GI01/M055: Supervised Learning

7. Tree-based learning algorithms and Boosting

December 14, 2009

John Shawe-Taylor

Today’s plan

- Classification and regression trees (CART)
- Weak learners
- Adaboost
- Bounding generalisation of boosting
- Linear programming boosting

Bibliography: These lecture notes are available at:

http://www.cs.ucl.ac.uk/staff/J.Shawe-Taylor/courses/index-gi01.html

Lecture notes are based on Hastie, Tibshirani & Friedman, Chapters 8.3 and 9.2,9-4

Tree-based learning algorithms

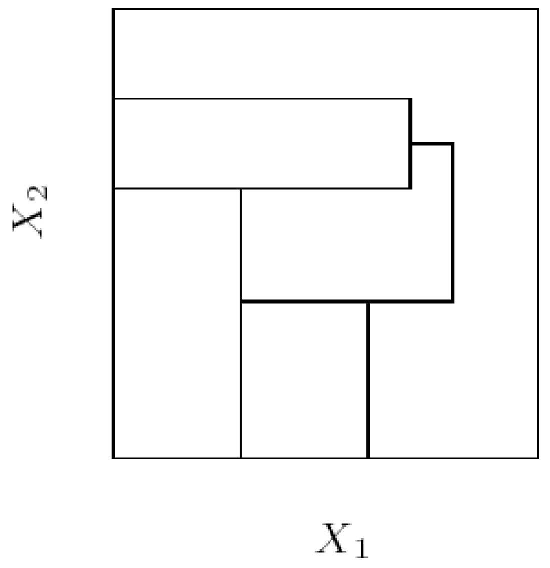
Tree methods attempt to partition the input space into a set of rectangles and fit a simple model (e.g. a constant) in each one

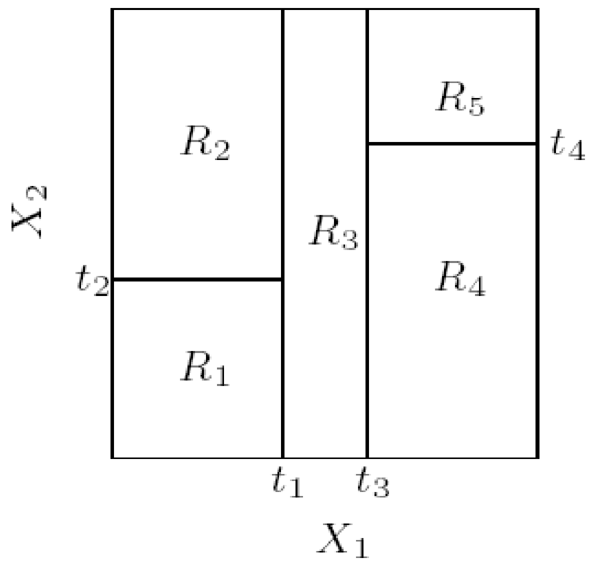
$$f(\mathbf{x}) = \sum_{n=1}^{N} c_n I\{\mathbf{x} \in R_n\}$$

- $\{R_n\}_{n=1}^N$: hyper-rectangles partitioning the input space: $\bigcup_{n=1}^N R_n = \mathbb{R}^d$, $R_n \cap R_\ell = \emptyset$
- $I\{\mathbf{x} \in R_n\} = 1$ if $\mathbf{x} \in R_n$ and 0 otherwise (indicator function)
- $\{c_n\}_{n=1}^N$: some real parameters. A natural choice is

$$c_n = \text{ave}(y_i|\mathbf{x}_i \in R_n) \equiv \frac{\sum_{i=1}^m y_i I\{\mathbf{x}_i \in R_n\}}{\sum_{i=1}^m I\{\mathbf{x}_i \in R_n\}}$$

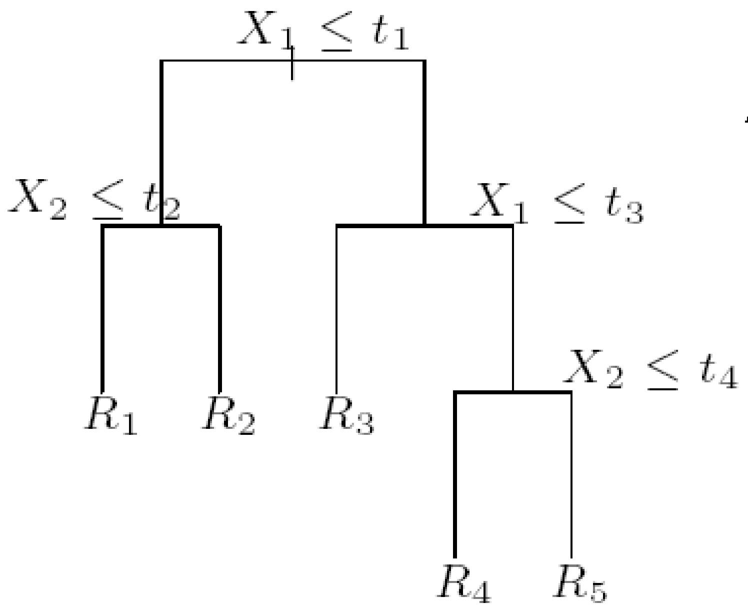
Recursive binary partitions

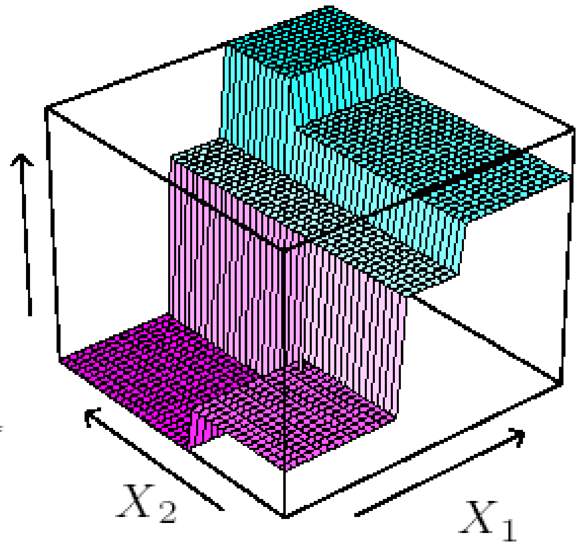




- Left plot: Each partition line has a simple description, yet partitions may be complicated to describe
- Right plot: recursive binary partition (first split the space into two regions then iterate in each region)

Tree representation





- Left plot: tree representation for the recursive binary partition above
- Right plot: prediction function $f(\mathbf{x}) = \sum_{n=1}^{5} c_n I\{\mathbf{x} \in R_n\}$

Some remarks

- A nice feature of a recursive binary tree is its **interpretability** (e.g. in medical science the tree stratifies the population into strata of high and low outcome on the basis of patient characteristics)

- Warning: risk for **overfitting**! (with enough splits the tree can fit arbitrarily well the data)

Key question: how to compute the tree (hence the regions R_n) and how to control overfitting?

Regression trees (I)

The algorithm needs to automatically decide on the splitting variables (1st or 2nd in above example) and split points. Recall that:

$$c_n = \mathsf{ave}(y_i|\mathbf{x}_i \in R_n)$$

which corresponds to minimizing the square error in region n

Ideally we would like to solve the problem

$$\min_{R_1,\ldots,R_N} \left\{ \sum_{i=1}^{m} \left(y_i - \sum_{n=1}^{N} \mathsf{ave}(y_j|\mathbf{x}_j \in R_n) I\{\mathbf{x}_i \in R_n\} \right)^2 \right\}$$

But it is generally computationally intractable. So we resort to an alternate heuristic procedure

Regression trees (II)

Let’s find the first split in the tree

Define the pair of axis parallel half-planes:

$$R_1(j,s) = \{\mathbf{x} | x_j \le s\}, \quad R_2(j,s) = \{\mathbf{x} | x_j > s\}$$

and search for optimal values $\bar{j}$ and $\bar{s}$ which solve the problem

$$\min_{j,s} \left\{ \min_{c_1} \sum_{\mathbf{x}_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{\mathbf{x}_i \in R_2(j,s)} (y_i - c_2)^2 \right\}$$

Regression trees (III)

$$\min_{j,s}\left\{\min_{c_1}\sum_{\mathbf{x}_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{\mathbf{x}_i\in R_2(j,s)}(y_i-c_2)^2\right\}$$

- The inner minimization is solved by

$$\bar{c}_1=\text{ave}(y_i|\mathbf{x}_i\in R_1(j,s))\qquad \bar{c}_2=\text{ave}(y_i|\mathbf{x}_i\in R_2(j,s))$$

- For each splitting variable j the search for the best split point s can be done in $O(m)$ computations (can split between any two training points)

Thus the problem is solved in $O(dm)$ computations

Regression trees (IV)

In order to build the tree we recursively assign training points to each obtained region and repeat the above steps in each one

If we do not stop the process we clearly overfit the data! So, when should we stop it?

- follow a split only if it decreases the empirical error more than a threshold? No, there might be better splits below a“bad” node

- when a maximum depth of the tree is reached? No, this could underfit or overfit. We need to look at the data to determine a good size of the tree

Regression trees (V)

We choose the tree **adaptively** from the data:

- first grow a large tree $\hat{T}$ (stopping when a minimum number of data (e.g. 5) is assigned at each node)
- then prune the tree using a **cost-complexity pruning**, i.e. look for $T_\lambda \subseteq \hat{T}$ which minimizes

  $$C_\lambda(T) = \sum_{n=1}^{|T|} m_n Q_n(T) + \lambda |T|$$

  where T is a subtree of $\hat{T}$, n runs over leaf nodes of T (a subset of the nodes of $\hat{T}$) m_n is the number of data assigned to node n and $Q_n(T) = \frac{1}{m_n} \sum_{\mathbf{x}_i \in R_n} (y_i - c_n)^2$ (so the first term in C_λ is just the training error)
- Can show that there is a unique $T_\lambda \subseteq \hat{T}$ which minimizes C_λ

Regression trees (VI)

A C_λ is a kind of regularized empirical error: $T_0 = \hat{T}$, the original tree. Large values of λ result in smaller trees. A good value of λ can be obtained by cross validation

- **weakest link pruning**: successively collapse the internal nodes that produce the smallest per node increase in $\sum_{n=1}^{|T|} m_n Q_n(T)$ and continue till the root (single-node) tree is produced

- search along the produced list of trees for the one which minimizes C_λ

One can show that T_λ is in the produced list of subtrees, hence this algorithm gives the optimal solution

Classification trees (I)

When the output is a categorical variable (say $\mathcal{Y} = \{1, \ldots, K\}$) we use the same algorithm above with two important modifications

- For each region R_n we defined the empirical class probabilities:

$$p_{nk} = \frac{1}{m_n} \sum_{\mathbf{x}_i \in R_n} I(y_i = k)$$

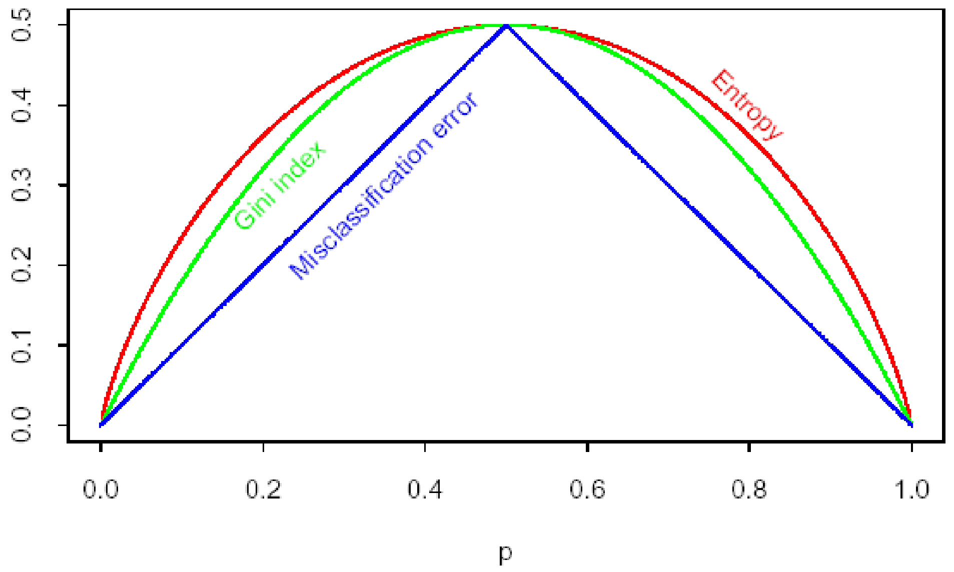
- We classify an input which falls in region n in the class with maximum probability

$$f(\mathbf{x}) = \mathrm{argmax}_{k=1}^{K} \sum_{n=1}^{N} p_{nk} I\{\mathbf{x} \in R_n\}$$

- We use different measures $Q_n(T)$ of node impurity

Classification trees (II)

- Misclassification error:
  $1 - p_{nk(n)}$,
  $k(n) := \text{argmax}_{k=1}^{K} p_{nk}$
- Gini index:
  $\sum_k p_{nk}(1 - p_{nk})$
- Cross entropy:
  $\sum_k p_{nk} \log p_{nk}$



Cross entropy or Gini index are used to grow the tree (they are more sensitive in changes in the node probabilities). Misc. error is used to prune it.

Spam email classification problem

Consider the problem of classifying an email you receive as “good email” or “spam email”

- Different features can be used to represent an email
  e.g. the bag of words representation (we may also use additional features to code the text in the subject section of the email etc.)
- Different learning methods (Naive Bayes, SVM, k-NN, CART, etc.) can be used to approximate this task...
- *Key observation:* it is easy to find “*rules of thumb*” that are often correct (say 60% of the time) e.g. “IF ‘business’ occurs in email THEN predict as spam”
- Hard to find highly accurate prediction rules

Weak learners and boosting

Weak learner: an algorithm which can consistently find classifiers (“rules of thumb”) at least slightly better than random guessing, say better than 55%

Boosting: a general method of converting rough rules of thumb into a highly accurate prediction rule (classifier)

Boosting algorithm

- Devise a computer program for deriving rough rules of thumb
- Choose rules of thumb to fit a subset of example
- Repeat T times
- Combine the classifiers by weighted majority vote

key steps are:

- how do we choose the subset of examples at each round? concentrate on **hardest examples** (those most often misclassified by previous classifiers)
- how do we combine the weak learners? by **weighted majority**

Adaboost algorithm

Given training data $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$
Initialize $D_1(i) = \frac{1}{m}$

For $t = 1, \ldots, T$:

- fit a classifier $h_t : \mathbb{R}^d \to \mathbb{R}$ using distribution D_t
- choose $\alpha_t \in \mathbb{R}$
- update

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}}{Z_t} \qquad (*)$$

  where Z_t is a normalization factor (so as to ensure that D_{t+1} is a distribution)

Output the final classifier $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$

Some remarks

- The basic idea in Adaboost is to maintain a distribution D on the training set and iteratively train a weak learner on it
- At each round larger weights are assigned to hard examples, hence the weak learner will focus mostly on those examples

Let ϵ_t be the weighted training error of classifier t:

$$\epsilon_t = \sum_{i=1}^{m} D_t(i) I\{h_t(\mathbf{x}) \neq y_i\}$$

We will justify later the following choice for α_t

$$\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t} \qquad (1)$$

Weight by majority

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

The final classifier is a weighted majority vote of the T base classifiers where α_t is the weight assigned to h_t

$$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t$$

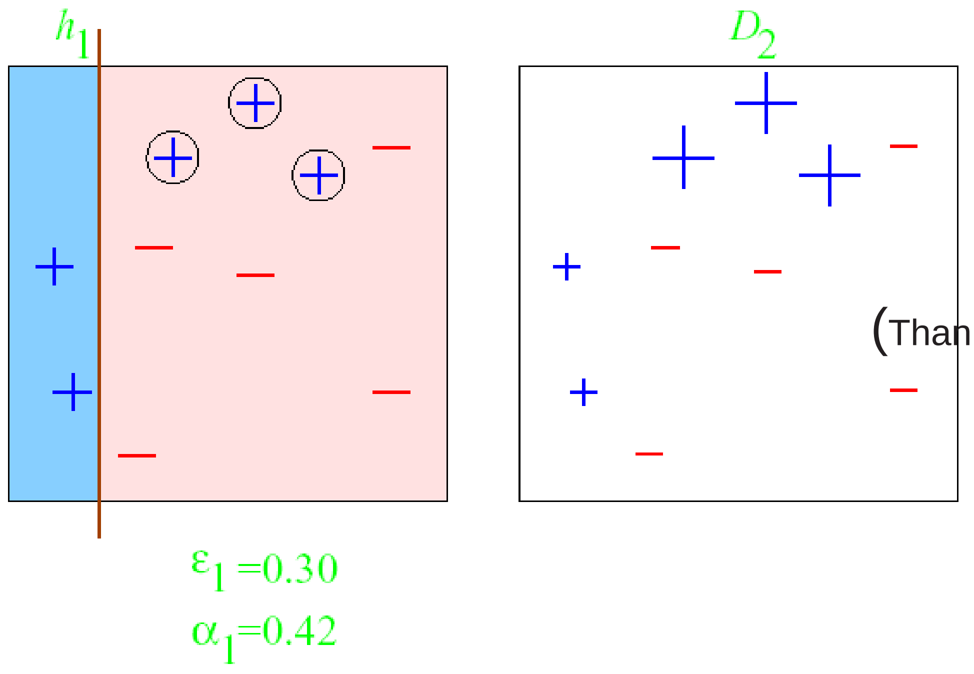
Typically $\epsilon_t \leq 0.5$ hence $\alpha_t \geq 0$ (this is always the case if h_t and $-h_t$ are both in the set of weak learners, e.g. for decision stumps)

Thus, f is essentially a convex combination of the h_t with weights controlled by the training error

Example (round 1)

Let's discuss a simple example where the weak learners are decision stumps (vertical or horizontal lines, i.e. trees with only two leaves)

the 2nd plot highlights the difficult examples

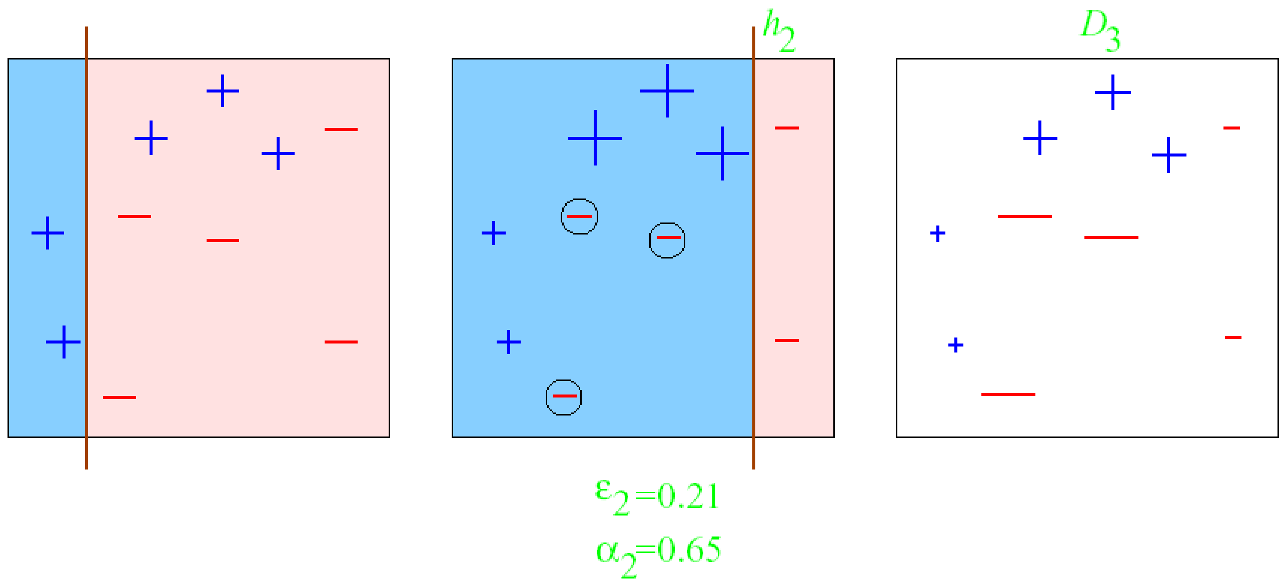


(Thanks to Rob Schapire for providing the figures for this example)

Example (round 2)

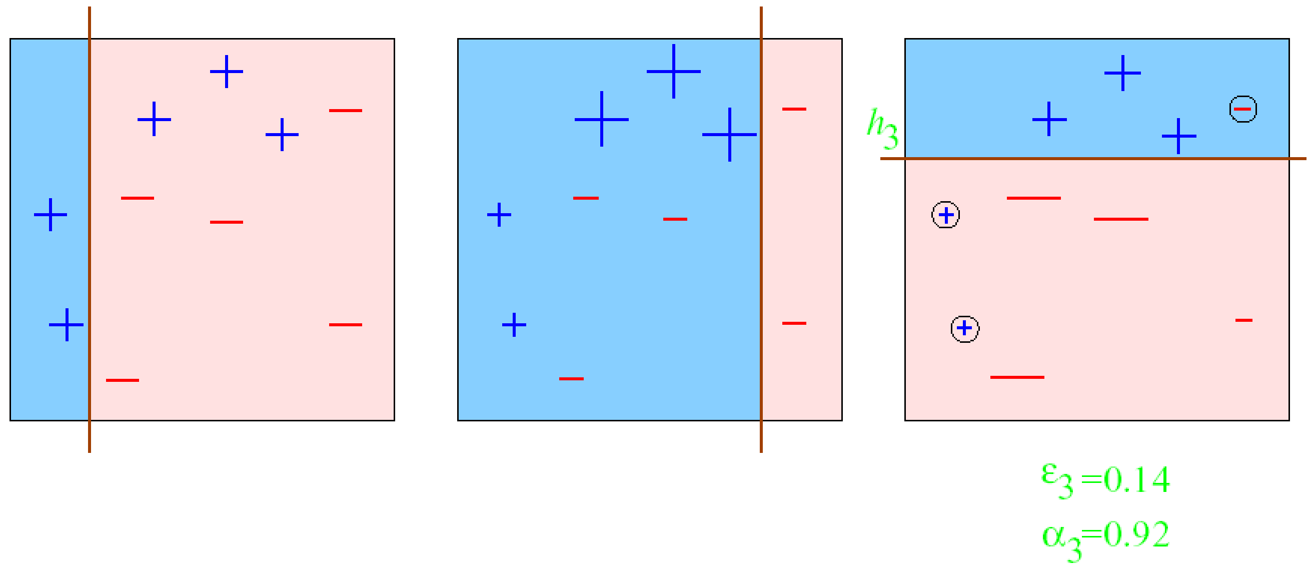
The second classifier concentrates more on the difficult examples

Examples are then re-weighted according to this classifier



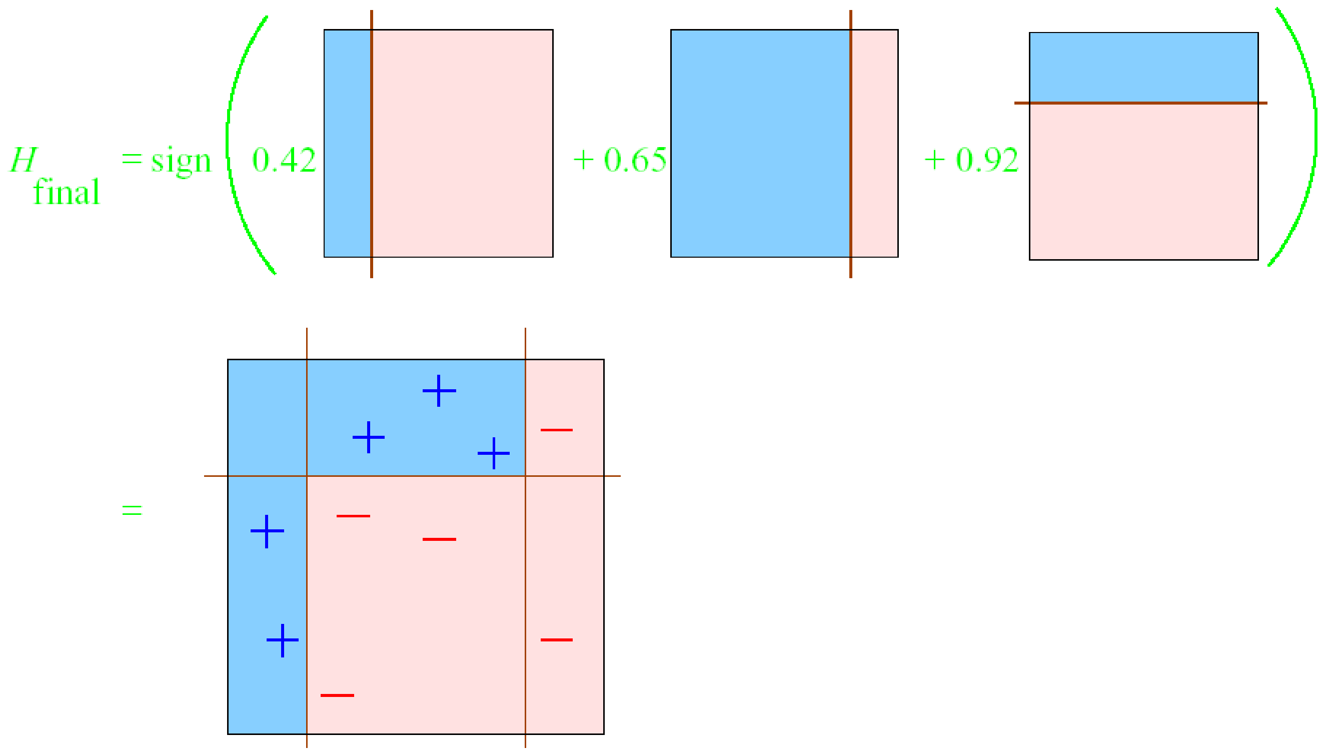
Example (round 3)

In this example the set of weak learners can be assumed to be finite. There are roughly $4m$ weak learners/decision stumps corresponding to choosing either the horizontal or vertical coordinate and splitting between any two points



Example (final classifier)

The final classifier has zero training error!


$H_{\text{final}} = \text{sign}\Big(0.42 \quad + 0.65 \quad + 0.92 \Big)$
=

Analysis of the training error (I)

The training error of the boosting algorithm is bounded as

$$\frac{1}{m}\sum_{i=1}^{m} I\{H(\mathbf{x}_i) \neq y_i\} \leq \frac{1}{m}\sum_{i=1}^{m} e^{-y_i f(\mathbf{x}_i)} = \prod_{t=1}^{T} Z_t$$

where we have defined $f = \sum_t \alpha_t h_t$ (so that $H(\mathbf{x}) = \mathrm{sign}(f(\mathbf{x}))$)

The inequality follows from: $H(\mathbf{x}_i) \neq y_i \;\Rightarrow\; e^{-y_i f(\mathbf{x}_i)} \geq 1$

The equality follows from the recursive definition of D_t (see also page 18)

Analysis of the training error (II)

The previous bound suggests that if at each iteration we choose α_t and h_t by minimizing Z_t the final training error of H will be reduced most rapidly

Generally we choose $h_t : \mathbb{R}^d \to \mathbb{R}$ (margin classifiers). However, if we constrain h_t to have range $\{-1, 1\}$ (so these are binary classifiers), Z_t is minimized by the choice in equation (1) above

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

We show this next

Analysis of the training error (III)

From formula (*) on page 18 we have

$$Z_t = \sum_i D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}$$

Using the fact that h_t returns binary values we also have that

$$\begin{aligned} Z_t &= e^{\alpha_t} \sum_{i: y_i \neq h_t(\mathbf{x}_i)} D_t(i) + e^{-\alpha_t} \sum_{i: y_i = h_t(\mathbf{x}_i)} D_t(i) \\ &= \epsilon_t e^{\alpha_t} + (1 - \epsilon_t) e^{-\alpha_t} \end{aligned}$$

Equation (1) now easy follows by solving $\frac{dZ_t}{d\alpha_t} = 0$

Analysis of the training error (IV)

Placing $\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$ in the above formula for Z_t we obtain

$$Z_t = \epsilon_t e^{\alpha_t} + (1-\epsilon_t)e^{-\alpha_t} = 2\sqrt{\epsilon_t(1-\epsilon_t)} = \sqrt{1-4\gamma_t^2}$$

where $\gamma_t = 1/2 - \epsilon_t$. Hence we have the following bound for the training error

$$Error \leq \prod_t Z_t = \prod_t \sqrt{1-4\gamma_t^2} \leq e^{-2\sum_t \gamma_t^2}$$

Thus, if each weak classifier is slightly better than random guessing (if $\gamma_t \geq \gamma > 0$) the training error drops **exponentially fast**

$$Error \leq e^{-2T\gamma^2}$$

Additive models and boosting

Boosting can be seen as a greedy way to solve the problem

$$\min_{\mathbf{w}} \sum_{i=1}^{m} V(y_i, \mathbf{w}^\top \phi(\mathbf{x}_i))$$

where the feature vector $\phi = (\phi_1, \phi_2, \ldots, \phi_N)$ is formed only by weak learners

At each iteration a new basis function is added to the current basis expansion $f^{(t-1)} = \sum_{s=1}^{t-1} \alpha_s h_s$

$$(\alpha_t, n(t)) = \mathrm{argmin}_{\alpha,n} \sum_{i=1}^{m} V(y_i, f^{(t-1)}(\mathbf{x}_i) + \alpha\phi_n(\mathbf{x}_i))$$

and $h_t = \phi_{n(t)}$. This is unlike in CART, where at each iteration previous basis function coefficients are re-adjusted

Additive models and boosting

$$\min_{\alpha_t, n} \sum_{i=1}^{m} V(y_i, f^{(t-1)}(\mathbf{x}_i) + \alpha_t \phi_n(\mathbf{x}_i))$$

In the statistical literature, this type of algorithm is called *forward stagewise additive model*

It is also possible to establish a relation between boosting and L^1 norm regularization

$$\min_{\mathbf{w}} \sum_{i=1}^{m} V(y_i, \mathbf{w}^\top \phi(\mathbf{x}_i)) + \lambda \sum_{n=1}^{N} |w_n|$$

which says that essentially boosting maximizes the L^1 margin $(\sum_{n=1}^{N} |w_n|)^{-1}$

Why the exponential loss?

We will show when V is the exponential loss that

$$e^{-y_i\left(f^{(t-1)}(\mathbf{x}_i)+\alpha_t h_t(\mathbf{x}_i)\right)} \propto D_t(i)e^{-y_i\alpha_t h_t(\mathbf{x}_i)}$$

In fact, using formula (*) on page 18 recursively it is easy to see that

$$D_t(i) = \frac{e^{-y_i f^{(t-1)}(\mathbf{x}_i)}}{m\prod_{s=1}^{t-1} Z_s}$$

from which, summing over i, it follows that

$$D_t(i) = \frac{e^{-y_i f^{(t-1)}(\mathbf{x}_i)}}{\sum_{j=1}^{m} e^{-y_j f^{(t-1)}(\mathbf{x}_j)}}$$

Note: as a special case we have that $\prod_{t=1}^{T} Z_t = \frac{1}{m}\sum_{i=1}^{m} e^{-y_i f(\mathbf{x}_i)}$ showing the equality on page 25

Why the exponential loss? (cont.)

The formula

$$e^{-y_i\left(f^{(t-1)}(\mathbf{x}_i)+\alpha_t h_t(\mathbf{x}_i)\right)} \propto D_t(i)e^{-y_i\alpha_t h_t(\mathbf{x}_i)}$$

implies that

$$\begin{aligned}\sum_{i=1}^{m} e^{-y_i\left(f^{(t-1)}(\mathbf{x}_i)+\alpha_t h_t(\mathbf{x}_i)\right)} &\propto \sum_{i=1}^{m} D_t(i)e^{-y_i\alpha_t h_t(\mathbf{x}_i)} \\ &= (e^{\alpha_t} - e^{-\alpha_t})\epsilon_t(h_t) + e^{-\alpha_t}\end{aligned}$$

where as before $\epsilon_t(h_t) = \sum_{i=1}^{m} D_t(i) I\{h_t(\mathbf{x}_i) \neq y_i\}$

Hence, for a fixed value of $\alpha_t > 0$, minimizing the exponential loss w.r.t. h_t is the same as minimizing w.r.t. the weighted misclassification error!

Summarizing Adaboost

In summary, Adaboost can be interpreted as a greedy way to minimize the exponential loss criterion via the forward-stagewise additive model approach

Alternatively, let $\{h_n\}_{n=1}^N$ the set of all weak learners (assume they are finite in number). Adaboost minimizes

$$\sum_{i=1}^{m} e^{-y_i \sum_{n=1}^{N} w_n \phi_n(\mathbf{x}_i)}$$

by coordinate descent, at each iteration t choosing coordinate $h_t = \phi_{n(t)}$ which produces the biggest decrease in error and updating $w_{n(t)} = \alpha_t$

Boosting and logistic regression

The expected error w.r.t. the exponential loss

$$\mathbf{E}_{\mathbf{x},y}\left[e^{-yf(\mathbf{x})}\right] = \mathbf{E}_{\mathbf{x}}\left[P(y=1|\mathbf{x})e^{-f(\mathbf{x})} + P(y=-1|\mathbf{x})e^{f(\mathbf{x})}\right]$$

is minimized (exercise) for

$$f^*(x) = \frac{1}{2}\log\frac{P(y=1|\mathbf{x})}{P(y=-1|\mathbf{x})} \quad \Rightarrow \quad P(y=1|\mathbf{x}) = \frac{1}{1+e^{-2f^*(\mathbf{x})}}$$

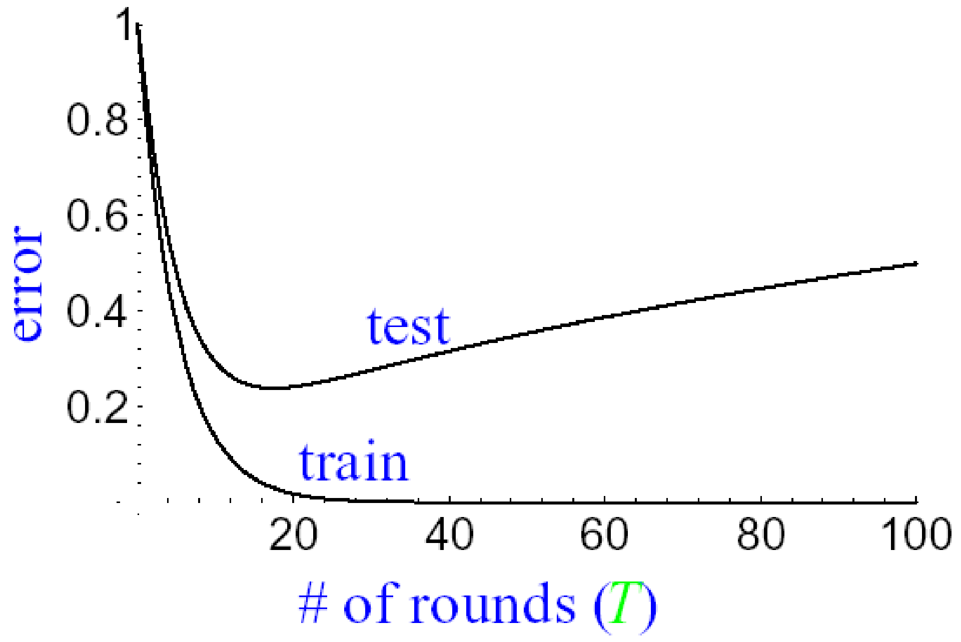
The last formula can be used to convert the output of Adaboost into a probability estimate

Generalization error (I)

How do we expect training and test error of boosting to depend on T?

At first sight, we'd expect that if T is too large overfitting kicks off

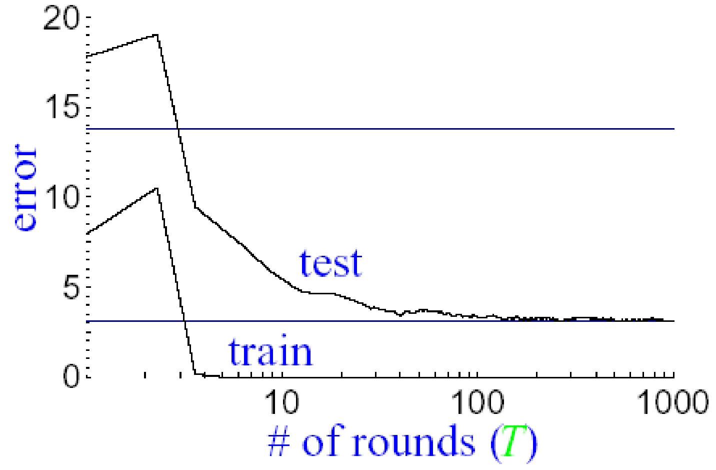
However, the plot shown here is not typically the case...

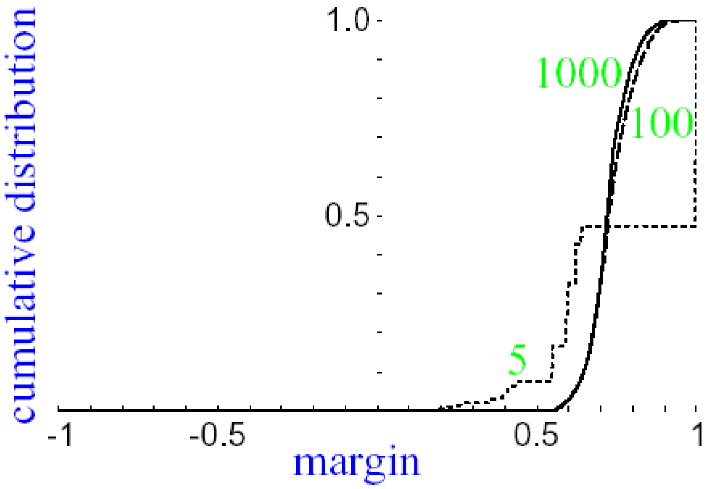


(Figures from Schapire *et. al*)

Generalization error (II)

Typically, the test error keeps decreasing even when the training error is zero! (eventually it will increase but may take many more iterations to do so)





However the **margin distribution** keeps decreasing as well which explains why test error does so

Generalization error (III)

The margin of point i is simply the quantity $\mathsf{margin}_i = y_i f(\mathbf{x}_i)$ where we assume that $\sum_t \alpha_{t=1}^T = 1$ (just normalize the weights after the last iteration of boosting).

The margin distribution is the empirical distribution of the margin

One can show that with high probability (say 99%)

$$\mathsf{generalization\ error} \leq \mathsf{Pr}_{\mathsf{emp}}(\mathsf{margin} \leq \theta) + O\left(\frac{\sqrt{d/m}}{\theta}\right)$$

where m is the number of training points, d the VC-dimension of the set of weak learners ($\log N$ in our case) and $\mathsf{Pr}_{\mathsf{emp}}$ denotes empirical probability of the margin (like the training error this converges exponentially fast to zero)

Generalization error (IV)

- A clearer bound is obtained using Rademacher complexity an alternative measure of function class complexity derived from its ability to fit to random ± 1 noise:

$$R_m(\mathcal{H}) = \mathbb{E}_S \mathbb{E}_{\sigma \in \{-1,+1\}^m} \left[\frac{2}{m} \sup_{h \in \mathcal{H}} \sum_{i=1}^{m} \sigma_i h(x_i) \right]$$

where we assume the class $\mathcal{H}$ the class is closed under negation.

Generalization error (V)

Using this definition we can bound the generalisation in terms of the margin distribution as with SVMs

$$\text{generalization error} \leq \sum_{i=1}^{m} \xi_i + BR_m(\mathcal{H}) + 2\sqrt{\frac{\log(1/\delta)}{2m}} \qquad (1)$$

where $\mathcal{H}$ is the class of weak learners with range $[-1, 1]$ and $B = \sum_{i=1}^{T} \alpha_i$.

Note that this no longer assumes the normalisation of the weights as for the previous bound, while the ξ_i, the margin slack variable, are computed as

$$\xi_i = \left(1 - y_i \sum_{j=1}^{N} \alpha_j h_j(x_i)\right)_+$$

Linear programming machine

- The previous bound suggests an optimisation similar to that of SVMs.
- seeks linear function in a feature space defined explicitly.
- For example using the 1-norm it seeks $\mathbf{w}$ to solve

$$\begin{array}{ll} \text{min}_{\mathbf{w},b,\xi} & \|\mathbf{w}\|_1 + C\sum_{i=1}^m \xi_i \\ \text{subject to} & y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right) \geq 1 - \xi_i,\ \xi_i \geq 0, \\ & i = 1, \ldots, m. \end{array}$$

Linear programming boosting

- Very slight generalisation considers the features as a set $\mathbf{H}_{ij}$ of ‘weak’ learners (and include the constant function as one weak learner and negative of each weak learner):

$$\begin{array}{ll} \min_{\mathbf{a},\xi} & \|\mathbf{a}\|_1 + C\sum_{i=1}^m \xi_i \\ \text{subject to} & y_i\mathbf{H}_i\mathbf{a} \geq 1 - \xi_i,\ \xi_i \geq 0,\ a_j \geq 0 \\ & i = 1, \ldots, m. \end{array}$$

Alternative version

- Can explicitly optimise margin with 1-norm fixed:

$$\begin{array}{ll} \max_{\rho,\mathbf{a},\xi} & \rho - D\sum_{i=1}^{m}\xi_i \\ \text{subject to} & y_i\mathbf{H}_i\mathbf{a} \geq \rho - \xi_i,\ \xi_i \geq 0, a_j \geq 0 \\ & \sum_{j=1}^{N} a_j = 1. \end{array}$$

- Dual has the following form:

$$\begin{array}{ll} \min_{\beta,\mathbf{u}} & \beta \\ \text{subject to} & \sum_{i=1}^{m} u_i y_i \mathbf{H}_{ij} \leq \beta,\ j = 1, \ldots, N, \\ & \sum_{i=1}^{m} u_i = 1,\ 0 \leq u_i \leq D. \end{array}$$

Column generation

Can solve the dual linear programme using an iterative method:

```
1   initialise $u_i = 1/m, i = 1, \ldots, m$, $\beta = \infty$, $J = \emptyset$
2   choose $j^\star$ that maximises $f(j) = \sum_{i=1}^{m} u_i y_i \mathbf{H}_{ij}$
3   if $f(j^\star) \leq \beta$ solve primal restricted to $J$ and exit
4   $J = J \cup \{j^\star\}$
5   Solve dual linear programme restricted to set $J$ to give $u_i, \beta$
6   Go to 2
```

- Note that u_i is a distribution on the examples
- Each j added acts like an additional weak learner
- $f(j)$ is simply the weighted classification accuracy
- Hence gives 'boosting' algorithm - with previous weights updated satisfying error bound (1)
- Guaranteed convergence and soft stopping criteria

LASSO Method

- L_1 regularised least squares regression is known as LASSO (Least Absolute Shrinkage and Selection Operator)

- $$\begin{array}{ll} \min_{\mathbf{a},\xi} & \|\mathbf{a}\|_1 + C\sum_{i=1}^{m} \xi_i^2 \\ \text{subject to} & y_i - \mathbf{H}_i\mathbf{a} = \xi_i, \\ & i = 1, \ldots, m,\ a_j \geq 0. \end{array}$$

- Again leads to many of the coefficients a_j being set to 0

dual LARS (Least Angle Regression) algorithm

- Lagrangian:

$$L(\mathbf{a}, \beta) = \sum_{j=1}^{N} a_j + \sum_{i=1}^{m} (y_i - \mathbf{H}_i \mathbf{a})^2 + \sum_{j=1}^{N} \beta_j a_j$$

Implying $$1 - 2C \sum_{i=1}^{m} (y_i - \mathbf{H}_i \mathbf{a}) H_{ij} + \beta_j = 0$$

- Using KT conditions: if $a_j \neq 0$ then $\beta_j = 0$ and so for index set I

$$\mathbf{H}^T \mathbf{H}[I, I] \mathbf{a}_I = \mathbf{H}^T [I, :] \mathbf{y} - \frac{1}{2C} \mathbf{j}[I]$$

with $\mathbf{H}^T \mathbf{H}[j, I] \mathbf{a}_I - \mathbf{H}^T [I, j] \mathbf{y} + \frac{1}{2C} \geq 0$ when $j \in I$

Sample exam question

a) Explain how Adaboost uses the distribution D_t that weights the examples $((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$ to choose a weak learner from a set H at stage t.

b) If the input space is $\Re^d$, describe the set of decision stumps often used as weak learners. How many decision stumps need to be considered at each stage to evaluate all achievable training set behaviours?

Sample exam question (cont)

c) The Adaboost algorithm reweights the ith example at iteration t using the update

$$D_t(i) \propto D_{t-1}(i) \exp\left(-y_i \alpha_t h_t(\mathbf{x}_i)\right)$$

with $D_0(i) = 1/m$ for all i, where h_t is the weak learner selected with coefficient $\alpha_t > 0$. Give an intuitive justification for this update.

d) Show that the distribution $D_t(i)$ is proportional to the exponential of the margin of the function

$$f_t(\mathbf{x}) = \sum_{j=1}^{t} \alpha_j h_j(\mathbf{x}).$$

.