GHENT
UNIVERSITY

Lecture Notes

E016350: Artificial Intelligence

Decision Trees

Aleksandra Pizurica

Spring 2025

Contents

Disclaimer: *These lecture notes were written by Prof. Aleksandra Pizurica to accompany the slides of the course E016350: Artificial Intelligence, facilitating their understanding. The lecture notes are not meant to be self-contained, and do not cover all the study material in the course. They are by no means meant to replace the recommended textbook and do not necessarily cover all the relevant aspects that are presented in the slides and explained in the lectures. Some sections are adapted from the book of S. Russel and P. Norvig: Artificial Intelligence: A Modern Approach.*

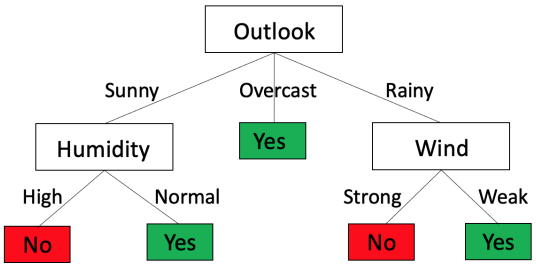

Figure 1: An example of a simple decision tree.

1 Decision trees

Decision trees underlay many of today's most successful learning approaches. They are able to learn complex, **nonlinear** relationships between variables, using a series of simple, **intuitive** decision rules: start with one test, and depending on its outcome decide what the next test will be. This process continues until a decision is reached.

1.1 Interpretation and basic types of decision trees

Fig. 1 shows a simple example of a decision tree, which is often given in the introductory materials on this topic. Here, the decision on "*Should I play tennis today?*" is based on the values of three attributes: outlook from the window (with three possible outcomes sunny, overcast or rainy), humidity (which can be high or normal) and wind (which can be strong or weak). The training will be done based on some training set that contains examples with different combinations of the attribute values and "play" ("yes") or "not play" ("no") decisions for each of them.

Formally, like in any supervised ML approach, a decision tree is learned from examples $(\mathbf{x}, y) \in \mathcal{D}_{train}$, where $\mathbf{x}$ are the values of some **features** (or **attributes**) $\mathbf{X}$ and y is the output label. The structure and the elements of a decision tree have the following interpretation:

- **Root and internal nodes** test a feature X_i. In our tree: $X_1 = Outlook$, $X_2 = Humidity$, $X_3 = Wind$
- **Branching** is determined by the feature value E.g. $x_3 = wind \in \{strong, weak\}$
- **Leaf nodes** are outputs (decisions, predictions)

We will interpret the decision as a prediction – we use the decision tree as a *predictive model*. When a nominal (categorical) variable is predicted, the tree is called a classification tree (like the tree in Fig. 1). We can also use decision trees to predict a numerical variable, this is then a regression tree. In principle, the tree can also predict multiple variables at once (or, equivalently, a tuple-valued variable); such trees are sometimes called multi-target trees. Classification trees that do not merely predict a class, but define a conditional probability for each class given the input, are called probability estimation trees. In summary, the output of a decision tree can be of different types, including:

- **numerical** (our model is then a **regression tree**)
- **categorical** (we call it then a **classification tree**)
- **tuple-valued** (in the so-called **multi-target trees**)

- $P(y|\mathbf{x})$ (we call these models **probability estimation trees**)

The decision trees can be used in many settings and we can define other sub-categories of these models next to those that are listed above. Notably, as it repeatedly divides a data set into subsets, a tree implicitly defines a hierarchical clustering. Trees learned for this purpose are called **clustering trees**. The difference between a hierarchical clustering defined by a clustering tree, and one defined by other clustering algorithms, is that each cluster in a clustering tree is defined precisely by a set of test outcomes. **Density estimation trees** partition the dataset into regions of high and low density, and as such can be used to describe the joint probability distribution of the data.

The decision trees are widely used because they are **easy to understand and interpret** and because they **require little or no data preparation**. Moreover, they provide basis to some of the best performing ML models today: **Random forests** or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. In classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned [4].

1.2 Case study: Restaurant domain

We will study decision trees on a use case *Restaurant domain* from [6]: the problem of deciding whether to wait for a table at a restaurant based on the following attributes:

1. *Alternate (Alt)*: Is there a suitable alternative restaurant nearby?
2. *Bar (Bar)*: Is there a comfortable bar area in the restaurant, where I can wait?
3. *Fri/Sat (Fri)*: True on Fridays/Saturdays
4. *Hungry (Hun)*: Are we hungry?
5. *Patrons (Pat)*: How many people are in the restaurant (*None*, *Some* or *Full*)
6. *Price (Price)*: the restaurant's price range ($, $$, $$$)
7. *Raining (Rain)*: Is it raining outside?
8. *Reservation (Res)*: Did we make a reservation?
9. *Type (Type)*: the kind of restaurant (French, Italian, Thai or burger)
10. *WaitEstimate (Est)*: the wait time estimated by the host (0-10, 10-30, 30-60, or>60 min)

The training set consists of 12 examples that are given in the following table [6]:

	Input Attributes										*Output*
Example	*Alt*	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
10	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
11	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
12	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

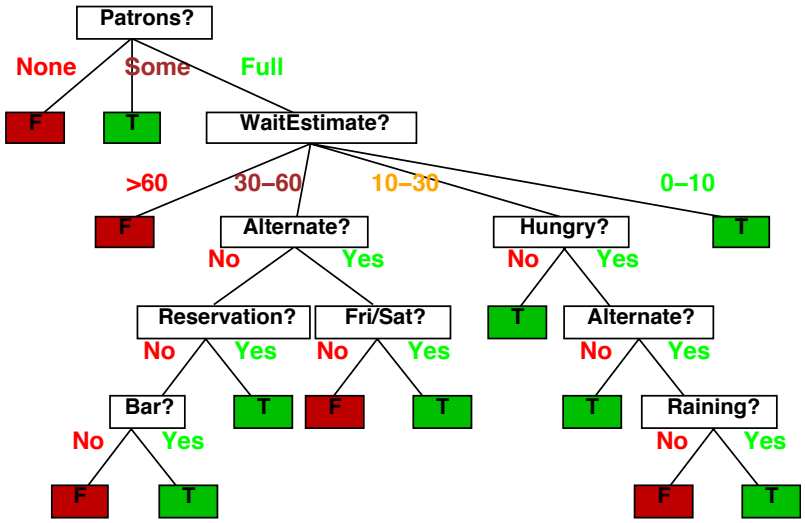

Figure 2: Ground truth tree for the restaurant problem.

Each raw in this table is an example $(\mathbf{x}^{(i)}, y^{(i)})$, where $\mathbf{x}^{(i)}$ contains values of the 10 attributes and the output $y^{(i)}$ is true (T) or false (F). One possible tree that correctly represents these examples is shown in Fig. 2. This is also the 'ground truth' tree, which represents exactly the actual decision function that was used by the person (Stuart Russel) who gave these examples [6].

We suppose our AI system doesn't have access to this 'true' tree but needs to *learn* a good decision tree from the supplied examples. We will then compare the learned tree to this true one to see how well our system learned a good decision model.

Note that there are $2^6 \times 3^2 \times 4^2 = 9216$ combinations for the attributes in this problem while we are given only 12. This is the essence of *induction*: make the best guess for many missing output values given only the evidence of few examples.

1.3 Expressiveness of decision trees

Decision trees can express any function of the input attributes. For Boolean functions, each row in the truth table is one path to the leaf (see an illustration in Fig. 3). For many problems,

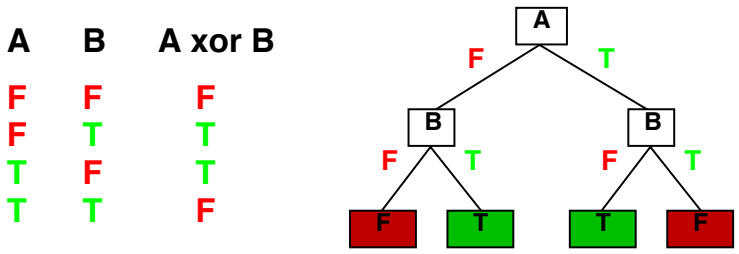

Figure 3: Truth table for the logical operation 'XOR' and the corresponding decision tree.

the decision tree format yields a nice, concise, understandable result. But some functions cannot be represented concisely. For example, the majority function, which returns true if and only if more than half of the inputs are true, requires an exponentially large decision tree [6].

In general, we will increase the expressiveness of the tree by using more attributes. With more attributes, the decision tree has more potential splitting points to choose from. This allows to model more complex relationships in the data. However, the number of possible trees grows combinatorially. For example, for a Boolean function with n attributes the truth table has 2^n rows, which means there are 2^{2n} distinct truth tables. With only 10 Boolean attributes there are 10^{308} possible trees. Finding the best hypothesis in a such a huge hypothesis space becomes very difficult. Moreover, the risk of overfitting increases.

In summary, more attributes generally increase the tree's expressiveness but at the cost of higher risks of overfitting and computational complexity. Conversely, fewer attributes lead to simpler, potentially less expressive models but with reduced risks of overfitting and improved interpretability. Effective feature selection and regularization techniques are essential to balance the expressiveness and generalization of decision trees.

2 Decision tree learning

In general, the goal of decision tree learning is to find a tree that is consistent with the provided examples and is as small as possible. Unfortunately, it is intractable to find a guaranteed smallest consistent tree [6]. Therefore we resort to a greedy approach and it turns out that with some simple heuristics, we can efficiently find a tree that is close to the smallest consistent tree.

2.1 Decision tree learning algorithm

Here we describe a greedy algorithm known as the decision tree learning algorithm. Its idea is to choose the "most significant" attribute as the root and repeat this recursively for each subtree. We start with the whole training set and an empty decision tree. Then pick the feature that gives the best split. We split on that feature and repeat the process on the sub-partitions.

Fig. 4 gives a pseudo-code for the decision tree learning algorithm. The function IMPORTANCE measures the importance of attributes (as explained next). The PLURALITY-VALUE function selects the most common output value among a set of examples, breaking ties randomly.

```
function LEARN-DECISION-TREE(examples, attributes, parent_examples) returns a tree

  if examples is empty then return PLURALITY-VALUE(parent_examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return PLURALITY-VALUE(examples)
  else
      A ← argmax_{a ∈ attributes} IMPORTANCE(a, examples)
      tree ← a new decision tree with root test A
      for each value v of A do
          exs ← {e : e ∈ examples and e.A = v}
          subtree ← LEARN-DECISION-TREE(exs, attributes − A, examples)
          add a branch to tree with label (A = v) and subtree subtree
      return tree
```

Figure 4: Pseudo-code of the decision tree learning algorithm [6].

2.2 Choosing important attributes based on the information gain

The key question is how to choose the most *important* attribute in each phase of the decision tree learning. The general idea is that a good (i.e., important) attribute is one that makes the most difference to the classification of an example. With Boolean attributes, this means an attribute that splits well the examples into subsets that are (ideally)"all positive" or "all negative".

Common techniques and criteria used to identify important attributes include **information gain** (which is equivalent to **entropy reduction**) and the **Gini index** (a measure for the "impurity" of a dataset). We will focus our attention to the first criterion.

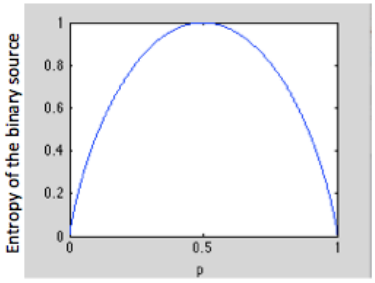

Figure 5: Entropy of a binary information source $H(\langle p, 1-p\rangle)$.

Information answers questions – the more clueless we are about the answer initially, the more information is contained in the answer. In information theory, *entropy* is a measure of the uncertainty of a random variable, the "expected surprisal". The more information, the less entropy.

Let us denote the **entropy of an information source** with n outcomes occurring with probabilities $P_1, \ldots P_n$ as

$$H(\langle P_1, \ldots, P_n\rangle) = \sum_{i=1}^{n} -P_i \log_2 P_i$$

This is information in an answer when the prior is $\langle P_1, \ldots, P_n\rangle$. For the binary source we have:

$$H(\langle p, 1-p\rangle) = -p \log_2(p) - (1-p) \log_2(1-p)$$

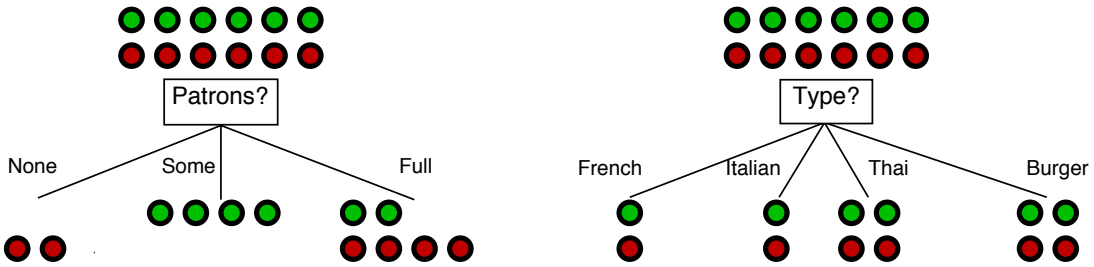

Figure 6: Splitting the examples from the *Restaurant* domain by testing on two different attributes. Note that splitting on *Type* brings us no nearer to distinguishing between positive and negative examples while splitting on *Patrons* does a good job of separating positive and negative examples. Example from [6].

The corresponding plot is shown in Fig. 5. Note that $H(\langle 1,1\rangle) = 1$, i.e., 1 **bit** is the information entropy of a random binary variable that takes values 0 and 1 with equal probability. Or, put in other words, 1 bit is an answer to Boolean question with prior $\langle 0.5, 0.5\rangle$.

Suppose we have p positive and n negative examples at the root. Then

$$B(\frac{p}{p+n}) = H(\langle \frac{p}{p+n}, \frac{n}{p+n}\rangle)$$

bits are needed to classify a new example. For the restaurant use case, out of the 12 training examples we had six positive and six negative ones, so $p = n = 6$, and thus we need exactly 1 bit of information to classify a new example. The result of a test on an attribute A will give us some information, thus reducing the overall entropy by some amount. We can measure this reduction by looking at the entropy remaining after the attribute test.

An attribute A with d distinct values divides the training set E into subsets $E_1, \ldots, E_d$ each of which (we hope) needs less information to complete the classification. Let E_i have p_i positive and n_i negative examples. This means that if we go along that branch, we will need

$$H(\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\rangle)$$

bits of information to answer the question. A randomly chosen example from the training set has the kth value for the attribute (i.e., is in E_k with probability $(p_k+n_k)/(p+n)$). Thus, the expected number of bits (EBS) needed if A is at the root is

$$EBS(A) = \sum_i \frac{p_i+n_i}{p+n} H\Big(\Big\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\Big\rangle\Big)$$

This is also the expected entropy remaining after testing attribute A. The **information gain** from the attribute test on A is the expected **reduction in entropy**:

$$Gain(A) = B(\frac{p}{p+n}) - EBS(A)$$

Take for example the attribute *Patrons* from the *Restaurant* domain. It has three possible outcomes and the splitting of the training examples based on this attribute (as can be read from

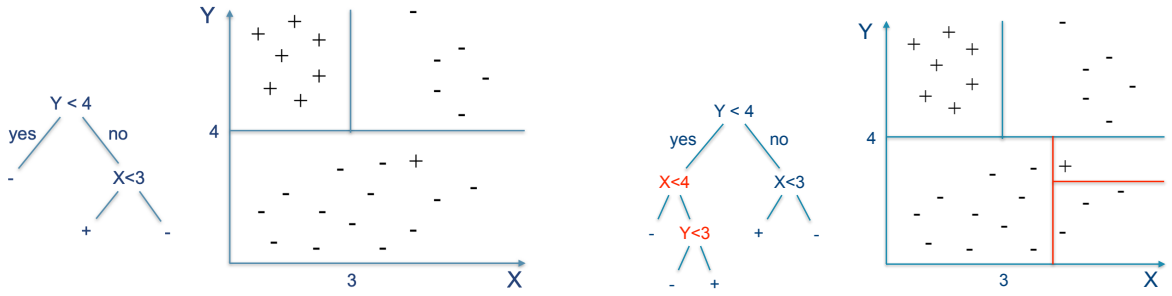

Figure 7: Left: a small tree fits the training data almost perfectly. It can be grown to fit perfectly (right), but a relatively large area to the right will then be predicted positive, while the data contains very little evidence for this. Example from [1].

the Table in Section 1.2) is shown in Fig. 6. For the value *None* (set E_1) we have $p_1 = 0$; $n_1 = 2$; for *Some* (set E_2): $p_2 = 4$; $n_2 = 0$ and for *Full* (set E_3): $p_3 = 4$; $n_3 = 0$. Thus,

$$\begin{aligned} EBS(Patrons) &= \sum_i \frac{p_i + n_i}{p + n} H\Big(\Big\langle \frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i} \Big\rangle\Big) \\ &= \frac{2}{12} \underbrace{H\Big(\Big\langle \frac{0}{2}, \frac{2}{2} \Big\rangle\Big)}_{0} + \frac{4}{12} \underbrace{H\Big(\Big\langle \frac{4}{4}, \frac{0}{4} \Big\rangle\Big)}_{0} + \frac{6}{12} H\Big(\Big\langle \frac{4}{6}, \frac{2}{6} \Big\rangle\Big) \\ &= \frac{1}{2} H\Big(\Big\langle \frac{2}{3}, \frac{1}{3} \Big\rangle\Big) = 0.4591 \text{ bits} \end{aligned}$$

the information gain of this attribute is

$$Gain(Patrons) = B(\frac{p}{p+n}) - EBS(Patrons) = 1 - 0.4591 \approx 0.541 \text{ bits}$$

It is easy to verify that for the attribute *Type* (for which the splitting is also shown in Fig. 6) the expected number of bits is still 1 and thus $Gain(Type) = 0$ bits. This confirms our intuition that *Patrons* is a better attribute to split on first. In fact, *Patrons* has the maximum information gain of any of the attributes and thus would be chosen by the decision tree learning algorithm as the root [6].

2.3 Some considerations

Another important question when dealing with decision trees is *when is it no longer useful to split a subset into smaller subsets?* For classification trees, it is clear that when a subset has zero class-entropy (that is, all cases in the subset have the same class), further splitting is no longer useful. For regression trees, the equivalent would be zero variance, but that is almost never achievable. Some learners stop splitting when the best test does not lead to a significant reduction of entropy or variance. When the subset to be split is very small, further reductions are almost certainly not significant; for that reason, many learners only split subsets whose size is above some threshold value.

It is known that too large trees tend to overfit the data: they fit the training data well, but tend to perform worse on other data. Fig. 7 illustrates this problem. Ideally, a tree learner stops

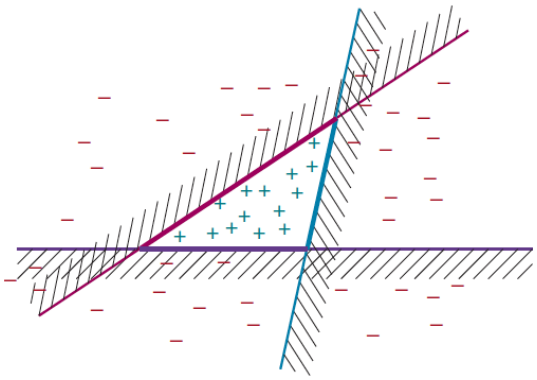

Figure 8: An example from [6] showing how ensemble learning can increase expressive power of simple (in this case, linear) models. By combining three linear models a triangular decision region can be achieved, which is beyond the possibility of any of the three base models alone.

splitting just before such overfitting occurs. However, it turns out it is very hard to determine the right moment. Statistical significance tests do not work well in this context, and it is perfectly possible that even if no single test leads to a substantial improvement, a combination of tests will. For this reason, many learners grow the tree beyond its optimal size, and prune away useless branches afterwards. This pruning process typically makes use of a so-called validation set: a set of data not used for learning the tree, but used to estimate the quality of the full tree and its pruned variants during the pruning process. Since the validation set was not used while growing the tree, it provides an unbiased view of the actual predictive accuracy of the tree. The pruning process then consists of pruning away branches that did not lead to a higher accuracy on the validation set (i.e., the improvement they gave on the training set was most likely due to overfitting) [1].

3 Ensemble learning

Traditional learning methods use a single hypothesis to make predictions. **Ensemble learning** aims to improve performance by combining multiple hypotheses (called *base models*), denoted $h_1, h_2, \ldots, h_n$, into a single **ensemble model** [6]. The models are typically aggregated using techniques such as averaging, majority voting, or by meta-learning [5].

The motivation behind this approach is twofold:

1. **Reducing bias** – A single model may be too simplistic, resulting in high bias (e.g., a linear classifier's limitations). An ensemble can represent more complex decision boundaries. Take an example from Fig. 8 where three linear classifiers together define a triangular region, which a single linear model cannot capture. Using n base models adds only n times more computation, which is often more efficient than training a more general, highly flexible model that may require exponential resources.

2. **Reducing variance** – Ensembles can also reduce variance of the prediction. Consider an example from [6]:
   Let an ensemble consist of $K = 5$ binary classifiers that we combine by majority voting. For a misclassification to occur, at least 3 out of 5 must be wrong, which is less likely than a single classifier making an error.

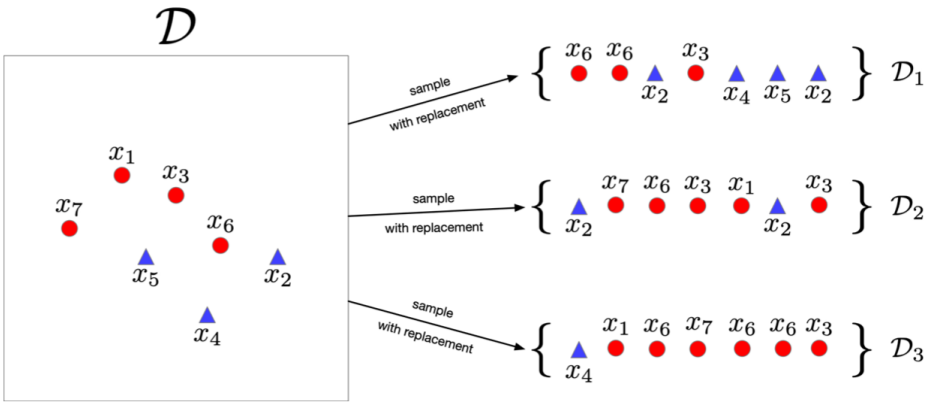

Figure 9: An illustration of sampling with replacement in the bagging procedure.

Suppose each individual classifier is 80% accurate. If we train 5 classifiers on different data subsets to promote independence (though it may slightly reduce individual performance to, say, 75%), the ensemble can still achieve 89% accuracy. With 17 classifiers, accuracy can reach 99%, assuming independence. Verify this yourself!

Note: In practice, full independence is rare—models often share data or assumptions and thus may make correlated errors. However, if base models are sufficiently diverse, ensembles can still reduce misclassifications.

Widely used ensemble models include **bagging**, **random forests**, **stacking** and **boosting**.

3.1 Bagging

The term **bagging** stems from "**b**ootstrap **agg**regat**ing**". Bootstrapping is a statistical method that involves resampling data *with replacement* (it is allowed to draw the same data point multiple times) to estimate the sampling distribution of a statistic or the uncertainty of a model.

The bagging procedure can be conceptually explained as consisting of the two steps:

1. **Generate K training sets by sampling with replacement and train K models**. We randomly pick N examples from the training set $\mathcal{D}$, allowing to pick the same examples multiple times. We then run a machine learning algorithm an these N examples and repeat the process K times. This way we obtain K distinct training sets and K corresponding models. In the example from Fig. 9, the training set consists of the data points $\mathbf{x}_i$ from two classes (circles and triangles), $K = 3$ and $N = 7$.

2. **Aggregate the predictions of the K models**: For a new input $\mathbf{x}$, each of the K trained models gives its own prediction $h_i(\mathbf{x})$, $i = 1, \ldots, K$. In classification problems, the different h_i's are aggregated by a voting procedure and in regression problems the average is taken as the final hypothesis:

$$h(\mathbf{x}) = \frac{1}{K} \sum_{i=1}^{K} h_i(\mathbf{x})$$

Fig. 10 illustrates this aggregation for the regression task with $K = 3$ models, continuing on the example from Fig. 9.

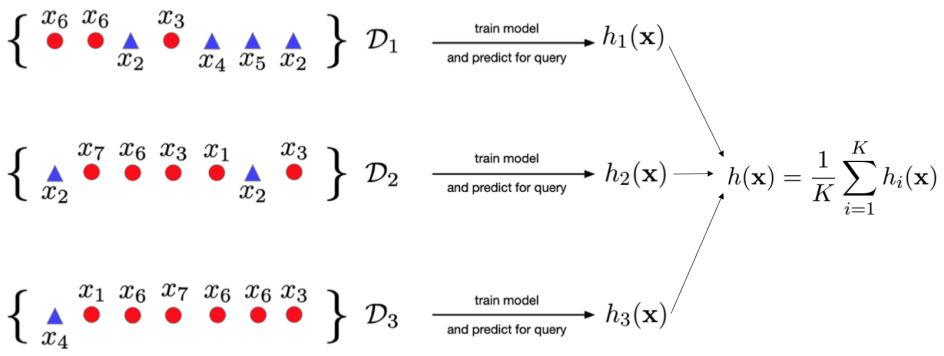

Figure 10: An illustration of sampling with replacement in the bagging procedure.

Bagging lowers variance and is useful when there is limited data or when a model is prone to overfitting. While it can be applied to any model, it is most often used with decision trees due to their sensitivity to small changes in data. Bagging mitigates this instability, and is particularly efficient when computations are done in parallel across multiple machines [6].

3.2 Random forests

Random forests are an enhanced version of decision tree bagging designed to reduce variance and prevent overfitting. Unlike regular bagging, which often leads to correlated trees due to the dominance of certain attributes, random forests introduce additional randomness to create a more diverse ensemble. At each split, a random subset of attributes is considered, and for some variations, like **extremely randomized trees (ExtraTrees)**, the values at each split are also randomly sampled. This added randomness helps ensure that the trees are diverse and reduces the chance of overfitting. Random forests are also computationally efficient, as they can be built in parallel across multiple processors. Furthermore, they do not require pruning, as the ensemble approach naturally mitigates overfitting. Key hyperparameters, such as the number of trees and the number of attributes per split, can be tuned using cross-validation or by measuring out-of-bag error [6].

Despite their complexity, random forests are surprisingly resistant to overfitting, with error rates typically improving as more trees are added to the model. This is because the random selection of attributes leads to trees that cover different areas of the input space, making the model more robust and less likely to be overly sensitive to individual data points. However, while random forests are not immune to overfitting, the model's performance generally stabilizes as more trees are added, and the error does not grow indefinitely. Random forests have found widespread use across a variety of domains, from Kaggle data science competitions to practical applications in finance (credit card default prediction, income prediction) and bioinformatics (diabetic retinopathy, gene expression analysis). Although deep learning is becoming a dominant approach in AI, random forests remain a powerful and versatile machine learning tool offering strong performance across a wide range of tasks. For more in-depth coverage, see [2–4, 6].

3.3 Boosting

Boosting is a powerful ensemble method that improves the performance of "weak learners" by focusing on the training examples that are hardest to classify. It works by assigning weights to

each training example, initially treating all examples equally. After training the first model, the algorithm increases the weights of misclassified examples and decreases the weights of correctly classified ones, encouraging subsequent models to focus on the hard cases. This process continues for a predefined number of iterations, building a sequence of models (hypotheses), each trained on a different distribution of weights. Unlike bagging, boosting is a sequential and greedy algorithm, meaning it cannot parallelize model training. In the final ensemble, each model contributes to the prediction with a weight based on its accuracy:

$$h(\mathbf{x}) = \frac{1}{K} \sum_{i=1}^{K} z_i h_i(\mathbf{x})$$

A widely used variant of boosting is *AdaBoost*, which is especially effective when using simple models as the base models. It is usually applied with decision trees as the component hypotheses. A key theoretical result of *AdaBoost* is that if the base model performs just slightly better than random guessing (a weak learner), then the boosting process can combine them into a **strong learner** that achieves *perfect accuracy on the training set*, for large enough K, regardless of the base model's simplicity or the complexity of the function to be learned [6] Thus the algorithm guarantees to overcome bias in the base model, as long as the base model is better than random guessing.

Boosting demonstrates a counter-intuitive but powerful behavior: even after the ensemble perfectly fits the training data, adding more weak learners can continue to improve test performance. This challenges traditional intuitions like Ockham's razor, which cautions against increasing model complexity unnecessarily. The insight here is that boosting doesn't just fit the data—it refines confidence in predictions, particularly around difficult examples. Theoretical interpretations suggest this may stem from boosting's resemblance to Bayesian learning [6], gradually improving approximation to an optimal classifier. As a result, boosting can generalize better even as the ensemble grows more complex, although the improvements may become smaller or stabilize after a certain number of weak learners.

References

[1] H. Blockeel. *Machine Learning.* KU Leuven.

[2] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[3] Tin Kam Ho. Random decision forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition (ICDAR)*, pages 278–282. IEEE, 1995.

[4] T.K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[5] Robi Polikar. Ensemble learning. In Zhi-Hua Zhou, Thomas G. Dietterich, and Gavin Brown, editors, *Ensemble Machine Learning: Methods and Applications*, pages 1–34. Springer, 2012.

[6] S Russel and Norvig. P. *Artificial Intelligence: A Modern Approach, 4th Edition.* Pearson, 2021.