Lecture 4: Combining predictors – Part I

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

November, 2015

There is more than one way to average predictors

- Bayesian averaging
- Bagging
- Stacking
- Mixtures of Experts
- Forward Fitting and Backfitting

Reducing bias: Boosting

- AdaBoost
- Properties on the training set

Reading HTF: 15. Random Forests, 16. Ensembles of predictors, 8. Model inference and averaging, 10. Boosting, Murphy: 16.3, 16.3.1 Generalized Additive models (ignore the regularization, for "smoother" read "minimize loss function"), 16.4.1-5 [and optionally 8] Bosting, 3.2.4 Bayesian model averaging, 16.6.3, 16.6.1 Ensembles of predictors

There is more than one way to average predictors

Classification will be the running exaple here, but most results hold for other prediction problems as well.
Denote $\mathcal{B} = \{b\}$ a **base classifier** family
Averaging:

$$f(x) = \sum_{k=1}^{M} \beta^k b^k(x) \tag{1}$$

f is real-valued even if the b^k's are ± 1 valued
Why average predictors?

- to reduce bias
- to compensate for local optima (a form of bias)
- to reduce variance
- if $b_1, b_2, \ldots b_M$ make independent errors, averaging reduces expected error (loss). We say that b_1 and b_2 make **independent errors** $\Leftrightarrow$ $P(b_1 \text{ wrong} \mid x) = P(b_1 \text{ wrong} \mid x, b_2 \text{ wrong})$
- because the b^k functions are given: real world domain experts (weather prediction), a set of black-box classifiers (from a software package), a set of [expert designed] features (speech recognition, car/human recognition) each of them weakly informative of y
- because $\mathcal{B}$ is a set of (simple) "basis functions" and we need a more complex predictor in our task

Averaging is not always the same thing

Depending how we choose $\mathcal{B}$, $b^1, b^2, \ldots b^M$ and $\beta^1, \beta^2, \ldots \beta^M$, we can obtain very different effects.
We will examine

- Bayesian averaging (briefly)
- Mixtures of experts (briefly)
- Bagging (briefly)
- Backfitting (briefly)
- Stacking (see Murphy)
- Boosting

Bayesian averaging

Assume any predictor $b \in \mathcal{B}$ could be the "true" predictor with some **prior** probability $P_0(b)$. Learning means changing the probability distribution of b after seeing the data.

Before seeing data	$P_0(b)$	prior probability of b
After seeing $\mathcal{D}$	$P(b\|\mathcal{D})$	posterior probability of b
Bayes formula	$P(b\|\mathcal{D}) = \frac{P_0(b)P(\mathcal{D}\|b)}{\sum_{b' \in \mathcal{B}} P_0(b')P(\mathcal{D}\|b')}$	

Classification of a new instance by Bayesian averaging:

$$f(x) = \int_{\mathcal{B}} b(x) dP(b|\mathcal{D})$$

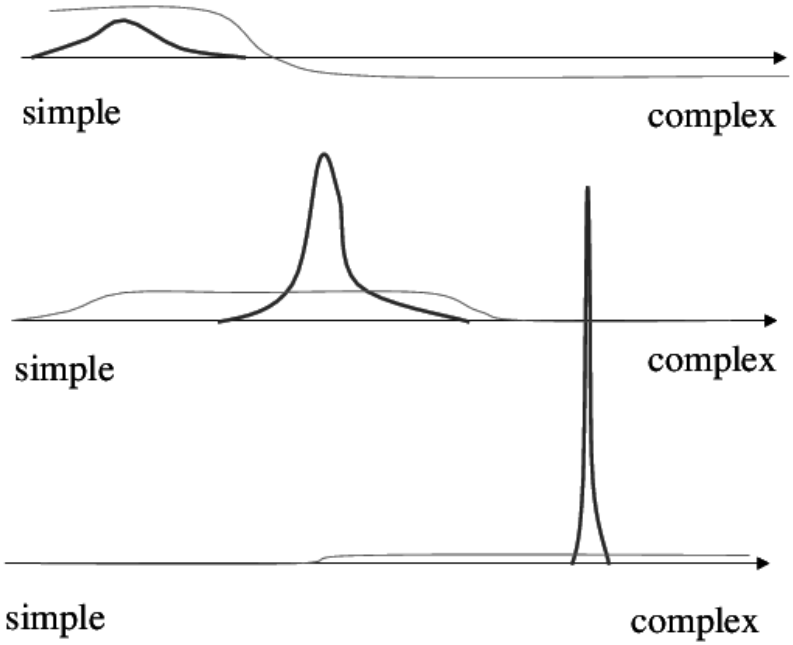
or

$$P(y|x, \mathcal{D}) = \int_{\mathcal{B}} \mathbf{1}_{b(x)=y} dP(b|\mathcal{D})$$

Hence classifiers (or more generally predictors) are weighted by their posterior probability.

Intuition: The likelihood becomes more concentrated when N increases

Bayesian averaging and model complexity



- model too simple: likelihood low, prior high
- model about right: likelihood high, prior not too low
- model too complex: likelihood high, prior very low

Bayesian averaging in practice.

$$\hat{P}(b^k|\mathcal{D}) = \frac{P_0(b^k)P(\mathcal{D}|b^k)}{\sum_{k'=1}^{M} P_0(b_{k'})P(\mathcal{D}|b_{k'})} \equiv \beta_k$$

$b^{1:M}$ are either sampled from $\mathcal{B}$, or trained separately (e.g local minima of $\hat{L}$, models of different complexities)

Priors in practice

- non-informative
- complexity penalizing, sparsity inducing, etc

Reducing variance: Bagging

What if we had several (independently sampled) trainig sets $\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_M$?

- we could train classifiers $\{b^1, b^2, \ldots b^M\}$ on the respective $\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_M$
- we could estimate $E_{P(b)}[b] \sim f = \frac{1}{M} \sum_{k=1}^{M} b^k$
- f has always lower variance than b^k

Idea of **bagging**: sample $\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_M$ from the given $\mathcal{D}$ and estimate b^k on $\mathcal{D}_k$

$$f(x) = \operatorname{sgn} \frac{1}{M} \sum_{k=1}^{M} b^k(x)$$

Thus, bagging is a form of boostrap.

Bagging reduces variances

It was shown theoretically and empirically that bagging reduces variance.
Bagging is good for

- base classifiers with high variance (complex)
- unstable classifiers (decision trees, decision lists, neural networks)
- noisy data

Example

Random Forests A large ensemble of decision trees is fitted to the same data set, introducing randomness in various ways, such as (1) resampling the data set, (2) taking random splits, with probabilities that favor "good" splits, etc. The output predictor is the (unweighted) average of all the trees.

- can be extended to more than classification (regression, feature selection)
- training can be done in parallel
- computing $f(x)$ on new example is fast enough
- VERY POPULAR in industry

Stacking

- very general method (any kind of predictors or costs)
- for complex base classifiers

1. Fit predictors $b^1, \dots b^M$ to the data $\mathcal{D}$. The predictors can be from different model classes (i.e neural networks, CART, nearest neighbors, logistic regressions) or use different sets of features.
   For $k = 1 : M$, for data point $i = 1 : N$
   - train b^k_{-i} from the model class of b^k on $\mathcal{D} \setminus \{(x^i, y^i)\}$
2. Fit coefficients $\beta_{1:M}$ by minimizing the leave-one-out (loo) empirical cost $\hat{L}^{loo}$

$$\hat{L}^{loo}(\beta_{1:M} = \frac{1}{N} \sum_{i=1}^{N} L(y^i, \sum_{k=1}^{M} \beta_k b^k_{-i}(x^i)) \tag{2}$$

   (or by cross-validation).

Note that if L is a non-linear function, the minimization in (2) is a non-linear minimization, and in particular for convex losses this is a convex optimization problem in $\beta_{1:M}$.

Mixtures of Experts

Here $\beta_k = \beta_k(x)$, and $\sum_k \beta^k(x) = 1$ for all x in the domain of the inputs.
Idea each b^k is an "expert" in one region of the input space.
Wanted $f \approx b^k(x)$ in the region of expertise of expert b^k
The vector function $\beta(x) = [\beta^1(x) \ \ldots \ \beta^M(x)]$ is sometimes called the gating function.

Example

Suppose the true function to learn is $f^*(x) = |x|$, $x \in \mathbb{R}$. This can be well approximated by two linear experts $f^1(x) = x$, $f^2(x) = -x$ with weights $\beta^1 = \phi(x), \beta^1 = \phi(-x)$, where ϕ is the sigmoid function (hence $\beta^1 + \beta^2 = 1$ everywhere.

The example highlights that, by using a mixture of experts we can construct a more complex classifier by from simple classifiers (linear). "Simple" can mean easy to fit, or low complexity (aka simple decision region), or both. The effective sample size for each b^k is smaller than N, and corresponds to the data for which $\beta^k(x^i)$ is away from 0.

For more than two experts, a natural gating function is the softmax function

$$\beta^k(x) = \frac{e^{v_k^T x}}{\sum_{l=1}^{M} e^{v_l^T x}} \text{ with } v_l \in \mathbb{R}^n \text{ a vector of parameters} \tag{3}$$

Training By descent methods. Often the experts $f^{1:M}$ and gating functions $\beta^{1:M}$ are trained simultaneously (to maximize log-likelihood) by steepest descent, or EM algorithm (which we'll study later).

Forward Fitting and Backfitting

(b^k, β^k) are fitted iteratively (sequentially), one k at a time.
In some cases, the weights β^k can be absorbed into b^k.
f^t is f at iteration t The residual $r \in \mathbb{R}^N$ is defined as $r^t(x^i) = y^i - f^t(x^i)$.

ForwardFitting Algorithm

Input M, labeled training set $\mathcal{D}$
Initialize $f = 0$
repeat
  for $k = 1, 2, \ldots M$
    fit k-th predictor $\beta^k, b^k = \text{argmin} \hat{L}(f + \beta b)$
    update $f = f + b^k \beta^k$
until change in $\hat{L}$ small enough (or, change in b^k small enough)
Output $f(x) = \sum_{k=1}^{M} \beta^k b^k(x)$

Note that M does not have to be set in advance (just like in boosting).

Backfitting

Set M at the beginning, and cycle through the M predictors, updating predictor k while keeping the others fixed. Denote by

$$f^{-k}(x) = \sum_{l \neq k} \beta^k b^k(x) \tag{4}$$

the combined predictor f "minus" the k-th base predictor b^k.

BackFitting Algorithm

Input M, labeled training set $\mathcal{D}$

Initialize $b^{1:M} = 0$, [$\beta^{1:M} = 0$ if there are coefficients β]

repeat

for $k = 1, 2, \ldots M$

calculate $r^k(x^i) = y^i - f^{-k}(x^i)$, $i = 1 : N$

optimize $\hat{L}$ w.r.t k-th base predictor $\beta^k, b^k = \text{argmin}\hat{L}(r^k + \beta b)$

until change in $\hat{L}$ small enough (or, change in $b^{1:M}$ small enough)

Output $f(x) = \sum_{k=1}^{M} \beta^k b^k(x)$

Often these methods are used when b^k are assumed to be simple, weak, and the predictor f is built from the cooperation of several b's. Thus, they are generally bias reduction method.

Example

Least squares regression In this problem, it is useful to denote

$$r^k(x^i) \;=\; y^i - f^{-k}(x^i) \tag{5}$$

the residual of f^{-k} at data point x^i. Then, $L_{LS}(y^i, f(x^i)) = y^i - f^{-k}(x^i) - \beta^k b^k(x^i)$. Hence, optimizing $\hat{L}_{LS}$ w.r.t. β^k, b^k can be expressed as

$$\beta^k, b^k \;=\; \underset{\beta, b}{\operatorname{argmin}} \sum_{i=1}^{N} (r^i - \beta b(x^i))^2 \,. \tag{6}$$

In other words, each step of backfitting is a least squares regression problem, where the output variable values y^i are replaced with the current residuals r^i.

See also Additive Models

Reducing bias: Boosting

Base classifier family $\mathcal{B}$ has large bias (e.g. linear classifier, decision stumps) but learning always produces b that is better (on the training set) than random guessing.
Preconditions for boosting

1. Learning algorithm accepts **weighted** data sets. Training minimizes

$$\hat{L}_{01}^{w}(b) = \sum_{i=1}^{N} w_i L_{01}(y^i, b(x^i)) \quad \text{with} \quad \sum_{i=1}^{N} w_i = 1.$$

2. $\mathcal{B}$ is a weak classifier family. For any $\mathcal{D}$ and any weights $w_{1:N}$ there can be found $b \in \mathcal{B}$ such that the training error of b on $\mathcal{D}$ is bounded below one half.

$$0 < \hat{L}_{01}^{w}(b) \leq \delta < \frac{1}{2}$$

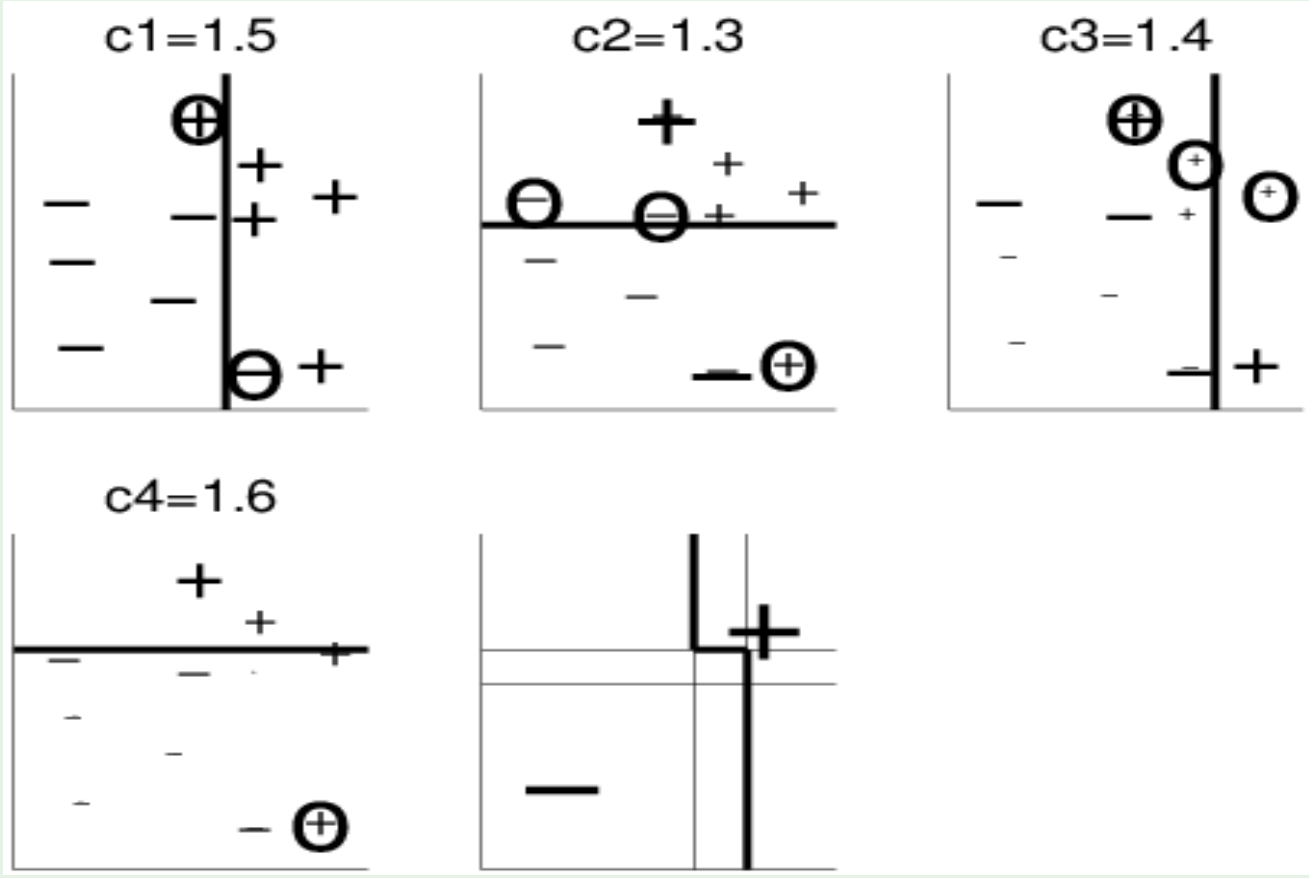
Idea of boosting: train a classifier b^1 on $\mathcal{D}$, then train a b^2 to correct the errors of b^1, then b^3 to correct the errors of b^2, etc.

Example

Boosting with stumps

Stumps are decision trees with a single split.

(below, $c_1 \ldots c_4$ denote the coefficients $\beta_{1:4}$).



AdaBoost Algorithm

ADABOOST ALGORITHM

Assume $\mathcal{B}$ contains functions b taking values in $[-1, 1]$ or $\{\pm 1\}$

Input M, labeled training set $\mathcal{D}$

Initialize $f = 0$

$w_i^1 = \frac{1}{N}$ weight of datapoint x_i

for $k = 1, 2, \ldots M$

1. "learn classifier for $\mathcal{D}$ with weights w^k" $\Rightarrow$ b^k
2. compute error $\varepsilon^k = \sum_{i=1}^N w_i^k \frac{1 - y_i b^k(x_i)}{2}$
3. set $\beta^k = \frac{1}{2} \ln \frac{1-\varepsilon^k}{\varepsilon^k}$
4. compute new weights $w_i^{k+1} = \frac{1}{Z^k} w_i^k e^{-\beta^k y_i b^k(x_i)}$ where Z^k is the normalization constant that makes $\sum_i w_i^{k+1} = 1$

Output $f(x) = \sum_{k=1}^M \beta^k b^k(x)$

Remarks

1. If $b(x) \in \{\pm 1\}$ then $y^i b(x^i) \in \{\pm 1\}$, and $\frac{1-y^i b(x^i)}{2} = 1$ if an error occurs and 0 otherwise. Thus, ε^k in step 2 adds up the weights of the errors.
   If $b(x) \in [-1, 1]$ then the errors contribute different amounts to the loss depending on their margin.
2. In both cases, $\varepsilon^k \in [0, \delta]$, $\delta < 0.5$ by the weak learner property of $\mathcal{B}$
3. $\beta^k > 0$ whenever $\varepsilon^k < 1/2$.
4. If $b \in \{\pm 1\}$, then step 4 can be written equivalently (up to a multiplicative constant)

$$w_i^{k+1} = \begin{cases} \frac{1}{Z^k} w_i^k & \text{if } b^k(x_i) = y_i \\ \frac{1}{Z^k} w_i^k e^{2\beta^k} & \text{if } b^k(x_i) \neq y_i \end{cases} \tag{7}$$

   This form corresponds to the DiscreteAdaBoost algorithm, the first AdaBoost algorithm published, which assumed $b(x) \in \{\pm 1\}$. As we shall see later, modern boosting algorithms dispense with the assumption $b \in [-1, 1]$ too.

An interpretation of the weights

$$w_i^{k+1} = \frac{1}{N} \prod_{k' \leq k} \frac{e^{-\beta^{k'} y_i b^{k'}(x_i)}}{Z_{k'}} = \frac{e^{-y_i f^k(x_i)}}{N \prod_{k' \leq k} Z^k} \tag{8}$$

- weight of example i at step k is proportional to $e^{-y_i f^{k-1}(x_i)}$ the exponential of its negative margin
- Examples that have been hard to classify get exponentially high weights.
- Examples that are classified with high margins get vanishingly small weights.

The normalization constant is an average loss

If we sum both sides of (8) over i we obtain

$$1 \;=\; \frac{\sum_i e^{-y_i f^k(x_i)}}{N \prod_{k' \le k} Z^k}, \tag{9}$$

or

$$\prod_{k' \le k} Z^k \;=\; \frac{\sum_i e^{-y_i f^k(x_i)}}{N} \;\equiv\; \hat{L}_\phi(f^k) \tag{10}$$

where

$$\phi(z) \;=\; e^{-z}. \tag{11}$$

and

$$L_\phi(y, f(x)) \;=\; \phi(yf(x)) \tag{12}$$

Hence, the r.h.s of (10) is the average over the data set of the **exponential loss** L_ϕ. The function ϕ decreases with the margin, thus decreasing $\hat{L}_\phi$ will produce a better classifier (on the training set). In this sense, L_ϕ is an alternative loss function for classification.

$\hat{L}_\phi$ decreases exponentially with M

For simplicity, we show this in the special case $b(x) \in \{\pm 1\}$ for all $b \in \mathcal{B}$.

$$
\begin{aligned}
Z^k &= \sum_{i=1}^{n} w_i^k e^{-\beta^k (y_i b^k(x_i))} && (13)\\
&= e^{\beta^k} \underbrace{\sum_{i=err} w_i^k}_{\varepsilon^k} + e^{-\beta^k} \underbrace{\sum_{i=corr} w_i^k}_{1-\varepsilon^k} && (14)\\
&= e^{\beta^k} \varepsilon^k + (1-\varepsilon^k) e^{-\beta^k} && (15)\\
&= \sqrt{\frac{1-\varepsilon^k}{\varepsilon^k}} \varepsilon^k + \sqrt{\frac{\varepsilon^k}{1-\varepsilon^k}} (1-\varepsilon^k) = 2\sqrt{(1-\varepsilon^k)\varepsilon^k} \leq \gamma && (16)
\end{aligned}
$$

where $\gamma < 1$ depends on δ the maximum error. It follows that

$$
\hat{L}_\phi(f^k) = \prod_{k=1}^{k} Z^{k'} \leq \gamma^k \qquad (17)
$$

The training set error $\hat{L}_{01}$ decreases exponentially with M

Note that $\phi(z) \geq 1_{z<0}$ for all z (see also figure **??**). Therefore

$$\hat{L}(f^k) = \frac{1}{N}\sum_{i=1}^{N} 1_{[y_i f^k(x_i)<0]} \tag{18}$$

$$\leq \frac{1}{N}\sum_{i=1}^{N} e^{-y_i f^k(x_i)} = \hat{L}_\phi(f^k) \leq \gamma^k \tag{19}$$

In other words, the training error $\hat{L}(f^k)$ is bounded by a decaying exponential. Moreover, since $\hat{L}(f^k) \in \{0, 1/N, 2/N, \ldots 1\}$, it follows that after a finite number of steps, when $\gamma^{k^0} < 1/N$, the training error will become 0 and the training data will be perfectly classified!

The test set error and overfitting

- Do NOT take $M = k^0$. The number of steps M for good generalization error is often much larger than k^0 (and sometimes smaller).
- Below is a typical plot of $\hat{L}$ and L (which can be estimated from an independent sample) vs the number of boosting iterations.