Training decision trees

Terence Parr
MSDS program
University of San Francisco

Training overview

- Training partitions feature space into rectangular hypervolumes chasing reduced y impurity in subregions
- Hypervolumes are specified by sequence of *splits* that test a single feature and value at a time
- Each split becomes a decision node in decision tree
- Records in an “atomic” hypervolume form a single leaf
- Hypervolume described by conditionals on path from root to leaf

How to create a decision node

- Each split chosen greedily to minimize impurity in subregion y's
  - Regressor: variance or MSE
  - Classifier: gini criterion or entropy
- To choose split, exhaustively try each (variable,value) pair and pick the pair with min average impurity for subregions created by that split

Fitting decision trees

1 **Algorithm:** *dtreefit*$(X,\ y,\ min\_samples\_leaf,\ loss)$
2 **if** $|X| < min\_samples\_leaf$ **then** return Leaf(y)
3 $col, split = bestsplit(X, y, loss)$
4 **if** $col =$ *-1* **then** return Leaf(y)
5 $lchild = fit(X[X \leq split], y[X \leq split])$
6 $rchild = fit(X[X > split], y[X > split])$
7 **return** $DecisionNode(col, split, lchild, rchild)$

Best split var/value

```
1  Algorithm: bestsplit(X, y, loss)
2     best = (col = −1, split = −1, loss = loss(y))
3     for j = 1..p do
4        foreach split ∈ X_{−,j} do
5           yl = y[X ≤ split]
6           yr = y[X > split]
7           if |yl| = 0 or |yr| = 0 then continue
8           l = (loss(yl)+loss(yr)) / 2
9           if l = 0 then return col, split
10          if l < best[loss] then best = (col, split, l)
11       end
12    end
13    return col, split
```

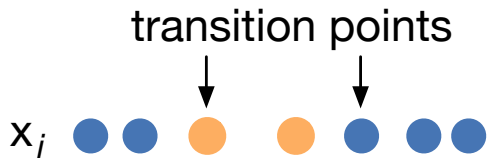
Should pick midpoint between split value and next smallest X

Decision tree prediction

```
1 Algorithm: predict(node,x)
2    if node is leaf then
3       if classifier then return mode(node.y)
4       return mean(node.y)
5    end
6    if x[node.col] ≤ node.split then return predict(node.lchild, x)
7    return predict(node.rchild, x)
```

This find_best_split() is inefficient

- It has a nested loop; tries all combinations of p variables and worst-case n unique values in each variable at root: O(n*p)
- Cost of computing loss on all values in subregion each iteration is also expensive
- For classification, can mitigate by sorting by ith var then we know at a specific x value, everything to left is less and right is greater; keep track of class counts to left/right
- Reduce computation by focusing on transitions points in x, effectively focusing on unique(x)



Improving generality

- Select a subset of values as candidates, k; then we reduce O(n*p) to O(k*p) for k << n (n is often huge)
- We should really pick split point between two x values: $(x^{(i)}+x^{(i-1)})/2$ (if sorted)
- More likely split point is between not on x values, so midpoint is good guess as to underlying distribution
- And, of course, we can reduce tree height with min_samples_leaf to restrict complexity