

K-Nearest Neighbors, Decision Trees and Random Forests

Machine Learning 1 — Lecture 10

23th May 2023

Robert Peharz

Institute of Theoretical Computer Science

Graz University of Technology

Algorithms for Supervised Learning

- So far, we discussed parametric models for supervised learning
 - linear models
 - linear models with non-linear features
 - neural networks
- Each of these models was specified via a parameter vector θ and we specified a training loss $\mathcal{L}_{train}(\theta)$ which we aimed to minimize w.r.t. θ
- There are, however, many other approaches to supervised learning

Algorithms for Supervised Learning

- “No free lunch theorem:” for any machine learning algorithm, there is some task where the algorithm does not perform best when provided with finitely many data
- Thus, getting best performance requires trying different learning algorithms
- Today, we are going to discuss
 - **K-nearest neighbours (KNN)**
 - **Decision trees**
 - **Random forests**

K-Nearest Neighbours

- Assume a training set $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(N)}, y^{(N)})\}$
- No training required: just dump the dataset in your memory
- Assume a test sample \mathbf{x}^* for which we don't know target y^*
- Let i be the index of the **nearest neighbour** of \mathbf{x}^* , i.e.

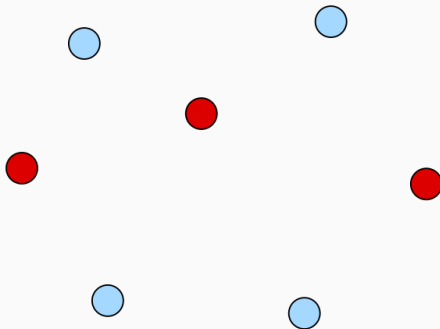
$$i = \arg \min_i \|\mathbf{x}^* - \mathbf{x}^{(i)}\|$$

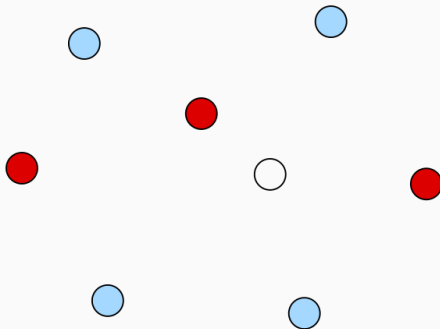
for some norm $\|\cdot\|$

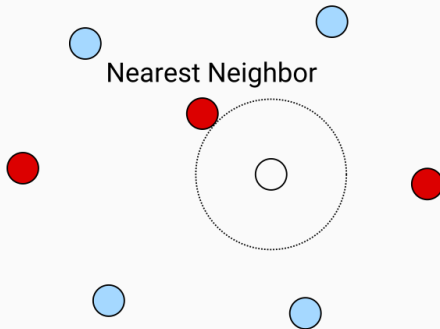
- The **nearest neighbour** predictor returns the target of the nearest neighbour

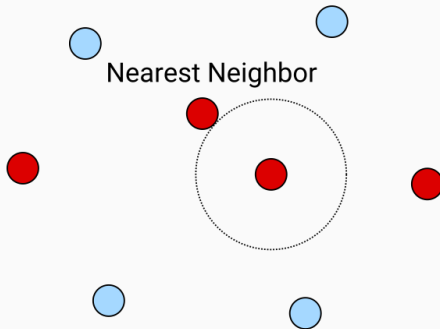
$$f_{NN}(\mathbf{x}^*) = y^{(i)}$$

- Note that this works for both classification and regression







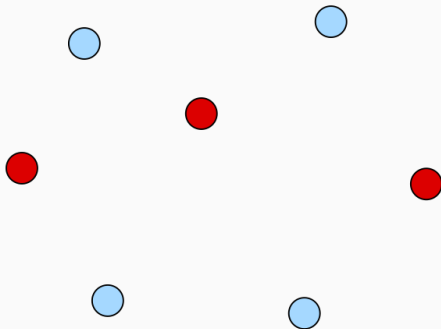


- Why should we restrict to only one neighbour?
- Instead, we can combine the targets of the **K nearest neighbours**
- Sort training points according to distance to \mathbf{x}^* :

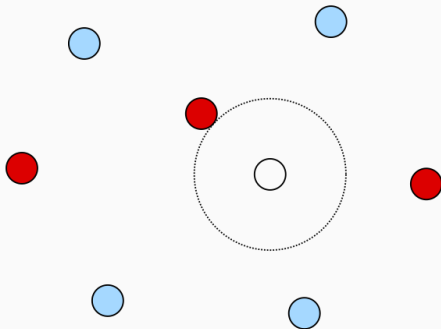
$$\|\mathbf{x}^{(i_1)} - \mathbf{x}^*\| \leq \|\mathbf{x}^{(i_2)} - \mathbf{x}^*\| \leq \dots \leq \|\mathbf{x}^{(i_K)} - \mathbf{x}^*\| \leq \dots \leq \|\mathbf{x}^{(i_N)} - \mathbf{x}^*\|$$

- **Classification:** Label test sample according to **majority vote** of K nearest neighbours. Break ties randomly.
- **Regression:** Average targets of K nearest neighbours

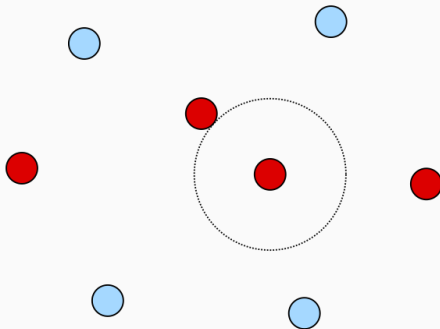
$$\hat{y} = \frac{1}{K} \sum_{k=1}^K y^{(i_k)}$$



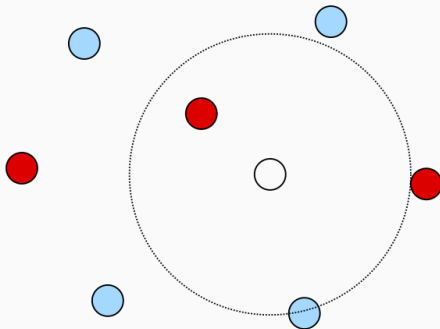
$k=1$



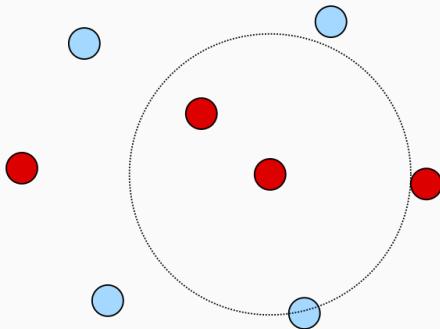
$k=1$



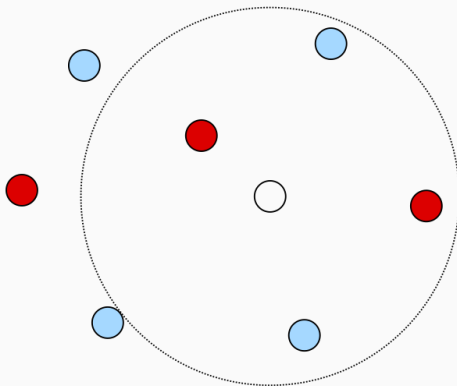
$k=3$



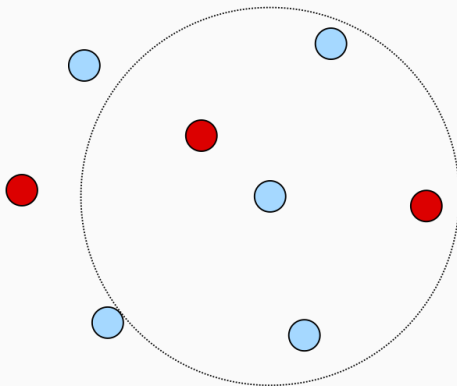
$k=3$



$k=5$



$k=5$



- Assume a classification setting, i.e. y is discrete
- Assume we would know the **true data-generating** distribution $p_{true}(\mathbf{x}, y)$ (**Recall Lecture 8**)
- Then we could compute $p_{true}(y | \mathbf{x}) = \frac{p_{true}(\mathbf{x}, y)}{p_{true}(\mathbf{x})} = \frac{p_{true}(\mathbf{x}, y)}{\sum_y p_{true}(\mathbf{x}, y)}$
- Then, a natural classification rule would be:

$$\hat{y} = \arg \max_y p_{true}(y | \mathbf{x})$$

- This rule is known as the **Bayes optimal classifier**
- It suffers the least **true classification error**, i.e.

$$\mathbb{E}_{\mathbf{x}, y \sim p_{true}} [\mathbb{1}(\hat{y} \neq y)]$$

is minimal among all possible classifiers.

Consistency of KNN

KNN is **simple**, but **theoretically sound**!

In particular, if

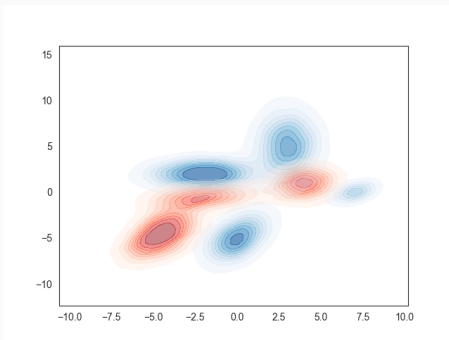
- number of training points $N \rightarrow \infty$
- number of neighbours $K \rightarrow \infty$
- N grows quicker than K , i.e. $\frac{K}{N} \rightarrow 0$

then KNN converges towards the **optimal predictor**.

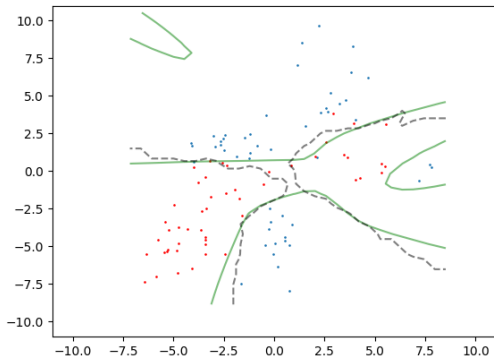
Specifically, the KNN classifier converges to the **Bayes optimal classifier**. A learning algorithm with this property is called **Bayes consistent**.

We can demonstrate Bayes consistency of KNN on a toy dataset.

Assume a binary classification problem and assume the following true data distribution $p_{true}(\mathbf{x}, y)$ (red and blue for the two classes):



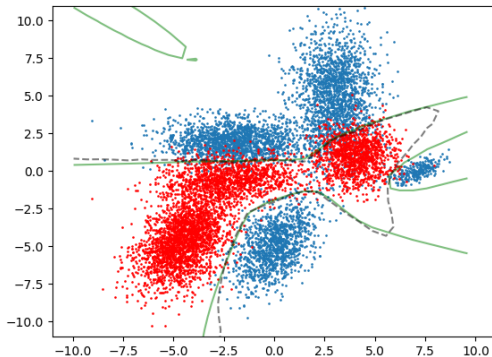
Thus, we can actually compute the Bayes optimal classifier!



Green line: Bayes optimal classifier (94.67% accuracy)

Dashed line: KNN $N_{train} = 100$ and $K = 9$ (90.26% accuracy)

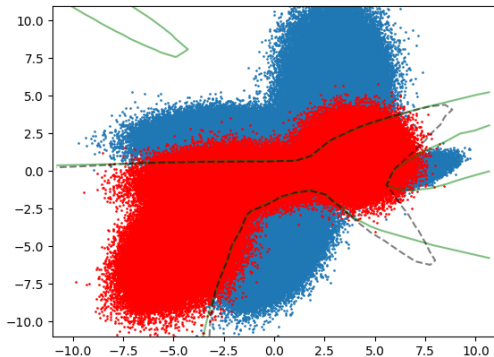
(accuracies computed on a 1 million test samples.)



Green line: Bayes optimal classifier (94.67% accuracy)

Dashed line: KNN $N_{train} = 10k$ and $K = 99$ (94.56% accuracy)

(accuracies computed on a 1 million test samples.)



Green line: Bayes optimal classifier (94.67% accuracy)

Dashed line: KNN $N_{train} = 1M$ and $K = 999$ (94.65% accuracy)

(accuracies computed on a 1 million test samples.)

Decision Trees





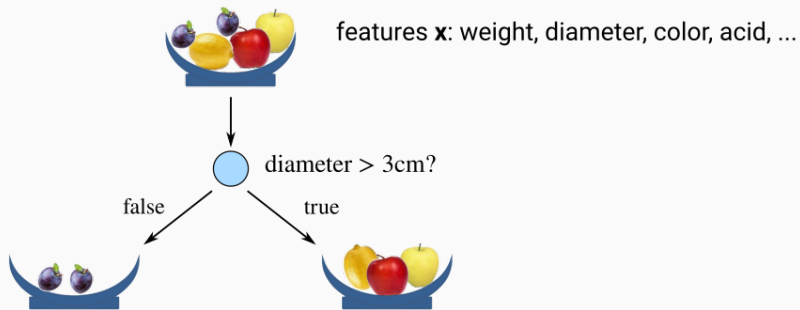
features x : weight, diameter, color, acid, ...

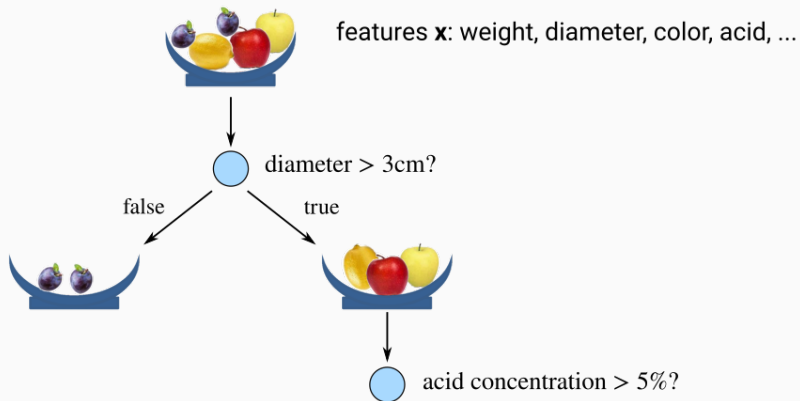


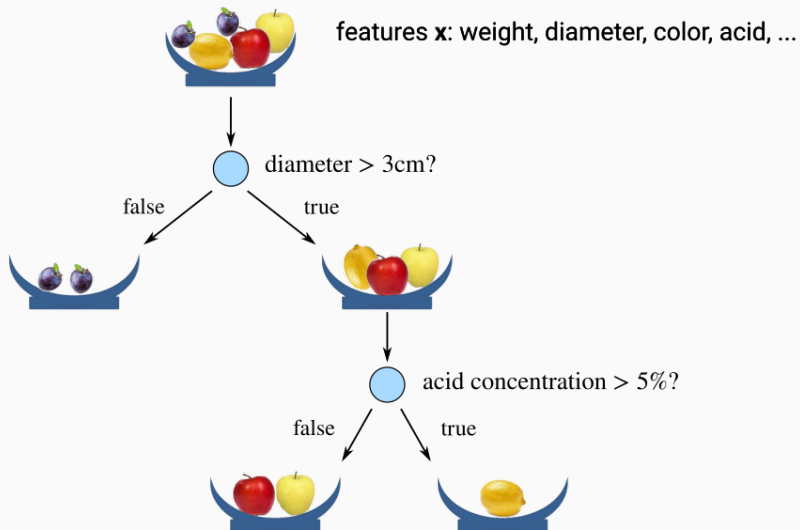
features x : weight, diameter, color, acid, ...



diameter $> 3\text{cm}$?







Given a set of features X_1, X_2, \dots, X_D (discrete or continuous) and a class variable Y , a Decision Tree is a **directed binary tree** with two types of nodes:

- **Decision Nodes** (internal nodes)

Decision nodes are associated with a feature X_i and a boolean function (decision, test)

$$f: X_i \mapsto \{true, false\}$$

Decision nodes have 2 children labeled *true* and *false*, which are selected by f .

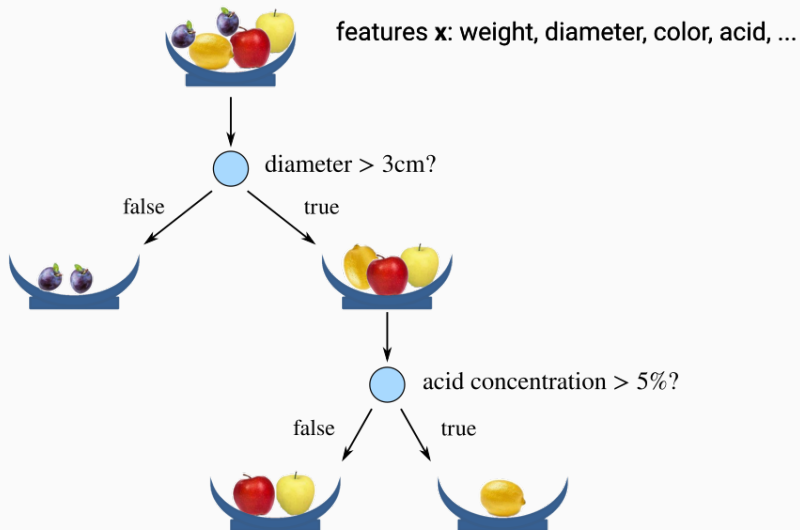
- **Prediction Nodes** (leaves)

Prediction nodes are associated with either

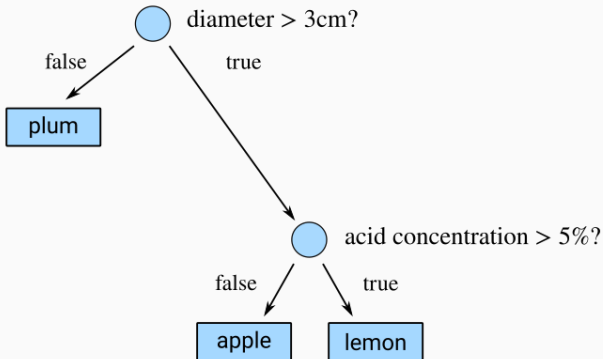
- a fixed label y
- a distribution over labels $p(Y)$

Classification using a Decision Tree

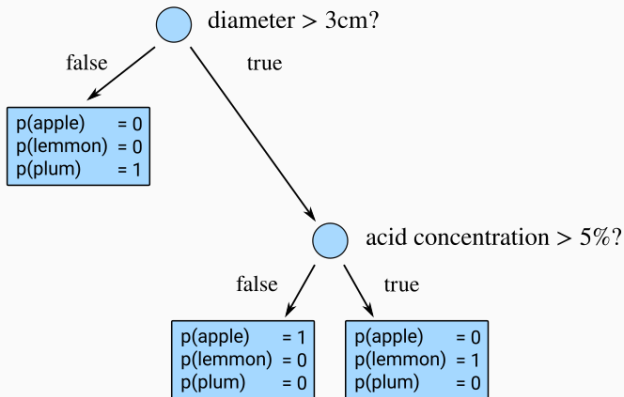
- Just parse the decisions from root to leaf
- Since the decision tree is a tree, there is a **unique** path from root to each leaf
- Leaf is selected, if **all** tests on this path are true
- Selected leaf is guaranteed to be unique, since decision nodes select exactly one child
- Use the predictor in the leaf for classification:
 - a fixed label y
 - a distribution over labels $p(Y)$
- Decision trees can also be used for regression (not discussed)



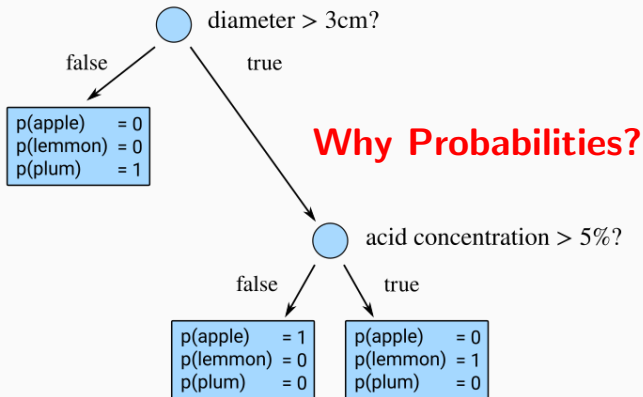
features x : weight, diameter, color, acid, ...

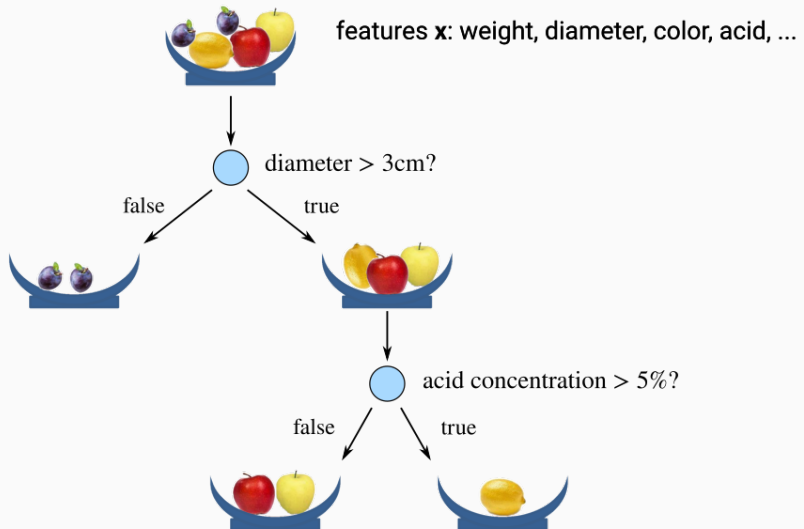


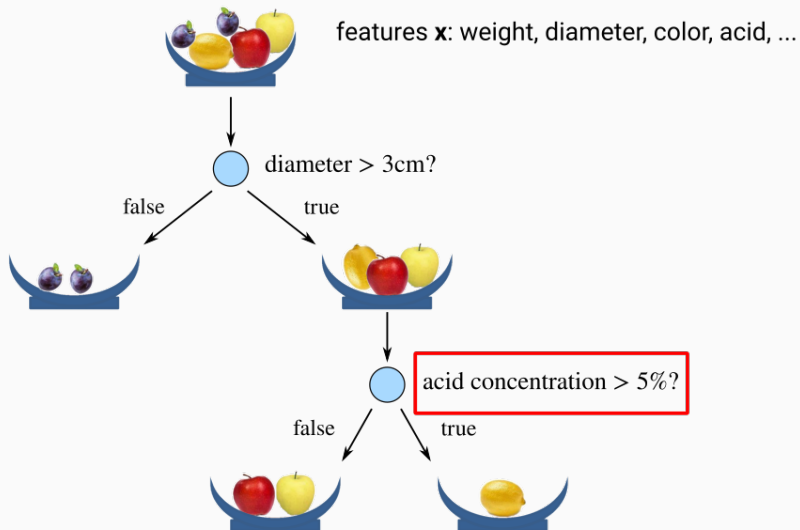
features x : weight, diameter, color, acid, ...

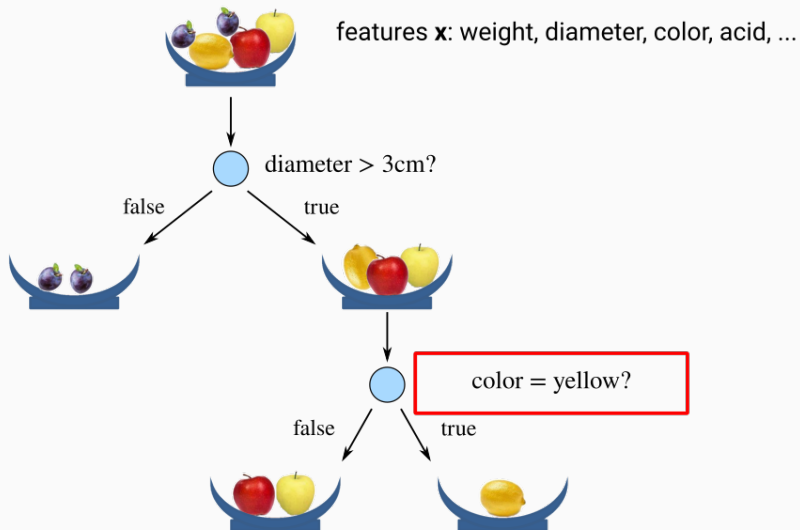


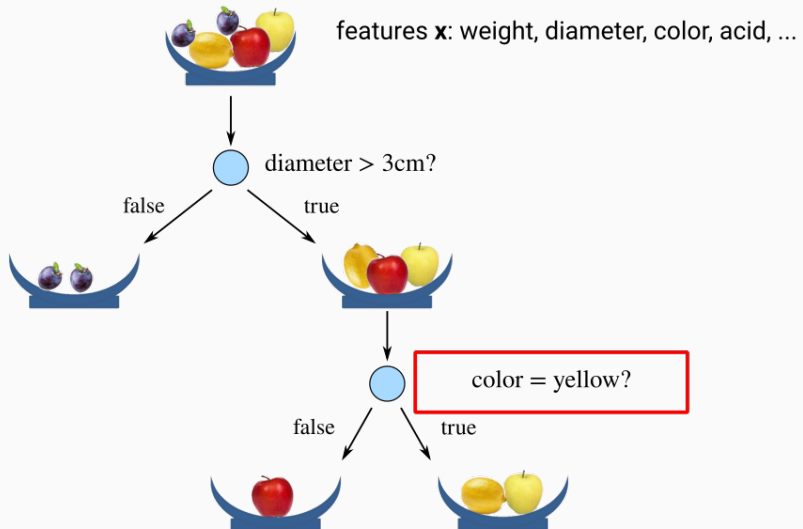
features x : weight, diameter, color, acid, ...



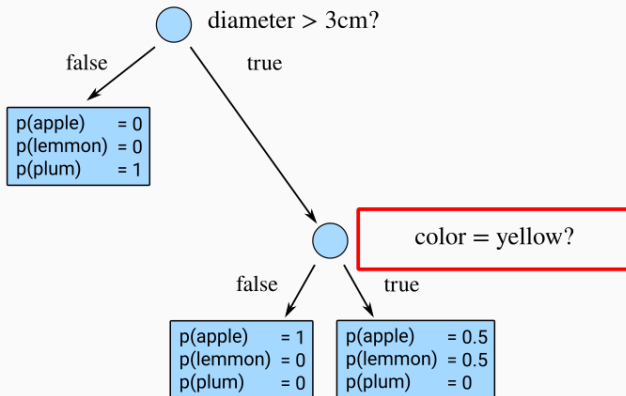




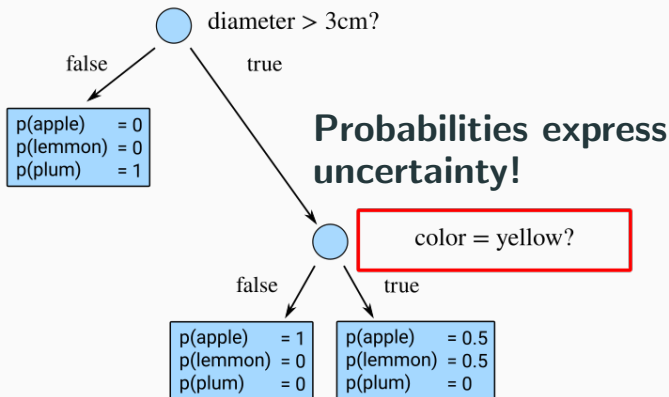




features x : weight, diameter, color, acid, ...

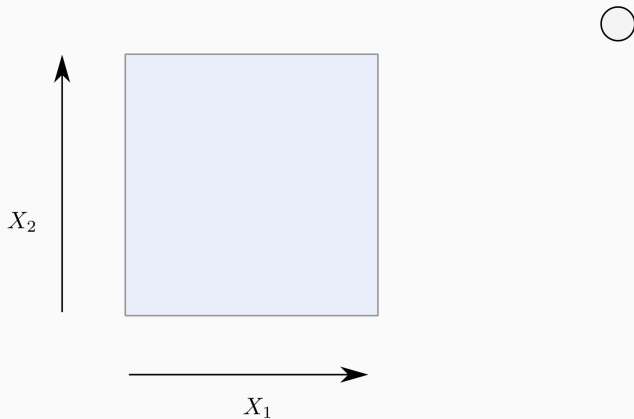


features x : weight, diameter, color, acid, ...



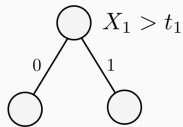
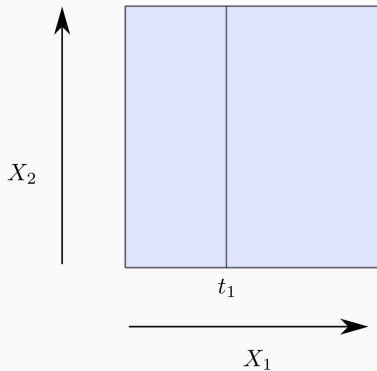
Partition of the Input Space

Decision trees represent a partition of the input space:



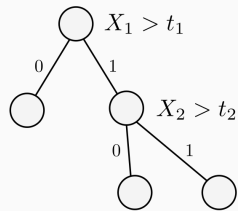
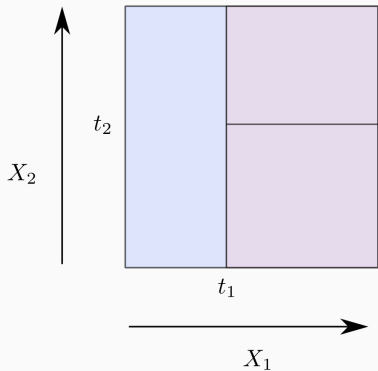
Partition of the Input Space

Decision trees represent a partition of the input space:



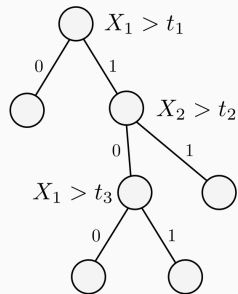
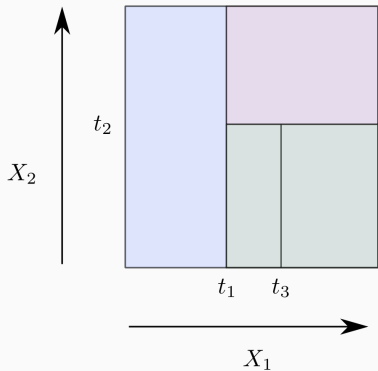
Partition of the Input Space

Decision trees represent a partition of the input space:



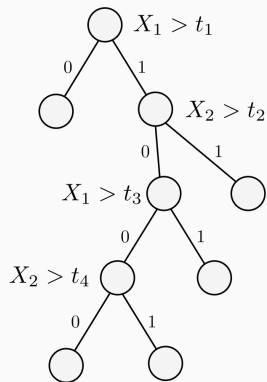
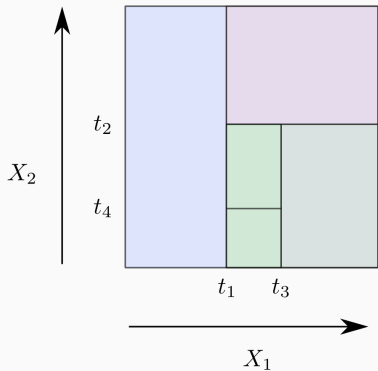
Partition of the Input Space

Decision trees represent a partition of the input space:



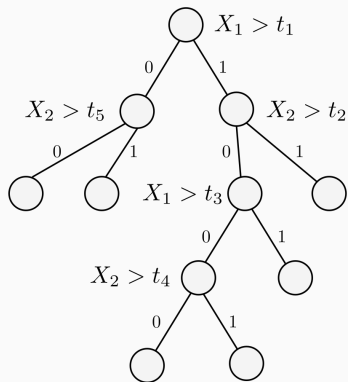
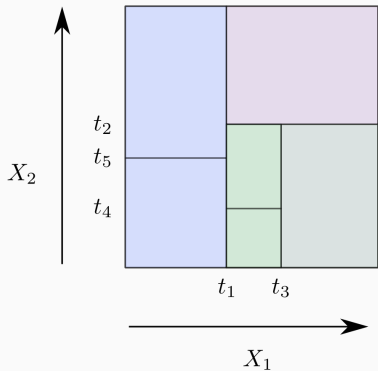
Partition of the Input Space

Decision trees represent a partition of the input space:



Partition of the Input Space

Decision trees represent a partition of the input space:



- **CART** (Classification and Regression Tree) [Breiman'84]
- ID3 [Quinlan'86]
- C4.5 [Quinlan'93]
- ...

CART

```
1 input: Data  $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$ 
2 result: Decision Tree
3 Initialize single leaf  $L$ , assign all data points  $\mathbf{x}^{(i)}$  to  $L$ , call split( $L$ )
4 split( $L$ ):
5   if stopping criterion is true for  $L$  then
6     |   Learn  $p(Y)$  from class proportions of samples assigned to  $L$ 
7     |   return
8   end
9   for considered decisions  $X_d, f$  do
10    |   compute  $\text{cost}(X_d, f)$ 
11  end
12  Let  $X_{\min}, f_{\min}$  be the decision with minimal cost
13  Apply  $f_{\min}(X_{\min})$  to  $L$ , yielding new leaves  $L_0$  and  $L_1$ 
14  Assign  $\mathbf{x}^{(i)}$  with  $f(\mathbf{x}^{(i)}) = \text{false}$  to  $L_0$ , call split( $L_0$ )
15  Assign  $\mathbf{x}^{(i)}$  with  $f(\mathbf{x}^{(i)}) = \text{true}$  to  $L_1$ , call split( $L_1$ )
```

CART

```
1 input: Data  $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$ 
2 result: Decision Tree
3 Initialize single leaf  $L$ , assign all data points  $\mathbf{x}^{(i)}$  to  $L$ , call split( $L$ )
4 split( $L$ ):
5   if stopping criterion is true for  $L$  then
6     |   Learn  $p(Y)$  from class proportions of samples assigned to  $L$ 
7     |   return
8   end
9   for considered decisions  $X_d, f$  do
10    |   compute  $\text{cost}(X_d, f)$ 
11  end
12  Let  $X_{min}, f_{min}$  be the decision with minimal cost
13  Apply  $f_{min}(X_{min})$  to  $L$ , yielding new leaves  $L_0$  and  $L_1$ 
14  Assign  $\mathbf{x}^{(i)}$  with  $f(\mathbf{x}^{(i)}) = \text{false}$  to  $L_0$ , call split( $L_0$ )
15  Assign  $\mathbf{x}^{(i)}$  with  $f(\mathbf{x}^{(i)}) = \text{true}$  to  $L_1$ , call split( $L_1$ )
```

Possible Decisions

input : Data $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$

Result: Decision Tree

- 1 Initialize single leaf L , assign all data points $\mathbf{x}^{(i)}$ to L , call **split**(L)
- 2 **split**(L):
- 3 **if** *stopping criterion is true for L* **then**
- 4 Learn $p(Y)$ from class proportions of samples assigned to L
- 5 **return**
- 6 **end**
- 7 **for** **considered decisions** X_d, f **do**
- 8 compute $\text{cost}(X_d, f)$
- 9 **end**
- 10 Let X_{min}, f_{min} be the decision with minimal cost
- 11 Apply X_{min}, f_{min} to L , yielding new leaves L_0 and L_1
- 12 Assign $\mathbf{x}^{(i)}$ with $f(\mathbf{x}^{(i)}) = \text{false}$ to L_0 , call **split**(L_0)
- 13 Assign $\mathbf{x}^{(i)}$ with $f(\mathbf{x}^{(i)}) = \text{true}$ to L_1 , call **split**(L_1)

Which Decisions to Consider?

Continuous Feature X_d

- **thresholds** $X_d > t$
- sort data values for X_d
- consider thresholds **halfway between consecutive values**
- only finitely many decisions to consider
- e.g., for values $X_d \in [0.1, 1.3, 3.1415]$, we consider thresholds $\frac{0.1+1.3}{2} = 0.7$ and $\frac{1.3+3.1415}{2} = 2.22$

Discrete Feature X_d

- **one-vs-all** $\{1, 2, 3, 4\} \rightarrow \{1, 2, 4\}, \{3\}$
- **complete split** (not a binary tree)
 $\{1, 2, 3, 4\} \rightarrow \{1\}, \{2\}, \{3\}, \{4\}$

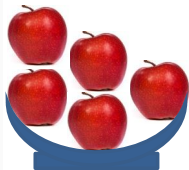
Cost

input : Data $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$

Result: Decision Tree

- 1 Initialize single leaf L , assign all data points $\mathbf{x}^{(i)}$ to L , call $\text{split}(L)$
- 2 $\text{split}(L)$:
- 3 **if** *stopping criterion is true for L* **then**
- 4 Learn $p(Y)$ from class proportions of samples assigned to L
- 5 **return**
- 6 **end**
- 7 **for** *considered decisions X_d, f* **do**
- 8 compute $\text{cost}(X_d, f)$
- 9 **end**
- 10 Let X_{\min}, f_{\min} be the decision with minimal **cost**
- 11 Apply X_{\min}, f_{\min} to L , yielding new leaves L_0 and L_1
- 12 Assign $\mathbf{x}^{(i)}$ with $f(\mathbf{x}^{(i)}) = \text{false}$ to L_0 , call **split**(L_0)
- 13 Assign $\mathbf{x}^{(i)}$ with $f(\mathbf{x}^{(i)}) = \text{true}$ to L_1 , call **split**(L_1)

Low impurity



High impurity



- “Pure leaves are good”
- Let N be the number of data points at leaf L
- For a particular **candidate decision** let N_0 , N_1 be the number of data points at new leaves L_0 and L_1 , respectively
- $\text{cost} = \frac{N_0}{N} \text{impurity}(L_0) + \frac{N_1}{N} \text{impurity}(L_1)$
- Note that $\frac{N_0}{N} + \frac{N_1}{N} = 1$ (weighted average)

Impurity Measures

Let $p(Y)$ be the empirical frequencies of class labels, e.g.



$$p(\text{plum}) = \frac{5}{8} = 0.625$$

$$p(\text{lemon}) = \frac{3}{8} = 0.375$$

- **Gini Impurity**

$$G = 1 - \sum_y p(y)^2$$

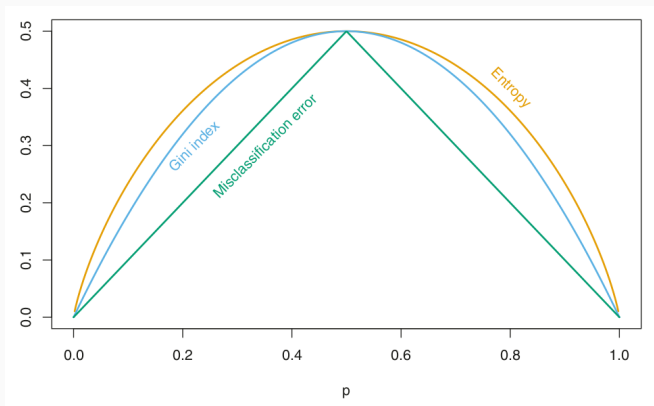
- **Entropy**

$$E = - \sum_y p(y) \log(p(y))$$

- **Self-Classification Error**

$$C = 1 - \max_y p(y)$$

Impurity measures for binary classification, as a function of $p(y = 0)$ (and $p(y = 1)$ due to symmetry). Entropy has been scaled to pass through $(0.5, 0.5)$.



Cost

- Let N be the number of data points at leaf L
- Let N_0, N_1 be the number of data points at new leaves L_0 and L_1 , respectively.
- $\text{cost} = \frac{N_0}{N} \text{impurity}(L_0) + \frac{N_1}{N} \text{impurity}(L_1)$
- Where impurity might be

- **Gini Impurity**

$$G = 1 - \sum_y p(y)^2$$

- **Entropy**

$$E = - \sum_y p(y) \log(p(y))$$

- **Self-Classification Error**

$$C = 1 - \max_y p(y)$$

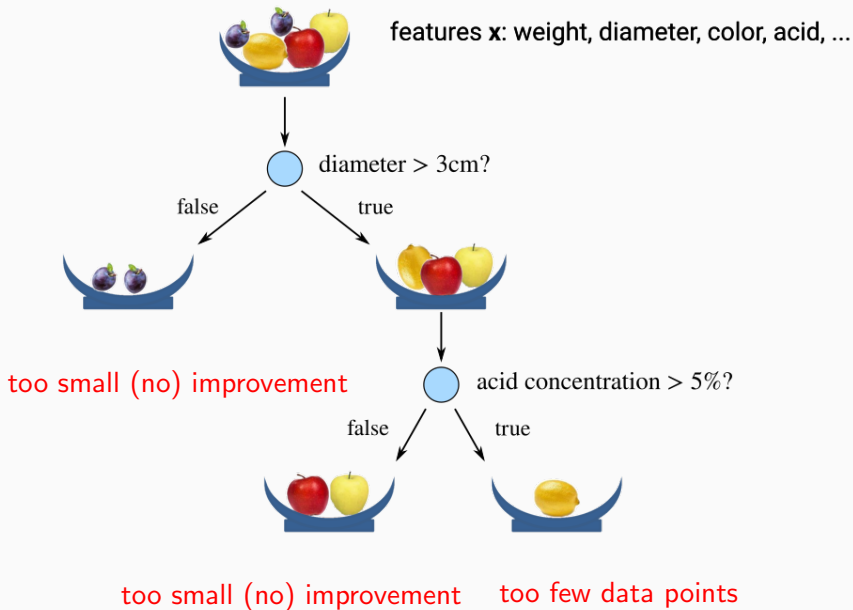
Stopping Criterion

input : Data $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$

Result: Decision Tree

- 1 Initialize single leaf L , assign all data points $\mathbf{x}^{(i)}$ to L , call $\text{split}(L)$
- 2 $\text{split}(L)$:
- 3 **if** **stopping criterion** *is true for* L **then**
- 4 Learn $p(Y)$ from class proportions of samples assigned to L
- 5 **return**
- 6 **end**
- 7 **for** *considered decisions* X_d, f **do**
- 8 compute $\text{cost}(X_d, f)$
- 9 **end**
- 10 Let X_{\min}, f_{\min} be the decision with minimal cost
- 11 Apply X_{\min}, f_{\min} to L , yielding new leaves L_0 and L_1
- 12 Assign $\mathbf{x}^{(i)}$ with $f(\mathbf{x}^{(i)}) = \text{false}$ to L_0 , call **split**(L_0)
- 13 Assign $\mathbf{x}^{(i)}$ with $f(\mathbf{x}^{(i)}) = \text{true}$ to L_1 , call **split**(L_1)

Stopping Criterion



Stopping Criterion

- Threshold determining too small improvement of cost
- Another threshold for determining too few data points
- Furthermore, one might use a maximal depth of the decision tree
- These thresholds are **hyper-parameters** of the CART algorithm

Random Forests

- Decision trees: well interpretable models, but don't perform too well in practice
- **Random Forests**: **ensemble** of decision trees
- Output of random forest is computed by **aggregating** the outputs of the individual decision trees
 - Majority vote for classification
 - Averaging for regression
- A theoretical underpinning for ensembles is the **bias-variance** trade-off

Bias-Variance Trade-Off

- Let \mathcal{H} be our considered **hypothesis class**
- Recall from Lecture 8 that the ideal goal of supervised learning is to find an optimal predictor f^* minimizing the **true loss**:

$$f^* = \arg \min_{f \in \mathcal{H}} \mathcal{L}_{true}(f) = \mathbb{E}_{p_{true}}[\ell(f(\mathbf{x}), y)]$$

- However, we only have a finite training set of N samples drawn from p_{true}
- At best, we can find a minimizer f^e of the **empirical loss**:

$$f^e = \arg \min_{f \in \mathcal{H}} \mathcal{L}_{empirical}(f) = \frac{1}{N} \sum_{i=1}^N \ell(f(\mathbf{x}^{(i)}), y^{(i)})$$

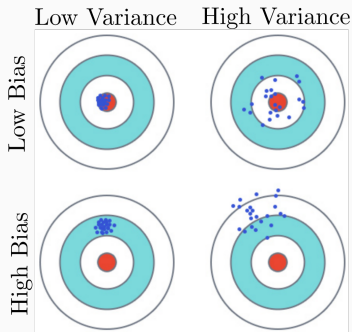
The Bias-Variance Trade-Off

- Figure that we repeatedly draw training sets of size N from p_{true} :
 - the training set is random
 - thus, the learned model f^{emp} is random
 - thus, $\mathcal{L}_{true}(f^{emp})$ is also random
 - thus, the **excess loss** $\delta = \mathcal{L}_{true}(f^{emp}) - \mathcal{L}_{true}(f^*)$ is also random
- The $\mathbb{E}[\delta]$ is denoted as **bias**, measuring how far $\mathcal{L}_{true}(f^{emp})$ is from $\mathcal{L}_{true}(f^*)$ on average
- The **variance** $\text{var}[\delta] = \mathbb{E}[(\delta - \mathbb{E}[\delta])^2]$ measures how much the true loss of f^{emp} varies

The Bias-Variance Trade-Off cont'd

Bias and variance depend on the model complexity:

- simple models tend to have **high bias** and **low variance**
⇒ **underfitting**
- expressive, flexible models tend to have **low bias** and **high variance**
⇒ **overfitting**



Why Ensembles?

- Individual models (like decision trees) easily overfit, i.e. they have low bias but high variance
- **Idea:** instead of training one model, train K models and aggregate them
- We can expect that $\text{var}(\text{ensemble}) \approx \frac{\text{var}(\text{single model})}{K}$
- What is the problem with this idea?

- We have only one dataset—use **bootstrapping** to generate synthetic copies of the training data
- In statistics, bootstrapping is a **resampling method** to produces uncertainty estimates
- Given N samples, generate K new datasets of size N by **sampling with replacement**
- These new datasets are called **bootstraps**
- For large N , each bootstrap contains only $1 - \frac{1}{e} \approx 63.21\%$ of the original samples

Original Data	\mathbf{x}_1	\mathbf{x}_2	\mathbf{x}_3	\mathbf{x}_4	\mathbf{x}_5	\mathbf{x}_6	\mathbf{x}_7	\mathbf{x}_8	\mathbf{x}_9	\mathbf{x}_{10}
Bootstrap 1	\mathbf{x}_7	\mathbf{x}_8	\mathbf{x}_{10}	\mathbf{x}_8	\mathbf{x}_2	\mathbf{x}_5	\mathbf{x}_{10}	\mathbf{x}_{10}	\mathbf{x}_5	\mathbf{x}_9
Bootstrap 2	\mathbf{x}_1	\mathbf{x}_1	\mathbf{x}_9	\mathbf{x}_1	\mathbf{x}_2	\mathbf{x}_3	\mathbf{x}_2	\mathbf{x}_7	\mathbf{x}_3	\mathbf{x}_2
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
Bootstrap K	\mathbf{x}_1	\mathbf{x}_8	\mathbf{x}_5	\mathbf{x}_{10}	\mathbf{x}_5	\mathbf{x}_5	\mathbf{x}_9	\mathbf{x}_6	\mathbf{x}_3	\mathbf{x}_7

- **Bagging** with Decision Trees
- **Bagging = Bootstrapping and Aggregating**
- Generate K bootstraps, and learn a decision tree on each of them
- Additionally, randomize CART learning (to get more diverse trees): Rather than considering all variables for a new split, restrict to m randomly chosen variables, $m < D$
- Typical values, $K = 100, 200, 500$, $m \approx \frac{D}{3}$
- Run-time: K times the run-time of Decision Trees

Decision Tree vs. Random Forest

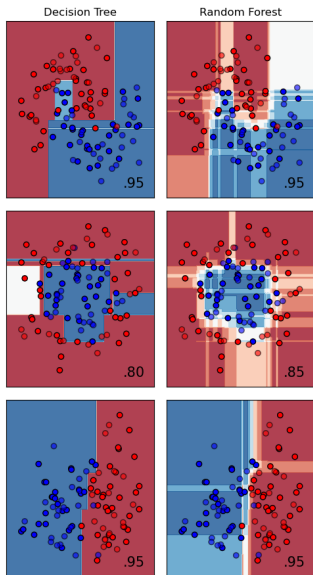
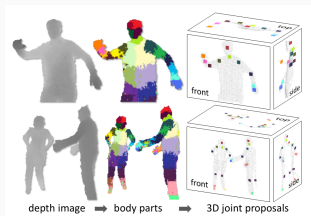


Image: <https://scikit-learn.org/>

Random Forests in Practice

- Kaggle reported in 2019 that decision trees and random forests are the most widely used tool in data mining and applied machine learning, after logistic regression
- Fernández-Delgado et al. (2014) random forests perform best among a wide range of algorithms on **tabular data**
- Prominent example: Body-part detection for skeleton tracking (Kinect 2)



- **Decision trees**: top-down decision diagram
- Decision nodes split input space recursively
- Leaves contain localized predictors
- CART algorithm: recursively find optimal decision (split) by minimizing impurity
- **Random forest**: randomized ensemble of decision trees
- **Bagging**: bootstrapping and aggregating