Decision Trees and Random Forests

Tom Rainforth
3YP Saving Oneself

23/01/17
Overview

- Motivation
- Supervised Learning
- Prediction using a Decision Tree
- Learning a Decision Tree
- Tree Ensembles and Random Forests
- Practicalities
- Extensions
Motivation
Motivation

- Extremely fast
  - Training in $O(NL\lambda \log(N))^2$)
  - Prediction in $O(L\log(N))$
- Trivial to use - most packages require only the data itself
- Ambivalent to data type - continuous, categorical etc
- Exceptional “out-of-the-box” performance
10 out of top 20 classifiers from recent survey of 180 classifiers on 82 datasets are based on ensembles of decision trees.

[Rainforth & Wood 2015]
[Fernandez-Delgado et al 2014]
Used for literally everything
Trivial to Use

Size of forest (bigger the better)

RF = TreeBagger(200,XTrain,YTrain); % Training
preds = predict(RF,XTest); % Predict
Supervised Learning
Supervised Learning

Supervised learning setting

<table>
<thead>
<tr>
<th>Samples</th>
<th>feature 1</th>
<th>feature 2</th>
<th>...</th>
<th>feature M</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>3.1</td>
<td>1.3</td>
<td></td>
<td>0.9</td>
</tr>
<tr>
<td>S₂</td>
<td>3.7</td>
<td>1.0</td>
<td></td>
<td>1.3</td>
</tr>
<tr>
<td>S₃</td>
<td>2.9</td>
<td>2.6</td>
<td></td>
<td>0.6</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sₙ</td>
<td>1.7</td>
<td>2.0</td>
<td></td>
<td>0.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>type 1</td>
</tr>
<tr>
<td>type 2</td>
</tr>
<tr>
<td>type 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>type 5</td>
</tr>
</tbody>
</table>

\[ y = f(X) \]

\( f \): mapping \hspace{2cm} X: Design matrix \hspace{2cm} y: outcome

[Tsanas 2015]
Linear Regression

\[ f(x) = mx + c \]
Prediction using Decision Trees
Decision Trees

• Predictive models that impose sequential divisions of an input space
• This assigns points to “leafs”
• Prediction based on local leaf model
• Only need to consider binary trees (i.e. nodes split into two children) as can rearrange to multiple splits
Decision Trees - Prediction

- **is sex male?**
  - yes
  - **is age > 9.5?**
    - died
      - is sibsp > 2.5?
        - died
          - died
            - 0.17 61%
        - survived
          - 0.73 36%
      - survived
        - 0.89 2%
  - no
    - survived
      - 0.05 2%
Decision Trees - Prediction

Root node

Decision node

Leaf node
Decision Trees - Prediction
Decision Trees - Prediction

The decision tree is composed of nodes and branches, with each node testing a feature of the input data. The branches split the data based on the outcome of the feature test. For example, if $x_1 < -0.71$, the data moves down one branch; if $x_2 < 1.42$, it moves down another. The tree structure helps in making predictions based on the input features $x_1$ and $x_2$. The graphical representation shows how different regions of the input space are classified, with different colors indicating different classes or outcomes.
Decision Trees - Prediction

Decision tree with conditions:
- $x_2 < -3.03$
- $x_1 < -0.71$
- $x_2 < 1.42$
- $x_1 < -1.46$
- $x_1 < 2.81$

Graph showing $X_1$ and $X_2$ with data points.
Decision Trees - Prediction
Learning a Decision Tree
What makes a good split?

[Fig. 3.1: (a) Input data points. The ground-truth label of training points is denoted with different colours. Grey circles indicate unlabelled, previously unseen test data. (b) A binary classification tree. During training a set of labelled training points \{v\} is used to optimize the parameters of the tree. In a classification tree the entropy of the class distributions associated with different nodes decreases (the confidence increases) when going from the root towards the leaves.]

Given a labelled training set learn a general mapping which associates previously unseen test data with their correct classes. The need for a general rule that can be applied to "not-yet-available" test data is typical of inductive tasks. In classification the desired output is of discrete, categorical, unordered type. Consequently, so is the nature of the training labels. In fig. 3.1a data points are denoted with circles, with different colours indicating different training labels. Testing points (not available during training) are indicated in grey.

More formally, during testing we are given an input test data \(v\) and we wish to infer a class label \(c\) such that \(c \in \{c_k\}\), with \(C = \{c_k\}\). More generally we wish to compute the whole distribution \(p(c|v)\).

[As opposed to transductive tasks. The distinction will become clearer later.]

[Criminsini et al 2011]
Shannon Entropy

• Measure of uncertainty

• Expected amount of information conveyed in a message or observation

\[
H[Y] = - \sum_{k=1}^{K} p(y = k) \log_b p(y = k) \quad Y \sim p(y)
\]

• When \( b=2 \), measured in bits

• Higher entropy means that less is known about the outcome, thus corresponds to more evenly distributed probability

• Lower Entropy = More certainty in outcome
Information Gain

• The expected information gain is the reduction in entropy from one state to another

• Thus the information gain from a split is the entropy before the split minus the entropy after the split

• Need to weight the two post split entropies by the size of the respective nodes

\[ IG = H(Y) - \frac{N_{\text{left}}}{N} H(Y_{\text{left}}) - \frac{N_{\text{right}}}{N} H(Y_{\text{right}}) \]

• Calculate the entropies using the empirical distributions, i.e. \( p(y=k) \) is the proportion of class \( k \) at the node.

• Note that \( 0 \cdot \log(0) = 0 \) so empty classes have no effect. This also means that the minimum entropy is 0 and occurs when only a single class
Splitting

The random decision forest model

Fig. 2.3: Information gain for discrete, non-parametric distributions. (a) Dataset $S$ before a split. (b) After a horizontal split. (c) After a vertical split.

Figure 2.3a shows a number of data points on a 2D space. Different colours indicate different classes/groups of points. In fig. 2.3a the distribution over classes is uniform because we have exactly the same number of points in each class. If we split the data horizontally (as shown in fig. 2.3b) this produces two sets of data. Each set is associated with a lower entropy (higher information, peakier histograms). The gain of information achieved by splitting the data is computed as:

$$ I = H(S) - \frac{1}{2} \left( H(S_1) + H(S_2) \right) $$

with the Shannon entropy defined mathematically as:

$$ H(S) = -\sum_{c} p(c) \log(p(c)) $$

In our example a horizontal split does not separate the data well, and yields an information gain of $I = 0$. When using a vertical split (such as the one in fig. 2.3c) we achieve better class separation, corresponding to lower entropy of the two resulting sets and a higher information gain ($I = 0.69$). This simple example shows how we can use information gain to select the split which produces the highest information (or confidence) in the final distributions. This concept is at the basis of the forest training algorithm.

[Criminsini et al 2011]
Splitting a Node - Exhaustive Search

- For each feature
  - Sort data by that feature
  - Calculate information gain resulting in putting the split between consecutive data points
- Choose the split (i.e. feature + position) that gives the highest gain
- Place the split halfway between the two data points
Training - Putting it Together

1. Start with a root node with all the data points
2. Exhaustively search possible splits and choose the best to create two new child nodes
3. Split the data into the new nodes. For each of these go back to 2 and grow in a self similar fashion
4. Continue until all points in a node have the same label or some other criterion is met (e.g. min number of points in a node).
Overfitting

- Best scoring solution perfectly separates the data
- This can clearly assign more structure than really exists
- Always possible to construct a function that fits the training data perfectly but does terribly on the test data
- Need to regularise to reduce the variance it estimates
Pruning

• Collapsing down some nodes to a single leaf node

• Start at the leaves and step upwards deciding whether to collapse based on some metric

• Smaller tree that is less prone to overfitting

• Computational expensive and unreliable
Tree Ensembles
A Better Approach: Ensembles

• Train lots of trees and average the predictions

• Reduces the variance on prediction / less overfitting - not all trees will make the same “mistakes”

• Fundamentally more powerful model - can have more complex decision boundaries

• Massive performance improvements

• Estimation of uncertainty - not all trees will make the same prediction, particularly near the boundaries

• No need to prune - can actually be faster!
Random Rotation Ensembles

Figure 1: Comparison of the decision boundary for the standard random forest algorithm (RF, left column) and the modified version with randomly rotated feature space for each tree (RR-RF, right column) on the binary classification task of chapter 2 of Hastie et al. (2009). The top row illustrates a typical decision boundary for a single tree, while the bottom row depicts a fully grown ensemble comprised of 10000 trees in each case. Ntree is the total number of trees in the forest, mtry the number of randomly selected features considered at each decision node.

(a) RF (ntree=1, mtry=1)  
(b) RR-RF (ntree=1, mtry=1)  
(c) RF (ntree=10000, mtry=1)  
(d) RR-RF (ntree=10000, mtry=1)

[Blaser & Fryzlewicz 2015]
Randomising Trees

- Training process previously described is completely deterministic and thus always produces the same tree.
- Need to impart some randomness to the training process.
- Trade-off between diversity of the ensemble and predictive power of individual trees.
Bootstrap Sampling

• Sample with replacement a dataset of the same size

• Each sample is drawn independently from the full dataset

for n=1:N
    i ~ UniformDiscrete(1:N)
    $x'_n = x_i$
end
Bagging

- Train each tree using a slightly different dataset
- Generate datasets by taking bootstrap sampling - i.e. sample with replacement a dataset of the same size
- Predict by aggregating the predictions of the different trees

[Beatrix Mathews 2016]
Random Subspacing

- At **each** node randomly select some subspace of features to search over
  
  1: Subsample features ids $\delta$ by sampling from \{1, \ldots, $D$\} $\lambda$ times without replacement.  
  2: Set $\mathcal{X} \leftarrow X(:,\delta)$

- This forces different trees to split along different dimensions at each node

- As this changes the partitioning at the first few nodes, this encourages further diversity at the latter nodes

- Typically $\lambda \ll D$
Random Forests

• Use bagging and feature subspacing to de-correlate trees

• Aggregate the predictions by voting

• One of the most successful algorithms of the 21st century with ~26000 citations

• Most people still use this “vanilla” version (better variants to come later)
Algorithm 1: Train random forest

Inputs: Features $X \in \mathbb{R}^{N \times D}$, classes $Y \in \mathbb{I}^{N \times K}$, number of features to sample $\lambda \in \{1, \ldots, D\}$, number of trees to construct $L$

Outputs: Forest of $L$ trees $\mathcal{T}_{1:L}$

1: for $\ell \in 1:L$ do
2:   Construct bootstrap sample of data $\{X'_\ell, Y'_\ell\}$ by sampling $N$ times with replacement from $\{X, Y\}$
3:   $\mathcal{T}_\ell \leftarrow \text{GROWTREE}(X'_\ell, Y'_\ell, \lambda)$
4: end for

See next slide
Algorithm 2: GrowTree

Inputs: Features $X \in \mathbb{R}^{N \times D}$, classes $\mathcal{Y} \in \mathbb{I}^{N \times K}$, number of features to sample $\lambda \in \{1, \ldots, D\}$

Outputs: Subtree: a tuple \{split dimension, split point, left branch, right branch\} if root is a discriminant node, otherwise a class label $u \in 1^{1 \times K}$ representing a leaf.

1: Subsample features ids $\delta$ by sampling from $\{1, \ldots, D\}$ $\lambda$ times without replacement.
2: Set $\mathcal{X} \leftarrow X(:, \delta)$
3: $E_{\text{base}} = \text{Entropy}(\mathcal{Y})$
4: for $\nu = 1 : \lambda$ do
5: $u \leftarrow \text{SORT}(\mathcal{X}(\cdot, \nu))$
6: for $i = 2 : N$ do
7: $S_{i, \nu} \leftarrow (u_i + u_{i-1}) / 2$
8: $j_\ell \leftarrow \{j \in \{1, \ldots, N\} : \mathcal{X}_{j, \nu} \leq S_{i, \nu}\}$
9: $j_r \leftarrow \{1, \ldots, N\} \setminus j_\ell$
10: $G_{i, \nu} = E_{\text{base}} - \frac{\|j_\ell\|_0}{N} \text{Entropy}(\mathcal{Y}(j_\ell)) - \frac{\|j_r\|_0}{N} \text{Entropy}(\mathcal{Y}(j_r))$
11: end for

12: end for
13: $\{i^*, \nu^*\} = \text{argmax}_{i, \nu} G_{i, \nu}$
14: if $G_{i^*, \nu^*} \leq 0$ then
15: return $\text{Mode}(\mathcal{Y})$
16: end if
17: $j_\ell = \{j \in \{1, \ldots, N\} : U_{j, \nu^*} \leq S_{i^*, \nu^*}\}$
18: $j_r = \{1, \ldots, N\} \setminus j_\ell$
19: return $\{\nu^*, i^*, \text{GrowTree}(X(j_\ell, :)\mathcal{Y}(j_\ell, :)\lambda), \text{GrowTree}(X(j_r, :)\mathcal{Y}(j_r, :)\lambda)\}$
Practicalities
Uncertainty

• Each tree has its own prediction, often these predictions will disagree

• Can estimate probabilities rather by taking the proportion of trees that estimated that class
Number of Trees

- More the merrier - limited by computational budget
- 500 is relatively standard
- More trees gives a smoother surface

[Figure 3.3: A first classification forest and the effect of forest size. (a) Training points belonging to two classes. (b) Different training trees produce different partitions and thus different leaf predictors. The colour of tree nodes and edges indicates the class probability of training points going through them. (c) In testing, increasing the forest size \( T \) produces much smoother class posteriors. All experiments were run with \( D = 2 \) and axis-aligned weak learners. See text for details.]

Data which is “different” than the training data. The larger the difference, the larger the uncertainty. Thanks to all trees being different from one another, increasing the forest size from \( T = 1 \) to \( T = 200 \) produces much smoother posteriors (fig. 3.3c). Now we observe higher confidence near the training points and lower confidence away from training regions of space; an indication of good generalization behaviour.

For few trees (e.g. \( T = 8 \)) the forest posterior shows clear box-like artifacts. This is due to the use of an axis-aligned weak learner model. Such artifacts yield low quality confidence estimates (especially [Criminsini et al 2011])

[Criminsini et al 2011]
Subspace Size

• Usually makes surprisingly little difference and often chosen for speed more than accuracy

• Avoid extreme values

• Common choices are $\lambda = D^{0.5}$ and $\lambda = \log_2 D$
Gini Split Criterion

- Sometimes people use the Gini split criterion instead of entropy

\[ GINI(t) = 1 - \sum_j [p(j | t)]^2 \]

- GINI Index for a given node t:
  - Maximum (1-1/n): records equally distributed in n classes
  - Minimum 0: all records in one class

- Usually only makes minimal difference and neither is particularly considered to be better on average than the other

[Amr Barakat 2015]
Tree Ensembles vs Other Algorithms

Positives

• Very easy to use - 1 line of code
• Fast at train and test time
• State-of-the-art out-of-box performance for many problems
• Easily deal with different data types
• Very simple compared to most machine learning algorithms
• Decent robustness against overfitting
• Great as a baseline before trying something more involved

Negatives

• Little flexibility or ability to incorporate prior knowledge
• Typically worse performance than deep neural nets for huge datasets
• Vanilla RF poor on data with highly correlated features (CCFs and Rotations forests still good though)
• Can be quite dependent on the quality of the features used - typically helpful to preprocess data using some sort of feature extractor
• Provides uncertainty estimates but these often have poor accuracy
• Not good at extrapolation (though at the end of the day nothing really is)
Uneven Classes

- Need to be careful with any parameter tuning as predictive accuracy likely to be an unreliable metric - just going for the most common class will do well
- If you have more data than you can actually use, try to generate a more even sample as training data
- Alternative you can also create an artificial dataset with duplicate instances of the small classes to balance them out
Extensions
Regression

- Same principle but leaves have a local regression model (e.g. linear regression, Gaussian process) rather than a class
- Unlike classification, usually necessary to have a minimum number of points in each leaf node for this local model
- Less common than classification but still powerful

![Diagram of regression models](image)

Fig. 4.2: **Example predictor models.** Different possible predictor models. (a) Constant. (b) Polynomial and linear. (c) Probabilistic-linear. The conditional distribution $p(y|x)$ is returned in the latter.

[Criminsini et al 2011]
Regression (2)

- Use different split criterion, e.g. total variance instead of entropy

\[
p(y|v) = \frac{1}{T} \sum_{t=1}^{T} p_t(y|v).
\]

Fig. 4.3: Regression forest: the ensemble model. The regression forest posterior is simply the average of all individual tree posteriors.
Custom Features

• Instead of operating on the input data $X$ directly, construct additional features $X \leftarrow [X, g(X)]$

• This allows more complex splits in the original space

• For example, $g(X) = (x_1 - a)^2 + (x_2 - b)^2$ allows splitting on the distance from a point $(a, b)$

• More generally, use the output from some nonlinear feature generator such as a neural net
Random Rotations

- Randomly rotate the dataset separately for each tree

- Reduces correlation between trees which can improve performance

- No longer restricted to piecewise linear decision surfaces => smoother

- Can damage performance when little correlation between features
Random Projections

• Similar to random rotations, but applied at each node separately, rather than globally for each tree

• Again can give performance improvements or worsen performance depending on dataset

• On average does slightly better than random forests
Rotation Forests

• Instead of projecting randomly, use principle component analysis (PCA) on small groups of randomly sampled features

• Significantly reduces sensitivity to correlation between features, without damaging performance when there is little correlation

• Typically does not use random subspaceing => significantly slower

• Can give large performance improvements, particularly when strong correlation

[Rodriguez et al 2006]
Extremely Randomized Trees

• Choose splits randomly rather than in a principled manner

• Amazingly can actually improve performance compared with random forests in some situations

• Comparable performance on average

[Geurts et al 2006]
Canonical Correlation Forests

• Project to maximally decorrelated space at each node using canonical correlation analysis (CCA)

• Speed of random forests but with big improvements in accuracy, particularly with correlated data

[Rainforth and Wood 2015]
Canonical Correlation Forests (2)

- Not being axis aligned means each tree can fit the data better
- Also reduces correlation between trees predictions which improves accuracy further

![Diagram of tree resulting from the axis aligned decision tree algorithm CART Breiman et al. (1984) (left) and CCT (right) on dataset from demonstration in Figure 2. It is clear that the partitioning the CART tree defines is inferior due to axis-aligned splitting restriction and will give worse predictive performance.](image)
Canonical Correlation Forests (3)

Figure 1: Decision surfaces for artificial spirals dataset. (a) Shows the hierarchical partitions and surface for a single axis aligned tree while (b) shows the effect of averaging over a number of, individually randomized, axis aligned trees. (c) Shows a single canonical correlation tree (CCT) and (d) demonstrates that averaging over CCTs to give a canonical correlation forest leads to “smoother” decision surfaces which better represent the data than the axis aligned equivalent.
Uneven Voting

- Some trees may better than others
- Can get small performance improvements by not weighting trees evenly
- For example weighting simpler trees higher or using cross-validation schemes to calculate weights

[Robnik-Sikonja 2004]
Useful Packages

  - Gui with java back end. Operates on csv files and allows lots of algorithms to be used at the same package

  - open source package with lots of machine learning algorithms built in.

- **TreeBagger (Matlab)** - [https://uk.mathworks.com/help/stats/treebagger.html](https://uk.mathworks.com/help/stats/treebagger.html)
  - in built random forest package

- **randomforest-matlab** - [https://code.google.com/archive/p/randomforest-matlab/](https://code.google.com/archive/p/randomforest-matlab/)
  - Faster open source matlab version
Useful Packages (2)

- Canonical correlation forests (matlab, python) - https://bitbucket.org/twgr/ccf (matlab), https://github.com/asross/oo_trees (python, not our implementation so no guarantees) - our algorithm, state-of-the-art predictive accuracy (classification only)

- C++ - https://github.com/bjoern-andres/random-forest

- R - https://www.tutorialspoint.com/r/r_random_forest.htm
Further Reading


• Nando de Freitas’ lectures - https://youtu.be/-dCtJjlEEgM and https://youtu.be/3kYujfDgmNk


Thanks for listening, any questions?

Feel free to email me at twgr@robots.ox.ac.uk