Author: Nan Chen
Date: Nov. 9, 2019

The C++ programs (id3.cpp) are developed to creates an ID3 decision tree to classify a set of input training data and then reports the classification performance on a separate set of input testing data. The branch factor of a ID tree is set to 2, in other words , the shape of a ID3 tree is very similar to the shape of a binary search tree , each tree node in ID3 tree contains : a set of 2D data (includes all attributes and class label), a left child pointer and a right child pointers, a double variable splitVal contains an average value of two different adjacent number in the same sorted attribute column, an integer which stores the specific attribute column number the ID3 algorithm is splitting the data set on.

The reason setting branch factor to 2 in this data ming project is that, in the training and testing the data set, all data are float and continuous in each attribute. The average operation guaranteed separate the data set into 2 parts.

```
struct Node{
  vector<vector<double> > dataSet; // data set contains all attributes and class labels
  vector<double> label; // label for data set
Node*  leftNodePtr; // left Child of a ID3 tree node
  Node*  rightNodePtr;// right Child of a ID3 tree node
  double splitVal;  // float splitting number initialized to -1.0
  int splitIndex;   // indicates the positive of the attributes , initialized to -1
  Node()  // default constructor
  {
    leftNodePtr = NULL;
    rightNodePtr = NULL;
    splitVal = -1.0;
    splitIndex = -1;
  }};
```

ID3 algorithm consists of two parts: the 1st part of algorithm is to build the ID3 tree by training the model using the given data: the algorithm starts building the tree from the root node, the root node contains the entire dataset to start with. Then the ID3 algorithm begins to look for the best binary split point by looking at possible binary split point, and find the maximum information gain (I.G.) by the formula (total information subtracted by expectation of an attribute)

$$I.G.(Attr_i) = I(C_1,C_2 \ldots C_n) - E(Attr_i)$$

$I(C_1,C_2 \ldots C_n)$ is the total information , which can be found by the formula

$$I(C_1,C_2 \ldots C_n) = P(C_1)*\text{-log2}(P(C_1)) + P(C_2)*\text{-log2}(P(C_2)) + \ldots + P(C_n)*\text{-log2}(P(C_n))$$

Interestingly the total information $I(C_1, C_2 \ldots C_n)$ can be greater or equal to 1 in some cases, for example: in Iris data set, we have 3 different class labels (n=3): 0 , 1 , 2

$$I(0,1,2) = P(0) *\text{-log2}(P(0)) + P(1)*\text{-log2}(P(1)) + P(2)*\text{-log2}(P(2)) = 1.58$$

The expectation of an attribute can be find by the formula

$$E(Attr_i) = P(x<=sp)*\{P(C_1|\ x<=sp)*\text{-log2}(P(C_1)) +\ldots+ P(C_n|\ x<=sp)*\text{-log2}(P(C_n))\ \}$$
$$+ P(x > sp)*\{P(C_1|\ x>sp)*\text{-log2}(P(C_1)) +\ldots+ P(C_n|\ x>sp)*\text{-log2}(P(C_n))\ \}$$

As we can see the formula above, we just need to consider two parts in each binary split, and we need to consider all possible binary split positions in our data set using the formula below , and then find the best split with highest I.G.($Attr_i$).

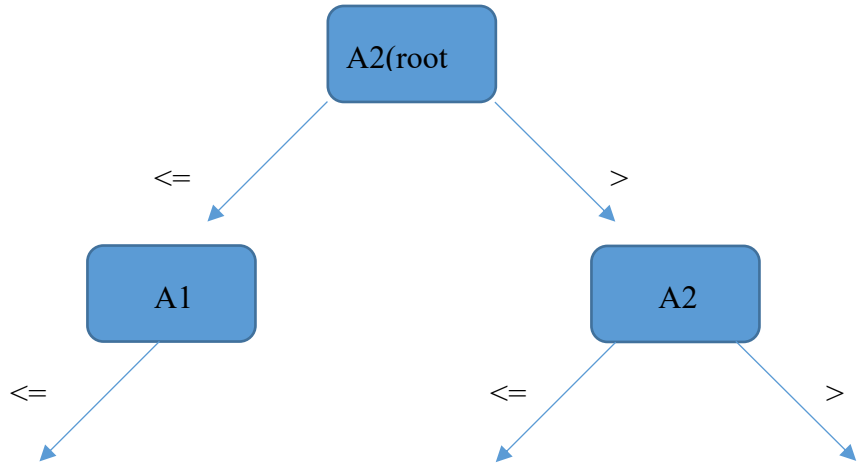
$$I.G.(Attr_i) = I\ (C_1,C_2\ldots C_n) - E(Attr_i)$$

In case there are two or more split points with the same I.G.($Attr_i$) , tires is broken such that the split best preserving the prior probability. The degree of the prior probability preservation can be expressed in a formula below:

$$Distance = sqrt\{[P(x<=sp) - P(C_1|\ x<=sp)]^2 +\ldots.+ [P(x<=sp) - P(C_n|\ x<=sp)]^2 +$$
$$[P(x>sp) - P(C_1|\ x>sp)]^2 +\ldots.+ [P(x >sp) - P(C_n|\ x > sp)]^2\ \}$$

In rare cases that, there are two or more split points have the same prior probability preservation degree (distance), ID3 algorithm picks the 1st split points on the left (and up)

In this way, once the split position is determined, the original dataset is split into 2 smaller subsets from the root:



During tree-building (training) phase, ID3 algorithm repeats the same splitting process until the data set cannot be split into 2 parts. (Data set on terminal have the same class label)

After tree-building (training) phase, the 2nd part of ID algorithm is used to classify unknown data: testing data. For a given testing data, we need to traverse the ID3 binary tree by comparing its attribute value with the tree node until we reach the terminal node with a particular class label. Then we compare this class label with the class label given in the testing data set (ground truth).

If two class labels match with each other, we add one to a counter. If two class labels do NOT match with each other, we do nothing.

In building ID3Tree by the training data set (set size =100), a recursive ID3BinarySplit function is implemented to split the data set into 2 parts:

```
Node* ID3BinarySplit(vector<vector<double> > data){
	Node *newnode = new Node;
	newnode->dataSet = data;
	Node *current = newnode;
	newnode->FindSlit();
	if (newnode->terminalBoolState == false) // not termial node
	{
		newnode->leftNodePtr = ID3BinarySplit(newnode->dataSetL);
		newnode->rightNodePtr = ID3BinarySplit(newnode->dataSetR);
	}
	else   return current; // base-case: terminal node, done with splitting
	return current;
}
```

Once the ID3 is built by training data set (set size = 100), a recursive traverseID3Tree function is implemented to traverse the ID3 tree to classify the test data set (set size = 50)

```
void traverseID3Tree(vector<double>  dataSet, Node *NodePtr)
{
	int TreeNodeLabel;
	int attributeNum;
	double temp;

	if (NodePtr->terminalBoolState == true) // this node is terminal node
	{
		TreeNodeLabel = NodePtr->label;
		if (TreeNodeLabel == dataSet[dataSet.size() - 1]) // two labels match
		{
			correctlyClassifiedDataCounter++;  // define to Global ?
			return;
		}
		else
		{
			return; //continue;
		}
	} //if

	else  // this node is an internal node
	{
		attributeNum = NodePtr->rowNum;		// retrieve attributeNum
```

```
                temp = dataSet[attributeNum];
                if (temp <= NodePtr->splitVal)          //recursive call myself with LC:
NodePtr->leftNodePtr
                        traverseID3Tree(dataSet, NodePtr->leftNodePtr);
                else // temp > head->splitVal ,recursive call myself with RC:
NodePtr->rightNodePtr
                        traverseID3Tree(dataSet, NodePtr->rightNodePtr);
        } //else

        return;
}  // traverseID3Tree ()
```

In this way, the execution of id3.cpp outputs the number of testing examples classified correctly by the decision tree.

The classification correction rate is very high for both Iris (Training set size =100, testing set size = 50), 47 out of 50 testing data set are classified correctly, classification correction rate = 94%

and the cancer data set (Training set size =100, testing set size = 50), 48 out of 50 testing data set are classified correctly, classification correction rate = 96%. Both Iris and Cancer training set (100) and setting set (50) is randomly chosen by "shuf" linux command which pre-exited in ranger/system64.

I have utilized my id3 program to perform cross-validation analysis (repeated random sub-sampling) on the iris and cancer data sets (see below). The data split can be done by using split.bash on instructor's website. The usage of split.bash is shown below:

$cat iris-data | ./split.bash ./id3 4

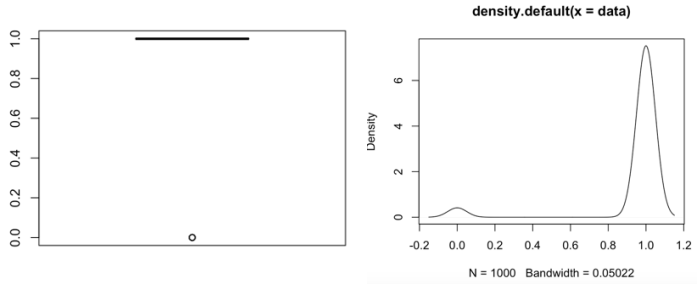
However, ./split.bash is dependent on the existence of shuf command exclusively on ranger/system64.

I have used bash tools to create 1000 training/testing sets of size (n-1)/1 and (n-10)/10 and performed n-fold cross validation:
for ((x=0;x<1000;x++)); do cat iris-data.txt | ./split.bash 10 ./id3 4 > Iris_result_10.txt; done
for ((x=0;x<1000;x++)); do cat cancer-data.txt | ./split.bash 1 ./id3 9 > Cancer_result_1.txt; done
for ((x=0;x<1000;x++)); do cat cancer-data.txt | ./split.bash 10 ./id3 9 > Cancer_result_10.txt; done

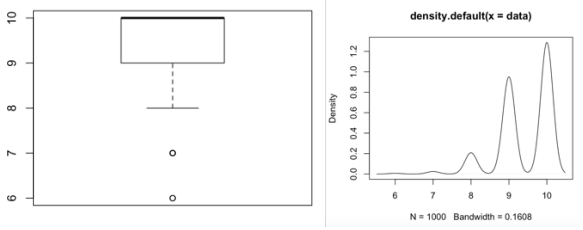
I have calculated the mean and standard deviation in R of the percentage of testing examples correctly classified by my ID3 decision trees:
For Iris data, one random data is pick out as the setting data, n-1 is the training set
for ((x=0;x<1000;x++)); do cat data.txt | ./split.bash 1 ./id3 4 > result.txt; done

This process repeats 1000 times, then mean and median operations are performed in R using the commends below:

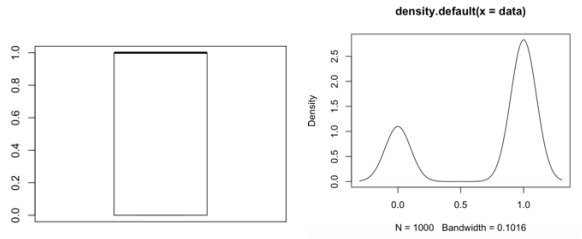
```
setwd ("/Users/davidchen/Desktop/p3/Nov7")
data <- scan("Iris_result_1.txt")
mean(data/1)
[1] 0.948
median(data/1)
[1] 1
sd(data)
[1] 0.2221381
boxplot(data)
plot(density(data))
```



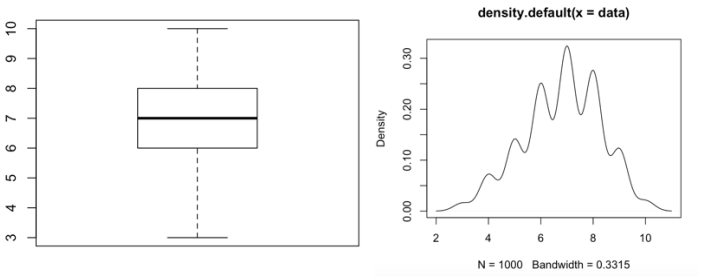

```
data <- scan("Iris_result_10.txt")
mean(data/10)
[1] 0.9406
median(data/10)
[1] 1
sd(data)
[1] 0.7111045
boxplot(data)
plot(density(data))
```



```
data <- scan("Cancer_result_1.txt")
mean(data/1)
[1] 0.72
median(data/1)
[1] 1
sd(data)
[1] 0.4492236
boxplot(data)
plot(density(data))
```




```
data <- scan("Cancer_result_10.txt")
mean(data/10)
[1] 0.6813
median(data/10)
[1] 0.7
sd(data)
[1] 1.46635
boxplot(data)
plot(density(data))
```



The mean (average) ID3 classification correction rate in 1000 runs are tabulated in the label below:

	1/ (n-1)	10/ (n-10)
Iris	94.8%	94.06%
Cancer	72%	68.13%

Interestingly, as we can see the table above: ID3 classification correction rate is lower in 10/ (n-10) case than the correction rate of 1/ (n-1) case for both Iris & Cancer data sets, because 1/ (n-1) case have more training data set than training data of 10/ (n-10) case when n is the same, so 1/

(n-1) case have more opportunity to be better trained to form a better classifier model (decision tree), therefore 1/ (n-1) cases have better correction rate than 10/ (n-10) cases.

The standard ID3 classification correction rate in 1000 runs are tabulated in the label below:

	1/ (n-1)	10/ (n-10)
Iris	0.2221381	0.7111045
Cancer	0.4492236	1.46635

Interestingly, as we can see the table above: ID3 classification standard deviation is also lower in 10/ (n-10) case than the standard deviation of 1/ (n-1) case for both Iris & Cancer data sets, because 1/ (n-1) case have more number of training data set than number of training data of 10/ (n-10) case when n is the same, so 1/ (n-1) case have more opportunity to be better trained to form a better classifier model (decision tree). Therefore 1/ (n-1) cases have better (lower)standard deviation rate than 10/ (n-10) cases, as the classified results in 1/ (n-1) cases are classified more correctly and consistently.