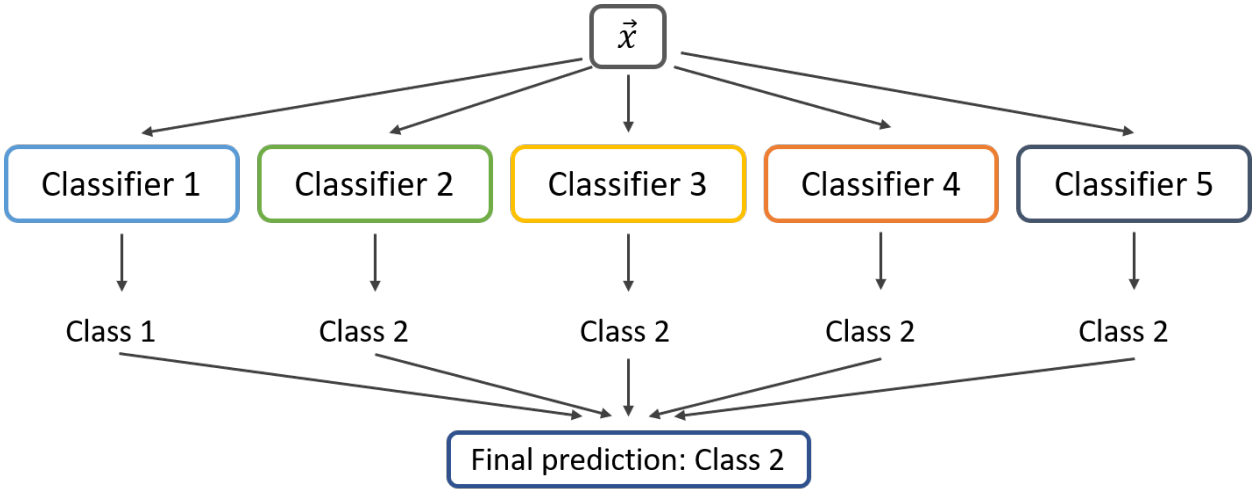
Machine Learning
An Algorithmic Perspective

Decision by Committee: Ensemble Learning (Chapter 13)

Decision by Committee: Ensemble Learning

- The old saying has it that two heads are better than one
- Even more heads are better -> decision by committee!
  - The basic idea is that by having lots of learners that each get slightly different results on a dataset—some learning certain things well and some learning others—and putting them together, the results that are generated will be significantly better than any one of them on its own (provided that you put them together well... otherwise the results could be significantly worse).



The basic idea of ensemble learning:

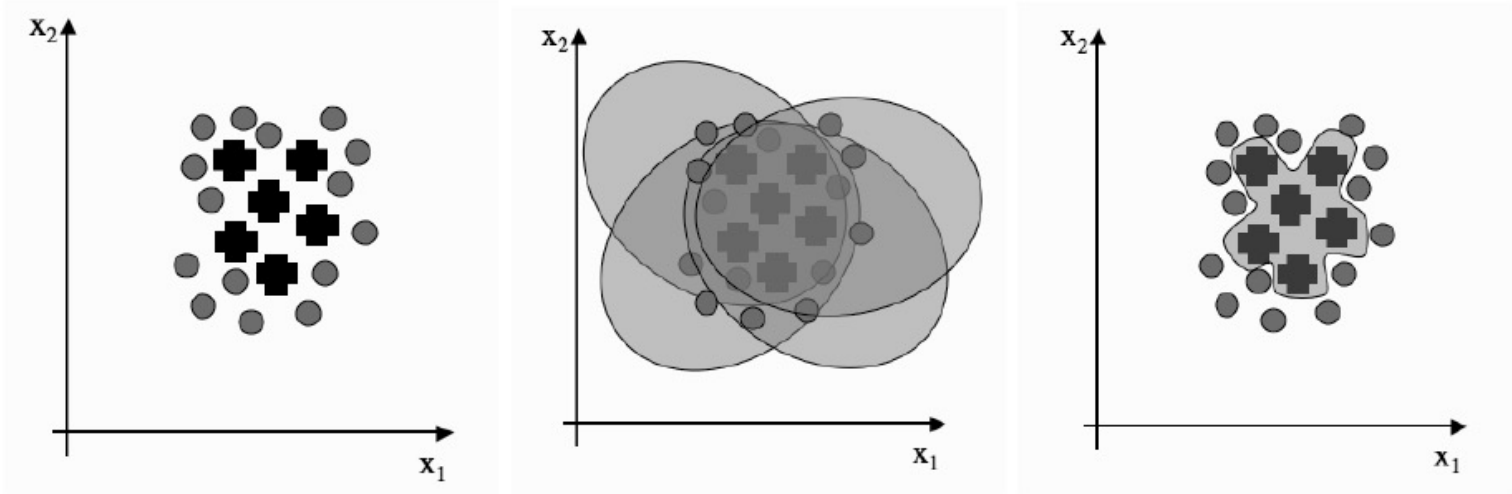


FIGURE 13.1 By combining lots of simple classifiers (here that simply put an elliptical decision boundary onto the data), the decision boundary can be made much more complicated, enabling the difficult separation of the pluses from the circles.

- In general, one type of classifiers is used at a time, but it not a rule.
- A common choice of classifier is the decision tree.

Ensemble methods: Boosting and Adaboost

- Boosting: If we take a collection of very poor learners (weak classifiers), each performing only just better than chance, then by putting them together it is possible to make a strong ensemble learner.
- Adaboost (adaptive boosting) : The innovation that AdaBoost uses is to give weights to each data point according to how difficult previous classifiers have found to get it correct.

Ensemble methods: Boosting and Adaboost

AdaBoost Algorithm

- Initialise all weights to $1/N$, where N is the number of datapoints
- While $0 < \epsilon_t < \frac{1}{2}$ (and $t < T$, some maximum number of iterations):
  - train classifier on $\{S, w^{(t)}\}$, getting hypotheses $h_t(x_n)$ for datapoints x_n
  - compute training error $\epsilon_t = \sum_{n=1}^{N} w_n^{(t)} I(y_n \neq h_t(x_n))$
  - set $\alpha_t = \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  - update weights using:

$$w_n^{(t+1)} = w_n^{(t)} \exp(\alpha_t I(y_n \neq h_t(x_n))/Z_t, \qquad (13.1)$$

    where Z_t is a normalisation constant
- Output $f(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

$I(y_n$ 6= $h_t(x_n))$:
1, target!=output
0, target==output

Ensemble methods: Boosting and Adaboost

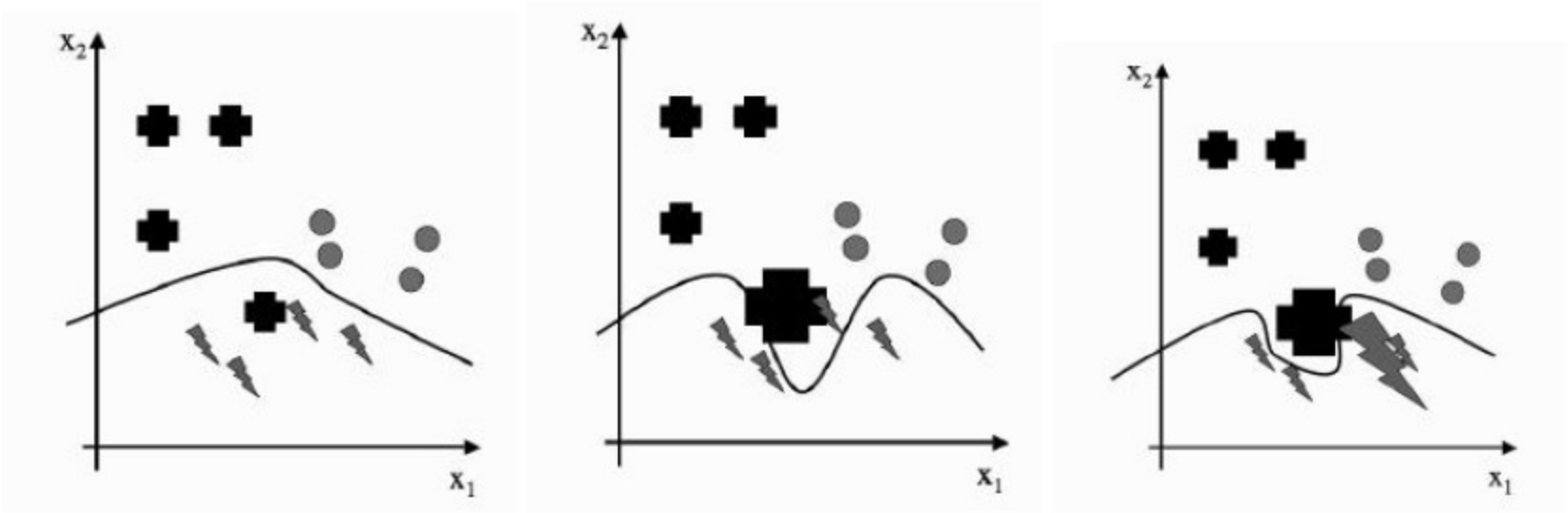


FIGURE 13.2 As points are misclassified, so their weights increase in boosting (shown by the datapoint getting larger), which makes the importance of those datapoints increase, making the classifiers pay more attention to them.

As a simple example of how boosting works:

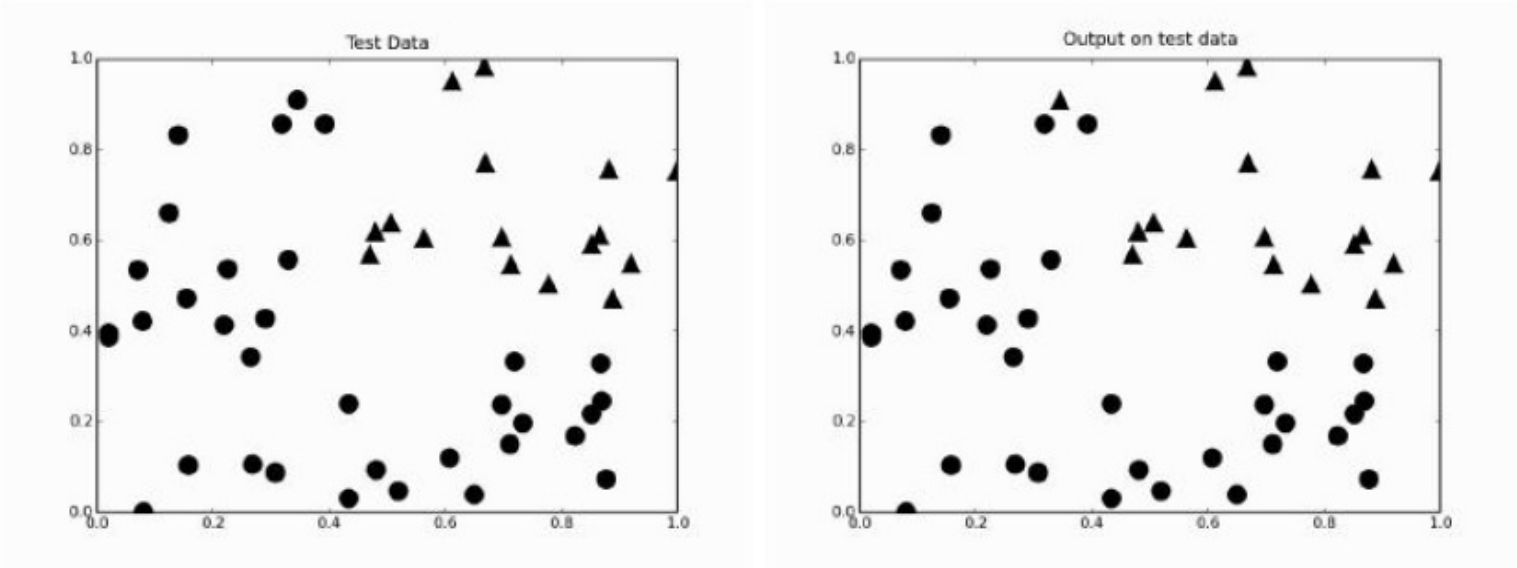


FIGURE 13.3 Boosting learns this simple dataset very successfully, producing an ensemble classifier that is rather more complicated than the simple horizontal or vertical line classifier that the algorithm boosts. On the independent test set shown here, the algorithm gets only 1 datapoint wrong, and that is one that is coincidentally close to one that was misclassified to simulate noise in the training data.

As a simple example of how boosting works:

- Impressive results! To better understand it, let's compute the loss function for AdaBoost:

$$G_t(\alpha) = \sum_{n=1}^{N} \exp\left(-y_n(\alpha h_t(x_n) + f_{t-1}(x_n))\right), \tag{13.2}$$

where $f_{t-1}(x_n)$ is the sum of the hypotheses of that datapoint from the previous iterations:

$$f_{t-1}(x_n) = \sum_{\tau=0}^{t-1} \alpha_\tau h_\tau(x_n). \tag{13.3}$$

Exponential loss functions are well behaved and robust to outliers. The weights $w^{(t)}$ in the algorithm are nothing more than the second term in Equation (13.2), which can therefore be rewritten as:

$$G_t(\alpha) = \sum_{n=1}^{N} w^{(t)} \exp\left(-y_n \alpha h_t(x_n)\right). \tag{13.4}$$

Ensemble methods: Boosting and Adaboost

- It is possible to choose **other loss functions** providing that they are differentiable, they will provide useful boosting-like algorithms which are collectively known as **arcing** algorithms (for adaptive reweighting and combining).

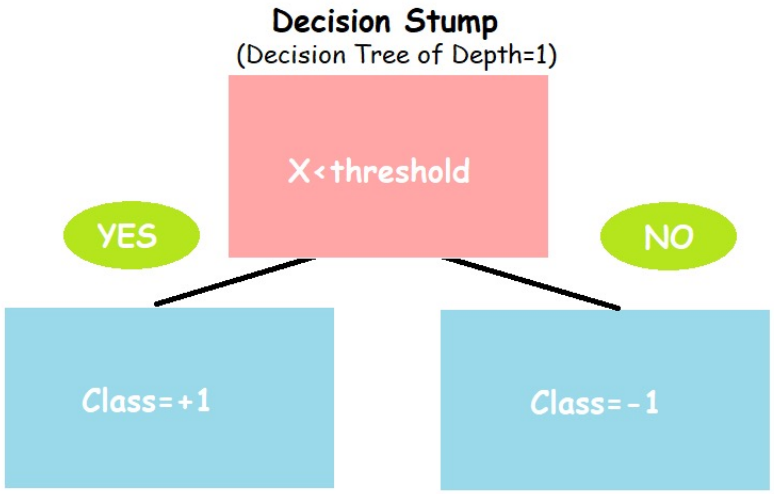
- AdaBoost can be modified to **perform regression** rather than classification (known as **real adaboost**, or sometimes **adaboost.R**).

- There is another **variant on boosting** (also called AdaBoost, confusingly) that uses the weights to sample from the full dataset, **training on a sample of the data** rather than the full weighted set, **with more difficult examples more likely to be in the training sample.**

Stumping

An extreme form of boosting that is applied to trees.

- The stump of a tree is the tiny piece that is left over when you chop off the rest.
- Stumping consists of simply taking the root of the tree and using that as a weak classifier (often worse than chance)
- But when ***boosting*** (using the weights to sort out **when that classifier should be used**, and **to what extent** as opposed to the other ones) the overall output of stumping can be very successful.



Ensemble methods: bagging (bootstrap aggregating)

- **Bagging:**
  - Is a way to decrease the variance in the prediction by generating additional data for training from dataset using **combinations with repetitions to produce multi-sets** of the original data.
  - Having taken a set of bootstrap samples, the bagging method simply requires that we **fit a model to each dataset**, **and then combine** them by taking the output to be the majority vote of all the classifiers.

- **Why?**
  - The benefit of it is that we will get lots of learners that perform slightly differently, which is exactly what we want for an ensemble method.

Ensemble methods: bagging (bootstrap aggregating)

A NumPy implementation could be:

```
# Compute bootstrap samples
samplePoints = np.random.randint(0,nPoints,(nPoints,nSamples))
classifiers = []

for i in range(nSamples):
 sample = []
 sampleTarget = []
 for j in range(nPoints):
  sample.append(data[samplePoints[j,i]])
  sampleTarget.append(targets[samplePoints[j,i]])
 # Train classifiers
 classifiers.append(self.tree.make_tree(sample,sampleTarget,features))
```

Subagging (‘subsample’ and ‘bagging’)

- It is the fairly obvious idea that you **don’t need to produce samples that are the same size as the original data**.
- If you make **smaller datasets**, then it makes sense to **sample without replacement.**
- It is common to use a dataset size that is half that of the original data, and the **results can be comparable to full bagging**.

Thank you