Adam Gross, Josh Palmer

February 14, 2017

Assignment 3 Write Up

Our dataset contains 100 instances with the following attributes {and their respective domains of values}:

Colorfulness	{0, 1, 2}
Color Diversity	{0, 1, 2}
Brightness	{0, 1, 2}
Human Present	{0, 1}
Animal Present	{0, 1}
Day, Night, Unknown	{0, 1, 2}
Action	{0, 1}
Seriousness/Intensity	{0, 1, 2}
Paint/Pencil/Other	{0, 1, 2}
URL	{https://www.artsy.net/artwork/[a-z\-]*}

We collected 100 samples – 33 surrealist paintings, 33 impressionist paintings, and 34 stock photo images. We used artsy.net to find high quality photos of famous paintings, and google for the stock photos. We manually evaluated the attributes for each photo, thus performing at least 1000 operations with our brains.

There are two relevant files: id3.py (the driver) and node.py (provides the two classes that form the tree structure). The driver reads in the Tennis and Art datasets. For both sets, it builds a decision tree iteratively, one node at a time, recording the errors on the training and testing sets as well as the monotonically increasing number of non-leaf nodes. For the Art dataset, it does 20 different iterations of the ID3, where it builds the tree and then prunes it iteratively (each iteration generates a new validation set randomly). As for node.py, there is the basic Tree structure with which the user interfaces. It possesses all the metadata (e.g. whether to perform cross-validation or not) and further sets up the problem by identifying the attributes and all their possible values. It maintains pointers to all of its leaves to make its display method easier (draws all possible paths to leaves) as well as candidates for possible pruning (i.e. those nodes whose children are all leaves). It also stores a FIFO queue for node exploration/expansion. Expansion ends when all leaves are pure or there are no more attributes to split on. Lastly, it has a pointer to the root node through which the evaluate function is directed in order to determine the performance of the tree on a particular dataset. As you could guess, the other structure is the Node class. The node contains a lot of information. A node is represented as a child with a particular value belonging to the attribute on which its parent was split on. (It maintains a pointer to its parent.) Non-leaf nodes are also represented by the attribute on which it itself is split. The node is responsible for expanding, if possible, following the information gain heuristic for choosing attributes. If it is a leaf node, then evaluation will return the plurality of instances that fall under it. If not, evaluation will then be passed on to the appropriately-valued child. It maintains the path of how to get from the root to the node. Every node has a pointer back to the singular Tree object that allows each to update globally relevant information (e.g. list of leaf nodes and pruning candidates).

When we ran our code on the PlayTennis example dataset, we unsurprisingly suffered from a small dataset. We didn't have enough instances to make a meaningful chart, but we still saw the general behavior patterns of ID3 that we've discussed in class. For example, in our paths.txt file that tracked the decision tree traversal, we see that we were able to short circuit evaluate after two attributes maximum. When running the code on our own dataset, we were able to see the pattern of the testing set error increase as iterations increased. Clearly this was caused by overfitting.

ID3 also performed pruning. When it tree.prune() (which returns 0 if node was removed and -1 if no node should be removed). This then iterated through all potential pruning candidates. This list is dynamically updated by the nodes to ensure that a node is a candidate iff every one of its children is a leaf node; this is done so that pruning starts from the bottom and works upwards. It temporarily prunes each candidate, evaluating the performance of the validation set on this temporary tree. It "unprunes" the candidate and moves to the next one. It then chooses the node which performs the best, removing any from the list that don't improve performance. If no node improves performance, then it returns -1 signaling the user to stop pruning. Note: each successful pruning reduces the number of non-leaf nodes by 1.

In running the pruning on our own dataset, we ran 20 validation iterations. Only a few actually yielded any pruning (i.e. most only had candidates that didn't improve performance). For example most of our runs did not prune, but run 18 had two successful prunes. In this run our error was 0.2 across the board whereas in many others it fluctuated between 0.2 and 0.3.