Machine Learning Project 1

jjjj222

1 Overview

This ID3 program is implemented in Ruby. It uses information gain discribed in textbook as splitting criterion to build the tree, and it applies error-complexity pruning for pruning. The datasets used for testing are Car Evaluation [1], Credit Approval [2], Iris [3], Chronic Kidney Disease [4], and Mushroom [5]. Next section will discuss how to setup the program, followed by implementation details and experimental results.

2 Setup

2.1 Installation

There is no installation requirement since this ID3 program is developed in Ruby, However, please make sure the Ruby executable is at the path:

```
/usr/bin/ruby
```

2.2 Execution

Type the following command under the same directory as "`id3.rb`":

```
./id3.rb <data_file> <attribute_file>
```

The format of "`<data_file>`" and "`<attribute_file>`" are explained below.

2.2.1 Data File

A data file is a comma-separated file with one example per line, and the target attribute should be placed in the last column, with missing data specified as "`?`". Note that all the white space (ex: space, tab) will be removed before parsing.

```
Sunny,       85,     High,   Weak,   No
Rain,        57,     High,   Weak,   Yes
Rain,        64,     Normal, ?,      Yes
Rain,        ?,      Normal, Weak,   No
Sunny,       65,     Normal, Strong, Yes
Overcast,    33,     ?,      Strong, Yes
...
```

2.2.2 Attribute File

In attribute file, every line indicates an attirbute, and there should be exact 2 columns in a line: the first column contains the name of the attribute, and the second column specifies either this attribute is continuous (c) or discrete (d). Note that the total number of lines in the attribute file should be the same as the total number of columns in data file.

```
Outlook,        d
Temperature,    c
Humidity,       d
Wind,           d
PlayTennis,     d
```

2.3 Project Structure and Evnironment

The development environment is as follows:

- `OS: OSX El Capitan 10.11.3`
- `Ruby: ruby 2.3.0p0 (2015-12-25 revision 53290)[x86_64-darwin15]`

The project structure is as follows:

- `id3.rb` : excutable
- `report.pdf` : report
- `makefile`: for test only
- `data/` : datasets for testing
- `test/` : results

3 Implementation

3.1 Pre-processing

Both data and attribute file are parsed and pre-processed in "`class ID3Tester`". All possible values for each discrete attribute will be extracted, and examples will be shuffled and then be partitioned into 10 parts for 10-fold cross-validation. In each iteration, 1 part of the sub-examples will be kept for testing, and the rest will be passed down to "`class ID3`" for tree construction.

3.2 Learning

The ID3 tree is built and pruned in "`class ID3`". All the examples used for leaning will be shuffled again, and then 1/3 of them are put aside for pruning; the rest 2/3 examples will be passed down to the root node "`class TreeNode`", and the algorithm will run on each node, building ID3 tree recursively.

For each node, ID3 calculates information gain for each candidate attributes, and then

pick the one with the greatest information gain as target attribute. Based on the partition of the target attribute, sub-examples will be passed down to each child node respectively, used for building the sub-trees. This recursive construction will stop while there is no candidate attirbutes or the maximum information gain is 0.

Note that the target attribute will be kept in the candidate attributes if it's continuous, because ID3 might decide to use it for partition again. The tree-building algorithm is in function "`ID3#build_rec`".

3.2.1 Information Gain

For discrete attributes, examples in current node will be separated into sub-examples based on their values of the target attribute; for continuous attributes, all values of the target attribute in current node will be extracted, sorted and uniquified, and then every mid point between 2 consecutive values will be used as splitting point. For each splitting point, examples will be partitioned into 2 groups, one with greater values than splitting point and the other with smaller ones. Therefore, instead of one, continuous attributes might have many different partitions.

Every partitions come with an information gain, and the equations for the gain is the same as those on the textbook.

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

$$Gain(S, A) = Entropy(S) - \sum_{S_v \in P_S} \frac{|S_v|}{|S|} Entropy(S_v)$$

Where S denotes the examples in the current node; p_i is the propotional of class i; P_S is the set of sub-examples after partition, and S_v is one of the sub-examples.

The calculations are in the function "`LearningData#information_gain_with_split(attribute)`"

3.2.2 Missing Value

In construction phase, examples with missing values are handled during the partition stage. While partitioning, the examples with known value will be separated first, and then those with unkonwn value will be assigned to one of the sub-examples with certain probabilities, which are based on the ratio of the number of examples in each sub-examples.

For instance, if we have, after partition, 2 sub-examples A and B, and the number of examples in A = 4 and B = 6, then each example with unkonwn value will be assigned to A and B with 40% and 60% chances respectively. It's implemented in function "`LearningData#split(attribute)`" for discrete attribute and function "`LearningData#split_at(value)`" for continuous attribute.

In test phase, examples with missing values will traverse the tree based on the probabilities propotional to the total number of examples in each sub-tree. For instance, if

the total examples in the sub-tree A, B and C are 2, 3 and 5, then examples with unknown value will have 20%, 30% and 50% posssibilities to go down the sub-tree A, B and C respectively. It's implemented in function "`TreeNode#classify(example)`"

3.3 Pruning

This ID3 uses error-complexity pruning. In each round, the α value for every node will be calculated as follows:

$$\alpha = \frac{R(t) - R(T_t)}{N_T - 1}$$

Where $R(T_t)$ is the error rate of the sub-tree before pruning; $R(t)$ is the error rate after pruning, and N_T is the total number of leaves in sub-tree.

In the end of each round, the node with the minimum α value will be removed, forming a new tree with less leaves. The pruning will stop when there is only root node left, and we will have a series of trees, such as those showed below.

```
0.05
 `- root [151 ckd, 89 notckd]
     |- Serum_Creatinine=[, 1.25) [26 ckd, 89 notckd]
     |   |- Specific_Gravity=1.020 [6 ckd, 49 notckd]
     |   |   |- Albumin=0 [1 ckd, 49 notckd]
     |   |   |   |- Red_Blood_Cells=normal, class=notckd [0 ckd, 49 notckd]
     |   |   |   `- Red_Blood_Cells=abnormal, class=ckd [1 ckd, 0 notckd]
     |   |   |- Albumin=1, class=ckd [2 ckd, 0 notckd]
     |   |   |- Albumin=4, class=ckd [1 ckd, 0 notckd]
     |   |   |- Albumin=3, class=ckd [2 ckd, 0 notckd]
     |   |   |- Albumin=2, class=notckd [0 ckd, 0 notckd]
     |   |   `- Albumin=5, class=notckd [0 ckd, 0 notckd]
     |   |- Specific_Gravity=1.025, class=notckd [0 ckd, 40 notckd]
     |   |- Specific_Gravity=1.010, class=ckd [9 ckd, 0 notckd]
     |   |- Specific_Gravity=1.015, class=ckd [10 ckd, 0 notckd]
     |   `- Specific_Gravity=1.005, class=ckd [1 ckd, 0 notckd]
     `- Serum_Creatinine=[1.25, ), class=ckd [125 ckd, 0 notckd]

0.041666666666666664
 `- root [151 ckd, 89 notckd]
     |- Serum_Creatinine=[, 1.25) [26 ckd, 89 notckd]
     |   |- Specific_Gravity=1.020, class=notckd [6 ckd, 49 notckd]
     |   |- Specific_Gravity=1.025, class=notckd [0 ckd, 40 notckd]
     |   |- Specific_Gravity=1.010, class=ckd [9 ckd, 0 notckd]
     |   |- Specific_Gravity=1.015, class=ckd [10 ckd, 0 notckd]
     |   `- Specific_Gravity=1.005, class=ckd [1 ckd, 0 notckd]
     `- Serum_Creatinine=[1.25, ), class=ckd [125 ckd, 0 notckd]

0.16666666666666666
 `- root [151 ckd, 89 notckd]
     |- Serum_Creatinine=[, 1.25), class=notckd [26 ckd, 89 notckd]
     `- Serum_Creatinine=[1.25, ), class=ckd [125 ckd, 0 notckd]

0.36666666666666664
 `- root, class=ckd [151 ckd, 89 notckd]
```

The smallest tree within one standard deviation of the minimum error rate will be chosen as the final decision.

```
ID3 (6 leaves)
 `- root [151 ckd, 89 notckd]
     |- Serum_Creatinine=[, 1.25) [26 ckd, 89 notckd]
     |   |- Specific_Gravity=1.020, class=notckd [6 ckd, 49 notckd]
     |   |- Specific_Gravity=1.025, class=notckd [0 ckd, 40 notckd]
     |   |- Specific_Gravity=1.010, class=ckd [9 ckd, 0 notckd]
     |   |- Specific_Gravity=1.015, class=ckd [10 ckd, 0 notckd]
     |   `- Specific_Gravity=1.005, class=ckd [1 ckd, 0 notckd]
     `- Serum_Creatinine=[1.25, ), class=ckd [125 ckd, 0 notckd]
```

The pruning is implemented in function "`ID3#prune`"

4 Experimental Result

The datasets used for testing are Car Evaluation (car) [1], Credit Approval (crx) [2], Iris (iris) [3], Chronic Kidney Disease (kidney) [4], and Mushroom (mushroom) [5].

All the detail information, including the ID3 tree built in each round, can be found in files "`./test/<CASE>/<CASE>.out`".

name	classifier	leaf	accuracy	SE	95% CI
car	before	218.30	90.91	2.22	(85.96, 95.86)
	after	218.30	90.91	2.22	(85.96, 95.86)
	majority	N/A	70.02	3.76	(61.64, 78.40)
crx	before	110.90	78.41	5.16	(66.89, 89.92)
	after	2.00	85.51	4.05	(76.48, 94.53)
	majority	N/A	55.51	5.84	(42.50, 68.52)
iris	before	5.70	96.00	3.27	(88.72, 100.00)
	after	3.00	96.00	4.42	(86.14, 100.00)
	majority	N/A	21.33	4.00	(12.41, 30.25)
kidney	before	15.70	97.00	4.00	(88.08, 100.00)
	after	11.10	96.00	3.39	(88.44, 100.00)
	majority	N/A	62.50	6.80	(47.33, 77.67)
mushroom	before	33.00	100.00	0.00	(100.00, 100.00)
	after	33.00	100.00	0.00	(100.00, 100.00)
	majority	N/A	51.80	2.46	(46.32, 57.28)

Table 1: Results for ID3, ID3 without pruning, and majority classifier

References

[1] Car Evaluation Data Set
`http://archive.ics.uci.edu/ml/datasets/Car+Evaluation`

[2] Credit Approval Data Set
`http://archive.ics.uci.edu/ml/datasets/Credit+Approval`

[3] Iris Data Set
http://archive.ics.uci.edu/ml/datasets/Iris

[4] Chronic Kidney Disease Data Set
http://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease

[5] Mushroom Data Set
http://archive.ics.uci.edu/ml/datasets/Mushroom