CS 678 Machine Learning
Programming Assignment #3

Information-based Learning using Decision Trees

Specification

The basic idea is to write a program that, given a collection of training data for a classification problem, generates a Decision Tree via the ID3 algorithm.

Background

Decision trees are hierarchical data structures functioning as classifier systems. They are constructed based on a set of training data for which the value of the target function is known (i.e. they are a form of Supervised Learning). ID3 is a greedy algorithm that generates shortest-path decision trees.

Resources

- Your text contains a psuedocode presentation of the ID3 algorithm (Figure 9.3).
- A tutorial describing the operation of the ID3 algorithm has been posted on the course github repo.
- The course github repo also includes a link to the UCI Machine Learning Repository, a good source of databases culled from many different domains.

Implementation

Implement the basic ID3 algorithm to create a decision tree classifier.

ID3 (*S*)
    if all examples in *S* are of the same class
        return a leaf with that class label
    else if there are no more attributes to test
        return a leaf with the most common class label
    else
        choose the attribute *a* that maximizes the Information Gain of *S*
        let attribute *a* be the decision for the current node
        add a branch from the current node for each possible value *v* of attribute *a*
        for each branch
            "sort" examples down the branches based on their value *v* of attribute *a*
            recursively call ID3(S_v) on the set of examples in each branch

To implement the algorithm, you will need:

- A measure of purity (e.g. Entropy):

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

  where S is the collection of examples, k is the number of categories, and p_i is the ratio of the cardinality of category i to the cardinality of S, as in $p_i = N_i / N$

- The formula for Information Gain:

$$\text{Gain}(S, a) = \text{Entropy}(S) - \sum_{v=\text{values}(a)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

  where *values(a)* is the set of all possible values for attribute a, and S_v is the subset of set S for which attribute a has value v.

Data Sets

Sample datasets have been posted on the course Web page. Datafile format is:

NumTargets
targetNames
NumAttributes
attributeName numValues *attributeValues* // each attribute takes multiple values
attributeName "real" // continuous-valued attribute
NumExamples
attributeValues *targetValue* // one example per line

You may assume there will be discrete (nominal) attribute values for all training data. A continuous-valued dataset is posted (Iris.data) for additional analysis. The datasets contain a "header" containing metadata – you may modify them in any way you choose.

Requirements

Submit a written report (PDF) and be prepared to present your solution to the class.

- □ Include complete documentation of your code (this can be in the pdf for ipython notebooks or uploaded to the repo separately).
- □ Describe your approach, choice of metric, any interesting problems encountered or experiments performed, special packages used, etc.
- □ Demonstrate the effectiveness of your classifier on a test set. Include a discussion/analysis of your results.

<u>Further Investigation</u>

- Extensions
  - Find/create/use a different problem domain and dataset
  - Add "*Classification* mode" to your program (i.e. input an unseen example and use the decision tree to output a prediction/classification)
  - Extract the *rule-base* (IF-THEN) from your decision tree.
- Alternate implementations
  - Experiment with alternate splitting functions
  - Experiment with weighted training data
- Structural Enhancements
  - Implement pruning
- Usability
  - Incorporate numeric (continuous-valued) training data
- Visualization
  - Create a visualization of your growing/final tree
- Ensemble Learning
  - Employ bagging (bootstrap aggregating) to implement Random Forests and investigate their performance (you could compare your implemented algorithm to preset packages that perform these).