PROJECT 3
CLASSIFICATION AND REGRESSION TREES

DAVID ATLAS

Abstract. This paper covers decision tree induction, a non-parametric set of models designed to partition a feature space. The Iterative Dichotomiser 3 (ID3) algorithm for classification will be introduced. The more general Classification and Regression Tree (CART) algorithm will be introduced as well. The ID3 algorithm will be applied to 3 classification problems. Post-fit pruning will be used to help prevent overfitting. The CART algorithm will be applied to 3 regression problems. A validation set will be used to tune an early stopping parameter to prevent overfitting. It is found that the tree-based algorithms outperform baseline metrics.

1. Problem Statement & Hypothesis

Decision trees are a non-parametric supervised learning technique. The general class of models employ greedy searches to best partition the feature space so as to explain the target variable. As such, they would be expected to excel when the relationship between the features and the target holds locally, or when the interaction of multiple features drives the target (although there are some difficulties with this in practice). Decision trees also have an advantage when many, potentially irrelevant or correlated features are included, as the correlation shouldn't hurt the fitting procedure, and irrelevant features are generally ignored. In contrast to some of the earlier non-parametric density estimation techniques introduced in this course, decision trees should therefore perform adequately in very high dimensional spaces, although the fitting procedure will become more computationally expensive.

However, this is all with one major caveat - find the globally optimal tree is an NP-complete problem, and therefore not easily solved in a reasonable amount of time. As a result, the tree can be tripped up by many patterns. For example, a U-Shaped continuous valued relationship would be hard for most decision tree algorithms to learn. The interaction between features may never be uncovered if there is no relationship between the individual features and the target, as the individual splits may not be particularly good at any step. Therefore, it would be expected that the decision tree algorithms will perform well on many problems, but are certainly susceptible to patterns that they cannot learn.

Another consideration with respect to the decision tree is that when there is not much data available, several problems are introduced.

(1) Decision trees are prone to overfitting, as without any explicit treatment for overfitting, each datapoint might end up with its own leaf node, partitioning the feature space into very small partitions. As a result, a validation set is generally necessary to inform the decision to stop adding nodes (or which nodes to remove in the case of pruning).
(2) If there are not enough datapoints to fill up relevant parts of the feature space, some of the partitions may have only a few (or even just 1) datapoint. This may lead to poor performance on prediction.
(3) There may be future datapoints that do not follow any path in the decision tree - for example, a multivalued discrete feature may have a node where none of the datapoints take on one of the values. This presents an estimation problem that must be handled.

2. Description of Algorithms

2.1. **Iterative Dichotomizer 3 (ID3).**

The Iterative Dichotomizer 3 (ID3)[**?**] algorithm is a non-parametric decision supervised learning algorithm, originally proposed for classification problems.

Given a set of features $X = (x_1, \cdots, x_k)$, where x_i is a vector of n observations of feature i, and a target variable y, composed of the class labels for the n observations, the algorithm proceeds as follows:

For each feature i, find the optimal splitting points. If feature i is a discrete valued feature, there is only one split to consider - each of the values gets its own split. If feature i is a continuous valued split, all possible split points must be considered. The definition of split optimality will be discussed below.

Suppose feature j is a discrete-valued feature, chosen as being the optimal split. X and y are then partitioned into rows in which $x_j = c$, for each of the c possible values of x_j. If j is a continuous-valued feature, X and y are then partitioned into rows where $x_j > c$, where c is the optimal split point.

After the data is partitioned, the process is repeated on each partition. This proceeds recursively until all of the y observations in a given leaf node are of the same class, the y observations cannot be further split, or until an early stopping threshold is triggered.

For continuous-valued features, the optimal splitting point must be found. To do this, the feature x_j is first sorted. The unique midpoints between the rows of the feature are calculated. Each midpoint is tested as a possible split point, with the exception of midpoints between two points with the same value of y. This is because such points cannot be the optimal split points, as moving the edge points to the other partition would make the split more optimal. After all of the midpoints are tested, the most optimal one is selected.

2.1.1. *Entropy, Maximal Gain and Gain Ratio.* To determine the most optimal split point, a metric must be defined. Given a classification problem, the entropy (or impurity) of a given set of y values is defined as

$$I(y) = -\sum_{i=1}^{C} \frac{C_i}{n} \log \frac{C_i}{n}. \tag{1}$$

In Equation 1, C is the number of unique values in y, and C_i is the number of points in y that are of class i. n is the number of points in y. The entropy of a set of values y will be largest when each class has the same number of elements in y. As such, choosing the split that minimizes the entropy will result in choosing the split that best seperates the classes in y at each step.

The expected gain of a split f_i is defined as

$$\mathrm{E}(f_i) = \sum_{j=1}^{m_i} \frac{C^j_{\pi,1} + \cdots + C^j_{\pi,k}}{C_{\pi,i} + \cdots + C_{\pi,k}} I(C^j_{\pi,1}, \cdot, C^j_{\pi,k}) \tag{2}$$

In Equation 2, the sum over the partitions is taken (m_i is the number of partitions created by split f_i). For each partition, the ratio of total points that are included in the partition is multiplied by the entropy of the partition. This is a weighted average entropy calculation, where the partitions are weighted by their size.

The intrinsic value of a feature is defined as

$$\mathrm{IV}(f_i) = -\sum_{j=1}^{m_i} \frac{C_{i,1} + \cdots C_{i,k}}{C_1 + \cdots C_k} \log \frac{C_{i,1} + \cdots C_{i,k}}{C_1 + \cdots C_k} \tag{3}$$

In Equation 3, m_i is the number of partitions created by splitting on f_i, and $C_{i,k}$ is the number of points in class C_k in partition i. The intrinsic value calculation penalizes splits with many partitions. The motivation is to discourage multivalued discrete splits that partition the data effectively as a result of creating many partitions. This is likely to result in overfitting, and a model that will not generalize effectively to out-of-sample data. The most extreme example of this phenomenon is an ID number for each of the instances - a single split would perfectly partition all of the datapoints, but would not be very useful going forward.

Finally, to select the optimal split f_i^*, the split with the largest **gain ratio** is used. This is simply defined as

$$f_i^* = \arg\max_{f_i} \frac{\mathrm{E}(f_i)}{\mathrm{IV}(f_i)}.$$

To summarise the algorithm, the data is recursively partitioned, choosing at each step the partition that maximizes the gain ratio with respect to the target. Note that the algorithm is greedy, in so much that it chooses the best split at each point. There is no guarentee that the tree found is the globally optimal tree.

2.1.2. *Pruning.* After a tree has been grown to completion, one issue remains. It is not unlikely that the tree is overfit, as recursive partitioning (ideally) results in pure nodes, each of which may correspond to a very small partition in the feature space. As a result, it would be desirable to reduce nodes that do not help fit to out-of-sample data.

This process is called post-pruning. In post-pruning, a validation set (seperate from the train/test sets), is split off from the data. The tree is fit to completion on the training set. Then, beginning with the root node, the validation dataset is applied to the node and all of its children. If any of the children subtrees provide superior performance relative to the parent node, the parent node is swapped for the best performing child, and all the other children are disregarded. If none of the subtrees improve performance, no updates are made. At this point, the process is applied recursively, with each of the children nodes acting as the parent node.

This creates an effect where splits that do not account for a reduction in entropy in the validation set are recursively removed from the tree. This can also be thought of as a backtracking mechanism, as the greedy nature of the fitting algorithm may miss signal on the forward pass, and the post-pruning allows the tree to move some of its nodes for more optimal performance.