* Q8.2 – Why is *tree pruning* useful in decision tree induction? What is the drawback of using a separate set of tuples to evaluate pruning?

Tree pruning is useful in decision tree induction, because it looks to correct the issue of noise and outliers throughout the branches of the tree. These issues may be a result of the model overfitting the training set of data and creating unnecessary branches for future data. Pruning aims to remove these excess branches and noise by statistical means, which results in more concise trees that are more efficient for processing.

It is good to have a separate set of data available to evaluate the results of pruning; however, this may also introduce some problems as well. The user must ensure the second set of tuples are truly representative of the population and not impacted by skewness or additional noise prior to using it for evaluation. This may lead to a lower than expected accuracy value if the data suffers from these issues.

* Q8.3 – Given a decision tree, you have the option of (a) *converting* the decision tree to rules and then pruning the resulting rules or (b) *pruning*the decision tree and then converting the pruned tree to rules. What advantage does (a) have over (b)?

With method (a), we can estimate the accuracy of all rules and/or conditions and determine if it improves the accuracy of the model. If it does not, then it can be pruned. These methods can include class-based ordering scheme to review the accuracy. On the other hand, method (b) will result in entire subtrees and resulting rules being removed/consolidated before the accuracy of rules can be reviewed against each other to see which should be pruned.

* Q8.4 – It is important to calculate the worst-case computational complexity of the decision tree algorithm, given data set, D, the number of attributes, n, and the number of training tuples, |D|, show that the computational cost of growing a tree is at most n x |D| x log(|D|).

The worst-case scenario would occur when would occur when all available attributes have to be used to classify the tuples and there is only one tuple classified at each level of the decision tree. Therefore, each level would require the total number of attributes, n, to be multiplied by the number of tuples, |D|, on each level. Next, this would need to be repeated for every level, so we would multiple by the total number levels that would be in the decision tree, log(|D|). As a result, the worst-case scenario for computational complexity would be n x |D| x log(|D|).

* Q8.5 – Given a 5-GB data set with 50 attributes (each containing 100 distinct values) and 512 MB of main memory in your laptop, outline an efficient method that constructs decision trees in such large data sets. Justify you answer by rough calculation of your main memory usage.

For large data sets, a user could turn to the RainForest tree induction method. This could be an appropriate method for this problem, because it adapts to the amount of main memory available. It does this by creating AVC-sets for each attribute at each node, which gives the class label count for each attribute at the node The AVC-group is then a collection of all the AVC-sets at that particular node. As a result, the size at each node would be 100 x 50 x C, where C is the number of classes present in the data set. This should fit into the main memory; however, there are further methods available to adjust this approach where the AVC-groups would not fit into main memory.

* Q8.9 – Design an efficient method that performs effective naïve Bayesian classification over an *infinite* data stream (i.e., you can scan the data stream only once). If we wanted to discover the *evolution* of such classification schemes (e.g., comparing the classification scheme at this moment with earlier schemes such as one from a week ago), what modified design would you suggest?

Since this is an infinite data stream that can only be scanned once, the most efficient method would be to create a table that collects/summarizes the count of all tuples based on the corresponding attribute and class values. This can be continuously updated to reflect the new count as more data is scanned. At any time, the desired classifications can be determined via probability calculations and naïve Bayesian classification by maximizing P(X|Ci)P(Ci).

One method to discover the *evolution* of classification schemes, would be to keep tables historical data on a running basis. For example, if we wanted to compare the data that was collected a week ago, then it would be wise to have a table that stores summarized counts for the past seven days. As another day of data is collected, then the oldest set of data in the table would be replaced with the new set of data for the day. This can be adapted for numerous timeframes (i.e. hourly, daily, weekly, monthly, yearly) depending on available memory and storage space.

