Project 1 – Decision Tree

CS529 Machine Learning

Jun Peng

Due Feb. 6th, 2015

1 Introduction

Decision tree learning is one of the most widely used and practical methods for inductive inference. It is a method for approximating discrete-valued functions that is robust to noisy data and capable of learning disjunctive expressions. In this project, we are required to implement a decision-tree algorithm and apply it to molecular biology, a Promoter_is region of DNA that facilitates the transcription of a particular gene. Given training data and validation data, we create the decision tree with training data, then use chi-square test to post-prune the tree, finally test the decision tree with validation data. In training part, we are required to use both accuracy (misclassification impurity) and information gain (entropy impurity) to select the best attribute to test.

2 Code Explanation

The code is written in Python. It runs under Python2. Due to the difference between Python2 and Python3, it can not be run under Python3. Program has been tested on CS machine. It requires some widely used python libraries to work with such as numpy and scipy (already installed on CS machine). The whole code can be divided into four parts, as described below.

2.1 Classes

Two classes are defined in this program, one is class Item, which is used to store the entry of training data/testing data, the other is class DTNode, which is used to build the decision tree. Class Item has two member variables: `value` and `label`. `Value` is used to record the DNA sequence (eg. cagaaacgttttattcgaacatcgatctcgtcttgtgttagaattctaacatacggt) and `label` is used to indicate if the DNA sequence is Promoter or not (eg.'+'/'-').

Class DTNode have several member variables {`isLeaf, targetIndex, descendants, label, stats`}. `isLeaf` indicates current DTNode is a leaf or not. `targetIndex` is used to record the index that we want to test at current DTNode. `descendants` is a dictionary which the key is the possible values of DNA sequence at `targetIndex` and the value is the next DTNode it should test. `label` records the most common label at current DTNode. `stats` is a dictionary which records the {label, number of training instances of label} pair, eg. {'+': 10, '-':20}.

2.2 Training

We are required to use both accuracy (misclassification impurity) and information gain (entropy impurity).

Information Gain

The criterion we use to determine which attribute should be selected is information gain. We select the attribute which give us the maximum information gain.

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i$$

$$Gain(S,A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

In the program, function `entropy` is used to calculate the entropy using the formula listed above. Function `informationGain` is used to calculate the information gain using the formula listed above.

Function `DTBuilderForID3` takes one argument, which is the collection of training items, and returns the root of the decision tree. The logic for this function is that, it first create a new DTNode, then test if all training items have the same label. If yes, assign the label of training items to node `label` and set `isLeaf` to true. Otherwise we use the criterion – informaiton gain – to select the attributes we want to test at current node. Then partition training items according to the selected attribute. Recursively construct the `descendants` with partitioned items.

Misclassification Impurity

The criterion we use to determine which attribute should be selected is accuracy. The criterion here is the misclassification error. In the calculation of gain, we use misclassification error instead of entropy. Then we select the one that gives us the maximum gain.

$$classificationError(S) \equiv 1 - \max_i \{p_i\}$$

$$Gain(S,A) \equiv classificationError(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} classificationError(S_v)$$

In the program, function `impurity` is used to calculate the misclassification error using the formula listed above. Function `impurityGain` is used to calculate the gain using the formula listed above.

Function `DTBuilderForImp` takes one argument, which is the collection of training items, returns the root of the decision tree. The logic for this function is that, it first create a new DTNode, then test if all training items have the same label. If yes, assign the label of training items to node `label` and set `isLeaf` to true. Otherwise we use the criterion – impurity gain (accuracy) – to select the attributes we want to test at current node. Then partition training items according to the selected attribute. Recursively construct the `descendants` with partitioned items.

2.3 Post-pruning

Here we use chi-square test method to post-prune the decision tree in order to overcome overfitting problem. Given confidence level, we first calculate $\alpha = 1 - confidenceLevel$. If current node is a leaf, we don't have to calculate chi-square. Otherwise we calculate the chi-square for non-leaf node. Then calculate P-value by using chi-square and degree of freedom via library function (`stats.chi2.cdf`). If P-value is greater than α, we prune all branches of current node and set the current node to be a leaf. Otherwise we don't prune it.

$$\chi^2 = \sum_{i=1}^{v} \frac{(p_i - p_i')^2}{p_i'} + \sum_{i=1}^{v} \frac{(n_i - n_i')^2}{n_i'}$$

The above formula shows how to calculate chi-square. The degree of freedom in our project is

$$\text{DOF} = \text{ number of children of current node} - 1$$

.

2.4 Validation

We are given validation data to calculate the accuracy of prediction. For each validation item, we start from the root of decision tree. If the root is a leaf, we check if the root label equals the item label. Otherwise, we test the `targetIndex` of item and go down to the corresponding descendant. If the value of tested index of item isn't in `descendants`, we return the most common label at current node as prediction result which later is used to compare with item label. We count how many predictions are consistent with the actual label. Then we can easily calculate the correction percentage.

3 Results

The result under confidence level 0.99, 0.95, 0, which is required by the project, is shown below.

Confidence Level	0.99	0.95	0
Information Gain (Entropy impurity)	85.7%	82.9%	85.7%
Accuracy (misclassification impurity)	77.1%	71.4%	68.5%

Confidence level 0 represents no post-prune at all. From the results shown above, we can see that information gain always performs better than accuracy (misclassification impurity). Post-pruning on accuracy (misclassification impurity) performs better than no pruning. It reflects that this method has overfitting problem. However it doesn't mean the higher confidence level, the higher correction percentage we can get. Suppose the confidence level is 1, the decision tree would have only one node after post-pruning and it gives bad correct percentage. So we have to choose an appropriate confidence level for each algorithm. As we can see, in information gain, the confidence level 0.95 is worse than confidence level 0.99 or 0. We may want to test different confidence level to find the best one.

The correction percentage is correlated to the training data and validation data since they determine how the tree grows. For different combination of training data and validation data, it may have different confidence level for best correction percentage. This is a parameter we can tune for getting the best performance.

From the experimental results above, we can conclude that information gain method along with confidence level 0.99 performs best. Although information gain with no prune gives us the same correction percentage. But we believe that is caused by our validation data. If we change validation data, the result would be different. In theory, post-pruning reduces the overfitting problem.

4 Conclusion

Comparing the results of accuracy (misclassification impurity) and information gain, we conclude that

(1) the performance of information gain method is usually better than accuracy (misclassification impurity).

(2) Post-pruning affects the correction percentage of our decision tree. It does extremely well on accuracy (misclassification impurity) for correction percentage. However it seems not predictable on information gain according to our results.

(3) information gain method is preferred over accuracy (misclassification impurity) for decision tree learning algorithm.