# CS3600: Decision Tree Project Tips

Osama Sakhi

Georgia Institute of Technology

---

## Contents

# 1 Notation:

### `examples`

A list of dictionaries. Each dictionary, `example`, represents a single data entry in the dataset.

### `className`

The key in a single `example` dictionary that corresponds to the output label.

### `classValue`

The value of the output label `className` for a single `example`.

### `attrName`

A single key in an `example`, this can be any feature (attribute).

### `attrValues`

The list of possible values an attribute can take on. A single value will just be `attrValue`.

# 2 Helper functions

These functions allow us to partition our data into various subsets and count up distributions of examples.

### `getMostCommonClass()`

This function takes a list of dictionaries `examples` and outputs the most common label among the examples. The dictionary element with key name `className` is the output label for that example, so this function counts the different values for those and returns the most common one.

This one's implemented for you already.

### `getClassCounts()`

This function generates a dictionary with keys as possible values of `className`, and the values as the count of the number of occurrences of that `classValue` in `examples`.

You'll want to iterate over each example in `examples` and extract the `classValue`. The `classCounts` dictionary will then be indexed into using `classValue` and the count will be incremented with each example.

**NOTE:** Make sure to account for the first time you see a particular `classValue` that you haven't added to the dictionary before. The dictionary function `example.get()` might be helpful here.

## `getPertinentExamples()`

This function takes `examples`, and returns a list of the subset of those examples that have the feature `attrName` set to `attrValue`.

You'll want to iterate over the examples, extract the feature `attrName`, and check if the value there is equal to `attrValue`. If it is, append that `example` to the list the function returns, called `newExamples`.

## `getAttributeCounts():`

This function returns a dictionary where the keys are each of the `attrValues`, and the values are dictionaries themselves. These "inner" dictionaries have as keys the various values the output label `className` can take on, and the values are the counts of each of those classes.

**Example:**
Let's say we have these actors as `examples`, and the `className` feature is "graduated", which can be "Yes"/"No":

```
[
{"name": "Barney", "favoriteShow": "How I Met Your Mother", "graduated": "Yes"},
{"name": "Joey", "favoriteShow": "Friends", "graduated": "No"},
{"name": "Rachel", "favoriteShow": "Friends", "graduated": "Yes"},
]
```

If we call `getAttributeCounts()` with the appropriate parameters for the feature `favoriteShow`, then we'll get this dictionary returned:

```
{
"How I Met Your Mother": {"Yes": 1},
"Friends": {"Yes": 1, "No": 1},
}
```

**Implementation:**
First, initialize empty dictionaries for each `attrValue`. Each `attrValue` will then serve as a key to `attributeCounts` and the values will be those empty dictionaries, which we'll refer to as `innerDict`. Now we need to populate the `innertDicts`.

Start by iterating over all `examples`. Extract the `classValue` and `attrValue` of the `example` you are currently looking at. Index into `attributeCounts` to get an `innerDict`, and then increment the count for the key `classValue` by 1, similar to how you incremented class counts in `getClassCounts()`.

# 3 Entropy Functions

## `setEntropy()`

This function provides as input the number of occurrences of each `classValue`, called `classCounts`, and it outputs the entropy of the dataset.

**Entropy Formula:**
The entropy for a given dataset $D$ is:

$$H(D) = -\sum_{y_i} p(y_i) log_2 p(y_i)$$

where:

$$p(y_i) = \frac{\text{number of examples labeled } y_i \text{ in } D}{\text{number of examples in } D}$$

and $y_i$ is all possible labels. In other words, $y_i \in$ `classValues`.

**Implementation**:
To compute $H(D)$, you'll need to compute the values $p(y_i)$ for each `classValue`. This is simply the count of a particular `classValue` divided by the sum of all counts of all `classValues`. Don't forget, the base for the formula is $log_2$, which you can always compute as:

$$log_2(x) = \frac{log(x)}{log(2)}$$

## `remainder()`

This function computes the "remainder" or Conditional Entropy, $H(D|A)$, where $A$ is an attribute we're concerned with, `attrName`.

**Conditional Entropy Formula:**
The conditional entropy for a given dataset $D$ and attribute of interest $A$ is:

$$H(D|A) = \sum_{a_i \in A} \frac{|D_{a_i}|}{|D|} H(D_{a_i})$$

where:

$$a_i = \text{a particular value of feature } \texttt{attrName}$$

$$D_{a_i} = \text{subset of examples in } D \text{ with attribute } \texttt{attrName} \text{ set to } a_i$$

**Implementation**:
Computing $D_{a_i}$ is the toughest part of this, but luckily you've already implemented `getAttributeCounts()`. Use this function along with `setEntropy()` to compute the remainder value, and return that.

### `infoGain()`

Information Gain is computed as:

$$IG(D, A) = H(D) - H(D|A)$$

Use your functions `setEntropy()` and `remainder()` to determine this value.

# 4 Gini Functions

### `giniIndex()`

The Gini index of a dataset $D$ is given by

$$\text{gini}(D) = 1 - \sum_{y_i} p(y_i)^2$$

here again we use $y_i$ to mean a particular `classValue`, and $p(y_i)$ to be the ratio of counts of class $y_i$ to all other classes within the dataset.

### `giniGain()`

The Gini Gain is defined for a dataset $D$ and attribute $A$ as:

$$\text{giniGain}(D, A) = \sum_{a_i} \frac{|D_{a_i}|}{D} \text{gini}(D_{a_i})$$

where we again we have:

$$a_i = \text{a particular value of feature } \texttt{attrName}$$

$$D_{a_i} = \text{subset of examples in } D \text{ with attribute } \texttt{attrName} \text{ set to } a_i$$

This will be very similar to your implementation of `infoGain()`, except now you'll have to call on `giniIndex()`. This function will actually return the inverse of the giniGain() function described above. Pay special attention to this comment from the code:

> *The inverse is returned so as to have the highest value correspond to the highest information gain as in entropyGain. If the sum is* 0*, return* `sys.maxint`.

# 5 Decision Tree Learning

### `makeSubtrees()`

There are 4 cases that you'll have to handle here:

1. **We have no examples left**. If we don't have even a single example, we should create a `LeafNode` with the default label chosen.

2. **We have no more attributes to split on.** A leaf should be created with the default label here too.

3. **All examples have the same class label.** A leaf should be created with that most common class label.

4. **None of the above.** We need to determine the attribute $A$ among the remaining attributes that maximizes the `gainFunc()`. This function wraps around the Entropy and Gini functions we created earlier.

   Once we determine the attribute $A$, we need to then split on $A$'s values. We need to create a `Node` object (already defined in `DecisionTrees.py`), and recursively call `makeSubtrees()` on the subsets $D_{a_i}$, subsets where the attribute $A$ takes on value $a_i$.

   These recursive calls return `Node` objects, which must then be hooked up as the `children` of the `Node` object we created during this function call.

   When we make these recursive calls, we also have to adjust the `defaultLabel` being sent to each of these calls. **HINT:** The function `getMostCommonClass()` will be of use here.

# 6 Decision Tree Classification

Let's recap on how a decision tree works. When you encounter a new `example` (with an unknown `className`), you start at the root of the tree, and go down a branch. The node itself branches on different values of a particular attribute, `attrName`, and each value, `attrValue` has a branch down to the next level. This process repeats until a leaf node is reached, and this leaf node's label will serve as the prediction for the `className` for this example.

## `classify()`

This function provides you a dictionary called `classificationData`, which is exactly like an `example`, except without an output label. You'll want to recurse through the nodes of the tree, checking which attribute (`attrName`) each node splits on, determining the branch corresponding to the `attrValue` this `classificationData` takes on, and go down that branch to another node. This process continues until you hit a leaf node (see the `LeafNode` class at the top of the file). Once you hit a leaf, return the value of that leaf node as your prediction for `className`.