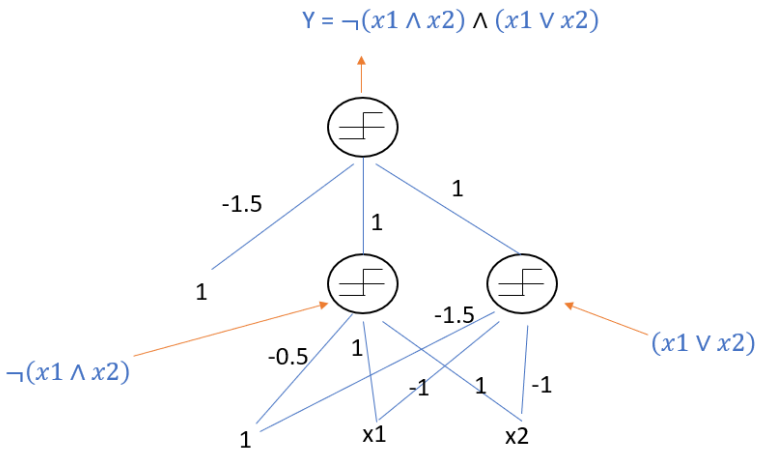
Assignment 3

1. Neural network expressiveness
   In class, we have discussed that neural network can express any arbitrary boolean functions. Please answer the following question about neural networks. You can use a step function for the activation function.

a. It is impossible to implement a XOR function $y = x_1 \oplus x_2$ using a single unit (neuron). However, you can do it with a neural net. Use the smallest network you can. Draw your network and show all the weights.

Answer:

An XOR function can be represented with $y = \neg(x1 \wedge x2) \wedge (x1 \vee x2)$. Using a neural network with multiple unit and 2 layers, we would be able to represent it, which is shown below.



Neural Network
With 2 layers : XOR

b. Explain how we can construct a neural network to implement a Naive Bayes Classifier with Boolean features?

Answer:

Suppose we have a Naïve Bayes classifier, which predicts y = 1 when P (X| y = 1) > P(X| y = 0). Taking log of the Bernouli model we can have the following:

$$P(X|y=1) = \sum_{i=1}^{N}(\log P(\ x_i = 1 \mid y = 1) + (\ 1 - \ x_i) \log P(x_i = 0 \mid y = 1)) + \log P(y = 1)$$

This is a linear function of x_i, whose coefficients are defined by

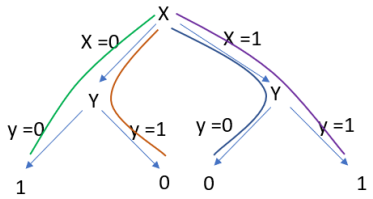
$$P\ (x_i = 1 \mid y = 1),\ P(x_i = 1 \mid y = 1)\ and\ P(y = 1)$$

So, we will be able to represent $P(X|y = 1)$ through a neural network. And will be able to compute $P(X|y = 0)$ similarly. After computing both $P(X|y = 1)\ and\ P(X|y = 0)$ we can use a unit in the final layer with an activation function that would return 1
if $P(X|y = 1) - \ P(X|y = 1) > 0$, 0 otherwise.

c. Explain how can we construct a neural network to implement a decision tree classifier with boolean features?

Answer:

Any decision tree can be represented as disjunctive normal form (using AND + OR). Let's look at the decision tree for X-NOR below:

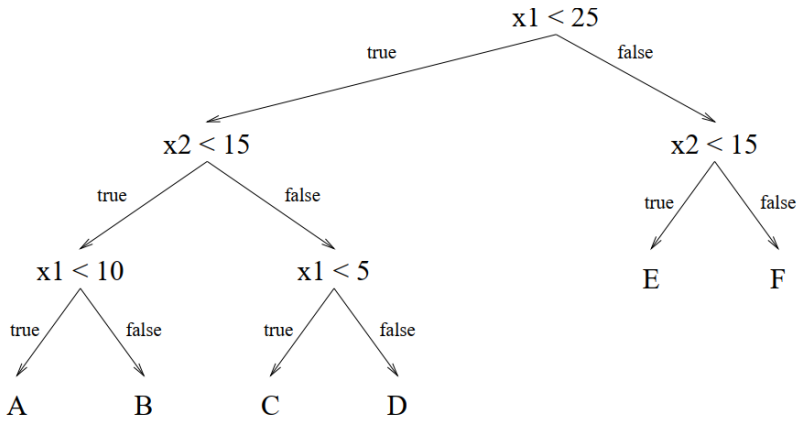


We can represent each path of the decision tree using 'AND', and to get the final Boolean function we 'OR' all the representations we received for each path. For example, the X-NOR decision tree above can be represented as the Boolean function below according to the way specified above:

$$y = (\neg x1 \wedge \neg x2) \vee (\neg x1 \wedge x2) \vee (x1 \wedge \neg x2) \vee (x1 \wedge x2)$$

And like the XOR in question (a) we will be able to represent this Boolean function using a neural network. So, by representing each path of our decision tree with AND and adding them with OR we would able to find the Boolean function that we would be able to represent using a neural network.

2. Consider the following decision tree:



(a) Draw the decision boundaries defined by this tree. Each leaf of the tree is labeled with a letter. Write this letter in the corresponding region of input space.

Answer:

The decision boundaries of the decision tree of the figure above is given below. {A, B, C, D, E, F} are the leaf labels which are shown to the corresponding regions in the decision boundary below.

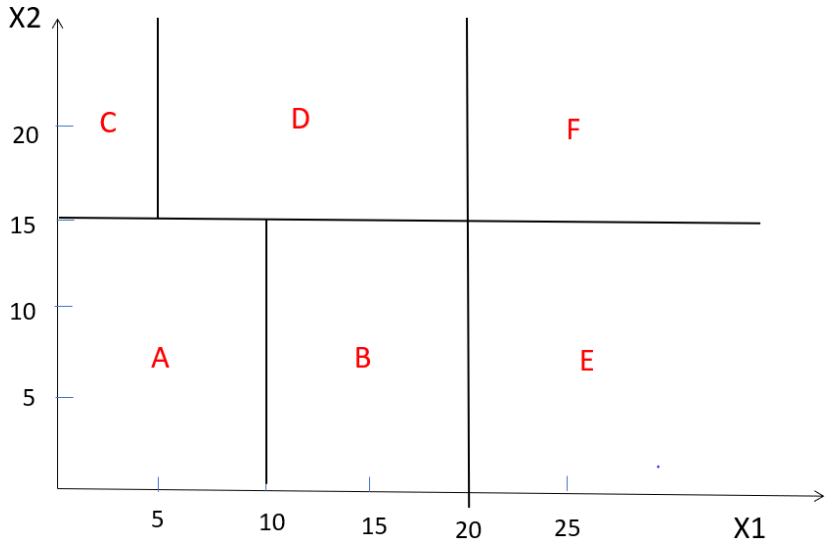
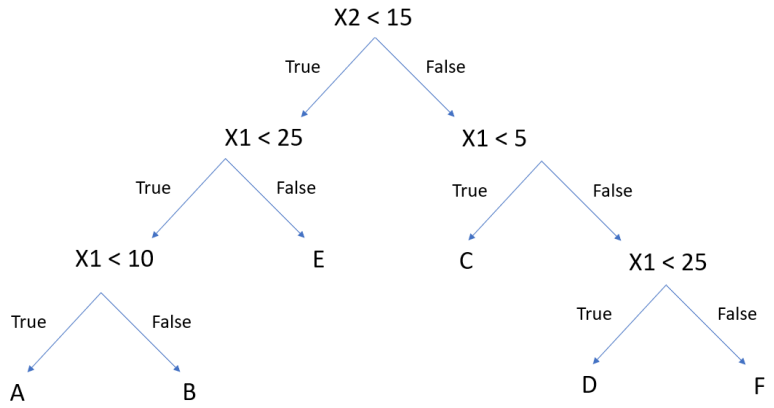


Figure 1

(b) Give another decision tree that is syntactically different but defines the same decision boundaries. This demonstrates that the space of decision trees is syntactically redundant. How does this redundancy influence learning (does it make it easier or harder to find an accurate tree)?

Answer:



The decision tree above is syntactically similar to the decision tree provided in the question but both the decision tree corresponds to the same decision boundary of figure 1.
This syntactical redundancy is beneficial for learning because it nullifies the effect of using subpar heuristics, in other words it increases the chance to achieve a good accuracy using any sequence of features for node expansion.

3. In the basic decision tree algorithm (assuming we always create binary splits), we choose the feature/value pair with the maximum information gain as the test to use at each internal node of the decision tree. Suppose we modified the algorithm to choose at random from among those feature/value combinations that had non-zero mutual information, and we kept all other parts of the algorithm unchanged

(a) What is the maximum number of leaf nodes that such a decision tree could contain if it were trained on *m* training examples?

Answer:

Maximum # of leaf nodes: m nodes.
The given decision tree chooses features with non-zero mutual information to do the splits for each node. So, if we split the node, both of its child will have non-zero training examples. No child will be a leaf node with zero training example. So, in the worst case we may have 'm' leaf nodes; which is also the case where we have maximum number of nodes. Where each leaf node corresponds to each training example.

(b) What is the maximum number of leaf nodes that a decision tree could contain if it were trained on *m* training examples using the original maximum mutual information version of the algorithm? Is it bigger, smaller, or the same as your answer to (a)?

Answer:

Maximum # of leaf nodes: m nodes.
If we choose the features for splitting based on maximum mutual information version of decision tree algorithm in the worst case, we will have the same number of leaf nodes as the number of training example. This number of leaf node is same to my answer of (a). The scenario is much clear for the decision tree made for question 4, which has 6 training examples and the decision tree generated also has 6 leaf nodes. Using the maximum mutual information doesn't guarantee minimization of leaf nodes cause for some steps we might have multiple features having the same information gain; when we need to randomly pick one.

(c) How do you think this change (using random splits vs. maximum information mutual information splits) would affect the accuracy of the decision trees produced on average? Why?

Answer:

In the worst case both random splits and maximum mutual information split decision tree algorithm will have the same decision tree will have same number of leaf nodes. But in average the size of the decision tree is smaller if we split a node while trying to maximize mutual information gain. This definitely impacts the accuracy of the generated decision trees cause, if we randomly split using irrelevant feature we are just dividing the training data into multiple parts without any information gain. So, we now have multiple similar subproblem with much smaller size of data which will give lower accuracy, cause using more data enables us to gain more information and much better accuracy.

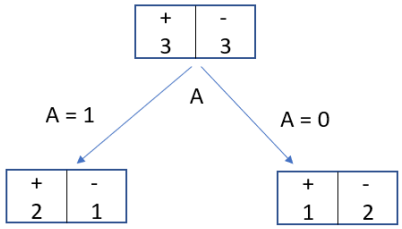
4. Consider the following training set:

A	B	C	Y
0	1	1	0
1	1	1	0
0	0	0	0
1	1	0	1
0	1	0	1
1	0	1	1

Learn a decision tree from the training set shown above using the information gain criterion.

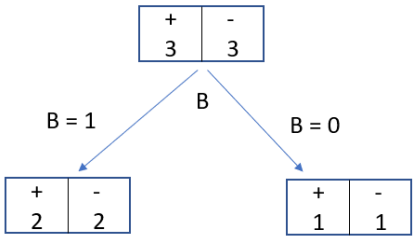
Answer:

Decision tree step 1: Root split based on A



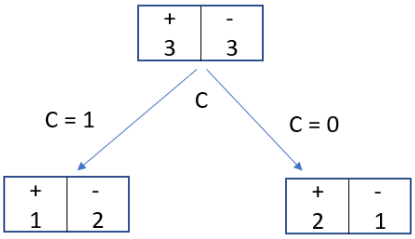
$$H(y\,|A=0) = -\frac{1}{3}\,log_2\frac{1}{3} - \frac{2}{3}\,log_2\frac{2}{3} = 0.92$$
$$H(y\,|A=1) = -\frac{2}{3}\,log_2\frac{2}{3} - \frac{1}{3}\,log_2\frac{1}{3} = 0.92$$
$$H(y) = P(A=0) * H(y\,|A=0) + P(A=1) * H(y\,|A=1)$$
$$= 0.5 * 0.92 + 0.5 * 0.92$$
$$= .92$$

Decision tree step 1: Root split based on B



$$H(y\,|B=0) = -\frac{1}{2}\,log_2\frac{1}{2} - \frac{1}{2}\,log_2\frac{1}{2} = 1$$
$$H(y\,|B=1) = -\frac{2}{4}\,log_2\frac{2}{4} - \frac{2}{4}\,log_2\frac{2}{4} = 1$$
$$H(y) = P(B=0) * H(y\,|B=0) + P(B=1) * H(y\,|B=1)$$
$$= 0.5 * 1 + 0.5 * 1$$
$$= 1$$

Decision tree step 1: Root split based on C



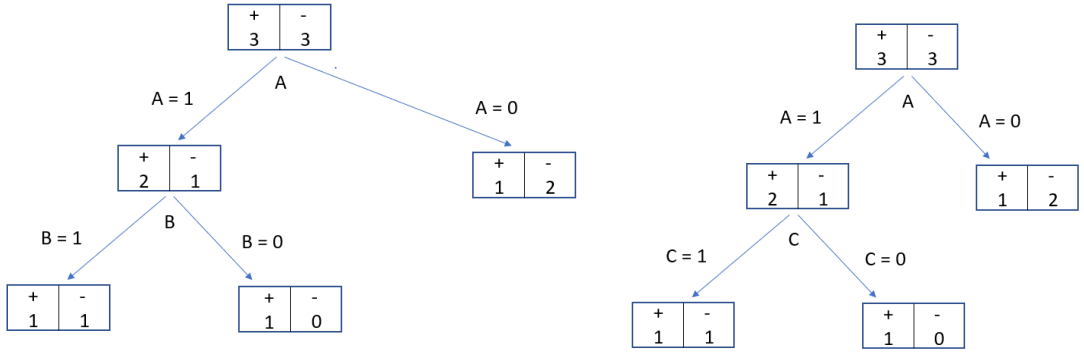
$$H(y\,|C=1) = -\frac{1}{3}\,log_2\frac{1}{3} - \frac{2}{3}\,log_2\frac{2}{3} = 0.92$$

$$H(y\,|C=0) = -\frac{2}{3}\,log_2\frac{2}{3} - \frac{1}{3}\,log_2\frac{1}{3} = 0.92$$

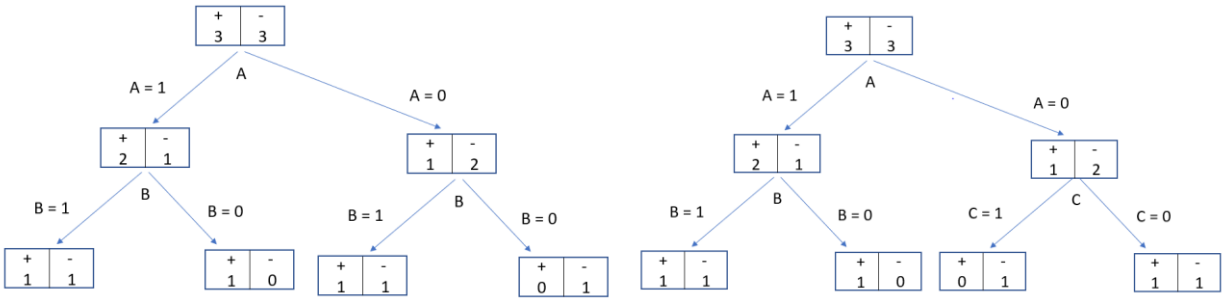
$$H(y) = P(C=0) * H(y\,|C=0) + P(C=1) * H(y\,|C=1)$$
$$= 0.5 * .92 + 0.5 * .92$$
$$= .92$$

So, from 3 different possible root node selection we can see that Information gain is higher if we split based on A and C (equal for both cases). We don't gain any information if we split based on B.

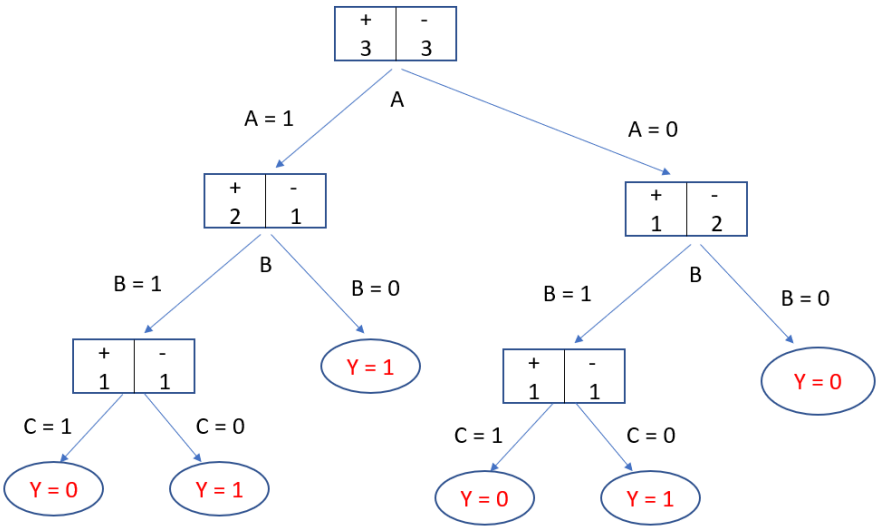
Assume, A has been picked randomly to be the root node. Now let's concentrate on the left child of A which can be split two ways using B and C. Which are shown in the following:



If we calculate uncertainty for B and C respectively we would see that H (y| B) = H (y | C), So information gain is same whether we split using feature B or C. Let us take feature B for splitting left child of A. Now let's concentrate on the right child of A (A = 0).



From the pictures above, we can see the Information gain for both B and C are same. We can randomly pick any of them and recurse the same procedure for the last uncertain node. Taking feature B for splitting the right child of A we get the following final decision tree after further splitting it using feature C. Which will give us the decision tree below:



5. Please show that in each iteration of Adaboost, the weighted error of h_i on the updated weights D_{i+1} is exactly 50%. In other words, $\sum_{j=1}^{N} D_{i+1}(j)\, I(h_i(X_j) = y_j)$ = 50%.

Answer:

Let us assume the weighted error of h_i is ε_i, where $\varepsilon_i = \sum_{j=1}^{N} D_i(j)\, I(h_i(X_j) = y_j)$ where I(.) is the identity function that returns 1 when the argument is true, which is in this case our hypothesis correctly predicts label y_j. Now adaboost uses the following $\alpha = \frac{1}{2}\ln\frac{(1-\varepsilon)}{\varepsilon}$ to achieve 0.5 error of h_i on the updated weights D_{i+1}. Adaboost multiplies the incorrect examples by $e^{-\alpha}$ and the correct examples by e^{α}. To achieve 50% error rate after updating h_i the following needs to be satisfied:

$$\varepsilon\, e^{\alpha} = (1-\varepsilon)e^{-\alpha}$$

$$\Rightarrow \frac{(1-\varepsilon)}{\varepsilon} = e^{2\alpha}$$

$$\Rightarrow \ln\frac{(1-\varepsilon)}{\varepsilon} = 2\alpha$$

$$\Rightarrow \alpha = \frac{1}{2} ln\frac{(1-\varepsilon)}{\varepsilon}$$

So, adaboost uses the correct α value that puts 50% error of h_i on the updated weights D_{i+1}

6. In class we showed that Adaboost can be viewed as learning an additive model via functional gradient descent to optimize the following exponential loss function:

$$\sum_{i=1}^{N} \exp(-y_i \sum_{l=1}^{L} \alpha_l h_l(x_i))$$

Our derivation showed that in each iteration l, to minimize this objective we should seek an h_l that minimizes the weighted training error, where the weight of each example $w_l^i = \exp(-y_i \sum_{t=1}^{l-1} \alpha_t h_t(x_i))$ prior to normalization. Show how this definition of w_l^i is proportional to the $D_l(i)$ defined in Adaboost.

Answer:

In Adaboost, if an example is correct the weight is multiplied by exp($-\alpha$) and if incorrect we multiply it with exp(α). Now we can see the sign of α then depends on the correctness of the example. So, if the example is positive $y_i h_{l-1}(x_i) = 1$, if the example is negative $y_i h_{l-1}(x_i) = -1$. We can define the sign of α_{l-1} using $y_i h_{l-1}(x_i)$, which gives us a general form exp($-\alpha_{l-1}\, y_i h_{l-1}(x_i)$). Now, we can write the following:

$$D_l(i) = D_{l-1}(i) * \exp(-\alpha_{l-1}\, y_i h_{l-1}(x_i))$$

So,

$$D_l(i) \propto \exp(-\alpha_{l-1}\, y_i h_{l-1}(x_i))$$

$$D_l(i) \propto \exp(-\alpha_1\, y_i h_1(x_i) - \alpha_2\, y_i h_2(x_i) \dots\dots\dots - \alpha_{l-1}\, y_i h_{l-1}(x_i))$$

$$D_l(i) \propto \exp(\, y_i(-\alpha_1 \quad h_1(x_i) - \alpha_2 \quad h_2(x_i) \dots\dots\dots - \alpha_{l-1} \quad h_{l-1}(x_i)\,))$$

$$D_l(i) \propto \exp(-y_i \sum_{t=1}^{l} \alpha_t\, y_i h_t(x_i))$$

$$D_l(i) \propto \exp(w_l(i))$$

$$D_l(i) \propto w_l(i)$$

(proved)