *samples* attribute counts how many training instances it applies to
- A node's *value* attribute tells you how many training instances of each class this node applies to
- A node's *gini* attribute measures its impurity, a node is "pure" (gini = 0), if all training instances it applies to belong to the same class

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.

> **TIP**
>
> The tree structure, including all the information shown in Figure 6-1, is available via the classifier's `tree_` attribute. Type `help(tree_clf.tree_)` for details, and see the this chapter's notebook for an example.

- Scikit-learn uses *CART* algorithm, which produces only *binary trees*: non leaf nodes always have two children
- However. Other algorithms such as ID3 can produce Decision Trees with nides that have more than two children.

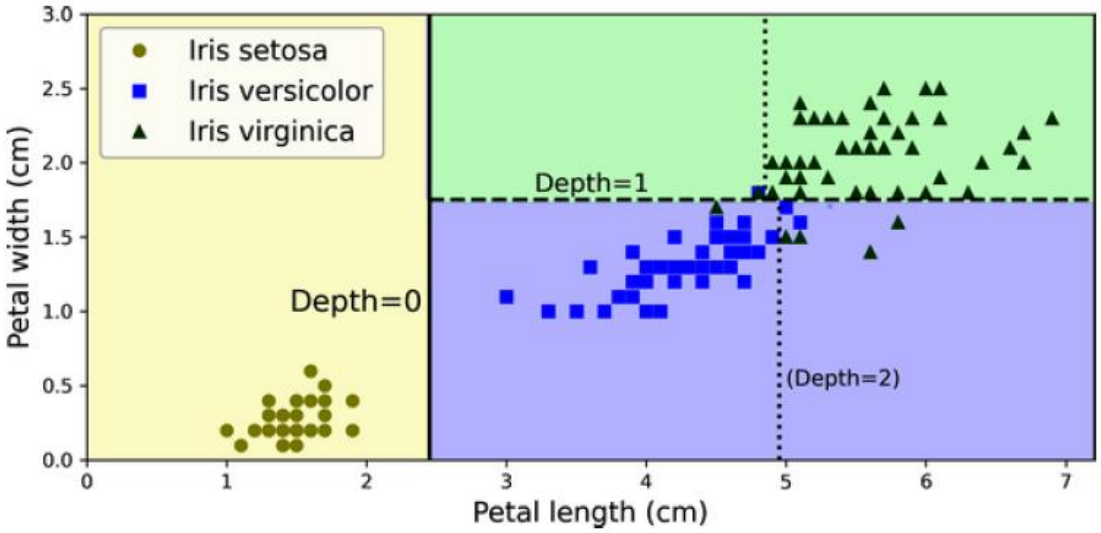


Figure 6-2. Decision tree decision boundaries

Model Interpretation: White Box vs Black Box

- Decision Trees are intuitive, their decisions are easy to interpret, provide nice, simple classification rules that can even be applied manually. They are called *White box models.*
- Random Forests or neural networks, it is hard to explain in simple terms why the predictions were made, they are considered *Black box models*
- The field of **interpretable ML** aims at creating ML systems that can explain their decisions in a way humans can understand, to ensure the system does not make unfair decisions.

Estimating Class probabilities

- A Decision Tree can also estimate the probability that an instance belongs to a particular class K
- First it traverses the tree to find the leaf node for this instance, and then returns the ratio of training instances of class K in this node.

```
>>> 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

- The *Cart* algorithm works by first splitting the training set into two subsets using a single feature ***k*** and a threshold t_k ("petal length < 2.45 cm")
- It searches for the pair**(K, t_k)** that produces the purest subsets (minimizes impurity)

Equation 6-2. CART cost function for classification

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

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

- The cart algorithm continues to split the subsets, recursively; until it reaches the maximum depth
- There is a few hyperparameters that can control additional stopping conditions (**min_samples_split, min_samples_leaf, min_weight_fraction_leaf, and max_leaf_nodes**)
- Cart algorithm is a greedy algorithm, it greedily searches for an optimum split at the top level, then repeats process at each subsequent level.
- A greedy algorithm often produces a reasonably good solution but not optimal
- Finding the optimal tree is an **NP-complete problem,** it requires **O(exp(m))** time, making it impossible even for small training sets

Computational Complexity

- The training algorithm compares all features(or less if **max_features** is set) on all samples at each node
- The training complexity is **O(n x m $\log_2$(m))**
- For small training sets, scikit-learn can speed up training by using **presort = True,** but slows down training for larger training sets
- Making predictions requires traversing the DT from the **root** to **leaf.**
- Traversing the decision tree requires going through roughly **O($\log_2$(m))** nodes

Gini impurity or Entropy?

- **Gini impurity** is used by default, to select the **entropy impurity** set the **criterion** hyperparameter to "**entropy"**
- A reduction of entropy is called an information gain.
- A set's entropy is Zero when it contains instances of only one class

Equation 6-3. Entropy

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

- Use gini impurity or entropy?
  - Most of the time it doesn't make a difference
  - Gini is slightly faster to compute, so it is a good default
  - When they differ, gini tends to isolate the most frequent class in its own branch, while entropy tends to produce slightly more balanced trees

Regularization Hyperparameters

- Decision Trees make very few assumptions about the training data, unlike linear models(assume the data is linear)
- Unconstrained Decision trees will adapt to the training data, fitting very closely (overfitting), this is called **non-parametric model** (the no. of parameters is not determined before training)
- A **parametric model** (linear model) has a determined no. of parameters, the degree of freedom is limited reducing the risk of overfitting(but increasing the risk of underfitting).
- How to restrict the Decision Tree's freedom to avoid overfitting?
  - **max_depth** hyperparameter will regularize the model(reduce the risk of overfitting)
  - **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)
  - **max_leaf_nodes** (Maximum number of leaf nodes)
  - **max_features**(Maximum number of features that are evaluated for splitting at each node)
  - increasing min_* hyperparameters or reducing max_* hyperparameters will regularize the model
- other algorithms work by first training the algorithm without any restrictions, then delete the unnecessary nodes
  - A node whose children are all leaf nodes is considered unnecessary if it is not statistically significant.
  - Standard statistical tests, such as the χ test (chisquared test), are used to estimate the null hypothesis.
  - If this probability of the p-value is higher than a given threshold, then the node is considered unnecessary, and its children are deleted.

```
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)
```

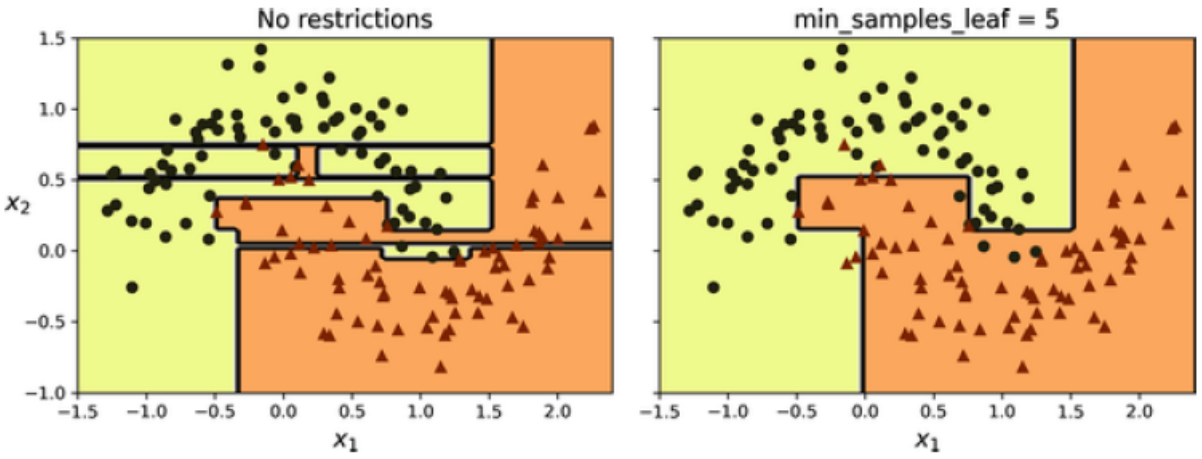


Figure 6-3. Decision boundaries of an unregularized tree (left) and a regularized tree (right)

```
>>> 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
```

- The second Tree has a better accuracy on the test set

Regression

- Decision Trees are also capable of performing regression tasks

```
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.

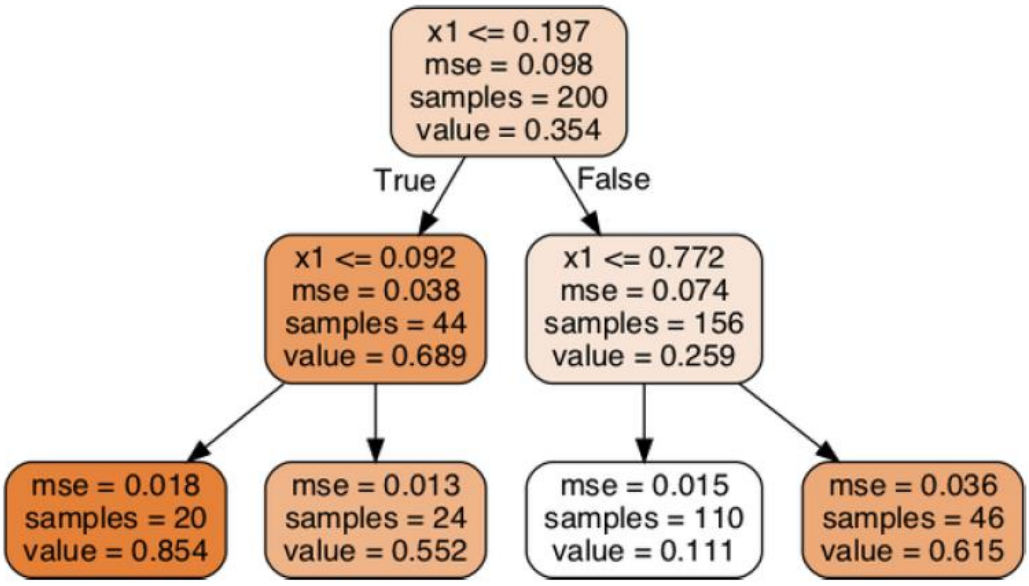


Figure 6-4. A decision tree for regression

- It predicts a value instead of a class in each node
- The **value** prediction is the average target value of the training instances associated with this leaf node
- The **mse** is the mean squared error over the instances in this leaf node
- The algorithm splits each region in a way that makes most training instances as close as possible to that predicted value.

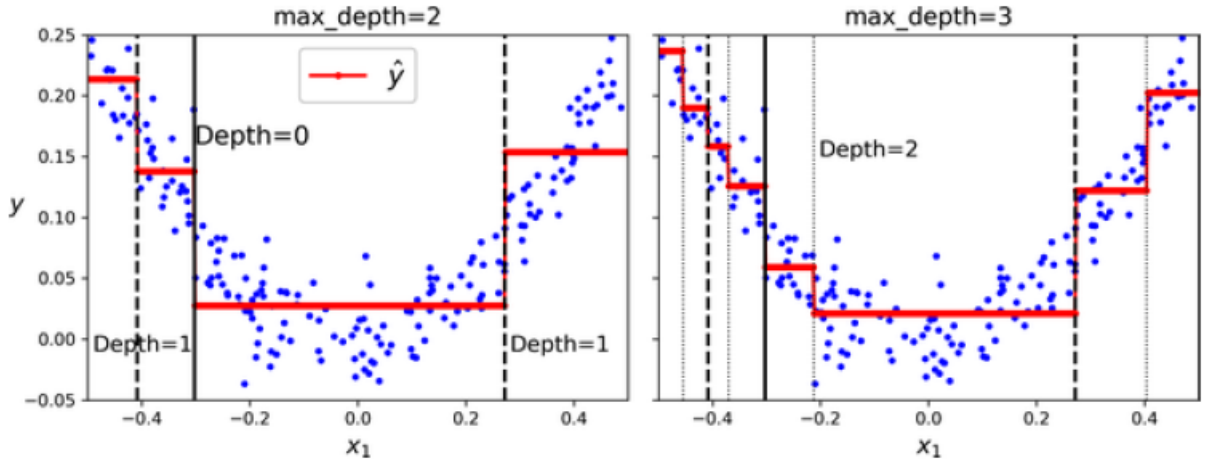


Figure 6-5. Predictions of two decision tree regression models

- The Decision Tree Regressor tries to split the training set in a way that minimizes the **MSE** instead of impurity

Equation 6-4. CART cost function for regression

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

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

- Decision Tree regressor are prone to overfitting too

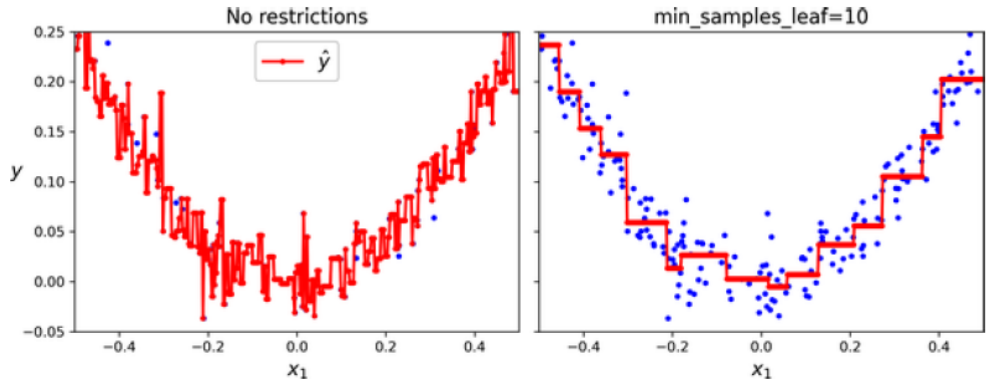
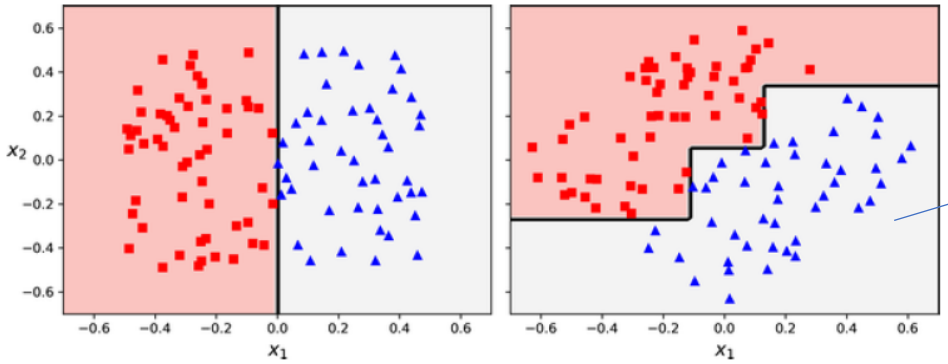


Figure 6-6. Predictions of an unregularized regression tree (left) and a regularized tree (right)

Instability

- Decision Trees has a few limitations, Decision Trees love orthogonal decision boundaries(all splits are perpendicular to an axis), which makes them sensitive to training set rotation

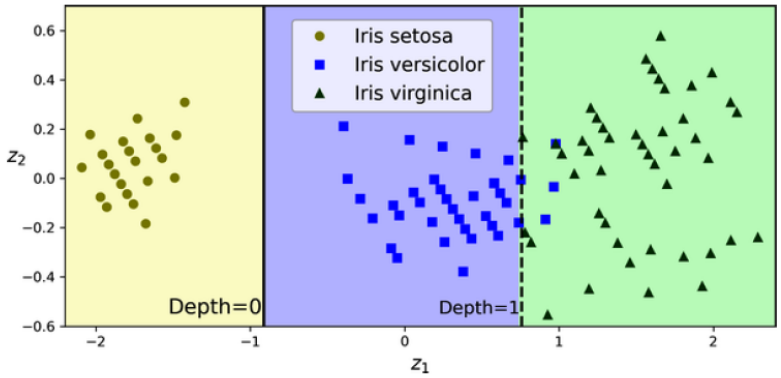


Rotate by 45◦

Figure 6-7. Sensitivity to training set rotation

- Both Decision Trees fit the training set perfectly , it is very likely that the model on the right will not generalize well
- One way to limit this problem is to scale the data, then apply a principal component analysis transformation(PCA)
  - it rotates the data in a way that reduces the correlation between the features, which often (not always) makes things easier for trees.

```
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)
```



the rotation makes it possible to fit the dataset pretty well

using only one feature, z_1 , which is a linear function of the original petal length and width.

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

Decision Trees have a high Variance

- the main issue with decision trees is that they have quite a high variance, small changes to the hyperparameters or to the data may produce very different models
- the training algorithm used by Scikit-Learn is stochastic (it randomly selects the set of features to evaluate at each node), you might get a different model every time you train the model

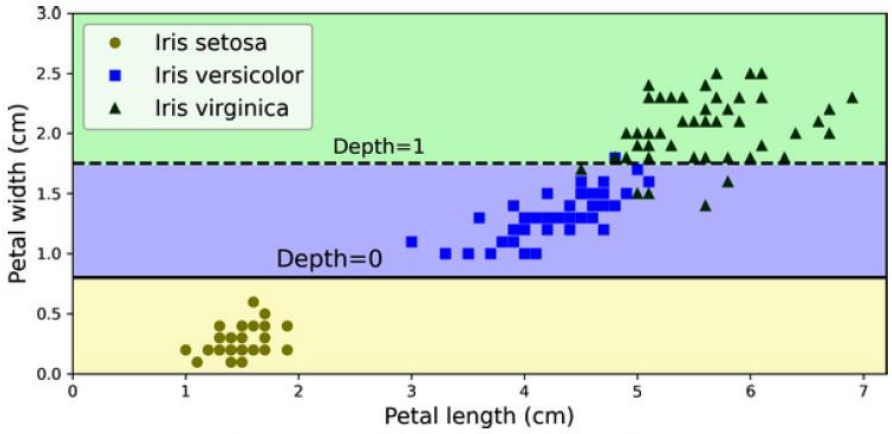


Figure 6-9. Retraining the same model on the same data may produce a very different model

- Random Forests can reduce variance significantly by averaging predictions over many trees.

---

1. What is the approximate depth of a decision tree trained (without restrictions) on a training set with one million instances?

   A binary decision tree will have a depth of $\log_2(10^6) = 20$ (maybe a bit more since the tree will generally not be perfectly well balanced

2. Is a node's Gini impurity generally lower or higher than its parent's? Is it generally lower/higher, or always lower/higher?

   A node's gini impurity is generally lower than its parent's. however, it is possible for a node to have a higher Gini impurity than its parent , as long as this increase is more than compensated for by a decrease in the other child's impurity.

3. If a decision tree is underfitting the training set, is it a good idea to try scaling the input features?

   Decision Trees don't care whether or not the training data is scaled or centered

4. If it takes one hour to train a decision tree on a training set containing one million instances, roughly how much time will it take to train another decision tree on a training set containing ten million instances? Hint: consider the CART algorithm's computational complexity
   Decision tree complexity is $O(n*m*\log_2(m))$, so $K = (n * 10m * \log_2(10m)) / (n * m * \log_2(m))$, if $m = 10^6$
   Then k = 11.7, so the training time will be roughly 11.7 hours
5. If it takes one hour to train a decision tree on a given training set, roughly how much time will it take if you double the number of features?
   Decision tree complexity is $O(n*m*\log_2(m))$, so $K = (2n*m*\log_2(m)) / (n*m*\log_2(m)) = 2$, the training time will be doubled
6. If your training set contains 100,000 instances, will setting presort = True speed up training?

   Presort speeds up training if the dataset is smaller than a few thousand instances, in case of 100k instances it will considerably slow down training