Bagging

Also known as Bootstrap Aggregating (Breiman 96). Bagging is an **ensemble** method.

Bagging Reduces 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 $h_D \to \bar{h}$.

Weak law of large numbers

The weak law of large numbers says (roughly) for i.i.d. random variables x_i with mean μ, we have,

$$\frac{1}{m}\sum_{i=1}^{m} x_i \to \bar{x} \text{ as } m \to \infty$$

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 becomes zero, *i.e.*
$\mathbb{E}[(h_D(x) - \bar{h}(x))^2] \to 0$

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 $Q((\mathbf{x_i}, y_i)) = \frac{1}{n} \qquad \forall (\mathbf{x_i}, y_i) \in D$
Draw D_i Q^n, i.e. $|D_i| = n$, and D_i is picked with replacement
Q: What is $\mathbb{E}[|D \cap D_i|]$?
Bagged classifier: $\hat{h}_D = \frac{1}{m}\sum_{i=1}^{m} h_{D_i}$
Notice: $\hat{h}_D = \frac{1}{m}\sum_{i=1}^{m} h_{D_i} \nrightarrow \bar{h}$(cannot use W.L.L.N here, W.L.L.N only works for i.i.d). However, in practice bagging still reduces variance very effectively.

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)

$$Q(X = x_i) = \underbrace{\sum_{k=1}^{n} \binom{n}{k} p_i^k (1-p_i)^{n-k}}_{\substack{\text{Probability that are} \\ \text{k copies of } x_i \text{ in D}}} \underbrace{\frac{k}{n}}_{\substack{\text{Probability} \\ \text{pick one of} \\ \text{these copies}}}$$

$$= \frac{1}{n} \underbrace{\sum_{k=1}^{n} \binom{n}{k} p_i^k (1-p_i)^{n-k} k}_{\substack{\text{Expected value of} \\ \text{Binomial Distribution} \\ \text{with parameter } p_i \\ \mathbb{E}[\mathbb{B}(p_i,n)]=np_i}}$$

$$= \frac{1}{n} n p_i$$

$$= p_i \leftarrow \underline{TATAAA}!! \text{ Each data set } D_l' \text{ is drawn from P, but not independently.}$$

Bagging summarized

1. Sample m data sets $D_1, \ldots, D_m$ 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
- Works well with many (high variance) classifiers
- Provides an unbiased estimate of the test error, which we refer to as the *out-of-bag error*. For each training point $(\mathbf{x}_i, y_i) \in D$ compute the out-of-bag error as the average error obtained on this training point from all the classifiers that were trained **without** it.
  For $(\mathbf{x}_i, y_i) \in D$, let

$$z_i = \sum_{\substack{D_j \\ (\mathbf{x}_i, y_i) \notin D_j}} \mathbf{1} \leftarrow \text{number of data sets w/o } (\mathbf{x}_i, y_i)$$

$$\epsilon_{\text{OOB}} = \sum_{(\mathbf{x}_1, y_1) \in D_j} \frac{1}{z_i} \sum_{\substack{D_l \\ (\mathbf{x}_i, y_i) \notin D_l}} l(h_{D_l}(\mathbf{x_i}), y_i)$$

- From the predictions of the individual classifiers in the bag we can estimate the variance of our ensemble. This can be very helpful to estimate the uncertainty with which the classifier makes a prediction. For example, if each one of the m classifiers agrees on the label the ensemble is very certain. On the other hand, is only 51% agree on the label but the other 49% disagree, the classifier would be very uncertain.

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 $D_1, \ldots, D_m$ 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.