Decision Trees (DTs)

Week	THURS. Week 4
Assignment Due	
Assignment Done	☐
Due Date	
Notes Done	☑

Presentation

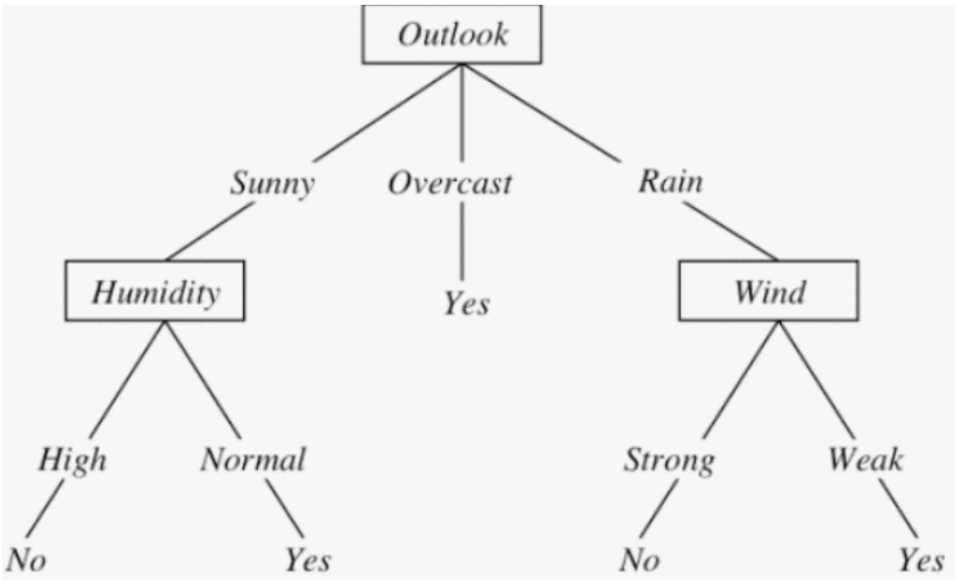
https://s3-us-west-2.amazonaws.com/secure.notion-static.com/03f75000-5e34-48e4-aaa4-4f67237aea7c/L2-DTs.pdf

Class Notes

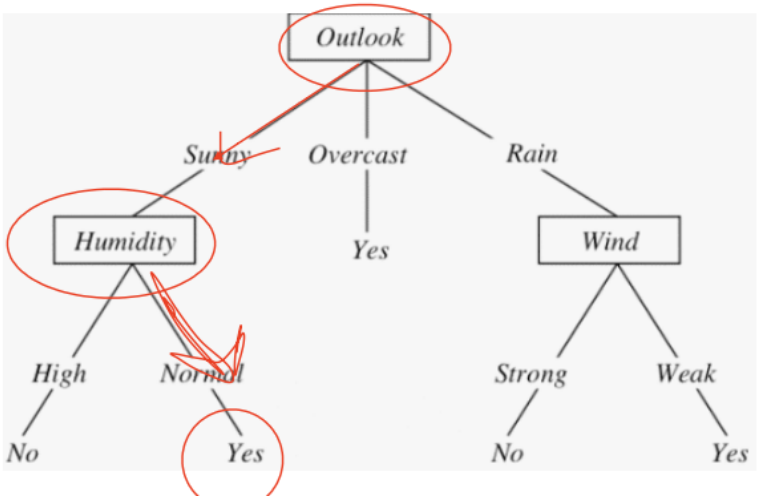
Decision Trees

With a decision tree, we start at the root.

- Each of the non-leaf (internal) nodes are a feature/attribute.
  - The branch we travel down from the root to the leaf will be a possible value of that feature/attribute.
  - A feature will tell us what branch to go down.
  - The leaf nodes are our actual decisions, assigning a classification.



- Ex:
  - Outlook, humidity, and wind are the features.
  - Based on observation's features, we traverse down the decision tree until hitting a leaf node, which tells us our class.
    - $X = [O = S, H = N, W = W]$



Consistent Decision Trees

If we have D features and all were binary features with a binary class, then there are an exponential number of decision trees → 2^{2^D}

- It is NOT possible to create every decision tree and validate which is the best.
- If data is noiseless and without any randomness in its values for the features, then there will be at least one trivially **consistent** decision tree for the dataset, meaning that it fits the data perfectly.
    - In fact, there are many potential ones!
    - We must ask how do we consider one tree to be better than another? We can consider:
        - Tree short in height → compact in nature, takes less to traverse
- The taller that a decision tree is, the more likely it is that there is overfitting occurring.

Example

Ex: Situations for which I will/won't play tennis

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes

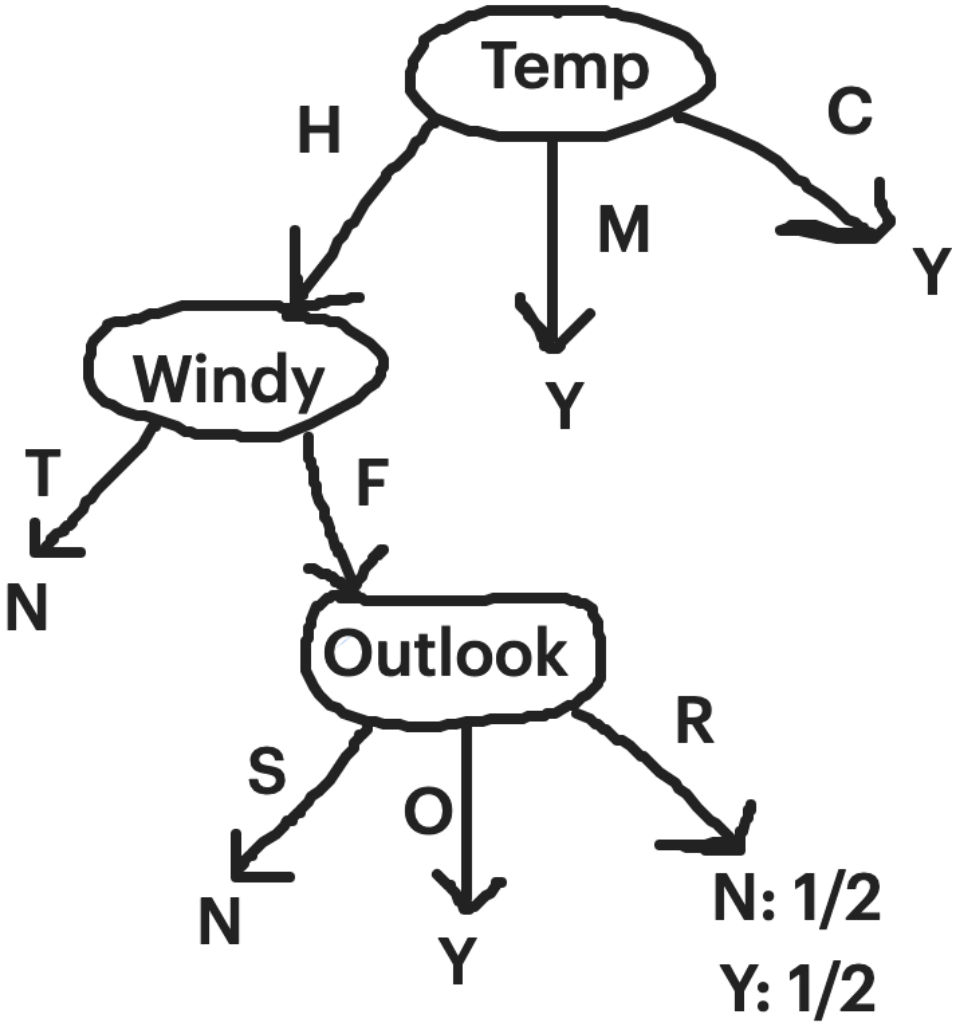
Randomly, let us say that we begin our decision tree with the Temperature feature (which has 3 value possibilities):

- Here, we don't have a consensus for all of the features and their values, so we can take more features into account.

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes



- Upon considering temperature, we can consider windy as a feature, looking at where there is consensus.
    - We can see that there is only one sample with consensus under the description of having a hot temperature and windiness present.
- We go further and consider outlook as a feature and see that there are 3 observations left to consider and all of them have consensus for a class.
    - We’re left with no samples considering rainy as an outlook. We have no observations here in which the temperature is hot, it is not windy and the outlook is rainy.
        - Typically, we would simply choose what the *mode* is of the classes.

- The only observations that we've had up to this point have had a 50/50 split (the sample with a sunny outlook has N and the sample with an overcast outlook has Y) within the classes observed so we'd put these probabilities as a way to classify a class up to this point.

Decision Tree Learning

In practice, it may be impossible to build a consistent decision tree.

- Factors hindering a consistent decision tree:
  - There may be noise.
  - There may be randomness.
  - We don't have enough information/features.
- Ultimately, we 'd like to find a small (compact) tree consistent with as many training examples as possible.

Each time we're building our decision tree, we want to choose the ***most significant attribute/feature*** as the root of the (sub)tree, effectively taking a greedy approach.

- Without thinking about the "best feature", the general recursive DT algorithm looks like this:

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Arguments:
  - ***examples*** → all observation data
  - ***attributes*** → features from observation data
  - ***default*** → most common class for default label, often mode of all data
- Run-through of pseudocode:
  - If the ***examples*** are empty and there are no more to consider, then we return a node with the default class.
    - Note: This scenario would be similar to how we treated the case when we had no more observations for the outlook being rainy. In our case though, we added a probability instead of assigning a label to the mode of the classes.
  - If there is consensus, then we'll put the observation with the classification.
  - If we've already exhausted exploring all of the features down a path, then there are no more features to split on.
    - We take the data that is remaining and find the mode of the classification that the observation that we are seeing that can no longer be split due to no more available features.
  - Otherwise, we still have data to iterate through that does not agree on a class label *and* there are more features down that path to utilize.
    - Now, we must consider which remaining feature would be the best to split on.
    - Then we'll create a branch for each value of that best feature, creating a new *subtree*.
      - For each leaf, we'll grab that subset of the data that arrived at that node.
      - We pass in the mode of the classes of those example datapoints that arrived at that node as the *default class* to assign to.

ID3

Ideally, what makes a great feature is one s.t. down each branch, each observation will have consensus within them (no entropy):

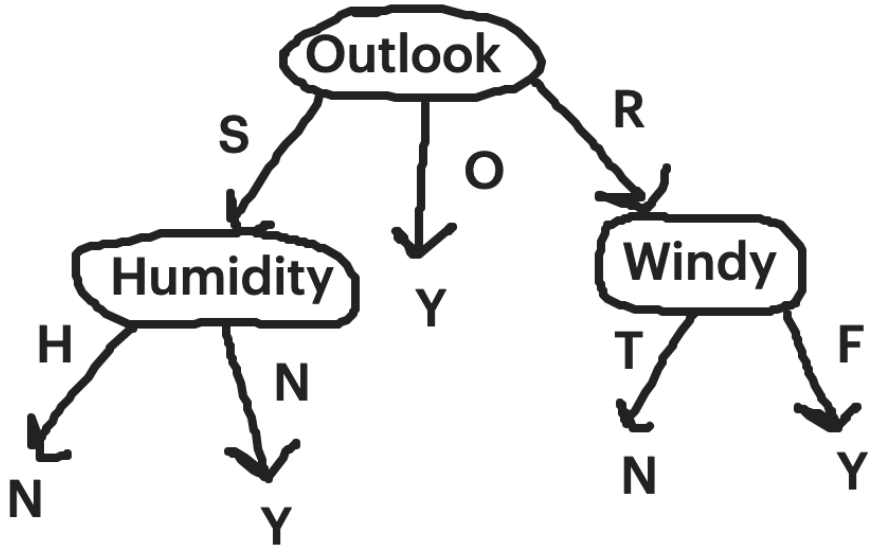
- This is likely not the case for the features that we'll have, but we can attempt to choose the feature that is *closest* to this goal of no entropy (i.e.: lowest class entropy).
- The algorithm where to choose the "best" feature we use the entropy of the class labels of the subsets that we create is referred to as an **ID3 decision tree**.
- When creating a decision tree with using entropy (or information gain) is called the **ID3 decision tree algorithm**.
    - Initially, we begin by finding the root of our tree. All 4 features are fair game that we're considering.
- What would've had the lowest class entropy is outlook → sunny, overcast, rainy
    - From here, we need to look at the subsets created by their observations:
        - In starting off by looking at observations where Outlook is Sunny ($n = 5$), we see that there is no consensus, so we can't make a decision yet here on what classification to make.
            - At this point, we've only used the Outlook feature to classify, but we still have other features to work with, such as Temperature, Humidity, and Windy.
        - We look at the remaining possible features and determine which have the "best" entropy (the lowest) based on the data that has *arrived there*.
            - In considering the temperature feature as the next feature to split by, we can see that the 5 samples that we can see have consensus on two of the 3 labels that temperature has:

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

    - Two of the branches created would have 0 entropy, while one branch would have complete randomness.
- Instead, we can consider the Humidity feature, as it has consensus with the observations that we're considering at point of having a Sunny outlook:

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

- We continue this down the entire tree, seeing that with the Overcast outlook feature value, there is consensus
    - The third branch where the outlook is rainy, we don't have consensus, so we continue to use the next feature here to split upon.
        - Note: At this point, we can still consider humidity as a feature to split by! We only would ignore Outlook since it's at the root of our branch and we've already made a decision on it.



- Preference of professor: Choose class label based on a probability, not the mode.
  - This avoids complications that arise with there being ties or close ties.
  - In a way, using the mode artificially biases the results to some value.

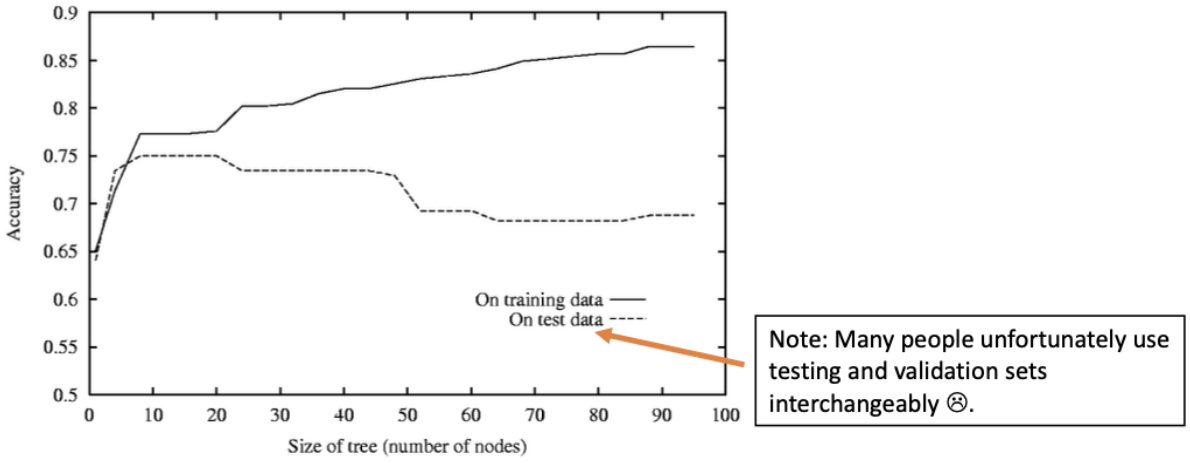
Overfitting

When creating a decision tree, we may fit our data as closely as possible.

- However, we may overfit the data so much that the tree doesn't generalize as well 😞
  - This means that what was learned is **overfitting** the training data and may not generalize well to data that it was not trained on.
- Since this is an iterative (or recursive) algorithm, the use of a validation set to decide between different versions is quite natural.
- This figure displays the effect of the size of the tree on the accuracy.
  - As we're building our tree (seeing an increase of nodes), initially, we see a large jump in accuracy of the *training AND validation data*, as we haven't made too many decisions, just the best broad stroked decision.
    - As we're fine tuning the nuances of our decision tree, this will improve the accuracy of our *training data*.

- However, at a certain point, we may see that it is not generalizing well with the validation accuracy!
    - Where we see a dip is where what is being learned by the decision tree is the nuanaces of the specific training data, not the big picture of all the possible data as a whole.

- It is good to monitor this effect to identity when it happens and consider techniques for how to deal with this overfitting.



Reduced Error Approaches

There are general ways to deal with overfitting and algorithm-specific ways to address overfitting:

- A general way to deal with ***overfitting*** is to use MORE of the data for training.
    - More data for training gives a more general solution → could possibly use cross validation to maximize the amount of data that we use as training data, which will allow for a better representation of all of the data possible, allowing for better generalization.
- Another general way to deal with ***overfitting*** is to reduce the dimensionality of our number of features.
    - Less features = less internal nodes = less branching = less subtleties that we'll deal with
- For decision trees, we can specifically do:

- **Reduced error growing** - addresses what the figure is displaying as a problem
  - As we're *growing* the tree, we can monitor the validation performance.
  - If we notice validation performance starts decreasing, we can STOP growing our tree as we're in the process of building it.
- **Reduced error pruning** - takes an approach in the opposite direction of **reduce error growing** and *prunes* the tree
  - Hypothesis behind approach: The last splits of the decisiontree are usually the ones trying to squeeze out the last bits of subtleties in the training data. If we remove these splits, then maybe this will reduce the likelihood of overfitting.
  - This approach allows the tree to grow but then evaluate the import on the *validation set* of pruning each possible node (plus those below it).

Working with Continuous Valued Inputs

If our features are continuous, we run into complications with branching, as we cannot branch on every possible value of the feature.

- Commonly, we take the continuous values, and apply threshold values to bin them into categorical or nominal enumerations and possibly use a probability distribution model as discussed before.
- Idea: We can assume that the feature's intraclass data follows a distribution (like a Normal/Gaussian distribution).
  - We can then compute the parameters (μ, σ) for that model from the class data (as in Naïve Bayes').
    - These parameters can then be used to help decide which features should go down which branch.
    - From here, we can compute the average weighed class entropy in the created datasets.
- This is not a focus that we're doing, but it is important to mention that there *are methods* that exist to do this classification similar to how we've computed entropy using continuous values.