CS 565 Term Project: Decision Tree for Classification

(due on the class on 11/30/16)

The term project will let you be familiar with the implementation of decision tree (a popular classifier) and the procedure of evaluating the performance a classifier. In this project, the decision tree is a binary tree, i.e., each node has at most two children. For simplicity, you can assume that all attributes in a given dataset are binary.

Specifically, the term project contains two parts. In the first part, you will first need to construct a randomized decision tree. The construction procedure is a simplified implementation of the tree growth algorithm. Then you will evaluate the classification performance of the randomized decision tree by 2-fold cross validation. In the second part, you will construct a gain based decision tree by modifying two subroutines in the first part.

As well as the first project, you can use Weka library in your Java implementation. Especially, you can apply weka.core package for the necessary work on data preprocessing. Weka provides implementations of several different versions of decision trees. These implementations can be used as references to help you finish the first part of the project. But it is important to note that our implementation is different from the implementations of Weka decision trees.

1 (150 pts) Part I: Randomized decision tree

1.1 Construction of a decision tree

We use the recursive treeGrowth algorithm to construct a decision tree as follows:

The treeGrowth algorithm contains 4 important subroutines:

- *createNode()*. Simply create a node. (The node may be labeled as a leaf or an internal node, depending on the stopping condition.)
- *majorityVote(E)*. Decide a class label based on the majority of labels of instances in E. For example, assume that we have a small dataset E of 3 instances as shown in the table below. Then the majority of the label of the dataset E is **yes**.

instanceId	carType	familySize	label
1	family	5	Yes
2	sport	2	Yes
3	luxury	4	No

- *stoppingCond(E,F)*. Compute the ratio of majority label. If the ratio is larger than **0.5**, then return *true*. Otherwise, *false*. For the above example of 3-instance dataset, the ratio is $2/3 > 0.5$, and thus the method will return *true*.
- *findBestSplit()*. Randomly select an attribute as the best attribute. Assume the attribute set F has 4 attributes, excluding the label. The method randomly guesses a number i ranging from 1 to 4, and selects the i-th attribute as the best attribute.

Algorithm 1 Tree Growth Algorithm

Input: E - training records
F - attribute set
Output: root (or leaf)
1. if stoppingCond(E,F) = true, then
2. leaf = createNode();
3. leaf.label = majorityVote(E);
4. return leaf;
5. else
6. root = createNode();
7. root.testCond = findBestSplit(E, F);
8. let $V = \{v | v$ is possible outcome of root.testCond $\}$
9. for each $v \in V$ do
10. $E_v = \{e |$root.testCond$(e) = v$ and $e \in E\}$
11. child = treeGrowth(E_v, F);
12. add child as descendent of root and label the edge as v;
13. endfor
14. endif
15. return root;

1.2 Classification and evaluation

Given a test instance, you will run a classification algorithm on the constructed decision. The classification algorithm is also the recursive algorithm. The basic steps of the algorithm is the following:

Algorithm 2 classification

Input: T - the root of a decision tree
i - instance
Output: label - classified label of the instance
1. if T is a leaf, then
2. return T.label;
5. else
6. currentAttribute = T.testCond
7. find a child such that i's currentAttribute == T.leftName or T.rightName
8. return classification(T.child, i)

To evaluate the overall performance of a decision tree, you will apply 2-fold cross validation to estimate the classification accuracy of a decision tree.

2 (150 pts) Part II: Gain based decision tree

In Part II, you are going to construct a gain based decision tree by modifying the methods *stopping-Cond(E,F)* and findBestSplit(). In this project, the impurity and gain is measured in terms of *entropy*.

- *stoppingCond(E,F)*. Compute the impurity (entropy) of E, based on the label. If the impurity is less than a *threshold*, then return *true*. Otherwise, *false*. The threshold is a class variable that can be set during the creation of an object of the decision tree class.
- *findBestSplit()*. Find the best attribute by maximizing the gain, equivalently, by minimizing the overall impurity of children (each child is a possible value for a specific attribute).

As mentioned above, you can assume that all attributes in a given dataset are binary. **You can get up to 50 bonus points if your algorithm can handle continuous attributes and multi-value nominal attributes.**

Similar to Part I, you then use 2-fold cross validation to evaluate the classification accuracy of the decision tree. Do experiments with different thresholds used in stoppingCond().

This project is a team project, and you can form a group of at most two.

On submission: Submit the source code and readme to Blackboard. The source code should be well-documented, i.e., having good readability. The readme file should summarize: i) tasks that you completed, and ii) tasks that you did not complete.