Machine Learning – Brett Bernstein

Week 9 Lecture: Concept Check Exercises

Trees

1. (a) How many regions (leaves) will a tree with k node splits have?
   (b) What is the maximum number of regions a tree of height k can have? Recall that the height of a tree is the number of edges in the longest path from the root to any leaf.
   (c) Give an upper bound on the depth needed to exactly classify n distinct points in $\mathbb{R}^d$. [Hint: In the worst case each leaf will have a single training point.]

   Solution.

   (a) Given a fixed tree, if we split a leaf node we add a single leaf to the tree. Thus k splits corresponds to $k+1$ leaves.
   (b) A tree of height k can have at most 2^k regions (leaves).
   (c) A tree of height $\lceil \log_2(n) \rceil$ is sufficient to distinguish all possible values for the first feature. At each leaf we can then put another tree of this height that distinguishes the second feature, and so forth. These give an upper bound of $d\lceil \log_2(n) \rceil$.

2. This question involves fitting a regression tree using the square loss. Assume the n data points for the current node are sorted by the first feature. Give pseudocode with $O(n)$ runtime for optimally splitting the current node with respect to the first feature.

   Solution.

```
import numpy as np
def bestSplit(y) :
    '''
    Greedily computes the best splitting point for the current node.
    Assumes there is at least 2 values. There are more numerically
    stable ways of doing this
    [see The Art of Computer Programming p. 232, Vol 2, 3rd Edition]
    @param y : array-like, shape = [n_samples,], contains the
    output values for each sample. Assumes the
    values are sorted by the corresponding first
    feature values.
    @return the value i such that inputs 0,...,i belong in the
    left subtree.
    '''
    sums = np.cumsum(y) #partial sums of y-values
    sumsq = np.cumsum([a**2 for a in y]) #partial sums of squared y-values
```

```
S = sums[−1] #sum of all y−values
SS = sumsq[−1] # sum of all squared y−values
bestIdx = −1
bestLoss = None
N = len(y)
for idx in range(N−1) :
    leftLoss = sumsq[idx]/(idx+1.0) − (sums[idx]/(idx+1.0))**2
    rightLoss = (SS−sumsq[idx])/(N−(idx+1.0))−( (S−sums[idx])/(N−(idx+1.0)) )**2
    loss = leftLoss+rightLoss
    if bestIdx == −1 or loss < bestLoss :
        bestIdx = idx
        bestLoss = loss
return bestIdx
```

3. Suppose we are looking at a fixed node of a classification tree, and the class labels are, sorted by the first feature values,

$$4, 1, 0, 0, 1, 0, 2, 3, 3.$$

We are currently testing splitting the node into a left node containing $4, 1, 0, 0, 1, 0$ and a right node containing $2, 3, 3$. For each of the following impurity measures, give the value for the left and right parts, along with the total score for the split.

(a) Misclassification error.

(b) Gini index.

(c) Entropy.

Solution.

(a) Left: $3/6$, Right: $1/3$, Total: $6(3/6) + 3(1/3) = 4$

(b) Left: $3/6(3/6) + 2/6(4/6) + 1/6(5/6) = 22/36$, Right: $1/3(2/3) + 2/3(1/3) = 4/9$, Total: $6(22/36) + 3(4/9) = 30/6 = 5$

(c) Left: $-3/6 \log(3/6) - 2/6 \log(2/6) - 1/6 \log(1/6)$, Right: $-1/3 \log(1/3) - 2/3 \log(2/3)$, Total:

$$6[-3/6 \log(3/6) - 2/6 \log(2/6) - 1/6 \log(1/6)] + 3[-1/3 \log(1/3) - 2/3 \log(2/3)].$$

Bagging

1. Let $X_1, \ldots, X_n$ be an i.i.d. sample from a distribution with mean μ and variance σ^2. How large must n be so that the sample mean has standard error smaller than .01?

Solution. Recall that the sample mean has variance

$$\mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{\mathrm{Var}(X_1)}{n} = \frac{\sigma^2}{n},$$

with standard error $\sigma/\sqrt{n}$. Thus we have

$$\sigma/\sqrt{n} < .01 \iff n > 10000\sigma^2.$$

2. Let $X_1, \ldots, X_{2n+1}$ be an i.i.d. sample from a distribution. To estimate the median of the distribution, you can compute the sample median of the data.

   (a) Give pseudocode that computes an estimate of the variance of the sample median.

   (b) Give pseudocode that computes an estimate of a 95% confidence interval for the sample median.

   Solution.

   (a) i. Draw B bootstrap samples $D^1, \ldots, D^B$ each of size $2n+1$. The samples are formed by drawing uniformly with replacement from the original data set $X_1, \ldots, X_{2n+1}$. We will make a total of $B(2n+1)$ draws.
   ii. For each D^i compute the corresponding median $\hat{m}_i$.
   iii. Compute the sample variance of the B medians $m_1, \ldots, m_B$.

   (b) i. Draw B bootstrap samples $D^1, \ldots, D^B$ each of size $2n+1$. The samples are formed by drawing uniformly with replacement from the original data set $X_1, \ldots, X_{2n+1}$. We will make a total of $B(2n+1)$ draws.
   ii. For each D^i compute the corresponding median $\hat{m}_i$.
   iii. Compute the 2.5% and 97.5% sample quantiles of the list $\hat{m}_1, \ldots, \hat{m}_B$. Use these as the estimates of the left and right endpoints of the confidence interval, respectively.

Boosting

1. ($\star$) Show the exponential margin loss is a convex upper bound for the $0-1$ loss.

   Solution. Recall that the exponential margin loss is given by $\ell(y, a) = e^{-ya}$ where $y \in \{-1, 1\}$ and $a \in \mathbb{R}$, and the $0-1$ loss is $\mathbf{1}(y \neq \mathrm{sgn}(a))$. If $\mathrm{sgn}(y) \neq a$ then $ya \leq 0$ and

   $$e^{-ya} \geq 1 - ya \geq 1 = \mathbf{1}(y \neq \mathrm{sgn}(a)).$$

   In general $e^{-ya} \geq 0$ so the we obtain the upper bound. To prove convexity, we compute the second derivative and note that it is positive:

   $$\frac{\partial^2}{\partial a^2} e^{-ya} = y^2 e^{-ya} > 0.$$

2. Show how to perform gradient boosting with the hinge loss.

   Solution. Recall that the hinge loss is given by $\ell(y, a) = \max(0, 1 - ya)$. Define g by

$$g(y, a) = \begin{cases} -y & \text{if } 1 - ya > 0, \\ 0 & \text{else.} \end{cases}$$

   Then $g(y, a)$ is a subgradient of $\ell(y, a)$ with respect to a. At stage m of gradient boosting, we alredy have formed

$$f_{m-1} = \sum_{i=1}^{m-1} \nu_i h_i.$$

   We then compute the pseudoresiduals r_m given by

$$r_m = -\left(g(y_1, f_{m-1}(x_1)), \ldots, g(y_n, f_{m-1}(x_n))\right).$$

   After building the mock dataset $D^m = \{(x_1, (r_m)_1), \ldots, (x_n, (r_m)_n)\}$ we perform a least squares fit to obtain $h_m \in \mathbb{H}$. Then we can determine ν_m (usually a small fixed value). Finally we let $f_m = f_{m-1} + \nu_m h_m$.

3. Suppose we are using gradient boosting. On each step we can do a better job of fitting the pseudoresiduals if we allow for deeper trees. Why might deep trees be discouraged while gradient boosting?

   Solution. Deep trees can lead to overfitting the data.