Alternating decision tree

From Wikipedia, the free encyclopedia

An **alternating decision tree** (ADTree) is a machine learning method for classification. It generalizes decision trees and has connections to boosting.

An ADTree consists of an alternation of decision nodes, which specify a predicate condition, and prediction nodes, which contain a single number. An instance is classified by an ADTree by following all paths for which all decision nodes are true, and summing any prediction nodes that are traversed.

History [edit]

ADTrees were introduced by Yoav Freund and Llew Mason.[1] However, the algorithm as presented had several typographical errors. Clarifications and optimizations were later presented by Bernhard Pfahringer, Geoffrey Holmes and Richard Kirkby.[2] Implementations are available in Weka and JBoost.

Motivation [edit]

Original boosting algorithms typically used either decision stumps or decision trees as weak hypotheses. As an example, boosting decision stumps creates a set of T weighted decision stumps (where T is the number of boosting iterations), which then vote on the final classification according to their weights. Individual decision stumps are weighted according to their ability to classify the data.

Boosting a simple learner results in an unstructured set of T hypotheses, making it difficult to infer correlations between attributes. Alternating decision trees introduce structure to the set of hypotheses by requiring that they build off a hypothesis that was produced in an earlier iteration. The resulting set of hypotheses can be visualized in a tree based on the relationship between a hypothesis and its "parent."

Another important feature of boosted algorithms is that the data is given a different distribution at each iteration. Instances that are misclassified are given a larger weight while accurately classified instances are given reduced weight.

Alternating decision tree structure [edit]

An alternating decision tree consists of decision nodes and prediction nodes. **Decision nodes** specify a predicate condition. **Prediction nodes** contain a single number. ADTrees always have prediction nodes as both root and leaves. An instance is classified by an ADTree by following all paths for which all decision nodes are true and summing any prediction nodes that are traversed. This is different from binary classification trees such as CART (Classification and regression tree) or C4.5 in which an instance follows only one path through the tree.

Example [edit]

The following tree was constructed using JBoost on the spambase dataset[3] (available from the UCI Machine Learning Repository).[4] In this example, spam is coded as 1 and regular email is coded as -1.

Wikipedia The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

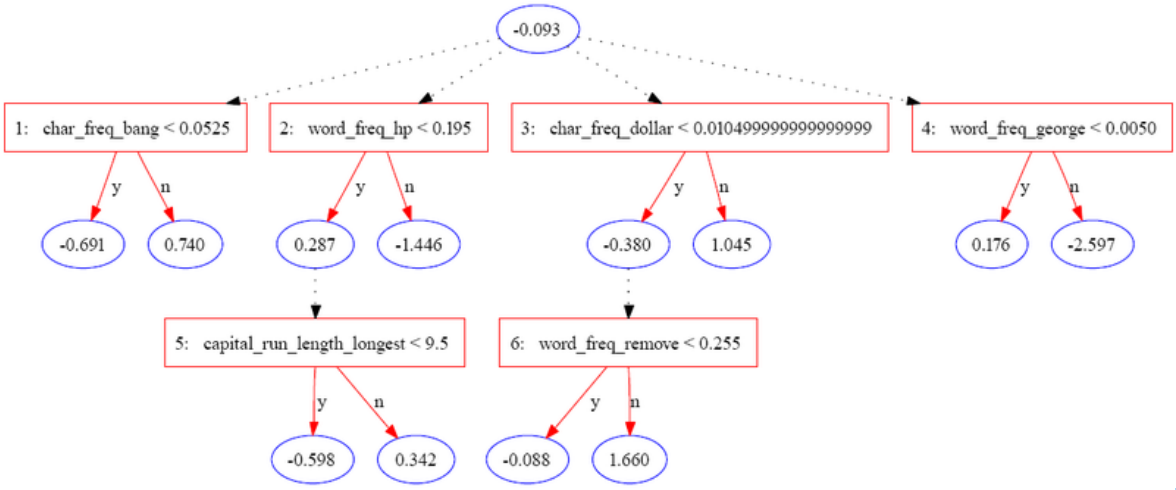
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Español

Edit links



The following table contains part of the information for a single instance.

An instance to be classified

Feature	Value
char_freq_bang	0.08
word_freq_hp	0.4
capital_run_length_longest	4
char_freq_dollar	0
word_freq_remove	0.9
word_freq_george	0
Other features	...

The instance is scored by summing all of the prediction nodes through which it passes. In the case of the instance above, the score is calculate as

Score for the above instance

Iteration	0	1	2	3	4	5	6
Instance values	N/A	.08 < .052 = f	.4 < .195 = f	0 < .01 = t	0 < 0.005 = t	N/A	.9 < .225 = f
Prediction	-0.093	0.74	-1.446	-0.38	0.176	0	1.66

The final score of 0.657 is positive, so the instance is classified as spam. The magnitude of the value is a measure of confidence in the prediction. The original authors list three potential levels of interpretation for the set of attributes identified by an ADTree:

- Individual nodes can be evaluated for their own predictive ability.
- Sets of nodes on the same path may be interpreted as having a joint effect
- The tree can be interpreted as a whole.

Care must be taken when interpreting individual nodes as the scores reflect a re weighting of the data in each iteration.

Description of the algorithm [edit]

The inputs to the alternating decision tree algorithm are:

- A set of inputs $(x_1, y_1), \ldots, (x_m, y_m)$ where x_i is a vector of attributes and y_i is either -1 or 1. Inputs are also called instances.
- A set of weights w_i corresponding to each instance.

The fundamental element of the ADTree algorithm is the rule. A single rule consists of a precondition, a condition, and two scores. A condition is a predicate of the form "attribute <comparison> value." A precondition is simply a logical conjunction of conditions. Evaluation of a rule involves a pair of nested if statements:

```
if(precondition)
    if(condition)
        return score_one
    else
        return score_two
    end if
else
    return 0
end if
```

Several auxiliary functions are also required by the algorithm:

- $W_+(c)$ returns the sum of the weights of all positively labeled examples that satisfy predicate c
- $W_-(c)$ returns the sum of the weights of all negatively labeled examples that satisfy predicate c
- $W(c) = W_+(c) + W_-(c)$ returns the sum of the weights of all examples that satisfy predicate c

The algorithm is as follows:

```
1  function ad_tree
2  input Set of m training instances
3
4  w_i = 1/m for all i
5  a = 1/2 ln( W_+(true) / W_-(true) )
6  R_0 = a rule with scores a and 0, precondition "true" and condition "true."
7  P = {true}
8  C = the set of all possible conditions
9  for j = 1...T
10     p ∈ P, c ∈ C get values that minimize
   z = 2( sqrt(W_+(p ∧ c) W_-(p ∧ c)) + sqrt(W_+(p ∧ ¬c) W_-(p ∧ ¬c)) ) + W(¬p)
11     P+ = p ∧ c + p ∧ ¬c
12     a_1 = 1/2 ln( (W_+(p ∧ c) + 1) / (W_-(p ∧ c) + 1) )
13     a_2 = 1/2 ln( (W_+(p ∧ ¬c) + 1) / (W_-(p ∧ ¬c) + 1) )
14     R_j = new rule with precondition p, condition c, and weights a_1 and a_2
15     w_i = w_i e^(-y_i R_j(x_i))
16 end for
17 return set of R_j
```

The set $\mathcal{P}$ grows by two preconditions in each iteration, and it is possible to derive the tree structure of a set of rules by making note of the precondition that is used in each successive rule.

Empirical results [edit]

Figure 6 in the original paper[1] demonstrates that ADTrees are typically as robust as boosted decision trees and boosted decision stumps. Typically, equivalent accuracy can be achieved with a much simpler tree structure than recursive partitioning algorithms.

References [edit]

1. ^ a b Yoav Freund and Llew Mason. The Alternating Decision Tree Algorithm. Proceedings of the 16th International Conference on Machine Learning, pages 124-133 (1999)
2. ^ Bernhard Pfahringer, Geoffrey Holmes and Richard Kirkby. Optimizing the Induction of Alternating Decision Trees. Proceedings of the Fifth Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining. 2001, pp. 477-487
3. ^ "UCI Machine Learning Repository". Ics.uci.edu. Retrieved 2012-03-16.
4. ^ "UCI Machine Learning Repository". Ics.uci.edu. Retrieved 2012-03-16.

External links [edit]

- An introduction to Boosting and ADTrees (Has many graphical examples of alternating decision trees in practice).
- JBoost software implementing ADTrees.

Categories: Decision trees | Classification algorithms