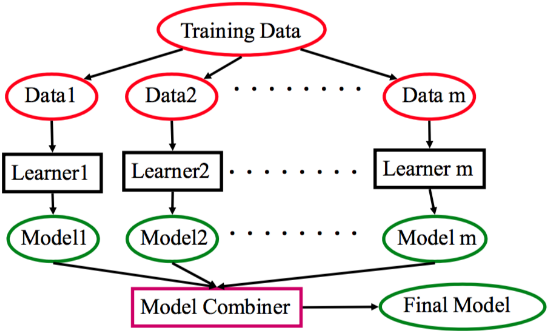
Statistical Machine Learning

Pier Francesco Palamara
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
http://www.stats.ox.ac.uk/~palamara/SML_BDI.html

Learning Ensembles

- Learn multiple alternative definitions of a concept using different training data or different learning algorithms.
- Combine decisions of multiple definitions, (e.g. using weighted voting).

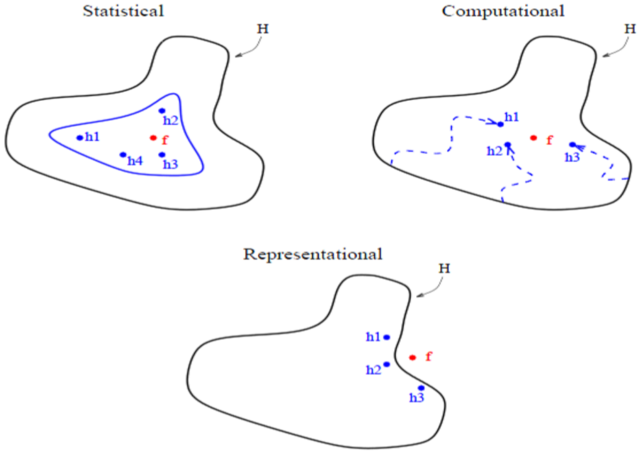


Three fundamental reasons for good ensembles

- It is desirable to build good ensembles for three fundamental reasons. (Dietterich, 2000):
  - Statistical: if little data
  - Computational: enough data, but local optima produced by local search
  - Representational: when the true function f cannot be represented by any of the hypothesis in $\mathcal{H}$ (weighted sums of hypotheses drawn from $\mathcal{H}$ might expand the space)

Three fundamental reasons for good ensembles

- It is desirable to build good ensembles for three fundamental reasons. (Dietterich, 2000):



Value of Ensembles

- “No Free Lunch” Theorem
  - No single algorithm wins all the time
- When combining multiple independent and diverse decisions each of which is at least more accurate than random guessing, random errors cancel each other out, correct decisions are reinforced.
- Examples: Human ensembles are demonstrably better
  - How many jelly beans in the jar?: Individual estimates vs. group average.
  - Who want to be a millionaire: Audience vote.

Homogeneous Ensembles

- Use a single arbitrary learning algorithm, but manipulate training data to make it learn multiple models.
  - Data 1 $\neq$ Data 2 $\neq \cdots \neq$ Data m
  - Lerner 1 $=$ Lerner 2 $= \cdots =$ Lerner m
- In this course, we consider two methods of this kind:
  - Bagging: Resample training data (last time)
  - Boosting: Reweight training data (today)

Approach: Bagging (Boostrap + Aggregating)

- Create ensembles by “bootstrap aggregation”, (i.e., repeatedly randomly resampling the training data) to generate training sets (Breiman, 1996).
- Bootstrap: draw N data points with replacement from original data set of size N.
- For each resampled data set, train base learners using an unstable[1] learning procedure (like decision trees).
- During test, combine learners by e.g. taking the average.
- This decreases error by decreasing the **variance** in the results due to unstable learners.

[1] Unstable algorithm: when small change in the training set causes a large difference in the base learners (high variance).

Approach: Boosting

Weak learners vs Strong learners

- In boosting, we actively try to generate complementary base-learners by training the next learner on the mistakes of the previous learners. We build a **strong learner** using **weak learners**.
- Example: in a binary classification problem, a weak learner does at least a bit better than random guessing, but not much better. A strong learner has arbitrarily small error probability.

In boosting, focus is on reducing **bias**, rather than variance.

Approach: Boosting

History

- In 1988 Kearns and Valiant posed the question of whether one can “boost” a weak learner to a strong learner.
- Two years later Rob Schapire published his landmark paper “The Strength of Weak Learnability” closing the theoretical question by providing the first “boosting” algorithm.
- Schapire and Yoav Freund worked together for the next few years to produce a simpler and more versatile algorithm called Adaboost.
- They received the 2003 Gödel Prize. “Best off-the-shelf classifier in the world” (Breiman 1998).

AdaBoost: Overview

- Adaptive Boosting (AdaBoost)
- As in bagging, we will use the same training set over and over.
- Classifiers must be “simple” (i.e. **weak**) so they do not overfit.
- Can combine an arbitrary number of base learners. (parameters)
- When testing, given an instance, all the classifiers make predictions and a weighted vote is taken.
- The weights are proportional to the base learners’ accuracies on the training set.

Probably Approximately Correct (PAC)

- Definition: PAC (not examinable)
  An algorithm $A(\epsilon, \delta)$ is said to PAC-learn the concept class $\mathcal{H}$ over the set $\mathcal{X}$ if, for any distribution $\mathcal{D}$ over $\mathcal{X}$ and for any $0 < \epsilon, \delta < 1/2$ and for any target concept $c \in \mathcal{H}$, the probability that A produces a hypothesis h of error at most ϵ is at least $1 - \delta$. In symbols, $P_{\mathcal{D}}(err_{c,\mathcal{D}}(h) \leq \epsilon) > 1 - \delta$. Moreover, A must run in time polynomial in $1/\epsilon, 1/\delta$ and n, where n is the size of an element $x \in \mathcal{X}$.

- Weak PAC-learning model requires the algorithm to have accuracy that is slightly better than random guessing. That is the algorithm will output a classification function which will correctly classify a random label with probability at least $\frac{1}{2} + \eta$ for some small, but fixed, $\eta > 0$.

We call an algorithm that produces PAC guarantees a **strong learner**, while an algorithm with the latter guarantees is called a **weak learner**.

Strong and Weak PAC-learning

- It turns out that strong learning and weak learning are equivalents! We can obtain a strong learner by combining weak learners. How?
- We can maintain a large number of separate instances of the weak learner, run them on our dataset, and then combine their hypotheses with a majority vote.

Strong learners from weak learners

- This is a bit too simplistic: what if the majority of the weak learners are wrong?
- We can do better:
  Instead of taking a majority vote, we can take a weighted majority vote.
- That is, give the weak learner a random subset of your data, and then test its hypothesis on the data to get a good estimate of its error.
- Then you can use this error to say whether the hypothesis is any good, and give good hypotheses high weight and bad hypotheses low weight (proportionally to the error).
- Then the “boosted” hypothesis would take a weighted majority vote of all hypotheses on an example.

Strong learners from weak learners

- Rather than use the estimated error just to say something about the hypothesis, we can identify the mislabeled examples in a round and somehow encourage A to do better at classifying those examples in later rounds.
- This turns out to be the key insight, and it's why the algorithm is called AdaBoost. (Ada stands for "adaptive").
- We are adaptively modifying the distribution over the training data we feed to A based on which data A learns "easily" and which it does not.
- So as the boosting algorithm runs, the distribution given to A has more and more probability weight on the examples that A misclassified.

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers.
- Notation: we indicate the weak learner using $h(\cdot)$.
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing the weighted classification error

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
  3. Update weights on training points

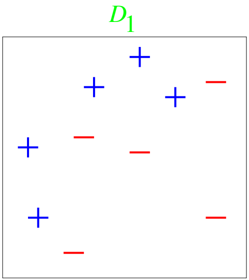
$$w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

  and normalize them such that $\sum_n w_{t+1}(n) = 1$
- Output the final classifier

$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x})\right]$$

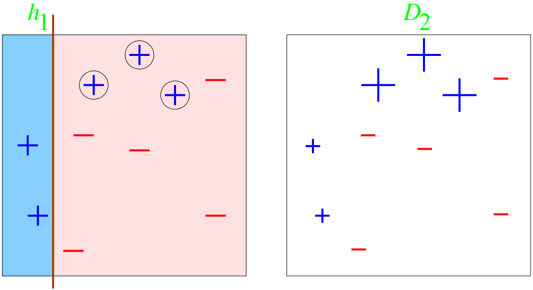
Example

10 data points and 2 features



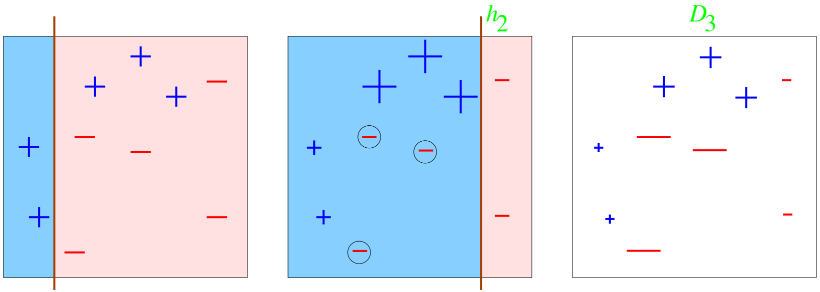
- The data points are clearly not linear separable
- In the beginning, all data points have equal weights (the size of the data markers "+" or "-")
- Base classifier $h(\cdot)$: either horizontal or vertical lines
  - These 'decision stumps' are just trees with a single internal node, i.e., they classifying data based on a single attribute

Round 1: $t = 1$



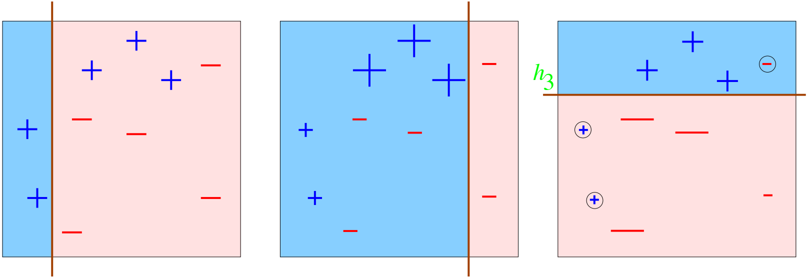
- 3 misclassified (with circles): $\epsilon_1 = 0.3 \rightarrow \beta_1 = 0.42$.
- Weights recomputed; the 3 misclassified data points receive larger weights

Round 2: $t = 2$



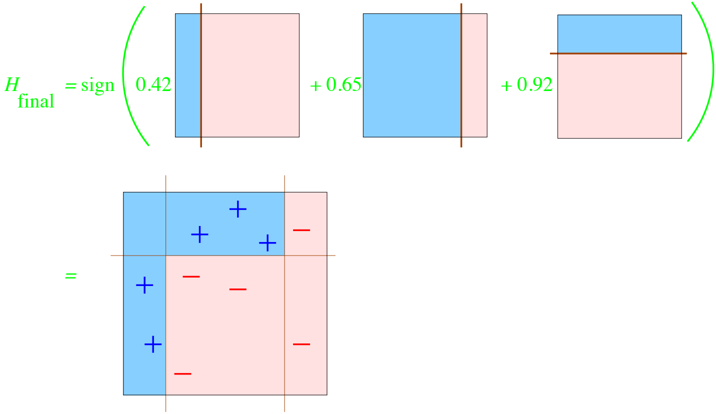
- 3 misclassified (with circles): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.
  Note that $\epsilon_2 \neq 0.3$ as those 3 data points have weights less than $1/10$
- 3 misclassified data points get larger weights
- Data points classified correctly in both rounds have small weights

Round 3: $t = 3$



- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Previously correctly classified data points are now misclassified, hence our error is low; what's the intuition?
    - Since they have been consistently classified correctly, this round's mistake will hopefully not have a huge impact on the overall prediction

Final classifier: combining 3 classifiers



- All data points are now classified correctly!

Why AdaBoost works?

It minimizes a loss function related to classification error.

Classification loss

- Suppose we want to have a classifier

$$h(\boldsymbol{x}) = \text{sign}[f(\boldsymbol{x})] = \begin{cases} 1 & \text{if } f(\boldsymbol{x}) > 0 \\ -1 & \text{if } f(\boldsymbol{x}) < 0 \end{cases}$$

- Our loss function is thus

$$\ell(h(\boldsymbol{x}), y) = \begin{cases} 0 & \text{if } yf(\boldsymbol{x}) > 0 \\ 1 & \text{if } yf(\boldsymbol{x}) < 0 \end{cases}$$

  Namely, the function $f(\boldsymbol{x})$ and the target label y should have the same sign to avoid a loss of 1.

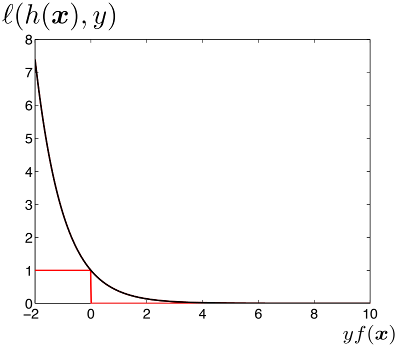
Surrogate loss

As we discussed for logistic regression, the $0-1$ loss function $\ell(h(\boldsymbol{x}), y)$ is non-convex and difficult to optimize. But as we did with logistic regression, we can come up with a tractable approximation of the $0-1$ loss:

Exponential Loss

$$\ell^{\text{EXP}}(h(\boldsymbol{x}), y) = e^{-yf(\boldsymbol{x})}$$

$\ell^{\text{EXP}}(h(\boldsymbol{x}), y)$ is easier to handle numerically as it is differentiable



Choosing the t-th classifier

Suppose we have built a classifier $f_{t-1}(\boldsymbol{x})$, and we want to improve it by adding a weak learner $h_t(\boldsymbol{x})$

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

How can we choose optimally the new classifier $h_t(\boldsymbol{x})$ and the combination coefficient β_t?
Adaboost greedily **minimizes the exponential loss function**.

$$\begin{aligned}
(h_t^*(\boldsymbol{x}), \beta_t^*) &= \underset{(h_t(\boldsymbol{x}),\beta_t)}{\operatorname{argmin}} \sum_n e^{-y_n f(\boldsymbol{x}_n)} \\
&= \underset{(h_t(\boldsymbol{x}),\beta_t)}{\operatorname{argmin}} \sum_n e^{-y_n [f_{t-1}(\boldsymbol{x}_n) + \beta_t h_t(\boldsymbol{x}_n)]} \\
&= \underset{(h_t(\boldsymbol{x}),\beta_t)}{\operatorname{argmin}} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)}
\end{aligned}$$

where we have used $w_t(n)$ as a shorthand for $e^{-y_n f_{t-1}(\boldsymbol{x}_n)}$

The new classifier

We can decompose the **weighted** loss function into two parts

$$
\begin{aligned}
&\sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(\boldsymbol{x}_n)] \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} (1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]) \\
&= (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}
$$

We have used the following properties to derive the above

- $y_n h_t(\boldsymbol{x}_n)$ is either 1 or -1 as $h_t(\boldsymbol{x}_n)$ is the output of a binary classifier
- The indicator function $\mathbb{I}[y_n = h_t(\boldsymbol{x}_n)]$ is either 0 or 1, so it equals $1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$

Finding the optimal weak learner

Summary

$$(h_t^*(\boldsymbol{x}), \beta_t^*) = \underset{(h_t(\boldsymbol{x}),\beta_t)}{\operatorname{argmin}} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)}$$

$$= \underset{(h_t(\boldsymbol{x}),\beta_t)}{\operatorname{argmin}} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + e^{-\beta_t} \sum_n w_t(n)$$

What term(s) must we optimize to choose $h_t(\boldsymbol{x}_n)$?

$$h_t^*(\boldsymbol{x}) = \underset{h_t(\boldsymbol{x})}{\operatorname{argmin}}\, \epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

Minimize weighted classification error as noted in step 1 of Adaboost!

How to choose β_t?

Summary

$$
\begin{aligned}
(h_t^*(\boldsymbol{x}), \beta_t^*) &= \underset{(h_t(\boldsymbol{x}),\beta_t)}{\operatorname{argmin}} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \underset{(h_t(\boldsymbol{x}),\beta_t)}{\operatorname{argmin}} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \\
&\qquad + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}
$$

What term(s) must we optimize?

We need to minimize the entire objective function with respect to β_t!

We can do this by taking derivative with respect to β_t, setting to zero, and solving for β_t. After some calculation and using $\sum_n w_t(n) = 1$, we find:

$$
\beta_t^* = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}
$$

which is precisely step 2 of Adaboost! (**Exercise – verify the solution**)

Updating the weights

Once we find the optimal weak learner we can update our classifier:

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x})$$

We then need to compute the weights for the above classifier as:

$$\begin{aligned} w_{t+1}(n) &= e^{-y_n f(\boldsymbol{x}_n)} = e^{-y_n [f_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x}_n)]} \\ &= w_t(n) e^{-y_n \beta_t^* h_t^*(\boldsymbol{x}_n)} = \begin{cases} w_t(n) e^{\beta_t^*} & \text{if } y_n \neq h_t^*(\boldsymbol{x}_n) \\ w_t(n) e^{-\beta_t^*} & \text{if } y_n = h_t^*(\boldsymbol{x}_n) \end{cases} \end{aligned}$$

Intuition Misclassified data points will get their weights increased, while correctly classified data points will get their weight decreased

Meta-Algorithm

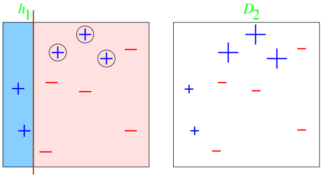
Note that the AdaBoost algorithm itself never specifies how we would get $h_t^*(\boldsymbol{x})$ as long as it minimizes the weighted classification error

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t^*(\boldsymbol{x}_n)]$$

In this aspect, the AdaBoost algorithm is a meta-algorithm and can be used with any type of classifier

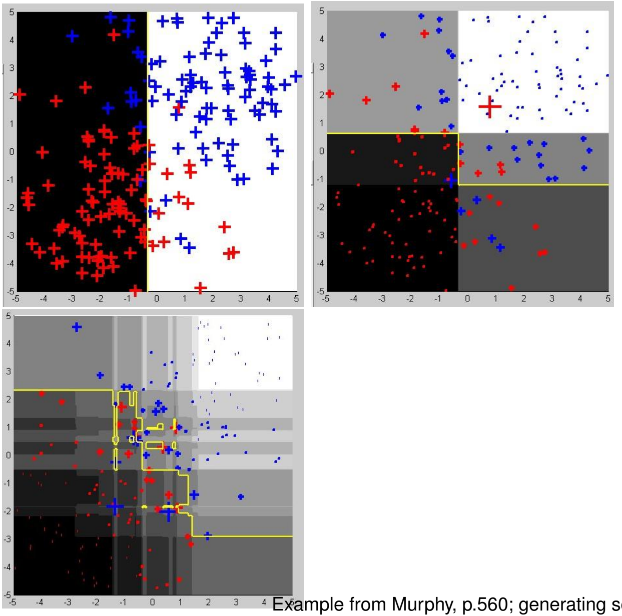
E.g., Decision Stumps

How do we choose the decision stump classifier given the weights at the second round of the following distribution?



- Presort data by each feature in O($dN \log N$) time
- Evaluate $N + 1$ thresholds for each feature at each round in O(dN) time
- In total O($dN \log N + dNT$) time – this efficiency is an attractive quality of boosting!

Interpreting boosting as learning nonlinear basis



Example from Murphy, p.560; generating script written by R.Stapenhurst

Example: Netflix

- The Netflix Prize: improve the accuracy of predictions about how much someone is going to love a movie based on their movie preferences.
- http://www.netflixprize.com
- Training data is a set of users and past ratings (1 to 5 stars).
- Construct a classifier that predicts user rating for unrated movies.
- Winning team (BellKor's Pragmatic Chaos) employed boosting. They received 1M$.

FIN

Syllabus I

Part I: Introduction to unsupervised learning

- Dimensionality reduction
  - Principal component analysis, SVD, Biplots, Multidimensional scaling, Isomap
- Clustering
  - K-means
  - Hierarchical clustering

Syllabus II

Part II: Supervised learning

- Empirical risk minimization
- Regression
  - Linear
  - Non-linear basis functions
  - Gradient descent
- Overfitting, cross-validation
- Regularization
- Bias/variance tradeoff
- Classification
  - Discriminant analysis
  - Logistic regression
  - Naïve Bayes
  - K-nearest neighbors
- Generative vs discriminative methods
- Performance evaluation

Syllabus III

Part III: Useful algorithms for supervised learning

- Decision trees
- Bagging/Random forests
- Neural networks
- Deep learning
- Boosting