Joshua Sloan
University of Wyoming
jsloan3@uwyo.edu

# Homework 2
COSC 5010 - Data Mining

## 1 Problems from the Book

### 1.1 Chapter 4 - Problem 1

Parity functions must return 1 is there are an even number of Boolean attributes that are equal to 1 and return 0 is there is are an odd number of Boolean attributes that are equal to 1.

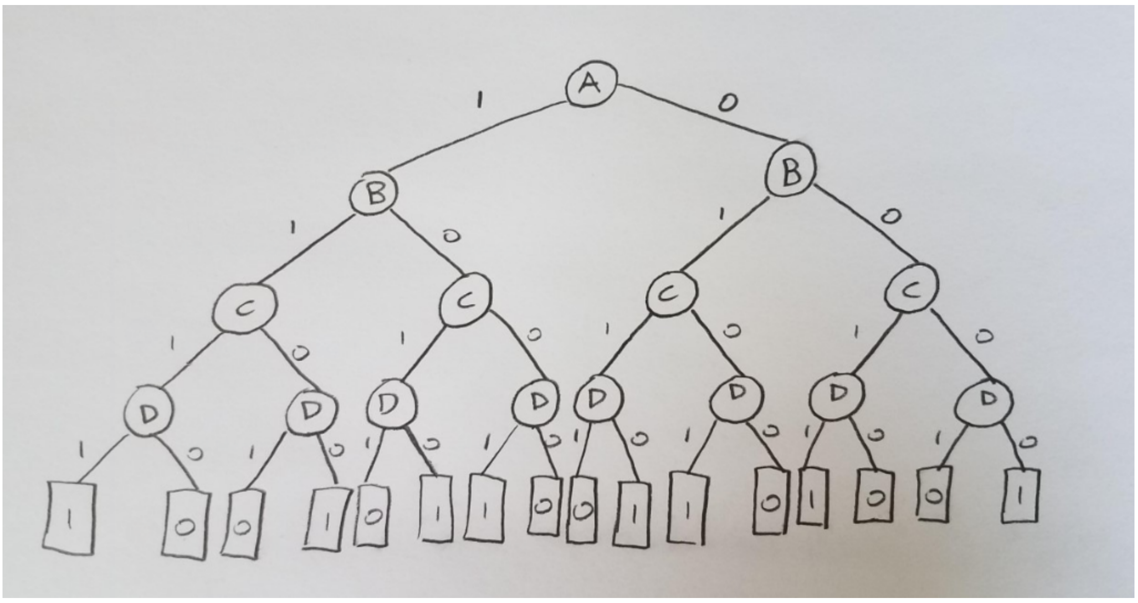

Figure 1: Decision Tree of Parity Function

No, this tree cannot be simplified. Because the output depends on all other attribute values to determine whether or not the parity function must return 0 or 1, you cannot prune any branches and account for all cases. Each branch corresponds to a unique case.

### 1.2 Chapter 4 - Problem 5

**a)**

Baseline Entropy of the given data:

$E_{base} = -\frac{4}{10} * log_2(\frac{4}{10}) - \frac{6}{10} * log_2(\frac{6}{10}) \approx 0.97095$

Now we must compute the information gain from A and B to decide which attribute to use for splitting.

For A, we can compute information gain as:

$E_{A1} = -\frac{4}{7} * log_2(\frac{4}{7}) - \frac{3}{7} * log_2(\frac{3}{7}) \approx 0.98523$

$E_{A0} = -\frac{3}{3} * log_2(\frac{3}{3}) - \frac{0}{3} * log_2(\frac{0}{3}) = 0$

S.T. $Gain_A = E_{base} - \frac{7}{10} * E_{A1} - \frac{3}{10} * E_{A0} \approx 0.281289$

Now for B, we can compute information gain as:

$E_{B1} = -\frac{3}{4} * log_2(\frac{3}{4}) - \frac{1}{4} * log_2(\frac{1}{4}) \approx 0.81128$

$E_{B0} = -\frac{1}{6} * log_2(\frac{1}{6}) - \frac{5}{6} * log_2(\frac{5}{6}) \approx 0.65002$

S.T. $Gain_B = E_{base} - \frac{4}{10} * E_{B1} - \frac{6}{10} * E_{B0} \approx 0.256426$

$\therefore$ attribute A would be chosen for the split when using entropy based information gain. $\square$

**b)**

Baseline Gini of the given data:

$Gini_{base} = 1 - (\frac{4}{10})^2 - (\frac{6}{10})^2 = 0.48$

Now we must compute the Gini gain from A and B to decide which attribute to use for splitting.

For A, we can compute Gini gain as:

$Gini_{A1} = 1 - (\frac{4}{7})^2 - (\frac{3}{7})^2 \approx 0.4898$

$Gini_{A0} = 1 - (\frac{3}{3})^2 - (\frac{0}{3})^2 = 0$

S.T. $Gain_A = Gini_{base} - \frac{7}{10} * Gini_{A1} - \frac{3}{10} * Gini_{A0} \approx 0.13714$

Now for B, we can compute Gini gain as:

$Gini_{B1} = 1 - (\frac{3}{4})^2 - (\frac{1}{4})^2 = 0.375$

$Gini_{B0} = 1 - (\frac{1}{6})^2 - (\frac{5}{6})^2 \approx 0.2778$

S.T. $Gain_B = Gini_{base} - \frac{4}{10} * Gini_{B1} - \frac{6}{10} * Gini_{B0} \approx 0.16332$

$\therefore$ attribute B would be chosen for the split when using Gini gain. $\square$

**c)**

Yes, the results from part a) and b) are an excellent example of this. While both entropy and Gini index are monotonously increasing on the range [0,0.5] and monotonously decreasing in the range [0.5,1], we can see that they can arrive at a different solution even when given the exact same data. Therefore, it is possible that information gain and Gini gain may favor different attributes. $\square$

## 1.3 Chapter 4 - Problem 6

**a)**

To determine the best attribute for splitting for our root node, we must first compute the classification error rate for all 3 attributes.

C1X$_0$ = 5 + 0 + 10 + 45 = 60
C1X$_1$ = 10 + 25 + 5 + 0 = 40
C2X$_0$ = 40 + 15 + 5 + 0 = 60
C2X$_1$ = 5 + 0 + 20 + 15 = 40
S.T. Error$_X$ = $(\frac{120}{200} * \frac{60}{120}) + (\frac{80}{200} * \frac{40}{80}) = 0.5$

C1Y$_0$ = 5 + 0 + 10 + 25 = 40
C1Y$_1$ = 10 + 45 + 5 + 0 = 60
C2Y$_0$ = 40 + 15 + 5 + 0 = 60
C2Y$_1$ = 5 + 0 + 20 + 15 = 40
S.T. Error$_Y$ = $(\frac{100}{200} * \frac{40}{100}) + (\frac{100}{200} * \frac{40}{100}) = 0.4$

C1Z$_0$ = 5 + 10 + 10 + 5 = 30
C1Z$_1$ = 0 + 45 + 25 + 0 = 70
C2Z$_0$ = 40 + 5 + 5 + 20 = 70

$C2Z_1 = 15 + 0 + 0 + 15 = 30$
S.T. $\text{Error}_Z = (\frac{100}{200} * \frac{30}{100}) + (\frac{100}{200} * \frac{30}{100}) = 0.3$

$\therefore$ the best attribute to select for our root node is Z (given that it has the lowest classification error).

Now to compute the next attribute, we must repeat the process given each remaining attribute value after the data has been split according to the value of attribute Z, such that:

$C1X_{Z0X0} = 5 + 10 = 15$
$C1X_{Z0X1} = 10 + 5 = 15$
$C2X_{Z0X0} = 40 + 5 = 45$
$C2X_{Z0X1} = 5 + 20 = 25$
S.T. $\text{Error}_{Z0X} = (\frac{60}{100} * \frac{15}{60}) + (\frac{40}{100} * \frac{15}{40}) = 0.3$

$C1Y_{Z0Y0} = 5 + 10 = 15$
$C1Y_{Z0Y1} = 10 + 5 = 15$
$C2Y_{Z0Y0} = 40 + 5 = 45$
$C2Y_{Z0Y1} = 5 + 20 = 25$
S.T. $\text{Error}_{Z0Y} = (\frac{60}{100} * \frac{15}{60}) + (\frac{40}{100} * \frac{15}{40}) = 0.3$

and

$C1X_{Z1X0} = 0 + 45 = 45$
$C1X_{Z1X1} = 25 + 0 = 25$
$C2X_{Z1X0} = 15 + 0 = 15$
$C2X_{Z1X1} = 0 + 15 = 15$
S.T. $\text{Error}_{Z1X} = (\frac{60}{100} * \frac{15}{60}) + (\frac{40}{100} * \frac{15}{40}) = 0.3$

$C1Y_{Z1Y0} = 0 + 25 = 25$
$C1Y_{Z1Y1} = 45 + 0 = 45$
$C2Y_{Z1Y0} = 15 + 0 = 15$
$C2Y_{Z1Y1} = 0 + 15 = 15$
S.T. $\text{Error}_{Z1Y} = (\frac{40}{100} * \frac{15}{40}) + (\frac{60}{100} * \frac{15}{60}) = 0.3$

In this case, both X and Y has the same error rate when Z = 0 and Z = 1,
$\therefore$ it does not matter which attribute is selected next and we can decide arbitrarily, or just choose X if we are settling ties alphabetically (by attribute name).

The overall error rate of the tree is $(\frac{60}{200} * \frac{15}{60}) + (\frac{40}{200} * \frac{15}{40}) + (\frac{60}{200} * \frac{15}{60}) + (\frac{40}{200} * \frac{15}{40}) = 0.3$

**b)**

Since we have already chosen X as the root node, we now need to compute the classification error rate for both Y and Z to determine the best attribute for our next split after the data has been split according to the attribute value of X, such that:

$C1Y_{X0Y0} = 5 + 0 = 5$
$C1Y_{X0Y1} = 10 + 45 = 55$
$C2Y_{X0Y0} = 40 + 15 = 55$
$C2Y_{X0Y1} = 5 + 0 = 5$
S.T. $\text{Error}_{X0Y} = (\frac{60}{120} * \frac{5}{60}) + (\frac{60}{120} * \frac{5}{60}) \approx 0.0833$

$C1Z_{X0Z0} = 5 + 10 = 15$
$C1Z_{X0Z1} = 0 + 45 = 45$
$C2Z_{X0Z0} = 40 + 5 = 45$
$C2Z_{X0Z1} = 15 + 0 = 15$
S.T. $\text{Error}_{X0Z} = (\frac{60}{120} * \frac{15}{60}) + (\frac{60}{120} * \frac{15}{60}) = 0.25$

and

C1Y$_{X1Y0}$ = 10 + 25 = 35
C1Y$_{X1Y1}$ = 5 + 0 = 5
C2Y$_{X1Y0}$ = 5 + 0 = 5
C2Y$_{X1Y1}$ = 20 + 15 = 35
S.T. Error$_{X1Y}$ = $(\frac{40}{80} * \frac{5}{40}) + (\frac{40}{80} * \frac{5}{40}) = 0.125$

C1Z$_{X1Z0}$ = 10 + 5 = 15
C1Z$_{X1Z1}$ = 25 + 0 = 25
C2Z$_{X1Z0}$ = 5 + 20 = 25
C2Z$_{X1Z1}$ = 0 + 15 = 15
S.T. Error$_{X1Z}$ = $(\frac{40}{80} * \frac{15}{40}) + (\frac{40}{80} * \frac{15}{40}) = 0.375$

In this case, Y has the lowest classification error for X=0 and X=1
$\therefore$ Y will be the next attribute used to split the decision tree.

The overall error rate of the tree is $(\frac{60}{200} * \frac{5}{60}) + (\frac{60}{200} * \frac{5}{60}) + (\frac{40}{200} * \frac{5}{40}) + (\frac{40}{200} * \frac{5}{40}) = 0.1$

**c)**

Looking at our results from part a) and b), we can see that the overall error rate of the tree from part a) is greater than that of part b), showing that a greedy approach does not always produce an optimal solution. There may be a variety of factors that determine an optimal solution (such as attribute correlation) which a greedy approach is not able to encapsulate. Therefore, while a greedy approach might be able to give a good solution (given that is at least is always optimizing the problem based on the current state), there is no way of knowing that the optimized tree is the best possible optimization, given that a greedy algorithm cannot look many steps ahead to see if there is an alternative path with less optimal immediate decisions which can lead to a solution which is more optimal overall.

## 1.4 Chapter 4 - Problem 9

Given that the data set contains 16 binary attributes, the cost of encoding each attribute (internal tree node) is log$_2$(16) bits = 4 bits.

Given that there are 3 classes (i.e. $C_1, C_2$ and $C_3$), the cost of encoding each class (leaf node) is log$_2$(3) bits = 2 bits.

The cost of encoding each classification error is log$_2$(n) bits.

Decision tree (a) with 7 errors has (2*4)+(3*2)+(7*log$_2$(n)) = 8 + 6 + 7log$_2$(n) = 14 + 7log$_2$(n) bits.

Decision tree (b) with 4 errors has (4*4)+(5*2)+(4*log$_2$(n)) = 16 + 10 + 4log$_2$(n) = 26 + 4log$_2$(n) bits.

In order to determine which tree is better according to the MDL principle, we must find the range of values for n (i.e. the total number of training instances) that are associated with a better tree.

Setting the tree costs equal to one another, we have:
14 + 7log$_2$(n) = 26 + 4log$_2$(n)
$\Rightarrow$ 12 = 3log$_2$(n)
$\Rightarrow$ 4 = log$_2$(n)
$\Rightarrow$ 16 = n,
meaning that when n = 16, both trees have equivalent cost.

$$\therefore \text{Best Tree} = \begin{cases} \text{(a)}, \text{n} < 16 \\ \text{either}, \text{n} = 16 \\ \text{(b)}, \text{n} > 16 \end{cases} \tag{1}$$

□

### 1.5 Chapter 5 - Problem 6

Given: P(G) = 0.2, P(U) = 0.8, P(S|G) = 0.23 and P(S|U) = 0.15.
From this we can infer P(S) = P(G)*P(S|G) + P(U)*P(S|U) = 0.2 * 0.23 + 0.8 * 0.15 = 0.166.

**a)**
P(G|S) = (P(S|G) * P(G))/P(S) = (0.23 * 0.2)/0.166 = 0.271 or 27.1%. □

**b)**
A randomly selected student is more likely to be a undergraduate student given that P(U) $>$ P(G). □

**c)**
P(U|S) = (P(S|U) * P(U))/P(S) = (0.15 * 0.8)/0.166 = 0.723 or 72.3%.

$\therefore$ a randomly selected student who smokes is more likely to be an undergraduate student, given that P(U|S) $>$ P(G|S). □

**d)**
Now we have the added information that: P(D|G) = 0.3, P(D|U) = 0.1.
From this we can infer P(D) = P(G)*P(D|G) + P(U)*P(D|U) = 0.2 * 0.3 + 0.8 * 0.1 = 0.14.

We need to compute P(G|D,S) and P(U|D,S), however, we can assume independence between students who live in the dorm and those who smoke, such that:

P(D,S|G) = P(D|G) * P(S|G) = 0.3 * 0.23 = 0.069
and
P(D,S|U) = P(D|U) * P(S|U) = 0.1 * 0.15 = 0.015.

This also means that that P(D,S) = P(D) * P(S) = 0.14 * 0.166 = 0.02324.

Using Bayes Thm. we have:
P(G|D,S) = (P(D,S|G)*P(G))/P(D,S) = (0.069*0.2)/0.02324 $\approx$ 0.5938
and
P(U|D,S) = (P(D,S|U)*P(U))/P(D,S) = (0.015*0.8)/0.02324 $\approx$ 0.5164

$\therefore$ if a student smokes and lives in the dorms, then they are more like to be a graduate student (given that P(G|D,S) $>$ P(U|D,S)). □

## 2 Decision Tree Learning

### 2.1 What does zero entropy mean?

When we calculate zero entropy at a given node, that means that the node is completely homogeneous. In other words, all instances belong to a single class, meaning that the node is completely pure and we have obtained an ideal decision to arrive at a final prediction class label.

### 2.2 What is maximum value for the entropy of a random variable that can take n-values? justify.

The maximum entropy we can calculate for a random variable that can take n-values is $\log_2(n)$. This maximum occurs when a node is completely heterogeneous, meaning that all of the instances are equally distributed among all of the possible n-value class labels. In other words, none of the attributes values tell us anything more about which class an instance belongs to (based off of the attribute value it takes). In this manner, we are merely separating out instances into their attribute specific values, without getting any closer to arriving at a class label decision.

For example, when n=2, the resulting entropy calculation would be:
$-(\frac{1}{2} * log_2(\frac{1}{2}) + \frac{1}{2} * log_2(\frac{1}{2}))$ = 1 = $\log_2(2)$.
When n=3, the resulting entropy calculation would be:
$-(\frac{1}{3} * log_2(\frac{1}{3}) + \frac{1}{3} * log_2(\frac{1}{3}) + \frac{1}{3} * log_2(\frac{1}{3})) \approx 1.585 \approx log_2(3)$.
etc.

**Note that the resulting fractions are the same when more instances appear in each split, as long as their ratio remains the same(i.e. $\frac{1}{3} = \frac{2}{6} = \frac{3}{9}$ etc.).**

If any one class value has more instances attributed to it than another class, then the entropy will only drop from this value.

## 2.3 What kind of real attributes create problems for entropy-based decision trees. How can we solve this problem?

Continuous values are a very common attribute which can create problems for entropy-based decision trees. If left alone, every single unique continuous value will constitute its own group, meaning that the values 1.23456789 and 1.23456788 will be treated just as differently as 1.23456789 and 10. The only way that two continuous values will belong to the same group is if they both correspond to the EXACT same value, which depending on the resolution of the data, may be increasingly improbable (and therefore is unlikely to be useful as a decision point for a class label value prediction).

This is not ideal, as continuous values often give us very important insight regarding label prediction, but just cannot be captured well by entropy-based decision trees in their base state. To combat this however, we can engineer our continuous data to be more useful by applying a condition to the continuous data (binary decision), or by binning ranges of the continuous spectrum into larger groups (discretization). By doing so, we can extract the relevant information contained within the continuous values in a manner which is conducive to entropy-based decision trees. Therefore, with properly data engineering, we can obtain much lower entropy values, thus increasing the likelihood that we will utilize our continuous data for decision tree generation.

Additionally, to further utilize continuous attribute values, we have the option the build an oblique decision tree, which utilizes a combination of continuous attributes before applying a condition or binning (discretizing) the subsequent result. While determining the best way to do this might be computationally expensive, it can be a very powerful way of generating a useful split point for a decision tree.

Another real attribute type which creates similar problems is unique nominal attributes (i.e. attributes where we are only likely to ever see 1 value associated with 1 instance such as a name or student ID). These sort of values tend to be less useful for class prediction, as it is very likely that the corresponding nodes of this attribute will be complete heterogeneous. For this reason, we can simply omit this attribute during tree construction (if our algorithm doesn't inherently). There may be a unique case where a seemingly nominal attribute is actually ordinal (i.e. perhaps a student ID is assignment by a chronologically increasing value index), such that you can apply a binary decision (all IDs less than a certain value indicate that this student started in X semester) or discretization (all student IDs in this range are currently Freshman), but this requires advanced domain knowledge which may not be available/commonly known.

Finally, we have missing attribute values (which although are not a type of attribute, they can occur for a given attribute). The best response depends on the sparsity of a particular attribute. If there are very few missing values, you can simply fill them in using the data you do have available to you (i.e. attribute specific median, mode, mean, etc.) or, if there is a large number of missing values for a given attribute, it might simply be in your best interest to omit the attribute from your potential selection of a split point within your decision tree.

## 2.4 Describe pre-pruning and post-pruning techniques for dealing with decision tree overfitting.

In order to help prevent a tree from overfitting, we want to restrict the depth of the tree (such that we are not learning decision rules which are too specific to the training data). In other words, we need ensure that our tree generalizes well on unseen data, which can be achieved by tolerating some error in our decision tree on the training data (but not too much as to cause the model to underfit). We can achieve this by two means, pre-pruning and post-pruning.

**Pre-pruning:**
In pre-pruning we want to catch any branches which might be problematic for overfitting the training data during the construction of the tree, so that when we are finished, we know our tree will likely generalize well on the testing data. This is achieved by early stopping based off of a variety of implementable rules (depending on the specific application) after a node specific analysis during tree construction.

Examples may include:
Stopping a node if all instances belong to the same class, stopping a node if all instances have the same attribute value, stopping a node if the total number of instances left to classify is below a certain threshold, using a chi-squared test to test if the distribution of instances are independent of the remaining features and stopping accordingly, stopping a node if there is no improvement in any sort of impurity measure (ex. Gini/information gain).

**Post-pruning:**
In post-pruning, our goal to to simply create a complete tree (using any algorithm) and then prune nodes off of the tree in a bottom-up fashion. To do this, as with pre-pruning, you have some options. You can use error estimation (optimistic/pessimistic) of a given node to determine whether to replace the sub tree by a leaf node, or even randomly prune some deep branches and then evaluate the performance on a testing set for generalization error improvement or utilize the MDL principle to compare separate tree variations.

## 2.5 Is the Gini gain (Gini of the parent subtracted by the Gini of the split) always positive? What about entropy's gain? What if you use classification error? Prove or provide counter-examples.

No, Gini gain is not always positive. While it is true the Gini gain is always non-negative (a much more difficult proof), it is possible to achieve a Gini gain of 0 (which is neither positive or negative). We can see this in the case that the two children nodes do not split the class-labels any more than the parent node does.

For example, in the case of a binary classification, say the parent node has 6 instances with the positive class label and 6 instances with the negative class label.

$Gini_{parent} = 1 - ((\frac{6}{12})^2 + (\frac{6}{12})^2)$ = 1 - (0.25+0.25) = 1 - 0.5 = 0.5.

Now, let's say that the two children nodes are split such that each has 3 instances with positive labels and 3 instances with negative labels each. The corresponding Gini of each child node would be equal to that of the parent (given the same probabilities occur in each, i.e. 6/12 = 3/6).

$Gini_{child1} = Gini_{child2} = 1 - ((\frac{3}{6})^2 + (\frac{3}{6})^2)$ = 1 - (0.25+0.25) = 1 - 0.5 = 0.5

and $Gini_{split} = \frac{6}{12} * Gini_{child1} + \frac{6}{12} Gini_{child2} = 0.5 * 0.5 + 0.5 * 0.5 = 0.5$

S.T. $Gini_{gain} = Gini_{parent} - Gini_{split}$ = 0.5 - 0.5 = 0. □

Similarly for entropy's gain, the prior example shows us that we can have an information gain of 0:

$E_{parent} = -\frac{6}{12} * log_2(\frac{6}{12}) - \frac{6}{12} * log_2(\frac{6}{12}) = 1$

$E_{child1} = E_{child2} = -\frac{3}{6} * log_2(\frac{3}{6}) - \frac{3}{6} * log_2(\frac{3}{6}) = 1$

S.T. Gain = $E_{parent} - (\frac{6}{12} * E_{child1} + \frac{6}{12} * E_{child2}) = 1 - (0.5 + 0.5) = 0$

Finally, classification error also can be seen using the same example.

Given that Error = 1 - max[P(i|t)]:

Err$_{parent}$ = 1 - $\frac{6}{12}$ = $\frac{6}{12}$ = 0.5

Err$_{child1}$ = 1 - $\frac{3}{6}$ = $\frac{3}{6}$ = 0.5

Err$_{child2}$ = 1 - $\frac{3}{6}$ = $\frac{3}{6}$ = 0.5

Taking the proportion of the child nodes in comparison to the parent node, we have:

Err$_{children}$ = $\frac{6}{12}$*Err$_{child1}$ + $\frac{6}{12}$*Err$_{child2}$ = 0.25 + 0.25 = 0.5, such that

Classification Gain = Err$_{parent}$ - Err$_{children}$ = 0.5 - 0.5 = 0 gain in classification error.
This is trivial to see, given that we are still misclassifying the same number of instances between 2 nodes as we

were when using only 1 node.

However, while all 3 of these gain computations are not ALWAYS positive, they are always non-negative (i.e $\geq 0$). This is essentially because although the Gini or entropy of a child node can be greater than that of the parent node, this has an adverse effect on the other child node, causing the net summation of the two in conjunction with the proportion of the children with respect to the parent to be in the worst case equal to the parent. This results in a worst case gain of 0 for both impurity measures (a formal proof would be much more involved). Misclassification is a bit easier to reason about though, given that it is impossible to misclassify more instances in a child node than the parent node. As a result, in the worst case you are simply able to misclassify the same amount (distributed between the two child nodes), such as the example above. □

## 3 Naive Bayes Classifier

Let n = the number of training instances, d = number of features (d for dimensions) and c = number of class labels.

### 3.1 What is the time complexity for learning a Naive Bayes Classifier?

O(nd)

Explanation:
In order to train a Naive Bayes Classifier, all we need to do is get the frequency of each class label for all the features in the data. This can be done in O(nd)-time, given that for each new feature value/class label pair we can just start a new count (or add 1 to any pairs already established) and then divide by the number of the respective class label (which can also be used for the required marginal probability).

### 3.2 What is the time complexity for classifying using the Naive Bayes Classifier?

O(cd)

Explanation:
To classify an instance given our Naive Bayes Classifier, all we need to do is calculate the corresponding product of the feature values for each potential class label (and the marginal probability, but this does not change the time-complexity) and then choose the maximum value from all calculations. With this, we are operating under the assumption of a constant-time lookup for each attribute value/class label pairing.