Decision trees

Yacine Debbabi

December 5, 2020

Contents

1 Tree models **1**

2 How to estimate a tree response? **1**

3 How to grow a tree? **2**

4 How to prune a single tree? **3**

5 How to quantify the relative importance of features? **3**

6 Random forests **3**

1 Tree models

Consider our data consists of p inputs and a response for N observations. We model the response as a constant-by-part function in M distinct regions, i.e.

$$f(x) = \sum_{m=1}^{M} c_m \mathbb{1}(x \in R_m). \tag{1}$$

Here we first show how to estimate the response for a given set of regions, and then discuss how trees can be used to build region splits.

2 How to estimate a tree response?

We estimate $\hat{c} = (\hat{c}_1, ..., \hat{c}_M)$ in each region by minimizing an impurity/loss function, i.e.

$$\hat{c}_m = \text{argmin}_c Q_m(c) \tag{2}$$

for each $m \in 1, ..., M$. For regression problems, this is is often chosen as a residual sum of squares, i.e.

$$Q_m(c) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - c)^2. \tag{3}$$

The solution $\hat{c}_m$ is easily obtained, by differentiating $Q_m(c)$ against c, as the average of responses y_i whose input x_i lie in R_m. For classification problems, we classify the observations in node m to class $\hat{k}_m = \text{argmax}_k \hat{p}_{k,m}$, where $\hat{p}_{k,m}$ denotes the proportion of class k observations in node m. Impurity is often quantified using the misclassification rate

$$Q_m(c) = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{1}(y_i \neq c), \tag{4}$$

when cost-complexity pruning the tree (see later but not sure why), and using the Gini index

$$Q_m(c) = \sum_{k \neq k'} \hat{p}_{k,m} \hat{p}_{k',m} \tag{5}$$

when growing the tree. The Gini index is preferable for such task because (a) it is differentiable (and hence more amenable to numerical optimization), and (b) more sensitive to node probabilities than the misclassification rate. The latter can be further demonstrated by comparing two scenarios: (a) (100,300) vs (100,300) splits, and (b) (200,400) vs (200,0) splits. Both scenarios show the same misclassification rate but a lower Gini index for the second scenario.

3 How to grow a tree?

We now discuss how to grow a regression tree. Finding a multi-region partition that minimizes impurity is generally computationally infeasible. Hence we proceed with a greedy algorithm. Starting with all the data, we consider a splitting variable j and a split point s, and define the half planes $R_1(j,s) = \{X|X_j \leq s\}$ and $R_2(j,s) = \{X|X_j > s\}$. Then we seek the splitting variable and point that solve

$$\min_{j,s} \left[\min_{c_1} N_{m_1} Q_{m_1}(c_1) + \min_{c_2} N_{m_2} Q_{m_2}(c_2) \right]. \tag{6}$$

For each splitting variable j, the j-feature observations x_i^j, $i = 1, ..., N$, can be sorted, and the objective function be evaluated on splitting points s sampled from the j-feature values. The procedure can be repeated separately on the subregions, and the resulting region splits can be represented as a binary tree.

```
from sklearn.datasets import load_boston
from sklearn.tree import DecisionTreeRegressor, plot_tree

X, y = load_boston(return_X_y=True)

regressor = DecisionTreeRegressor(
    criterion="mse",
    max_depth=2,
    min_samples_leaf=10,
).fit(X, y)

plot_tree(regressor)
plt.show()
```

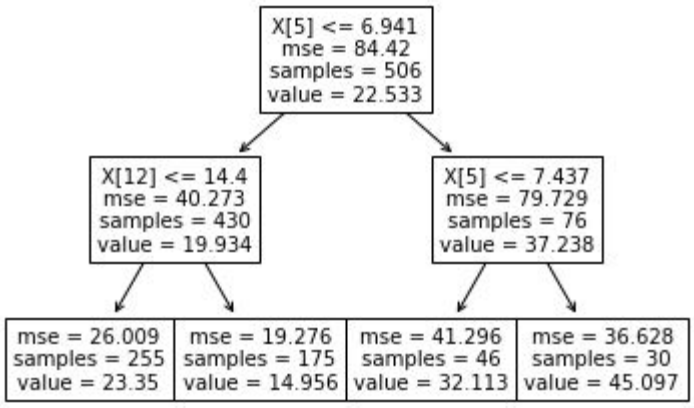

Figure 1: Example of regression decision tree.

When shall we stop growing a tree? Clearly a very large tree might overfit the data, while a small tree might not capture the important structure. The preferred strategy is to grow a large tree T_0, which is then pruned. Although fully grown and unpruned trees can potentially be very large on some data sets, we usually cap the depth of the tree and/or the minimum number of observations in each region. This helps reduce memory consumption and training runtimes.

4 How to prune a single tree?

We now discuss the **cost-complexity pruning** approach. We define a subtree $T \subset T_0$ to be any tree that can be obtained by pruning T_0, i.e. by removing any number of its internal (non-terminal) nodes (and their descendants). A pruned tree is obtained by minimizing the cost-complexity criterion

$$\hat{T}_\alpha = \text{argmin}_T \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|, \tag{7}$$

where m indexes the terminal nodes and $|T|$ denotes the number of terminal nodes of tree T. This is achieved with the **weakest link pruning** approach; this is discussed further here. One can show the solution is unique. Estimation of α is achieved by cross-validation.

5 How to quantify the relative importance of features?

For a single decision tree T, the relevance of predictor variable X_l is defined as

$$I_l^2(T) := \sum_m i_m^2 \mathbb{1}(\text{f splits m}) \tag{8}$$

where m is an internal node of T and i_m^2 is the increase in $N_m Q_m$ which is obtained from the split. This is generalized for a forest by averaging the above over all trees.

6 Random forests

Random forests are built by constructing a set of trees randomly, by randomly selecting a fixed-size set of feature candidates, and "averaging" their predictions.

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Figure 2: Random forest construction algorithm.

The parameters to tune are (a) the number of trees, (b) the max number of feature candidates, and (c) (single) tree depth parameters. Random forests allow estimation of an "out-of-bag" (OOB) error. An error is calculated for each sample by averaging the trees which did not use this sample for training. We empirically find in general that the OOB error converges as we increase the number of trees. This can be used to derive an upper bound of the number of trees required. The other parameters can be optimized by cross-validation using the GridSearchCV estimator.

```
from sklearn.ensemble import RandomForestRegressor

params = {
   "n_estimators": 100,
   "max_features": "sqrt",
   "max_depth": 4,
   "oob_score": True,
   "random_state": 0,
   "max_samples": .8
}

#

oob_scores = []
n_est_grid = [int(x) for x in np.logspace(1.2, 3, 20)]

for n in n_est_grid:
   params.update({"n_estimators": n})
   rf = RandomForestRegressor(**params).fit(X, y)
   oob_scores.append(rf.oob_score_)

plt.semilogx(n_est_grid, oob_scores, "bo-")
plt.xlabel("Number of trees"); plt.ylabel("OOB score"); plt.show()

# Hyperparameter optimization

from sklearn.model_selection import GridSearchCV
```

```
gs = GridSearchCV(estimator=rf, param_grid={"n_estimators": [100], "max_depth": [2, 4, 6],
    "max_features": ["auto", "sqrt"]}, cv=5).fit(X, y)
gs.best_estimator_
```

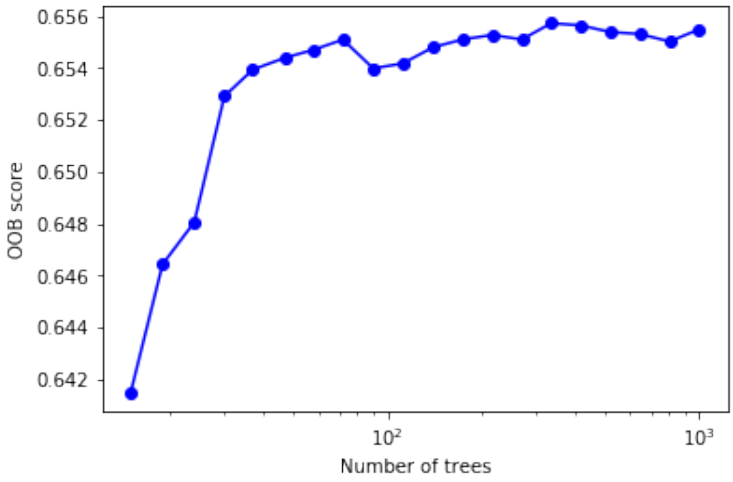

Figure 3: OOB scores converge as we increase the number of trees.