Letter Classification

Two Tree-Based Approaches to Handwriting Recognition

Prepared for ST599 - Big Data Analysis by:
Andrew Bernath
Heather Kitada
Ethan Edwards

Due date: 6/2/2014

Project and Data Background

From note-taking software to automated check processing, handwriting recognition has long been a question of great interest. As early as 1915 (41 full years before ENIAC), a patent was awarded to Hyman Eli Goldberg for his Controller, a primitive handwriting recognition interface[1]. There are a number of ways to approach this problem, primarily depending on the data you believe you will be working with; however, provided that you are working within the context of a known alphabet, handwriting recognition can most easily be thought of as a predictive, multiclass classification problem. For instance, in the case of the Roman alphabet's capital letters, there are twenty-six distinct classes, and the desired outcome is to accurately sort any new observations into one of these classes.

To better understand the work done in this field, a dataset of 20000 randomly distorted letters from 20 different fonts was obtained from the UCI Machine Learning Repository[2]. The dataset was then randomly partitioned into a learning set of 13333 observations and a cross-validation set of 6667 observations. Given the problem's nature, classification trees were a natural approach, and thus two separate methods were devised: *bagging of classification trees* (BCT) and a *logistic regression binary search tree* (LRBST).

Methods Used

The BCT method relies on two separate methods within machine learning: bagging and classification trees. First, a classification tree is constructed by the CART algorithm. The Gini impurity index (GII) – a measure of how often a randomly chosen element within the node would be misidentified if randomly labeled according to the class distribution of the node – is calculated, and from there several splits are tested for each covariate to determine what split will minimize the GII of the following nodes. This process is repeated until the GII of all end nodes is sufficiently low; ideally, this will result in twenty-six distinct nodes (one for each classification level). In our case, they did not; the first tree constructed only had nineteen distinct classes, and thus had a poor prediction rate (~47.6%). Bagging (bootstrap aggregation) was introduced to remedy this: fifty classification trees were constructed in this manner and subsequently aggregated through R's built-in functions. This resulted in the final method, where each observation from the cross-validation set was ran through all fifty trees, given a vote for a specific class from each tree, and assigned to the class with the most votes.

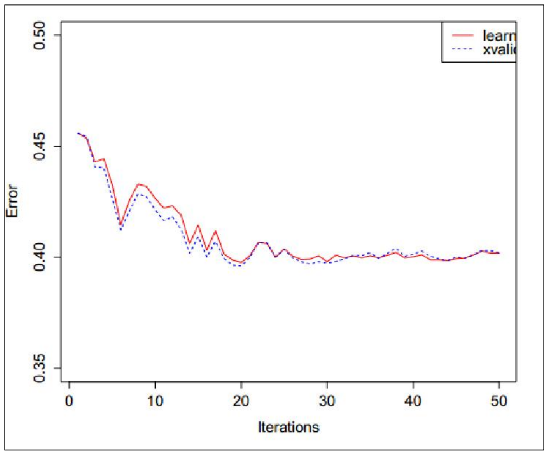

Figure 1: Bagging Error vs. Iterations (BCT)

The LRBST method similarly relies on classification trees. However, its construction method is significantly different, as is its method of choosing which path to traverse. Summary statistics are obtained for each class through R's dplyr package, and the Euclidean distances between each class's summaries are then used in a complete hierarchical clustering algorithm to create a

1 Goldberg, Hyman E.. "Controller." Robert P. Lamont, assignee. Patent US 1165663 A. 28 Dec. 1915. Print.

2 http://archive.ics.uci.edu/ml/datasets/Letter+Recognition

dendrogram. From here, a logistic regression model is created using the training set for each node in the dendrogram, with 0 corresponding to the observation's true class being down the left path and 1 corresponding to it being down the right path. Any new observation (drawn from the cross-validation set) is then fed into the first node's regression; if the sample probability is greater than 0.5 the right path is taken, and if the sample probability is less than 0.5 the left path is taken. This decision process is repeated for each node it arrives at until it reaches a terminal node, and at that point a class is assigned.

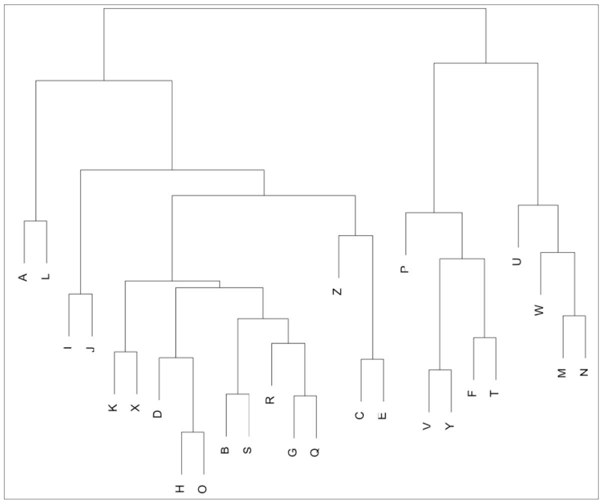

Figure 2: Branching Dendrogram (LRBST)

Findings

The BCT method resulted in a 61.3% identification rate; while this is certainly a substantial improvement over guessing randomly (~3.8% identification rate) and even the original single-CART model, there are still some flaws to the process. In particular, the algorithm had difficult processing the letters F, H, O, R, and S (with less than 50% identification rate for each – minimum 22% for S, compared to maximum 82% for V).

Meanwhile, the LRBST method proved to be more accurate than the BCT method, with a 74.8% identification rate (faltering only at the letter H). However, this came at a steep cost: where the BCT method took roughly fifteen minutes to run, creating and running the LRBST took almost eight hours.

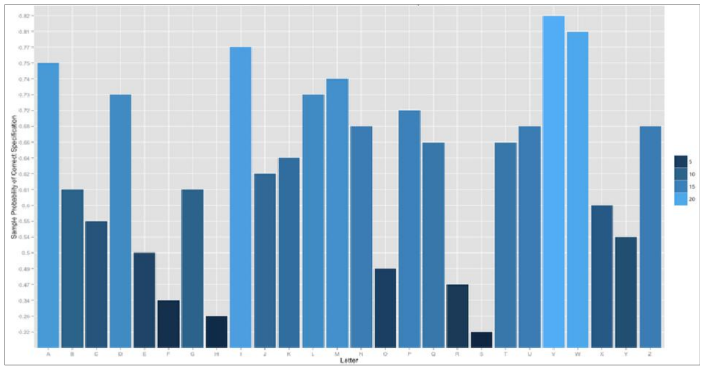

Figure 3: Classification Rates (BST)

Assumptions, Limitations and Scalability

Each method clearly has its pros and cons; the LRBST method provides greater accuracy at significant computational cost, while the BCT method has lower accuracy in exchange for running much quicker. However, there are several ways to address these issues. For the LRBST method, optimizing the code and searching for pre-existing functions to replace loops in the code may make a significant difference. As for the BCT method, the inability of the algorithm to split based on more than one covariate might be a major point to address. Doing so would add computational complexity, of course, but an acceptable trade-off should be discoverable.

In terms of scalability, it would seem that the BCT as written would scale much better for larger datasets. As has been mentioned, however, part of this may be due to the LRBST being built entirely by hand; with proper code optimization, it is entirely possible that both would handle larger datasets on similar levels.