Lecture 21: Bagging and Boosting

Hao Helen Zhang

Outlines

- Bagging Methods
    - Bagging Trees
    - Random Forest
- Adaboost
    - Strong learners vs Weak Learners
    - Motivations
    - Algorithms
- Additive Models
- Adaboost and Additive Logistic Regression

Ensemble Methods (Model Averaging)

A machine learning *ensemble* meta-algorithm designed to improve the stability and accuracy of machine learning algorithms

- widely used in statistical classification and regression.
- can reduce variance and helps to avoid overfitting.
- usually applied to decision tree methods, but it can be used with any type of method.

Bootstrap Bagging (Bagging)

Bagging is a special case of the model averaging approach.

Bootstrap aggregation = Bagging

- Bagging leads to "improvements for unstable procedures" (Breiman, 1996), e.g. neural nets, classification and regression trees, and subset selection in linear regression (Breiman, 1994).
- On the other hand, it can mildly degrade the performance of stable methods such as K-nearest neighbors (Breiman, 1996).

Basic Idea

Given a standard training set D of size n, bagging

- generates B new training sets D_i, each of size n', by sampling from D uniformly and with replacement.
- The B models are fitted using the above B bootstrap samples and combined by averaging the output (for regression) or voting (for classification).

This kind of sample is known as a bootstrap sample.

- By sampling with replacement, some observations may be repeated in each D_i.
- If $n' = n$, then for large n, the set D_i is expected to have the fraction $(1 - 1/e) \approx 63.2\%$ of the unique examples of D, the rest being duplicates.

Bagging Procedures

Bagging uses the bootstrap to improve the estimate or prediction of a fit.

- Given data $\mathbf{Z} = \{(x_1, y_1), ..., (x_n, y_n)\}$, we generate B bootstrap samples $\mathbf{Z}^{*b}$
    - **Empirical distribution** $\hat{\mathcal{P}}$: putting equal probability $1/n$ on each (x_i, y_i) (discrete)
    - Generate $\mathbf{Z}^{*b} = \{(x_{1*}, y_{n*}), ..., (x_{n*}, y_{n*})\} \sim \hat{\mathcal{P}}, b = 1, ..., B$
- Obtain $\hat{f}^{*b}(x)$, $b = 1, ..., B$.
- The Monte Carlo estimate of the bagging estimate

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x).$$

Properties of Bagging Estimates

Advantages:

- Note $\hat{f}_{\text{bag}}(x) \longrightarrow E_{\hat{\mathcal{P}}}\hat{f}^*(x)$ as $B \to \infty$,
- $\hat{f}_{\text{bag}}(x)$ typically has smaller variance than $\hat{f}(x)$;
- $\hat{f}_{\text{bag}}(x)$ differs from $\hat{f}(x)$ only when the latter is nonlinear or adaptive function of data.

Bagging Classification Trees

In multiclass ($K \geq 3$)problems, there are two scenarios

(1) $\hat{f}^b(x)$ is indicator-vector, with one 1 and K-1 0's (hard classification)

(2) $\hat{f}^b(x) = (p_1, ..., p_K)$, the estimates of class probabilities (soft classification)

The bagged estimate is the average prediction at x from B trees

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x),$$

Bagging Classification Trees (cont.)

There are two types of averaging:

(1) uses the majority vote;

(2) use the averaged probabilities.

The bagged classifier

$$\hat{G}_{\text{bag}}(x) = \arg\max_{k=1,\cdots,K} \hat{f}_{\text{bag}}(x).$$

Example

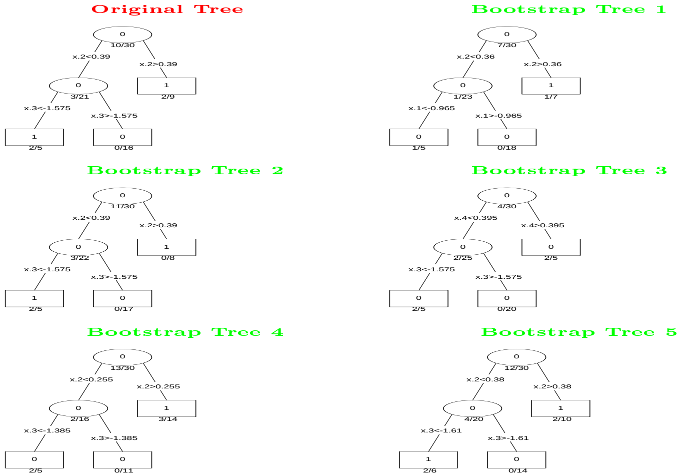
- Sample size $n = 30$, two classes
- $p = 5$ features, each having a standard Gaussian distribution with pairwise correlation $\text{Corr}(X_j, X_k) = 0.95$.
- The response Y was generated according to

$$\Pr(Y = 1 | x_1 \leq 0.5) = 0.2, \quad \Pr(Y = 1 | x_1 > 0.5) = 0.8.$$

- The Bayes error is 0.2.
- A test sample of size $2,000$ was generated from the same population.

We fit

- classification trees to the training sample
- classification trees to each of 200 bootstrap samples


Original Tree
0
10/30
x.2<0.39
x.2>0.39
0
3/21
1
2/9
x.3<-1.575
x.3>-1.575
1
2/5
0
0/16
Bootstrap Tree 1
0
7/30
x.2<0.36
x.2>0.36
0
1/23
1
1/7
x.1<-0.965
x.1>-0.965
0
1/5
0
0/18
Bootstrap Tree 2
0
11/30
x.2<0.39
x.2>0.39
0
3/22
1
0/8
x.3<-1.575
x.3>-1.575
1
2/5
0
0/17
Bootstrap Tree 3
0
4/30
x.4<0.395
x.4>0.395
0
2/25
0
2/5
x.3<-1.575
x.3>-1.575
0
2/5
0
0/20
Bootstrap Tree 4
0
13/30
x.2<0.255
x.2>0.255
0
2/16
1
3/14
x.3<-1.385
x.3>-1.385
0
2/5
0
0/11
Bootstrap Tree 5
0
12/30
x.2<0.38
x.2>0.38
0
4/20
1
2/10
x.3<-1.61
x.3>-1.61
1
2/6
0
0/14

About Bagging Trees

The original tree and five bootstrap trees are all different:

- with different splitting features
- with different splitting cutpoints
- The trees have high variance due to the correlation in the predictors

Averaging reduces variance and leaves bias unchanged.

- Under squared-error loss, averaging reduces variance and leaves bias unchanged.
- Therefore, bagging will often decrease MSE.

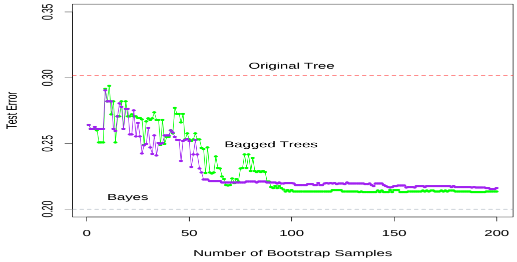


Figure 8.10: *Error curves for the bagging example of Figure 8.9. Shown is the test error of the original tree and bagged trees as a function of the number of bootstrap samples. The green points correspond to majority vote, while the purple points average the probabilities.*

About Bagging

- Bagging can dramatically reduce the variance of unstable procedures like trees, leading to improved prediction
    - Bagging smooths out this variance and hence reducing the test error
    - Bagging can stabilize unstable procedures.
- The simple structure in the model can be lost due to bagging
    - A bagged tree is no longer a tree.
    - The bagged estimate is not easy to interpret.
- Under 0-1 loss for classification, bagging may not help due to the nonadditivity of bias and variance.

Random Forest

- Random forest is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees.
- The algorithm was developed by Breiman (2001) and Cutler.
- The method combines Breiman's "bagging" idea and the random selection of features, in order to construct a collection of decision trees with controlled variation.
- The selection of a random subset of features is an example of the random subspace method, a way to implement stochastic discrimination.

Learning Algorithm for Building A Tree

Denote the training size by n and the number of variables by p. Assume $m < p$ is the number of input variables to be used to determine the decision at a node of the tree.

- Randomly choose n samples with replacement (i.e. take a bootstrap sample).
- Use the rest of the samples to estimate the error of the tree, by predicting their classes.
- For each node of the tree, randomly choose m variables on which to base the decision at that node. Calculate the best split based on these m variables in the training set.
- Each tree is fully grown and not pruned.

Advantages of Random Forest

- highly accurate in many real examples; fast; handles a very large number of input variables.
- ability to estimate the importance of variables for classification through permutation.
- generates an internal unbiased estimate of the generalization error as the forest building progresses.
- impute missing data and maintains accuracy when a large proportion of the data are missing.
- provides an experimental way to detect variable interactions.
- can balance error in unbalanced data sets.
- compute proximities between cases, useful for clustering, detecting outliers, and (by scaling) visualizing the data
- can be extended to unlabeled data, leading to unsupervised clustering, outlier detection and data views

Disadvantages of Random Forest

- Random forests are prone to overfitting for some data sets. This is even more pronounced in noisy classification/regression tasks.
- Random forests do not handle large numbers of irrelevant features

Motivation: Model Averaging

They are methods for improving the performance of weak learners.

- strong learners: Given a large enough dataset, the classifier can arbitrarily accurately learn the target function with probability $1 - \tau$ (where $\tau > 0$ can be arbitrarily small)
- weak learners: Given a large enough dataset, the classifier can barely learn the target function with probability $\frac{1}{2} + \tau$
  - The error rate is only slightly better than a random guessing

Can we construct a strong learner from weak learners and how?

Boosting

Motivation: combines the outputs of many weak classifiers to produce a powerful "committee".

- Similar to bagging and other committee-based approaches
- Originally designed for classification problems, but can also be extended to regression problem.

Consider the two-class problem

- $Y \in \{-1, 1\}$, the classifier $G(\mathbf{x})$ has training error

$$\text{err} = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq G(\mathbf{x}_i))$$

- The expected error rate on future predictions is $E_{\mathbf{X},Y} I(Y \neq G(\mathbf{X}))$.

Classification Trees

Classifications trees can be simple, but often produce noise or weak classifiers.

- Bagging (Breiman 1996): Fit many large trees to bootstrap-resampled versions of the training data, and classify by a majority vote.
- Boosting (Freund & Shapire 1996): Fit many large or small trees to re-weighted versions of the training data. Classify by a weighted majority vote.

In general, Boosting $>$ Bagging $>$ Single Tree

- Breiman's comment "AdaBoost .... best off-the-shelf classifier in the world". (1996, NIPS workshop)

AdaBoost (Discrete Boost)

Adaptively resampling the data (Freund & Shapire 1997; winners of the 2003 Godel Prize)

1. sequentially apply the weak classification algorithm to repeatedly modified versions of the data (re-weighted data)
2. produces a sequence of weak classifiers

$$G_m(\mathbf{x}), \quad m = 1, 2, ..., M.$$

3. The predictions from G_m's are then combined through a weighted majority vote to produce the final prediction

$$G(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})\right).$$

Here $\alpha_1, ..., \alpha_M \geq 0$ are computed by the boosting algorithm.

AdaBoost Algorithm

1. Initially the observation weights $w_i = 1/n, i = 1, ..., n$.
2. For m=1 to M
   (a) Fit a classifier $G_m(\mathbf{x})$ to the training data using weights w_i.
   (b) Compute *the weighted error*
   $$\text{err}_m = \frac{\sum_{i=1}^n w_i I(y_i \neq G_m(\mathbf{x}_i))}{\sum_{i=1}^n w_i}.$$
   (c) Compute *the importance* of G_m as
   $$\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$$
   (d) Update $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(\mathbf{x}_i))], i = 1, ..., n$.
3. Output $G(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^M \alpha_m G_m(\mathbf{x})\right)$.

Weights of Individual Weak Learners

In the final rule, the weight of G_m

$$\alpha_m = \log\left(\frac{1 - \text{err}_m}{\text{err}_m}\right),$$

where err_m is the weight error of G_m.

- The weights α_m's weigh the contribution of each G_m.
- The higher (lower) err_m, the smaller (larger) α_m.
  - The principle is to give higher influence (larger weights) to more accurate classifiers in the sequence.

Data Re-weighting (Modification) Scheme

At each boosting, we impose the updated weights $w_1, ..., w_n$ to samples $(\mathbf{x}_i, y_i), i = 1, ..., n$.

- Initially, all weights are set to $w_i = 1/n$. The usual classifier.
- At step $m = 2, \ldots, M$, we modify weights for observations individually: increasing weights for those observations misclassified by $G_{m-1}(\mathbf{x})$ and decreasing weights for observations classified correctly by $G_{m-1}(\mathbf{x})$.
  - Samples difficult to correctly classify receive ever increasing influence
  - Each successive classifier is forced to concentrate on those training observations that are missed by previous ones in sequence
- The classification algorithm is re-applied to the weighted observations

Elements of Statistical Learning ©Hastie, Tibshirani & Friedman 2001 Chapter 10

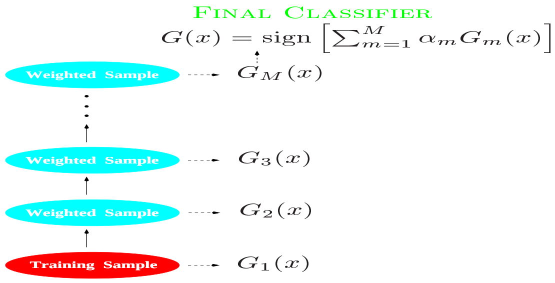


Figure 10.1: *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

Power of Boosting

- Ten features $X_1, ..., X_{10} \sim N(0,1)$
- two-classes: $Y = 2 \cdot I(\sum_{j=1}^{10} X_j^2 > \chi_{10}^2(0.5)) - 1$
- sample size $n = 2000$, test size $10,000$
- weak classifier: stump (a two-terminal node classification tree)
- performance of boosting and comparison with other methods:
    - stump has 46% misclassification rate
    - 400-mode tree has 26% misclassification rate
    - boosting has 12.2% error rate

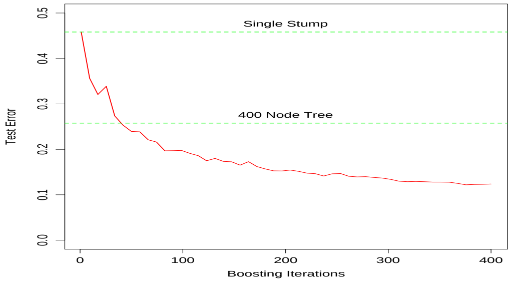


Figure 10.2: *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 400 node classification tree.*

Boosting And Additive Models

The success of boosting is not very mysterious. The key lies in

$$G(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})\right).$$

- Adaboost is equivalent to fitting an additive model using the exponential loss function (a very recent discovery by Friedman et al. (2000)).
- AdaBoost was originally motivated from a very different perspective

Introduction on Additive Models

An additive model typically assumes a function form

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m b(\mathbf{x}; \gamma_m),$$

- β_m's are coefficients. $b(\mathbf{x}, \gamma_m)$ are basis functions of $\mathbf{x}$ characterized by γ_m.

The model $\hat{f}$ is obtained by minimizing a loss averaged over the training data

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^{n} L\left(y_i, \sum_{m=1}^{M} \beta_m b(\mathbf{x}_i; \gamma_m)\right). \tag{1}$$

It is feasible to rapidly solve the sub-problem of fitting just a single basis.

Forward Stagewise Fitting for Additive Models

Forward stagewise modeling approximate the solution to (1) by

- sequentially adding new basis functions to the expansion without adjusting the parameters and coef. of those that have been added.
- At iteration m, one solves for the optimal basis function $b(\mathbf{x}, \hat{\gamma}_m)$ and corresponding coefficient $\hat{\beta}_m$, which is added to the current expansion $f_{m-1}(\mathbf{x})$.
    - Previously added terms are not modified.
- This process is repeated.

Squared-Error Loss Example

Consider the squared-error loss

$$L(y, f(\mathbf{x})) = [y - f(\mathbf{x})]^2$$

At the *m*th step, given the current fit $f_{m-1}(\mathbf{x})$, we solve

$$\min_{\beta,\gamma} \quad \sum_i L(y_i, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}, \gamma)) \Longleftrightarrow$$

$$\min_{\beta,\gamma} \quad \sum_i [y_i - f_{m-1}(\mathbf{x}_i) - \beta b(\mathbf{x}_i; \gamma)]^2 = \sum_i [r_{im} - \beta b(\mathbf{x}_i, \gamma)]^2.$$

The term $\hat{\beta}_m b(\mathbf{x}; \hat{\gamma}_m)$ is the best fit to the current residual. This produces the updated fit

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \hat{\beta}_m b(\mathbf{x}; \hat{\gamma}_m).$$

Forward Stagewise Additive Modeling

1. Initialize $f_0(\mathbf{x}) = 0$.
2. For $m = 1$ to M:
   (a) Compute

$$(\hat{\beta}_m, \hat{\gamma}_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{n} L\left(y_i, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \gamma)\right).$$

   (b) Set $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \hat{\beta}_m b(\mathbf{x}; \hat{\gamma}_m)$

Additive Logistic Models and AdaBoost

Friedman et al. (2001) showed that AdaBoost is equivalent to forward stagewise additive modeling

- using the exponential loss function

$$L(y, f(\mathbf{x})) = \exp\{-yf(\mathbf{x})\},$$

- using individual classifiers $G_m(\mathbf{x}) \in \{-1, 1\}$ as basis functions

The (population) minimizer of the exponential loss function is

$$\begin{aligned} f^*(\mathbf{x}) &= \arg\min_f E_{Y|\mathbf{x}}[e^{-Yf(\mathbf{x})}] \\ &= \frac{1}{2} \log \frac{\Pr(Y = 1|\mathbf{x})}{\Pr(Y = -1|\mathbf{x})}, \end{aligned}$$

which is equal to one half of the log-odds. So, AdaBoost can be regarded as an additive logistic regression model.

Forward Stagewise Additive Modeling with Exponential Loss

For exponential loss, the minimization at *m*th step in forward stagewise modeling becomes

$$(\beta_m, \gamma_m) \quad = \quad \arg\min_{\beta,\gamma} \sum_{i=1}^{n} \exp\{-y_i[f_{m-1}(x_i) + \beta b(x_i, \gamma)]\}$$

In the context of a weak learner G, this is

$$(\beta_m, G_m) = \arg\min_{\beta,G} \sum_{i=1}^{n} \exp\{-y_i[f_{m-1}(x_i) + \beta G(x_i)]\},$$

or equivalently, $(\beta_m, G_m) = \arg\min_{\beta,G} \sum_{i=1}^{n} w_i^{(m)} \exp\{-y_i \beta G(x_i)\},$

where $w_i^{(m)} = \exp\{-y_i f_{m-1}(\mathbf{x}_i)\}$.

Iterative Optimization

In order to solve

$$\min_{\beta,G} \sum_{i=1}^{n} w_i^{(m)} \exp\{-y_i \beta G(x_i)\},$$

we take the two-step (profile) approach

- first, fix $\beta > 0$ and solve for $\hat{G}$.
- second, solve for β with $G = \hat{G}$.

Recall that both Y and $G(\mathbf{x})$ take only two values $+1$ and -1. So

$$y_i G(\mathbf{x}_i) = +1 \Longleftrightarrow y_i = G(\mathbf{x}_i),$$

$$y_i G(\mathbf{x}_i) = -1 \Longleftrightarrow y_i \neq G(\mathbf{x}_i).$$

Solving for $\hat{G}_m$

$$\begin{aligned}
(\beta_m, G_m) &= \arg\min_{\beta, G} \sum_{i=1}^{n} w_i^{(m)} \exp\{-y_i \beta G(x_i)\} \\
&= \arg\min_{\beta, G} e^{\beta} \sum_{y_i \neq G(x_i)} w_i^{(m)} + e^{-\beta} \sum_{y_i = G(x_i)} w_i^{(m)} \\
&= \arg\min_{\beta, G} (e^{\beta} - e^{-\beta}) \sum_{y_i \neq G(x_i)} w_i^{(m)} + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)}
\end{aligned}$$

For any $\beta > 0$, the minimizer $\hat{G}_m$ is a $\{-1, 1\}$-valued function

$$\hat{G}_m = \arg\min_{G} \sum_{i=1}^{n} w_i^{(m)} I[y_i \neq G(x_i)],$$

the classifier that minimizes training error for the weighted data.

Solving for $\hat{\beta}_m$

Define

$$\text{err}_m = \frac{\sum_{i=1}^n w_i^{(m)} I(y_i \neq \hat{G}_m(\mathbf{x}_i))}{\sum_{i=1}^n w_i^{(m)}}$$

Plugging $\hat{G}_m$ in the objective gives

$$\beta_m = \text{argmin}_\beta \{e^{-\beta} + (e^\beta - e^{-\beta})\text{err}_m\} \sum_{i=1}^n w_i^{(m)}$$

The solution is

$$\hat{\beta}_m = \frac{1}{2} \log \left(\frac{1 - \text{err}_m}{\text{err}_m} \right).$$

Connection to Adaboost

Since

$$-yG_m(x) = 2(I(y \neq G_m(x)) - 1,$$

we have

$$\begin{aligned} w_i^{(m+1)} &= w_i^{(m)} \exp\{-\beta_m y_i G_m(\mathbf{x}_i)\} \\ &= w_i^{(m)} \exp\{\alpha_m I(y_i \neq G(\mathbf{x}_i))\} \exp\{-\beta_m\} \end{aligned}$$

where $\alpha_m = 2\beta_m$ and $\exp\{-\beta_m\}$ is constant across the data points. Therefore

- the weight update is equivalent to line 2(d) of the AdaBoost
- line 2(a) of the Adaboost is equivalent to solving the minimization problem

Weights and Their Update

- At the *m*th step, the weight for the *i*th observation is

$$w_i^{(m)} = \exp\{-y_i f_{m-1}(\mathbf{x}_i)\},$$

  which depends only on $f_{m-1}(\mathbf{x}_i)$ but not on β or $G(\mathbf{x})$.
- At the $(m+1)$th step, we update the weight using the fact

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}_i) + \beta_m G_m(\mathbf{x}_i).$$

It leads to the following update formula

$$\begin{aligned} w_i^{(m+1)} &= \exp\{-y_i f_m(\mathbf{x}_i)\} \\ &= \exp\{-y_i(f_{m-1}(\mathbf{x}_i) + \beta_m G_m(\mathbf{x}_i))\} \\ &= w_i^{(m)} \exp\{-\beta_m y_i G_m(\mathbf{x}_i)\}. \end{aligned}$$

Why Exponential Loss?

- Principal virtue is computational
- Exponential loss concentrates much more influence on observations with large negative margins $yf(x)$. It is especially sensitive to misspecification of class labels.
- More robust losses: computation is not as easy (Use Gradient Boosting).

Exponential Loss and Cross Entropy

Define $Y' = (Y+1)/2 \in \{0,1\}$.
The binomial negative log-likelihood loss function is

$$\begin{aligned} -l(Y, p(\mathbf{x})) &= -\left[Y' \log p(\mathbf{x}) + (1-Y')\log(1-p(\mathbf{x}))\right] \\ &= \log\left(1+\exp\{-2Yf(\mathbf{x})\}\right), \end{aligned}$$

where $p(\mathbf{x}) = [1+e^{-2f(\mathbf{x})}]^{-1}$.

- The population minimizers of the deviance $E_{Y|\mathbf{x}}[-l(Y, f(\mathbf{x})]$ and $E_{Y|\mathbf{x}}[e^{-Yf(\mathbf{x})}]$ are the same.

Loss Functions for Classification

Choice of loss functions matters for finite data sets.
The *Margin* of $f(\mathbf{x})$ is defined as $yf(\mathbf{x})$.
The classification rule is $G(\mathbf{x}) = \text{sign}[f(\mathbf{x})]$
The decision boundary is $f(\mathbf{x}) = 0$

- Observations with positive margin $y_i f(\mathbf{x}_i) > 0$ are correctly classified
- Observations with negative margin $y_i f(\mathbf{x}_i) < 0$ are incorrectly classified

The goal of a classification algorithm is to produce positive margins as frequently as possible

- Any loss function should penalize negative margins more heavily than positive margins, since positive margin are already correctly classified

Various Loss Functions

Monotone decreasing loss functions of the margin

- Misclassification loss: $I(\text{sign}(f(\mathbf{x})) \neq y)$
- Exponential loss: $\exp(-yf)$ (not robust against mislabeled samples)
- Binomial deviance: $\log\{1 + \exp(-2yf)\}$ (more robust against influential points)
- SVM loss: $[1 - yf]_+$

Other loss functions

- Squared error: $(y - f)^2 = (1 - yf)^2$ (penalize positive margins heavily)

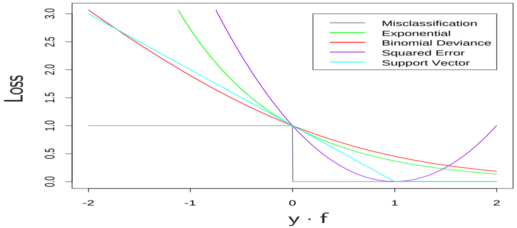


Figure 10.4: *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\operatorname{sign}(f)$*. The losses are misclassification:* $I(\operatorname{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1 + \exp(-2yf))$*; squared error:* $(y - f)^2$*; and support vector:* $(1 - yf) \cdot I(yf > 1)$ *(see Section 12.3). Each function has been scaled so that it passes through the point* $(0, 1)$*.*

Brief Summary on AdaBoost

- AdaBoost fits an additive model, where the basis functions $G_m(x)$ stage-wise optimize exponential loss
- The population minimizer of exponential loss is the log odds
- There are loss functions more robust than squared error loss (for regression problems) or exponential loss (for classification problems)