CS534 — Written Homework Assignment 3 — Due Nov 17th 23:59PM, 2018

Please submit electronically via TEACH in a single pdf file.

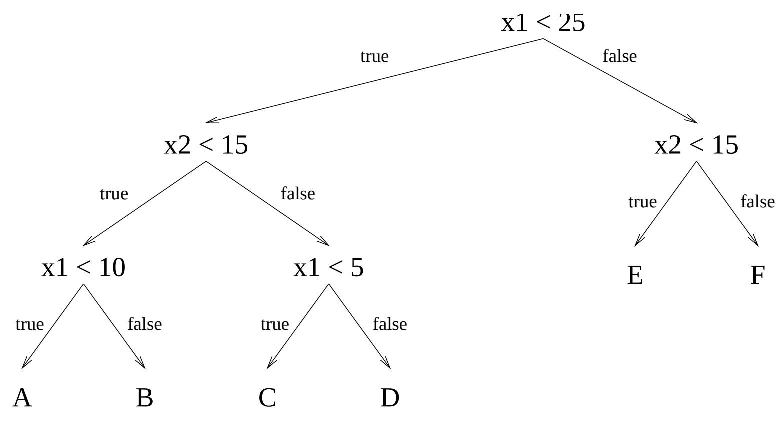
1. Neural network expressiveness
   In class, we have discussed that neural network can express any arbitrary boolean functions. Please answer the following question about neural networks. You can use a step function for the activation function.

   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.
   A XOR function can be represented as $(x_1 \wedge \neq x_2) \vee (\neg x_1 \wedge x_2)$. *This can be represented using two AND units and a OR units as described in class.*

   b. Explain how can we construct a neural network to implement a Naive Bayes Classifier with Boolean features.
   Without loss of generality, here we focus on a binary Naive Bayes classifier in the form that predicts $y = 1$ *if* $P(X, y = 1) \geq P(X, y = 0)$. *To implement this, we can compute* $\log P(X, y = 1) = \sum_i \log P(x_i|y = 1) + \log P(y = 1)$. *Because* x_i *is boolean, we have* $\log P(X, y = 1) = \sum_i (x_i \log P(x_i = 1|y = 1) + \log P(x_i = 0|y = 1)(1 - x_i)) + \log P(y = 1)$, *which is essentially a linear function of* x_i*'s, and the coefficients are determined by* $P(x_i = 1|y = 1), P(x_i = 0|y = 1), P(y = 1)$ *etc. We can compute* $P(X, y = 0)$ *in a similar fashion. We can then use a hard threshold function as the activation function to output 1 if* $\log P(X, y = 1) - \log P(X, y = 0) \geq 0$.

   c. Explain how can we construct a neural network to implement a decision tree classifier with boolean features.
   Any decision tree can be represented as a logical formula in disjunctive normal form. Specifically, each path in a decision tree can be represented as a collection of conjunctions (AND), and the paths are put together using disjunctions (OR). We can easily construct a hidden node for each path and the output layer simply implements a OR gate to represent the decision tree.

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.

   See Figure 1 for the decision boundary. Note that we label A, B, C, D, E, F with binary labels, for example $\{A, B, C\}$ = *positive, and* $\{D, E, F\}$ =*negative, then we can further determine that some of the line segments do not separate the two classes and thus omitted from this figure.*

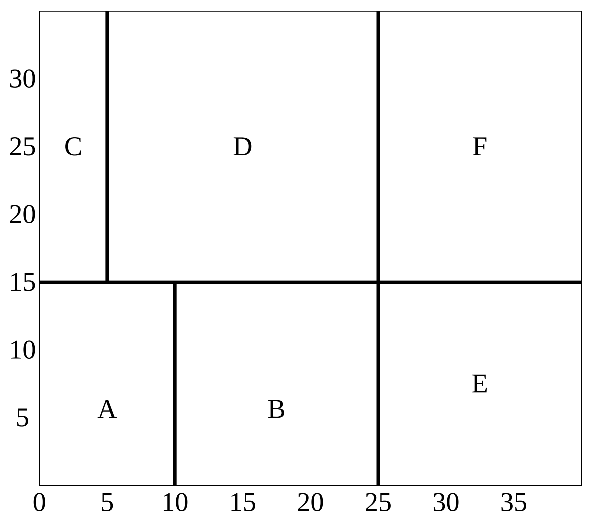


Figure 1: Decision boundary

(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)?

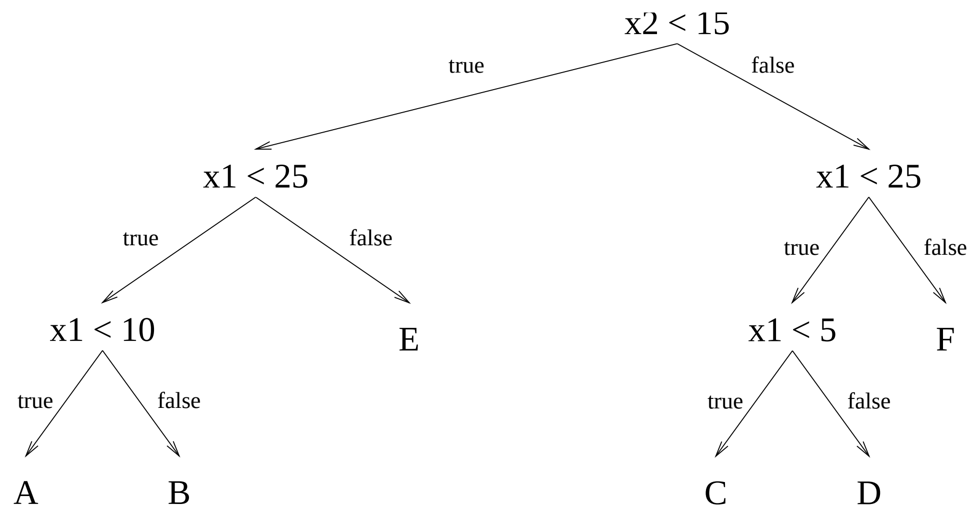


Figure 2: Alternative decision tree.

See Figure 2 for the alternative tree. The redundancy is likely to be a computational advantage, because it makes it easier for an imperfect greedy heuristic to find a good solution. For depth-first search methods (such as our greedy algorithm), the ideal search space is one such that every path leads to a solution. Redundancy increases the chance that any random sequence of node expansions will lead to a good tree.

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?

A learned tree can have no leaves with zero training examples, because otherwise it will lead to zero information gain. The maximum number would be obtained if each training example were alone in its own leaf: m leaves. (Actually, let m' be the number of distinct training examples. Then no tree can have more than m' leaves.)

(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 (b)?

Using maximum mutual information, in the worst case we could also get one example in each leaf node. (or m' distinct examples and leaves). But on average, we will have smaller number of leaves.

(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?

Although in the worst case, both decision trees will have the same size, we expect that in the average case, using randomized splits would give lower accuracy, particularly if there are irrelevant or noisy features in the data. A random split is more likely to split on an irrelevant or inappropriate feature. This will have the unfortunate result of subdividing the data randomly. This effectively creates two learning problems equivalent to the original learning problem, but each has only half as much data. We know that the more data we have available, the more accurate the hypothesis will be on average. Conversely, the less data we have available, the less accurate the hypothesis will be on the average. Therefore, we expect the mutual information selection method to produce more accurate trees on average.

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.

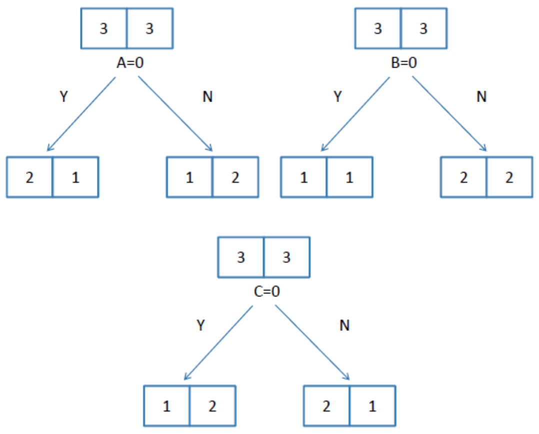


Figure 3: Decision tree step 1.

See Figure 3 for root node selection.

$$H(y|A=0) = -\frac{1}{3}log_2\frac{1}{3} - \frac{2}{3}log_2\frac{2}{3} = 0.9183$$

$$H(y|A=1) = -\frac{1}{3}log_2\frac{1}{3} - \frac{2}{3}log_2\frac{2}{3} = 0.9183$$

$$H(Y|A) = p(A=0) * H(y|A=0) + p(A=1) * H(y|A=1) = 0.9183$$

$$H(y|B=0) = -\frac{1}{2}log_2\frac{1}{2} - \frac{1}{2}log_2\frac{1}{2} = 1$$

$$H(y|B=1) = -\frac{1}{2}log_2\frac{1}{2} - \frac{1}{2}log_2\frac{1}{2} = 1$$

$$H(Y|B) = p(B=0) * H(y|B=0) + p(B=1) * H(y|B=1) = 1$$

$$H(y|C=0) = -\frac{1}{3}log_2\frac{1}{3} - \frac{2}{3}log_2\frac{2}{3} = 0.9183$$

$$H(y|C=1) = -\frac{1}{3}log_2\frac{1}{3} - \frac{2}{3}log_2\frac{2}{3} = 0.9183$$

$$H(Y|C) = p(C=0) * H(y|C=0) + p(C=1) * H(y|C=1) = 0.9183$$

Since A and C give the same amount of information gain, we randomly pick C as the root node test.

For the next step, we focus on the left branch. See Figure 4. It is clear from the figure without computation that B is a better choice. We can continue and finally reach the decision tree shown in

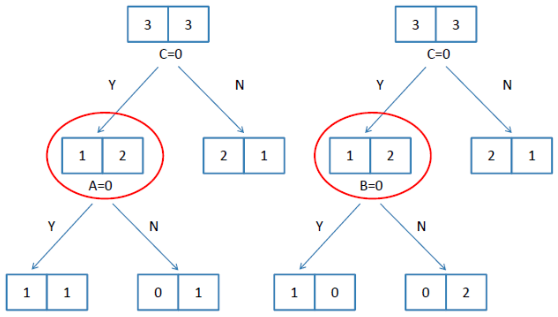


Figure 4: Decision tree step 2.

Figure 5. Note that if you happen to choose A as the root node, the resulting decision tree will be bigger, but correct none-the-less.

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, $\sum_{j=1}^{N} D_{i+1}(j)I(h_i(X_j) \neq y_j) = 50\%$.

   Solution: *Let ϵ_i be the weighted error of h_i, that is $\epsilon_i = \sum_{j=1}^{N} D_i(j)I(h_i(X_j) \neq y_j)$, where $I(\cdot)$ is the indicator function that takes value 1 if the argument is true, and value 0 otherwise. Following the update rule of Adaboost, let's assume that the weights of the correct examples are multiplied by $e^{-\gamma}$, and those of the incorrect examples are multiplied by e^{γ}. After the updates, to make sure that h_i has exactly 50% accuracy, we only need to satisfy the following:*

$$\epsilon e^{-\gamma} = (1-\epsilon)e^{\gamma} \Rightarrow$$

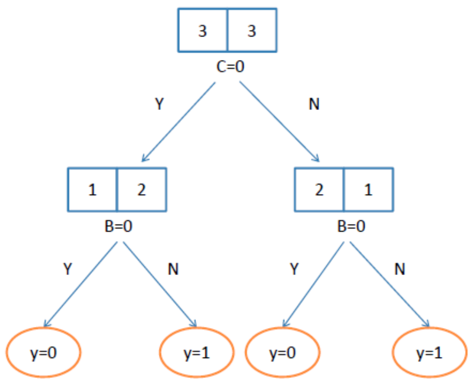


Figure 5: Final decion tree for problem 3.

$$\frac{\epsilon}{1-\epsilon} = e^{2\gamma} \Rightarrow$$
$$\log\frac{\epsilon}{1-\epsilon} = 2\gamma \Rightarrow$$
$$\gamma = \frac{1}{2}\log\frac{\epsilon}{1-\epsilon}$$

which is exactly the value Adaboost uses.

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.

In Adaboost, we have:

$$D_1 = 1/N$$

and in each iteration l, the weight is multiplied by a factor $\exp^{\alpha_{l-1}}$ for examples that are mistaken (i.e., $y_i h_{l-1}(x_i) = -1$), and $\exp^{-\alpha_{l-1}}$ for examples that are correct (i.e., $y_i h_{l-1}(x_i) = 1$).

In short, this can be expressed as:

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

As such, we have

$$D_l(i) \propto \exp^{-\alpha_1 y_i h_1(x_i) - \alpha_2 y_i h_2(x_i) - \ldots - \alpha_{l-1} y_i h_{l-1}(x_i)} = \exp^{-y_i \sum_{t=1}^{l-1} \alpha_t h_t(x_i)} = w_l^i$$