Lecture 18: (Closing remarks on ERM and) Decision trees

CS 3780/5780, Sp25
Tushaar Gangavarapu (TG352@cornell.edu)

While this lecture mainly focuses on introducing decision trees, we will first conclude our discussion on empirical risk minimization (ERM) with some practical advice on training learning algorithms.

1 Closing remarks on ERM

In the past two lectures, we realized the framework of ERM, as way to find the optimal hypothesis in a given class of hypotheses, $\mathcal{H}$, that minimizes the training error. In bounding the generalization error of the ERM-chosen hypothesis $\hat{h}$, we noted the tradeoff between the complexity of $\mathcal{H}$ (in our discussions, we noted this to be $k = |\mathcal{H}|$) and the generalization error of the best-in-class hypothesis $h^\star$. To recall, we had, with probability $1 - \delta$:

$$\varepsilon(\hat{h}) \leq \varepsilon(h^\star) + 2\sqrt{\frac{1}{2n}\log\frac{2k}{\delta}}.$$ [1]

We furthered this understanding of there being some inherent tradeoff by decomposing the generalization error into "bias" and "variance." Informally, bias is the generalization error that occurs even when there is no noise and we have an infinitely large dataset; think: linear model fitting quadratic data. On the other hand, variance can be characterized as the variation across models learned on different training datasets drawn from the same underlying distribution; think: $n - 1$-th order polynomial fitted to n points.

Complexity-regularized ERM. Hence, our goal is to find a $\hat{h} \in \mathcal{H}$ such that

$$\hat{h} = \arg\min_{h \in \mathcal{H}} \hat{\varepsilon}(h) + \lambda\Omega(h),$$

where $\hat{\varepsilon}(h)$ is the training error of h, $\Omega(h)$ measures the complexity of h (e.g., norm of weights in regularized least squares), and $\lambda > 0$ is the regularization (hyper)parameter. Just to connect the dots, viewing from a Bayesian standpoint, you may realize the above objective to be resultant of MAP estimation.

1.1 Debugging learning algorithms

As a motivating example, consider building a ChatGPT detector using carefully-chosen 100 words (say) as features[2] and a regularized logistic regression model with gradient descent. Now, at the end of your training cycle, if you have 30% validation error (unacceptably high), what can you do to improve? Several strategies to improve performance:

- Try to obtain more training data
- Too many features—use a smaller set of features, or contrarily, suspect that your features are not "good" enough, and you obtain a larger feature set
- Run gradient descent for more iterations
- Try using a different optimizer?—second-order: Newton, momentum, adaptive gradients with momentum, etc.

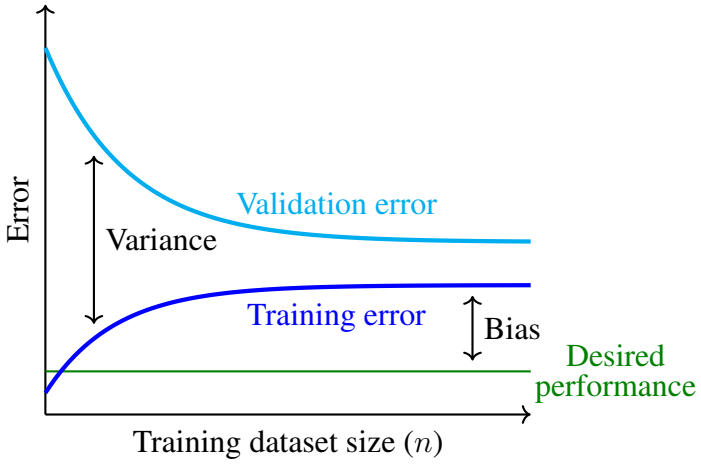
[1] See Lecture 16, §2.2: "Relating training error to generalization error" for specifics.

[2] https://reddit.com/r/ChatGPT/comments/15zgl4y/what_are_the_most_common_words_chatgpt_says.

- Use a different regularization hyperparameter λ
- Try a different model, say, support vector machines (SVM) with different kernels

Now, one could "brute-force" their way through all the strategies above to find the optimal algorithm, but it can be immensely time-consuming (and more the matter of luck!). A better approach is to try and diagnose what the problem actually is.

Overfitting vs. underfitting. First thing to check is if we are overfitting (high variance) or underfitting (high bias). In cases of high variance, the training error is often much lower than the validation error, while in cases of high bias, both the training and validation errors are both high.



Clearly, in cases of high variance, getting more training examples (i.e., increasing n helps), or fitting a less-complex model (say, via smaller feature set) could help. Conversely, using a larger feature set helps with high bias situations. In the current ChatGPT regime, you'll often hear researchers say, "we have compute, but we are going to run out of data!!"—this is mainly because *large* language models are in an over-parameterized regime with complex models; if we were to deal with a high-bias model, getting more data wouldn't help.

Bad optimization or bad objective function. Say, θ_{SVM} and θ_{LR} be an SVM and logistic regression classifier parameters learned on the training data. Now, let us consider (a) the log-likelihood $\ell(\theta) = \sum_{j=1}^{n} \log p(y^{(j)}|x^{(j)};\theta)$, which was maximized *by logistic regression* during training, and (b) the model accuracy $\text{acc}(\theta)$ as the fraction of examples correctly classified.

Two cases: say, we have,

$\ell(\theta_{\text{SVM}}) > \ell(\theta_{\text{LR}})$	SVM "better" maximizes logistic regression objective
$\text{acc}(\theta_{\text{SVM}}) > \text{acc}(\theta_{\text{LR}})$	SVM achieves higher accuracy

Despite logistic regression being trained to minimize $\ell(\theta)$, SVM achieves a lower $\ell(\theta)$ value, hinting that the optimization may have been unsuccessful. Hence, the problem is with the optimization algorithm or the convergence thereof.

Alternatively consider the case where,

$\ell(\theta_{\text{SVM}}) \leq \ell(\theta_{\text{LR}})$	logistic regression better maximizes log-likelihood
$\text{acc}(\theta_{\text{SVM}}) > \text{acc}(\theta_{\text{LR}})$	but, SVM achieves higher accuracy

Here, the optimizer succeeded to maximize the objective, but SVM does better on the accuracy measure. This hints that maybe $\ell(\theta)$ might not be the right function to maximize, if we care about accuracy. Hence, the problem here is with the choice of the objective function.

To summarize,

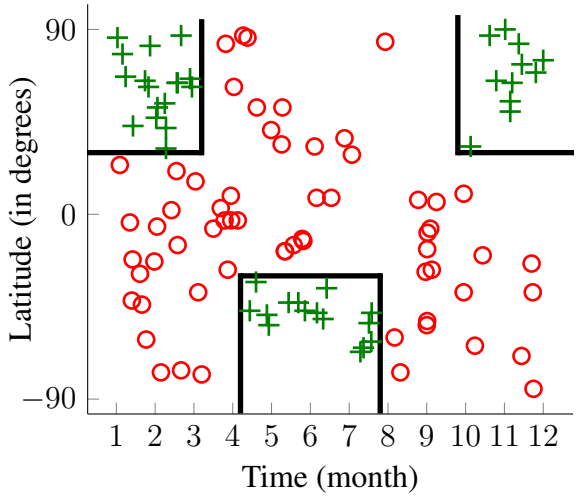
- Try to obtain more training data [fixes high variance]
- Too many features—use a smaller set of features [fixes high variance], or contrarily, suspect that your features are not "good" enough, and you obtain a larger feature set [fixes high bias]
- Run gradient descent for more iterations [fixes optimization problem]
- Try using a different optimizer?—second-order: Newton, momentum, adaptive gradients with momentum, etc. [fixes optimization problem]
- Use a different regularization hyperparameter λ [fixes objective function]
- Try a different model, say, support vector machines (SVM) with different kernels [fixes objective function]

As a final piece of advice: rather than saying, "here's an algorithm that works," it is more useful to say: "here's an algorithm that works because of *some* component."

2 Decision trees

In this class, we spent significant time in "drawing lines" (e.g., perceptron, logistic regression, naive Bayes, SVM, etc.), and in cases where we couldn't draw lines in the given feature space, we drew lines in a higher-dimensional space (through kernels). However, in some cases, such line-drawing might not be the most natural way to separate inherently non-linear data.

As a motivating example, consider the following example of whether or not it is possible to ski at a given location (in latitude; $-90°$: south pole, $0°$: equator, $90°$: north pole), during a specific month:[3]



Clearly, there is no linear boundary that splits the dataset. However, there are different *regions* of positive and negative samples that can be isolated and one such configuration is shown above. Concretely, we wish to partition the given input space $\mathcal{X}$ into r disjoint regions $R^{(i)}$s such that,

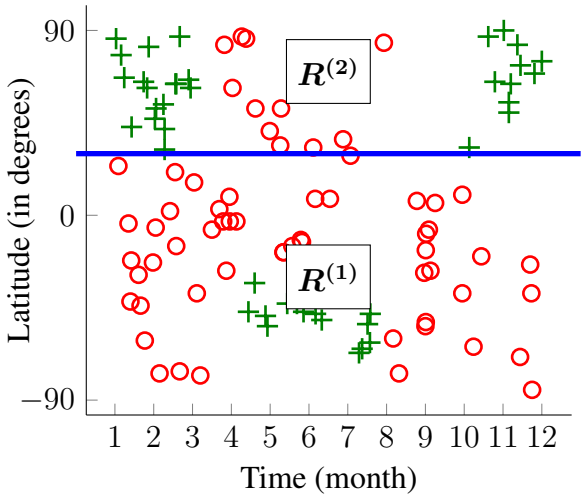
$$\mathcal{X} = \bigcup_{i=1}^{r} R^{(i)}; \quad R^{(i)} \cap R^{(k)} = 0 \text{ if } i \neq k.$$

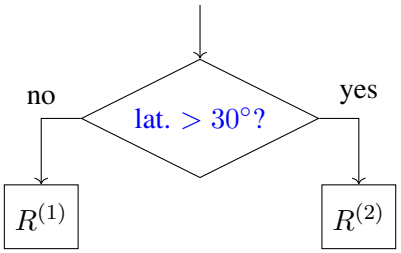
As it turns out, partitioning the space into maximally compact regions is an NP-hard problem. To this end, we will try to design a greedy, top-down approach that recursively partitions the input space.

[3] Inspired from Townshend (2018).

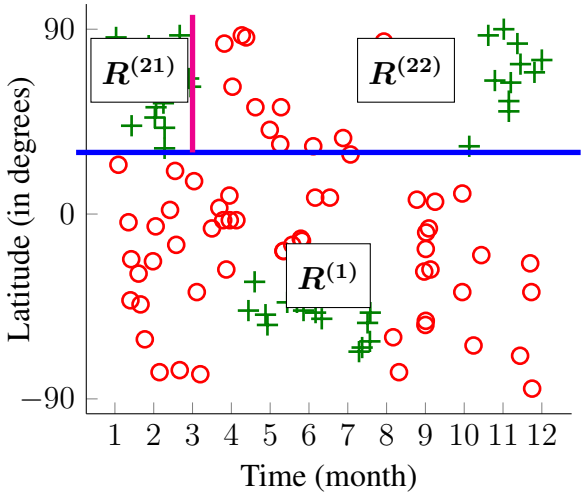
2.1 Greedy, top-down, recursive partitioning

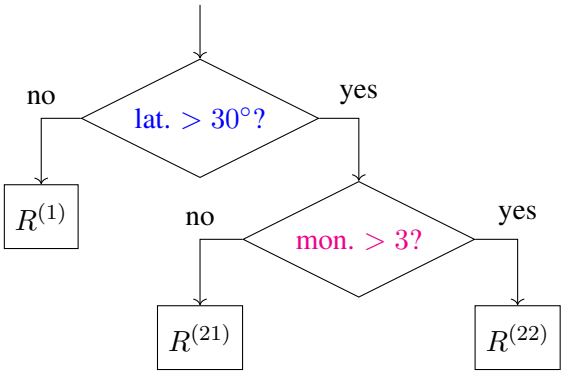
Th core idea is that we are going to play "20 Questions" with the input space to partition it. For example, we could ask: "Is the latitude $> 30°$?", which would partition the input space $\mathcal{X}$ as:





Once divided, we can then further partition $R^{(1)}$ and $R^{(2)}$ *recursively*, by asking additional questions; say, "Is month > 3, when latitude $> 30°$ (i.e., in $R^{(2)}$)?" further splits $R^{(2)}$ as:





Now, we realize that by asking these questions, we can recursively split the space into regions. Two things to note here: (1) we haven't yet discussed what the "right" questions are, or how to come up with right questions, and (2) the stopping criteria, i.e., when do we stop recursively partitioning a space.

Let's first answer the second question on stopping criteria first. Realize that the question "Is month > 3, when latitude $> 30°$?" (or, the region $R^{(21)}$) is homogeneous since it only contains positive samples. Therefore, further splitting $R^{(21)}$ yields us no benefit. To this end, one way of defining a stopping criterion for this recursive partitioning is to stop splitting a region when the region is homogeneous (also referred to as "pure").[4]

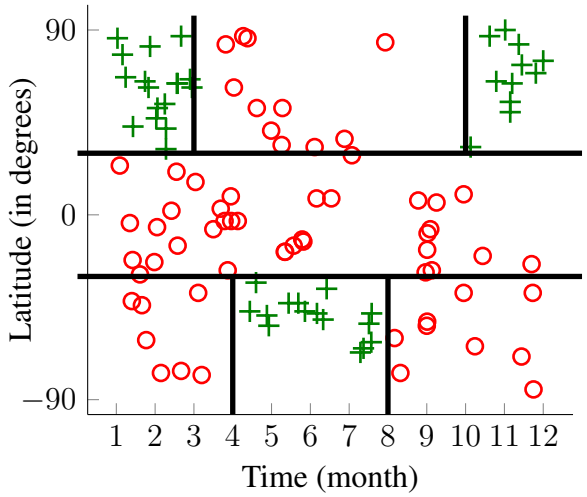
Now, before understanding how to choose the "right" questions to ask, we realize that *any* question is characterized by two entities: a feature (e.g., latitude) and some threshold on that

[4]This need not always be the case; owing to computational efficiency, one could simply stop after a set number of iterations and then take the majority vote of the samples in the region. Even when computational efficiency is not a concern, if identical samples have different labels, we would have to resort to a majority vote.

feature (e.g., $30°$). Formally, given a parent region $R^{(p)}$, a feature-i, and a threshold $t \in \mathbb{R}$, we obtain two child regions $R^{(p1)}$ and $R^{(p2)}$ as:

$$R^{(p1)} = \{x^{(j)} | x_i^{(j)} \leq t, x_i \in R^{(p)}\};$$
$$R^{(p2)} = \{x^{(j)} | x_i^{(j)} > t, x_i \in R^{(p)}\}.$$

For completeness, the desired partitioning of the ski dataset is as follows:



2.2 Asking the "right" questions: Region-based cost function

Recall that our goal is to greedily partition the input space by asking as few questions as possible. Simply put, at each step, we wish to choose a (feature-i, threshold t) pair that partitions the region into purest-possible subregions. More formally, if $J(R^{(p)})$ measures the "impurity" of $R^{(p)}$ (for some definition of purity), then, we seek to find i, t such that

$$\arg\max_{i,t} J(R^{(p)}) - \frac{|R^{(p1)}|J(R^{(p1)}) + |R^{(p2)}|J(R^{(p2)})}{|R^{(p1)}| + |R^{(p2)}|}.$$

Realize that we maximize the above when we (are able to) choose i, t such that $J(R^{(p1)}) = J(R^{(p2)}) = 0$, i.e., both subregions are completely pure. One final thing to note here is the cardinality-weighting in the objective: this is to ensure that a pure region with a single sample is weighted appropriately when measuring the decrease in impurity. Now, the last piece of this decision tree puzzle is to define an appropriate $J(R)$.

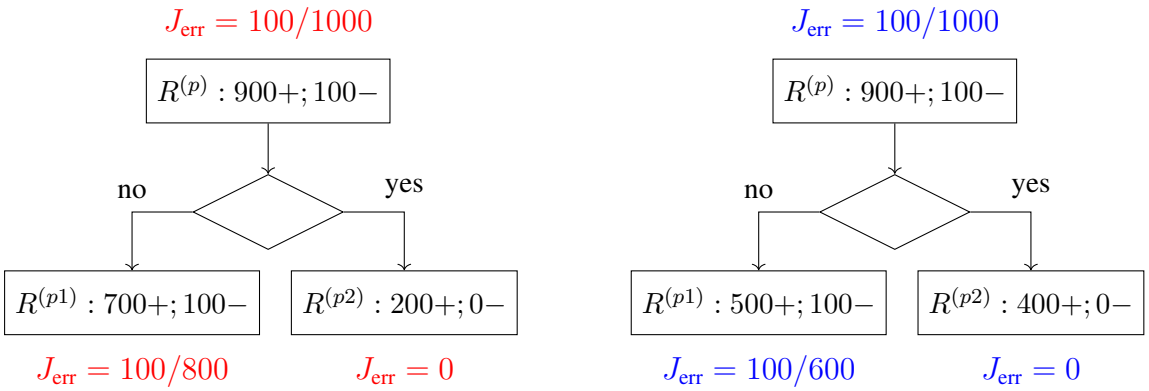
2.2.1 Impurity as misclassification error

In a classification setting, we are interested in misclassification error. To this end, we can define $J(R^{(i)})$ to be the misclassification error incurred by assigning majority vote in $R^{(i)}$. Simply put,

$$J_{\text{err}}(R^{(i)}) = 1 - \max_{y \in \mathcal{Y}} p_y^{(i)},$$

where $p_y^{(i)}$ is the proportion of samples in region $R^{(i)}$, labeled as belonging to class-y. For example, if R has 600 positive and 400 negative samples, the majority vote would be $y = +1$, resulting in $J_{\text{err}}(R) = 1 - (600/1000) = 0.4$. Additionally, note that, for pure regions, we have $J_{\text{err}}(R) = 0$ (as desired).

While the misclassification error is the final quantity of interest, it is not very sensitive to changes in class probabilities, i.e., J_{err} remains the same, so long as the majority class within a region remains the same. As an example, consider the following two decision trees as shown:



Now, the decrease-in-impurity objective for the left configuration is

$$\frac{100}{1000} - \frac{800(100/800) + 200(0)}{800 + 200} = 0;$$

similarly, for the right configuration, we have

$$\frac{100}{1000} - \frac{600(100/600) + 400(0)}{600 + 400} = 0,$$

indicating not only that the splits are identical, but also that neither splits decrease the impurity over that of the parent region. One could argue that the right decision tree is preferable since it isolates more positive samples.

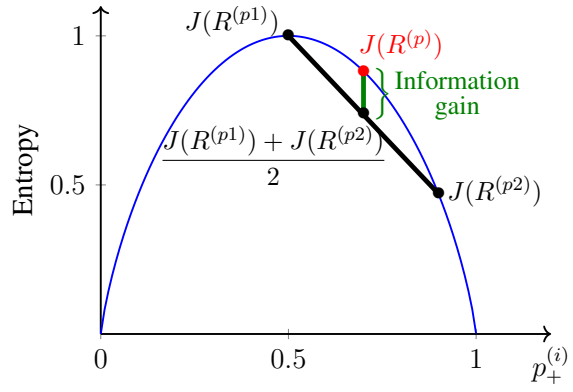
2.2.2 Impurity as randomness in a region

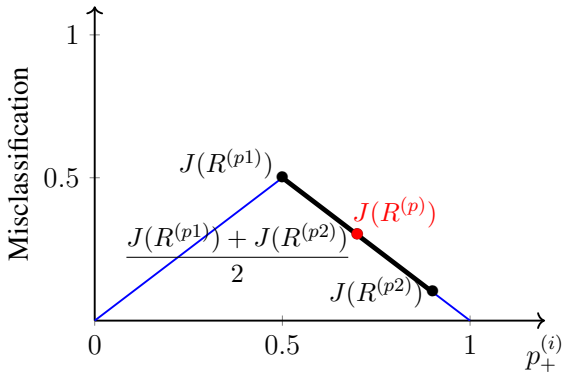
A more sensitive J can be to measure the randomness within a given region and use that to quantify the amount of impurity. Entropy measures the average number of bits needed to encode the class label of a randomly chosen point in the given region. Naturally, a pure region has no randomness, and hence would have zero entropy. Formally,

$$J_{\text{ent}}(R^{(i)}) = \sum_{y \in \mathcal{Y}} -p_y^{(i)} \log_2 p_y^{(i)}.$$

Given that we motivated J_{ent} as being more sensitive to class probabilities, we can visually justify the difference between J_{ent} and J_{err}. For convenience, let us assume binary classification; hence, we have $p_-^{(i)} = 1 - p_+^{(i)}$. Additionally, let us also assume that given $R^{(p)}$, we split it into $R^{(p1)}$ and $R^{(p2)}$, where $|R^{(p1)}| = |R^{(p2)}|$, which implies that

$$p_+^{(p)} = \frac{|R^{(p1)}| p_+^{(p1)} + |R^{(p2)}| p_+^{(p2)}}{|R^{(p1)}| + |R^{(p2)}|} = \frac{p_+^{(p1)} + p_+^{(p2)}}{2}.$$





Since J_{ent} is strictly concave, it is easy to realize that as long as $p_+^{(p1)} \neq p_+^{(p2)}$, then the weighted sum of the childrens' J_{ent} values would always be less than that of the parent. This decrease in cost is often termed as the *information gain*. Now, on the other hand, misclassification error is not strictly concave, meaning that for any two points, $p_+^{(p1)}$ and $p_+^{(p2)}$, there is no guarantee that the weighted sum would be less than that of the parent. Due to this added sensitivity, entropy is often used as the cost function in construction decision trees.

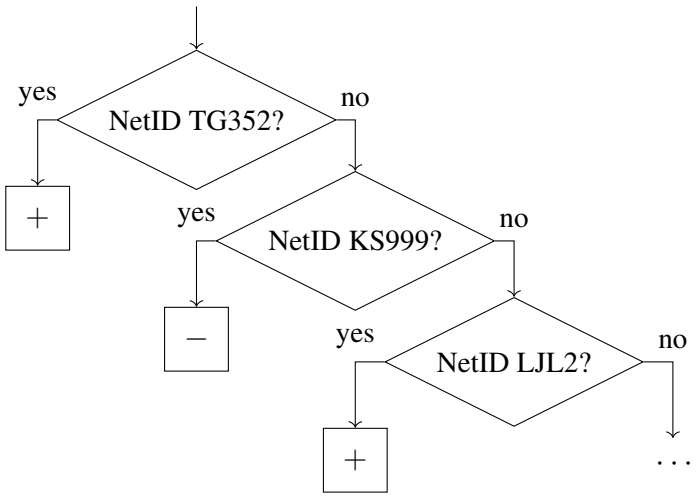
Aside: Gini impurity. Another choice for $J(R)$ is the Gini impurity, which is similar to entropy and is defined as

$$J_{\text{gini}}(R^{(i)}) = \sum_{y \in \mathcal{Y}} p_y^{(i)}(1 - p_y^{(i)}),$$

which is also strictly concave. Additionally, since Gini impurity avoids computing $\log_2$, it is computationally more efficient than entropy.

Categorical attributes. An advantage of decision tress is their ability to deal with categorical attributes easily. For example, in our ski example, if we used {northern hemisphere, equator, southern hemisphere}, instead of the latitude, we would simply amend "Is the latitude $> 30°$?" to "Is the location in northern hemisphere?" and all else remains the same.

Now, one thing to be cautious about when dealing with categorical attributes is when the attribute takes too many values (e.g., NetID). For example, if we were to split the dataset of 400 CS 3780/5780 students on the NetID, we could essentially be asking 399 "Is your NetID TG352?" questions:[5]



Since each of the resultant children regions (singletons) are pure, i.e., our cost J, for each child region is zero, i.e., we obtain maximum gain by splitting on the categorical attribute. However, it stands to reason that this is not a good idea; not only is this computationally intractable, but also, using an attribute this *specific* to the dataset results in a high degree of overfitting. Hence, in case of highly-branching categorical attributes, serious consideration must be given to whether the feature can be reformulated as a quantitative one instead.

Regression trees. Before concluding our discussion on cost functions, we note that all the above cost functions only work for classification settings; what if we were modeling a regression problem instead? Say, we are now modeling the amount of snowfall (in inches), instead of whether or not it is possible to ski.

[5] More generally speaking, for q categories, we have 2^q possible subsets, and hence 2^q possible questions.

The tree growing process itself remains the same, with the following two changes: (1) instead of the majority vote, we now use the average value in a region as the final prediction in that region:

$$\hat{y}^{(i)} = \frac{1}{|R^{(i)}|} \sum_{j \in R^{(i)}} y^{(j)},$$

and (2) we can use the average squared loss as our cost function in choosing the features and thresholds:

$$J_{\text{sq}}(R^{(i)}) = \frac{1}{|R^{(i)}|} \sum_{j \in R^{(i)}} (y^{(j)} - \hat{y}^{(i)})^2.$$

2.3 Regularization

By now, we realize that decision trees are generally low bias, high variance models, i.e., they are prone to overfitting. One way to deal with high variance is to regularize the model; some approaches to that end are:

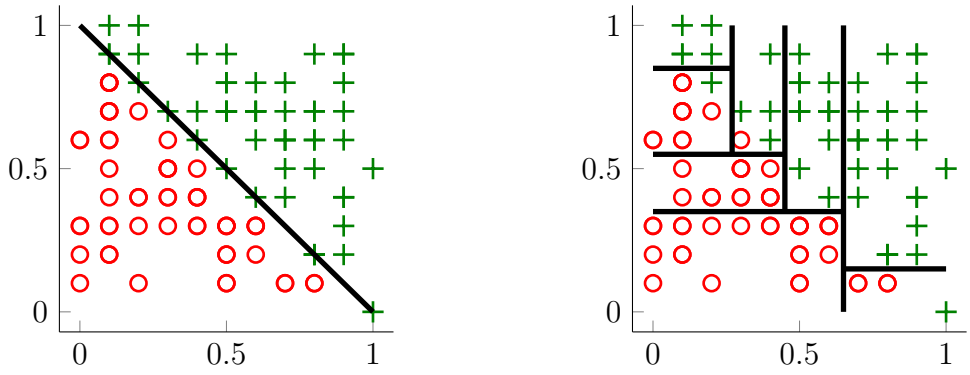
- Set a minimum leaf size: Do not split the region further when $|R|$ is below some preset value.
- Enforce a maximum tree depth: Do not split R if the number of splits taken to reach that region exceeds some preset value.
- Threshold on maximum number of nodes: Stop the tree growth if the tree has reached a set number of nodes.

What about thresholding the decrease in information gain? While appealing, this is not a good strategy since our tree-building is greedy. There might be steps with little gain, but are essential in further building the tree. In our ski example, the first split on "latitude $> 30°$?" doesn't produce much gain (at least not as much as the "is month > 3, when latitude $> 30°$?" split), but is quite crucial in realizing the final tree.

Another approach is to fully grow out the tree and then prune backwards using the misclassification error (or, squared loss) on the validation set.

3 Conclusion

In this lecture, we mainly discussed decision trees (often referred to as "CART" for classification and regression trees). One of the main advantages of decision trees is the high degree of interpretability they offer. Additionally, note that traversing a depth-d tree takes $\mathcal{O}(d)$ time, which is quite fast for reasonable d. In noting the possible high variance of decision trees, we discussed ways to regularize by adjusting the stopping criteria or by pruning backwards.



One major downside of decision trees is that we need to ask a lot of questions to approximate a linear boundary, and even then, the resultant linear approximation can be really bad. A simple example of this is of a 2D dataset that models $x_1 + x_2$ (shown above), where x_1 and x_2 are the features. A linear model could directly derive the linear boundary, while a decision tree requires many splits. This is mainly because our tree-building only considers splitting on one feature at a time; there are works that allow for considering multiple features at a time, but that comes at the cost of interpretability and an increased variance.

A Notation

$\mathcal{D}$	The training dataset of n samples
n	The number of training samples in the dataset $\mathcal{D}$
d	The number of features
$x^{(j)} \in \mathbb{R}^d$	The d-dimensional feature vector associated with the j-th training sample
$y^{(j)}$	The target value associated with the j-th training sample (real-valued in regression; discrete in classification)
$x_\ell^{(j)} \in \mathbb{R}$	The ℓ-th feature (element) of $x^{(j)}$
$\mathcal{X}$	The input space; each $x^{(j)} \in \mathcal{X}$
$\mathcal{Y}$	The output space; in binary classification, we have $\mathcal{Y} = \{-1, +1\}$
θ	The parameters of the model (a.k.a., hypothesis)
h	The hypothesis function(e.g., $h(x) = \theta^T x$ for linear regression)
$\hat{\varepsilon}(h)$	The training error of the hypothesis function h, computed for the dataset $\mathcal{D}$
$\mathcal{H}$	A class of hypothesis functions (can be an infinite)
$\hat{h}$	The ERM-chosen hypothesis with the lowest training error among all $h \in \mathcal{H}$
$\varepsilon(h)$	The generalization error of the hypothesis function h
$0 < \delta < 1$	From the union bound, we have that with *at least* $1 - \delta$ probability, $P(\neg h_j \in \mathcal{H}$ s.t. $\lvert\varepsilon(h_j) - \hat{\varepsilon}(h_j)\rvert > \gamma)$; "¬" indicates "logical not"
$h^\star$	The best-in-$\mathcal{H}$ hypothesis that achieves the lowest generalization error
$\lambda > 0$	The regularization hyperparameter
$\Omega(h)$	Measures the complexity of hypothesis h (e.g., norm of the weights)
$\ell(\theta)$	The log-likelihood function
$\mathrm{acc}(\theta)$	The accuracy function using the hypothesis parameterized by θ
$R^{(i)}$	The i-th (parent) region
$R^{(i1)}, R^{(i2)}$	The children regions obtained by splitting the i-th parent region
$\lvert R^{(i)}\rvert$	The number of samples in $R^{(i)}$, also known as the cardinality of $R^{(i)}$
$p_y^{(i)}$	The proportion of samples in $R^{(i)}$ with class label y; for example, in a region with $600+$ and $400-$ samples, $p_+^{(i)} = 600/1000 = 1 - p_-^{(i)}$
$\hat{y}^{(i)} \in \mathbb{R}$	($y^{(i)}$ with a hat) Average value in a region $R^{(i)}$, used in regression trees

References

T. Gangavarapu. MLU Lecture notes: Decision Trees. *Machine Learning University, Amazon*, 1(1):1–14, 2020. URL https://tushaargvs.github.io/assets/teaching/dt-notes-2020.pdf.

M. Pisani. Gentle introduction to the bias-variance trade-off in machine learning, Jun 2020. URL https://www.rootstrap.com/blog/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning.

A. Singh. 10-704 Lecture notes on Empirical Risk Minimization. *10-704 (Spring 2015) Information Processing and Learning Lecture notes*, 1(1):1–6, 2015. URL https://www.cs.cmu.edu/~aarti/Class/10704_Spring15/lecs/lec11.pdf.

R. J. L. Townshend. CS229 Lecture notes: Decision Trees. *CS229 Lecture notes*, 1(1): 1–9, 2018. URL https://cs229.stanford.edu/notes_archive/cs229-notes-dt.pdf.

(Last compiled: 4/8/2025, 11.26am ET.)