3. Tree-based learning and ensemble methods

COMP0078: Supervised Learning

Mark Herbster

15 October 2018

University College London
Department of Computer Science

Acknowledgments and References

Thanks

Thanks to Massi Pontil and John Shawe-Taylor for many of the slides.

Today

- Classification and regression trees (CART)
- Ensemble Learning
- Bagging
- Adaboost

Part I

Tree-based learning algorithms

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

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

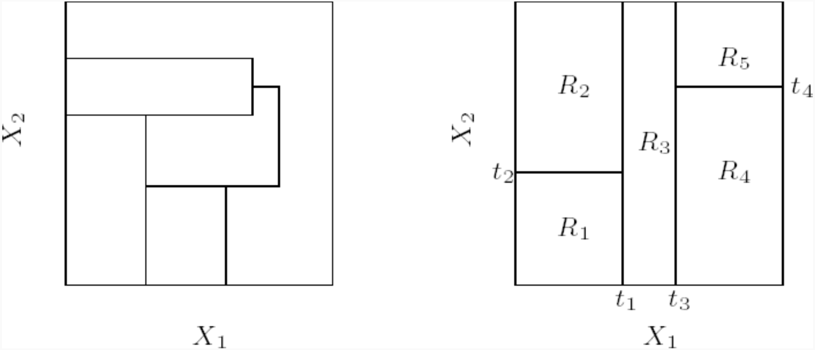
- We partition the input space with hyper-rectangles

  $$R_1, R_2, \ldots, R_N$$

  where $\bigcup_{n=1}^{N} R_n = \mathbf{R}^d$ and $R_a \cap R_b = \emptyset$ if $a \neq b$.
- Define $\mathcal{I}[\mathbf{x} \in R_n] = 1$ if $\mathbf{x} \in R_n$ and 0 otherwise (indicator function)
- $\{c_n\}_{n=1}^{N}$: some real parameters. A natural choice is

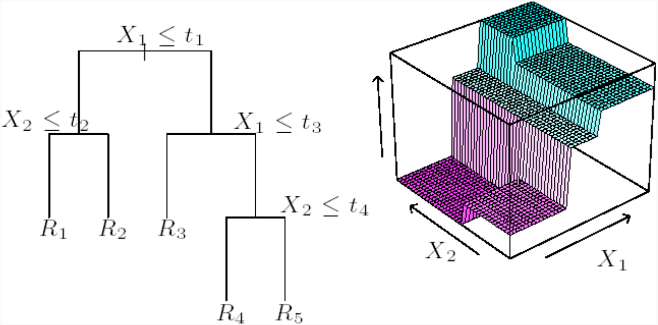
  $$c_n = \mathrm{ave}(y_i | \mathbf{x}_i \in R_n) \equiv \frac{\sum_{i=1}^{m} y_i \mathcal{I}[\mathbf{x}_i \in R_n]}{\sum_{i=1}^{m} \mathcal{I}[\mathbf{x}_i \in R_n]}$$

Recursive binary partitions



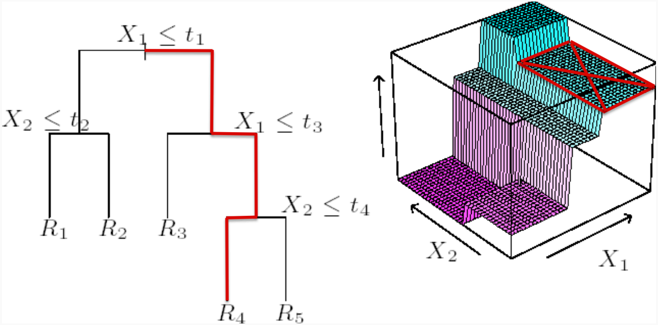
- Left plot: Each partition line has a simple description, yet partitions may be complicated to describe
- Right plot: recursive binary partition (first split the space into two regions then iterate in each region)

Tree representation – 1



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

Tree representation – 2



- Observe that each rectangle is defined by a conjunction.
- Thus

$$R_4 := \{\mathbf{x} : (x_1 > t_1) \wedge (x_1 > t_3) \wedge (x_2 \leq t_4)\}$$

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:

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

which corresponds to minimizing the square error in region n

Ideally we would like to solve the problem

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

But it is appears to be 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:

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

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

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

- For each splitting variable j the search for the best split point s can be done in $O(m)$ computations (a split may occur 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 maximum number of data (e.g. 5) is assigned at each node)
- then prune the tree using a **cost-complexity pruning**, i.e. look for a subtree $T_\lambda \subseteq \hat{T}$ 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 points 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)

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.

Final steps:

1. Use cross-validation to select optimal λ^*
2. Retrain on all the data and prune to find T_{λ^*}.

Classification trees (I)

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

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

$$p_{nk} = \frac{1}{m_n} \sum_{\mathbf{x}_i \in R_n} \mathcal{I}[y_i = k]$$

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

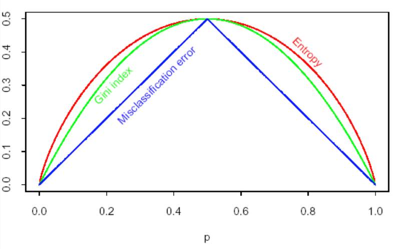
$$f(\mathbf{x}) = \mathrm{argmax}_{k=1}^{K} \sum_{n=1}^{N} p_{nk} \mathcal{I}[\mathbf{x} \in R_n]$$

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

Classification trees (II)

Node impurity measures:

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

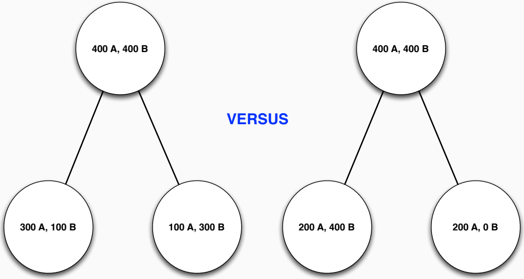


note: cross entropy scaled by 1/2 in figure

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

Classification trees (III)

Why use Gini or Entropy to as a splitting rule to grow the tree?



Splitting a node with $m_n = 800$ and two classes $\{A, B\}$.

Let's look at the reduction in $\sum_{n=1}^{|T|} m_n Q_n(T)$ for the left versus the right split for the entropy and misclassification impurity measures.

1. Misclassification: Left : $200 = (400 \times 1/4) + (400 \times 1/4)$; Right: $200 = (600 \times 1/3) + (200 \times 0)$
2. Entropy: Left: $649.02 = 400 \times (\frac{1}{4} \log 4 + \frac{3}{4} \log \frac{4}{3}) \times 2$;

   Right: $550.98 = 600 \times (\frac{1}{3} \log 3 + \frac{2}{3} \log \frac{3}{2}) + 200 \times 0$

Note that the right split is likely preferable.

Part II

Bagging and Boosting

Ensemble methods and the wisdom of the crowd – 1

In 1785 Marquis de Condorcet published, *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. In this work he observed among other results that if the probability of a voter being correct is larger than chance then the more voters the greater the likelihood that the majority is correct only increases. In the following we argue this quantitatively.

Marquis de Condorcet

Ensemble methods and the wisdom of the crowd – 2

A motivation:

• A single individual might often be wrong but the crowd majority may often be correct.

• An idealised scenario ...

• Suppose each individual in the crowd $h_1, h_2, \ldots, h_{2T+1}$ of size $2T+1$ predicts the outcome correctly with probability $\frac{1}{2}+\gamma$ independently from one another.

Now conside the "vote" of the crowd.

$$H_T := \text{sign}\left(\sum_{t=1}^{2T+1} h_t\right)$$

What is the probability that H_T will be wrong?

Computing the probability – 1

We have,

$$P(H_T \text{ is wrong}) = \sum_{i=0}^{T} \binom{2T+1}{i} \left(\frac{1}{2}+\gamma\right)^{i} \left(\frac{1}{2}-\gamma\right)^{2T+1-i}$$

Hard to evaluate exactly, let use an upper bound.

Chernoff bound

Let $X_1, X_2, \ldots, X_n$ be independent random variables. Assume $0 \le X_i \le 1$. Let $X := \sum_{i=1}^{n} X_i$ Let $\mu := E[X] := \sum_{i=1}^{n} E[X_i]$. Then for all $0 \le k \le \mu$,

$$P(X \le k) \le \exp\left(-\frac{(\mu-k)^2}{2\mu}\right)$$

Now, let $k := T$, let $n := 2T+1$ thus $\mu := (2T+1)(\frac{1}{2}+\gamma) = T + \frac{1}{2} + 2T\gamma + \gamma$.

Computing the probability – 2

Now, let $k := T$, let $n := 2T + 1$ thus $\mu := (2T+1)(\frac{1}{2} + \gamma) = T + \frac{1}{2} + 2T\gamma + \gamma$.
Now subsituting into the bound,

$$\begin{aligned}
P(H_t \text{ is wrong}) &\leq \exp\left(-\frac{(\mu - T)^2}{2\mu}\right) \\
&= \exp\left(-\frac{(\frac{1}{2} + 2T\gamma + \gamma)^2}{2(T + \frac{1}{2} + 2T\gamma + \gamma)}\right) \\
&\leq \exp\left(-\frac{4T^2\gamma^2}{5T}\right) \\
&\leq \exp\left(-\frac{4\gamma^2}{5} T\right)
\end{aligned}$$

- Bound is crude! But observe the probability that the crowd is wrong exponentially decays to zero.

Part II-A

Bagging

Bagging Algorithm

Idea : Reduce the *variance* of a classifier by training many variants of the classifiers and then voting.

Inputs:

1. training data $\mathcal{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\} \subset \mathbf{R}^d \times \{-1, 1\}$
2. ensemble size T
3. resample size M
4. classifier function $h_{\mathcal{S}}(\mathbf{x})$

Bagging algorithm:

```
1. **for** $t = 1$ to $T$ do
2.     $\mathcal{S}(t) := M$ examples sampled with replacement from $\mathcal{S}$
3. **end for**
4. **return**
       $$H(\mathbf{x}) := \operatorname{sign}\left(\sum_{t=1}^{T} h_{\mathcal{S}(t)}(\mathbf{x})\right)$$
```

Conventionally set $M = m$.

How often do examples appear in the bag?

Consider the conventional advice to set $M = m$, then "roughly" how many unique examples from $\mathcal{S}$ are in a "bag" $\mathcal{S}(t)$?

- The probability that **a particular** example does not appear in bag $\mathcal{S}(t)$ is $(1 - \frac{1}{m})^m$.
- Note that

$$\lim_{m \to \infty} \left(1 - \frac{1}{m}\right)^m = \frac{1}{e} \approx 0.368$$

- Thus **roughly** there will be 63% distinct examples in each $\mathcal{S}(t)$.

Random forests

Idea : Observe for the “wisdom of the crowds” argument to work the predictors need to be independent.

- Thus although decision trees are “high variance” and thus ideal for bagging.
- Let’s go further and try to make the trees more de-correlated.
- When building the tree for each split only check a subset of size k features.
- k is usually set to $\sqrt{d}$ or $\log d$.

Useful in practice.

Part II-B

Boosting

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 Glossary

- $D_t(i)$: the weight on example i at time t where $\sum_{i=1}^{m} D_t(i) = 1$.
- α_t : the weight on weak learner t where $\alpha_t \in \mathbf{R}$.
- $h_t(\cdot)$: is the weak learner "generated" at time t which is a function from $\mathbf{R}^d \rightarrow \{-1, 1\}$.
- $f(\cdot)$: $f(\mathbf{x}) := \sum_{i=1}^{T} \alpha_t h_t(\mathbf{x})$.
- $H(\cdot)$: $H(\mathbf{x}) := \text{sign}(f(\mathbf{x}))$. (Final classifier output by adaboost)
- ϵ_t : "weighted error of weak learner $h_t(\cdot)$ at time t"

$$\epsilon_t := \sum_{i=1}^{m} D_t(i)\, \mathcal{I}[h_t(\mathbf{x}_i) \neq y_i]$$

- weak learner generator : given input

$$D_t(1), \ldots, D_t(m), (\mathbf{x}_1, y_1) \ldots, (\mathbf{x}_m, y_m)$$

  outputs a weaker learner $h_t(\cdot)$ such that $\epsilon_t < \frac{1}{2}$.

Adaboost algorithm

Input: training data $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$
Initialise: $D_1(1) = \cdots = D_1(m) = \frac{1}{m}$
For $t = 1, \ldots, T$:

1. fit a classifier $h_t : \mathbf{R}^d \to \{-1, 1\}$ using distribution D_t
2. choose $\alpha_t \in \mathbf{R}$ (see (1) in following slide)
3. update: for each $i \in \{1, \ldots, m\}$

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

Where Z_t is a normalization factor (so as to ensure that D_{t+1} is a distribution, i.e. $\sum_{i=1}^{m} D_{t+1}(i) = 1$.)

Return: the final classifier

$$H(\mathbf{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$$

Some remarks

Terminology:

- 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:

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

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

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

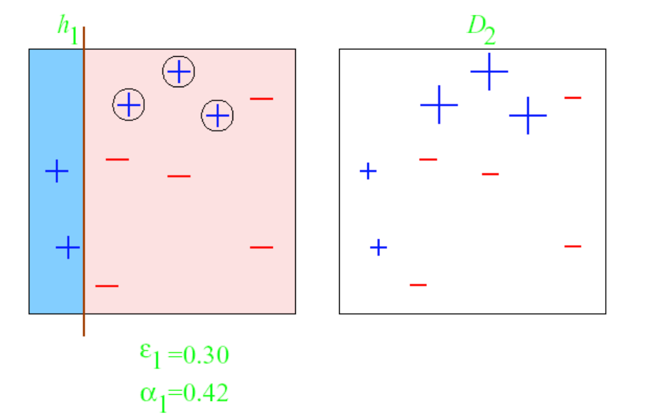
$$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t$$

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 linear 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)

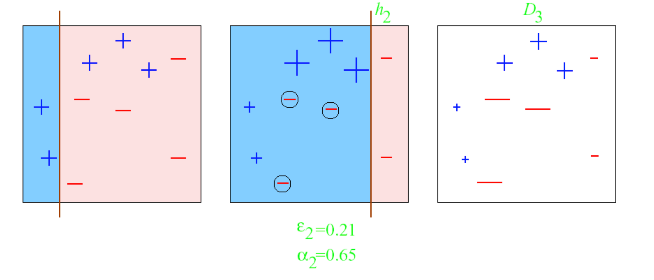


1. The 1st plot illustrates the misclassified examples
2. The 2nd plot illustrates how the difficult gain more weight

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

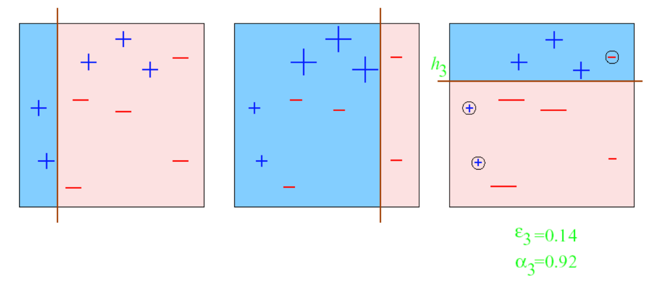
Example (round 2)

- The second classifier concentrates more on the difficult examples.
- Examples are then re-weighted according to this classifier



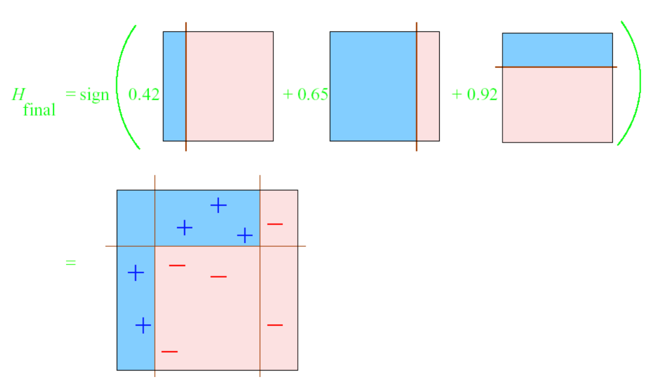
Example (round 3)

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 round)

The final classifier has zero training error!



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

Boosting Converges

Theorem

Given a training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ and assume that each iteration of Adaboost a weaker learner returns a hypothesis with weighted error $\epsilon_t \leq \frac{1}{2} - \gamma$. Then the training error of the output hypothesis of Adaboost is at most

$$\frac{1}{m}\sum_{i=1}^{m}\mathcal{I}[H(\mathbf{x}_i) \neq y_i] \leq \exp(-2\gamma^2 T)$$

We argue this over the next five slides.

Analysis of the training error (I)

The training error of the boosting algorithm is bounded as

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

where we have defined

$$f := \sum_t \alpha_t h_t \qquad \big(\text{so that } H(\mathbf{x}) = \operatorname{sign}(f(\mathbf{x}))\big)$$

1. The inequality follows from:

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

2. The equality follows from the recursive definition of D_t

Analysis of the training error (II)

Expanding on point 2:

$$D_{T+1}(i) = \frac{1}{m} \frac{\prod_{t=1}^{T} e^{-\alpha_t y_i h_t(\mathbf{x}_i)}}{\prod_{t=1}^{T} Z_t}$$

and expanding

$$\begin{aligned}
\frac{1}{m}\sum_{i=1}^{m} e^{-y_i f(\mathbf{x}_i)} &= \frac{1}{m}\sum_{i=1}^{m} e^{-y_i \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}_i)} \\
&= \frac{1}{m}\sum_{i=1}^{m}\prod_{t=1}^{T} e^{-y_i \alpha_t h_t(\mathbf{x}_i)} \\
&= \sum_{i=1}^{m} D_{T+1}(i) \prod_{t=1}^{T} Z_t \\
&= \prod_{t=1}^{T} Z_t \qquad \square
\end{aligned}$$

Analysis of the training error (III)

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

Since h_t maps to $\{-1, 1\}$ (they are binary classifiers) then Z_t is minimized by the choice of equation (1) above

$$\text{Recall (1):} \quad \alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

We show this next

Analysis of the training error (IV)

From formula (*) 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 follows by solving $\frac{dZ_t}{d\alpha_t} = 0$

Analysis of the training error (V)

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

$$\frac{1}{m}\sum_{i=1}^{m} \mathcal{I}[H(\mathbf{x}_i) \neq y_i] \leq \prod_t Z_t = \prod_t \sqrt{1-4\gamma_t^2} \leq e^{-2\sum_t \gamma_t^2}$$

In the final inequality we used the fact $1 - x \leq e^{-x}$. Thus, if each weak classifier is slightly better than random guessing (if $\gamma_t \geq \gamma > 0$) the training error drops **exponentially fast**

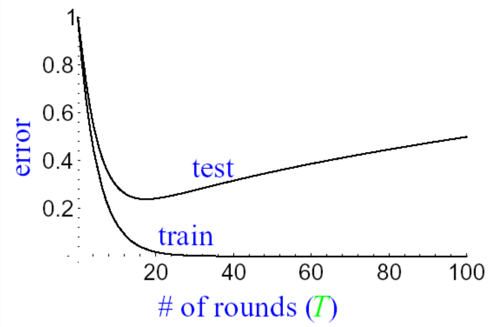
$$\text{training error} \leq e^{-2T\gamma^2}$$

Generalization error (I)

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

At first thought, we might expect that if T is too large that we'd overfit.

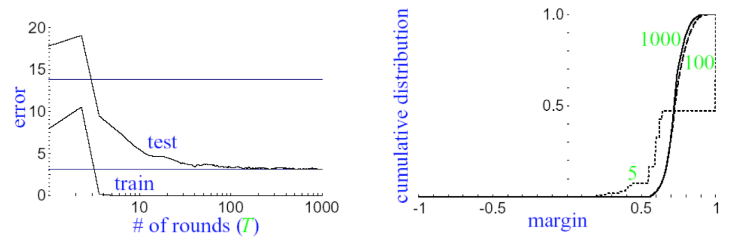
Thus possibly a curve like the following.



(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** "concentrates" as well which explains why test error improves.

Generalization error (III) [not examined 18/19]

The margin of point i is simply the quantity $\mathrm{margin}_i = y_i f(\mathbf{x}_i)$ 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%)

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

where m is the number of training points, vc is the VC-dimension of the set of weak learners and $\mathrm{Pr}_{\mathrm{emp}}$ denotes empirical probability of the margin (like the training error this converges exponentially fast to zero)
Note: that for the hypothesis class of decision stumps in $\mathbf{R}^d$ that

$$\mathsf{vc} \leq 2(\log d + 1)$$

Motivating Adaboost

A motivation for adaboost

Additive models and boosting (I)

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

$$\min \left\{ \sum_{i=1}^{m} V\left(y_i, \sum_{i=1}^{T} \alpha_t h_t(\mathbf{x}_i)\right) : \alpha_1, \ldots, \alpha_T \in \mathbf{R}^T, h_1, \ldots, h_T \in \mathcal{H}^T \right\}$$

where $\mathcal{H}$ is the hypothesis class which contain the "weak" learners. And the loss function is exponential loss as we shall see $V(y, \hat{y}) = \exp(-y\hat{y})$.

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

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

- This is unlike in CART, where at each iteration previous basis function coefficients are re-adjusted.

Additive models and boosting (II)

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

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

To derive adaboost, substitute in $V(y, \hat{y}) = \exp(-y\hat{y})$ and minimise. Thus,

$$\min_{\alpha_t, h_t} \sum_{i=1}^{m} e^{-y_i \left(f^{(t-1)}(\mathbf{x}_i) + \alpha_t h_t(\mathbf{x}_i) \right)}$$

Define $\mathcal{D}_t(i) := e^{-y_i f^{(t-1)}(\mathbf{x}_i)}$ and hence we have

$$\min_{\alpha_t, h_t} \sum_{i=1}^{m} \mathcal{D}_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}$$

Additive models and boosting (III)

Observe that (w.l.o.g.) that the minimum of the below with respect to h_t

$$\min_{\alpha_t, h_t} \sum_{i=1}^{m} \mathcal{D}_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}$$

is independent of positive α_t, i.e., rewriting the above as,

$$\min_{\alpha_t, h_t} \left(e^{\alpha_t} \sum_{i: y_i \neq h_t(\mathbf{x}_i)} \mathcal{D}_t(i) + e^{-\alpha_t} \sum_{i: y_i = h_t(\mathbf{x}_i)} \mathcal{D}_t(i) \right)$$

and then

$$\min_{\alpha_t, h_t} \left((e^{\alpha_t} - e^{-\alpha_t}) \sum_{i=1}^{m} \mathcal{D}_t(i) \, \mathcal{I}[y_i \neq h_t(\mathbf{x}_i)] + e^{-\alpha_t} \sum_{i=1}^{m} \mathcal{D}_t(i) \right)$$

Now we can see that we have derived adaboost! As

1. The h_t that minimises the above minimises the misclassification error weighted by $\mathcal{D}_t$
2. The $\mathcal{D}_t$ defined here is proportional to the adaboost "D_t" see *Analysis of training error (II)*
3. And now the minimisation we perform to find α_t is the same as in *Analysis of the training error (IV)*

Why the exponential loss for classification? (I)

Recall the following typical set-up for classification. We are given a family of hypothesis $\mathcal{F}$ and wish to find a $f \in \mathcal{F}$ so that we predict well on future data. One approach is to find an f that minimizes the loss $(\lambda = 0)$ on the empirical data or regularized loss $(\lambda > 0)$, thus solving

$$\min_{f \in \mathcal{F}} \sum_{i=1}^{m} V(y_i, f(\mathbf{x}_i)) + \lambda \text{complexity}(f)$$

Problems: In classification as opposed to regression we face two (related) problems:

1. The misclassification loss is not smooth hence difficult to optimize
2. To make the class the function $\mathcal{F}$ both rich and smooth typically we choose the class so that the functions f map to $\mathbf{R}$ rather than $\{-1, 1\}$ and then predict with $\text{sign}(f)$

Thus we are motivated for the purpose of finding a hypothesis $f \in \mathcal{F}$ to choose V to be different from the misclassification loss.

Why the exponential loss for classification? (II)

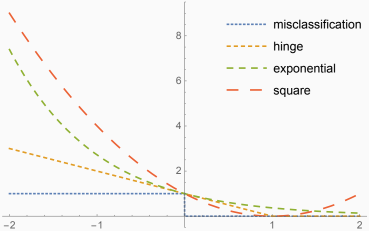
Consider the following typical loss functions:

$$V_{\mathrm{mc}}(y,\hat{y}) = \mathcal{I}[y = \mathrm{sign}(\hat{y})]$$

$$V_{\mathrm{hinge}}(y,\hat{y}) = \max(0, 1 - y\hat{y})$$

$$V_{\mathrm{exp}}(y,\hat{y}) = \exp(-y\hat{y})$$

$$V_{\mathrm{sq}}(y,\hat{y}) = (y - \hat{y})^2$$



$V_{\mathrm{mc}}(1,\hat{y})$, $V_{\mathrm{hinge}}(1,\hat{y})$, $V_{\mathrm{exp}}(1,\hat{y})$, and $V_{\mathrm{sq}}(1,\hat{y})$

The margin of a prediction $\hat{y}$ is $y\hat{y}$ for $y \in \{-1, 1\}$.

1. Misclassification is not continuous
2. Square loss unnecessarily punishes predictions with increasing positive margin
3. Hinge loss has a nice the nice property that it punishes negative margins but not positive margins
4. Hinge loss is not differentiable everywhere however.
5. Exponential loss punishes negative margins and promotes large positive margins (secondarily).

Boosting Summary

- Given a *weak* learner oracle
- Boosting finds a linear combination of weak hypotheses with arbitrarily small *training* error
- With certain assumptions this implies that the *test* error is bounded with high probability
- Boosting may be motivated as a *forward stagewise additive model* with exponential loss

Decision trees and ensembles summary

- Decision tree-based methods are useful in practice
- They provide “human-interpretable” models
- A strong advantage that they work on ordinal data. Less feature cleaning needed.
- Ensemble methods are often used on top of trees
- Bagging tends to use full trees
- Boosting often uses only decision stumps (single split trees)
- In the best cases (problem dependent)
  - Bagging tends to reduce variance
  - Boosting tends to reduce bias
- Boosting generally has more problems on very noisy data

Suggested Readings

The lecture notes are based on *The Elements of Statistical Learning ...*, Chapters 9.2 and 10.

Also recommended *The Boosting Approach to Machine Learning An Overview*, Schapire; *A decision-theoretic generalization of on-line learning and application to boosting*, Freund and Schapire, *Boosting the margin: A new explanation for the effectiveness of voting methods*, Schapire, Freund, Bartlett, and Lee

For a compact presentation of boosting and decision trees and further theory see Chapters 10 & 18 of *Understanding Machine Learning from Theory to Algorithms*

Problems – 1

1. Given the dataset

$$\{((1,1),9),((1,2),-4),((1,3),2),((2,2),4),((2,3),2)\}$$

   find the regression tree corresponding to the greedy recursive partitioning procedure with respect to square error.

   1.1 Draw the recursive partition.

   1.2 Draw the tree representation.

2. Explain how a random forest is constructed. What is the motivation for the tree construction method?
3. Both bagging and boosting produce predictions by creating an ensemble of functions. Explain how these ensembles differ (you do not need to describe the methods for arriving at the ensembles just how structurally the ensembles will differ).

Problems – 2

1. The Adaboost initilisation and update.
   1.1 Explain how Adaboost chooses its initial weighting $D_0(i)$ of the examples
   $$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$$
   1.2 Explain how it *updates* the distribution D_{t-1} after choosing a weak learner h_t with coefficient α_t at stage t including the normalising constant $Z(t)$.
   1.3 Give an intuitive justification for this update.
2. 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.
3. Using the result of part (1.2) obtain a bound for
   $$\frac{1}{m}\sum_{i=1}^{m} I[f_T(\mathbf{x}_i) \neq y_i] \leq \frac{1}{m}\sum_{i=1}^{m} \exp(-y_i f_T(\mathbf{x}_i)),$$
   in terms of the normalisation constants $Z(1), \ldots, Z(T)$, where $I[x]$ is the indicator function with value 1 when x is true and 0 otherwise, and $f_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$.

Problems – 3

1. A data set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ is *generic* iff $\mathbf{x}_i = \mathbf{x}_j \Rightarrow y_i = y_j$. A set of functions $\mathcal{H}$ is called *weakly-universal* if for every *generic* data set and weighting of the data there exists a function $h \in \mathcal{H}$ with "weighted error" < 0.5. **[subtle]**

   - A positive decision stump $h_{i,z} : \mathbf{R}^n \to \{-1, 1\}$ is defined as

     $$h_{i,z}(\mathbf{x}) := \begin{cases} 1 & \text{if } x_i \leq z \\ -1 & \text{if } x_i > z \end{cases} .$$

     The set of all decision stumps is defined as

     $$\mathcal{H}_{\text{ds}} := \{s\, h_{i,z} : s \in \{-1, 1\}, i \in \{1, 2, \ldots, n\}, z \in \mathbf{R}\} .$$

     Is $\mathcal{H}_{\text{ds}}$ weakly-universal? Provide an argument supporting your answer.
   - Let,

     $$S := \{a, b, aa, ab, ba, bb, aaa, \ldots\}$$

     denote the set of all strings of non-zero finite length over $\{a, b\}$. A string x is a *substring* of y, denoted as $x \sqsubseteq y$ iff x is contained as a consecutive sequence of characters in y. Thus $a \sqsubseteq abba$, $ab \sqsubseteq abba$, $abba \sqsubseteq abba$ but $aba \not\sqsubseteq abba$ and $abbaa \not\sqsubseteq abba$. A positive substring-match function $g : S \to \{-1, 1\}$ is defined as,

     $$g_z(x) := \begin{cases} 1 & \text{if } x \sqsubseteq z \\ -1 & \text{if } x \not\sqsubseteq z \end{cases} .$$

     The set of all substring-match functions is denoted as

     $$\mathcal{H}_{\text{ss}} := \{s\, g_z : s \in \{-1, 1\}, z \in S\} .$$

     Is $\mathcal{H}_{\text{ss}}$ weakly-universal? Provide an argument supporting your answer.