

Tree

Due: 6:00 PM 12/07/09

Introduction

On Homework 0, we presented you with a data set of patients with different symptoms and whether or not they had a new disease. Afterwards, we asked you whether or not a new patient with his different symptoms had this new disease. Recall that a decision tree is used to take an object (defined in terms of some set of features) and return the category that that object is most likely to come from. This makes them perfect for many real world problems, including the diagnosis of diseases.

In this project, you will implement an algorithm that takes data about objects in the world and *learns* from it by building a decision tree that will categorize novel objects in an intelligent way. This is a **machine learning** algorithm, which is undeniably the coolest part of Artificial Intelligence.¹

Decision Tree Learning Algorithm

The algorithm you will implement will be the Decision-Tree-Learning algorithm described in Russell and Norvig, pages 653-660, and especially figure 18.5 on page 658. Study this chapter carefully, as it contains in depth explanations of many of the more complicated aspects of the algorithm.

Take special note of the heuristic used in the "Choose-Attribute" function described on pages 659-660. We expect you to use this heuristic. When choosing the attribute on which to split the data, **choose the attribute with the highest information gain, as defined in the text.**

We would, however, like you to deviate from the book's algorithm in one way. In general, your decision trees will be able to give a more sophisticated kind of answer to the question "What category is this object in?" Since data can often be incomplete or ambiguous, the classifications given by your tree will sometimes be *probability distributions* over categories.

When will you want your decision tree to return a probability distribution? When constructing your decision tree, you will be working recursively down the branches of the tree, filtering the set of examples you are looking at by the attributes you have branched on. Sometimes, you might *run out of examples*. When this happens, you will want to assign that leaf a default classification which reflects the frequency of different categories in the last non-empty set of examples. That relative frequency will be translated into a probability distribution over categories. Other times, you might have *several examples from different categories with identical features*. In these cases,

¹This is a completely unbiased opinion.

you will again want to assign that leaf a classification which is a probability distribution reflecting the relative frequencies of the categories in the data set at that point, instead of using the "Majority-Vote" function described in the book.

For more information, see the provided javadocs.

Assumptions and Data Format

The data sets used in this project contain a number of examples. Each example has m discrete attributes. Let's call them a_1, a_2, \dots, a_m .

Each attribute a_i can take on a value from the range $0, 1, \dots, g_i - 1$. In other words, there are g_i values that attribute a_i can take on.

Each example is classified into one of k categories.

The examples will be read in by the `DataParser` as a string of `ints`: each example has a number to indicate its category, then m numbers to indicate the value of each attribute. See the data-format file (`/course/cs141/src/tree/data/data-format`) for the template of the files read by our parser and an annotated example file. Also in `/course/cs141/src/tree/data/` are several sample data sets to use as examples and in testing.

Bagging

After creating the decision tree classifier, we will want you to perform **bootstrap aggregating** or **bagging** to create a decision forest.

You can read more on bagging at <http://www.dtrek.com/treeforest.htm>. We'll use a slightly simplified method to produce decision forests (or bagged decision trees), as follows:

1. Create M different data sets from your original data set of size n . To create a randomized subset of the original data, you can sample from the original data set Cn times² with replacement, where $0 < C < 1$.

Then, discard any duplicate examples from the subset. Duplicate examples are those that are the same example from the training set. (Two numerically distinct examples are not duplicates even if their attributes are identical) Because of duplicates, you'll end up with somewhat fewer than n examples in each new subset. You should try different values of C to see its effect on performance on classification of the test set.

2. Create a decision trees for each of the M data subsets you created. (That is, grow a forest of M trees)

²Round to the nearest integer.

3. Classify examples in the test set by combining the classification outputted by each decision tree in the forest. There are two ways you can do this.
 - (a) The first method is very straightforward. For each decision tree in the forest and for each example in the test set, consider only the *most probable* or "best" category predicted for that example. Then, add the decisions of the forest together and return the category that has the most decisions in its favor. In other words, take a simple "majority vote."³
 - (b) The second method takes advantage of the more sophisticated, probabilistic classifications you will implement for your decision trees. Rather than aggregating the best categories returned by the forest, you can sum the probabilities of each category reported by each decision tree, and then normalize these values to get a new probability distribution over categories. The decision of the forest is then the best category from *this* distribution.

What To Do

1. Install the support code into your home directory by running `cs141-install tree`.
2. Familiarize yourself with the classes and support code we have provided for you. You can build and open javadocs with the command `ant javadocs` followed by `mozilla javadocs/index.html` & or by visiting <http://cs.brown.edu/courses/cs141/tree>. Also, familiarize yourself with the build file: try `ant help`. It provides extra functionality to help you run multiple trials and generate datasets.
3. Fill in the support classes: `Attributes`, `Examples`. Write a dummy function for `Examples.chooseAttribute()` that just returns an arbitrary `Attribute`. (Later, this procedure will need to be rewritten in in order to take into account **information gain**.)
4. Fill in the `DecisionTree` class. The `DecisionTree.createDecisionTree` should build the induced decision tree from examples, attributes, and the default classification. Again, see Russell and Norvig, page 658, for a great pseudocode summary of the algorithm. You might find the visualization code ⁴ we have provided helpful here. There is an optional flag, `outputfile`, you can pass to the `ant` script and the `main` function will put pictures of your trees in the `graphs/` folder under the name you provide.
5. Fill in the `DecisionForest` class. The constructor should build and create the decision forest so that after it is constructed the `classify()` function is ready to be called on. Use the algorithm described above in the `DecisionForest` section.
6. Change `Examples.chooseAttribute()` to return the function that provides the most information gain from the examples and attributes. Use the multi-class definition:

³Or, more precisely, a *plurality* vote.

⁴Written in Spring 2006 by former student Ethan Schreiber.

Let $\vec{c} = (c_1, \dots, c_n)$ be the normalized distribution of examples over categories where c_1 is the number of examples remaining that are in category 1. Additionally, let $c^{A_i} = (c_1^{A_i}, \dots, c_n^{A_i})$ be the normalized distribution of examples over categories where attribute A has value i and p_{A_i} the number of examples with value i for attribute A normalized by the total number of examples considered.

$\text{Gain}(A) = H(\vec{c}) - \sum_{i=1}^{g_A} p_{A_i} H(c^{A_i})$, where $H(\vec{p})$ is the entropy function ($H(\vec{p}) = -\sum_{i=1}^n p_i \log_2(p_i)$)

- Learn how to use `ant gen-data`. It takes 9 arguments (each with default values in parentheses):

`numCat(2)` = number of choices,
`numAttr(4)` = number of attributes,
`trainExamples(100)` = number of training examples,
`testExamples(25)` = number of test examples,
`maxChoice(4)` = maximum number of choices for any attribute,
`filename(data)` = base filename,
`beginIndex(0)` = starting index for file names,
`numSets` = number of data sets to output(1),
and `noiseFactor(0)` = adds random noise (product of noise factor and a number generated from a uniform distribution between 0 and 1).

- Do the writeup. **As with SAT, leave plenty of time for this step.** Collecting the data needed for this part of the project may be very time consuming, and will constitute a large portion of your grade.
- Be Happy!

Writeup

For this project we want a writeup that answers the following questions:

- How does increasing or decreasing a training set size effect the performance (error) of your decision tree and forest learner? What explains this result? In addition to a written analysis, please create a plot of the training set size vs. performance (defined as percentage of test examples correctly classified) on the test set. For this question, please use a test set of size 50 and vary the number of training examples from 10 to 210 by increment of 20s. Additionally, please set the number of categories to 2, the number of attributes to 15, and the noise factor to 0.5. Please plot the performance of the decision tree algorithm and decision forest of sizes 5, 50, and 500 ($C = 0.5$) on the same graph neatly! We should be able to read the error for both the tree and forest easily. See Figure 18.7, the “happy graph,” in Russell and Norvig page 661. Which algorithm (with parameters if applicable) and training size has the best performance? Why?

2. How does the number of categories in a training and test set effect its performance? Why do you think this is the case? Use again a test size of 50 and the training set size with the best results from part 1. Vary the number of categories from 2 to 10 and plot the performance of the decision tree algorithm and decision forest of sizes 5 and 20 ($C = 0.5$) on the same graph neatly!
3. How does varying C effect random forest performance? Use again a test size of 50 and the training set size with the best results from part 1. Plot performance varying C from .05 to .95, incremented by .05. Explain the effect of varying C on decision forest performance.

For each parameter setting, we would like you to run the algorithm at least 10 times on different randomly generated data sets, and average those results. Why? Because when working with random data sets, there will be *random noise* that affects the error statistics which could distort your results. Averaging the results of several samples will reduce the effect of random noise and contribute to a more precise statistic. (Incidentally, this is similar to how bagging improves machine learning results.) Conveniently for you, the `ant gen-data` is set up to easily generate multiple sets of data at once. Even more conveniently, the tas have created a script, `ant runxtimes`, to average the results of your decision tree and forests over multiple data sets. `ant runxtimes` takes four parameters:

```
basefile = root filename to use (what you gave to ant gen-data)
beginIndex = the index of your first generated data set
numSets = the number of data sets to run your learning algorithm on
numTrees = the number of trees to use (1 for decision tree)
```

This writeup doesn't have to be as extensive as the one you did for SAT. Like many topics in machine learning, decision trees raise some deep and intriguing questions; this assignment is meant as an introduction. Show us that you've thought about each question and tested your intuitions with your algorithm. Also, make sure you have some written explanation of the phenomena you observe in the graphs for each question.

Particularly well-researched or thought-out writeups will get **extra credit**.

What to Handin

Please hand in the following:

1. All of your code.
2. Your writeup, including any graphs or tables.

Our handin script turns in everything in the current directory and all subdirectories. To hand in this project, navigate into the directory containing the files and directories you wish to hand in and run `cs141-handin tree`.