CS 6604: Data Mining Fall 2007

Lecture 10 — September 19, 2007

Lecture: Naren Ramakrishnan *Scribe: Seungwon Yang*

1 Overview

In the previous lecture we examined the decision tree classifier and choices for impurity function whose main property is concavity. (Question: How can we define concavity without using the derivative?) In the BOAT algorithm, using bootstrapping, we obtain confidence intervals for the nodes (when the node attribute type is numerical) as well as a 'tentative' decision tree. We can find the final splitting criterion by scanning all the database tuples from secondary storage, and calculating the minimum value of the impurity function in the confidence interval.

In this lecture we continue discussing the BOAT algorithm, describe deriving rules from trees, both simultaneous and sequential covering, classifier validation methods, and other ways of classification. In addition, a new topic, Relational Datamining, is introduced.

2 BOAT algorithm

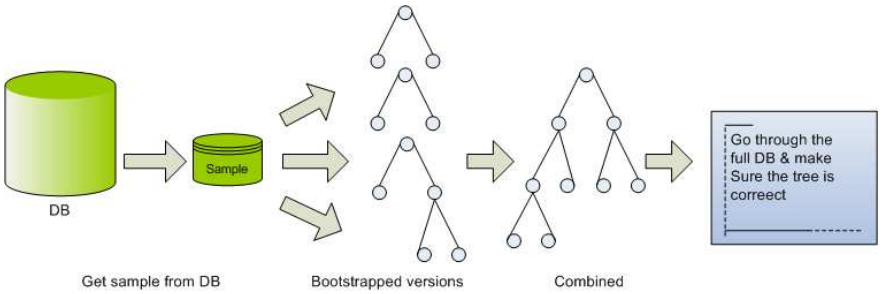

Figure 1: BOAT algorithm

Figure 1 shows the BOAT algorithm steps. Once the trees (from bootstrapping) are combined, each node's exact splitting criteria should be found. In order to be sure the split point in the nodes are not outside the confidence interval, we need to check the minimum value of the impurity function is the global minimum, not the local one. To do this, there should be lots of impurity function coefficient value calculations - inside and outside of the confidence intervals of nodes -, which is too much to store in the memory simultaneously during the full scan of DB. To do this, BOAT uses the concavity property of the impurity function - it is guaranteed that the minimum value will lie on the convex hull [1] (of what?).

3 Simultaneous & sequential covering rules

Think of decision trees as a compaction of rules.

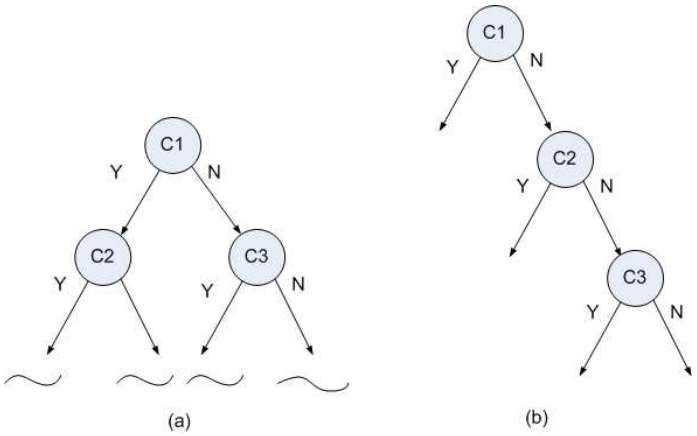

Figure 2: (a)Simultaneous covering tree; (b)Sequential covering tree

3.1 Simultaneous covering

Write down each path in Figure 2 (a) as a rule. This gives a total of four rules:

- If $C_1 = Y$ and $C_2 = Y$ then ...
- If $C_1 = Y$ and $C_2 = N$ then ...
- If $C_1 = N$ and $C_2 = Y$ then ...
- If $C_1 = N$ and $C_2 = N$ then ...

3.2 Sequential covering

As mentioned in lecture 8, rules are not required to be ordered in a simultaneous covering. The decision tree in Figure 2 (b) has a special structure, i.e., the "yes" branch of each node leads to a leaf. This is known as a decision list and is similar to an if-then-else construct. It implies that the rules must be applied in a certain order.

- if $C_1 = Y$ then ...
- else
  - $C_2 = Y$ then ...
  - else
    - if $C_3 = Y$ then ...
    - else ...

To find such rules we find one rule, remove the data points covered (classified correctly) by that rule, find the next rule, and so on.

If we have lots of features, it's better to use sequential covering. Whereas if there are not many features, simultaneous covering might be more useful since it shares conditions among its rules.

3.2.1 Levelwise search for rules in sequential covering algorithms

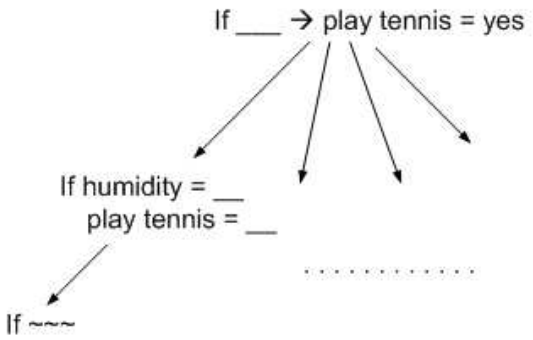

Figure 3: Level-wise search

The level-wise search in Figure 3 is different from the confident rule search because the sequence is growing in the figure above, i.e., the set of antecedents is growing down the tree. Beam Search algorithm, with beam width $= k$, keeps around the most promising k conditions in exploration.

The CN2 rule induction algorithm [2] outputs an ordered set of if-then rules which constitute a sequential covering (decision list). CN2 is designed to cope with noisy data. The heuristic function determines termination of the search based on the estimate of the noise in the data, which might result in rules with less than 100 % confidence since some data might be classified incorrectly. However, it has been shown to perform well with new data (i.e., has good generalization capabilities).

How can we do this in the secondary storage with the sequential algorithm? (no one appears to have tried it yet)

4 More on classification

4.1 Training and test set preparation

We use the training set to build the decision tree and use the test set to evaluate the performance of the learned tree. Table 1 shows some common distributions of training and test sets.

Training set	Test set
80%	20%
90%	10%

Table 1: Training and test set distribution

An undesirable situation results when the testset influences learning. One of the most common mistakes is to take an algorithm that has parameters (e.g., α, β), train the algorithm with different parameter value combinations, then pick the best performing one and then write a report about it. We need to have a validation set which is completely not known. Section 4.2 is about validation methods.

4.2 Validation methods

K-fold cross-validation method evaluates the performance of an algorithm as follows.

1. Partition the dataset into K parts
2. Retain a single part and train on the remaining (K - 1) parts
3. Test on the retained K-th part

By repeating steps 2 and 3, we have K times (folds) training and test sessions. These K results are then typically averaged to produce a single performance estimation. A special case of K-fold cross-validation is called *Leave-one-out cross-validation* which uses a single 'observation' from the original dataset as the test set and use the remaining observations as the training set. Hence it is exactly the same as *K-fold cross-validation* method when K is equal to the number of observations in the original dataset.

4.3 Other ways of classification

Let's say there is a set of points in the space. We want to find the hyperplane that separates them. As shown in Figure 4 (a), if we model it with a decision tree, the tree essentially induces a set of isothetic hyperplanes. For a 'diagonal' separation boundary, the tree will get enormously complicated and follow a zig-zag pattern.

There are algorithms that directly learn hyperplanes. One example are decision trees that have linear combinations of attributes as conditions. Another example is the support vector machine (SVM) which also learns a hyperplane separating the two classes.

Notice that there can be many hyperplanes separating given classes. The contribution of SVM is that it defines a criterion by which there can be only one hyperplane (optimizing it). That criterion has to do with the distance from the hyperplane to the nearest point on either side. The SVM method (Figure 4 (b)) then finds the hyperplane that maximizes this minimum distance.

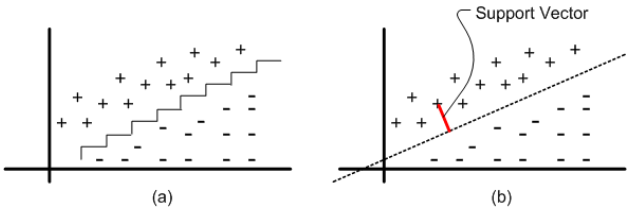

Figure 4: (a) Zigzag-shaped decision boundary; (b) Separating hyperplane and support vector in SVM.

The SVM comes with the ability to do the 'kernel trick.' The trick converts a linear classifier into a non-linear one by using a mapping function to map the original data into a high dimensional feature space.

5 Next topic - Relational Data mining

What does it mean? It is learning from multiple relations. Table 3 might not contain enough information by itself to determine a classification rule (for predicting if somebody can get a credit card approval). But given

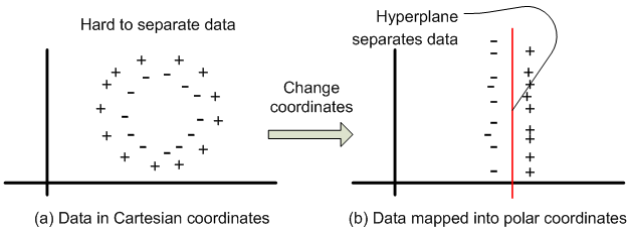

Figure 5: (a) Hard-to-separate data for a linear decision boundary; (b) Data suitably transformed to enable separation by linear boundaries.

...	...
Elizabeth	Jones smith
...	...
...	...
...	...

Table 2: Married-to

Name	Salary	CC
Dan Brown	$2M	Y
Jones Smith	$300K	Y
Elizabeth	$10K	Y
Naren	$20K	N

Table 3: Credit card approval

another table (e.g., Table 2) which gives relevant information, we might learn a pattern such as:

- *Creditrating(x, good)* $\leftarrow$ *Salary(x, y)*, $y \geq Q$.
- *Creditrating(x, good)* $\leftarrow$ *Marriedto(x, y), Salary(y, z)*, $z \geq Q$.

One way to learn such patterns is by multiplying the two tables, i.e, flattening them by joins. This is called 'propositionalization.' However, naive propositionalization is not an ideal way to mine multiple relations; we will investigate true relational techniques soon.

Let us recap some rules of logic.

- Modus Ponens: *Given* $\alpha, \alpha \rightarrow \beta$ *concludes* β
- Modus Tollens: *Given* $\neg\beta, \alpha \rightarrow \beta$ *concludes* $\neg\alpha$
- Resolution: *If* $\neg\alpha \vee \beta$ *and* $\neg\beta \vee \gamma$ *then* $\neg\alpha \vee \gamma$ (in other words, $\alpha \rightarrow \beta$, $\beta \rightarrow \gamma$ then $\alpha \rightarrow \gamma$)

Given the following facts

1. All CS courses are easy.
2. CS 6604 is a CS course.
3. CS 6604 is easy.

In deductive logic, (1) + (2) $\rightarrow$ (3) when Modus Ponens is applied. But in data mining, we try to have a generalized rule from the specifics (e.g., (2) + (3) $\rightarrow$ (1))

More in next class.

References

[1] Gehrke, J., Ganti, V., Ramakrishnan, R. and Loh, W., *BOAT - Optimistic Decision Tree Construction*, In Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (Philadelphia, Pennsylvania, United States, May 31 - June 03, 1999). SIGMOD '99. ACM Press, New York, NY, 169-180. DOI= http://doi.acm.org/10.1145/304182.304197

[2] Clark, P. and Niblett, T., *The CN2 Induction Algorithm*, Mach. Learn. 3, 4 (Mar. 1989), 261-283. DOI= http://dx.doi.org/10.1023/A:1022641700528