Economics 1660: Big Data Prof. Daniel Björkegren

PS 3: Trees

Poisonous Mushrooms, Continued

A foodie friend wants to cook a dish with fresh collected mushrooms. However, he knows that some wild mushrooms are delicious and others can be deadly.

'There is no test to determine edible versus poisonous mushrooms. Ignore any advice such as "a poisonous mushroom will tarnish a silver spoon," "if it bruises blue, it's poisonous," etc. These are old wives' tales and folk myths, and completely untrue.'[1]

You've managed to collect data on 8,124 mushrooms, their features, and whether they are edible. It would be difficult to remember all of these individual mushrooms, so your goal in this assignment is to determine if there are rules of thumb that can help your friend.

We've transformed the dataset from last week to make all the variables binary; the data is in the file "mushrooms_binary.csv"

1. **Trees**
   Write code to construct a classification tree out of a dataset, of a given number of levels. Here we use a greedy algorithm that may generate a local optimum.

   a. Construct an optimal tree of one level
      - Draw out the corresponding decision tree
      - In a sentence, write the rule of thumb corresponding to the tree
      - What percentage of mushrooms are correctly classified?
   b. Repeat (a), constructing an optimal tree of two levels
   c. Construct an optimal tree of five levels, and compute the percent accuracy. Include a description of the tree output from your code.
   d. Plot the percent accuracy as a function of tree depth
      - What is the maximum accuracy you can obtain? How many levels do you need to obtain this level of accuracy? Your plot should range from 1 to this number of levels. (This problem can get difficult so do the best you can.)
      - *(Hint: If your accuracy remains unchanged for several levels before it continues to improve, try to investigate why this happens)*

2. **Smell**
   Your friend doesn't have a good sense of smell so does not feel confident in evaluating a mushroom's odor. Repeat (1) omitting smell.

[1] http://mdc.mo.gov/discover-nature/outdoor-recreation/mushrooming/basic-mushrooming

3. Analysis

a) Your friend asks for a rule of thumb to remember. What is the tree you would share with him? (You may use one you have constructed so far or construct a new one.)

b) Your friend wants to construct an app ('IsItEdible') which will guide users through a sequence of questions and provide advice on whether a given mushroom is edible. Which tree would you use for this app?

Describe the reasoning behind your response, weighing comprehensibility and accuracy in your answer.

4. Loss Functions

Unless it has perfect accuracy, a tree that you generate can make two types of mistakes. It can mistakenly report that a poisonous mushroom is edible, and it can mistakenly report that an edible mushroom is poisonous. The algorithm you used makes an assumption about how costly these two types of mistakes are.

a) What does your algorithm assume about the relative costs of these two types of mistakes?

Based on your friend's preferences, the loss associated with mistakenly reporting that a poisonous mushroom is edible (which may lead him to eat a poisonous mushroom) is much higher than the loss associated with the other class of mistake (which may lead him to avoid a mushroom that is in fact edible).

b) Assume that eating an edible mushroom yields a utility benefit *U*, eating a poisonous mushroom yields a utility loss of *L*, and eating nothing yields utility zero. Consider a leaf node that represents a sample of mushrooms of which proportion *p* are edible mushrooms. What is the optimal decision rule as a function of *p*? (In what cases should the tree recommend that your friend eat the mushroom?)
c) If L goes to infinity, what is the optimal decision rule as a function of *p*?

Revisit the trees you constructed in Part 1. Assume that your friend's preferences are $U = 1$ and $L \to \infty$.

d) Replace the decision rule used at each leaf of the tree with the resulting optimal decision rule, but otherwise keep the same structure of the tree.
    - Plot the percent accuracy as a function of tree depth. What happened? Why?
    - Plot the expected utility as a function of tree depth.

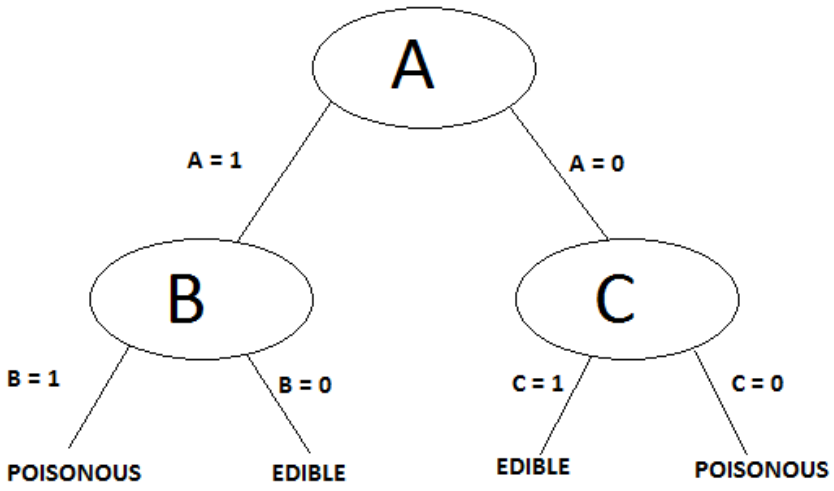
e) Given your friend's preferences, the structure of the tree can be improved. Come up with a method that improves the structure of the tree to increase his expected utility. (This is a difficult problem. Do the best you can.[2]) Using your method:
   - Plot the percent accuracy as a function of tree depth.
   - Plot the expected utility as a function of tree depth.
   - Describe how much your method has improved performance.

Reference: Trees and Python

The recommended way to represent decision trees in Python is as a tuple (*a*, *b*), where *a* is the attribute we are splitting on, and *b* is a dictionary that directs values of *a* to the next step (subtree) in the tree. Take the following example:

$$T = (A, \{1: (B, \{1: \text{Poisonous}, 0: \text{Edible}\}), 0: (C, \{1: \text{Edible}, 0: \text{Poisonous}\})\})$$

Suppose we have a mushroom *M* with the attributes A= 0, B = 1, C = 0, and we want to classify it as either "poisonous" or "edible" with the tree *T*. Looking at *T*, we see that A is the first attribute to split on. Since *M* has A = 0, we continue to the subtree rooted at C. Since *M* has C = 0, we classify *M* as "poisonous." Here is a visual representation of *T*:



***If you need to review tuples or dictionaries, explore the Python documents.

[2] Tweaking the top down algorithm may not be enough. You can start by perturbing existing trees, or try a bottom up algorithm.

Fit and Loss

There are many potential trees we should use; how can we evaluate their performance? When we are classifying a mushroom *m* as poisonous or edible using a decision tree, we are asking a question about *m* at each level of the tree to determine how we should proceed to the next level. Ideally, we'd like to choose questions whose answers give a lot of information about what our tree should predict. If there's a single yes/no question for which "yes" answers always correspond to "edible" outputs and "no" answers always correspond to "poisonous" outputs (or vice versa), this would be a good question to pick. Conversely, a yes/no question for which neither answer gives you much new information about what the prediction should be is probably not a good choice.

We capture this notion of "how much information" with a measure of loss: how well are we able to predict the results at a given node?

Imagine that we have a set *S* of data, each member of which is labeled as belonging to one of a finite number of classes C_1, …, C_n. If all the data points belong to a single class, then there is no real uncertainty, which means we'd like there to be low loss. If the data points are evenly spread across the classes, there is a lot of uncertainty and we'd like there to be higher loss.

We can use different measures of loss. The simplest is *misclassification error*. If p_i is the proportion of the data labeled as class C_i, and C_j is the predicted class for this data, the misclassification error is given by:

$$H(S) = 1 - p_j$$

(all but p_j are misclassified)

When building trees, we can compute the loss associated with a partition. Mathematically, if we partition our data S into subsets S_1, … , S_m containing proportions q_1, … , q_m of the data, then we compute the loss of the partition as a weighted sum:

$$H = q_1H(S_1) + \ldots + q_mH(S_m)$$

A partition has low loss if it splits the data into subsets that themselves have low loss, and high loss if it contains subsets that have high loss. When constructing our tree, we will want to make partitions that yield low loss, so that we can have high certainty about the classifications we make with it.

Building a Tree (ID3 Algorithm)

We will build our decision trees using the relatively simple *ID3* algorithm, which operates in the following manner. Let's say we're given some labeled data, and a list of attributes to consider branching on:

- If the data all have the same label, then create a leaf node that predicts that label and then stop.
- If the list of attributes is empty (i.e., there are no more possible questions to ask), then create a leaf node that predicts the most common label and then stop.
- Otherwise, try partitioning the data by each of the attributes
- Choose the partition with the lowest loss
- Add a decision node based on the chosen attribute
- Recur on each partitioned subset using the remaining attributes.

NOTE: Since this problem set asks you to build trees of a specified number of levels, you will have to add a slight modification to the above formula. Each time you recurse, decrease the number of levels by 1. When *num_levels* gets to 0, you have hit the bottom level of your tree, and you should create a leaf node that predicts the most common label ("poisonous" or "edible") of the data that reaches that subtree.

Using the Stencil Code

For convenience in completing the problem set, stencil code has been provided in mushroom_stencil.py. There are several support functions that will help you build decision trees. Three functions are left for you to fill in. We suggest you complete them in the following order:

- **load_data( )**: This function should load the mushrooms csv file and return the data in the form of a list of tuples (a, b) where 'a' is a dictionary of features and 'b' is the classification variable (either "poisonous" or "edible").
- **build_tree(inputs, num_levels, split_candidates = None)**: Write a recursive function to build a tree with the specified number of levels using the ID3 algorithm. You should use the "partition_by" and "partition_loss_by" support functions to help you. **These are the only support functions you should need to explicitly call at any point in your code.**
- **classify(tree, to_classify)**: Write a recursive function to classify a given observation as "poisonous" or "edible" using the given tree.

You should feel free to deviate from the support code and these suggestions if you wish, as long as you are able to answer the questions in the problem set.

The main( ) function at the bottom of the file is commented out. Once you have your code written, you may uncomment the function and the call to main( ) on the last line of the file. If

your code runs properly, you will print out a 1-level tree and an accuracy score (out of 1). Modify the main function as needed to answer the questions.

Hints

- You may want to explore the csv file before you begin coding. Figure out where the classification variable is, and what format the variable values are in.
- All of the variables in the dataset are binary.
- Remember that when you load the dataset into python, variables will be loaded as strings (not ints, floats, etc.). You may need to do some casting, depending on your implementation.