Growing a Tree with ID3...and Bagging

Cornell CS 4/5780 — Spring 2022

Using Impurity to Build A Tree: ID3-Algorithm

Base Cases: given dataset S

- If all examples in S have the same label y, return leaf with label y.
- If all examples in S have the same features x, return leaf with label y either the mode (for classificaton) or the mean (for regression) of labels in the dataset.

Recursive Case:

- Try all features and all possible splits.
  - For a discrete-valued feature, we split creating one child for each value the feature could take on.
  - For a continuous-valued feature we try all possible thresholds for a binary split.
- Choose the feature split that minimizes the impurity! (Or just pick one if multiple minimizers.)
- Then call ID3 recursively on each of the subsets of the split. (If one of those subsets is empty, just create for it a leaf node with label y either the mode (for classificaton) or the mean (for regression) of labels in the dataset.)
- Output a tree that stores the selected feature and split threshold and has the outputs of the recursive ID3 calls as children.

Quiz: Why don't we stop if no split can improve impurity?

CART summary: CART are very light-weight predictors. They are very fast during testing (if balanced, test time is roughly logarithmic in model size; how does this compare to linear models?). But, they're usually not competitive in accuracy. Fortunately, trees can become very strong through bagging (Random Forests) and boosting (Gradient Boosted Trees).

Bagging

Also known as Bootstrap Aggregating (Breiman 96). Bagging is an **ensemble** method, which combines multiple classifiers to reduce variance. Remember the Bias/Variance decomposition:

$$\underbrace{\mathbb{E}[(h_D(x)-y)^2]}_{\text{Error}} = \underbrace{\mathbb{E}[(h_D(x)-\bar{h}(x))^2]}_{\text{Variance}} + \underbrace{\mathbb{E}[(\bar{h}(x)-\bar{y}(x))^2]}_{\text{Bias}} + \underbrace{\mathbb{E}[(\bar{y}(x)-y(x))^2]}_{\text{Noise}}$$

Our goal is to reduce the variance term: $\mathbb{E}[(h_D(x)-\bar{h}(x))^2]$. For this, we want $h_D \to \bar{h}$. The weak law of large numbers says (roughly) for i.i.d. random variables x_i with mean $\bar{x}$, we have, $\frac{1}{m}\sum_{i=1}^m x_i \to \bar{x}$ in the limit as $m \to \infty$. Apply this to classifiers: Assume we have m training sets $D_1, D_2, \ldots, D_n$ drawn from P^n. Train a classifier on each one and average result:

$$\hat{h} = \frac{1}{m}\sum_{i=1}^m h_{D_i} \to \bar{h} \qquad as\ m \to \infty$$

We refer to such an average of multiple classifiers as an **ensemble** of classifiers.

Good news: If $\hat{h} \to \bar{h}$ the variance component of the error must also vanish, *i.e.* $\mathbb{E}[(\hat{h}(x)-\bar{h}(x))^2] \to 0$
Problem: We don't have m data sets $D_1, \ldots, D_m$, we only have D.

Solution: Bagging (Bootstrap Aggregating)

Simulate drawing from P by drawing uniformly with replacement from the set D: i.e. let $Q(X,Y|D)$ be a probability distribution that picks a training sample $(\mathbf{x}_i, y_i)$ from D uniformly at random. More formally, $Q((\mathbf{x_i}, y_i)|D) = \frac{1}{n}$ for all $(\mathbf{x_i}, y_i) \in D$ with $n = |D|$. We sample the set $D_i \sim Q^n$, i.e. $|D_i| = n$, and D_i is picked with replacement from $Q|D$.

Q: What is $\mathbb{E}[|D \cap D_i|]$?

Ensemble classifier/regressor: $\hat{h}_D = \frac{1}{m}\sum_{i=1}^m h_{D_i}$

Observe that $\hat{h}_D = \frac{1}{m}\sum_{i=1}^m h_{D_i} \nrightarrow \bar{h}$ in the limit as the size of the ensemble m and the original training set size $|D|$ go to infinity at appropriate rates. But we can't just use the weak law of large numbers to show this.

Analysis

Although we cannot prove that the new samples are i.i.d., we can show that they are drawn from the original distribution P. Assume P is discrete, with $P(X = x_i) = p_i$ over some set $\Omega = x_1, \ldots x_N$ (N very large) (let's ignore the label for now for simplicity)

$$Q(X=x_i)=\underbrace{\sum_{k=1}^{n}\binom{n}{k}p_i^k(1-p_i)^{n-k}}_{\substack{\text{Probability that are}\\ \text{k copies of } x_i \text{ in D}}}\underbrace{\frac{k}{n}}_{\substack{\text{Probability}\\ \text{pick one of}\\ \text{these copies}}}=\frac{1}{n}\underbrace{\sum_{k=1}^{n}\binom{n}{k}p_i^k(1-p_i)^{n-k}k}_{\substack{\text{Expected value of}\\ \text{Binomial Distribution}\\ \text{with parameter } p_i\\ \mathbb{E}[\mathbb{B}(p_i,n)]=np_i}}=\frac{1}{n}np_i=p_i$$

So each data set D_l' is drawn from P, but not independently. There is a simple intuitive argument why $Q(X=x_i)=P(X=x_i)$. So far we assumed that you draw D from P^n and then Q picks a sample from D. However, you don't have to do it in that order. You can also view sampling from Q in reverse order: Consider that you first use Q to reserve a "spot" in D, i.e. a number from 1,...,n, where i means that you sampled the i^{th} data point in D. So far you only have the slot, i, and you still need to fill it with a data point (x_i, y_i). You do this by sampling (x_i, y_i) from P. It is now obvious that which slot you picked doesn't really matter, so we have $Q(X=x)=P(X=x)$.

Bagging summarized

1. Sample m data sets $D_1,\dots,D_m$ from D with replacement.
2. For each D_j train a classifier $h_j()$
3. The final classifier is $h(\mathbf{x})=\frac{1}{m}\sum_{j=1}^m h_j(\mathbf{x})$.

In practice larger m results in a better ensemble, however at some point you will obtain diminishing returns. Note that setting m unnecessarily high will only slow down your classifier but will **not** increase the error of your classifier.

Advantages of Bagging

- Easy to implement
- Reduces variance, so has a strong beneficial effect on high variance classifiers.
- As the prediction is an average of many classifiers, you obtain a mean score *and variance*. Latter can be interpreted as the uncertainty of the prediction. Especially in regression tasks, such uncertainties are otherwise hard to obtain. For example, imagine the prediction of a house price is $300,000. If a buyer wants to decide how much to offer, it would be very valuable to know if this prediction has standard deviation +-$10,000 or +-$50,000.

- Bagging provides an unbiased estimate of the test error, which we refer to as the *out-of-bag error*. The idea is that each training point was not picked and all the data sets D_k. If we average the classifiers h_k of all such data sets, we obtain a classifier (with a slightly smaller m) that was not trained on $(\mathbf{x}_i, y_i)$ ever and it is therefore equivalent to a test sample. If we compute the error of all these classifiers, we obtain an estimate of the true test error. The beauty is that we can do this without reducing the training set. We just run bagging as it is intended and obtain this so called out-of-bag error for free.

  More formally, for each training point $(\mathbf{x}_i, y_i)\in D$ let $S_i=\{k|(\mathbf{x}_i, y_i)\notin D_k\}$ - in other words S_i is a set of all the training sets D_k, which do not contain $(\mathbf{x}_k, y_k)$. Let the averaged classifier over all these data sets be $\tilde{h}_i(\mathbf{x})=\frac{1}{|S_i|}\sum_{k\in S_i}h_k(\mathbf{x})$. Then the-of-bag error becomes simply the average error/loss that all these classifiers yield

  $$\epsilon_{\text{OOB}}=\frac{1}{n}\sum_{(\mathbf{x}_i,y_i)\in D} l(\tilde{h}_i(\mathbf{x_i}), y_i).$$

  This is an estimate of the test error, because for each training point we used the subset of classifiers that never saw that training point during training. if m is sufficiently large, the fact that we take out some classifiers has no significant effect and the estimate is pretty reliable.

Random Forest

One of the most famous and useful bagged algorithms is the **Random Forest**! A Random Forest is essentially nothing else but bagged decision trees, with a slightly modified splitting criteria. The algorithm works as follows:

1. Sample m data sets $D_1,\dots,D_m$ from D with replacement.
2. For each D_j train a full decision tree $h_j()$ (max-depth=∞) with one small modification: before each split randomly subsample $k\leq d$ features (without replacement) and only consider these for your split. (This further increases the variance of the trees.)
3. The final classifier is $h(\mathbf{x})=\frac{1}{m}\sum_{j=1}^m h_j(\mathbf{x})$.

The Random Forest is one of the best, most popular and easiest to use out-of-the-box classifier. There are two reasons for this:

- The RF only has two hyper-parameters, m and k. It is extremely *insensitive* to both of these. A good choice for k is $k=\sqrt{d}$ (where d denotes the number of features). You can set m as large as you can afford.
- Decision trees do not require a lot of preprocessing. For example, the features can be of different scale, magnitude, or slope. This can be highly advantageous in scenarios with heterogeneous data, for example the medical settings where features could be things like *blood pressure*, *age*, *gender*, ..., each of which is recorded in completely different units.

Useful variants of Random Forests:

- Split each training set into two partitions $D_l=D_l^A\cup D_l^B$, where $D_l^A\cap D_l^B=\emptyset$. Build the tree on D_l^A and estimate the leaf labels on D_l^B. You must stop splitting if a leaf has only a single point in D_l^B in it. This has the advantage that each tree and also the RF classifier become consistent.
- Do not grow each tree to its full depth, instead prune based on the leave out samples. This can further improve your bias/variance trade-off.