Bagging

Bagging = Bootstrap aggregating.

Bootstrapping is a statistical technique that one to gather many alternative versions of the single statistic that would ordinarily be calculated from one sample.

Typical bootstrapping scenario. (case resampling) Given a sample $D$ of size $n$, a bootstrap sample of $D$ is a sample of $n$ data items drawn randomly with replacement from $D$.

Note: On average, about 63.2% of items from $D$ will be found in a bootstrapping sample, but some items will be found multiple times.

Bootstrap Aggregating for Supervised Learning. Let $D$ be a training set, $|D| = N$. We construct a bagging classifier for $D$ as follows:

Training Stage: Given $D$, $k$ and a learning algorithm BaseLearner:

1. Create $k$ bootstrapping replications $D_1, \ldots, D_k$ of $D$ by using case resampling bootstrapping technique.
2. For each bootstrapping replication $D_i$, create a classifier $f_i$ using the BaseLearner classification method.

Testing Stage: Given $f_1, \ldots, f_k$ and a test record $d$:

1. Compute $f_1(d), \ldots, f_k(d)$.
2. Assign as $\text{class}(d)$, the majority (plurality) class among $f_1(d), \ldots, f_k(d)$.
Algorithm AdaBoost($D$, BaseLearner, $k$) begin
forEach $d_i \in D$ do $D_i(i) = \frac{1}{|D|}$; for $t = 1$ to $k$ do //main loop
$f_t := \text{BaseLearner}(D_t)$;
$e_t := \sum_{d_i \in D_t} \text{class}(d_i) \neq f_t(d_i) \cdot D_t(i)$; // $f_t$ is constructed to minimize $e_t$
if $e_t > 0.5$ then // large error: redo
$k := k - 1$; break;
endif
$a_t := \frac{1}{2} \ln \frac{1 - e_t}{e_t}$; //reweighting parameter
forEach $d_i \in D$ do $D_{t+1}(i) := D_t(i) \cdot e^{-a_t \cdot \text{class}(d_i) \cdot f_t(d_i)}$; //reweigh each tuple in $D$
$\text{Norm}_t := \sum_{i=1}^{|D|} D_{t+1}(i)$;
forEach $d_i \in D$ do $D_{t+1}(i) := \frac{D_{t+1}(i)}{\text{Norm}_t}$; //normalize new weights
endfor
$\text{final}(.) := \text{sign}(\sum_{t=1}^k a_t \cdot f_t(.)$)
end

Figure 1: AdaBoost: an adaptive boosting algorithm. This version is for binary category variable $Y = \{-1, +1\}$.

Boosting

Boosting. Boosting is a collection of techniques that generate an ensemble of classifiers in a way that each new classifier tries to correct classification errors from the previous stage.

Idea. Boosting is applied to a specific classification algorithm called BaseLearner$^1$.

Each item $d \in D$ is assigned a weight. On first step, $w(d) = \frac{1}{|D|}$. On each step, a classifier $f_t$ is built. Any errors of classification, i.e., items $d \in D$, such that $f(d) \neq \text{class}(d)$ are given higher weight.

On the next step, the classification algorithm is made to “pay more attention” to items in $D$ with higher weight.

The final classifier is constructed by weighting the votes of $f_1, \ldots, f_k$ by their weighted classification error rate.

AdaBoost. The Adaptive Boosting algorithm $^2$ (AdaBoost) is shown in Figure ??.

Weak Classifiers. Some classifiers are designed to incorporate the weights of training set elements into consideration. But most, like C4.5, do not do so. In order to turn a classifier like C4.5 into a weak classifier suitable for AdaBoost, this classifier can be updated as follows:

- On step $t$, given the weighted training set $D_t$, we sample $D_t$ to build a training set $D'_t$. The sampling process uses $D_t(i)$ as the probability of selection of $d_i$ into $D'_t$ on each step.

$^1$It is also commonly called weak classifier.

$^2$It is also commonly called weak classifier.
Voting

When multiple classification algorithms $A_1, \ldots, A_k$ are available, **direct voting** can be used to combine these classifiers.

Let $D$ be a training set, and $f_1, \ldots, f_k$ are the classifiers produced by $A_1, \ldots, A_k$ respectively on $D$. Then the combined classifier $f$ is constructed to return the class label returned by the plurality of classifiers $f_1, \ldots, f_k$.

Random Forests

Random Forests[?] are an extension of bagging. A bagging technique resamples the training set with replacement, but keeps all attributes in the dataset "active" for each resampled training set.

Random Forests build a collection of decision trees, where each decision tree is built based on a subset of a training set and a subset of attributes.

In a nutshell, a Random Forests classifier works as follows:

1. **Input:** Let $D = \{d_1, \ldots, d_n\}$ be the training set, with $\text{class}(d_i)$ defined. Let $C = \{c_1, \ldots, c_k\}$ be the class attribute, and let $A = \{A_1, \ldots, A_N\}$ be the set of attributes for vectors from $D$, i.e., given $d \in D$, $d = (x_1, \ldots, x_M)$.

2. **Attribute selection parameter:** A number $m << M$ is fixed throughout the run of a random forest classifier. This number indicates how many attributes is selected to build each decision tree in a forest.

3. **Forest construction:** The classifier builds $N$ decision trees $T_1, \ldots, T_N$. Each decision tree is built by selecting a subsample of the training set, and a subset of the attributes.

4. **Single decision tree construction:** Decision tree $T_j$ is built as follows.

   (a) Build a set $D_j \subseteq D$ drawing random $k$ data points from $D$ with replacement.

   (b) Select $m$ random attributes $A^1_{j1}, \ldots, A^1_{jm}$ from $A$ without replacement.

   (c) Using a decision tree induction procedure (see below), build a decision tree $T_j$ for the training set $D_j$ restricted to attributes $A^1_{j1}, \ldots, A^1_{jm}$.

   Do not prune the decision trees.

5. **Classification process:** For each data point $d \in D$, (attempt to) classify $d$ by traversing trees $T_1, \ldots, T_N$ to discover classification decisions $c^1, \ldots, c^N$. Choose, as $\text{class}(d)$, the most frequently occurring in $c^1, \ldots, c^N$ class.

Caveats. A decision tree $T_j$ may not contain all possible values (paths) for some attribute. This means that some trees won’t be able to classify some of the data points in $D$. The simplest way to deal with this is to ignore.

Decision tree induction procedures. Both versions of ID3 (C4.5 without the pruning) and CART, a decision-tree induction algorithm that uses the Gini impurity instead of Information Gain-based measures, can be used.
**Gini impurity measure.** The Gini impurity measure quantifies how often a randomly chosen and randomly labelled data point from a training set will be mislabelled.

Let \( D = \{d_1, \ldots, d_n\} \) be a training set.

Let \( C = \{c_1, \ldots, c_k\} \) be a class variable.

Let \( D_i = \{d \in D | \text{class}(d) = c_i\} \) be the set of all training set points from category \( c_i \).

Let \( f_i = |D_i| \).

The Gini impurity measure \( I_G \) is defined as follows:

\[
I_G(D) = \sum_{i=1}^{k} f_i \cdot (1 - f_i) = \sum_{i=1}^{k} k f_i - \sum_{i=1}^{k} k f_i^2 = 1 - \sum_{i=1}^{k} k f_i^2 = \sum_{i \neq j} f_i \cdot f_j.
\]

**References**
