CS412 Introduction to Data Mining – Homework 4

Name: Te-Yao Lee
NetID: teyaoyl2
UID: 650086527

Algorithm Design

- Decision Tree

Step1:

The main function is the createTree() function. The function will use the dictionary data type to store the tree. The structure of the dictionary will be: {splitting attribute name: {unique value of the splitting attribute1: label or sub-tree}}

In this function, it will first check whether check two stop conditions. The first condition is, if all the label of tuples is the same, it will return the value of the label. The second condition is, if there is no other attributes (columns beside the label column), it will call the voteResult() function which will return the majority of the label. If the candidate data sets do not meet those two conditions, it will call the selectAttByGini() function which will return the index of the splitting attribute(the attribute with the lowest Gini value) of this tree node.

Step2:

In the selectAttByGini() function, it will first call the createRainForestTable() function which will scan the data base once and return the rain forest table to selectAttByGini(). The selectAttByGini() will calculate the Gini value of each attribute and return the index of the attribute with the lowest Gini back to the createTree().

After receiving the index of splitting from the selectAttByGini(), createTree() function will store the attribute name into the dictionary of decision tree, and remove the attribute name from the list that records the name of all attribute name.

Step3:

The createTree() function will identify unique attribute value of the splitting attribute. The createTree() will create a sub-tree for each unique attribute value of the splitting attribute by the next two procedures. The first procedures is creating sub-candidate by calling the createTupleList() function and createTupleList() will create a list of candidate tuples of that specific sub-tree. After the list for sub-tree is created, createTree() will call itself recursively until it reach the stop condition in step 1.

- Random Forest

It will start with the createForest() function. The function will create a list that store all the tree create by the createTree() function. Secondly, the Random Forest is just adding a function called randomAttName() before selectAttByGini(). The randomAttName() will randomly select a number of attributes for selectAttByGini() to select the best one to be splitting. For the number of trees, I use trial and error to decide, I found out that after 60 trees the accuracy rate stop growing. Therefore, I use 60 trees as my default. Also, for the random select of attribute, if the total attribute number is more than 5, it will select 20% of them. If the total attribute number is between 3~5, it will select 50% of them. If the number is below 3, it will use all of the attributes.

Result Evaluation

- Decision Tree

The overall accuracy varies greatly, but the overall trend is that the bigger the training set it is, the higher the accuracy rate it is. For the balance.scale.test data set, class number one is all wrong. I tried to reduce the tree depth, but it does not work very well. Maybe using binary split would have a better result. However, for class 2, the Indicators of each class are actually higher. For the led.test.new data set, the overall accuracy is 85%, but the indicator of class 1 is only 78%. For the nursery.data.test, it has the highest accuracy of 97.8%, but, for class 2 the overall indicator is only 70%. Also, the training data has a problem that it contains label of 5 which does not exist in the testing data, and it lead to a miss classification of some tuple. For the last data set poker.test, the accuracy rate is also not very high, and the classification on class 2 is not accurate enough as well.

Data set	Accuracy	Class	Indicator of each class			
			Sensitivity	Recall	f1-Score	Precision
balance.scale.test	0.613	1	0.00	0.00	0.00	0.00
		2	0.784314	0.784314	0.744186	0.707965
led.test.new	0.852	1	0.780627	0.780627	0.765363	0.750685
		2	0.88378	0.88378	0.891753	0.89987
nursery.data.test	0.978	1	0.969298	0.969298	0.964163	0.959082
		2	0.7	0.7	0.708171	0.716535
		3	0.980469	0.980469	0.985925	0.991442
		4	1.00	1.00	1.00	1.00
		5	0.00	0.00	0.00	0.00
poker.test	0.636	1	0.801743	0.801743	0.748728	0.70229
		2	0.287671	0.287671	0.337802	0.409091

- Random Forest

Compared to the Decision Tree, the Random Forest has better performance on each aspect of the classification process. It solves the issue of label value at data set nursery.data.test, and has an overall increasing at the Sensitivity, Recall, f1-Score and Precision of each class data, such as the accuracy of both classes at poker.test have increase dramatically. However, we still have some issue with the class 1 of balance.scale.test. Maybe using binary classification tree can solve this problem.

Data set	Accuracy	Class	Indicator of each class			
			Sensitivity	Recall	f1-Score	Precision
balance.scale.test	0.688889	1	0	0	0	0
		2	0.833333	0.833333	0.755556	0.691057
led.test.new	0.862434	1	0.766382	0.766382	0.775216	0.784257
		2	0.905492	0.905492	0.900889	0.896334
nursery.data.test	0.978015	1	0.979323	0.979323	0.966904	0.954795
		2	0.684615	0.684615	0.751055	0.831776
		3	0.978516	0.978516	0.984283	0.990119
		4	1	1	1	1
poker.test	0.679941	1	0.960784	0.960784	0.802548	0.689063
		2	0.091324	0.091324	0.155642	0.526316