Machine Learning University, Amazon Fall 2020

Lecture: Decision Trees — Oct. 12, 2020

Tushaar Gangavarapu *Email:* tusgan@amazon.com

1 Lecture Overview

In this lecture, let us turn our attention to the simple yet flexible decision trees classifier. (This lecture recommends that you have some fundamental knowledge of other relevant machine learning algorithms, specifically, the $k-$Nearest Neighbors ($k-$NN) and Support Vector Machines (SVM); while this is not necessary to understand the decision trees classifier, it serves as a base for motivating the need to employ decision trees.) We first explore the region-based and nonlinear nature of decision trees, followed by the approach to construct a tree for a given dataset. Following this, region-based loss functions to assess the quality of split are discussed. Finally, we detail some specific real-world use cases of decision tress, and conclude with the advantages and disadvantages of using such classifiers. Note that this lecture forms the basis for more sophisticated ensemble learning approaches, including bagging and boosting, which are shown to be quite effective [GJC20]; however, these topics remain out of the scope of this lecture.

2 Problem Setting: Supervised Learning

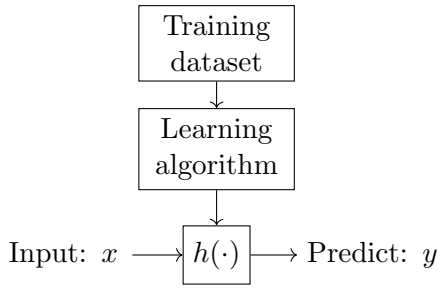


Figure 1: Supervised learning of $h(\cdot)$.

To establish the notation for future use, we will use $x^{(i)}$ to denote the input variables (*features*) and $y^{(i)}$ to denote the output (*target*) variable that we aim at predicting. A pair, $(x^{(i)}, y^{(i)})$ is called a *training example*, and the dataset used to learn to predict is a collection of m such pairs, i.e., $\{(x^{(i)}, y^{(i)}); i = 1, 2, \ldots, m\}$. (The use of superscript "(i)" is simply to denote the index in the training set.) Finally, we will use $\mathbf{X}$ to denote the space of input values (the domain), and $\mathbf{Y}$ to denote the space of output values (the range).

In a supervised learning problem, our goal is to learn a function $h : \mathbf{X} \mapsto \mathbf{Y}$, using the given training dataset, such that $h(x)$ ("the hypothesis") is a *good* estimator (predictor) for the corresponding value of y. Pictorially, this entire process is represented using Figure 1. When the target variable we are trying to predict is continuous, such as the house prices given the square footage, we call the learning to be a *regression* problem. On the other hand, if the target variable can only take-on a small number of discrete values (e.g., if we wanted to predict, for a given living area, if the dwelling is a house or an apartment (only two output values)), we call the problem, a *classification* problem.

3 Accommodating Nonlinearity in the Data

Transitioning from generalized linear models and vanilla (linear kernel) SVM, $k-$NN and **decision trees** are two of the most fundamental algorithms in the class of *nonlinear* machine learning approaches. Recall that a machine learning model is said to be linear, if the hypothesis function is

of the form ($x_0 = 1$):

$$h(x) = h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n = \sum_{j=0}^{n} \theta_j x_j = \begin{bmatrix} \theta_0 & \theta_1 & \cdots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x \qquad (1)$$

If a hypothesis function cannot be reduced to the form in (1), then such a hypothesis is regarded "nonlinear." Now is a good time to pause and ponder about why logistic regression is a linear model, despite the sigmoid function being nonlinear; also, how does it compare to feed-forward neural networks with one or more hidden layers, activated by a sigmoid function (*hint*: think in terms of the decision boundaries). Proceeding, you may also recall that the kernelization trick employed in SVM is one way to obtain nonlinear hypothesis functions. Decision trees are one such class of algorithms that can generate nonlinear hypothesis functions, without the need for feature mapping as is the case with kernelization.

3.1 Motivation for Decision Trees: From $k-$NN to Decision Trees

Data has some inherent structure. In the $k-$NN algorithm, it is assumed that that similar inputs have similar outputs, i.e., data points are not sprinkled in space at random, but are aligned in clusters of more-or-less homogeneous class alignments. For the sake of this discussion, let us now assume that the value of k chosen is quite large (in the order of 10^6). Now, rather than computing the distance between the test subject and these 10^6 data points, if you (somehow) knew that these 10^6 data points (and the test point) form a cluster of say, positive samples, then it is apparent that the test subject is also positive. Notice how it is less relevant for us to know the exact identity of the test point, than just the knowledge of whether or not the test point falls into the cluster of positive samples. This reasoning serves as the basis for decision trees classifier.

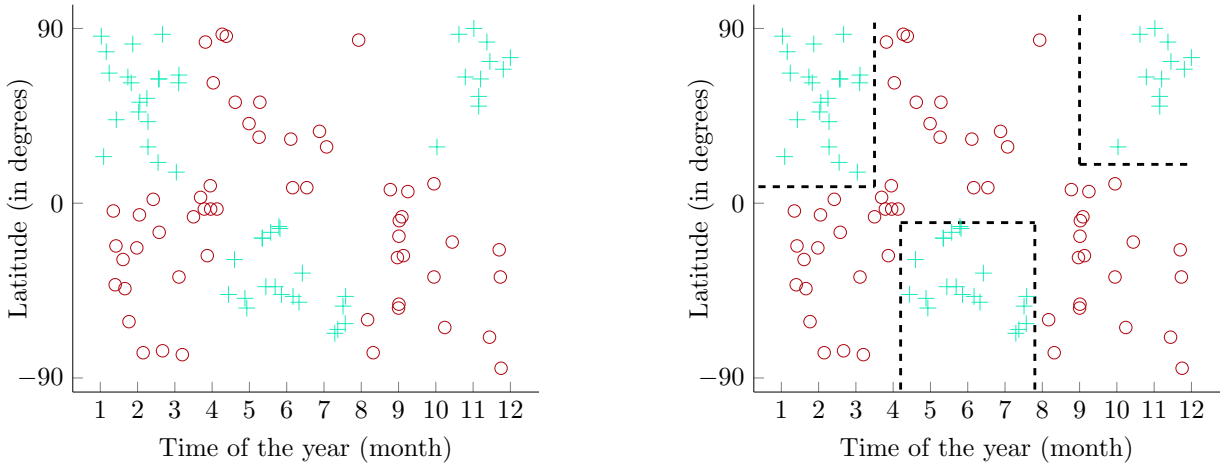


Figure 2: A toy classification dataset (left; adapted from Raphael John Lamarre Townshend [Tow19]) indicating whether or not it is possible to ski at a given location, during a given time of the year. For the sake of simplicity, time is represented as *month* of the year, and the location is represented as the *latitude* ($-90°$, $0°$, and $90°$ represent the south pole, equator, and north pole respectively).

Figure 2 (left) shows a toy dataset of whether or not it is possible to ski at a given location (expressed as latitude), during a given time of the year (represented as month of the year). It is quite evident

that there is no linear boundary separating the areas of positive and negative instances in the skiing dataset. However, it can be observed that the positive and negative areas can be isolated (recall the cluster motivation introduced earlier), and one such possibility is shown in Figure 2 (right). Hence, to build a classifier that can predict whether to ski or not, for a given location and time (i.e., determine the decision boundary), we need to be able to partition the input space $\mathbf{X}$ into non-overlapping regions of interest. Simply put,

$$\mathbf{X} = \bigcup_{j=1}^{r} \mathtt{R}^{(j)}, \; r \in \mathbb{Z}^{+} \tag{2}$$

At this point, it is quite natural to be inquisitive of the ways to split $\mathbf{X}$, i.e., partition it into r non-overlapping $\mathtt{R}^{(j)}$s[1]. Following Occam's razor[2] [BEHW87, Definition 2.2], we want to divide our input space such that our regions are maximally compact, i.e., the most compact decision boundary to separate the positive and negative samples. However, through reduction (exact cover by 3-sets $\propto$ maximal compact boundaries), this is shown to be an NP-hard problem [LR76].

3.2 Greedy Strategy to Choosing Regions

As discussed, selecting the optimal solutions is intractable; however, decision trees approximate it via *greedy*, *top-down* (since we start with the entire input space $\mathbf{X}$ and break it down into regions of interest), *recursive* partitioning. Imagine the construction of decision trees as the task of asking sufficient number of questions, till you can partition up the entire space into homogeneous (with respect to the target classes) clusters. Simply put, we partition the input space $\mathbf{X}$ based on some threshold t[3] $\in \mathbb{R}$ on a single attribute x_s. Each of the thus obtained child regions are partitioned using a new threshold t' on another attribute x'_s, and this process continues recursively until the leaf node is class homogeneous (pure). Mathematically, given a parent region $\mathtt{R}^{(\text{parent})}$, a feature index s, and a threshold t on the attribute s, we obtain the regions of interest as:

$$\begin{aligned} \text{split}(s,t) = \Big(\mathtt{R}^{(\text{parent.1})} &= \left\{ x^{(i)} \mid x_s^{(i)} < t, x^{(i)} \in \mathtt{R}^{(\text{parent})} \right\}, \\ \mathtt{R}^{(\text{parent.2})} &= \left\{ x^{(i)} \mid x_s^{(i)} \geq t, x^{(i)} \in \mathtt{R}^{(\text{parent})} \right\} \Big) \end{aligned} \tag{3}$$

With our toy skiing dataset, in the first step (a) would be to partition the input space $\mathbf{X}$ by the latitude, threshold at 10°, into $\mathtt{R}^{(\mathbf{X}.1)}$ and $\mathtt{R}^{(\mathbf{X}.2)}$. Next, let us split our region $\mathtt{R}^{(\mathbf{X}.2)}$ using the time threshold of 3.5, into two regions $\mathtt{R}^{(\mathbf{X}.2.1)}$ (**homogeneous** cluster) and $\mathtt{R}^{(\mathbf{X}.2.2)}$. Following this, let us split $\mathtt{R}^{(\mathbf{X}.1)}$ using a time threshold of 8 (see Figure 3). We continue to split the regions recursively until all the leaf nodes are pure (this need not always be the case, we could instead consider the majority vote, just like in $k-$NN; but more on this later). One such partitioning of the skiing dataset set is depicted in Figure 4 (notice the nonlinear decision boundaries). Finally, at the time of testing, we only need to traverse the tree from root to leaf nodes in a way that satisfies

[1] In the context of decision trees, the terms *cluster*, *area*, and *region* are used interchangeably.

[2] Occam's razor or the law of parsimony of explanations, popularly known as the law of economy, states that the entities are not to be multiplied beyond necessity, i.e., when you have multiple competing theories that make the exact same predictions, the simpler one among them is the better solution.

[3] Notice the use of t to represent the threshold, as opposed to a more conventional θ notation. This is to emphasize the fact that the threshold t need not be a numerical value (it is numerical when dealing with continuous attributes; e.g., time in our skiing dataset); for instance, splitting a binary attribute could have t set to '*yes*.' Also, it is worth mentioning that the threshold need not necessarily result in a binary split at the chosen attribute.

all the decisions on the internal nodes (time complexity: $[O(\log_2 m); O(m)]$), rather than storing the entire training data as is done in $k-$NN—hence, decision trees are quite fast while testing.

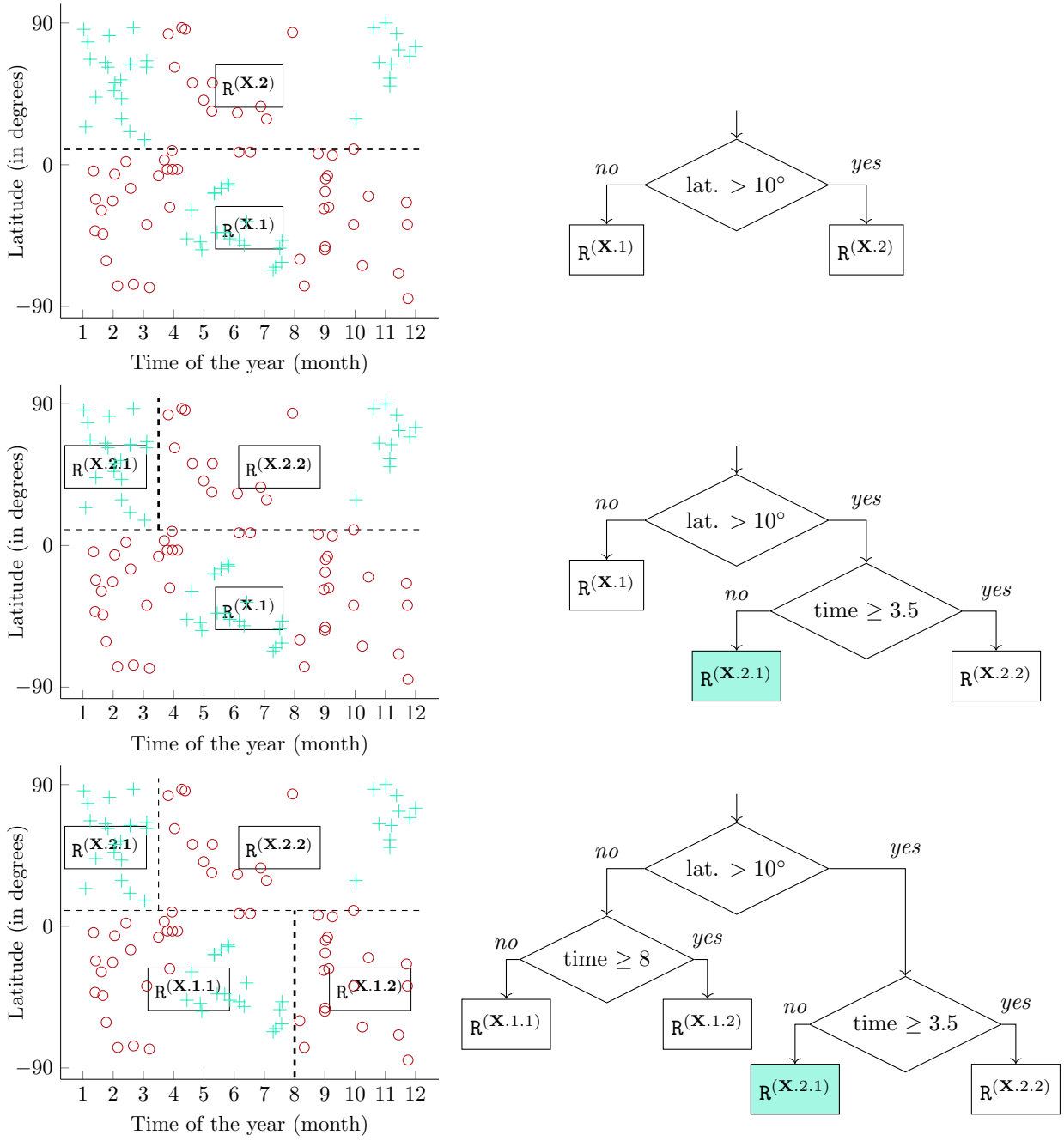
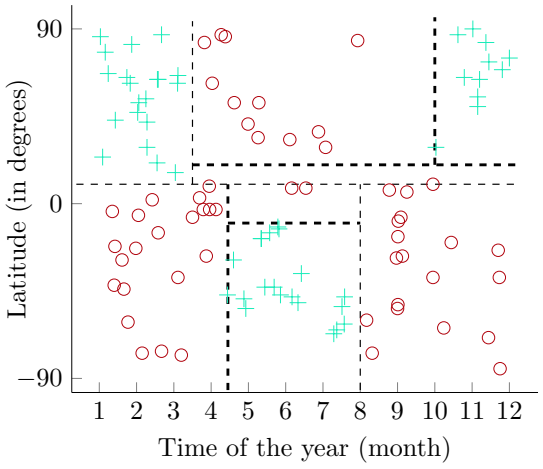


Figure 3: Top-down, recursive approach to partitioning the skiing dataset input space (see Figure 2 (left)) to be able to segregate positive samples from negative samples. The recursive partitioning is shown to the left, while the actual decision tree is depicted to the right (pure nodes are marked in green).

Before proceeding further, let us attempt to answer if it is always possible to obtain pure leaf nodes. Consider two training data points in the dataset such that they have identical features but different target labels, i.e., $(x^{(i)}, +)$ and $(x^{(j)} = x^{(i)}, -)$. Since the base assumption in decision trees

classifier is that similar data points have similar outputs (see § 3.1), it is not possible to build a consistent tree in such a scenario, in which case we resort to a majority vote. Note that we do not stop splitting recursively unless we have a pure node (which becomes a leaf), or we have no more attributes left to split the node on. So, why don't we stop if no split improves the impurity? (*hint*: think of a decision tree for XOR function).




```
find_threshold(x_j: cont., J(·): loss)
    x_j ← sort(x_j)
    min_loss ← ∞
    for each x_j^(i):
        threshold = (x_j^(i) + x_j^(i+1))/2
        R^(p.1), R^(p.2) ← split(j, t)
        loss ← J(R^(p.1), R^(p.2))
        if loss < min_loss:
            min_loss ← loss
            best_threshold ← threshold
    return best_threshold
```

Figure 4: Possible decision boundaries for skiing dataset.

If you observe carefully, the split threshold for discrete attributes seems to fall out naturally, i.e., the attribute is inherently split by the set of possible values it can take. For instance, if we were looking at the data that distinguishes vampires from normal people, we could look at the shadow; shadow is a binary attribute, with *yes* (for normal people) or *no* (for vampires) values. But with a continuous attribute, such as the ones in our skiing dataset, it is natural to ask about the ways of choosing the split threshold. One way of achieving this is to discretize (a.k.a. binning) the continuous attribute in advance, and choose the best threshold that minimizes the loss defined by the chosen loss function ($J(.)$; discussed in § 4).

Notes. Notice that, while performing discretization (as shown above), we can save some computation time by only the checking the split points that lie between the training examples of different classes, because only these splits are greedily optimal.

4 Region-based Loss Functions

The next question to ask would be on how to choose the best attribute at each step (being greedy). Recall by Occam's razor (see § 3.1) that our objective is obtain the smallest decision tree that separates the positive and negative samples (in case of a binary class problem). Essentially, we want to choose the attribute s (and threshold t in case of continuous attributes) such that we maximize the *decrease in loss*, i.e., $\arg\max_{s,t} \{J(\text{parent}) - J(\text{children})\}$:

$$\arg\max_{s,t} \left\{ J(\mathtt{R}^{(\text{parent})}) - \left(\frac{|\mathtt{R}^{(\text{parent.1})}|}{|\mathtt{R}^{(\text{parent})}|} J(\mathtt{R}^{(\text{parent.1})}) + \frac{|\mathtt{R}^{(\text{parent.2})}|}{|\mathtt{R}^{(\text{parent})}|} J(\mathtt{R}^{(\text{parent.2})}) \right) \right\} \tag{4}$$

Notice that in (4), the children node losses are cardinality-weighted. Since the value of $J(\mathtt{R}^{(\text{parent})})$

is fixed, it is apparent that our objective is to select the attribute (and threshold) that minimize the cardinality-weighted children loss ($L = J(\text{children})$; $\arg\min_{s,t} L$). The next thing to do would be to define a suitable loss function $J(\cdot)$.

4.1 Measuring the Misclassification Error

For a classification problem, we are interested in the misclassification loss $J_{\text{misclass}}(\cdot)$. Misclassification loss assumes the majority vote in a given region $\mathtt{R}^{(j)}$ and computes the fraction of misclassified samples. Given C target classes, let us define $p_c^{(j)}$ as the proportion of samples of a class $c \in C$, in $\mathtt{R}^{(j)}$. Hence, the misclassification loss can be computed as:

$$J_{\text{misclass}}(\mathtt{R}^{(j)}) = 1 - \max_{c \in C}(p_c^{(j)}) \tag{5}$$

While misclassification loss is simple and tries to handle our greedy expectations, it is not very sensitive to changes in class probabilities. As an example, consider the two decision trees shown in Figure 5:

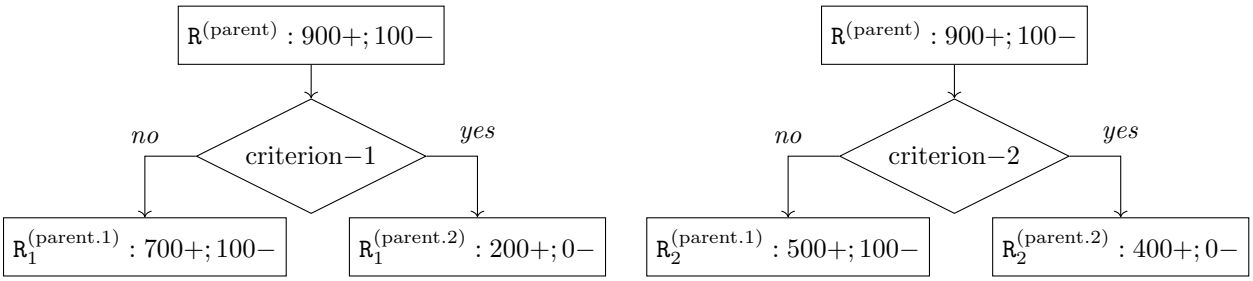


Figure 5: A representative example depicting the caveats in misclassification loss.

It is arguable that the right decision boundaries are better are the left one in Figure 5, since the right boundary is able to isolate more positive samples comparatively. However, if we compute the value of $L = J_{\text{misclassify}}(\text{children})$, we have:

$$J_{\text{misclass}}(\mathtt{R}^{(parent)}) = \frac{|\mathtt{R}_1^{(\text{parent.1})}|}{|\mathtt{R}^{(\text{parent})}|} J_{\text{misclass}}(\mathtt{R}_1^{(parent.1)}) = \frac{|\mathtt{R}_2^{(\text{parent.1})}|}{|\mathtt{R}^{(\text{parent})}|} J_{\text{misclass}}(\mathtt{R}_2^{(parent.1)}) = \frac{100}{1000}$$

$$\text{Also, } \frac{|\mathtt{R}_1^{(\text{parent.2})}|}{|\mathtt{R}^{(\text{parent})}|} J_{\text{misclass}}(\mathtt{R}_1^{(parent.2)}) = \frac{|\mathtt{R}_2^{(\text{parent.2})}|}{|\mathtt{R}^{(\text{parent})}|} J_{\text{misclass}}(\mathtt{R}_2^{(parent.2)}) = 0$$

$$\implies J_{\text{misclass}}(\mathtt{R}^{(parent)}) = L_{\text{criterion-1}} = L_{\text{criterion-2}} = 0.1$$

Therefore, the misclassification loss in this scenario leads one to believe that not only are the two splits (criterion−1 and criterion−2) identical through the lens of greedy tree building, but also that neither of these splits reduce the loss over parent region.

4.2 Measuring the Disorder: Cross-Entropy Loss

Now that we realize that the misclassification loss is not the most informative indicator, let us define a more sensitive, cross-entropy loss for region $\mathtt{R}^{(j)}$ containing data samples of C classes, $J_{\text{ent}}(\cdot)$ as follows:

$$J_{\text{ent}}(\mathtt{R}^{(j)}) = -\sum_{c \in C} p_c^{(j)} \log_2 p_c^{(j)} \tag{6}$$

The cross-entropy loss, or simply known as the 'entropy,' is derived from information theory, and it measures the number of shannons or bits (if the base of logarithm in (6) is 2, which is usually the case; nats if it is e) needed to specify the outcome (class), given that the distribution of $p_c^{(j)}$s is known. One can also think of entropy as the randomness or disorder in the region, and this view is borrowed from thermodynamics. Now is be a good time to indicate that the decrease in loss from the parent region to children nodes (see (4)) is often termed as the **information gain**. Observe that when $p_c^{(j)} = 0$, we have $p_c^{(j)} \log_2 p_c^{(j)} = 0$, a direct implication of the L'Hospital's rule for evaluating limits of indeterminate forms [L'H96].

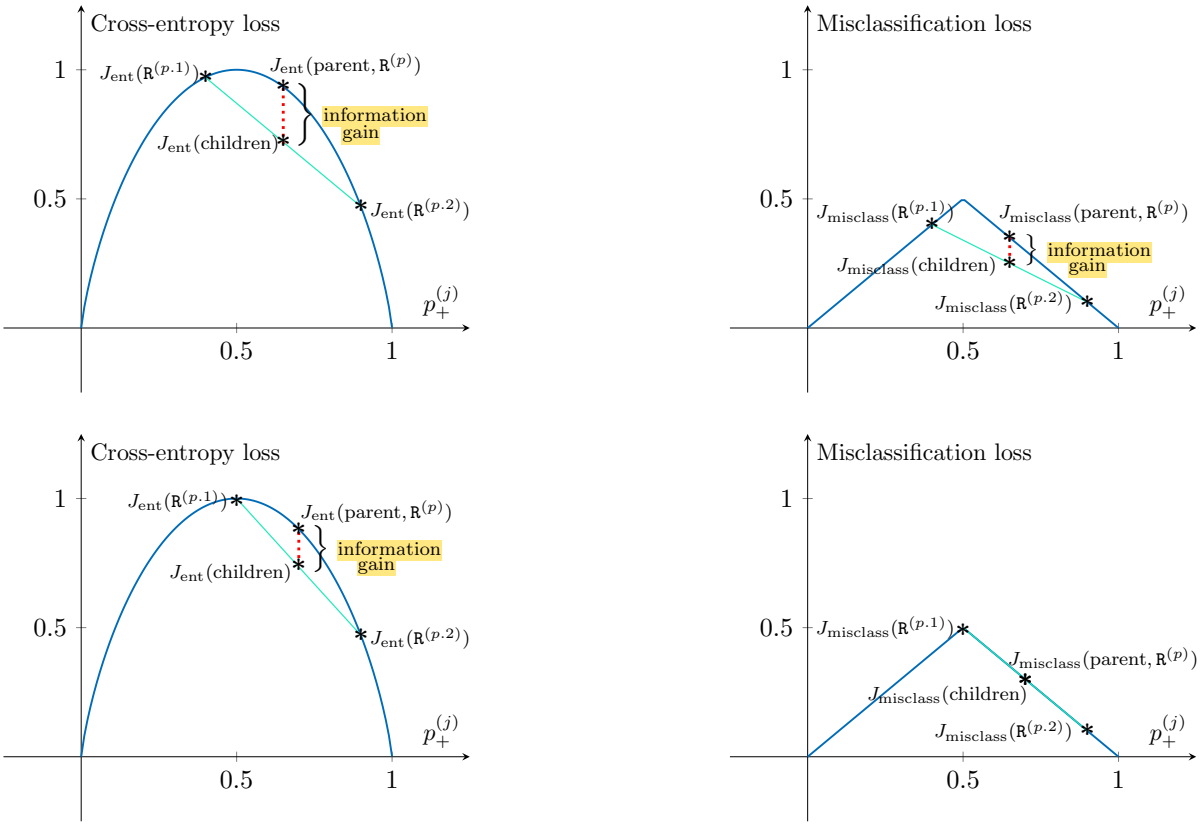


Figure 6: Cross-entropy loss (left) vs. misclassification loss (right) for a region $\mathtt{R}^{(p)}$ with + and − samples (binary), split into two regions $\mathtt{R}^{(p.1)}$ and $\mathtt{R}^{(p.2)}$. Notice that the information gain by misclassification loss is zero in the second case (bottom-right). For mathematical convenience, the graphs shown are plotted with the assumption of an even split of data between $\mathtt{R}^{(p.1)}$ and $\mathtt{R}^{(p.2)}$, but this is not a necessary condition, and the point being made here can be generalized for any split structure.

To emphasize on the relative sensitivity of the cross-entropy loss to misclassification loss, let us look at the loss functions geometrically, for a binary classification scenario. In the case of a binary classification, we have the cross-entropy and misclassification loss functions reduced as follows (from (5) and (6)):

$$J_{\text{binary-ent}}(\mathtt{R}^{(j)}) = -\left(p_+^{(j)} \log_2 p_+^{(j)} + (1 - p_+^{(j)}) \log_2 (1 - p_+^{(j)})\right) \tag{7}$$

$$J_{\text{binary-misclass}}(\mathtt{R}^{(j)}) = 1 - \max(p_+^{(j)}, 1 - p_+^{(j)}) \tag{8}$$

These functions are plotted in Figure 6. it is interesting to note that the (binary) cross-entropy loss

function is strictly concave[4], while the binary misclassification loss function is just concave[4]. Owing to this very property of the loss functions, it can be easily proven (and verified from the graphs in Figure 6) that for any two distinct values of $p_+^{(p.1)}$ and $p_+^{(p.2)}$ (both non-empty) obtained from splitting $\mathtt{R}^{(p)}$, the cardinality-weighted sum of the cross-entropy losses of $p_+^{(p.1)}$ and $p_+^{(p.2)}$ will always be less than that of the parent. However, the same is not true for misclassification loss, owing to it being not strictly concave (see the bottom-right graph of Figure 6 for a visual depiction). Due to this added sensitivity of cross-entropy loss, it is often employed in building the decision tree for classification.

4.3 Measuring the Misclassification Probability: Gini Impurity

Another popular approach to accommodate the sensitivity would be measure the probability of misclassification in a given region $\mathtt{R}^{(j)}$, and is called the Gini impurity of that region. The Gini impurity of $\mathtt{R}^{(j)}$, with data samples of C classes, $J_{\text{gini}}(\cdot)$ is as follows:

$$J_{\text{gini}}(\mathtt{R}^{(j)}) = \sum_{c \in C} p_c^{(j)} \left(1 - p_c^{(j)}\right) \tag{9}$$

For an intuitive understanding of the probability of misclassification, let us consider an example of a binary classification problem. The probability of misclassification of $+$ class involves choosing a $+$ sample (p_+) and classifying it as $-$ ($p_- = 1 - p_+$), i.e., $p_+(1 - p_+)$ (do the same for the negative class as well). Thus, we can reduce (9) to fit our binary classification case (plotted in Figure 7) as follows:

$$J_{\text{binary-gini}}(\mathtt{R}^{(j)}) = 2p_+^{(j)} \left(1 - p_+^{(j)}\right) \tag{10}$$

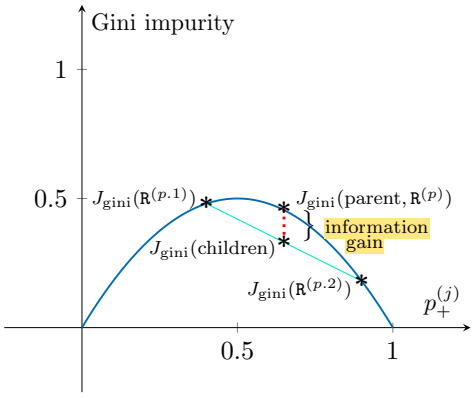


Figure 7: Gini impurity of a region $\mathtt{R}^{(p)}$ with samples of $+$ and $-$ classes.

It can be seen from Figure 7 that even Gini impurity is a strictly concave function. Therefore, similar to cross-entropy loss, even Gini impurity is sensitive to changes in class probabilities, and hence is often used in growing classification decision trees.

Notes. Observe that cross-entropy loss (6), when compared to Gini impurity (9), requires the computation of $\log_2 p_c^{(j)}$. It is quite well-known that computing the logarithmic value (usually done using Taylor series combined with Remez algorithm [Rem34a, Rem34b, Rem34c]) is expensive, and hence, Gini impurity is computationally faster.

5 Extensions on Decision Trees

The processes to build a decision trees classifier explained so far are indeed the ones employed in the Iterative Dichotomiser (ID3) algorithm [Qui86], one of the first decision trees algorithms developed.

[4]Intuitively, a real-valued function defined on an $n-$dimensional interval is said to be (strictly) *concave* if the line segment between any two points on the graph of the function lies (strictly) below the graph between the two points. Refer to the bottom two graphs of Figure 6, the left function is strictly concave, while the right one is just concave (follow the **green** line segment).

The ID3 algorithm creates a multiway tree, greedily finding the *categorical* attribute for each node that yields the largest information gain. Over the years, there have been significant modifications to overcome the limitations of ID3 algorithm, and we will explore a few of those within the contents of this section.

5.1 Decision Trees for Regression

Let us revisit the loss functions in § 4; notice that the loss functions are defines with respect to a supervised classification problem. Let us briefly cover the regression setting for decision trees (regression trees). For each data point $x^{(i)}$, we now have to predict $y^{(i)} \in \mathbb{R}$, rather than a target class label. Let us consider a toy dataset (as shown in Figure 8 (left)) that aims at predicting the amount of snowfall (in inches), given the location and time of the year.

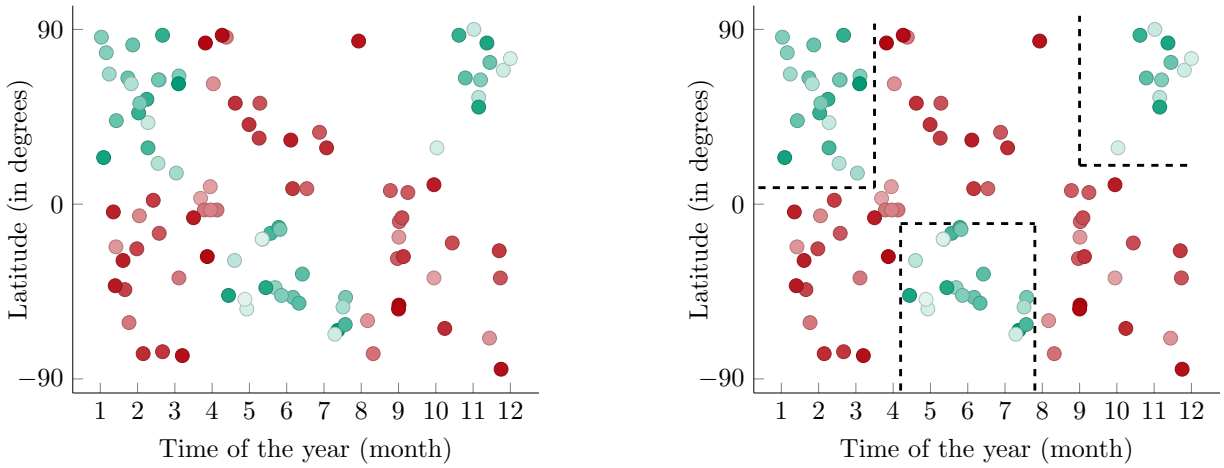


Figure 8: A toy regression dataset (left) indicating the amount of snowfall in inches (color scale: **red** (min) to **green** (max)) at a given location, during a given time of the year. For the sake of simplicity, time is represented as *month* of the year, and the location is represented as the *latitude* ($-90°$, $0°$, and $90°$ represent the south pole, equator, and north pole respectively).

Most of the tree building processes remain the same those employed for classification trees. However, rather than predicting the majority vote, the final prediction for a region $\mathtt{R}^{(j)}$ is now the mean of the output values of data samples in that region. Simply put,

$$y^{(j)} = \frac{1}{|\mathtt{R}^{(j)}|} \sum_{i \in \mathtt{R}^{(j)}} y^{(i)} \tag{11}$$

And, we can employ the squared loss as the loss function in choosing the best split attribute and an optimal threshold:

$$J_{\text{squared}}(\mathtt{R}^{(j)}) = \frac{1}{|\mathtt{R}^{(j)}|} \sum_{i \in \mathtt{R}^{(j)}} \left(y^{(i)} - y^{(j)} \right)^2 \tag{12}$$

The idea of regression trees developed with the Classification And Regression Trees (CART) algorithm [BFOS84]. CART algorithm constructs **binary trees** using the attribute and threshold that yield the largest information gain.

5.2 Categorical Attributes

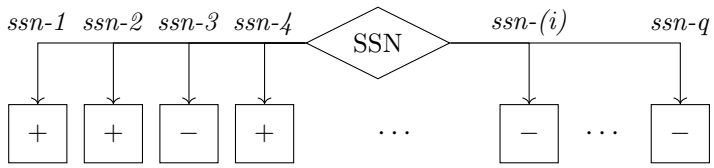


Figure 9: Decision tree stump (one level decision tree) for the Social Security Number (SSN) attribute.

We have already shown that decision trees can deal with categorical attributes fairly easily (only the split attribute is to be determined, the threshold falls out naturally). Now, consider a categorical attribute with a large number of values, for example, the social security number (decision stump is shown in Figure 9). What would happen if we end up splitting on such attributes? Since each obtained one-sample partition is a pure node, the value of $J(\text{children}) = 0$. Hence, we obtain maximum information gain by splitting on such attributes ($= J(\text{parent})$). Contrary to what is indicated by the information gain metric, it is quite evident that we need to ask 2^q yes-no questions at such a node to proceed any further. Furthermore, notice that the use of a highly-branching attribute with a large number of categories to split the input region results in a high degree of overfitting (high model variance).

One simple approach to solving this q categories problem is to convert such categorical attributes into numerical attributes. Replace the categorical feature (x_j or region $\mathtt{R}^{(j)}$) with a new feature that measures the value of class proportion at $\mathtt{R}^{(j)}$, i.e., $p^{(j)}_{y^{(i)}} \forall i = 1, 2, \ldots, m$. Now if we split the data using on the new feature, the child nodes will have enough points to avoid overfitting.

5.3 From Information Gain to Gain Ratio

Another solution to our highly-branching attribute problem involves modification of information gain to reduce its bias towards such attributes. It takes into account the intrinsic information of a split into account, i.e, the number and size of branches resulting from the split. Formally, with a split attribute s and threshold t, the gain ratio is expressed as:

$$\text{gain-ratio}(s,t) = \frac{\text{information-gain}(s,t)}{\text{intrinsic-info}(s,t)}; \text{intrinsic-info}(s,t) = -\sum_i \frac{|\mathtt{R}^{(s.i)}|}{|\mathtt{R}^{(s)}|} \log_2 \frac{|\mathtt{R}^{(s.i)}|}{|\mathtt{R}^{(s)}|} \tag{13}$$

Let us revisit our social security number example depicted in Figure 9, with q distinct values (one-sample partitions). The intrinsic information of the attribute would be $-q\frac{1}{q}\log_2\frac{1}{q} = -\log_2\frac{1}{q}$. Hence, the gain ratio would be $\frac{J(\text{parent})}{\log_2 q}$. It is interesting to note that the intrinsic information measures the number of shannons needed to determine the branch corresponding to an instance, given the cardinality distribution of the region.

While the gain ratio is an effective method to handling attributes with large number of values, it can overcompensate, i.e., choose attributes whose intrinsic information is relative small. Also, note that even with gain ratio, it is quite possible that highly-branching attributes still rank on the top because of their large information gain value; hence, it is advisable to perform some sort of an ad hoc test to avoid splitting on such attributes. More often than not, information gain and gain ratio are used together, to reap the benefits of both these measures.

C4.5 decision trees algorithm [Qui96], the successor of the ID3 algorithm employs gain ratio as opposed to information gain, and handles both numeric and categorical attributes.

5.4 Regularization of Decision Trees

It must be quite evident by now that decision trees are low bias[5] high variance[6] models. One way to think about this is to modify the stopping criteria for growing a tree from purity of a node, to *full growth*, i.e., each leaf node must contain exactly one training data sample—now it becomes easier to understand why decision trees are fairly high variance models. Some of the popular stopping heuristics to reduce the variance of decision tress include:

- *Threshold on the leaf size*: Stop splitting a region if the number of data samples in the region has reached a minimum threshold.
- *Threshold on the number of nodes in the tree*: Terminate the tree splitting if the number of leaf nodes exceeds a certain maximum threshold.
- *Enforce a maximum depth of the tree*: Decide to split a region based on the number of splits already taken to reach that region.

A **misleading** heuristic is to threshold on the obtained information gain after splits. Given that our approach is greedy, this is a problematic heuristic, especially if we have higher-order interactions between variables. We might have to ask a few non-optimal questions, and the follow-up questions combined with those initial questions might give us a better gain. Going back to our skiing dataset example, from Figure 3, we observe that the first and third steps do not produce much gain (at least not as much as that compared to the second step); however, they are crucial in achieving the final decision boundaries shown in Figure 4. (You could also think of the XOR function decision tree discussion, introduced earlier in § 3.2.) Hence, by just looking at it from the gain perspective, we might stop prematurely. A better approach to using information gain threshold is to grow out the entire tree during training and then prune away the nodes that provide minimal information gain, as measured on a validation set.

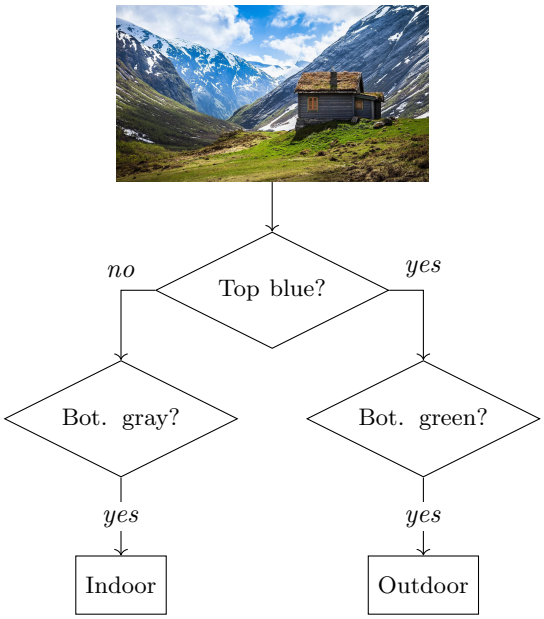


Figure 10: An illustrative example depicting a decision tree to differentiate indoor and outdoor scenes, adapted from Criminisi *et al.* [CSK11].

6 Specific Use Cases

The popularity of decision trees can be attributed to their high degree of interpretability, i.e., looking at the set of generated thresholds, we can understand and explain as to why the model made specific predictions (unlike with neural networks). Additionally, decision trees handle categorical attributes rather elegantly, and are quite fast both during training and testing. The following are some use cases of decision trees (or an extension of them):

[5]The bias of a machine learning model is the difference between the expected outputs and model predictions. A model with high bias does not pay attention to the training data, thus failing to learn and generalize.

[6]The variance of a machine learning model is the variability in the model prediction for different data points. A model with high variance pays a lot of attention to training data, failing to generalize on unseen data.

- Decision trees had been employed to autopilot an aircraft on a plane simulator by merely learning from the logs of human experts flying the simulator [SHKM92].
- Credit card companies employ decision trees to determine whether or not a loan can be granted to a customer [CFPS99].
- Since decision trees are quite intuitive to understand and explain, they often mimic the way a doctor thinks, when trained on a medical dataset. Decision trees have been used to analyze caesarean section risk [SMC+00].
- Elementary image classification (an illustrative example is presented in Figure 10) [CSK11] and posture detection (*further reading*: usually done using a number of small decision trees, a.k.a random forests) [SFC+11].

7 Concluding Remarks

Before closing in on the final thoughts about decision trees, let us consider to see what happens when we fit a decision trees classifier to linearly separable (additive) data. A toy additive dataset with the decision boundary of the form: $x_j + x_k$ is shown in Figure 11. It can be seen that decision trees can only 'approximate' the decision boundary through the use of multiple splits, since each split can only consider one attribute at a time (**lack of an additive structure**). Certain approaches have been devised to develop solutions that factor in the additive structure, by considering multiple features at once; however, the use of multiple features often leads to reduced interpretability and increased model variance.

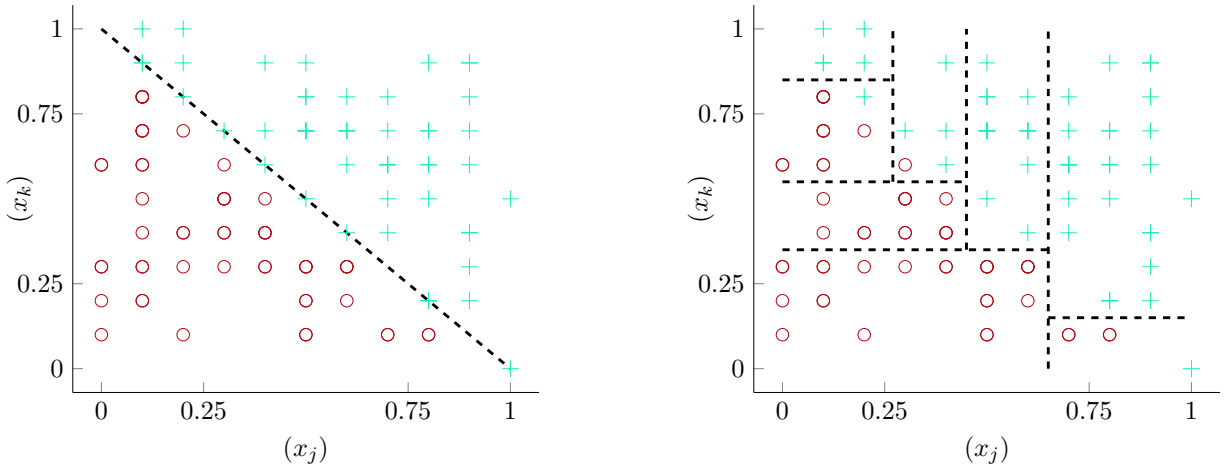


Figure 11: A toy additive (linearly separable) dataset fit using a regression line (left) and a decision tree (right). Notice that decision trees need to ask a number of questions to fit the linear decision boundary, which is rather easily obtained with a linear model.

To recap, some of the primary benefits of employing decision trees include: a) ease of interpretability and explainability, b) speed of training and testing, and c) support for categorical variables, among others. However, these classifiers often lead to low bias, high variance models, which fail to generalize on unseen data. Finally, as discussed earlier, fitting decision trees to model linearly separable data requires a lot of splits, since attributes are considered one at a time.

In reality, due to their non-additive nature and high variance, decision trees often result in low

predictive accuracy. A successful and commonly employed approach to address these shortcomings is to use ensemble learners of decision trees (bagging and boosting). So, why did we cover decision trees? While decision trees may not provide the best predictive performance, they do provide an ideal framework to examine ensemble learners, which have been shown to provide promising results in various applications.

References

[BEHW87] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Occam's razor. *Information processing letters*, 24(6):377–380, 1987. 3

[BFOS84] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. Classification and regression trees. belmont, ca: Wadsworth. *International Group*, 432:151–166, 1984. 9

[CFPS99] Philip K Chan, Wei Fan, Andreas L Prodromidis, and Salvatore J Stolfo. Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems and Their Applications*, 14(6):67–74, 1999. 12

[CSK11] Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. *Microsoft Research Cambridge, Tech. Rep. MSRTR-2011-114*, 5(6):12, 2011. 11, 12

[GJC20] Tushaar Gangavarapu, CD Jaidhar, and Bhabesh Chanduka. Applicability of machine learning in spam and phishing email filtering: review and approaches. *Artificial Intelligence Review*, pages 1–63, 2020. 1

[L'H96] L'Hospital, Guillaume-François-Antoine de. de l'analyse des infiniment petits, pour l'intelligence des lignes courbes. pages 145–146, 1696. 7

[LR76] Hyafil Laurent and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976. 3

[Qui86] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986. 8

[Qui96] J. Ross Quinlan. Improved use of continuous attributes in c4.5. *Journal of artificial intelligence research*, 4:77–90, 1996. 10

[Rem34a] Eugene Remes. Sur le calcul effectif des polynomes d'approximation de tchebichef. *CR Acad. Sci. Paris*, 199:337–340, 1934. 8

[Rem34b] Eugene Remes. Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation. *CR Acad. Sci. Paris*, 198:2063–2065, 1934. 8

[Rem34c] Eugene Y Remez. Sur la détermination des polynômes d'approximation de degré donnée. *Comm. Soc. Math. Kharkov*, 10(196):41–63, 1934. 8

[SFC+11] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In *CVPR 2011*, pages 1297–1304. Ieee, 2011. 12

[SHKM92] Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. Learning to fly. In

Machine Learning Proceedings 1992, pages 385–393. Elsevier, 1992. 12

[SMC+00] Cynthia J Sims, Leslie Meyn, Rich Caruana, R Bharat Rao, Tom Mitchell, and Marijane Krohn. Predicting cesarean delivery with decision tree models. *American journal of obstetrics and gynecology*, 183(5):1198–1206, 2000. 12

[Tow19] Raphael John Lamarre Townshend. Decision trees. pages 2–3, 2019. 2