

Trees on spam

Introduction to Machine Learning (CSCI 1950-F), Summer 2011

In these exercises, you will implement a decision tree classification algorithm, and evaluate its performance in classifying spam e-mails. We will use the “spambase” dataset from the UCI repository: you can visit <http://archive.ics.uci.edu/ml/datasets/Spambase> to understand how the data has been preprocessed and what each dimension of the input points represents. These instructions (this file) can be downloaded from the course website <http://www.dam.brown.edu/people/jmiller/ML> along with code and data for use in the exercises below.

(1) Implementation

The Matlab script `main.m` reads in the dataset and calls the (incomplete) function `tree_build.m` to construct a classification tree using the CART approach. Your task is to fill in the missing details in `tree_build.m`. The missing steps involved computing the “error measure” or “impurity” associated with a given split. Of the three commonly used such measures (misclassification rate, entropy, Gini index), your implementation should use the misclassification rate. Print out and submit the code you added to `tree_build.m`. (Do not submit the whole file, just submit the part you wrote.)

Feel free to modify the m-files (e.g. by using `fprintf` to display various quantities, etc.) in order to understand how they work and get your code working correctly. (For example, there is a line in `tree_build.m` you can uncomment to display the number of points with each call of the function. Also, you can play around with the number of data points n and `minimum_count`. You can visualize the resulting classification tree by enabling the last snippet of code in `main.m`.) However, once your code is working, for the exercises below you should use the original files (with your modifications to compute the impurity in `tree_build.m`).

(2) Evaluate the performance

Run the script `main.m` to evaluate the performance. It uses randomly selected training data sets, over a range of values for the `minimum_count` parameter. Print out and submit the results computed by running `main.m`, along with your answers to the following questions. How does the training performance change as `minimum_count` varies over this range? How does the test performance change? Can you explain why the training performance and test performance behave this way?

(3) Analyze the computational complexity

Big O notation: For a function f of x_1, \dots, x_k , we say that an algorithm “takes $\mathcal{O}(f(x_1, \dots, x_k))$ time” if there exist constants c and d such that whenever $x_i > d$ for all $i = 1, \dots, k$, the algorithm runs in no more than $cf(x_1, \dots, x_k)$ units of time.

(See http://en.wikipedia.org/wiki/Big_O_notation#Multiple_variables .)

Let n be the number of training example points, and let d be the number of dimensions. For this implementation of the algorithm, what is the time required (in terms of n and d) to choose the optimal split for the first node? (State your answer in terms of “big O notation”: for example, $\mathcal{O}(n^3 d^4 \log d)$.) You can assume that sorting n numbers takes $\mathcal{O}(n \log n)$ time. Your answer should be theoretically-based, rather than empirically-based.

Challenge problem

This problem is just for fun. You don't need to turn in a solution — it is optional and will not be graded.

Can you find an upper bound for the total time required to build the tree using this implementation of the algorithm? Would you expect it to take about this much time on average, or would it typically be much less? Why?

Can you think of a more efficient way to implement the algorithm?

Can you find an upper bound for the total time required to classify a new point?

How does the computational complexity of training (building the tree) and testing (classifying a new point) compare with the k nearest neighbor algorithm?