CpSc 2120: Algorithms and Data Structures

Instructor: Dr. Brian Dean — Fall 2014
Webpage: `http://www.cs.clemson.edu/~bcdean/` — MWF 9:05-9:55
Handout 17: Homework #4 — Vickery 100

1 Nearest-Neighbor Classification using kd-Trees

This assignment gives us a chance to work with multi-dimensional data by building and searching kd-trees. We will also gain experience with nearest-neighbor classification, a common technique in the domain of machine learning that can be quite relevant in many situations in practice.

The dataset we will use for this exercise, derived from the study in [1], is the following:

`/group/course/cpsc212/f14/hw04/wine.txt`

It contains data on several thousand brands of white wine, each with a real-valued human-assessed quality rating (in the range 0-10) and 11 real-valued physiochemical attributes (pH, alcohol content, sulphate concentration, sugar content, etc.). We can think of this data as a collection of points in 11-dimensional space, each labeled with a real number in the range 0-10. Our goal is to see if we can predict the quality of a wine just based on physiochemical properties alone. As is typical in machine learning, it is not known how successful we will be at either goal. We may later test our code on other datasets as well, depending on their availability.

2 Reading the Data

The first line of each data file contains two integers, n and d, specifying the number of data points and the number of attributes (dimensions) of each point. The next n lines each contain $d+1$ real values that describe a single data point: the first number is a "label" for the data point (i.e., the quality score); the remaining d values describe the d attributes of the data point. Our dataset therefore has n labeled points in d-dimensional space. Let $x_i(1) \ldots x_i(d)$ denote the d coordinate values for data point i. All the points in the dataset are guaranteed to be distinct.

Since we will be computing distances between points, we want to make sure each dimension contributes the roughly the same. For example, if one dimension was measured in millimeters and another in meters, distances could end up being dominated by the first dimension, being orders of magnitude times the scale of the second. We therefore apply the following two steps to each dimension $j = 1 \ldots d$:

1. Translate the data so that it has zero mean: $\sum_i x_i(j) = 0$.
2. Then rescale the data so it has unit variance: $\sum_i x_i(j)^2 = 1$.

These two steps will ensure that the data has roughly the same scale in each dimension, so all dimensions will figure in equally to our distance calculations.

After rescaling the data, you should build the data into a kd-tree. When deciding where to split in each dimension, feel free to choose a random point as the splitting point (ideally, we would use the median, but random choices should also lead to reasonable performance). Be sure to be consistent about how ties are broken; for example, if you split on a point with coordinate value v, then you may want to adopt the convention that points with values less than or equal to v should go in the left subtree.

3 Classification

To see how well nearest neighbor classification works, we will use "leave one out" testing, where we guess the label of each point by temporarily pretending that it is absent from the data set. If we are using nearest neighbor classification, then we would guess that each point should be labeled the same as its nearest neighbor (other than itself). The choice of how we compute distance is often an important consideration in nearest neighbor clustering; for simplicity, we will use the standard Euclidean distance in this assignment, where the distance between points x_i and x_j is given by

$$\sqrt{\sum_k [x_i(k) - x_j(k)]^2}.$$

If we use k-nearest neighbor classification (discussed in lecture), we will guess that the label of a data point should be a weighted average of the labels of its k nearest points (not including itself, again, of course). Mathematically, a weighted average of values $v_1 \ldots v_n$ using weights $w_1 \ldots w_n$ is given by

$$\frac{\sum_i w_i v_i}{\sum_i w_i}.$$

When computing this weighted average, we should assign higher weight to closer neighbors. For example, we can set the weight of a neighbor to $e^{-\alpha d}$, where d is the distance to the neighbor, and α is a parameter of our choosing, selected to make our final results as good as possible.

4 Running and Validating your Code

Your program should take two parameters on the command line: the name of an input file (e.g., `wine.txt`, and a value of k (up to 10). It should then perform k-nearest neighbor classification using a kd-tree on each of the n data points, and it should print out the average squared error of the final classification. That is, if a_i is the actual label of data point i, and g_i is our guess at the label of data point i, then you should output

$$\frac{1}{n}\sum_i (a_i - g_i)^2.$$

To assess whether this is a good result, you may wish to compare it to what you would get if you simply guessed randomly – here, the optimal choice that minimizes the squared output error is to guess that each g_i is just equal to the average of the a_i's. Does your classifier have a smaller error than this? Does the value of k have a large impact on this error? You do not need to include answers to these questions in your submission, but you will certainly be interested in knowing the answers to these questions in any case.

Note that the wine data set is sufficiently small that you should be able to check that your code is working properly by finding nearest neighbors using the kd-tree and also finding the same nearest neighbors by “brute force”, simply by sorting all the other points according to distance. This can be very helpful for debugging your code.

5 Submission and Grading

Please name your submitted file `wine.cpp`.

Your final grade will be out of 20 points, as with the previous homework assignments. Approximately 15 points will be given for correctness and efficiency, with roughly 5 points given for clarity of code. Programs that do not compile on the lab machines will receive zero points, so please be sure to check that your code compiles properly.

Final submissions are due by 11:59pm on the evening of Friday, November 21. No late submissions will be accepted.

References

[1] P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis, Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems 47(4):547-553, 2009