Training decision trees

Terence Parr
MSDS program
University of San Francisco

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 <u>greedily</u> 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 weighted average impurity for subregions created by that split

Fitting decision trees

subsets MSE or gini

```
**Algorithm:** $dtreefit(X,\ y,\ min\_samples\_leaf,\ loss)$
    **if** $|X| < min\_samples\_leaf$ **then** return Leaf($y$)
    $col, split = bestsplit(X, y, loss)$
    **if** $col$ = -1 **then** return Leaf($y$)    (*No better split?*)
    $lchild = dtreefit(X[X_{col} \leq split], y[X_{col} \leq split], min\_samples\_leaf, loss)$
    $rchild = dtreefit(X[X_{col} > split], y[X_{col} > split], min\_samples\_leaf, loss)$
    **return** $DecisionNode(col, split, lchild, rchild)$
```

Overall fit: pass in full X, y to dtreefit() and get back the decision tree

Best split var/value

```
Algorithm: bestsplit(X, y, loss)
  $best = (col = -1, split = -1, loss = loss(y))$
  for $col$ = 1..p do
    foreach $split \in X_{col}$ do
      $yl = y[X_{col} \leq split]$
      $yr = y[X_{col} > split]$
      if $|yl| = 0$ or $|yr| = 0$ then continue
      $l = \frac{|yl| \times loss(yl) + |yr| \times loss(yr)}{|y|}$        (weighted average of subregion losses)
      if $l = 0$ then return $col$, $split$
      if $l < best[loss]$ then $best = (col, split, l)$
    end
  end
  return $col$, $split$
```

Should pick midpoint between split value and next smallest x

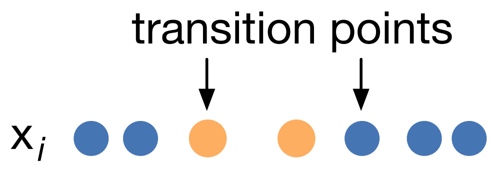
UNIVERSITY OF SAN FRANCISCO

Decision tree prediction

```
**Algorithm:** *predict(node,x)*
    **if** *node is leaf* **then**
        **if** *classifier* **then return** $mode(node.y)$
        **return** $mean(node.y)$
    **end**
    **if** $x[node.col] \leq node.split$ **then return** predict$(node.lchild, x)$
    **return** predict$(node.rchild, x)$
```

The usual bestsplit() is inefficient

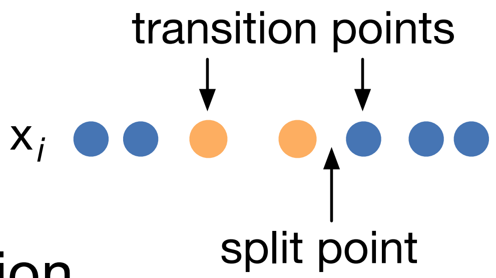
- It has a nested loop; tries all combinations of p variables and worst-case n unique values in each variable at root: O(n*p)
- Cost of computing loss on all values in subregion each iteration is also expensive
- For classification, can mitigate by sorting by ith var then we know at a specific x value, 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 and efficiency

- 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) (our project k=11)
- 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



Decision tree prediction via x subset

```
**Algorithm:** $bestsplit(X, y, loss)$
  $best = (col = -1, split = -1, loss = loss(y))$
  **for** $j$ = $1..p$ **do**
    $candidates$ = randomly pick $k \ll n$ values from $X_{\_,j}$
    **foreach** $split \in candidates$ **do**
      $yl = y[X \leq split]$
      $yr = y[X > split]$
      **if** $|yl| = 0$ $or$ $|yr| = 0$ **then continue**
      $l = \frac{|yl| \times loss(yl) + |yr| \times loss(yr)}{|y|}$        $(weighted\ average\ of\ subregion\ losses)$
      **if** $l = 0$ **then return** $col, split$
      **if** $l < best[loss]$ **then** $best = (col, split, l)$
    **end**
  **end**
  **return** $col, split$
```

Can even pick just 1 split randomly or in min..max range (see "Extremely random trees"); any small k value works.

Prediction

- Start at the root node and descend through the decision nodes to the appropriate leaf
- At each decision node, test a specific variable's value against the split value stored in the decision node

```
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 $x[node.col] \leq node.split$ then return predict($node.lchild, x$)
7     return predict($node.rchild, x$)
```