K-Nearest Neighbors, Decision Trees and Random Forests

Machine Learning 1 — Lecture 10

28th May 2024

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 $\boldsymbol{\theta}$ and we specified a training loss $\mathcal{L}_{train}(\boldsymbol{\theta})$ which we aimed to minimize w.r.t. $\boldsymbol{\theta}$
- 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

Nearest Neighbour

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

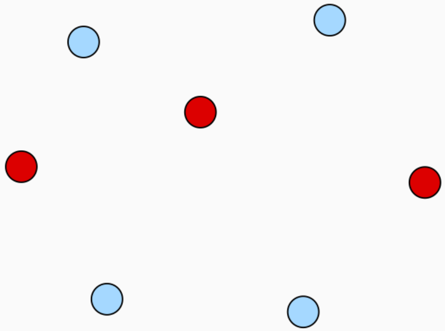
$$i = \arg\min_i \|\boldsymbol{x}^* - \boldsymbol{x}^{(i)}\|$$

  for some norm $\|\cdot\|$
- The **nearest neighbour** predictor returns the target of the nearest neighbour

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

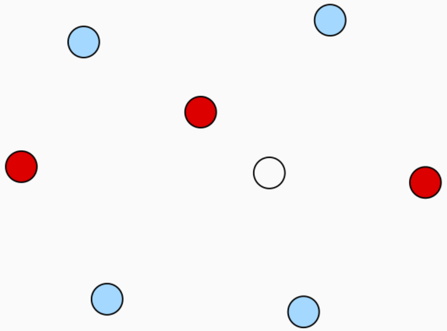
- Note that this works for both classification and regression

Nearest Neighbour Classifier



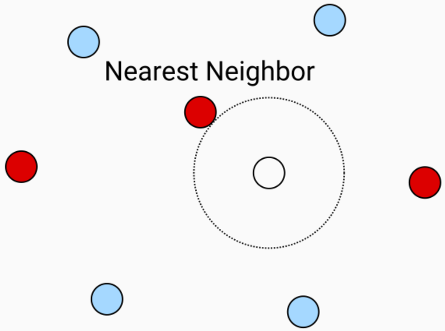
Nearest Neighbour Classifier

Example



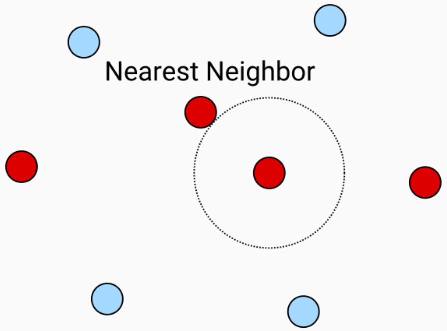
Nearest Neighbour Classifier

Example



Nearest Neighbour Classifier

Example



K-Nearest Neighbors (KNN)

- 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 $\boldsymbol{x}^*$:

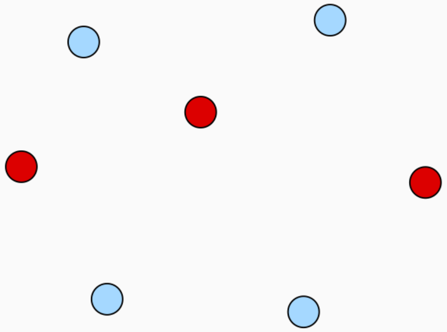
$$\|\boldsymbol{x}^{(i_1)} - \boldsymbol{x}^*\| \leq \|\boldsymbol{x}^{(i_2)} - \boldsymbol{x}^*\| \leq \cdots \leq \|\boldsymbol{x}^{(i_K)} - \boldsymbol{x}^*\| \leq \cdots \leq \|\boldsymbol{x}^{(i_N)} - \boldsymbol{x}^*\|$$

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

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

KNN Classification

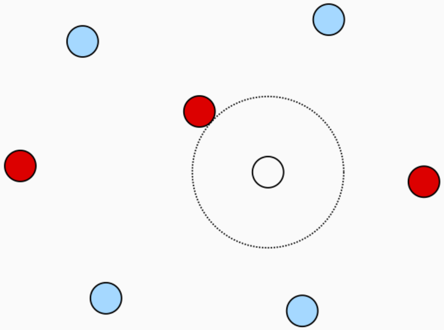
Example



KNN Classification

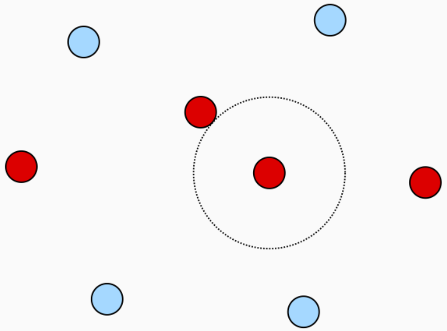
Example

k=1



KNN Classification

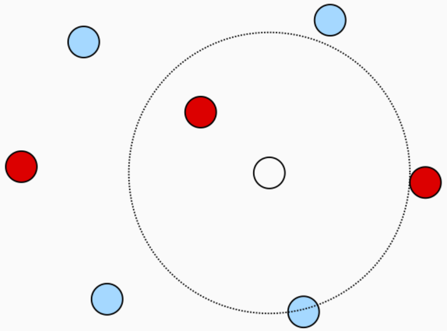
k=1



KNN Classification

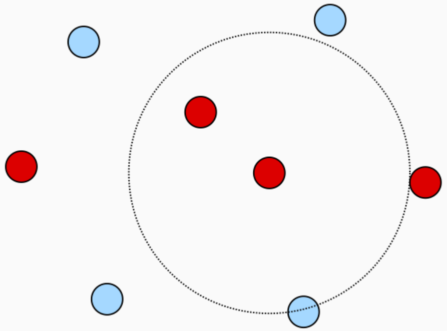
Example

k=3



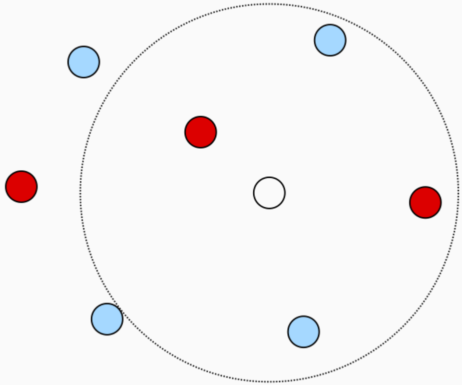
KNN Classification

k=3



KNN Classification

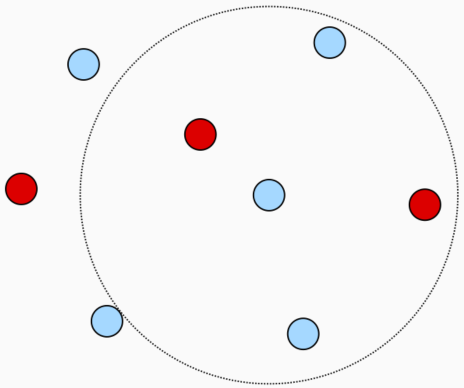
k=5



KNN Classification

Example

k=5



Bayes Optimal Classifier

Definition

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

$$\hat{y} = \arg\max_y \, p_{true}(y \mid \boldsymbol{x})$$

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

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

is minimal among all possible classifiers.

Consistency of KNN

Theorem

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

In particular, if

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

then KNN converges towards the **Bayes optimal classifier**.

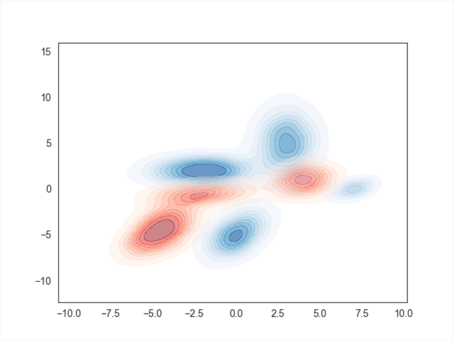
A classifier with this property is called **consistent**.

Furthermore, for regression, KNN converges to the **optimal regression function** $\mathbb{E}_{p_{true}(y \mid \boldsymbol{x})}[y]$.

Bayes Consistency of KNN

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

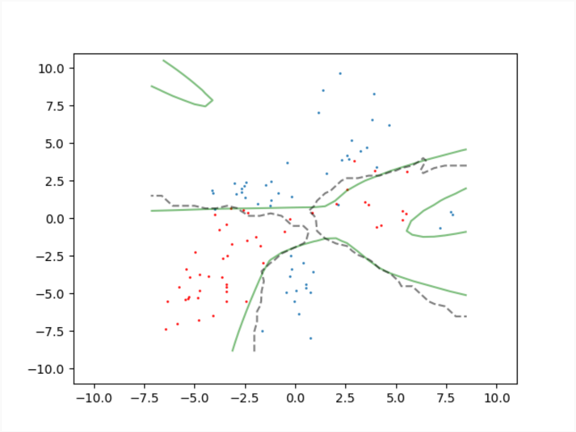
Assume a binary classification problem and assume the following true data distribution $p_{true}(\boldsymbol{x}, y)$ (red and blue for the two classes):



Thus, we can actually compute the Bayes optimal classifier!

Bayes Consistency of KNN cont'd

Example

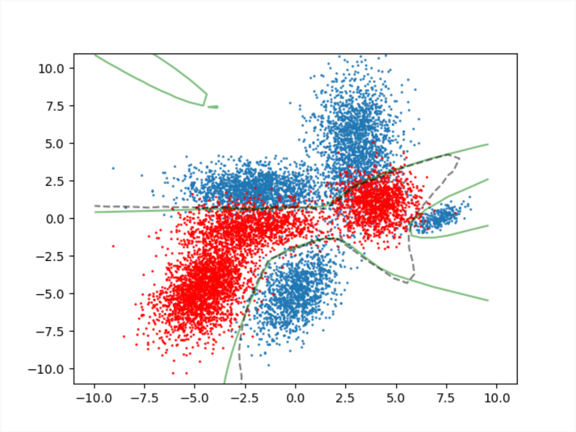


Green line: Bayes optimal classifier (94.67% test accuracy)
Dashed: KNN $N_{train} = 100$ and $K = 9$ (90.26% test accuracy)

(accuracies computed on a 1 million test samples.)

Bayes Consistency of KNN cont'd

Example

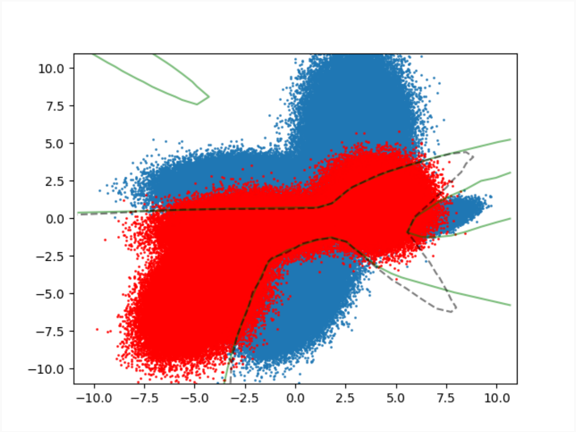


Green line: Bayes optimal classifier (94.67% test accuracy)
Dashed: KNN $N_{train} = 10k$ and $K = 99$ (94.56% test accuracy)

(accuracies computed on a 1 million test samples.)

Bayes Consistency of KNN cont'd

Example



Green line: Bayes optimal classifier (94.67% test accuracy)
Dashed: KNN $N_{train} = 1M$ and $K = 999$ (94.65% test accuracy)

(accuracies computed on a 1 million test samples.)

Summary

- simple but consistent predictor
- training is very easy: just store the data
- testing is $\mathcal{O}(N)$
- **k-d trees** organize the data in hierarchical way, then testing can be done on $\mathcal{O}(\log N)$

Decision Trees

Decision Trees

Example

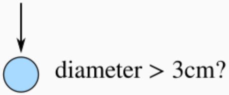
Decision Trees

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

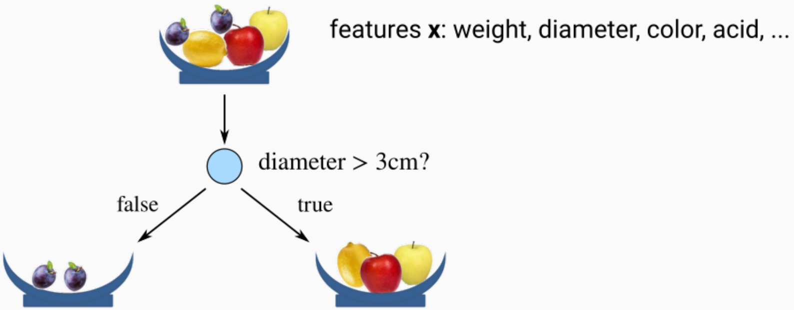
Decision Trees

Example

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

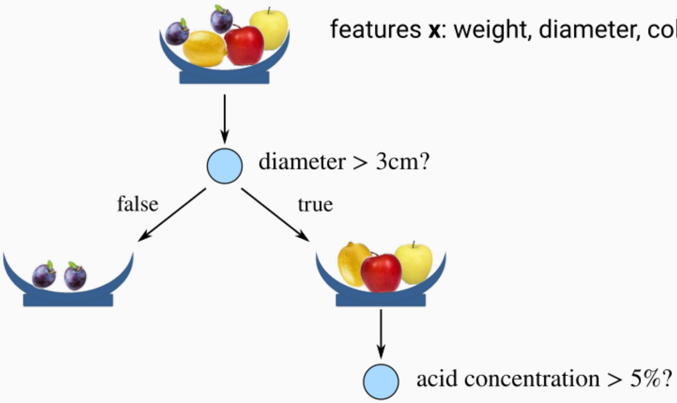


Decision Trees

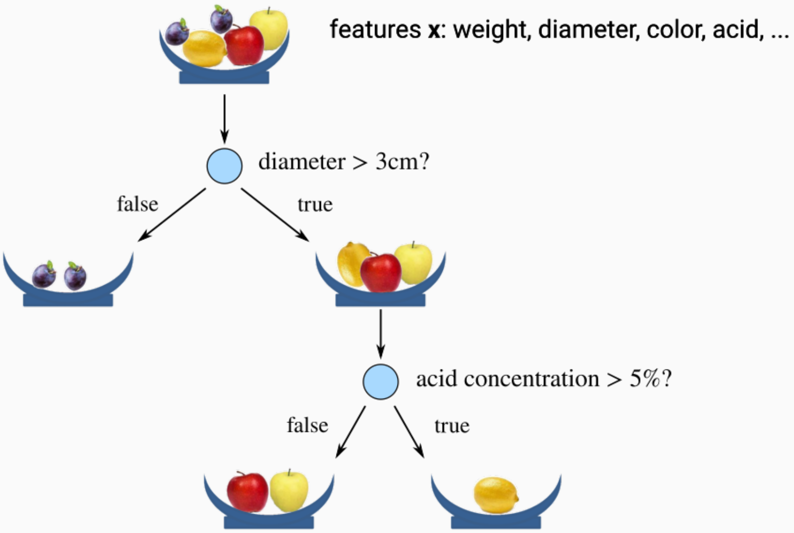


Decision Trees Example

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



Decision Trees



Decision Tree

Given a set of features $X_1, X_2, \ldots, X_D$ (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**)

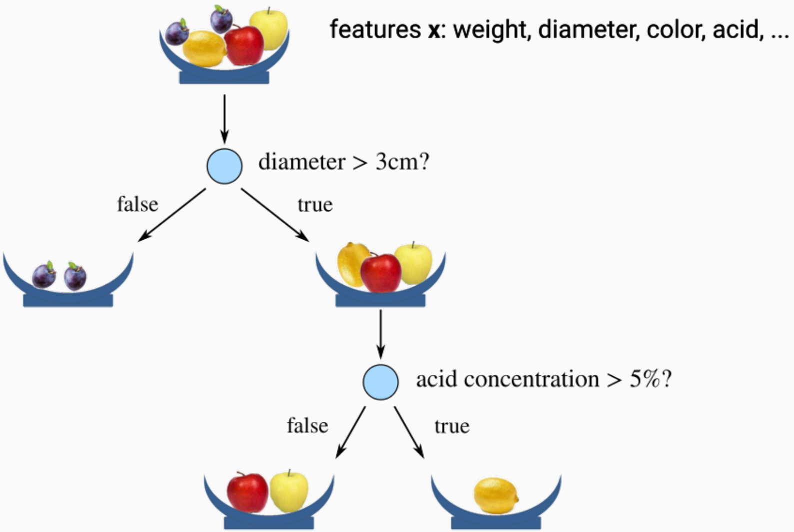
  $$f : X_i \mapsto \{true, false\}$$

  Decision nodes have 2 children labeled *true* and *false*, which are selected by the output of 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

- Parse 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 $p(Y)$
- Easily generalized to non-binary decision nodes
- Decision trees can also be used for regression (not discussed)

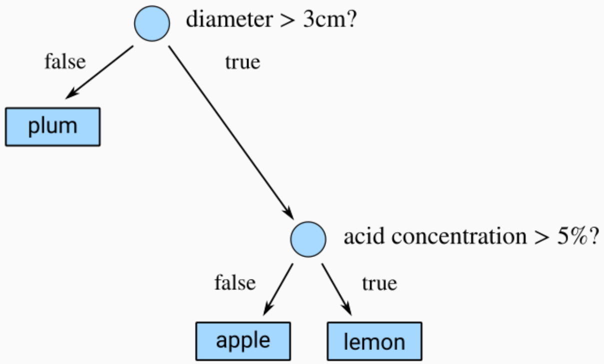
Decision Trees



Decision Trees

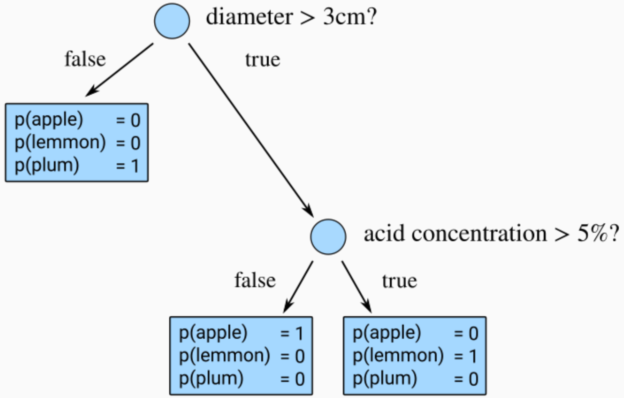
Example

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



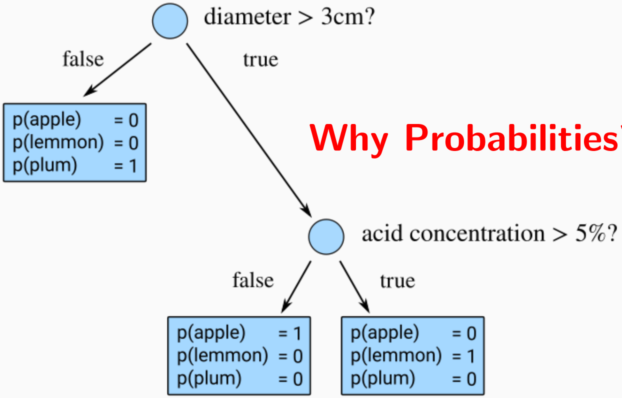
Decision Trees

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



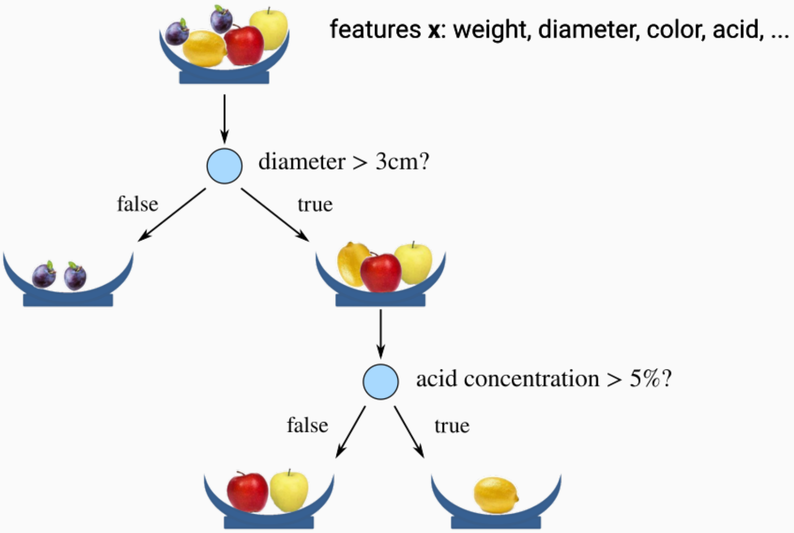
Decision Trees

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

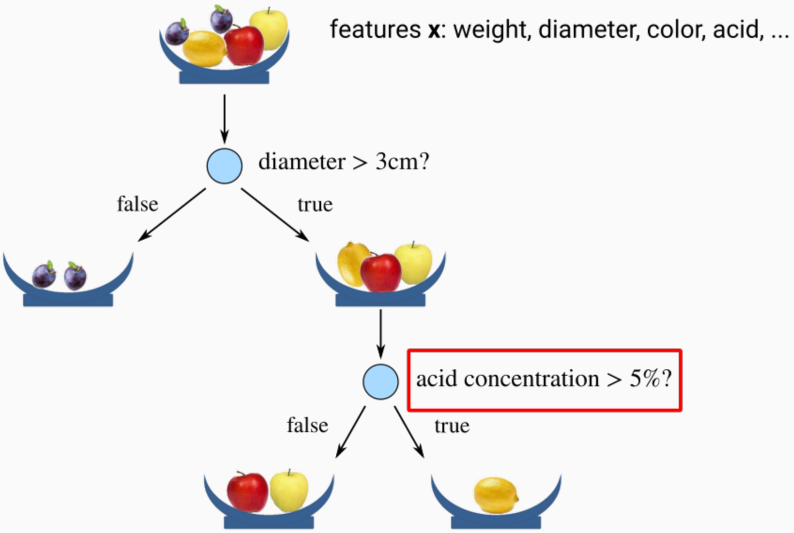


Why Probabilities?

Decision Trees

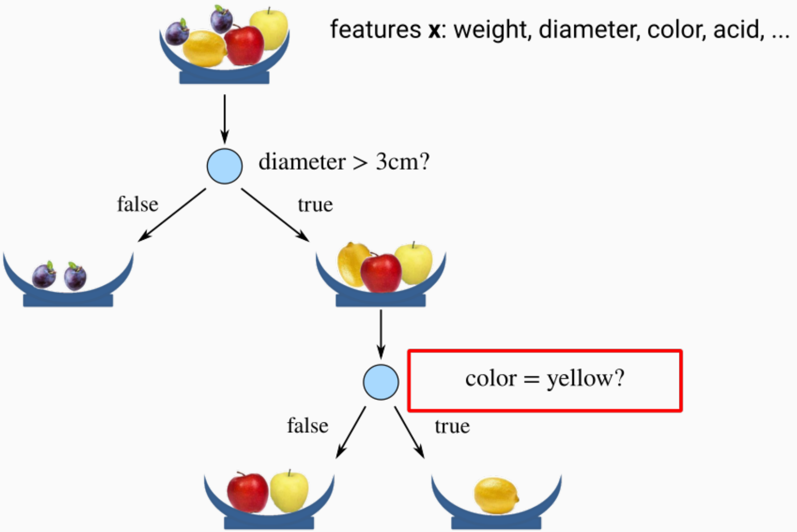


Decision Trees

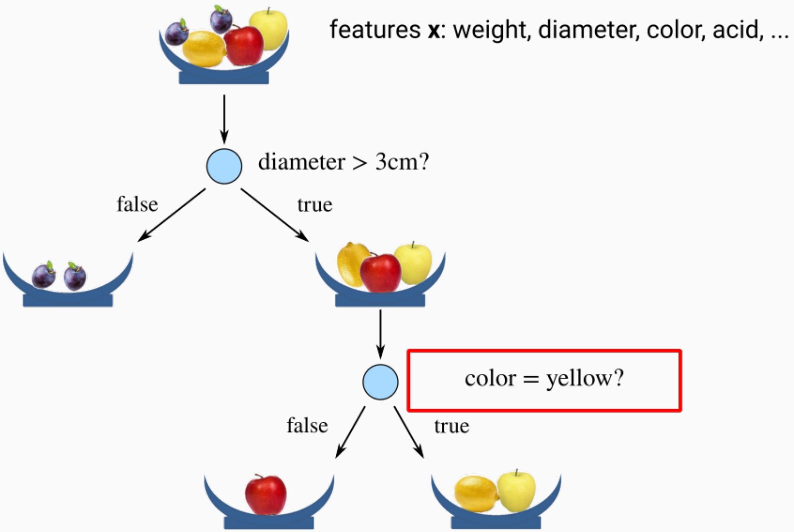


Decision Trees

Example



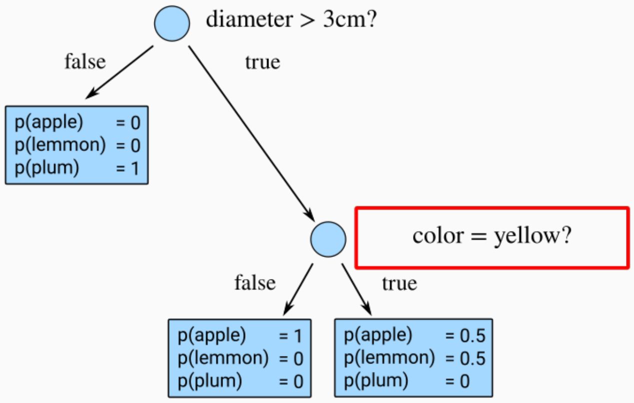
Decision Trees



Decision Trees

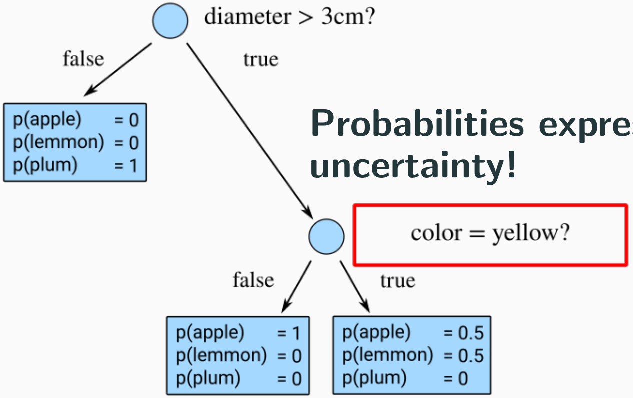
Example

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



Decision Trees

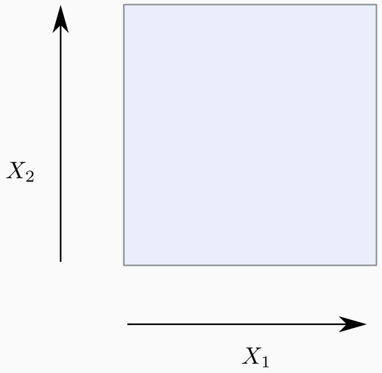
features **x**: weight, diameter, color, acid, ...



Probabilities express uncertainty!

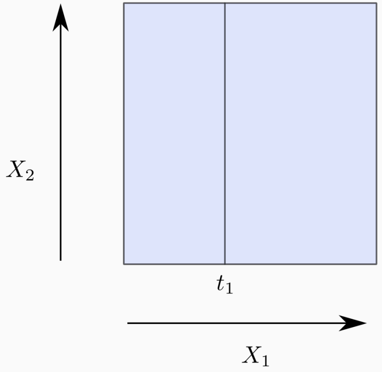
Partition of the Input Space

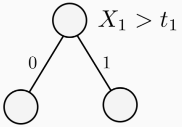
Decision trees represent a partition of the input space:



Partition of the Input Space

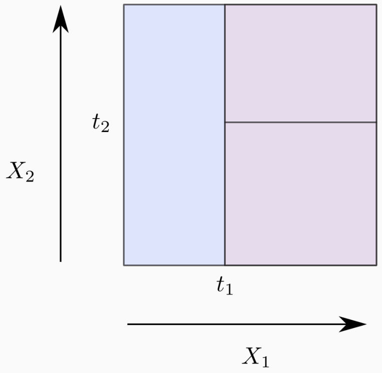
Decision trees represent a partition of the input space:

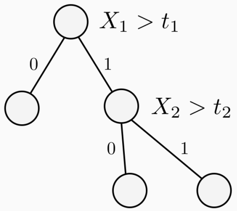




Partition of the Input Space

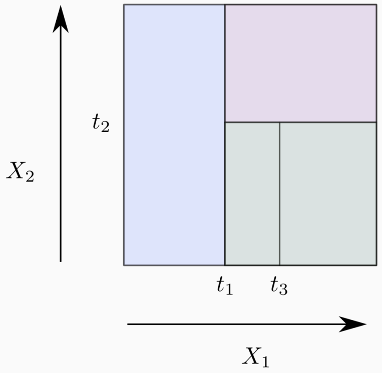
Decision trees represent a partition of the input space:

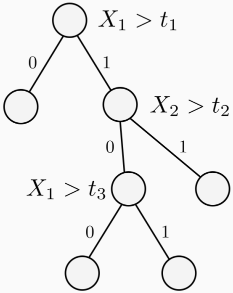




Partition of the Input Space

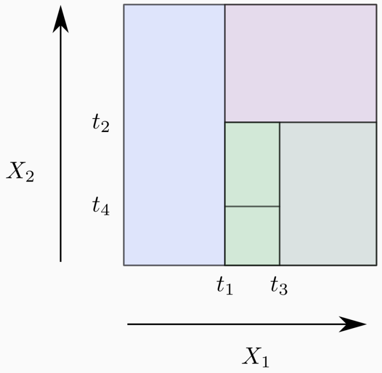
Decision trees represent a partition of the input space:

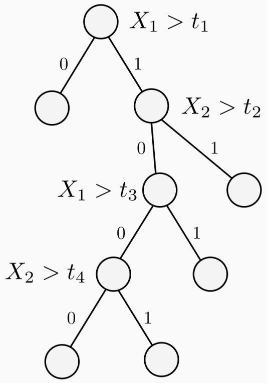




Partition of the Input Space

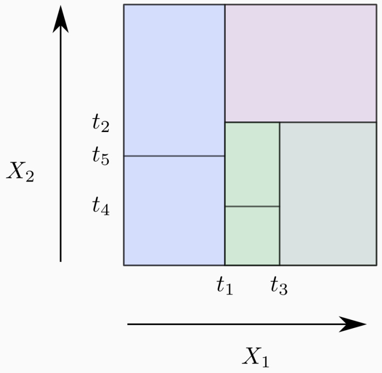
Decision trees represent a partition of the input space:

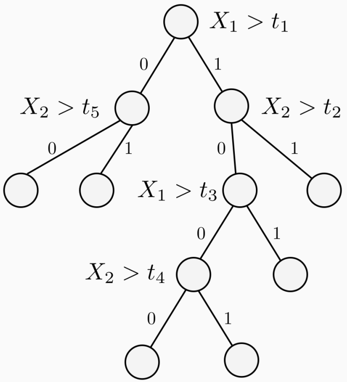




Partition of the Input Space

Decision trees represent a partition of the input space:





Learning Algorithms for Decision Trees

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

CART

```
input: Data $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)$
result: Decision Tree
Initialize single leaf $L$, assign all data points $\boldsymbol{x}^{(i)}$ to $L$, call split($L$)

split($L$):
if stopping criterion is true for $L$ then
    Learn $p(Y)$ from class proportions of samples assigned to $L$
    return
end
for considered variables/decisions $X_d$, $f$ do
    compute cost($X_d, f$)
end
Let $X_{min}$, $f_{min}$ be the decision with minimal cost
Apply $f_{min}(X_{min})$ to $L$, yielding new leaves $L_0$ and $L_1$
Assign $\boldsymbol{x}^{(i)}$ with $f(\boldsymbol{x}^{(i)}) = false$ to $L_0$, call split($L_0$)
Assign $\boldsymbol{x}^{(i)}$ with $f(\boldsymbol{x}^{(i)}) = true$ to $L_1$, call split($L_1$)
```

CART

```
input: Data $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_N, y_N)$
result: Decision Tree
Initialize single leaf $L$, assign all data points $\boldsymbol{x}^{(i)}$ to $L$, call split($L$)

split($L$):
if stopping criterion is true for $L$ then
    Learn $p(Y)$ from class proportions of samples assigned to $L$
    return
end
for considered variables/decisions $X_d$, $f$ do
    compute cost($X_d, f$)
end
Let $X_{min}$, $f_{min}$ be the decision with minimal cost
Apply $f_{min}(X_{min})$ to $L$, yielding new leaves $L_0$ and $L_1$
Assign $\boldsymbol{x}^{(i)}$ with $f(\boldsymbol{x}^{(i)}) = false$ to $L_0$, call split($L_0$)
Assign $\boldsymbol{x}^{(i)}$ with $f(\boldsymbol{x}^{(i)}) = true$ to $L_1$, call split($L_1$)
```

Possible Decisions

```
   input  : Data $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)$
   Result: Decision Tree
1  Initialize single leaf $L$, assign all data points $\boldsymbol{x}^{(i)}$ 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 variables/decisions $X_d$, $f$ do
8      compute 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 $\boldsymbol{x}^{(i)}$ with $f(\boldsymbol{x}^{(i)}) = false$ to $L_0$, call split($L_0$)
13 Assign $\boldsymbol{x}^{(i)}$ with $f(\boldsymbol{x}^{(i)}) = true$ 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

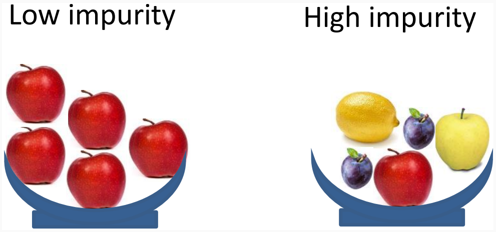
- **one-vs-all** $\{1,2,3,4\} \rightarrow \{1,2,4\},\{3\}$
- **complete split** (not a binary tree) $\{1,2,3,4\} \rightarrow \{1\},\{2\},\{3\},\{4\}$

Cost

```
   input  : Data $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)$
   Result: Decision Tree
1  Initialize single leaf $L$, assign all data points $\boldsymbol{x}^{(i)}$ 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 variables/decisions $X_d$, $f$ do
8      compute 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 $\boldsymbol{x}^{(i)}$ with $f(\boldsymbol{x}^{(i)}) = false$ to $L_0$, call split($L_0$)
13 Assign $\boldsymbol{x}^{(i)}$ with $f(\boldsymbol{x}^{(i)}) = true$ to $L_1$, call split($L_1$)
```

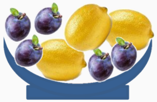
Cost



- "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
- *cost* $= \frac{N_0}{N} impurity(L_0) + \frac{N_1}{N} impurity(L_1)$
- 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.



$$p(plum) = \frac{5}{8} = 0.625$$
$$p(lemon) = \frac{3}{8} = 0.375$$

- **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 **Example**

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 $(0.5, 0.5)$.

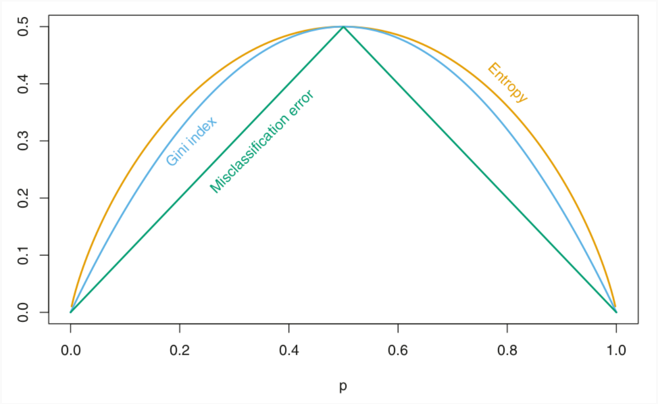


Image: Trevor Hastie et al., *Elements of Statistical Learning*, 2009.

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.
- $cost = \frac{N_0}{N} impurity(L_0) + \frac{N_1}{N} impurity(L_1)$
- 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**

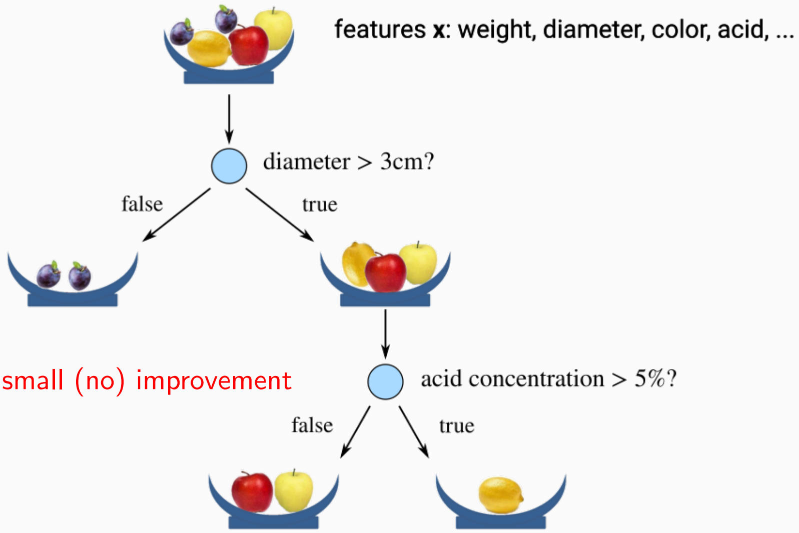
$$C = 1 - \max_y p(y)$$

Stopping Criterion

```
    input  : Data $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)$
    Result: Decision Tree
 1  Initialize single leaf $L$, assign all data points $\boldsymbol{x}^{(i)}$ 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 variables/decisions $X_d$, $f$ do
 8      compute 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 $\boldsymbol{x}^{(i)}$ with $f(\boldsymbol{x}^{(i)}) = false$ to $L_0$, call split($L_0$)
13  Assign $\boldsymbol{x}^{(i)}$ with $f(\boldsymbol{x}^{(i)}) = true$ to $L_1$, call split($L_1$)
```

Stopping Criterion



too small (no) improvement

too small (no) improvement

too few data points

Stopping Criterion and Hyperparameters

- Threshold determining when improvement of cost is too small
- Another threshold for determining when there are too few data points
- Furthermore, one introduce a maximal tree depth
- These thresholds are **hyper-parameters** of CART

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**:

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

- 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**:

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

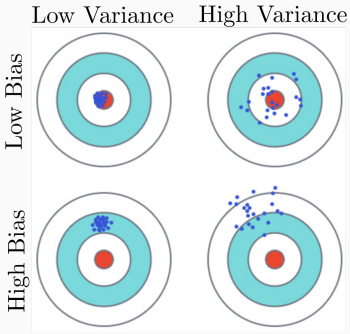
The Bias-Variance Trade-Off

- 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** **var**$[\delta] = \mathbb{E}\left[(\delta - \mathbb{E}[\delta])^2\right]$ 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** $\Rightarrow$ **underfitting**
- expressive, flexible models tend to have **low bias** and **high variance** $\Rightarrow$ **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 $\textbf{var}(\text{ensemble}) \approx \frac{\textbf{var}(\text{single model})}{K}$
- What is the problem with this idea?

Bootstrapping

- 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	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$	$\boldsymbol{x}_5$	$\boldsymbol{x}_6$	$\boldsymbol{x}_7$	$\boldsymbol{x}_8$	$\boldsymbol{x}_9$	$\boldsymbol{x}_{10}$
Boostrap 1	$\boldsymbol{x}_7$	$\boldsymbol{x}_8$	$\boldsymbol{x}_{10}$	$\boldsymbol{x}_8$	$\boldsymbol{x}_2$	$\boldsymbol{x}_5$	$\boldsymbol{x}_{10}$	$\boldsymbol{x}_{10}$	$\boldsymbol{x}_5$	$\boldsymbol{x}_9$
Boostrap 2	$\boldsymbol{x}_1$	$\boldsymbol{x}_1$	$\boldsymbol{x}_9$	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_2$	$\boldsymbol{x}_7$	$\boldsymbol{x}_3$	$\boldsymbol{x}_2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Boostrap K	$\boldsymbol{x}_1$	$\boldsymbol{x}_8$	$\boldsymbol{x}_5$	$\boldsymbol{x}_{10}$	$\boldsymbol{x}_5$	$\boldsymbol{x}_5$	$\boldsymbol{x}_9$	$\boldsymbol{x}_6$	$\boldsymbol{x}_3$	$\boldsymbol{x}_7$

Random Forests

- **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

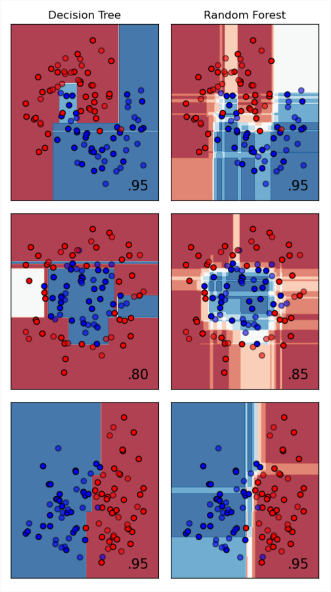
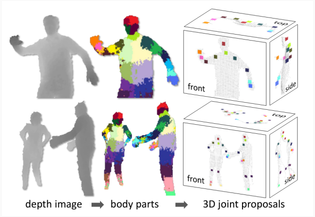


Image: https://scikit-learn.org/

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
- Fernánandez-Delgado et al. (2014) random forests perform best among a wide range of algorithms on **tabular data**
- Prominent example: Body-part detection for skeleton tacking (Kinect 2)



Shotton et al., Real-Time Human Pose Recognition in Parts from Single Depth Images, 2011

Summary

- **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