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.

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). This could be problematic with small datasets.
(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 if the datapoint is an outlier.
(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)[7] 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 guarantee that the tree found is the globally optimal tree. This relates to some of the points above; if the individual features have no effect on a target, but their interaction does, it may be difficult for the algorithm to find that pattern.

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[1]. 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. The performance on the validation set is calculated over the entire tree. Beginning with the root node, each of the children is (individually) collapsed from a subtree to a leaf node. Prediction is made over the entire tree, with one node truncated into a leaf. If performance on the validation set improves relative to the original tree, the leaf remains. Otherwise, it is swapped back to the original node. This process continues recursively over all the subtrees, and continues iteratively from the root node until no changes are made over the entire tree.

This creates an effect where subtrees that do not account for a reduction in entropy in the validation set are removed from the tree. This can also be thought of as a backtracking mechanism, as the greedy nature of the fitting algorithm may a pattern on the forward pass, and the post-pruning allows the tree to re-evaluate the need for nodes on a validation set.

2.2. **CART.** The next algorithm that will be discussed is the Classification and Regression Tree algorithm[4]. This is quite similar to the ID3 algorithm discussed above. However, the algorithm can easily be adapted for a regression problem by replacing the entropy expression used above with

$$MSE(y) = \frac{1}{n}\sum_{i=1}^{n}(y - \bar{y})^2,$$

where $\bar{y}$ is the mean of y. Note that this will lead the formulation above to minimize the variance of target variables in each node, which is analagous to minimizing the entropy within the nodes.

Commonly, the CART algorithm is implemented with all binary splits. In the implementation provided here, that is not explicitly encoded, but the discrete multi-valued attributes will be encoded as one-hot encodings, and treated as though they were continuous splits (although the only split point to consider will be .5).

Otherwise, the algorithm is the same as above.

2.2.1. *Early Stopping.* Like the ID3 algorithm above, a CART is prone to overfitting, as the recursive fitting procedure will continue until no more splits will reduce the training set MSE. As above, a mechanism is needed to prevent the overfitting. A post-pruning procedure could be applied to a CART, but in the implementation provided here, an early stopping parameter will prevent the tree from overfitting. This is sometimes known as pre-pruning[1]. Given an early-stopping parameter θ, fitting will proceed until either no further splits can be made, or until the gain associated with a split is less than θ. The larger θ is, the shorter the tree will be before it stops growing.

In the experiments below, a holdout validation set is used to determine the best value of θ.

3. Experimental Approach

In the classification experiments run below, both continuous and discrete (even multivalued) features are handled by the algorithm, and so no special encoding is needed there. The post-pruning process used on the trees leverages 10% of the dataset which is split off at the beginning of the training process as a validation set. A 5-Fold stratified cross-validation approach is used to estimate the out-of-sample performance on various subsets of the data.

In the regression experiments run below, continuous values are not transformed, but discrete values are mapped into one-hot encodings. This ensures binary splits at each step. For each experiment, the early stopping threshold is tuned on a 10% validation set. A set of values below the variance of the entire target vector is tried, as this lines up with the expected scale and order of magnitude of the correct parameter. A 5-Fold cross validation appproach is used to estimate the out-of-sample performance on various subsets of the data.

4. Experiment Results

4.1. **Abalone Rings Classification.** In the Abalone dataset[8], the number of rings in an abalone is predicted using various physical attributes of the abalone. In general, counting the rings (necessary for finding the age of the abalone) is a time-consuming, costly exercise, and so predicting it based on easy-to-measure attributes is of value.

The results can be found in Table 4.1:

Fold	Accuracy
1	25.5%
2	25.9%
3	25.3%
4	24.7%
5	23.6%

Table 1. The classification accuracy across the 5-Fold cross-validation for the Abalone Ring prediction problem.

4.2. **Car Data Experiment.** In the car data experiment, the suitablility of a car is predicted given various features about the car. The results of the classification experiment can be found in Table 4.2.

Fold	Accuracy
1	90.6%
2	91.3%
3	95.2%
4	95.8%
5	93.9%

Table 2. The classification accuracy across the 5-Fold cross-validation for the Car Suitability prediction problem.

4.3. **Image Segmentation Experiment.** In the image segmentation experiment[3], the portion of the image that a given pixel came from is classified based on characteristics of the pixel. The results are shown in Table 4.3

Fold	Accuracy
1	91.4%
2	86.1%
3	81.1%
4	87.5%
5	90.2%

Table 3. The classification accuracy across the 5-Fold cross-validation for the Image Segmentation prediction problem.

4.4. **Wine Quality Experiment.** In the Wine Quality dataset[2], the quality score of a wine is predicted using attributes about the wine. There are two relevant datasets - one containing red wines, and one containing white wines. The two are combined, and a one-hot encoding is used to indicate whether a given bottle is white or red. The results can be found in Table 4.4. The early stopping value chosen for the decrease in MSE is .2.

Fold	MSE
1	.65
2	.66
3	.64
4	.63
5	.64

TABLE 4. The regression MSE across the 5-Fold cross-validation for the Wine Quality prediction problem.

4.5. **CPU Performance Data.** In the CPU Performance experiment[6], the relative performance ranking of a CPU is predicted with attributes of the CPU. This is a regression problem. The results can be found in Table 4.5. The early stopping value for the decrease in MSE is 14000.

Fold	MSE
1	1944
2	5395
3	9405
4	6873
5	6621

TABLE 5. The regression MSE across the 5-Fold cross-validation for the CPU Performance prediction problem.

4.6. **Forest Fires Data.** The Forest Fires Area dataset[5] contains a regression problem in which the area of a forest fire is predicted using information about the fire. In this experiment, the month and day of the fire are both one-hot encoded as variables in the experiment.

A validation set is used to determine a value for early stopping based on the decrease in MSE for a split. This value is 1875. The results of the experiments can be found in Table 4.6. As in previous assignments, there is a lot of variance between folds, as the prediction problem contains some really large outliers that materially impact performance.

Fold	MSE
1	14840
2	701
3	387
4	5975
5	632

TABLE 6. The regression MSE across the 5-Fold cross-validation for the Forest Fire prediction problem.

5. Algorithm Behavior and Conclusions

The non-parametric form of the decision tree means that it performs reasonably well on the problems in the paper. Its largest structural bias is the preference bias that makes greedy splits. This means that the tree may miss interaction effects if the effects are, in isolation, not present.

Obviously, the regularization applied in this paper is of large importance. In the given experiments, trees performed very poorly on out-of-sample datasets before being regularized, which is not surprising.

The decision tree performs well in situations in which there may be many irrelevant features included in training. This is in contrast to previous assignments, when KNNs are used for the forest fire dataset, performance degrades as only a small subset of the features are relevant. The wine quality dataset also has many features, and the decision tree is able to find a subset that matter.

Given that the regression problems presented above use mean-prediction over the values in each leaf, problems with severe outliers, or non-normally distributed data, may suffer. For example, the rank target

of the CPU performance data and the heavily skewed Forest Fires area data both have such qualities. This manifests as large variance across cross-validation sets, as each test set may contain extreme values (and perform poorly) or not (and perform well).

Another note about the performance of the algorithm relates to the speed with which the decision trees fit. Given a set of discrete attributes, the tree is able to fit very quickly, as each split involves applying one split to each feature. However, for the continuous features in the dataset, many candidate splits must be applied.

6. Summary

In summary, the univariate decision tree is a non-parametric, greedy search algorithm that partitions the feature space into hyper-rectangles, with splits parallel to each feature axis. This results in good performance - non-linear patterns can be observed, and the interaction between features can be learned. The tree is also relatively interpretable, because each branch can be treated as a sequence of "IF-THEN" rules.

The decision tree performs well on the problems in this paper, both for regression and classification. It is able to pick relevant features from a large set of potentially irrelevant ones.

Regularization is key. Early stopping (pre-pruning) or a post-fit reconsideration of the nodes (post-pruning) can be used to improve performance via a validation holdout-set.

The decision tree is a reasonable choice for problems when little is known about the form of the feature space, and a flexible form is desired.

References

[1] Ethem Alpaydin. Introduction to machine learning. pages 222–223, 2014.
[2] Paulo Cortex. Wine quality dataset. URL: `https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/`.
[3] Vision Group. Image segmentation data. URL: `https://archive.ics.uci.edu/ml/machine-learning-databases/image/segmentation.data`.
[4] R Olshen CJ Stone Leo Breiman, JH Freidman. Classification and regression trees. 1984.
[5] Anibal Morais Paulo Corez. Forest fires. URL: `https://archive.ics.uci.edu/ml/machine-learning-databases/forest-fires/forestfires.csv`.
[6] Jacob Feldmesser Phillip Ein-Dor. Relative cpu performance data. URL: `https://archive.ics.uci.edu/ml/machine-learning-databases/cpu-performance/machine.data`.
[7] JR Quinlan. Induction of decision trees. *Machine Learning 1*, pages 81–106, 1986.
[8] Sam Waugh. Abalone data set. URL: `https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data`.