Decision Trees

PHYS 453

Dr Daugherity

Introduction

- 20 questions
- Guess Who?
- http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

Decision Trees

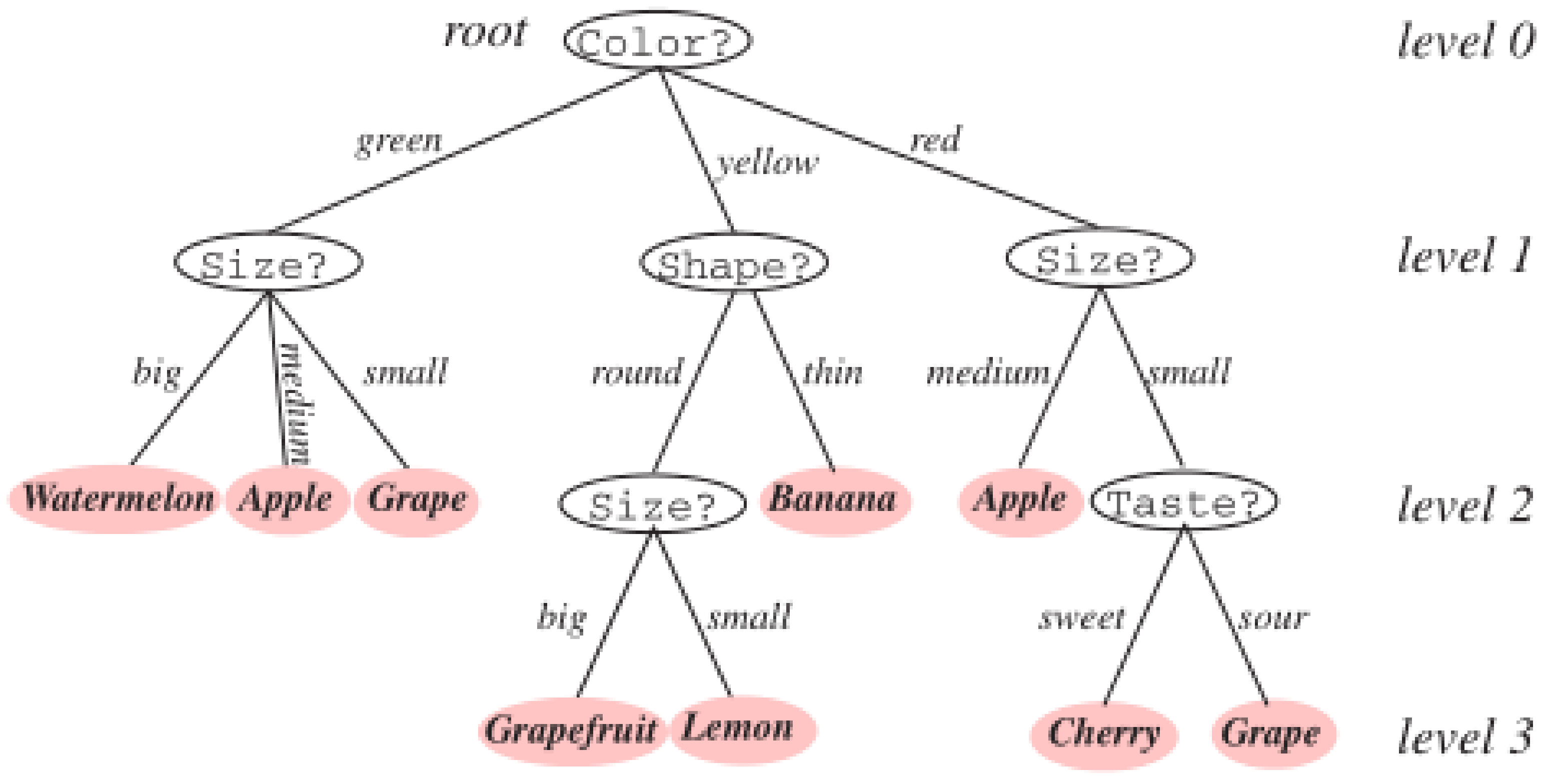


FIGURE 8.1. Classification in a basic decision tree proceeds from top to bottom. The questions asked at each node concern a particular property of the pattern, and the downward links correspond to the possible values. Successive nodes are visited until a terminal or leaf node is reached, where the category label is read. Note that the same question, `Size?`, appears in different places in the tree and that different questions can have different numbers of branches. Moreover, different leaf nodes, shown in pink, can be labeled by the same category (e.g., **Apple**). From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Binary Decision Trees

yes/no questions only!

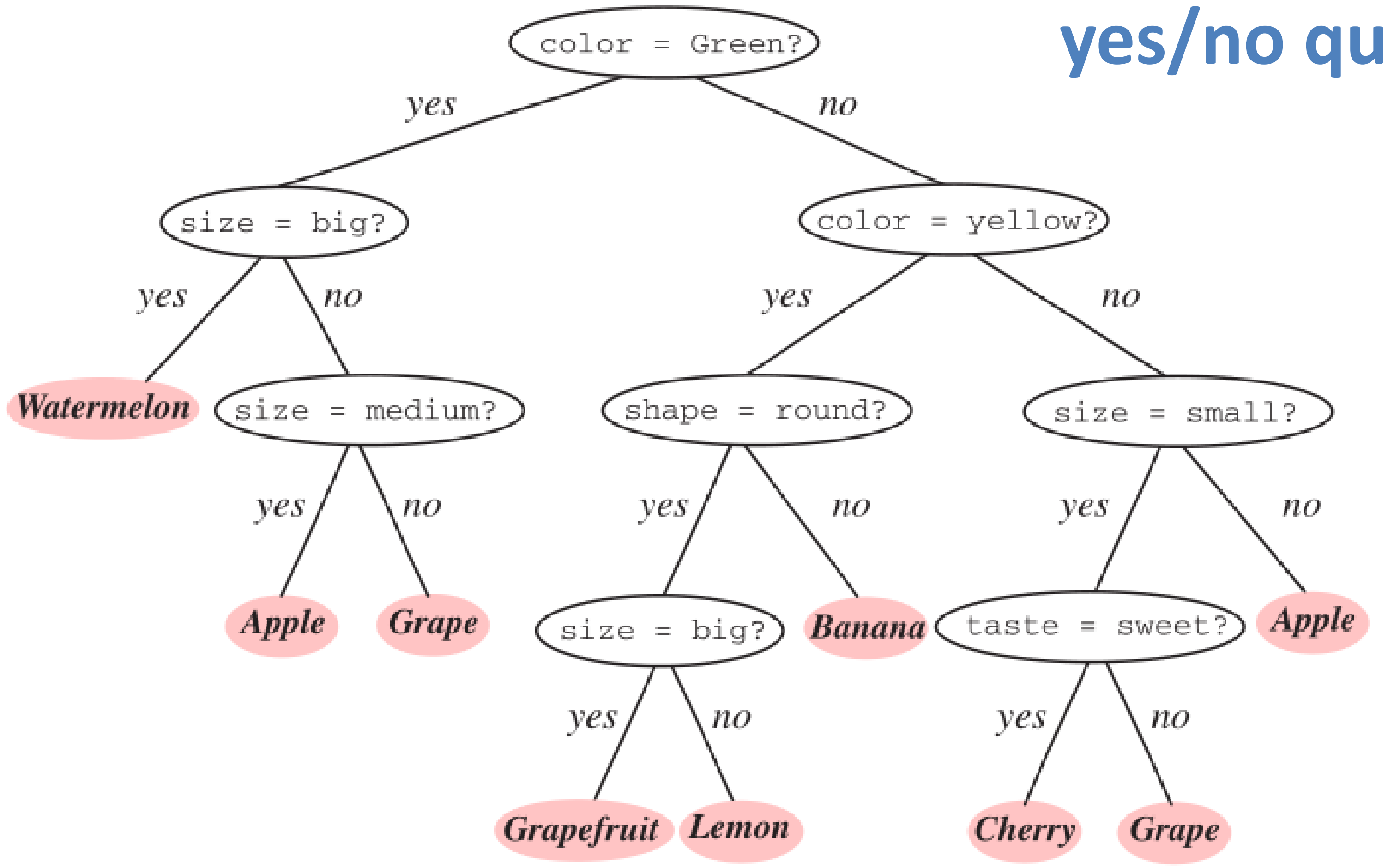


FIGURE 8.2. A tree with arbitrary branching factor at different nodes can always be represented by a functionally equivalent binary tree—that is, one having branching factor $B = 2$ throughout, as shown here. By convention the "yes" branch is on the left, the "no" branch on the right. This binary tree contains the same information and implements the same classification as that in Fig. 8.1. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

CONCEPTS

Suppose we have the following five positive examples (the first three are th
same as in Example 4.1):

p1: $\text{Length} = 3 \ \wedge \text{Gills} = \text{no} \ \wedge \text{Beak} = \text{yes} \ \wedge \text{Teeth} = \text{many}$
p2: $\text{Length} = 4 \ \wedge \text{Gills} = \text{no} \ \wedge \text{Beak} = \text{yes} \ \wedge \text{Teeth} = \text{many}$
p3: $\text{Length} = 3 \ \wedge \text{Gills} = \text{no} \ \wedge \text{Beak} = \text{yes} \ \wedge \text{Teeth} = \text{few}$
p4: $\text{Length} = 5 \ \wedge \text{Gills} = \text{no} \ \wedge \text{Beak} = \text{yes} \ \wedge \text{Teeth} = \text{many}$
p5: $\text{Length} = 5 \ \wedge \text{Gills} = \text{no} \ \wedge \text{Beak} = \text{yes} \ \wedge \text{Teeth} = \text{few}$

and the following negatives (the first one is the same as in Example 4.2):

n1: $\text{Length} = 5 \ \wedge \text{Gills} = \text{yes} \ \wedge \text{Beak} = \text{yes} \ \wedge \text{Teeth} = \text{many}$
n2: $\text{Length} = 4 \ \wedge \text{Gills} = \text{yes} \ \wedge \text{Beak} = \text{yes} \ \wedge \text{Teeth} = \text{many}$
n3: $\text{Length} = 5 \ \wedge \text{Gills} = \text{yes} \ \wedge \text{Beak} = \text{no} \ \wedge \text{Teeth} = \text{many}$
n4: $\text{Length} = 4 \ \wedge \text{Gills} = \text{yes} \ \wedge \text{Beak} = \text{no} \ \wedge \text{Teeth} = \text{many}$
n5: $\text{Length} = 4 \ \wedge \text{Gills} = \text{no} \ \wedge \text{Beak} = \text{yes} \ \wedge \text{Teeth} = \text{few}$

Can you tell “by eye” where to make the first split?

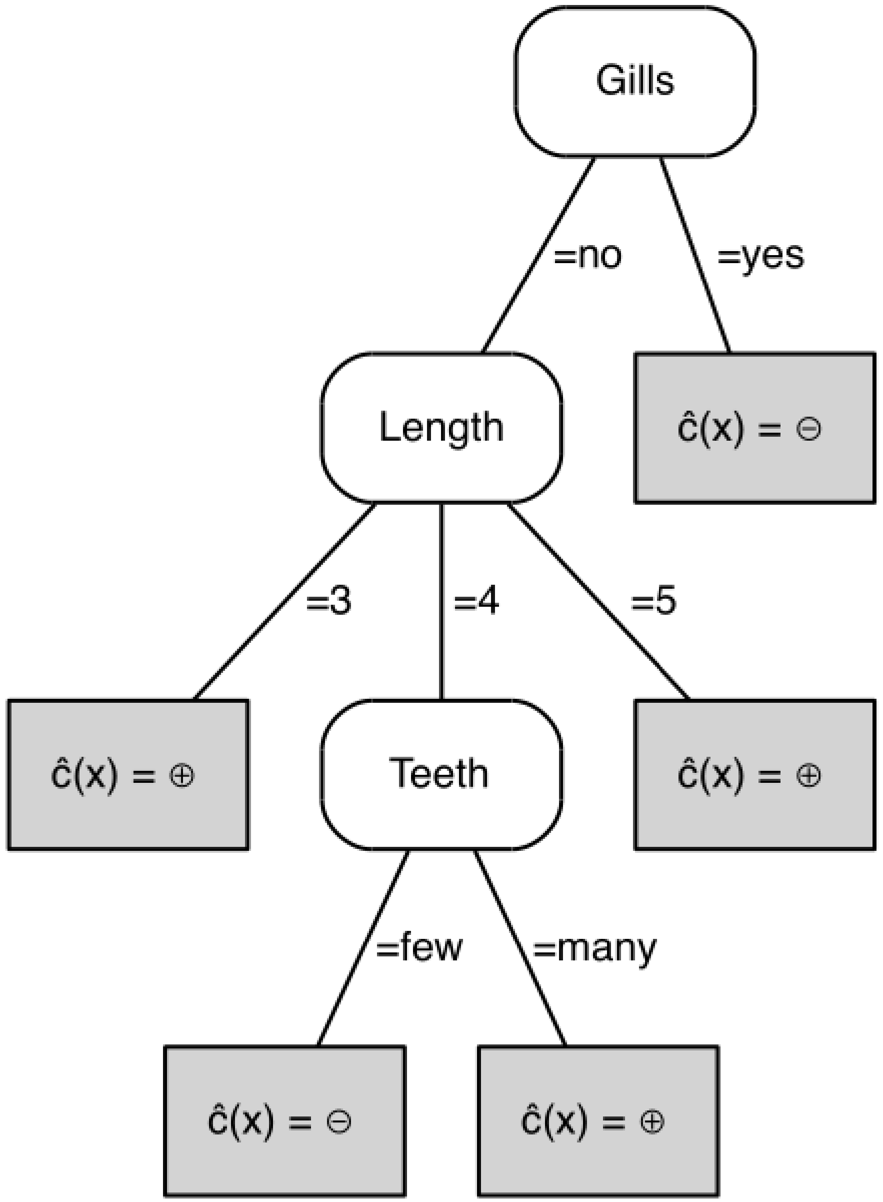
Not a binary tree

Suppose we have the following five positive examples (the first three are th
same as in Example 4.1):

p1: Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p2: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p3: Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few
p4: Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p5: Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few

and the following negatives (the first one is the same as in Example 4.2):

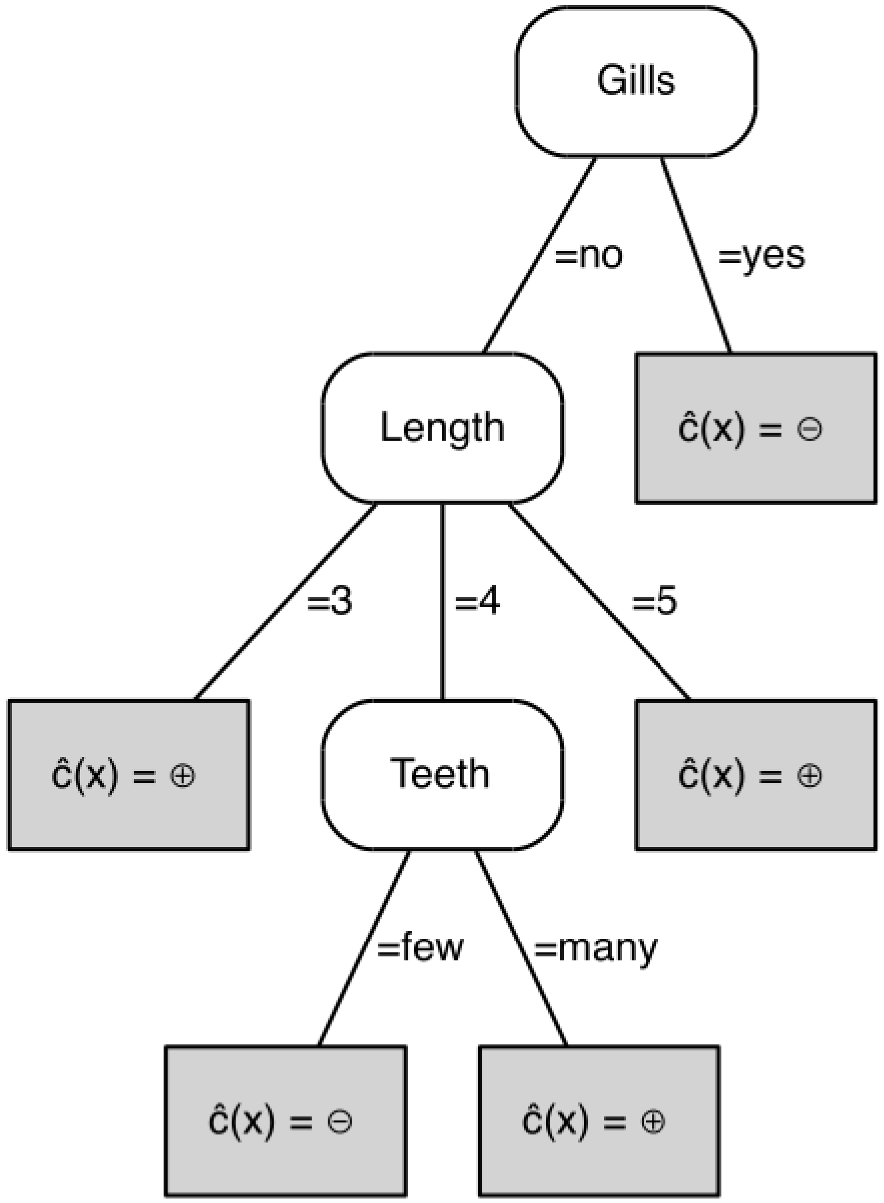
n1: Length = 5 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
n2: Length = 4 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
n3: Length = 5 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
n4: Length = 4 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
n5: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few



A **feature tree** is a tree such that each internal node (the nodes that are not leaves) is labelled with a feature, and each edge emanating from an internal node is labelled with a literal.

The set of literals at a node is called a **split**.

Each leaf of the tree represents a logical expression which is the combination of all nodes encountered on the path from root to leaf



Algorithm GrowTree(D, F) – grow a feature tree from training data.

```
Input  : data $D$; set of features $F$.
Output : feature tree $T$ with labelled leaves.
1 if Homogeneous($D$) then return Label($D$);
2 $S \leftarrow$BestSplit($D, F$) ;              // e.g., BestSplit-Class (Algorithm 5.2)
3 split $D$ into subsets $D_i$ according to the literals in $S$;
4 for each $i$ do
5     if $D_i \neq \emptyset$ then $T_i \leftarrow$GrowTree($D_i, F$) ;
6     else $T_i$ is a leaf labelled with Label($D$);
7 end
8 return a tree whose root is labelled with $S$ and whose children are $T_i$
```

To find the best split we try all the features and choose the one that minimizes **impurity**

Algorithm BestSplit-Class(D, F) – find the best split for a decision tree.

Input : data D; set of features F.

Output : feature f to split on.

```
$I_{\min} \leftarrow 1$;
for each $f \in F$ do
    split $D$ into subsets $D_1, \ldots, D_l$ according to the values $v_j$ of $f$;
    if $\mathrm{Imp}(\{D_1, \ldots, D_l\}) < I_{\min}$ then
        $I_{\min} \leftarrow \mathrm{Imp}(\{D_1, \ldots, D_l\})$;
        $f_{\text{best}} \leftarrow f$;
    end
end
return $f_{\text{best}}$
```

Impurity

Given two classes $\oplus$ and $\ominus$ with samples sizes n_+ and n_-

$$p = \frac{n_+}{n_+ + n_-}$$

proportion/fraction of + class

We want impurity that is:

- 0 whenever p is 0 or 1
- same if we swap p for 1-p
- maximum when p=1/2

Impurity

Minority class: $\min(p, 1-p)$

Gini Index: $\boldsymbol{2p(1-p)}$

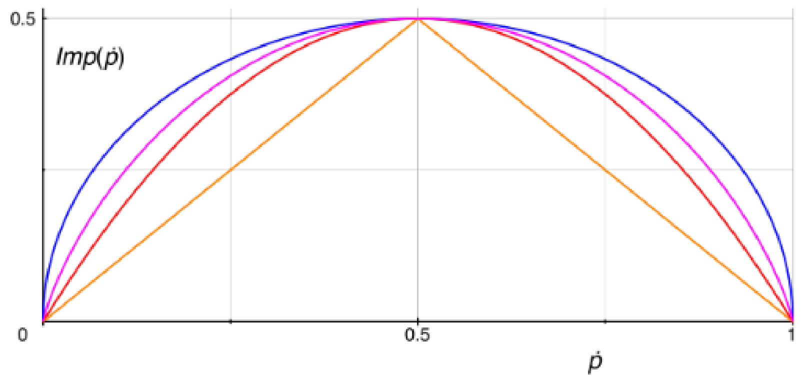
expected error if we labelled all samples randomly

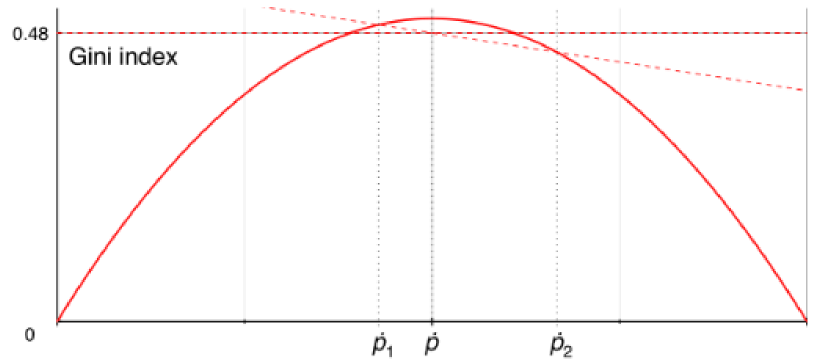
Entropy: $-p\log p \; - (1-p)\log(1-p)$

number of bits to describe classes

The most common choice is $Gini$ or $\sqrt{Gini}$

Entropy and Gini index are sensitive to fluctuations in the class distribution, $\sqrt{\text{Gini}}$ isn't.





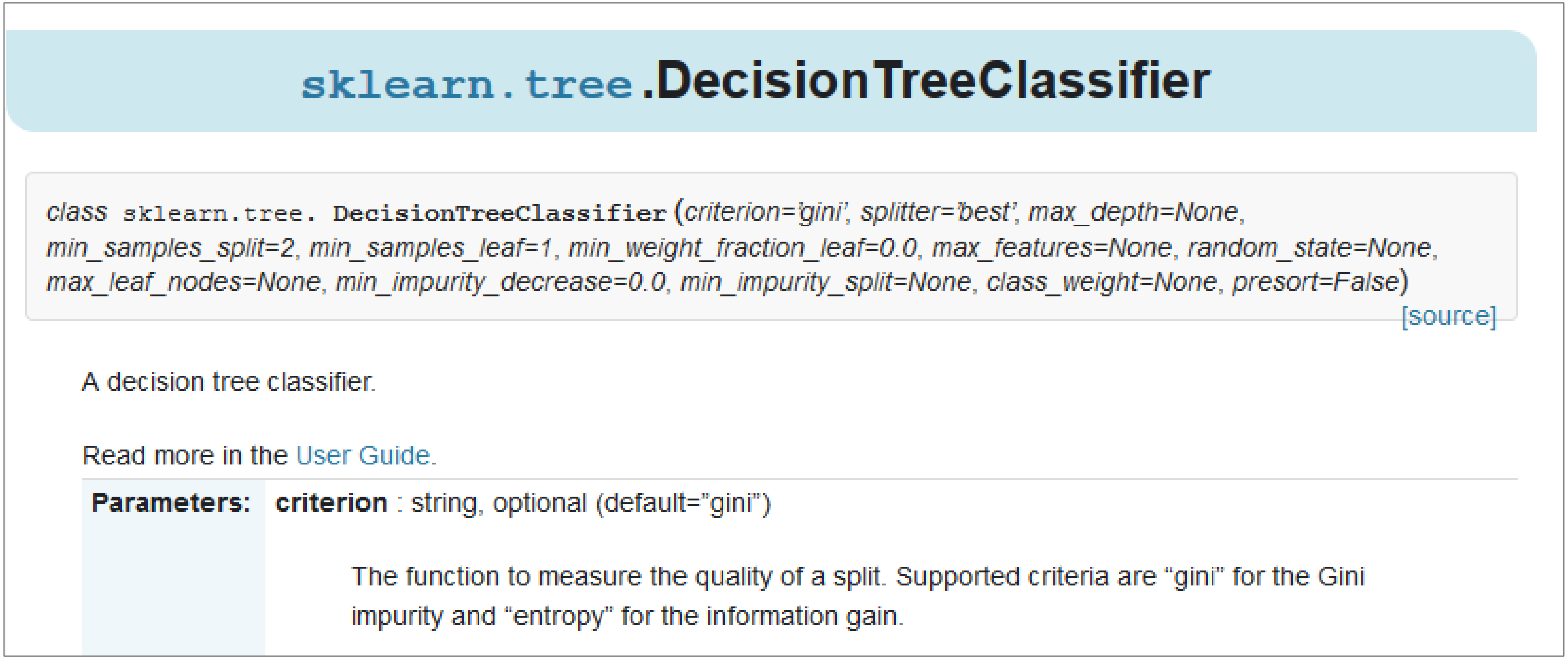
(left) Impurity functions plotted against the empirical probability of the positive class. From the bottom: the relative size of the minority class, $\min(\dot{p}, 1-\dot{p})$; the Gini index, $2\dot{p}(1-\dot{p})$; entropy, $-\dot{p}\log_2 \dot{p} - (1-\dot{p})\log_2(1-\dot{p})$ (divided by 2 so that it reaches its maximum in the same point as the others); and the (rescaled) square root of the Gini index, $\sqrt{\dot{p}(1-\dot{p})}$ – notice that this last function describes a semi-circle. **(right)**

Indicating the impurity of a single leaf D_j as $\mathrm{Imp}(D_j)$, the impurity of a set of mutually exclusive leaves $\{D_1, \ldots, D_l\}$ is defined as a weighted average

$$\mathrm{Imp}(\{D_1, \ldots, D_l\}) = \sum_{j=1}^{l} \frac{|D_j|}{|D|}\mathrm{Imp}(D_j)$$

Impurity

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html



sklearn.tree.DecisionTreeClassifier

class `sklearn.tree.` **`DecisionTreeClassifier`** (*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False*)

[source]

A decision tree classifier.

Read more in the User Guide.

Parameters:	**criterion** : string, optional (default="gini") The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

To find the next split:

Gini Index: $2p(1-p)$

- choose a feature and sort training data along it
- calculate impurity splitting between every point
- add Gini values weighted by number of samples:

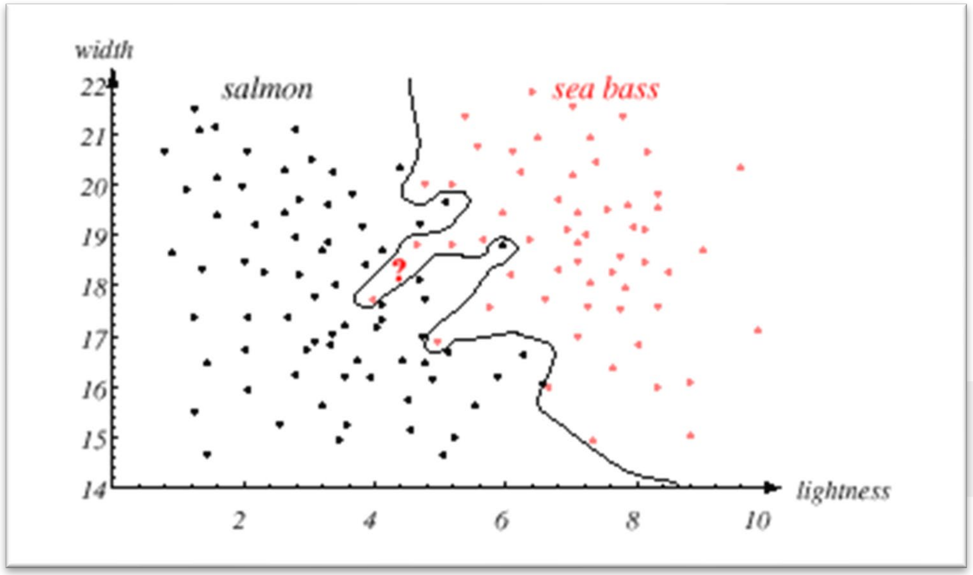
$$impurity = \frac{n_1 * Gini_1 + n_2 * Gini_2 + \cdots}{n_1 + n_2 + \cdots}$$

- Choose feature that gives biggest impurity decrease

Length = [3, 4, 5]	$[2+, 0-][1+, 3-][2+, 2-]$
Gills = [yes, no]	$[0+, 4-][5+, 1-]$
Beak = [yes, no]	$[5+, 3-][0+, 2-]$
Teeth = [many, few]	$[3+, 4-][2+, 1-]$

Length	$2/10 \cdot 2 \cdot (2/2 \cdot 0/2) + 4/10 \cdot 2 \cdot (1/4 \cdot 3/4) + 4/10 \cdot 2 \cdot (2/4 \cdot 2/4) = 0.35$
Gills	$4/10 \cdot 0 + 6/10 \cdot 2 \cdot (5/6 \cdot 1/6) = 0.17;$
Beak	$8/10 \cdot 2 \cdot (5/8 \cdot 3/8) + 2/10 \cdot 0 = 0.38;$
Teeth	$7/10 \cdot 2 \cdot (3/7 \cdot 4/7) + 3/10 \cdot 2 \cdot (2/3 \cdot 1/3) = 0.48.$

OVERFITTING



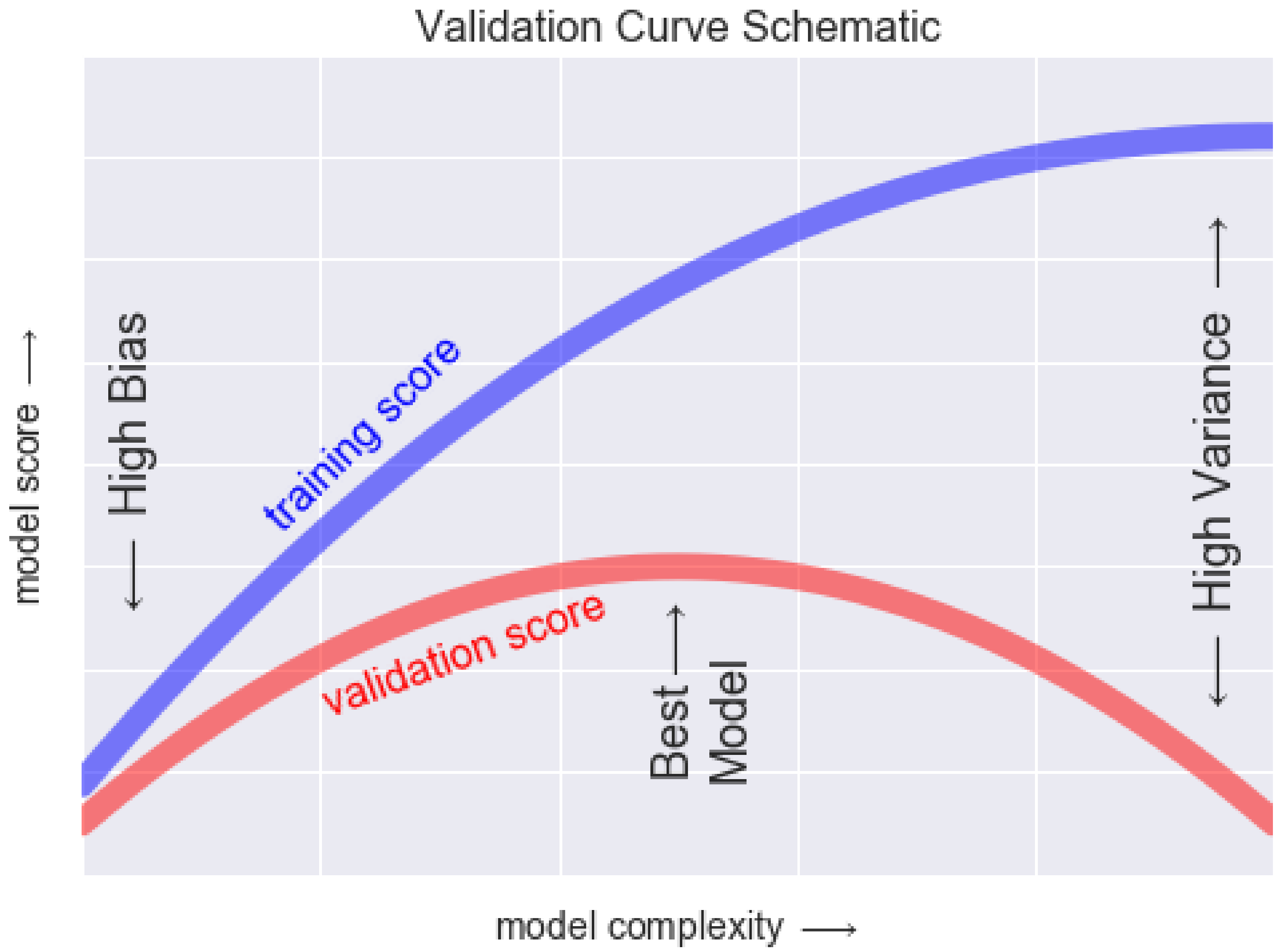
Overfitting

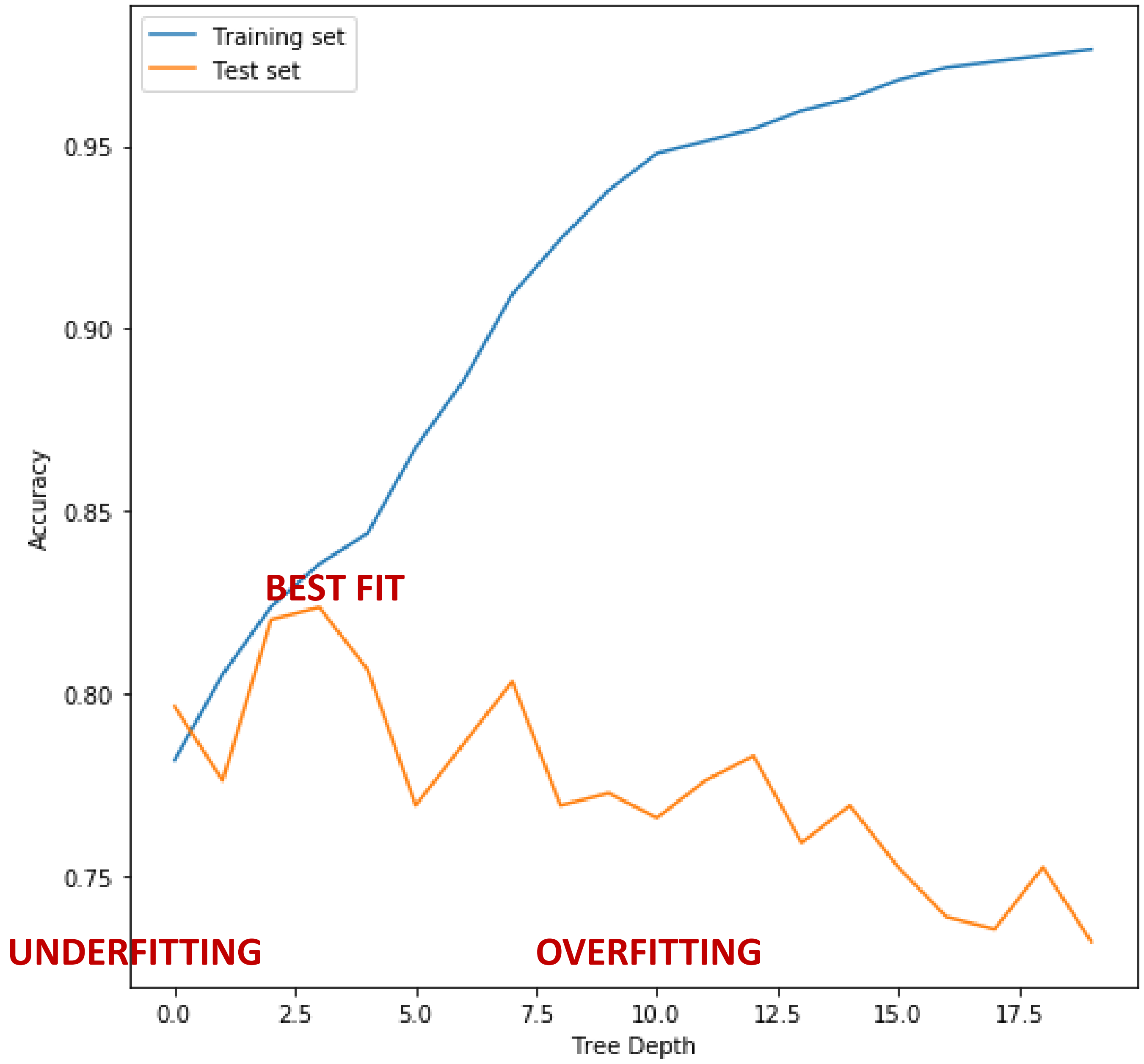
Imagine you are preparing for your *Machine Learning 101* exam. Helpfully, Professor Flach has made previous exam papers and their worked answers available online. You begin by trying to answer the questions from previous papers and comparing your answers with the model answers provided.

Unfortunately, you get carried away and spend all your time on memorising the model answers to all past questions. Now, if the upcoming exam completely consists of past questions, you are certain to do very well. But if the new exam asks different questions about the same material, you would be ill-prepared and get a much lower mark than with a more traditional preparation.

In this case, one could say that you were *overfitting* the past exam papers and that the knowledge gained didn't *generalise* to future exam questions.

http://nbviewer.jupyter.org/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.03-Hyperparameters-and-Model-Validation.ipynb



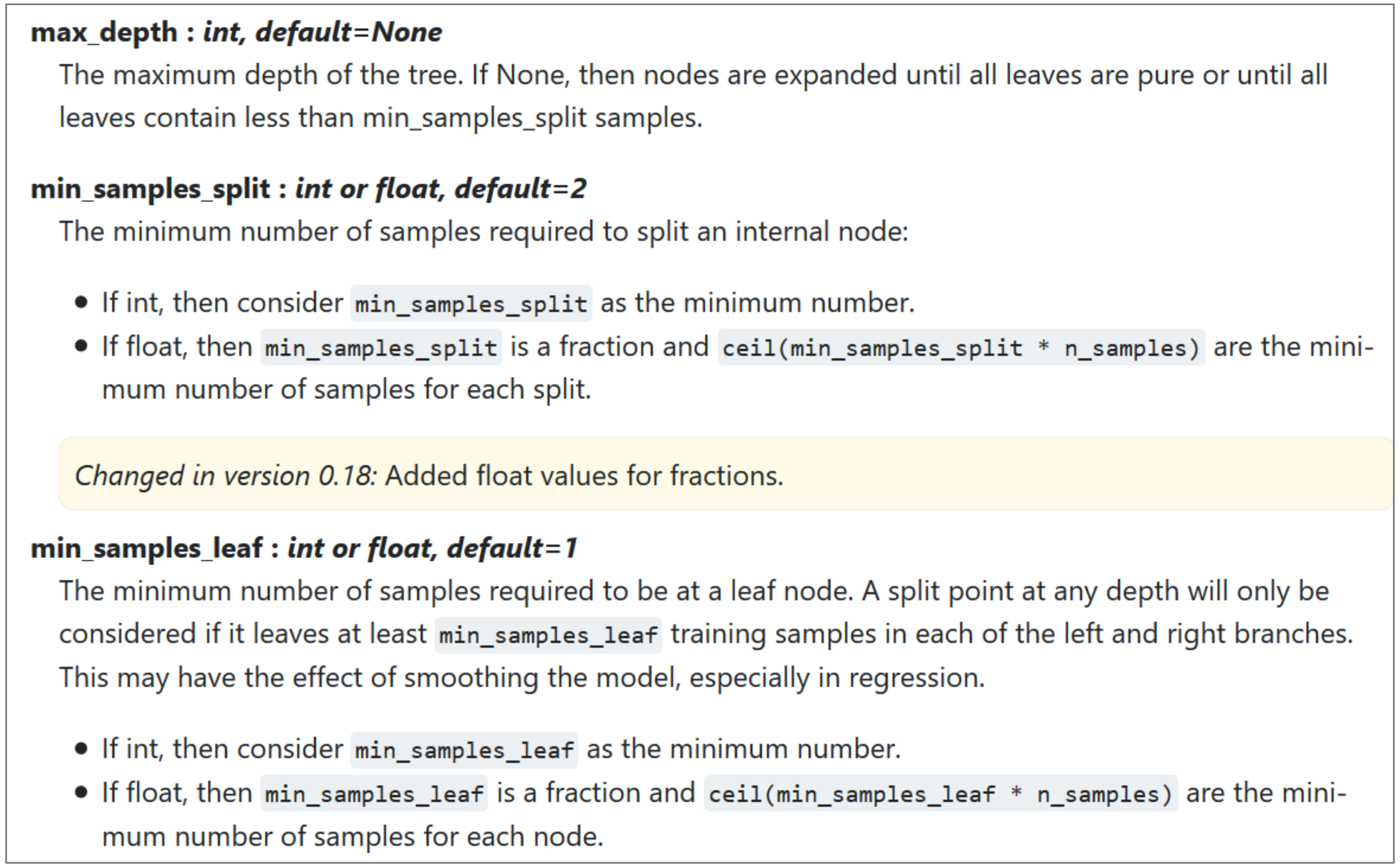

Training set
Test set
Accuracy
0.95
0.90
0.85
0.80
0.75
Tree Depth
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
BEST FIT
UNDERFITTING
OVERFITTING

Tree Size

- By default sklearn keeps growing the tree until no more splits are possible:

overfitting by default

- There are options to limit tree size:



max_depth : *int, default=None*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int or float, default=2*

The minimum number of samples required to split an internal node:

- If int, then consider `min_samples_split` as the minimum number.
- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Changed in version 0.18: Added float values for fractions.

min_samples_leaf : *int or float, default=1*

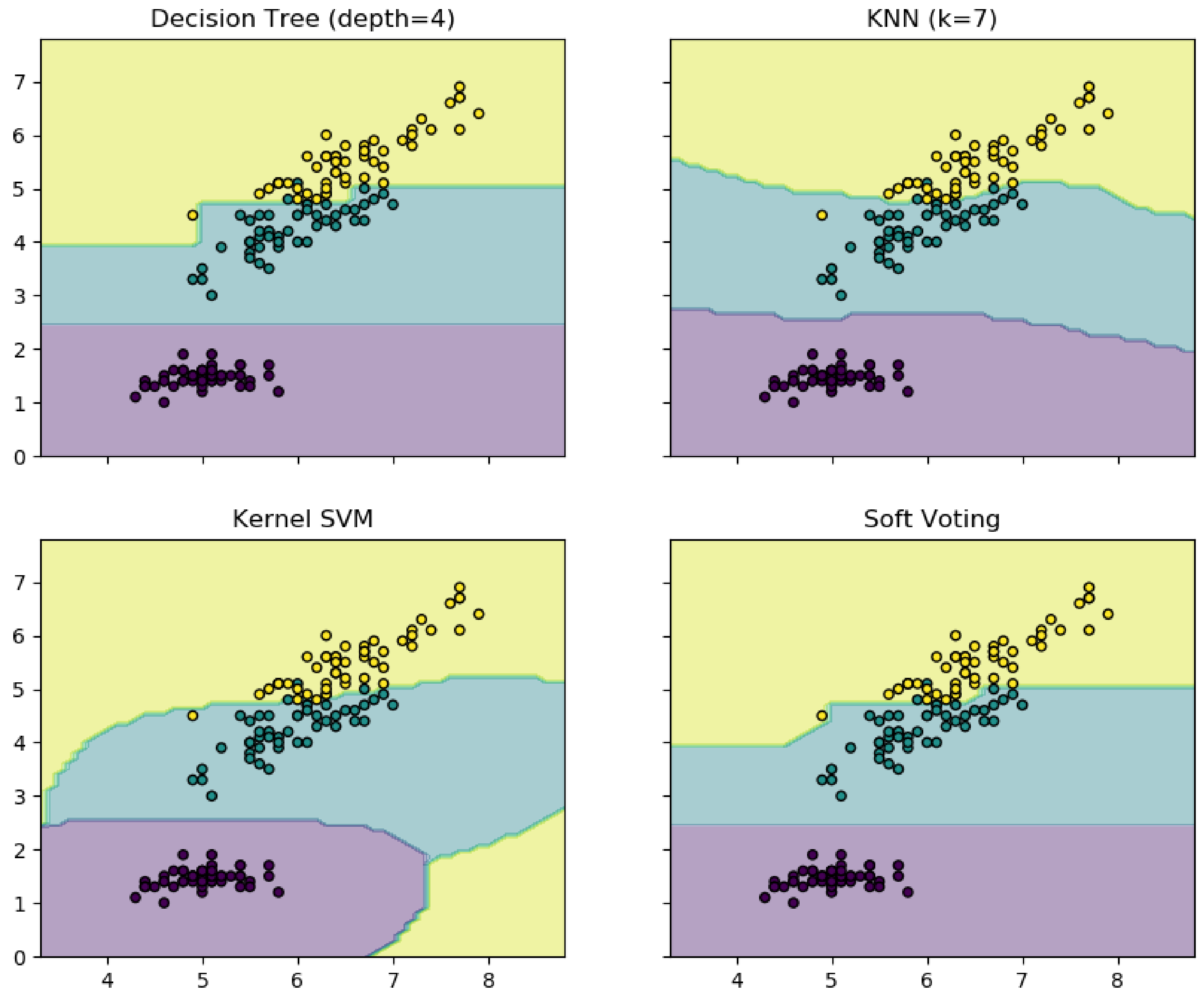
The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least `min_samples_leaf` training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

- If int, then consider `min_samples_leaf` as the minimum number.
- If float, then `min_samples_leaf` is a fraction and `ceil(min_samples_leaf * n_samples)` are the minimum number of samples for each node.

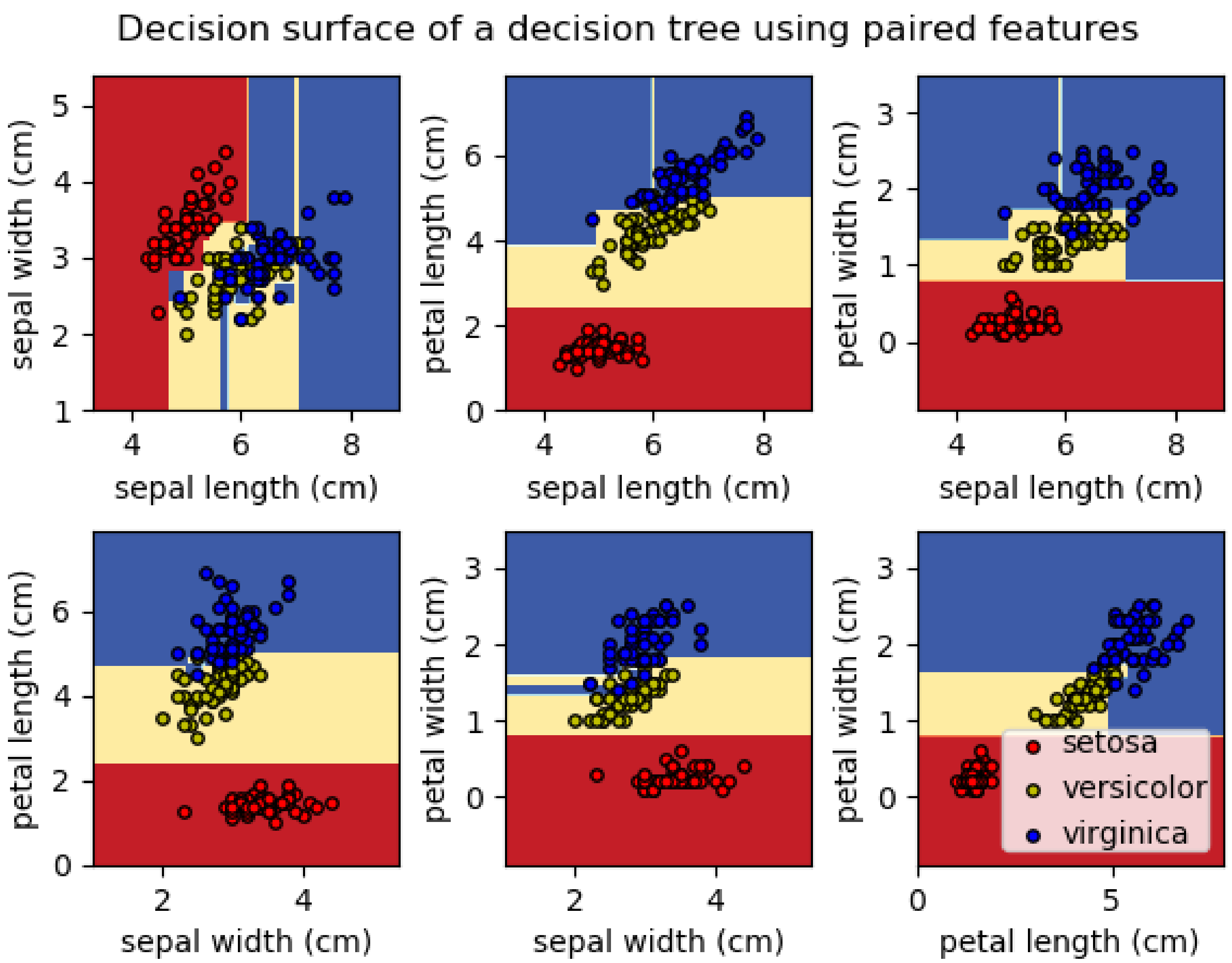
Prediction Probability

Classify a new sample by following the decision tree down to a leaf. Then classify the sample as the majority class in the leaf. We even get a probability!

However, if we didn't restrict the depth of our tree and there weren't ties in the training data, we may end up with everything being 100% certain http://scikit-learn.org/stable/auto_examples/ensemble/plot_voting_decision_regions.html#sphx-glr-auto-examples-ensemble-plot-voting-decision-regions-py



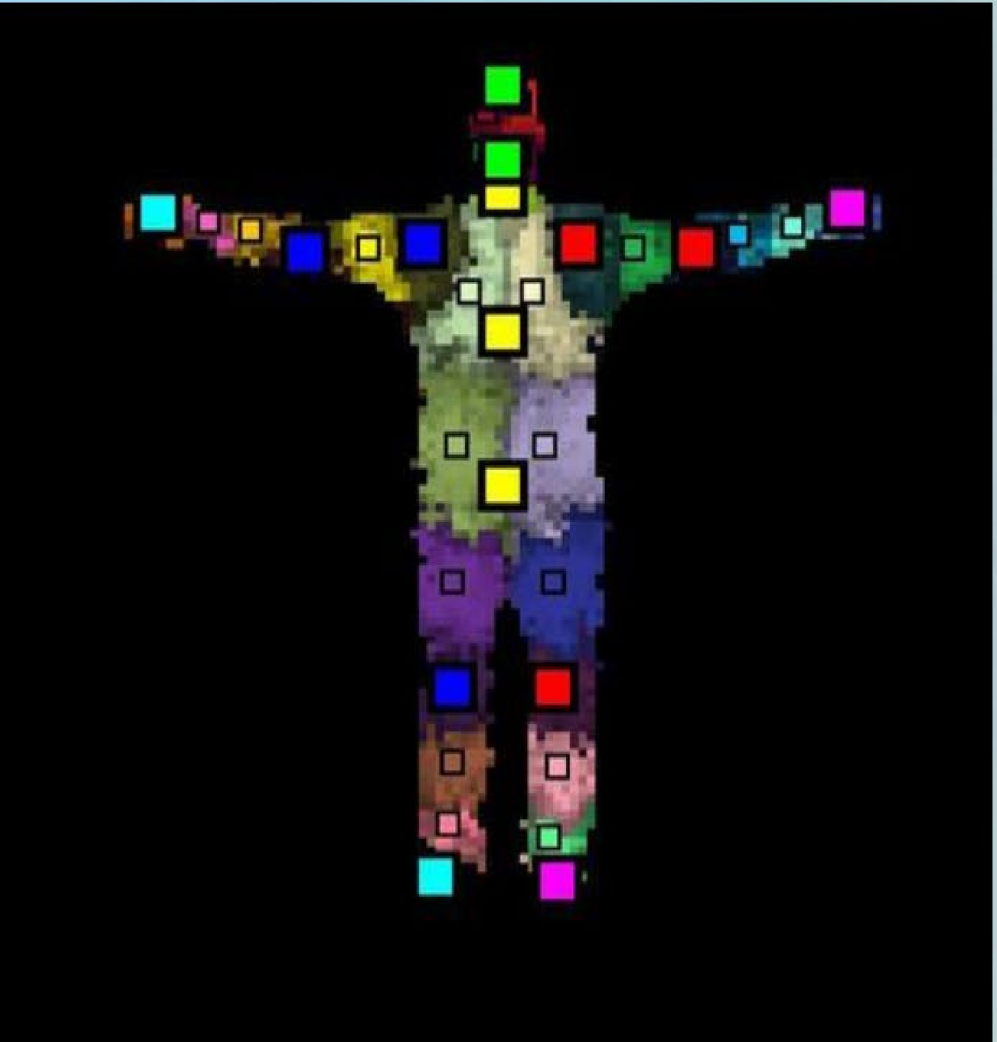
http://scikit-learn.org/stable/auto_examples/tree/plot_iris.html#sphx-glr-auto-examples-tree-plot-iris-py



APPLICATIONS

Kinect

How it works

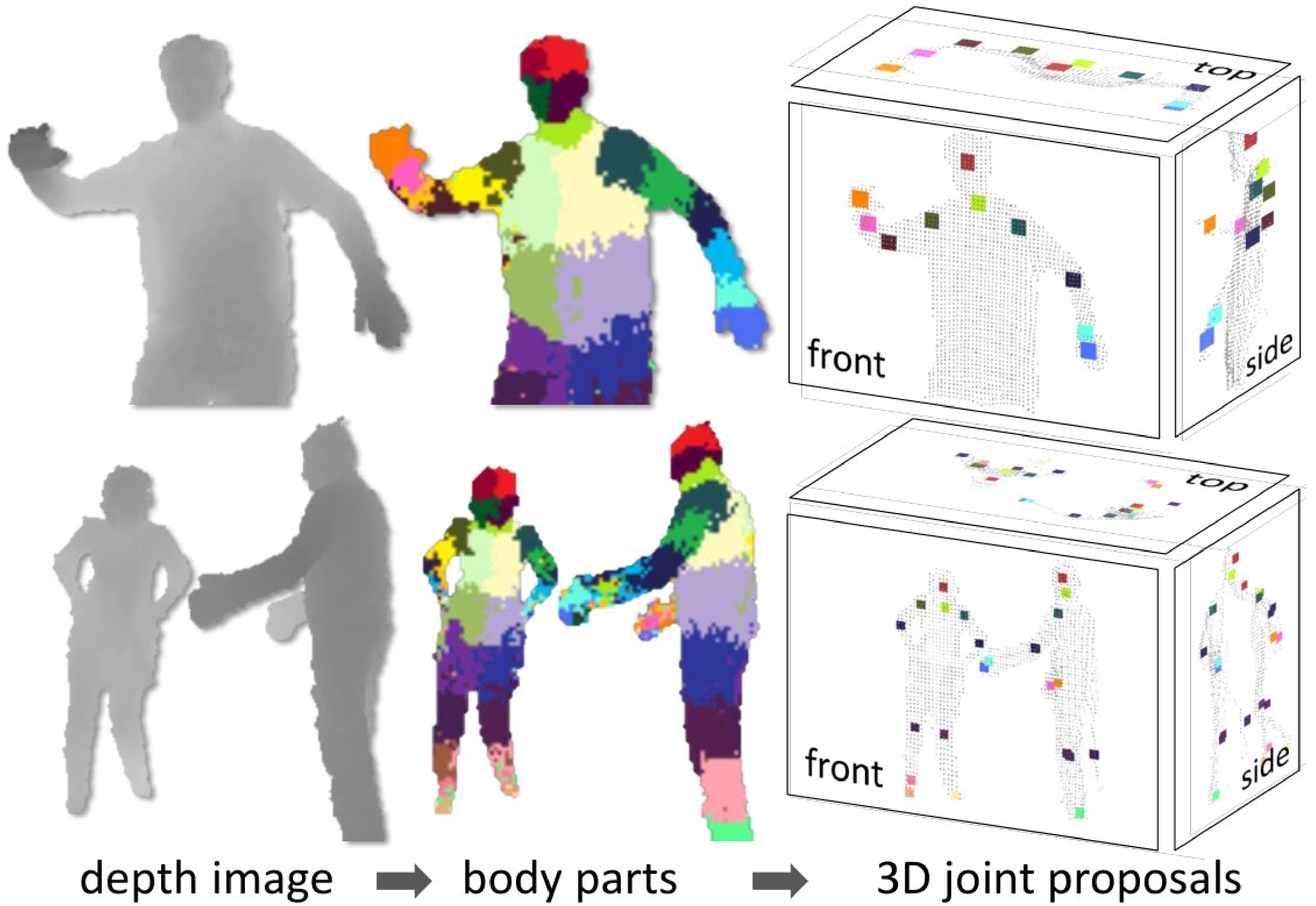


1. Classify each pixel's probability of being each of 32 body parts

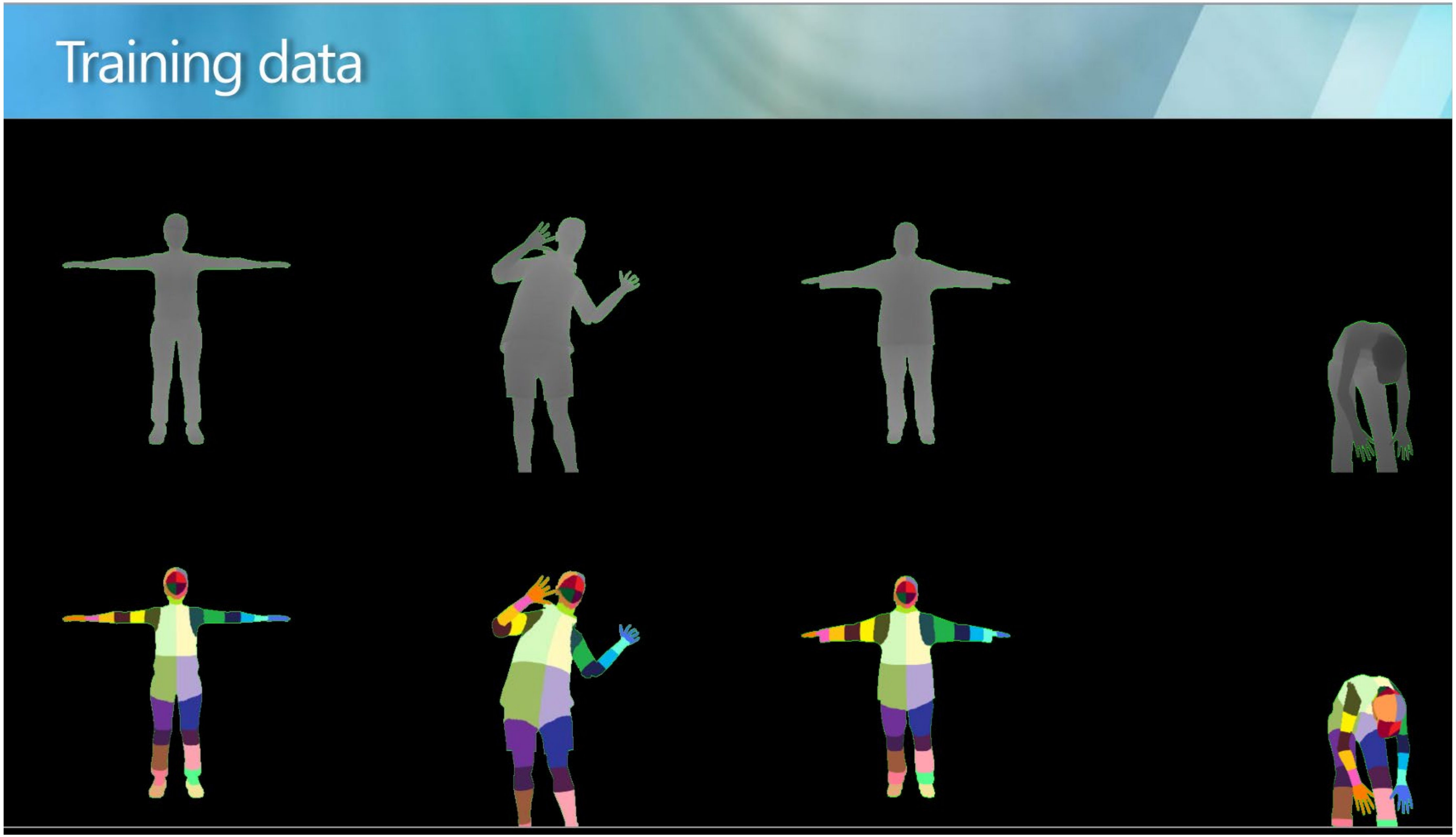
2. Determine probabilistic cluster of body configurations consistent with those parts

3. Present the most probable to the user

Kinect


top
front
side
top
front
side
depth image
body parts
3D joint proposals

Kinect


Training data

Kinect

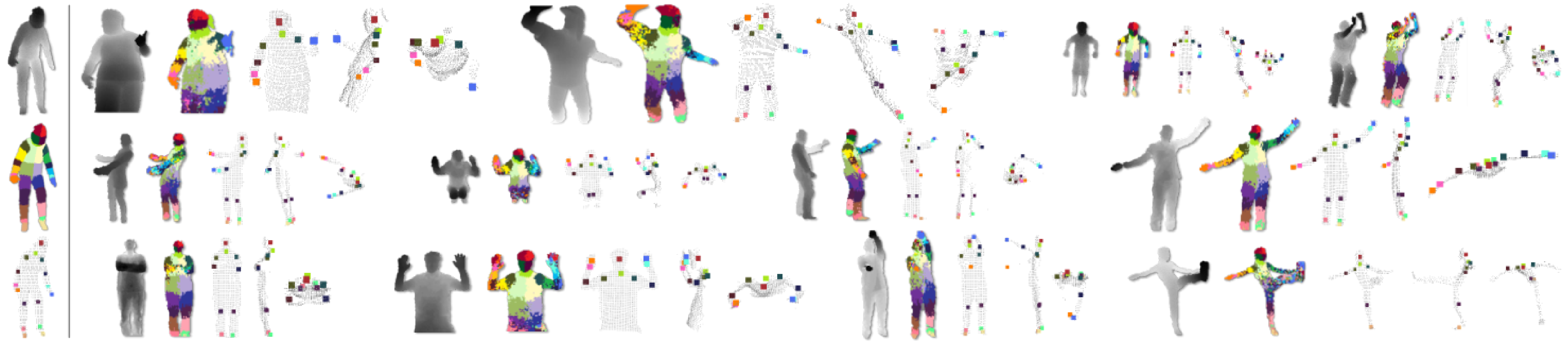


Figure 5. **Example inferences.** Synthetic (top row); real (middle); failure modes (bottom). Left column: ground truth for a neutral pose as a reference. In each example we see the depth image, the inferred most likely body part labels, and the joint proposals show as front, right, and top views (overlaid on a depth point cloud). Only the most confident proposal for each joint above a fixed, shared threshold is shown.

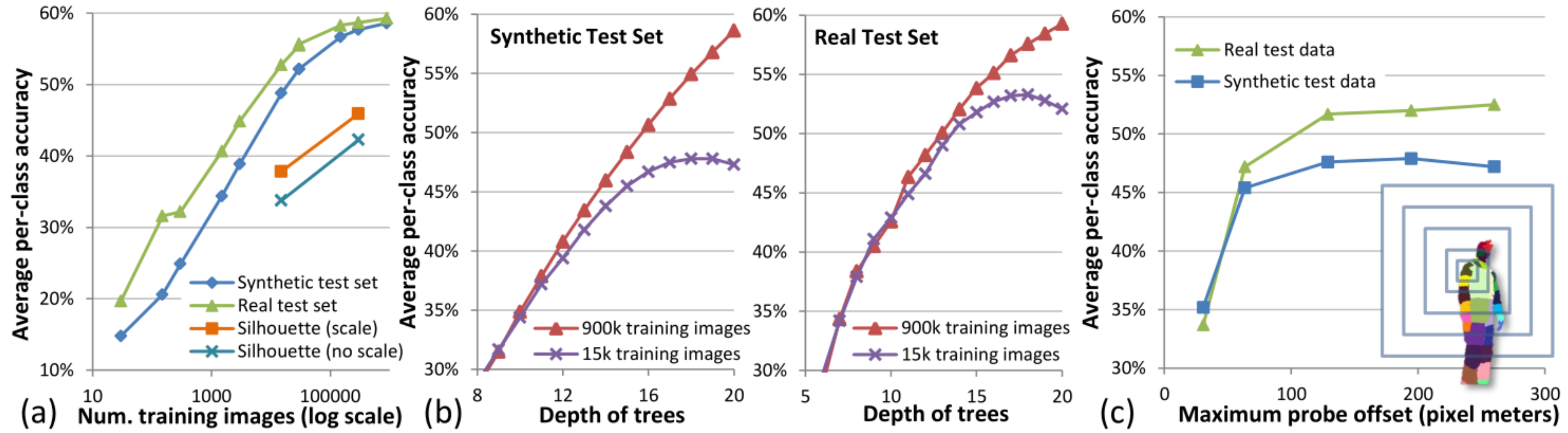


Figure 6. **Training parameters *vs*. classification accuracy.** (a) Number of training images. (b) Depth of trees. (c) Maximum probe offset.

Summary

Pros:

- extremely fast classification
- works with non-numeric data
- expressive

Cons:

- prone to overfitting, **LIMIT YOUR MAX DEPTH**
- “stair-step” decision boundary

Bonus Tricks

EXTENSIONS

optimizing the features (e.g. PCA) can be a big help with trees

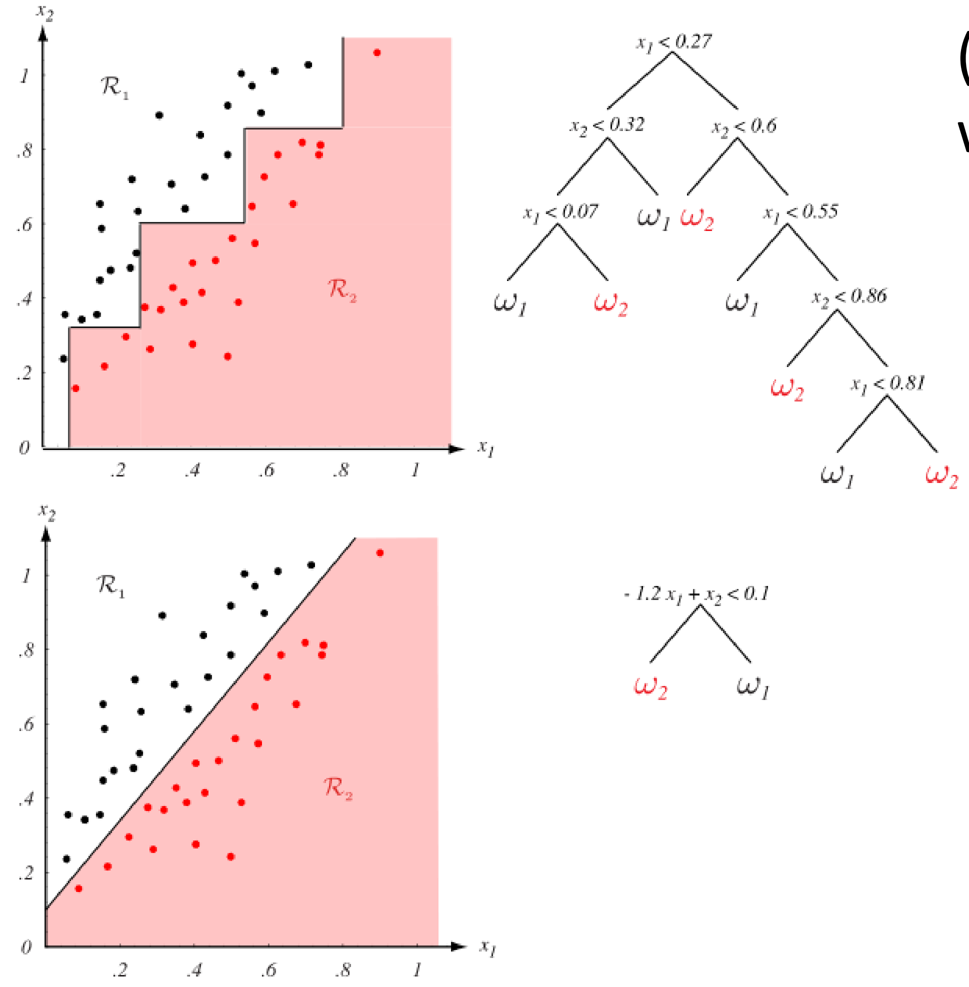


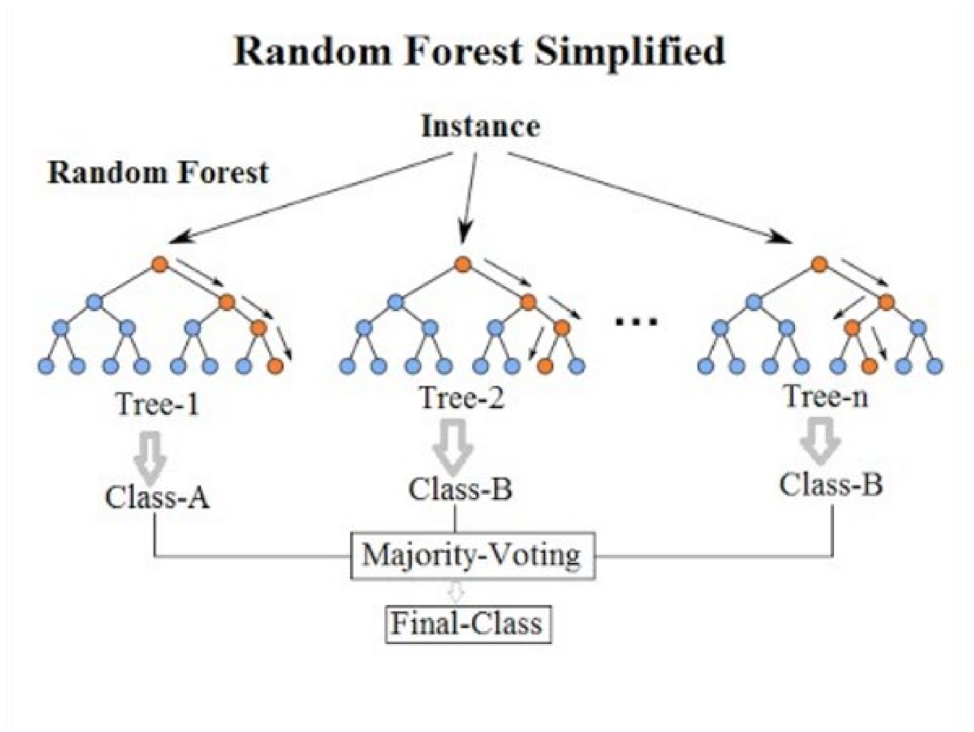
FIGURE 8.5. If the class of node decisions does not match the form of the training data, a very complicated decision tree will result, as shown at the top. Here decisions are parallel to the axes while in fact the data is better split by boundaries along another direction. If, however, "proper" decision forms are used (here, linear combinations of the features), the tree can be quite simple, as shown at the bottom. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

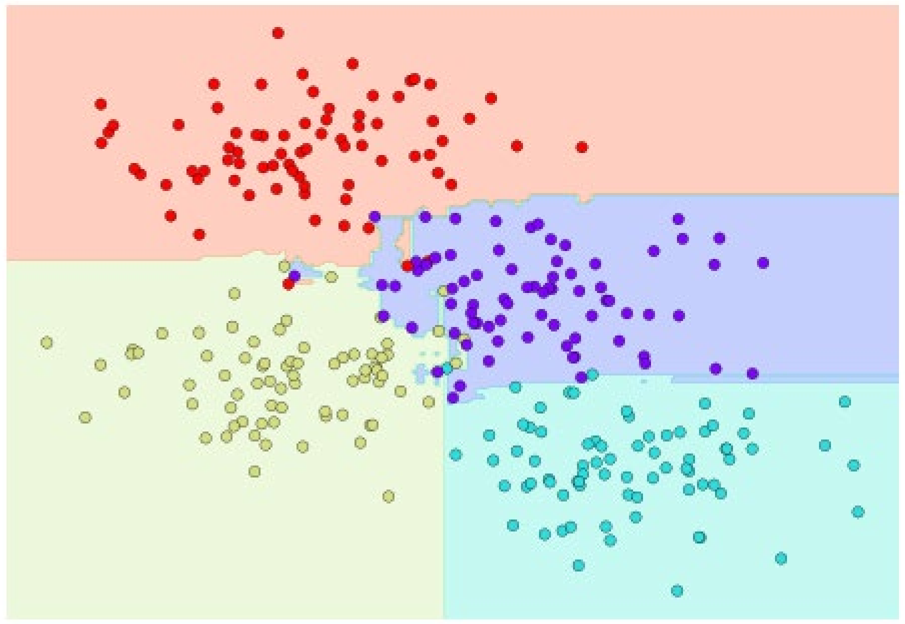
Random Forest

One common approach to minimize problems is to create a random forest:

- train lots of trees on different random samples of data
- each tree splits on random feature (instead of minimum impurity)
- the whole forest votes on classification
- **VERY** powerful technique that forms basis for many high-performing classifiers (e.g. XGBoost)

https://xgboost.readthedocs.io/en/stable/tutorials/model.html





http://nbviewer.jupyter.org/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.08-Random-Forests.ipynb