// CS570 May 8, 2013

// Heyan Huang Project 4 Decision Trees

Decision Prediction for Robot Decisions

**Abstract**

In this project, a decision tree algorithm is developed using the ID3 information gain algorithm, and it was implemented using c++ language. The algorithm selects decision split attributes based on information gain, and selects the variable with highest information gain from the available variable attribute open list. The algorithm will repeat this step until either run out of attributes, or run out of training data set. And later on, based on the tree rules we have built already using training data, we will evaluate the performance of the decision tree model by test the model on the testing data set. The model performance will be evaluated by misclassification/prediction rate. And based on the available 100 observations, I have built a pretty good decision medel with mis-prediction rate is almost 0.

**Data Structure**

Data structure is very important for this project. I need to figure the data structure out before I can develop any codes for this project. The most fundamental tree node is easy, just includes the attribute, arrived\_value, isLeaf lable and node pointer vector stores the node pointers from parent to children (for this project since every node has only two children, it should be much easier to simply use left and right pointers, but anyway I learned how to use vectors). The input data was read into a 2D array, and there are several vectors and vector of vectors to store the attribute strings for all available and remaining exploratory variables, the attribute value strings for each attribute, and total training and remaining training observations. The whole C++ code for this project is attached in the back for convenience of checking.

**Learning Algorithm**

For this project implementation, B(q) is defined as the entropy of a Boolean random variable that is true with probability q:

B(q) = -( q\*log2(q) + (1-q)\*log2(q) );

A randomly chosen example for the training set has the kth value of rth attribute with probability (pk+nk)/(p+n). So the expected entropy remaining after testing attribute A is :

Remainder(A) = sum(k=1 to d)((pk+nk)/(p+n))B(pk/(pk+nk));

And information gain from the attribute test on A is the expected reduction in entropy:

Gain(A) = B( p/(p+n) ) - Remainder(A);

**Pseudo code for the algorithm:**

function Decision-Tree-Learning (examples, attributes) returns a tree

if examples is empty then return default

else if all the examples have the same classification then return the classification

else if attribute is empty then return major decision

else

for each remaining attribute

calculate information gain

create a new node using the best attribute

separate the examples based on the best attribute

subtree nodes = Decision-Tree-Learning (examples, remaining attributes)

return tree

**Result**

According to the project instructions, I have applied the 2, 5, 10, 20 and 50 observations as the training dataset to build decision trees respectively, and the produced tree were tested using the remaining observations from original 100 observation data set. All the models are pretty good and really look alike. And the models are performing better and better as the training data set increasing the observations, which means the decision tree models are well-trained and performs great job for predictions.

|  |  |
| --- | --- |
| train\_obs\_cnt | mis\_predict\_test |
| 2 | 60 |
| 5 | 60 |
| 9 | 20 |
| 10 | 26 |
| 13 | 48 |
| 15 | 19 |
| 20 | 19 |
| 25 | 21 |
| 30 | 10 |
| 35 | 10 |
| 40 | 10 |
| 43 | 10 |
| 44 | 3 |
| 50 | 3 |

The final tree model that predicts best is listed as below:

the decision tree is:

SafSit

1

SafCriDec

1

FamSit

1

AskdBef

0

No cnt:3

1

Confident

1

Yes cnt:3

0

No cnt:2

0

Yes cnt:4

0

AskdBef

0

No cnt:9

1

FamSit

1

Confident

1

Yes cnt:1

0

No cnt:1

0

No cnt:3

0

FamSit

1

Confident

1

No cnt:1

0

Yes cnt:3

0

Yes cnt:14

tree\_size: 21

the total observation count is: 44.

Total number of misclasification for TRAINING data is: 0.

Total number of misclasification for TESTING data is: 3.

**Discussion**

As we can see from the graph above, between trains observation of 9 and 10, there was a dramatic reduce of misclassification rate for test data, and as I have detected, the observation number 10 created a bunch of new tree nodes and completely changed the tree structure, and which actually is not as good as predicting using the 9 observation tree before. These specific two trees are pasted below for reviewing convenience. Since these models are not mature ones yet, the trees are just presented as the indication of tree structure changes.

the training = 9, testing = 91, training data built decision tree is print out as followed:

the decision tree is:

SafSit

1

No cnt:7

0

Yes cnt:2

tree\_size: 3

the total observation count is: 9.

Total number of misclasification for TRAINING data is: 0.

Total number of misclasification for TESTING data is: 20.

the training = 10, testing = 90, training data built decision tree is print out as followed:

the decision tree is:

SafSit

1

LongDel

0

AskdBef

0

No cnt:2

1

Yes cnt:1

1

No cnt:5

0

Yes cnt:2

tree\_size: 7

the total observation count is: 10.

Total number of misclasification for TRAINING data is: 0.

Total number of misclasification for TESTING data is: 26.

Our data set has 8 variables and has only 100 observations. Theoretically we would expect at least 28= 256 observations to be an even representative training data set. Comparatively, our training dataset doesn’t have enough observation, and may produce bias.

Because I used the information gain algorithm to build the tree, the built tree will largely screwed by the number of observations in the node class and the spilt attribute will also be dependent on the number of observations fall into arrived\_value node. So decision tree is very sensitive to unrepresentative data.

Besides, since the tree depends on split attributes as well as attribute-values to build node, it is also very sensitive to missing values, and cannot perform predictions on those observations unless we take good care of the data cleaning and missing value modification.

For this project, I separated training and test data simply by using the first N numbers of observation as training data, and all the rest obs. as the test data, and misclassification rate as the fitness function, which is a just-so-so method to get project done. To extract representative training data set, a better approach would be m-fold training data, and using cross-validation will sure perform much better.