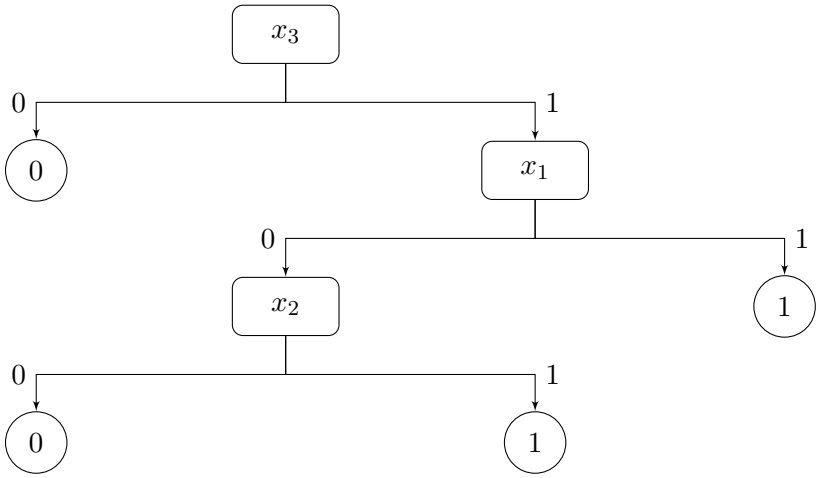
1: Decision trees

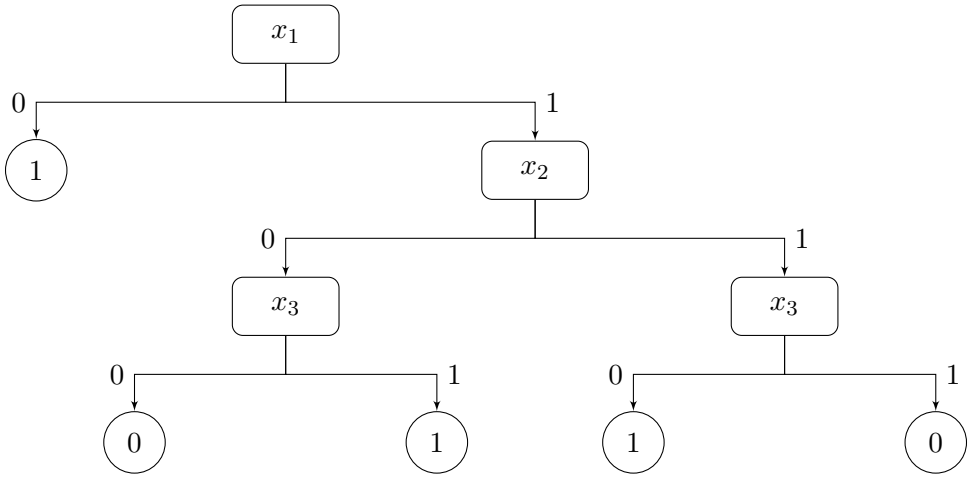
Note: Square nodes test for feature values and round leaf nodes specify the class labels.

(1) Representing Boolean functions as decision trees.

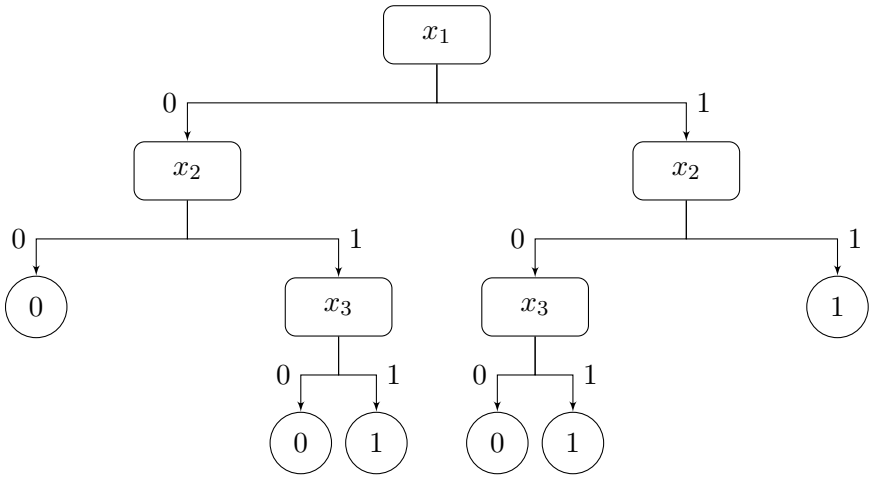
(a) $(x_1 \vee x_2) \wedge x_3$



(b) $(x_1 \wedge x_2)$ *xor* $(\neg x_1 \vee x_3)$



(c) The 2-of-3 function defined as follows: at least 2 of $\{x_1, x_2, x_3\}$ should be true for the output to be true.



(2) Pokémon Go decision tree to determine whether a Pokémon can be caught.

(a) There are 2 choices for Berry, 3 choices for Ball, 3 choices for Color, and 4 choices for type. This gives $2 * 3 * 3 * 4 = 72$ possible outputs. We are making a Boolean decision, so there are two ways to fill each of those 72 outputs giving 2^{72} possible functions.

2^{72} possible functions

(b) Entropy is defined as:

$$H(S) = -p_+ \log(p_+) - p_- \log(p_-)$$

The training set contains 16 examples, 8 of which result in a catch while the other 8 do not.

$$H(Caught) = -\frac{8}{16} \log(\frac{8}{16}) - \frac{8}{16} \log(\frac{8}{16}) = 1$$

$H(Caught) = 1$

(c) Information gain is defined as:

$$Gain(S, A) = H(S) \ - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

Berry = Yes. 7 out of 16 examples. Caught = 6/7 - Not Caught = 1/7. $H(berry_{yes}) = 0.592$.
Berry = No. 9 out of 16 examples. Caught = 2/9 - Not Caught = 7/9. $H(berry_{no}) = 0.764$.

$$Gain(S, Berry) = 1 - ((\frac{7}{16})(0.592) + (\frac{9}{16})(0.764)) = 0.689$$

$Gain(S, Berry) = 0.311$

Ball = Poké. 6 out of 16 examples. Caught = 1/6 - Not Caught = 5/6. $H(ball_{poke}) = 0.65$.

Ball = Great. 7 out of 16 examples. Caught = 4/7 - Not Caught = 3/7. $H(ball_{great}) = 0.985$.

Ball = Ultra. 3 out of 16 examples. Caught = 3/3 - Not Caught = 0/3. $H(ball_{ultra}) = 0$.

$$Gain(S, Ball) = 1 - ((\frac{6}{16})(0.65) + (\frac{7}{16})(0.985) + (\frac{3}{16})(0)) = 0.325$$

$Gain(S, Ball) = 0.325$

Color = Green. 3 out of 16 examples. Caught = 2/3 - Not Caught = 1/3. $H(color_{green}) = 0.918$.

Color = Yellow. 7 out of 16 examples. Caught = 3/7 - Not Caught = 4/7. $H(color_{yellow}) = 0.985$.

Color = Red. 6 out of 16 examples. Caught = 3/6 - Not Caught = 3/6. $H(color_{red}) = 1$.

$$Gain(S, Color) = 1 - ((\frac{3}{16})(0.918) + (\frac{7}{16})(0.985) + (\frac{6}{16})(1)) = 0.022$$

$Gain(S, Color) = 0.022$

Type = Normal. 6 out of 16 examples. Caught = 3/6 - Not Caught = 3/6. $H(type_{normal}) = 1$.

Type = Water. 4 out of 16 examples. Caught = 2/4 - Not Caught = 2/4. $H(type_{water}) = 1$.

Type = flying. 4 out of 16 examples. Caught = 3/4 - Not Caught = 1/4. $H(type_{flying}) = 0.811$.

Type = psychic. 2 out of 16 examples. Caught = 0/2 - Not Caught = 2/2. $H(type_{psychic}) = 0$.

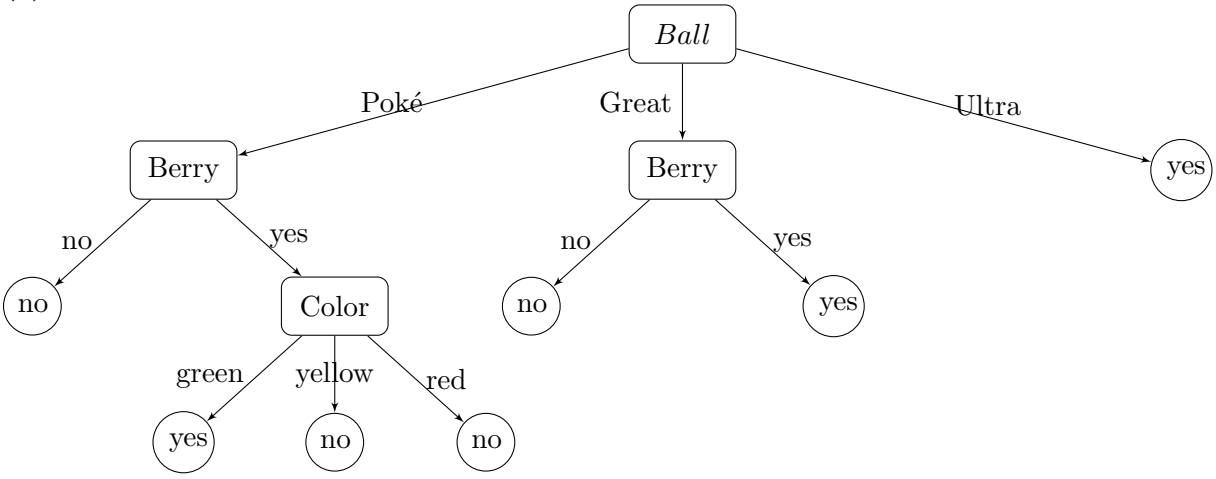
$$Gain(S, Type) = 1 - ((\frac{6}{16})(1) + (\frac{4}{16})(1) + (\frac{4}{16})(0.811) + (\frac{2}{16})(0)) = 0.172$$

$Gain(S, Type) = 0.172$

(d) When using the ID3 algorithm to create a decision tree, I would select the attribute Ball as the root of the tree.

Ball

(e)



The label 'no' on Ball[Poke] → Berry[yes] → Color[yellow] was chosen arbitrarily. The training examples with Ball[Poke] → Berry[Yes] have even probability between a 'yes' and 'no' label. The color yellow has no training data so a random choice was made.

(f)

Berry	Ball	Color	Type	Caught	Correct Prediction?
Yes	Great	Yellow	Psychic	Yes	Yes
Yes	Poké	Green	Flying	No	No
No	Ultra	Red	Water	No	No

Table 1: Predictions for test set

Out of the three examples in the test set, my decision tree predicted only one correct.

Accuracy = 33%

(g) I think using a decision tree to classify if a Pokémon will be caught or not is a good idea. The training set provided is very sparse and I would expect a better rate of accuracy given more training data.

(3) Using the Gini measure with the ID3 algorithm.

(a) Information gain is defined as:

$$Gain(S, A) = Gini(S) \; - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Gini(S_v)$$

Caught = Yes. 8 out of 16 examples. Caught = No. 8 out of 16 examples. $Gini(S) = 0.5$

Berry = Yes. 7 out of 16 examples. Caught = 6/7 - Not Caught = 1/7. $Gini(berry_{yes}) = 0.245$.
Berry = No. 9 out of 16 examples. Caught = 2/9 - Not Caught = 7/9. $Gini(berry_{no}) = 0.346$.

$$Gain(S, Berry) = 0.5 - ((\frac{7}{16})(0.245) + (\frac{9}{16})(0.346)) = 0.2$$

$Gain(S, Berry) = 0.2$

Ball = Poké. 6 out of 16 examples. Caught = 1/6 - Not Caught = 5/6. $Gini(ball_{poke}) = 0.278$.

Ball = Great. 7 out of 16 examples. Caught = 4/7 - Not Caught = 3/7. $Gini(ball_{great}) = 0.49$.

Ball = Ultra. 3 out of 16 examples. Caught = 3/3 - Not Caught = 0/3. $Gini(ball_{ultra}) = 0$.

$$Gain(S, Ball) = 0.5 - ((\frac{6}{16})(0.278) + (\frac{7}{16})(0.49) + (\frac{3}{16})(0)) = 0.181$$

$Gain(S, Ball) = 0.181$

Color = Green. 3 out of 16 examples. Caught = 2/3 - Not Caught = 1/3.
$Gini(color_{green}) = 0.444$.

Color = Yellow. 7 out of 16 examples. Caught = 3/7 - Not Caught = 4/7.
$Gini(color_{yellow}) = 0.49$.

Color = Red. 6 out of 16 examples. Caught = 3/6 - Not Caught = 3/6. $Gini(color_{red}) = 0.5$.

$$Gain(S, Color) = 0.5 - ((\frac{3}{16})(0.444) + (\frac{7}{16})(0.49) + (\frac{6}{16})(0.5)) = 0.015$$

$Gain(S, Color) = 0.015$

Type = Normal. 6 out of 16 examples. Caught = 3/6 - Not Caught = 3/6.
$Gini(type_{normal}) = 0.5$.

Type = Water. 4 out of 16 examples. Caught = 2/4 - Not Caught = 2/4. $Gini(type_{water}) = 0.5$.

Type = flying. 4 out of 16 examples. Caught = 3/4 - Not Caught = 1/4.
$Gini(type_{flying}) = 0.375$.

Type = psychic. 2 out of 16 examples. Caught = 0/2 - Not Caught = 2/2. $Gini(type_{psychic}) = 0$.

$$Gain(S, Type) = 0.5 - ((\frac{6}{16})(0.5) + (\frac{4}{16})(0.5) + (\frac{4}{16})(0.375) + (\frac{2}{16})(0)) = 0.094$$

$Gain(S, Type) = 0.094$

(b) I would pick Berry as the root of my decision tree. When using the Gini measure to calculate the information gain of each attribute, Berry has the highest information gain. This is different than when entropy is used to calculate information gain. When entropy is used, Ball is selected as the root.

Berry - different than when entropy is used

2: Linear Classifiers

(1) Weights **w** and a bias **b** classifies the dataset from problem 1.

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \textit{ and } b = -1$$

Looking at the dataset, one possible way to classify the data is either x_3 or x_4 must be set to produce an output of 1. Otherwise, the output is -1.

(2) The table below details the accuracy of the linear classifier from part 1 on the dataset from part 2:

x_1	x_2	x_3	x_4	o	*prediction*
1	0	1	1	1	1
0	1	0	1	1	1
1	0	1	0	1	1
1	1	0	0	1	-1
1	1	1	1	1	1
1	1	1	0	1	1
0	0	1	0	-1	1

Accuracy = 5/7 or 71.4%

(3) Weights **w** and a bias **b** classifies the combined dataset from part 1, 2, and 3.

$$\mathbf{w} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \textit{ and } b = -1$$

Looking at the combined dataset, we can see that either x_1 or x_4 must be set to produce an output of 1. Otherwise, the output is -1.

3: Setting A

(1a) I implemented my decision tree and the ID3 algorithm in Python. Python felt like the right language choice as it handles manipulating data well. For the representation of the tree, I used a Node and Edge class. I felt like this was the best choice since a node can have more than two children (binary tree won't do) and edges have labels on them. I wrote the file reading and handling with k-folding in mind, allowing multiple files as input data. This meant more development time upfront, but by making the code a little more generic I saved time come experimentation time.

For questions **1b**, **1c**, and **1d** my decision tree was trained on the **SettingA/training.data** file.

(1b) For the **SettingA/training.data** file, I had a 0% error rate with 3822/3822 examples correctly classified.

Error rate = 0%

(1c) For the **SettingA/test.data** file, I had a 0% error rate with 3822/3822 examples correctly classified.

Error rate = 0%

(1d) The maximum depth of my decision tree is three decisions.

Maximum depth = 3

(2a)

Tree Size	Cross-validation Accuracy	Standard Deviation
1	97.65%	0.051
2	98.14%	0.042
3	98.14%	0.042
4	98.14%	0.042
5	98.14%	0.042
10	98.14%	0.042
15	98.14%	0.042
20	98.14%	0.042

Table 2: Results for 6-fold cross-validation with various maximum tree sizes

Above are the results for 6-fold cross validation using the **SettingA/training_0X.data** files. We can see from the chart that the best accuracy comes from using a tree of size 2 or larger. I would choose a tree of size two, because a more sparse decision tree can help avoid overfitting.

(2b) I train a decision tree, with a maximum of size 2, on **SettingA/training.data**. When testing the **SettingA/test.data** file, I get an error rate of 0.22% with 1818/1822 examples correctly classified.

Error rate = 0.22%

3: Setting B

For questions **1a**, **1b**, **1c**, **1d** and **1e** my decision tree was trained on the **SettingB/training.data** file.

(1a) For the **SettingB/training.data** file, I had a 0% error rate with 3822/3822 examples correctly classified.

Error rate = 0%

(1b) For the **SettingB/test.data** file I had a 6.3% error rate with 1707/1822 examples correctly classified.

Error rate = 6.3%

(1c) For the **SettingA/training.data** file I had a 0.05% error rate with 3820/3822 examples correctly classified.

Error rate = 0.05%

(1d) For the **SettingA/test.data** file I had a 0.1% error rate with 1820/1822 examples correctly classified.

Error rate = 0.1%

(1e) The maximum depth of my decision tree is nine decisions.

Maximum depth = 9

(2a)

Tree Size	Cross-validation Accuracy	Standard Deviation
1	92.86%	0.0434
2	90.21%	0.0953
3	92.26%	0.043
4	92.7%	0.02
5	92.52%	0.0239
10	92.28%	0.0244
15	92.28%	0.0244
20	92.28%	0.0244

Table 3: Results for 6-fold cross-validation with various maximum tree sizes

Above are the results for 6-fold cross validation using the **SettingA/training_0X.data** files. We can see from the chart that the best accuracy comes from using a tree of size 1. I would choose a tree of size one, because a more sparse decision tree can help avoid overfitting and it has the best accuracy. Additionally, the standard deviations between the cross-validations errors is lower than a tree of size two, which is another good candidate.

Optimal tree size = 1

(2b) I train a decision tree, with a maximum of size 1, on **SettingB/training.data**. When testing the **SettingB/test.data** file, I get an error rate of 6.42% with 1705/1822 examples correctly classified.

Error rate = 6.42%

3: Setting C