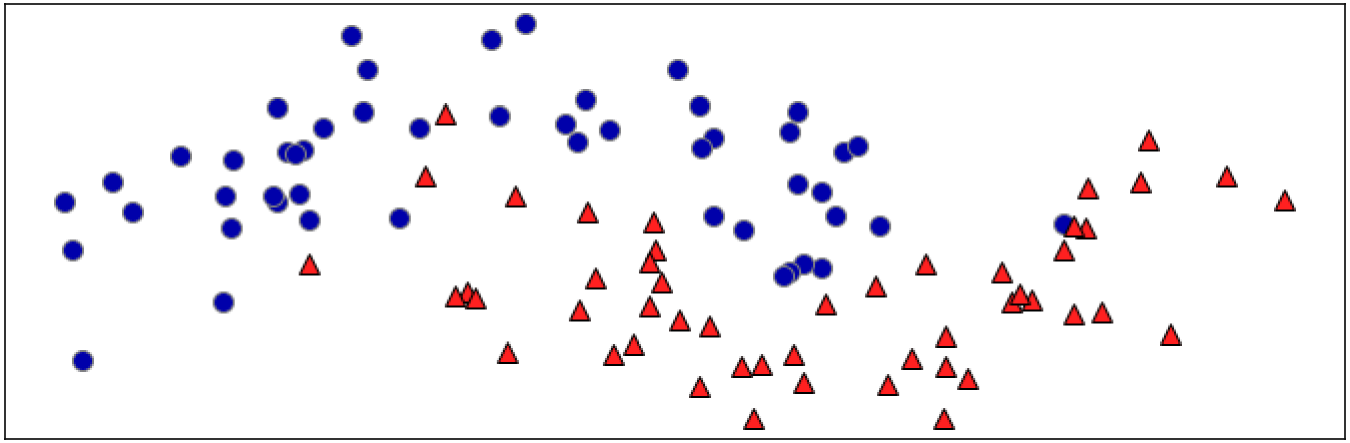
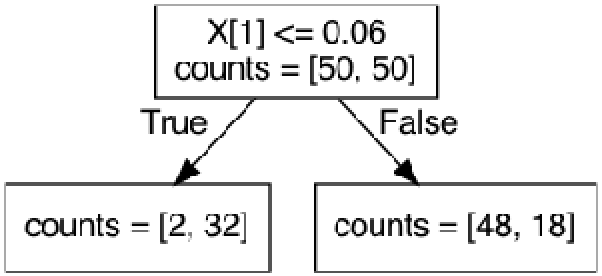
Recap: Decision Trees

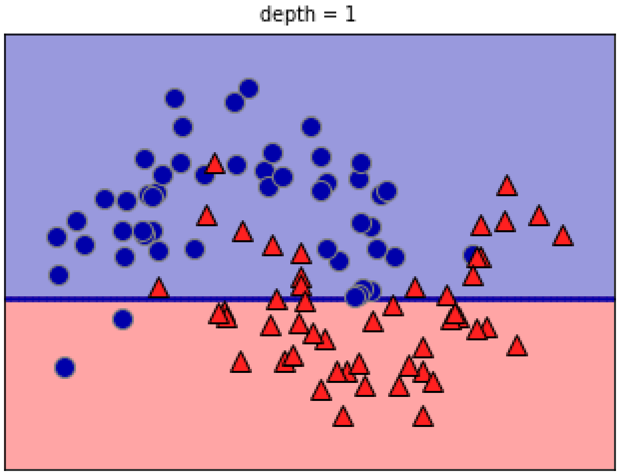
- Representation: Assume we can represent the concept we want to learn with a *decision tree*
    - Repeatedly split the data based on one feature at a time
    - Note: *Oblique trees* can split on combinations of features
- Evaluation (loss): One tree is better that another tree according to some heuristic
    - Classification: Instances in a leaf are all of the same class (`pure` leafs)
    - Regression: Instances in a leaf have values close to each other
- Optimization: Recursive, heuristic greedy search (Hunt's algorithm)
    - Make first split based on the heuristic
    - In each branch, repeat splitting in the same way

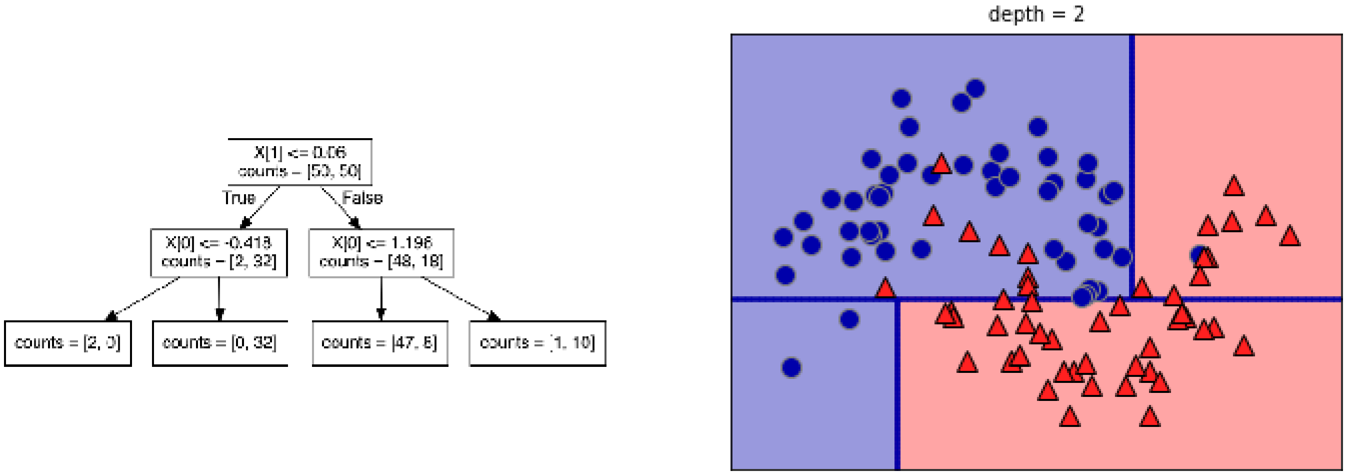
Decision Tree classification

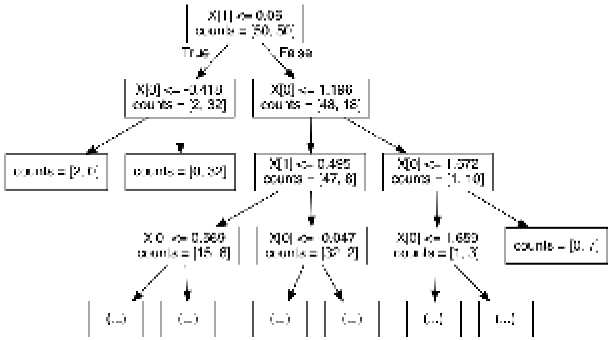
Where would you make the first split?

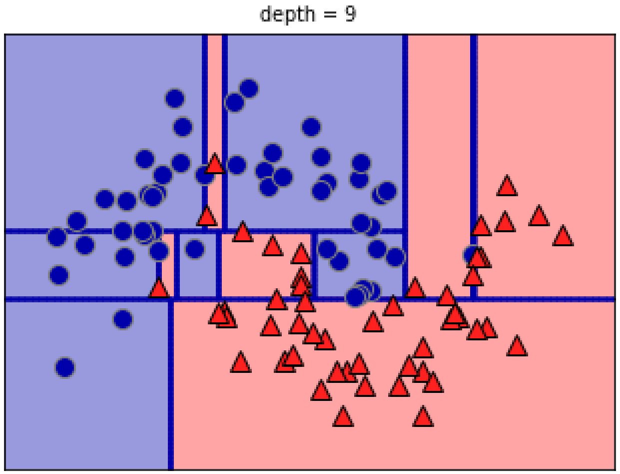



X[1] <= 0.06
counts = [50, 50]
True
False
counts = [2, 32]
counts = [48, 18]


depth = 1


X[1] <= 0.06
counts = [50, 50]
True
False
X[0] <= -0.418
counts = [2, 32]
X[0] <= 1.196
counts = [48, 18]
counts = [2, 0]
counts = [0, 32]
counts = [47, 8]
counts = [1, 10]
depth = 2


True
False


depth = 9

Heuristics

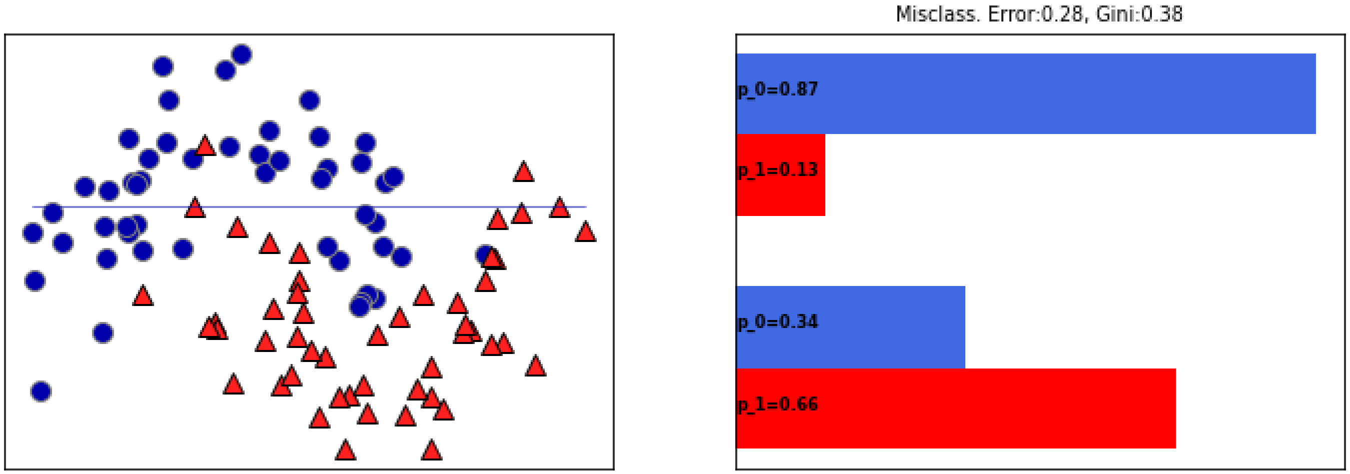
- We start from a dataset of n points $D = \{(x_i, y_i)\}_{i=1}^n$ where y_i is one of k classes
- Consider splits between adjacent data point of different class, for every variable
- After splitting, each leaf will have $\hat{p}_k$ = the relative frequency of class k

We can define several *impurity measuresz*:

- Misclassification Error (leads to larger trees): $1 - \underset{k}{\operatorname{argmax}} \hat{p}_k$
- Gini-Index: $\sum_{k \neq k'} \hat{p}_k \hat{p}_{k'} = \sum_{k=1}^K \hat{p}_k (1 - \hat{p}_k)$
- Sum up the heuristics per leaf, weighted by the number of examples in each leaf

$$\sum_{l=1}^{L} \frac{|X_{i=l}|}{|X_i|} Gini(X_{i=l})$$

Visualization: the plots on the right show the class distribution of the 'top' and 'bottom' leaf, respectively.



- Entropy (of the class attribute) measures *unpredictability* of the data:
    - How likely will random example have class k?

$$E(X) = -\sum_{k=1}^{K} \hat{p}_k \log_2 \hat{p}_k$$

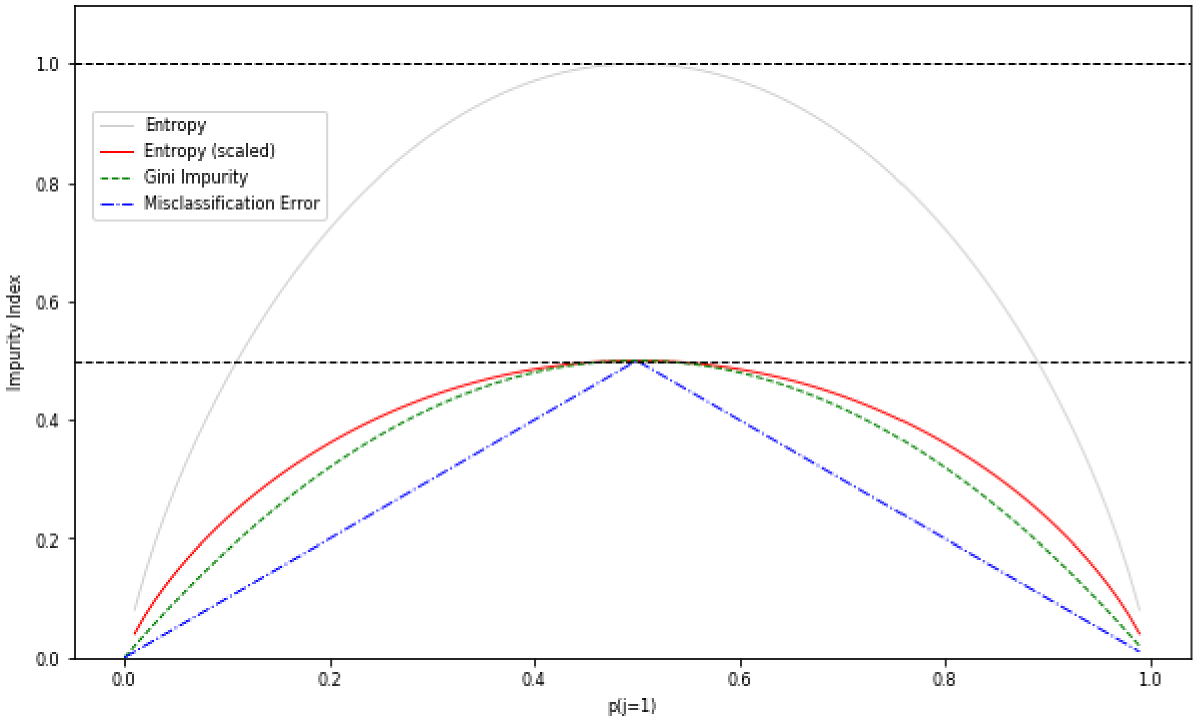
- Information Gain (a.k.a. Kullback–Leibler divergence) measures how much entropy has reduced by splitting on attribute X_i:

$$G(X, X_i) = E(X) - \sum_{l=1}^{L} \frac{|X_{i=l}|}{|X_i|} E(X_{i=l})$$

with X = the training set, l a specific leaf after splitting on feature X_i, $X_{i=l}$ is the set of examples in leaf l: $\{x \in X | X_i \in l\}$

Heuristics visualized (binary class)

- Note that `gini != entropy/2`



Example

Compute information gain for a dataset with categorical features:

Ex.	1	2	3	4	5	6
a1	T	T	T	F	F	F
a2	T	T	F	F	T	T
class	+	+	-	+	-	-

$$E(X)?$$

$$G(X, X_{a2})?$$

$$G(X, X_{a1})?$$

Ex.	1	2	3	4	5	6
a1	T	T	T	F	F	F
a2	T	T	F	F	T	T
class	+	+	-	+	-	-

$E(X)$ = $-(\frac{1}{2} * log_2(\frac{1}{2}) + \frac{1}{2} * log_2(\frac{1}{2})) = 1$ (classes have equal probabilities)
$G(X, X_{a2})$ = 0 (after split, classes still have equal probabilities, entropy stays 1)

Ex.	**1**	**2**	**3**	**4**	**5**	**6**
a1	T	T	T	F	F	F
a2	T	T	F	F	T	T
class	+	+	-	+	-	-

$$E(X) = -\sum_{k=1}^{K} \hat{p}_k \log \hat{p}_k \quad , \quad G(X, X_i) = E(X) - \sum_{v=1}^{V} \frac{|X_{i=v}|}{|X_i|} E(X_{i=v})$$

$$E(X_{a1=T}) = -\frac{2}{3}\log_2(\frac{2}{3}) - \frac{1}{3}\log_2(\frac{1}{3}) = 0.9183 \quad (= E(X_{a1=F}))$$

$$G(X, X_{a1}) = 1 - \frac{1}{2}0.9183 - \frac{1}{2}0.9183 = 0.0817$$

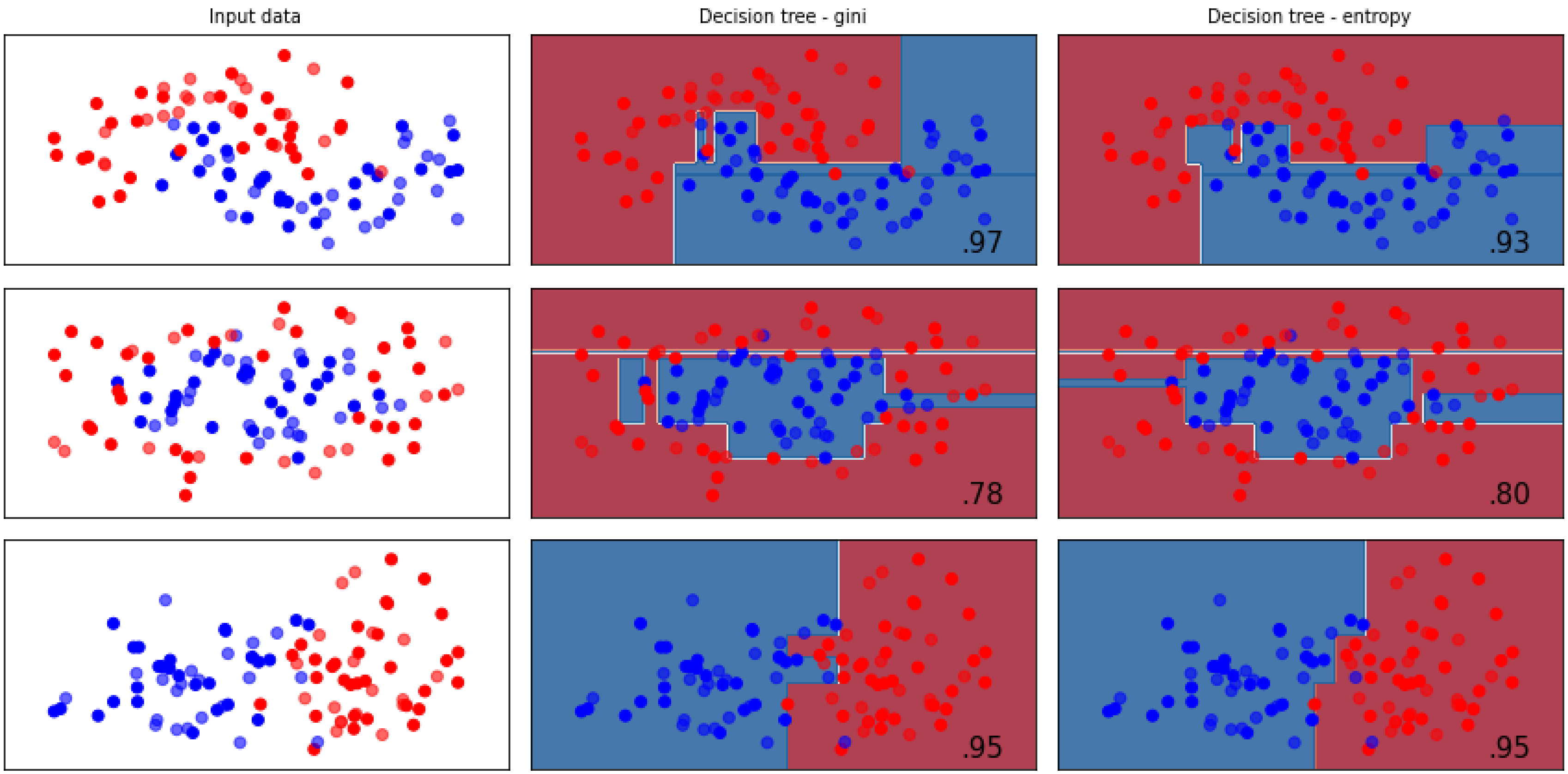
hence we split on a1

Heuristics in scikit-learn

The splitting criterion can be set with the `criterion` option in `DecisionTreeClassifier`

- `gini` (default): gini impurity index
- `entropy`: information gain

Best value depends on dataset, as well as other hyperparameters


Input data
Decision tree - gini
Decision tree - entropy
.97
.93
.78
.80
.95
.95

Handling many-valued features

What happens when a categorical feature has (almost) as many values as examples?

- Information Gain will select it

One approach: use Gain Ratio instead (not available scikit-learn):

$$GainRatio(X, X_i) = \frac{Gain(X, X_i)}{SplitInfo(X, X_i)}$$

$$SplitInfo(X, X_i) = -\sum_{v=1}^{V} \frac{|X_{i=v}|}{|X|} log_2 \frac{|X_{i=v}|}{|X|}$$

where $X_{i=v}$ is the subset of examples for which feature X_i has value v.

SplitInfo will be big if X_i fragments the data into many small subsets, resulting in a smaller Gain Ratio.

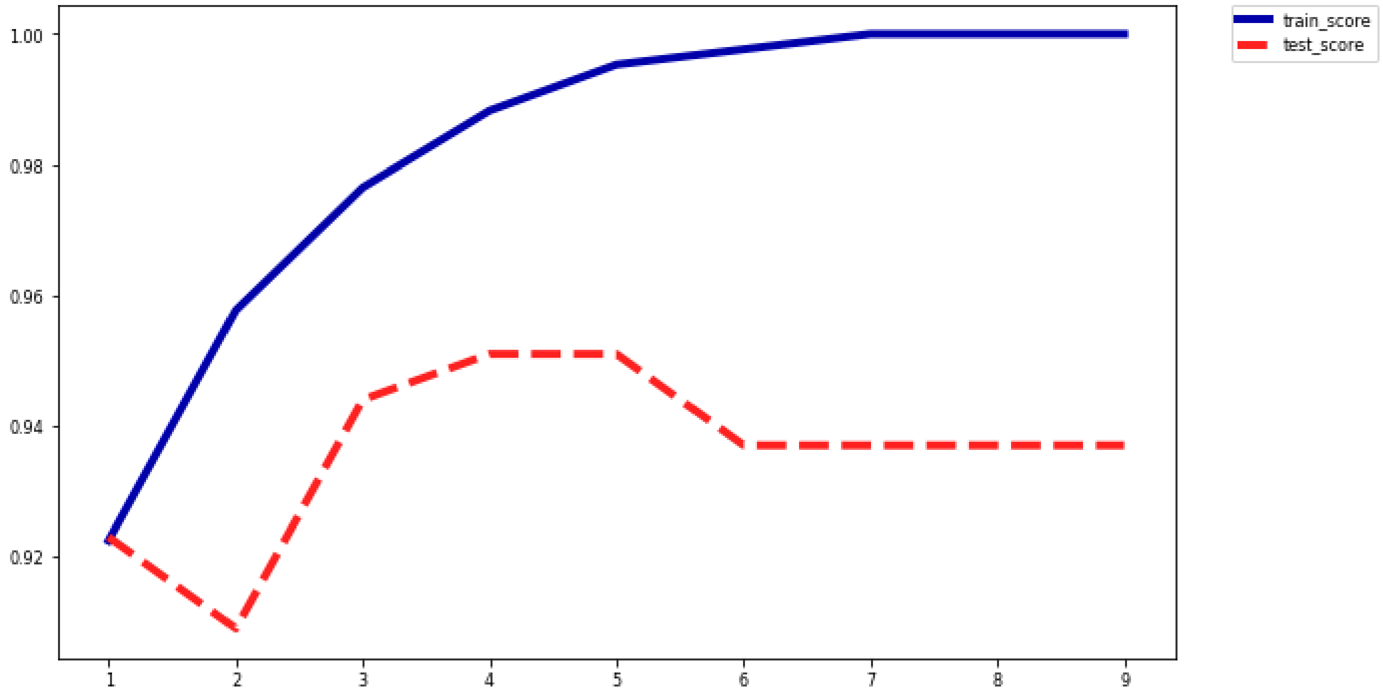
Overfitting: Controlling complexity of Decision Trees

Decision trees can very easily overfit the data. Regularization strategies:

- Pre-pruning: stop creation of new leafs at some point
    - Limiting the depth of the tree, or the number of leafs
    - Requiring a minimal leaf size (number of instances) to allow a split
- Post-pruning: build full tree, then prune (join) leafs
    - Reduced error pruning: evaluate against held-out data
    - Many other strategies exist.
    - scikit-learn supports none of them (yet)

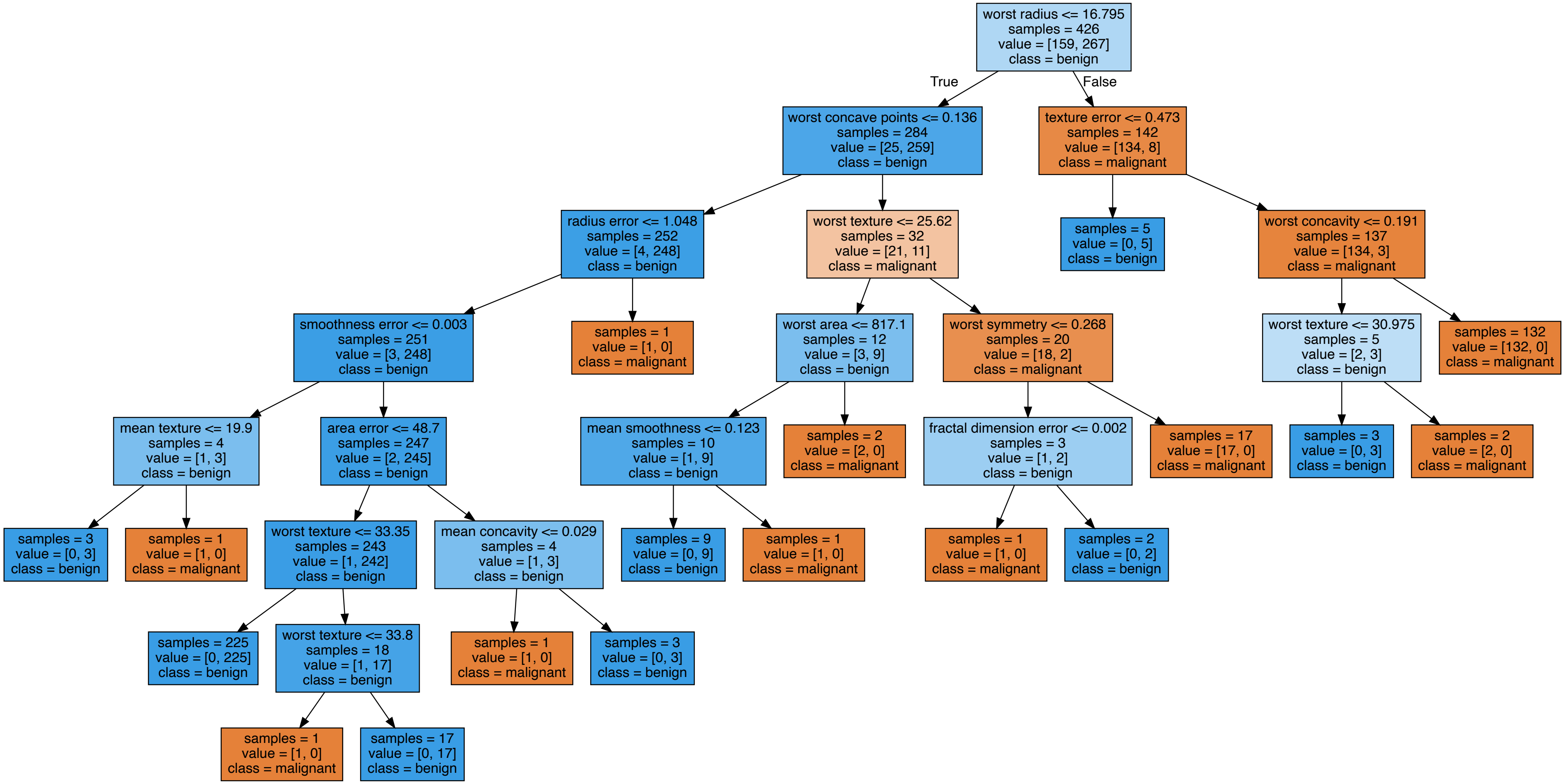
Effect of pre-pruning:

- Shallow trees tend to underfit (high bias)
- Deep trees tend to overfit (high variance)



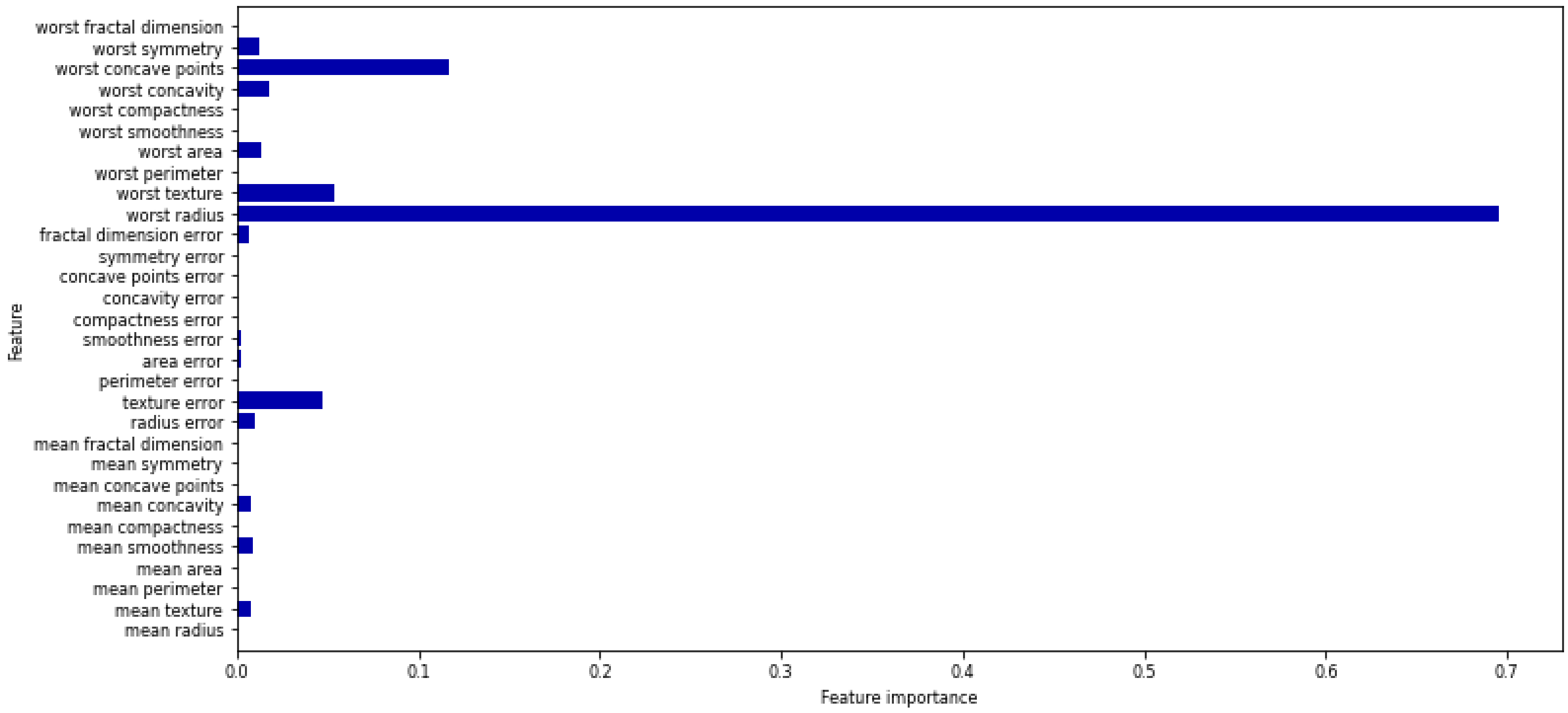
Decision Trees are easy to interpet

- Visualize and find the path that most data takes



`DecisionTreeClassifier` also returns *feature importances*

- In [0,1], sum up to 1
- High values for features selected early (near the root)



Decision tree regression

- Heuristic: *Minimal quadratic distance*
- Consider splits at every data point for every variable (or halfway between)
- Dividing the data on X_j at splitpoint s leads to the following half-spaces:

$$R_1(j,s) = X : X_j \leq s \quad and \quad R_2(j,s) = X : X_j > s$$

- The best split, with predicted value c_i (mean of all values in the leaf) and actual value y_i:

$$\min_{j,s}\left(\min_{c_1}\sum_{x_i \in R_1(j,s)}(y_i - c_1)^2 + \min_{c_2}\sum_{x_i \in R_2(j,s)}(y_i - c_2)^2\right)$$

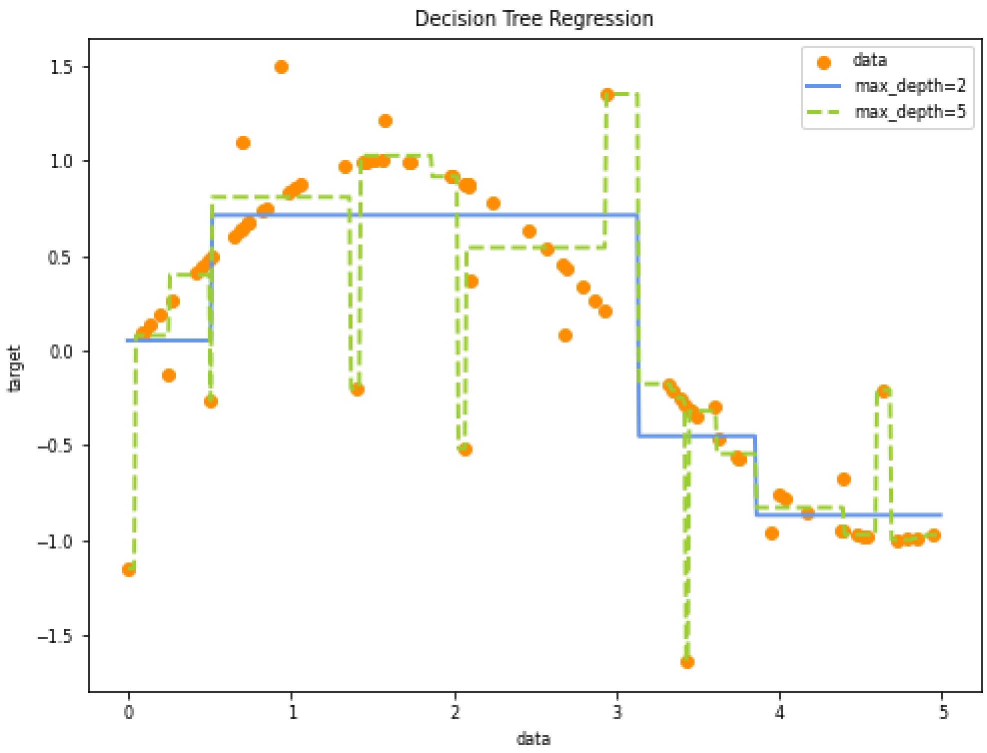
- Assuming that the tree predicts y_i as the average of all x_i in the leaf:

$$\hat{c}_1 = \text{avg}(y_i|x_i \in R_1(j,s)) \quad and \quad \hat{c}_2 = \text{avg}(y_i|x_i \in R_2(j,s))$$

with x_i being the i-th example in the data, with target value y_i

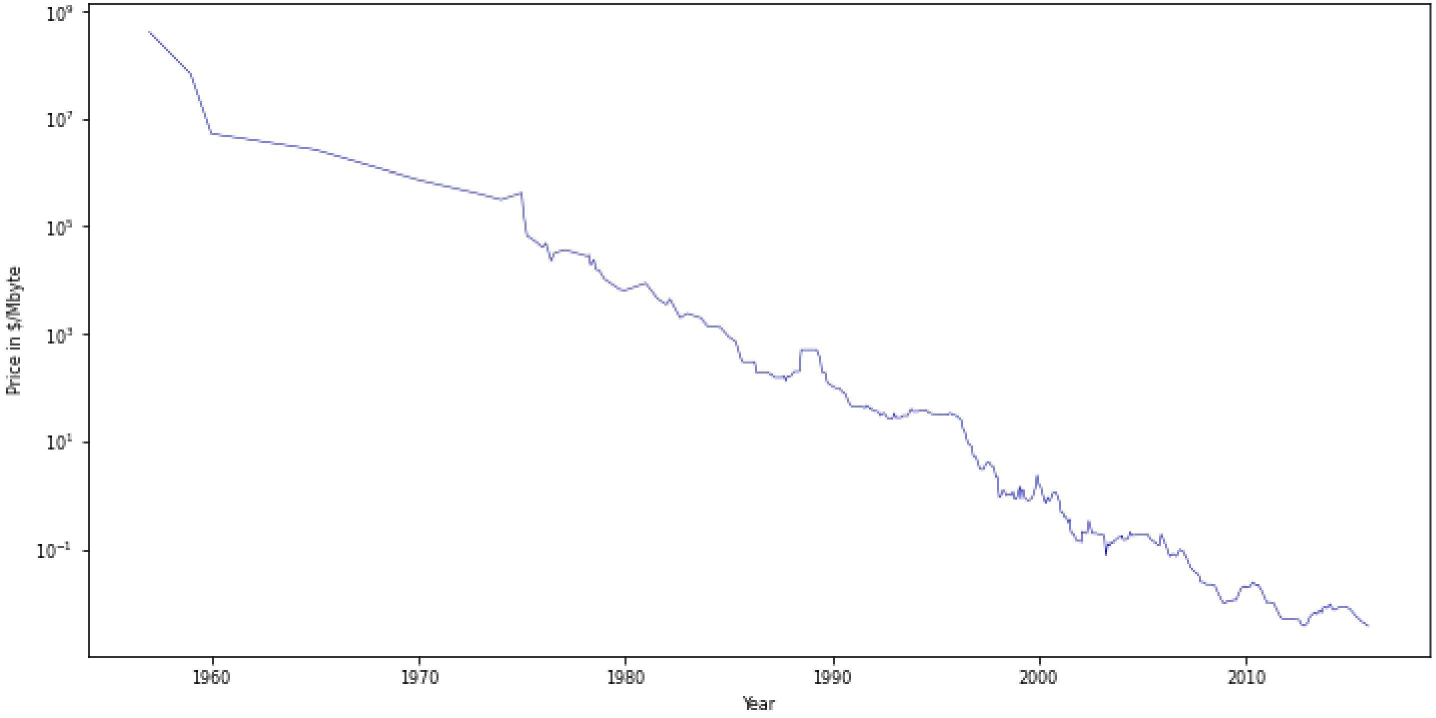
In scikit-learn

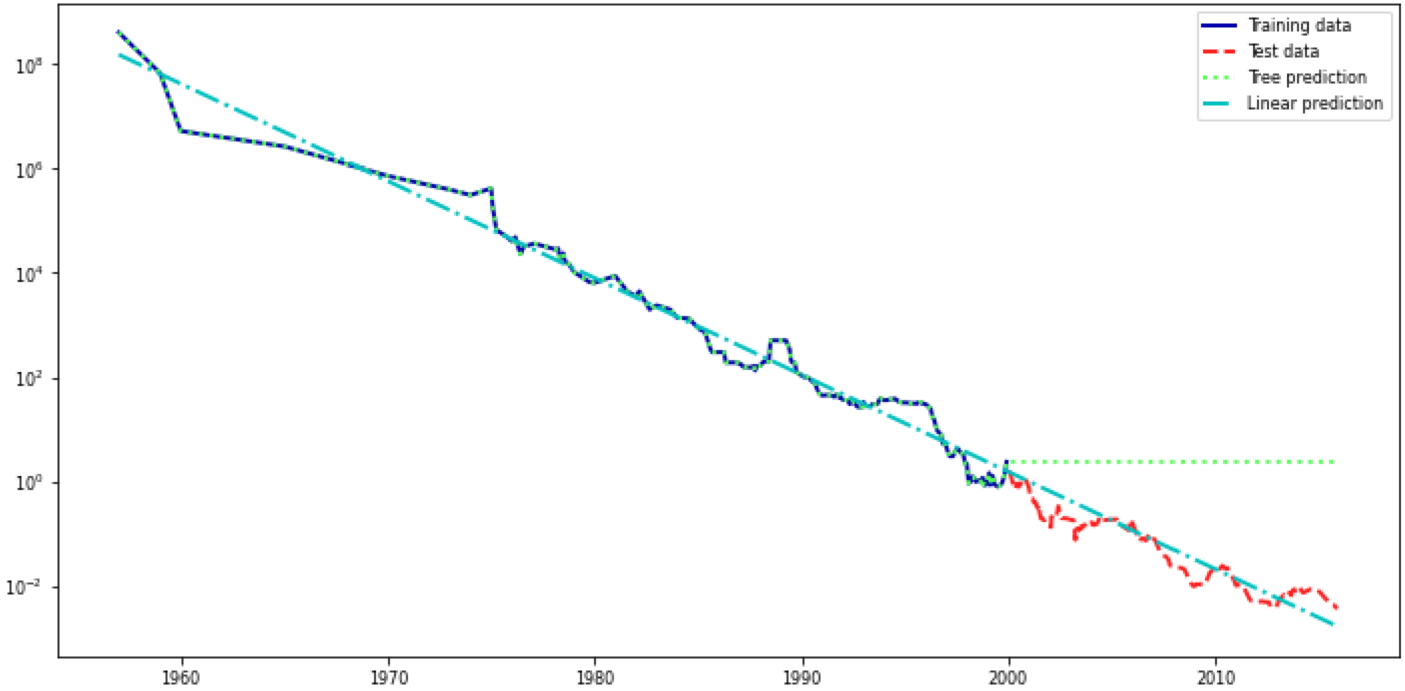
Regression is done with `DecisionTreeRegressor`



Note that decision trees do not extrapolate well.

- The leafs return the same *mean* value no matter how far the new data point lies from the training examples.
- Example on the `ram_price` forecasting dataset

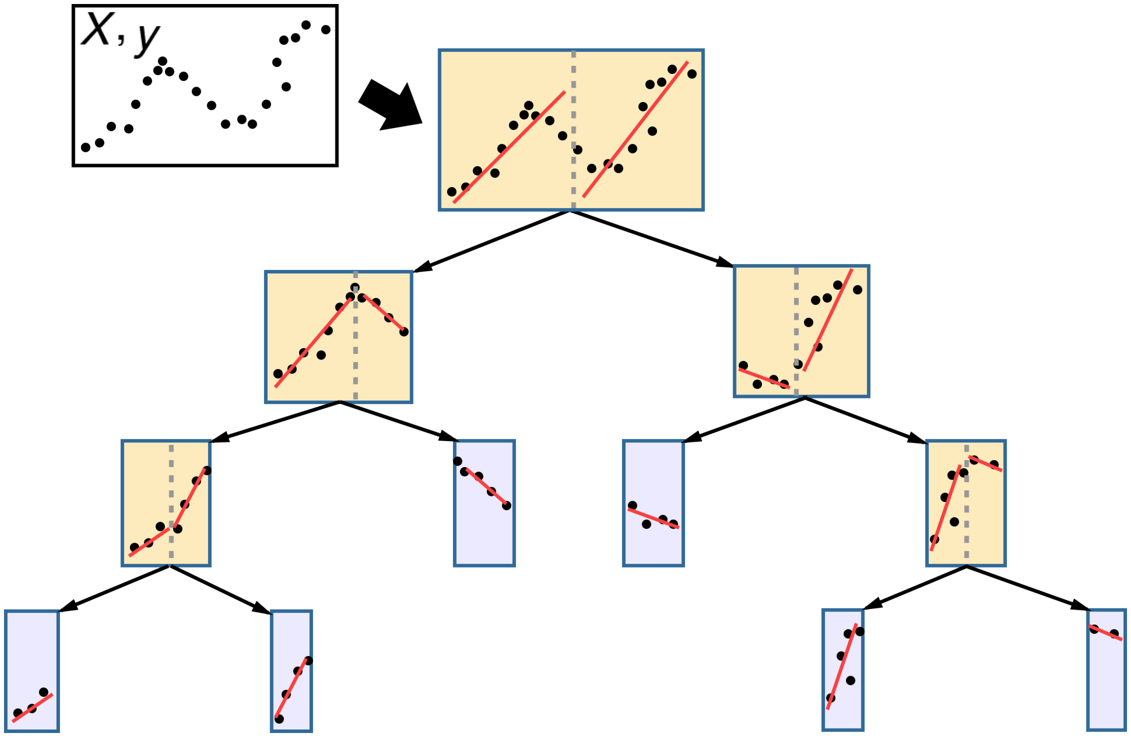




Training data
Test data
Tree prediction
Linear prediction
10^8
10^6
10^4
10^2
10^0
10^{-2}
1960
1970
1980
1990
2000
2010

Model trees

- Instead of predicting a single value per leaf (e.g. mean value for regression), you can build a model on all the points remaining in a leaf
    - E.g. a linear regression model
- Can learn more complex concepts, extrapolates better. Overfits easily.



Strengths, weaknesses and parameters

Decision trees:

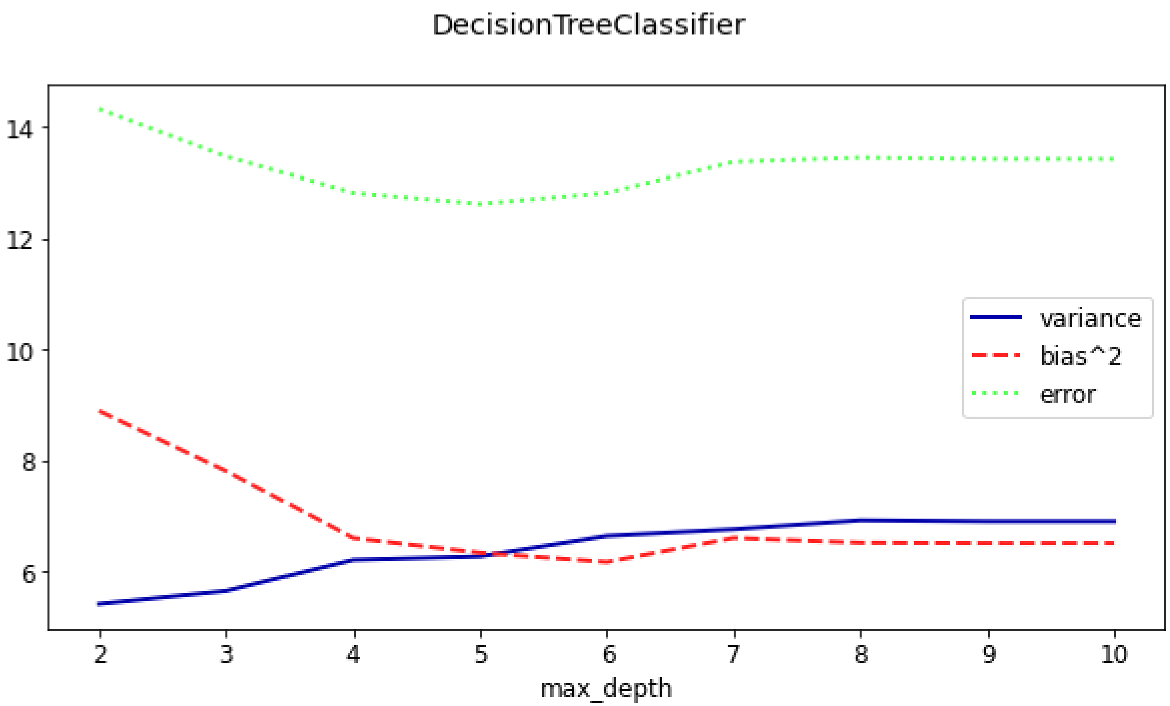
- Work well with features on completely different scales, or a mix of binary and continuous features
    - Does not require normalization
- Interpretable, easily visualized
- Tend to overfit easily.

Pre-pruning: regularize by:

- Setting a low `max_depth`, `max_leaf_nodes`
- Setting a higher `min_samples_leaf` (default=1)

On under- and overfitting

- Let's study which types of errors are made by decision trees
- Deep trees have high variance but low bias
    - What if we built many deep trees and average them out to reduce variance?
- Shallow trees have high bias but very low variance
    - What if we could correct the systematic mistakes to reduce bias?



Algorithm overview

Name	Representation	Loss function	Optimization	Regularization
Classification trees	Decision tree	Information Gain (KL div.) / Gini index	Hunt's algorithm	Tree depth,...
Regression trees	Decision tree	Min. quadratic distance	Hunt's algorithm	Tree depth,...
Model trees	Decision tree + other models in leafs	As above + used model's loss	Hunt's algorithm + used model's optimization	Tree depth,...