Intelligent Systems: Reasoning and Recognition

James L. Crowley

MoSIG M1 Winter Semester 2020-2021
Lessons 15 30 March 2021

Dichotomizers, CART and Random Forests

Outline

Sources

T. Hastie, R. Tibshirani and J. Friedman, "Elements of Statistical Learning", Springer, 2001

C. M. Bishop, "Pattern Recognition and Machine Learning", Springer Verlag, 2006.

https://en.wikipedia.org/wiki/Decision_tree_learning

Notation

x_d	A feature or attribute. An observed or measured value. For discrete features, the values can be labeled with integers $[1, N_d]$.
$\vec{X}$	A vector of D features.
D	The number of dimensions for the vector $\vec{X}$
S: $\{\vec{X}_m\}$ $\{y_m\}$	A set of training samples (observations) and their indicator variables.
M	The number of training samples.
C_k	The class k
k	Class index (Natural number from 1 to K)
K	Total number of classes
M_k	Number of examples for the class k. $M = \sum_{k=1}^{K} M_k$
$P(C_k)$	The probability distribution for the indicator variables of the training samples. $P(C_k) \equiv P(\vec{X}_m \in C_k) = P(y_m = C_k) = \frac{M_k}{M}$
$H(S)$	The entropy of a set of class labels for a set S of training samples. $H(S) = -\sum_{k=1}^{K} P(C_k)\log_2\left(P(C_k)\right)$
$H(S\|x_d)$	Conditional entropy of dividing a set into D subsets using the values for attribute x_d
$IG(S,x_d)$	Information gain for dividing a set into D subsets using the values for attribute x_d $IG(S,x_d) = H(S) - H(S \mid x_d)$
$I_G(P(C_k))$	The Gini Impurity index for a set S. $I_G(P(C_k)) = \sum_{k=1}^{K} P(C_k)\left(1 - P(C_k)\right)$

Decision Trees

A decision tree is a data structure for estimating a function $\hat{y} \leftarrow f(\vec{X})$. The function may be used to estimate a numerical value for a feature vector (regression), or may be used to estimate the most likely class label $\hat{y} \in [1,k]$ from a set of K possible labels (Classification).

Decision Trees were developed in the 1980s as a form of inductive learning for symbolic reasoning and for exploring the theoretical limits to machine learning. Decision trees are among a fairly small family of machine learning models that are easily interpreted to provide Explainable AI have thus recently received renewed attention. Two families of supervised learning techniques for decision trees were invented at about the same time: Dichotomizers and CART methods.

Dichotomizers, such as ID3, C4.5, C5.0 and their successors, are multi-branch decision trees that that apply a series of multiple-choice questions to determine the most likely class or most likely value for a feature vector. Dichotomizers are typically used with qualitative values. Dichotomizers are most appropriate for classification of feature vectors with symbolic or qualitative values for which the attributes have a small number of, possibly unordered, symbolic values. Dichotomizers may be used with unordered symbolic labels values, such as colors or nationalities.

Classification and Regression Tree (CART) are more appropriate for observations (feature vectors) with numerical values or symbolic values with an intrinsic order. CART methods apply a series of binary predicates (true-false tests) to progressively partition a feature space into rectangular cuboid volumes that are populated with relatively uniform examples from the training set. CART trees are invariant under scaling and various other transformations of feature values.

When used for classification, the resulting cuboids should ideally be populated with training samples that are predominately from a single class. We will usea measure called Gini Impurity to measure this. When used for regression (functional approximation), the cuboid regions should contain examples that permit an easy and accurate functional approximation, such as an average or a linear interpolation.

Both Dichotomizers and CART Trees are robust to missing or irrelevant features, and can be used to construct explanations for automatic reasoning. Both classes of decision tree will determine automatically which of the possible features are the most informative.

Aggregations (large sets) of decision trees, known as **random forests,** have recently been shown to provide classification performance similar to multi-layer perceptrons for many practical problems. Unfortunately, random forests obtain this performance at the cost of **loss of interpretability**, and can not easily be interpreted to provide explanations.

Ensemble techniques, such as random forests, construct more than one decision tree. A random forest classifier is a specific type of bootstrap aggregating tree learning algorithm that uses multiple decision trees constructed by repeatedly resampling training data with replacement. The output is a consensus obtained by voting. This is sometimes called a committee of classifiers.

Boosted learning can be used with random forests to provide arbitrarily good classifiers. Boosted trees incrementally building an ensemble (committee) of classifiers by training each new instance to emphasize the training instances mismodelled by previously constructed trees.

A rotation forest is a tree-learning algorithm that first applies principal component analysis on a random subset of the input features, and then learns decision rules on the principle axes.

Algorithms for constructing decision trees generally work top-down, by choosing a feature or attribute of the training data at each step that best splits the training data into subsets. The feature space is thus divided by a sequence of simple tests, corresponding to the path from the root to a leaf of a tree. While traversing the tree, each node provides a decision that further divides the feature space. Such processes are similar to K-nearest neighbors, and a variant of decision trees, K-D trees, are often used to implement KNN.

Different algorithms use different metrics for measuring "best". These generally measure the homogeneity of the target variable within the subsets. The choice of metric tends to depend on the nature of the features (discrete or continuous) and the desired function (classification or regression).

Some Background from Information Theory

Dichotomisers, CART trees and Random Forests rely on information theory to determine the order in which to test the attributes of an observation. In this section we review some basic concepts from information theory. We will then define the Information Gain for using the values of an attribute to divide a set, and the Gini impurity for the number of different classes in a set.

As in previous lectures, we will assume a labeled training set, S, of M samples of feature vectors (observations) $\{\vec{X}_m\}$, each with D features, along with indicator variables $\{y_m\}$. Each component, x_d of $\{\vec{X}_m\}$ is assumed to have N_d possible discrete values. For simplicity, these may be normalized to the first N natural numbers, $x: [1,N_d]$.

$$\vec{X} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix}$$

In the literature on decision trees, the features of the vector $\vec{X}$ are often referred to as attributes. We will use the term attribute and feature interchangeably.

For classification, every training sample must belong to one and only one class from the set of K classes $\{C_k\}$. The class indexes, k, are taken from the natural numbers in the range [1, K]. In this case, the indicator variable, y_m, represents an integer index, k, for the target class, C_k. In the case of a regression, the target variable, y_m, will be a numerical value.

Entropy and Information Gain

Dichotomizers are most appropriate when the attributes, x_d, can take on a discrete set of N_d possible values. We will identify these values with N_d natural numbers in the range $x_d \in [1,\ N_d]$.

For a set S of M training samples, of which M_k belong to class k, the probability that a sample belongs to class C_k, $P(\vec{X}_m \in C_k)$ is often written simply as $P(C_k)$. As we saw in lecture 3:

$$P(C_k) \equiv P(X_m \in C_k) = P(y_m = k) = \frac{M_k}{M}$$

The tree bar symbol, $\equiv$, indicates definition.

We can use the probability density of class labels to compute the entropy, *H(S)*, of the set S as

$$H(S) = -\sum_{k=1}^{K} P(C_k)\log_2\left(P(C_k)\right)$$

This is also valid for any subset of S. Suppose we use of the N_d values of the attribute (or feature) x_d to divide the set *S* into N_d subsets, with one subset for each of the N_d values of x_d. We will refer to these N_d subsets as S_n. We note that the union of these subsets is S. Let M_n represent the number of samples in the subset S_n.

$$S = \bigcup_{n=1}^{N_d} S_n \quad \text{and} \quad M = \sum_{n=1}^{N_d} M_n$$

The entropy of each subset, $H(S_n)$, computed as explained above for *H(S)*. The probability that any training sample, $\vec{X}_m$, of *S* will be in subset S_n is equivalent to the probability that an attribute x_d of a vector $\vec{X}_m$ from *S* will take on value *n*.

$$P(\vec{X}_m \in S_n) = P(x_d = n) = \frac{M_n}{M}$$

We can use this to define the conditional entropy, $H(S|x_d)$, for using the values of the attribute x_d to divide the set *S* into N_d subsets, S_n:

$$H(S \mid x_d) = \sum_{n=1}^{N_D} P(x_d = n)H(S_n)$$

The information gain of dividing S into N_d subsets using x_d is thus defined as:

$$IG(S, x_d) = H(S) - H(S \mid x_d)$$

ID3 uses the information gain for each feature, x_d, with respect to the current subset of the training data to select the next feature, x_d, to be used to divide the training set into N_d Subsets.

GINI index or Gini Impurity

CART (classification and regression tree) algorithms use the Gini index or Gini impurity to select the order of features for dividing a set S into subsets. This is similar to the Gini coefficient used in economics to measure of the distribution of income across a population, proposed by Corrado Gini in 1912. However, Gini Impurity is not the same as the Gini coefficient used in economics. Gini impurity measures how often an element from a set would be incorrectly labeled if it were labeled using a random value taken from the distribution of labels from the set. The Gini index is differentiable and can thus be used for learning using gradient descent.

As before, we will assume a labeled training set, S, of M samples of feature vectors (observations) $\{\vec{X}_m\}$, each with D features, along with indicator variables $\{y_m\}$, where the indicator variable is an integer index, k, for the target class, C_k, from the natural numbers in the range [1, K]. In the case of CART the attributes may be Discrete or continuous. In the case of a regression tree, the target variables will be a numerical values.

Let $\hat{y}$ be the output from a Discrimination function (Classifier) that maps a D dimensional feature vector, $\vec{X}$ into one of K classes $\{C_k\}$

$$\hat{y} = D(\vec{X})$$

The Gini index captures the effect of basing the predicted class on the probability distribution of target variables from the training set, independent of the actual feature vector. In this case $P(\hat{y}_m) = P(C_k)$.

The Gini index is computed from the probability distribution of the classes.

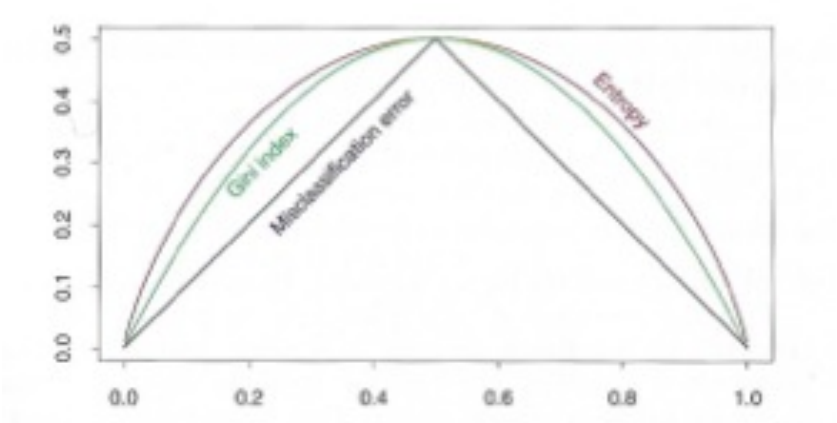
$$I_G(P(C_k)) = \sum_{k=1}^{K} P(C_k)\left(1 - P(C_k)\right)$$

This reaches zero when all cases in the node fall into a single target category. CART methods provide a decision tree that successively divides the training data into uniform classes (or target values) by minimizing the Gini index.

Comparison of Entropy, Gini , and Classification Error

Hastie et al, 2001, contains discusses a comparison of the use of Entropy, Gini Index and Misclassification error (Sum of Squared Errors) as a loss function for learning decision trees. This graph shows the value of Entropy, Gini Index and Misclassification error as a function of the Probability distribution, $P(C_k)$ for a K=2 class decision with class labels (P, N). The entropy, Gini index, and Misclassification error as a function of probability that a sample is Negative would be

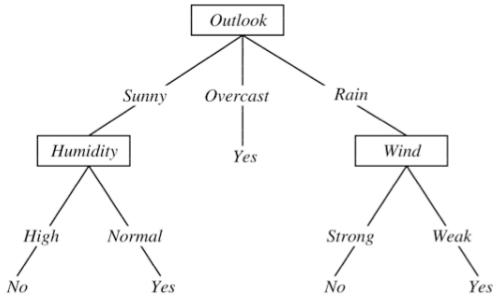
$$P(C_N) \equiv P(X_m \in N) = P(y_m = N)$$



Node Impurity measures for two-class classification, as a function of the proposition p in class 2.
Taken from T. Hastie, R. Tibshirani and J. Friedman, Elements of Statistical Learning, Springer, 2001

We can note that all three have similar values, with all three having maximal values when the data is completely balanced (50/50), zero when the training set is pure (all P or all N).

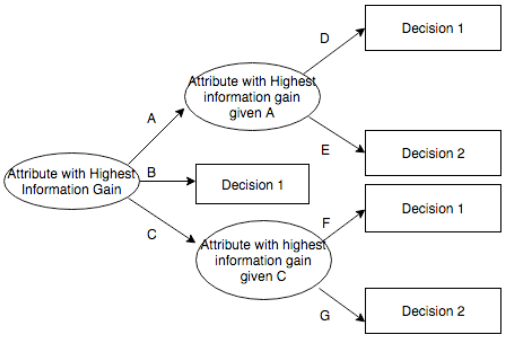
Iterative Dichotomizers



(source unknown)

Dichotomizers are multi-branch decision trees that that apply a series of multiple-choice questions to determine the most likely category (class) for an observation. Dichotomizers are most appropriate for classification of observations (feature vectors) with symbolic or qualitative values, particularly when the attributes have a small number of possibly unordered symbolic values such as colors or nationalities. The ID3 (Iterative Dichotomiser 3) algorithm was proposed by Ross Quinlan in 1979 as a formalization for induction tree learning algorithms.

The ID3 Algorithm



By Acoggins38 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=49453541

The ID3 algorithm uses a top-down greedy approach to build a decision tree. The ID3 algorithm begins with the original set of M training samples $\{\vec{X}_m\}$ with target variables $\{y_m\}$ as the root node.

The algorithm uses the information gain of the individual features from the current training set to compose a tree of multivalued decision functions. As described above, information gain is the reduction in entropy obtained by dividing a set into disjoint subsets based on the values of a particular feature or attribute. Information gain is

calculated by comparing the entropy of the dataset before and after a division into subsets.

On each iteration, the algorithm iterates through the unused features of the current subset, S, and calculates the conditional entropy, $H(S \mid x_d)$, for dividing the set S into N_d subsets, using the values of each feature x_d . The feature, x_d, with the largest information gain is then used to construct a multi-valued test, and the current subset of the data is partitioned into N_d subsets, according to the values of the selected feature. The algorithm continues recursively with each subset, considering only features that have not been previously selected for a test.

Recursion on a subset will stop if one of the following cases occurs:

1) Every element in the subset has the same indicator variable (or target value); in this case, the node is turned into a leaf node and labeled with the class given by the indicator variable.
2) There are no more attributes to be selected. In this case, the node is made a leaf node and labeled with the most frequent indicator variables of the samples in the subset.
3) There are no training samples in the subset, which can happen when no sample in the subset set was found to match a specific value of the selected attribute. An example could be the absence of a person among the population with from a specific country. In this case, a leaf node is created and labeled with the most common class of the examples in the parent node's set.

Throughout the algorithm, the decision tree is constructed with each non-terminal node (internal node) representing the selected attribute on which the data was split, and terminal nodes (leaf nodes) representing the class label of the final subset of this branch.

The resulting path through the tree can interpreted with a natural language explanation for the decision. An example for such an explanation would be:

<X> is likely to be Danish because he is tall, has blond hair and blue eyes.

The ID3 algorithm will produce errors when there are examples in the training data from different target classes that have identical feature vectors. When training on identical feature vectors with different target values, the algorithm will select the most likely target vector as a response.

Summary of ID3:
1) Calculate the entropy for each attribute x_d of with respect to the current subset of training data.
2) Partition ("split") the current set into N_d subsets using the attribute for which the information gain is greatest.
3) Create a decision tree node using the selected attribute as a test with one branch for each possible value of the attribute.
4) Recursively repeat on the above steps using the remaining attributes for each subset.

ID3 uses a greedy strategy by selecting the locally best attribute to split the dataset. This does not guarantee an optimal solution and can converge to local optima. The algorithm's optimality can be improved by using backtracking during the search for the optimal decision tree at the cost of possibly taking longer.

ID3 will generally over-fit the training data. To avoid over-fitting, smaller decision trees should be preferred over larger ones leading to the idea of an ensemble of trees (a forest).

In order to use ID3 with continuous features it is necessary to partition the features into a small set of bins such that the partition provides best information gain. Determining the "best" split is generally expensive procedure for which there is no established method.

Improved ID3: The C4.5 algorithm

The C4.5 algorithm is an extension of ID3. C4.5 builds decision trees from a set of training data in the same way as ID3, using the information gain to select the most informative features to test. However, C4.5 improves on ID3 in the following manner:
- C4.5 can accommodate continuous features by creating a threshold that defines a predicate for dividing the training set.
- C4.5 allows can this accommodate features with missing values. Missing attribute values are simply not used in the entropy calculations or for defining tests.
- C4.5 can use the cost of computing an attribute to prefer less expensive attributes, for use in on-line data discovery techniques.
- Pruning: Once the tree has been created, C4.5 goes back through the tree and attempts to remove branches that do not help by replacing them with leaf nodes.

As with ID3, C4.5 chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The splitting criterion is the normalized information gain (difference in entropy). The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm is then recalled recursively on the partitioned subsets of training data.

As with ID3, C4.5 has a few base cases.

- If all the samples in the list belong to the same class, C4.4 creates a leaf node for the decision tree saying to choose that class.
- If none of the features provide any information gain, C4.5 creates a decision node higher up the tree using the expected value of the class.
- If instances of a previously-unseen class are encountered, C4.5 creates a decision node higher up the tree using the expected value.

Classification and Regression Trees (CART)

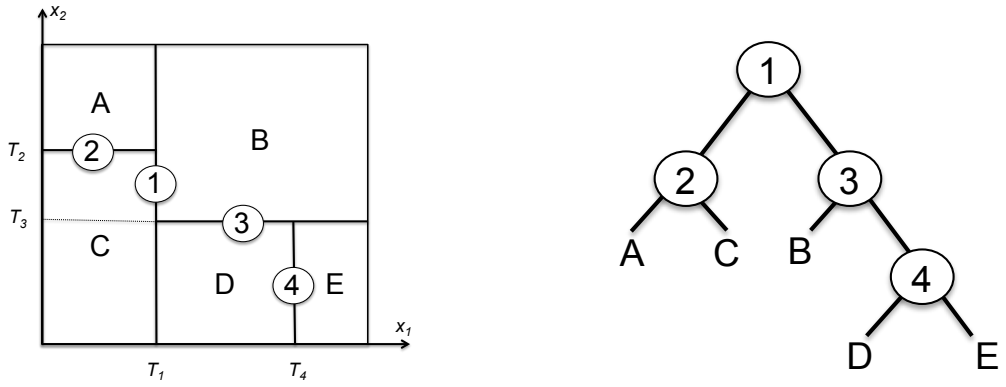
The Classification and Regression Tree methodology, also known as the CART, was proposed in 1984 by Leo Breiman et al. CART methods are commonly used for regression as well as classification, and operate on features with continuous values over an infinite range.

When used for classification, CART produces a decision tree that transforms a feature vector, $\vec{X}$, into a target label $\hat{y}$ from a finite set of K discrete target classes, $\{C_k\}$. CART can also be used to estimate (or learn) a function that estimates a numerical value (regression) from an observation. When used for regression the tree acts as a function $\hat{y} = f(\vec{X})$, that maps a feature vector, X into an estimated numerical value ($\hat{y} \in R$).

CART constructs a **binary tree**, where each node is a binary predicate that makes a yes/no (or T/F) decision by applying a threshold to one of the attributes. This requires choosing the best feature and determining the best threshold, and can be performed by exhaustively testing the features and computing the best threshold for the current subset. This works much like the bias in an ROC curve, and "best" can be minimum classification error or can depend on the constraints TN and TP imposed by the problem.

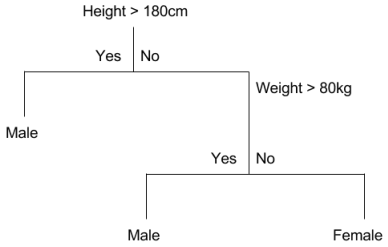
The predicate splits the available set of the training data into two subsets. These subsets are then passed to the next level of the tree where they are further divided until a target "impurity" criteria is met for the remaining subset. When the resulting subset is sufficiently pure, a leaf node is created that returns the most likely target variable from the subset.

When the feature values (attributes) are real numbers, then the tree of binary decisions can be seen as successively partitioning the D dimensional feature space into cuboid volumes using thresholds. In the case of D=2 features, this can be easily seen as dividing a plane into ever smaller rectangular regions.



For example, in the figure shown above node 1 would compare feature x_1 with the threshold T_1. IF $x_1 < T_1$ then it would apply node 2 else it would apply node 3, etc. For classification, the most likely class from each region would be returned. For regression, the average value of the target variables can be returned or some other form of interpolation can be used.

The resulting tree can be interpreted as a list of rules. For example:



If Height > 180 cm Then Male
If Height <= 180 cm AND Weight > 80 kg Then Male
If Height <= 180 cm AND Weight <= 80 kg Then Female

The tree can also be interpreted with an explanation such as

The gender subject <X> is most likely male because the height is > 180cm.

Classifying observations CART Models

As with Dichotomizers, learning a CART tree involves choosing the best features to divide the data. Exhaustive search can be prohibitively expensive. The search is typically performed using a recursive procedure which scans a subset of the data to determine the best features and best split point (threshold) for each attribute. Unlike ID3, CART uses a cost (or loss) function to determine which feature and threshold value to use to split the data into two subsets.

For regression problems, where the objective is to approximate a numerical estimate, the the sum of squared errors can be used as a loss function. For a CART tree, *f()*, the loss for a training set S of M samples would be:

$$L(f(-)\mid S) = \sum_{m=1}^{M} (f(\vec{X}_m) - y_m)^2$$

For classification, CART uses the most likely indicator variable from a subset to label samples from the subset and uses the GINI index to choose the attribute to be used to progressively divide the training set into subset. Given a subset S of M samples

$$L(f(-)\mid S) = I_G(P(\hat{C}_k)) = \sum_{m=1}^{M} P(\hat{C}_k)\left(1 - P(\hat{C}_k)\right)$$

where $\hat{C}_k = f(\vec{X}_m)$ is the predicted class provided by the decision tree.

As discussed above, for a set *S* composed of *M* samples of which M_k samples belonging to each of *K* classes, the probability distribution of class labels is $P(C_k)$, and the GINI index for the subset S under consideration by the current node of the tree is

$$I_G(P(C_k)\mid S) = \sum_{k=1}^{K} P(C_k)\left(1 - P(C_k)\right)$$

The CART learning Algorithm

Classification and Regression Tree learning is a form of greedy recursive splitting of the training data. Given a set, S, composed of M training samples, $\{\vec{X}_m\}$ with $\{y_m\}$ indicator (or target) variables.

Given a subset S of the training data

For each feature x_d, in $\{\vec{X}_m\}$, from d=1 to D,

1. Test the stopping Condition. If the stopping condition is not met then,
2. Determine a threshold value for x_d that minimizes a cost function for two subsets, S_1, and S_2 ($S_1 \cup S_2 = S$).
3. Select the feature x_d with the lowest Cost, and use the threshold value to define a the test for the node. Then divide the training set S into two subset and call the algorithm recursively with each of the subsets.

The most common stopping procedure is to use a minimum count on the number of training samples in the remining subset. If the number of samples is less than some minimum then the split is not accepted and the node is taken as a leaf node.

The stopping criteria is tuned to the problem and dataset, with typical values ranging from 5 or 10, as we saw with histograms.

This minimum number of samples per leaf node defines how specific to the training data the tree should be. If the number is too small, then the tree is too specific, and the model will over-fit the training data and likely have poor performance on the test set.

For classification, an alternative stopping condition for a classification tree is when the Gini index is 0 (all of the samples are from the same class). For a regression tree, and equivalent condition would be that the variance if of the target values are zero (or very small).

Random Forests

Random forests are an ensemble learning method for classification, regression and other tasks. Random forest learning operates by constructing a large number of decision trees, and correct for the tendency of decision trees to over-fit to the training set. Random forests generally outperform decision trees, and are frequently used as blackbox models for data-mining in businesses, as they can be used to generate reasonable predictions across a wide range of data while requiring little configuration (and minimal understanding).

Deep decision trees tend to learn highly irregular patterns: they over-fit their training sets. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the over fit. This comes at the loss of interpretability, but generally greatly boosts the performance in the final model.

Assume a training set $\{\vec{X}_m\}$ of M samples with indicator variables $\{y_m\}$ where the indicator variables are coded as the integer indices for K target classes. Bagging repeatedly (B times) selects a random sample $\{\vec{X}_s\}$ of the training samples with indicator variables $\{y_s\}$ from the training set and fits trees to these S samples.

$$f_b(\vec{X}) \Leftarrow DT(\{\vec{X}_s\},\{y_s\})$$

After training, the predicted class for a new observation, $\vec{X}$, can be determined by voting over the B trees:

Allocate a table h(k) of K cells initially 0.

$$\forall_b : h(f_b(\vec{X})) \leftarrow h(f_b(\vec{X})) + 1$$

$$\hat{y} = \arg\!-\!\max_k \{h(k)\}$$

A probability distribution table for the K classes can be obtained using soft-max over the *h(k)* predictions.

In the case of a regression forest, the predicted estimated value, $\hat{y}$, is determined averaging the predictions from all the individual regression trees on the observation $\vec{X}$.

$$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} f_b(\vec{X})$$