Project 4: Learning

Jing Zhang

Collaborator: Fengxiang Lan, Jiayi Qi

1. Introduction to Algorithm

In this project we will implemented and evaluate two machine learning algorithms. One is Decision Tree Learning and the other one is Linear Classifiers.

1.1 Decision Tree Learning

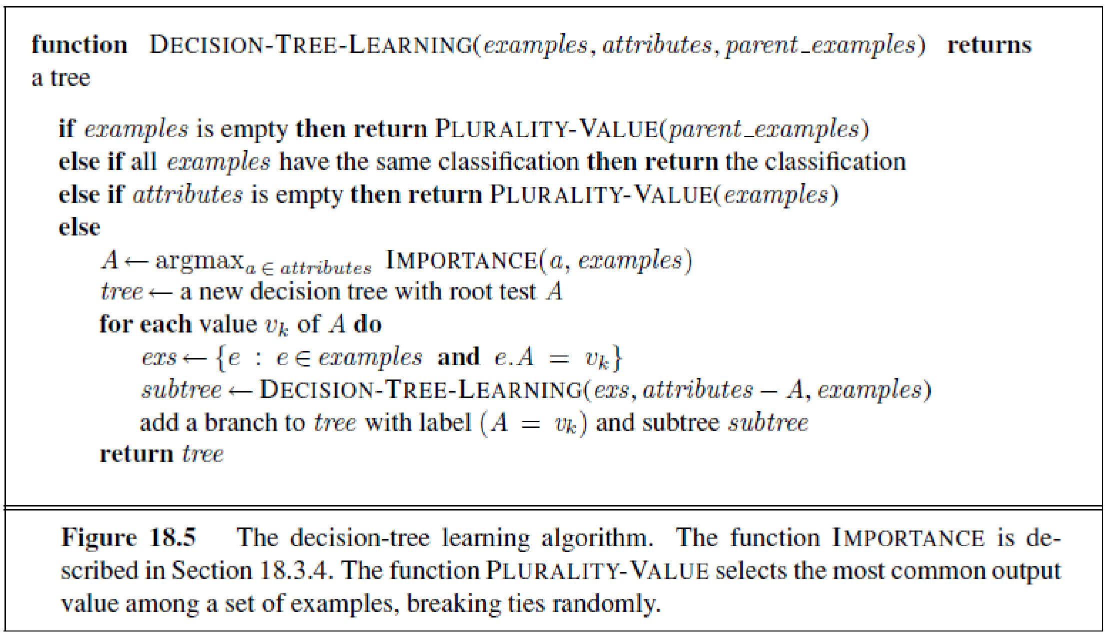
Decision Tree is one of the simplest forms of machine learning. A decision tree represents a function that takes as input a vector of attribute values and returns a single output value. In this project we will focus on problems where the inputs have discrete values, and the output has exactly two possible values.

A decision tree reaches its decision by performing a sequence of tests. Each internal node in the tree corresponds to a test of the value of one of the input attributes. And the branches from the node are labeled with the possible values of the attribute. Each leaf node in the tree specifies a value to be returned by the function. A decision tree has the power if explanation.

So how to induce a decision tree from examples? The DECISION-TREE-LEARNING algorithm is shown in the Figure 1 (AIMA Figure 18.5). There are 4 cases in the recursive process:

1) If the remaining examples all have same output, then we are done;
2) If there are examples with different outputs, we choose the best attribute (use entropy-based attribute selection method described in AIMA Section 18.3.4) to split them.
3) If no examples are left, that means no example are observed with such combination of attributes value. We choose the plurality classification as the default value to return.

4) If there are no attributes left, but examples with different values, that means we have some noise or error in the training data. We also use the plurality classification as the default value to return.



```
function DECISION-TREE-LEARNING($examples$, $attributes$, $parent\_examples$) returns a tree

  if $examples$ is empty then return PLURALITY-VALUE($parent\_examples$)
  else if all $examples$ have the same classification then return the classification
  else if $attributes$ is empty then return PLURALITY-VALUE($examples$)
  else
      $A \leftarrow \mathrm{argmax}_{a \in attributes}$ IMPORTANCE($a$, $examples$)
      $tree \leftarrow$ a new decision tree with root test $A$
      for each value $v_k$ of $A$ do
          $exs \leftarrow \{e : e \in examples$ and $e.A = v_k\}$
          $subtree \leftarrow$ DECISION-TREE-LEARNING($exs$, $attributes - A$, $examples$)
          add a branch to $tree$ with label $(A = v_k)$ and subtree $subtree$
      return $tree$
```

Figure 18.5 The decision-tree learning algorithm. The function IMPORTANCE is described in Section 18.3.4. The function PLURALITY-VALUE selects the most common output value among a set of examples, breaking ties randomly.

Figure 1 Decision Tree Learning Algorithm

1.2 Linear Classifiers

Linear functions can be used to do classification. The decision boundary is a line (or a plane or a super plane in higher dimensions) that separates the two classes. A linear decision boundary is called a linear separator. And data that can be divided by such a separator are called linearly separable.

1.2.1 Perceptron

We can write the classification hypothesis as

$$h_{\mathbf{w}}(\mathbf{x}) = 1 \text{ if } \mathbf{w} \cdot \mathbf{x} \geq 0 \text{ and } 0 \text{ otherwise.}$$

Or, we can think of h as the result of passing the linear function w*x through a threshold function:

$$h_{\mathbf{w}}(\mathbf{x}) = Threshold(\mathbf{w} \cdot \mathbf{x}) \text{ where } Threshold(z) = 1 \text{ if } z \geq 0 \text{ and } 0 \text{ otherwise.}$$

Besides the threshold function, we need a simple weight update rule that can converges to a solution (a linear separator that classifies the data perfectly if the provided data is linearly separable). For the perceptron algorithm the update rule will be

$$w_i \leftarrow w_i + \alpha\,(y - h_{\mathbf{w}}(\mathbf{x})) \times x_i .$$

This is also called the perceptron learning rule in our text book AIMA.

1.2.2 Linear classification with logistic regression

We have seen that passing the output of a linear function through the threshold functions creates a linear classifier. However, the hard threshold function may cause some problems:

1) Hypothesis $h_w(x)$ is not differentiable and is in fact a discontinuous function of its inputs and its weights; this makes learning with the perceptron rule a very unpredictable adventure.
2) Can't deal with noisy or error data.

So we introduce a kind of soft threshold function here: the logistic function

$$Logistic(z) = \frac{1}{1+e^{-z}} .$$

So we now have the hypothesis function as

$$h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w}\cdot\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}} .$$

The weight updating rule for minimizing the loss will be

$$w_i \leftarrow w_i + \alpha\,(y - h_{\mathbf{w}}(\mathbf{x})) \times h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x})) \times x_i .$$

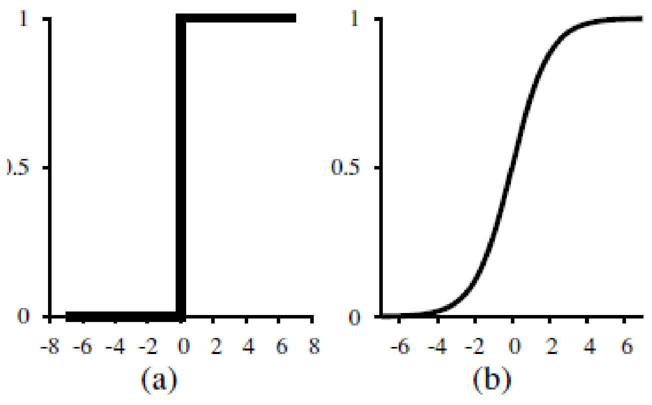


Figure 2 Hard & soft threshold function

2. Implementation

2.1 Decision Tree

Class DecisionTreeLearner:

1) DecisionTree learn(Set<Example> examples, List<Variable> attributes, Set<Example> parent_examples)

   Function returns a decision tree, implemented basing on the recursively learning process, illustrated in Figure 1.

2) String pluralityValue(Set<Example> examples)

   Function returns the plurality classification of the input examples set.

3) String uniqueOutputValue(Set<Example> examples)

   Function returns the single unique output value among the given examples is there is only one, otherwise null.

4) Set<Example> examplesWithValueForAttribute (Set<Example> examples, Variable a, String vk)

   Function returns the subset of the given examples for which Variable a has value vk.

5) int countExamplesWithValueForAttribute(Set<Example> examples, Variable a, String vk)

   Function returns the number of the given examples for which Variable a has value vk.

6) int countExamplesWithValueForOutput(Set<Example> examples, String vk)

   Function returns the number of the given examples for which the output has value vk.

2.2 Linear Classifier

2.2.1 Perceptron Classifier

Class PerceptronClassifier:

1) void update(double[] x, double y, double alpha)

Update rule according to AIMA Eq. 18.7.

2)double threshold(double z)

Hard threshold function.

2.2.2 Logistic Classifier

Class LogisticClassifier:

1) void update(double[] x, double y, double alpha)

Update rule according to 1.2.2 of this report.

2)double threshold(double z)

0/1 sigmoid threshold at z=0.

2.3 Cross Validation

double crossValidation(DecisionTreeLearner learner, set<Example> examples, int k)

Implemented cross validation function to evaluate the models by using the mean error.

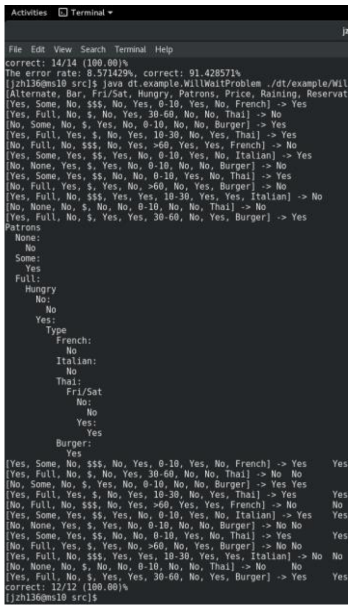
And the k –fold is 10.

3. Results

See the read me file submitted with our code, to learn how to run the experiments on different examples. Here are the results of the experiments:

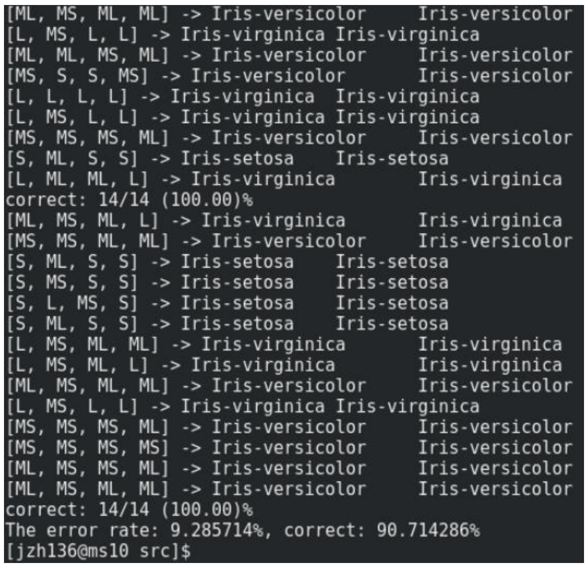
3.1 Decision Tree

WillWait Example:



```
correct: 14/14 (100.00)%
The error rate: 8.571429%, correct: 91.428571%
[jzh136@ms10 src]$ java dt.example.WillWaitProblem ./dt/example/Wil
[Alternate, Bar, Fri/Sat, Hungry, Patrons, Price, Raining, Reservat
[Yes, Some, No, $$$, No, Yes, 0-10, Yes, No, French] -> Yes
[Yes, Full, No, $, No, Yes, 30-60, No, No, Thai] -> No
[No, Some, No, $, Yes, No, 0-10, No, No, Burger] -> Yes
[Yes, Full, Yes, $, No, Yes, 10-30, No, Yes, Thai] -> Yes
[No, Full, No, $$$, No, Yes, >60, Yes, Yes, French] -> No
[Yes, Some, Yes, $$, Yes, No, 0-10, Yes, No, Italian] -> Yes
[No, None, Yes, $, Yes, No, 0-10, No, No, Burger] -> No
[Yes, Some, Yes, $$, No, No, 0-10, Yes, No, Thai] -> Yes
[No, Full, Yes, $, Yes, No, >60, No, Yes, Burger] -> No
[Yes, Full, No, $$$, Yes, Yes, 10-30, Yes, Yes, Italian] -> No
[No, None, No, $, No, No, 0-10, No, No, Thai] -> No
[Yes, Full, No, $, Yes, Yes, 30-60, No, Yes, Burger] -> Yes
Patrons
  None:
    No
  Some:
    Yes
  Full:
    Hungry
      No:
        No
      Yes:
        Type
          French:
            No
          Italian:
            No
          Thai:
            Fri/Sat
              No:
                No
              Yes:
                Yes
          Burger:
            Yes
[Yes, Some, No, $$$, No, Yes, 0-10, Yes, No, French] -> Yes     Yes
[Yes, Full, No, $, No, Yes, 30-60, No, No, Thai] -> No  No
[No, Some, No, $, Yes, No, 0-10, No, No, Burger] -> Yes Yes
[Yes, Full, Yes, $, No, Yes, 10-30, No, Yes, Thai] -> Yes       Yes
[No, Full, No, $$$, No, Yes, >60, Yes, Yes, French] -> No       No
[Yes, Some, Yes, $$, Yes, No, 0-10, Yes, No, Italian] -> Yes    Yes
[No, None, Yes, $, Yes, No, 0-10, No, No, Burger] -> No No
[Yes, Some, Yes, $$, No, No, 0-10, Yes, No, Thai] -> Yes        Yes
[No, Full, Yes, $, Yes, No, >60, No, Yes, Burger] -> No No
[Yes, Full, No, $$$, Yes, Yes, 10-30, Yes, Yes, Italian] -> No  No
[No, None, No, $, No, No, 0-10, No, No, Thai] -> No     No
[Yes, Full, No, $, Yes, Yes, 30-60, No, Yes, Burger] -> Yes     Yes
correct: 12/12 (100.00)%
[jzh136@ms10 src]$
```

Iris Example:



```
[ML, MS, ML, ML] -> Iris-versicolor     Iris-versicolor
[L, MS, L, L] -> Iris-virginica Iris-virginica
[ML, ML, MS, ML] -> Iris-versicolor     Iris-versicolor
[MS, S, S, MS] -> Iris-versicolor       Iris-versicolor
[L, L, L, L] -> Iris-virginica  Iris-virginica
[L, MS, L, L] -> Iris-virginica Iris-virginica
[MS, MS, MS, ML] -> Iris-versicolor     Iris-versicolor
[S, ML, S, S] -> Iris-setosa    Iris-setosa
[L, ML, ML, L] -> Iris-virginica        Iris-virginica
correct: 14/14 (100.00)%
[ML, MS, ML, L] -> Iris-virginica       Iris-virginica
[MS, MS, ML, ML] -> Iris-versicolor     Iris-versicolor
[S, ML, S, S] -> Iris-setosa    Iris-setosa
[S, MS, S, S] -> Iris-setosa    Iris-setosa
[S, L, MS, S] -> Iris-setosa    Iris-setosa
[S, ML, S, S] -> Iris-setosa    Iris-setosa
[L, MS, ML, ML] -> Iris-virginica       Iris-virginica
[L, MS, ML, L] -> Iris-virginica        Iris-virginica
[ML, MS, ML, ML] -> Iris-versicolor     Iris-versicolor
[L, MS, L, L] -> Iris-virginica Iris-virginica
[MS, MS, MS, ML] -> Iris-versicolor     Iris-versicolor
[MS, MS, MS, MS] -> Iris-versicolor     Iris-versicolor
[ML, MS, MS, ML] -> Iris-versicolor     Iris-versicolor
[ML, MS, ML, ML] -> Iris-versicolor     Iris-versicolor
correct: 14/14 (100.00)%
The error rate: 9.285714%, correct: 90.714286%
[jzh136@ms10 src]$
```

3.2 Perceptron

In earthquake-clean dataset

		Accuracy Rate		
		Learning Rate		
		0.01	0.05	0.1
Steps	1000	89.7	88.7	85.3
	10000	96	96.3	94 .3
	100000	100	100	100

In earthquake-noisy dataset

		Accuracy Rate		
		Learning Rate		
		0.01	0.05	0.1
Steps	1000	87.1	83.8	82
	10000	92	86.7	85.7
	100000	92.1	90.5	88.6

In Iris example

		Accuracy Rate		
		Learning Rate		
		0.01	0.05	0.1
Steps	1000	93.3	87	81
	10000	90	90	97
	100000	96	92	97

3.3 Logistic Classifier

In earthquake-clean data

		Accuracy Rate		
		Learning Rate		
		0.01	0.05	0.1
Steps	1000	88	96	94.3
	10000	93	95	92
	100000	100	94	96

In earthquake-noisy dataset

		Accuracy Rate		
		Learning Rate		
		0.01	0.05	0.1
Steps	1000	83.57	88.8	92
	10000	93.3	93.3	92
	100000	91.7	92	92

In Iris Example

		Accuracy Rate		
		Learning Rate		
		0.01	0.05	0.1
Steps	1000	71	85.6	89
	10000	95.6	94.4	95.6
	100000	96.7	96.7	96.7

4. Analysis

4.1 Decision Tree

The WillWait example only have 15 examples, so we did not implement the cross validation on this example. But we get an exact same decision tree model in the text book. So we can believe that the decision tree learning algorithm is well implemented.

For the Iris Example, we can see that the decision tree we got from the learning process has a strong ability of generalization. It has an over 90% accuracy rate (on average) on predict the decision according to the input.

4.2 Linear Classifier

According to the results, we can draw a general conclusion that the accuracy arises as the number of steps increases and the value of learning rate decreases.
And comparing the two kind of classifiers on the noisy dataset of the earthquake example. We can found that the logistic classifier (with a soft threshold) has a better performance that the perceptron one. It is because that hard threshold functions can't deal with the noise or error in the dataset.

5. Reference

[1]. Artificial Intelligence: A Modern Method 3rd Edition. Stuart J.Russell, Peter Norvig