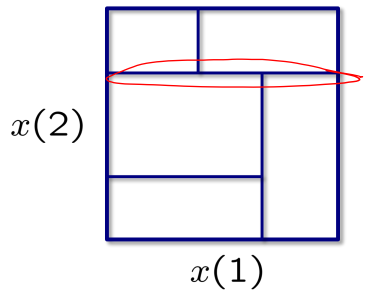
Binary decision trees

A binary decision tree ultimately boils down to taking a majority vote within each cell of a ***partition*** of the feature space (learned from the data) that looks something like this example in $\mathbb{R}^2$



Drawbacks

- Unstable
- "Jagged" decision boundaries

Ensemble methods

Ensemble methods address both of these deficiencies in decision trees as well as other algorithms

The first step is to generate a number of classifiers $f_1, \ldots, f_T$ (all using the same dataset) using some method that typically involves some degree of randomness

The second step is to combine these into a single classifier

Even if *none* of the individual classifiers are particularly good, the combined result can *far* outperform any of the individual classifiers and can be surprisingly effective

"The wisdom of the crowds"

Sir Francis Galton (1822-1911)

- cousin of Charles Darwin
- statistician (introduced correlation and standard deviation)
- father of eugenics… wary of democracy and distrustful of "the mob"

How much does this ox weigh?

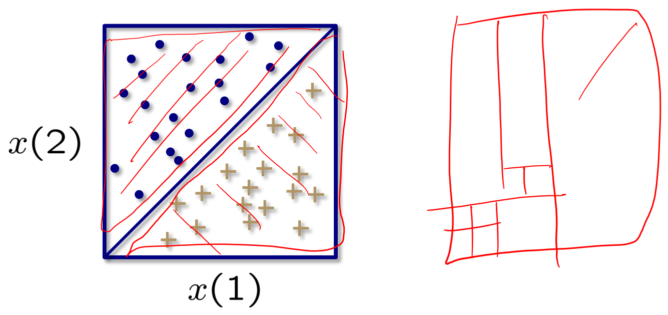
If we collect hundreds of uneducated farmers (with no particular expertise in weighing oxen), how well will they do?

Mean of the guesses: 1,197 pounds

Actual weight: 1,198 pounds

An example in classification

Suppose that the feature space is $[0, 1]^2$ and that the data looks like:



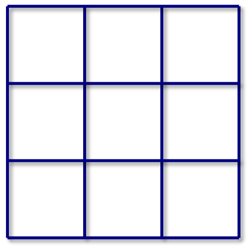
In this scenario, the Bayes risk is zero…

But the risk of certain simple classifiers can still be large

Histogram classifiers

Suppose that we are using ***histogram classifiers***

In particular, we are using classifiers based on a regular partition of $[0, 1]^2$ into 9 squares



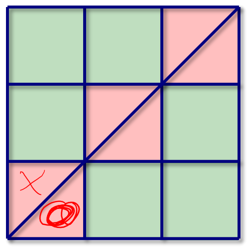
Label of each cell determined by majority vote

Histogram classifiers are pretty bad!

This classifier will not perform very well for the given distribution (or indeed, most distributions)

The risk of this classifier is:

$$R = \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{6}$$



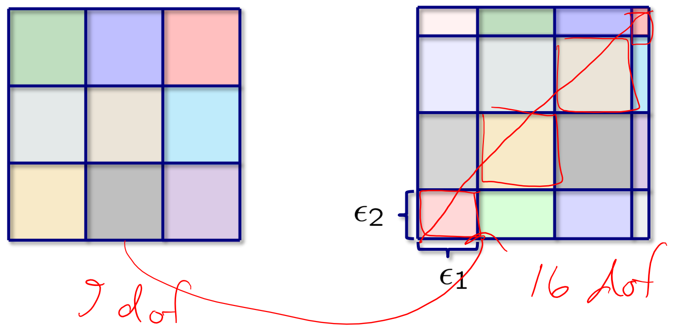
You can easily imagine that binary decision trees would have similar trouble with this example

However, we will see that with an appropriate ensemble method, we can make this classifier much more effective

Randomly shifted histogram classifiers

Suppose that we generate $\epsilon_1, \epsilon_2 \in [0, \frac{1}{3}]$ uniformly at random

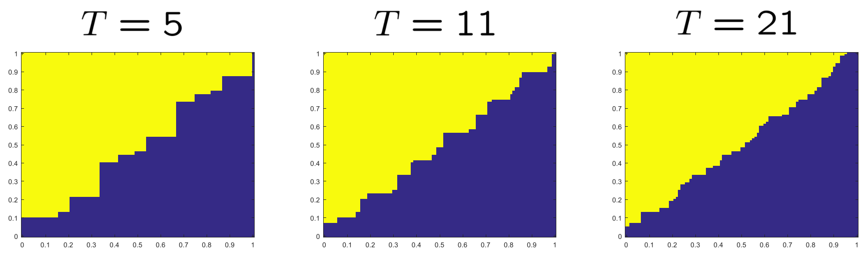
Then shift the partition by $[\epsilon_1, \epsilon_2]^T$



Ensemble histogram classifier

Generate $f_1, \dots, f_T$ as independent randomly shifted histogram classifiers and take majority vote

Example: $n = 1000$



Remarks

Not only is the ensemble method better performing…

It is also much more ***stable***

More formally, a machine learning method is ***stable*** if a small change in the input to the algorithm leads to a small change in the output (e.g., the learned classifier)

Decision trees are a primary example of an unstable classifier, and benefit considerably from ensemble methods

- more on this in a bit!

Notice that the main step in our approach above was to introduce some form of randomization into the algorithm…

Bagging

Another way to introduce some randomness is via ***bagging***

Bagging is short for ***bootstrap aggregation***

Given a training sample of size n, for $b = 1, \ldots, B$ let I_b be a list of size n obtained by sampling from $\underline{\{1, \ldots, n\}}$ ***with replacement***

Recall that I_b is called a ***bootstrap sample***

Suppose we have a fixed learning algorithm

Let f_b be the classifier we obtain by applying this learning algorithm to $\{(\mathbf{x}_i, y_i)\}_{i \in I_b}$

The ***bagging classifier*** is just the majority vote over $f_1, \ldots, f_B$

Random forests

A ***random forest*** is an ensemble of decision trees where each decision tree is (independently) randomized in some fashion

Bagging with decision trees is a simple example of a random forest

In the specific context of decision trees, bagging has one pretty big drawback

- bootstrap samples are highly correlated
- as a result, the different decision trees tend to select the same features as most informative
- this leads to partitions that tend to be highly correlated
- we would rather have partitions that are more “independent”

Random feature selection

One way to achieve this is to also incorporate ***random feature selection***

- generate an classifiers by choosing random subsets of features and designing decision trees on just those features
- can be combined with bagging

Random features lead to less correlated partitions, translating to a reduced variance for the ensemble prediction

Rule of thumb: use $\sqrt{d}$ random features

Random forests are possibly the best “off-the-shelf” method for classification

Approach also extends to regression

Boosting

Boosting is another ensemble method

Unlike previous ensemble methods, in boosting:

- the ensemble classifier is a *weighted* majority vote
- the elements of the ensemble are determined *sequentially*

Assume that the labels are ± 1

The final classifier has the form

$$f(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \underline{f_t(\mathbf{x})}\right)$$

where $\underline{f_1, \ldots, f_T}$ are called base classifiers and $\alpha_1, \ldots, \alpha_T$ are (positive) weights that reflect confidence in $f_1, \ldots, f_T$

Base learners

Let $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be the training data

Let $\mathcal{F}$ be a fixed set of classifiers called the ***base class***

A ***base learner*** for $\mathcal{F}$ is a rule that takes as input a set of weights $\underline{\mathbf{w}} = \underline{(w_1, \ldots, w_n)}$ satisfying $w_i \geq 0, \sum_i w_i = 1$ and outputs a classifier $f \in \mathcal{F}$ such that the ***weighted empirical risk***

$$\underline{\boxed{e_{\mathbf{w}}(f)} := \sum_{i=1}^{n} w_i 1_{f(\mathbf{x}_i) \neq y_i}(i)} \Leftarrow$$

is (approximately) minimized

Examples

Examples of base classifiers include:

- decision trees
- decision stumps (i.e., decision trees with a depth of 1)
- radial basis functions, i.e.,

$$f(x) = \pm\text{sign}\left(k_\sigma(\mathbf{x}, \mathbf{x}_i) + b\right)$$

where $b \in \mathbb{R}$ and k_σ is and rbf kernel

Note that the base classifiers can be extremely simple

In such cases, $e_{\mathbf{w}}(f)$ can be minimized by an exhaustive search over $\mathcal{F}$

For more complex classifiers (e.g., decision trees) the base learner can resample the training data (with replacement) according to $\mathbf{w}$ and then use standard learning algorithms

The boosting principle

The basic idea behind boosting is to learn $f_1, \ldots, f_T$ sequentially, where f_t is produced by the base learner given a weight vector $\mathbf{w}^t = (\underline{w_1^t}, \ldots, \underline{w_n^t})$

The weights are updated to place more emphasis on elements in the training set that are harder to classify

Thus the weight update rule should obey:

- **Downweight:** If $f_t(\mathbf{x}_i) = y_i$, set $w_i^{t+1} < w_i^t$
- **Upweight:** If $f_t(\mathbf{x}_i) \neq y_i$, set $\underline{w_i^{t+1} > w_i^t}$

Adaboost

Adaboost, short for ***adaptive boosting***, was the first concrete algorithm to successfully use the boosting principle

Algorithm

Input: $\{(\mathbf{x}_i, y_i)\}_{i=1}^n, T, \mathcal{F}$

Initialize: $\mathbf{w}^1 = (\frac{1}{n}, \ldots, \frac{1}{n})$

For $t = 1, \ldots, T$

- use the base learner to estimate f_t with $\mathbf{w}^t$ as input
- compute $\underline{r_t} = e_{\mathbf{w}^t}(f_t)$
- set $\alpha_t = \frac{1}{2} \log((1 - r_t)/r_t)$
- update the weights via $w_i^t \propto w_i^{t+1} e^{-\alpha_t y_i f_t(\mathbf{x}_i)}$

Output:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^T \alpha_t f_t(\mathbf{x})\right)$$

Weak learning

Adaboost can be justified by the following result

Theorem

Suppose that $r_t = \underline{e_{\mathbf{w}^t}}(f_t) \leq \frac{1}{2}$ and denote $\underline{\gamma_t = \frac{1}{2} - r_t}$

The training error of Adaboost satisfies

$$\frac{1}{n}\sum_{i=1}^n 1_{f(\mathbf{x}_i) \neq y_i}(i) \leq \exp\left(-2\sum_{t=1}^T \gamma_t^2\right)$$

In particular, if $\gamma_t \geq \underline{\gamma} > 0$ for all t, then

$$\frac{1}{n}\sum_{i=1}^n 1_{f(\mathbf{x}_i) \neq y_i}(i) \leq \exp\left(-2T\gamma^2\right)$$

Weak learning

The requirement that $\gamma_t \geq \gamma > 0$ is equivalent to requiring that $r_t < \frac{1}{2}$

This essentially means that all we require is that our base learner can do at least slightly better than random guessing

When this holds, our base learner is said to satisfy the ***weak learning hypothesis***

The theorem says that under the weak learning hypothesis, the Adaboost training error converges to 0 exponentially fast

To avoid overfitting, the parameter T should be chosen carefully (e.g., by cross-validation)

Remarks

- If $r_t = 0$, then $\alpha_t = \infty$
  - in other words, if there is a classifier in $\mathcal{F}$ that perfectly separates the data, Adaboost says to just use that classifier
- Adaboost can be interpreted as an iterative (gradient descent) algorithm for minimizing the empirical risk corresponding to the exponential loss
- By generalizing the loss, you can get different boosting algorithms with different properties
  - e.g., Logitboost