CS 345 - Topics in Data Warehousing

Autumn Quarter, 2004

Assignment #4 – Due Tuesday, November 30

Assignment #3 turned out to be much more difficult than it was supposed to be. To compensate for the difficulty of Assignment #3, the following policy will be used:

- All students who made a good-faith effort to complete Assignment #3 will receive full credit for Assignment #3, regardless of whether they were completely successful at finishing the assignment.
- Everyone will have two options for Assignment #4. You may either:
  1. Finish Assignment #3, or
  2. Write a program that learns a decision tree classifier as described in this handout.
- Grading of Assignment #4 will be based on the completeness and correctness of either (1) your aggregate selection, aggregate creation, and query re-writing programs, or (2) your decision tree learning program, depending on which option you choose.

If you choose the first option, then you do NOT need to implement a decision tree classifier, and you can ignore the rest of this handout. This means that if you were able to successfully complete Assignment #3 before the deadline, then you do not need to do anything for Assignment #4. If you almost had Assignment #3 working, but it wasn't quite finished, then all you need to do for Assignment #4 is to finish the remaining unfinished pieces of Assignment #3. If you thought that Assignment #3 was impossibly difficult and made little progress on it, then perhaps you are better off writing the program described in this handout instead of attempting to finish Assignment #3.

For extra credit, you may also choose to do both options: complete Assignment #3 and also write a decision tree classifier. You should only attempt the extra credit if you are already finished (or very nearly finished) with your course project!

Assignment Description In this assignment, you will write a program that learns a decision tree classifier. Training data will be provided in the form of two database tables with identical schemas. The data from one table will be used to compute sufficient statistics for building a decision tree, and the data from the other table will be used as a "holdout set" to validate the performance of the decision tree to prevent overfitting.

This document describes the assignment at a fairly high level. More detailed information is available on the Assignments section of the course web page. **Additional information is available from http://cs345.stanford.edu/ in the "Assignments" section.**

Configuration Information As in Assignment #3, the configuration information for your program will be provided through metadata tables. There will be 2 metadata tables, `DT_INFO` and `DT_COLS`.

The `DT_INFO` table has three columns, and this table will contain exactly one row. The columns and their meanings are as follows: The `TABLE_NAME` column holds the name of the table containing the training data; the `HOLDOUT_NAME` column holds the name of the table containing the holdout data that is used for cross-validation; and the `TARGET_COLUMN` column holds the name of the attribute whose value is supposed to be predicted by the decision tree classifier. This attribute will be one of the columns present in the schema of both the training data table and the holdout data table.

The `DT_COLS` table has a single column named `COLUMN_NAME`. This table has one row for each attribute that should be considered by the decision tree classifier when trying to predict the value of the target attribute. The

COLUMN_NAME column holds the name of each attribute column, which will be present in the schema of both the training data table and the holdout data table.

For example, suppose the DT_INFO table contains the row [BABCOCK.CENSUS2, BABCOCK.HOLDOUT, DINCOME1] and the DT_COLS table contains 5 rows with COLUMN_NAMEs of DHOUR89, DWEEK89, ISEX, IMILITARY, and IWORK89. That means that the training data is stored in two tables named BABCOCK.CENSUS2 and BABCOCK.HOLDOUT. Each table contains the following columns: DHOUR89, DWEEK89, ISEX, IMILITARY, IWORK89, DINCOME1. The objective of the decision tree classifier is to predict the value of the DINCOME1 column given the values of the other 5 columns.

Learning the Decision Tree The decision tree classifier is learned using a recursive algorithm. First, the algorithm is run on the entire data set in order to choose the single best attribute to use to partition the data. The "best" attribute is the one with the highest information gain (see below for details of what this means). Let's suppose that the ISEX attribute is selected, and this attribute has two possible values, either 0 or 1. Then the learning algorithm provides recursively by building two child decision trees for the subset of the data where ISEX=0 and another for the subset of the data where ISEX=1. Speaking more generally, the data is partitioned based on the value of the selected attribute, with each distinct value of the attribute becoming a separate partition, and a separate decision tree is built on each partition.

The holdout data is used to determine when the recursion should stop. After selecting the attribute with highest information gain to use to partition the data, your program should perform a check to see whether partitioning based on this attribute actually improves prediction performance on the holdout set, as opposed to doing no partitioning at all. This validation check works as follows: first, determine which value of the target attribute occurs most often in the training data set. The simplest possible decision tree always predicts that the target attribute will be equal to its most common value, regardless of the values of the other attributes. Second, consider what would happen if you partitioned the data set based on the attribute with highest information gain, and then within each partition, predicted that the value of the target attribute would be whatever value was most common within that particular partition. This corresponds to a one-level decision tree instead of a zero-level decision tree. Third, compare the performance of the zero-level decision tree and the one-level decision tree at predicting the value of the target attribute among the records in the holdout data set. If the number of accurate predictions made by zero-level decision tree is greater than equal to the number of accurate predictions made by the one-level decision tree, then splitting on the selected attribute is not likely to be a good idea; you should not perform the split, and the learning algorithm terminates. Otherwise, the split is worthwhile, and you should partition on the selected attribute and recursively construct a decision tree for each partition.

The discussion in the previous paragraph also applies to all recursive invocations of the learning algorithm, except that the phrase "the data set" does not refer to the entire data set, but only to that partition of the data set that is currently under consideration.

Information Gain The information gain from an attribute is defined as the expected reduction in entropy obtained by conditioning on that attribute. The entropy of a data set is defined as:

$$\sum_{i \in target} -p_i \log p_i$$

The summation is taken over all the possible values taken on by the target attribute. p_i represents the fraction of the records in the data set that have the ith distinct value of the target attribute.

To compute the entropy after conditioning on an attribute A, partition the data set based on the value of attribute A, with each distinct value of A becoming its own partition. For each partition, compute the entropy within that partition. When performing the partition-level entropy computation, p_i represents the fraction of the records *in that partition* that take on the ith distinct value of the target attribute, rather than the fraction of the records in the entire data set. The overall expected entropy after conditioning on attribute A is the weighted sum of the partition-level entropies. Each partition-level entropy is weighted by the number of records belonging to that partition. The attribute with highest information gain is equivalent to the attribute that produces the lowest expected entropy.

What To Submit Follow the same procedure as for Assignment #3.