Statistical Foundations of Data Science

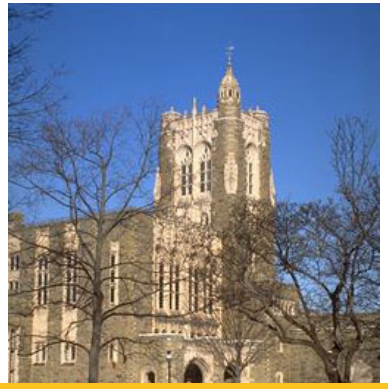
Jianqing Fan

Princeton University

https://fan.princeton.edu

ZOOM ID Lectures: **970 4936 8998** Office Hours: **996 4030 7631**

Annotated Lecture Notes: web view



5. Supervised Learning

5.1 Model-Based Classifiers

Classification: problem setup

■Data: i.i.d. samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ from a distribution. *y* categorical

$\mathbf{x}_i \in \mathbb{R}^p$: feature vectors; $y_i \in \mathcal{C}$: class labels. $\pm 1, \pm 2$ ticks

Misclassification error of a classifier δ: $R(\delta) = P(Y \neq \delta(\mathbf{X}))$.

★**Goal**: learn a classifier $\widehat{\delta}$ with small $R(\widehat{\delta})$ (test error).

Bayes rule is optimal: as $R(\delta) = 1 - E[P(Y = \delta(\mathbf{X}))|\mathbf{X}]$

$$\delta^*(\mathbf{x}) = \text{argmax}_{k \in \mathcal{C}} P(Y = k | \mathbf{X} = \mathbf{x}) = \text{argmax}_{k \in \mathcal{C}} (\pi_k f_k(\mathbf{x})),$$

where $f_k(\mathbf{x})$ is the PDF of class k and π_k is the class probability.

Gaussian mixture models: QDA

Gaussian mixture model: $(\mathbf{X}|Y=k) \sim N(\mu_k, \mathbf{\Sigma}_k)$, i.e.

$$f_k(\mathbf{x}) = (2\pi)^{-p/2}|\mathbf{\Sigma}_k|^{-1/2}\exp\left(-\frac{1}{2}(\mathbf{x}-\mu_k)^T\mathbf{\Sigma}_k^{-1}(\mathbf{x}-\mu_k)\right).$$

★Marginal density $f(x) = \sum_k \pi_k N(\mu_k, \mathbf{\Sigma}_k)$ is a Gaussian mixture

★Bayes rule $\delta^*(\mathbf{x}) = \text{argmax}_{k\in\mathcal{C}}\,\delta_k^{QDA}(\mathbf{x})$ with

$$\delta_k^{QDA}(\mathbf{x}) = \log\pi_k - \frac{1}{2}\log|\mathbf{\Sigma}_k| - \frac{1}{2}(\mathbf{x}-\mu_k)^T\mathbf{\Sigma}_k^{-1}(\mathbf{x}-\mu_k).$$

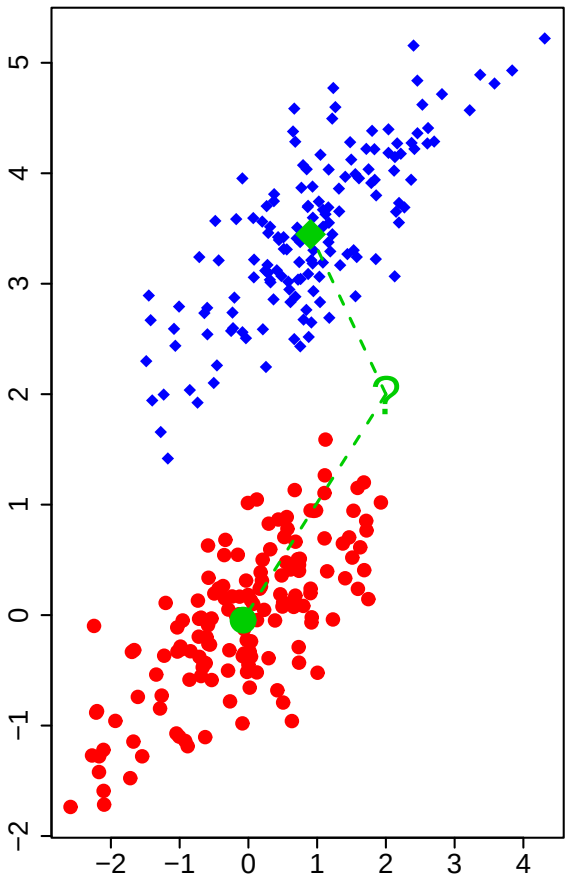
Quadratic Discriminant Analysis (QDA): replace π_k, μ_k and $\mathbf{\Sigma}_k$ by their estimates from training data.

Linear Discriminant Analusis

When $\mathbf{\Sigma}_k \equiv \mathbf{\Sigma}$, Bayes rule or QDA becomes LDA with

$$\delta_k^{LDA}(\mathbf{x}) = \log \pi_k - \frac{1}{2}\mu_k^T \mathbf{\Sigma}^{-1}\mu_k + \mu_k^T \mathbf{\Sigma}^{-1}\mathbf{x},$$

after ignoring the term $-\frac{1}{2}\log|\mathbf{\Sigma}_k| - \frac{1}{2}\mathbf{x}^T\mathbf{\Sigma}_k^{-1}\mathbf{x}$, indep of k.



LDA: $\pi_k \leftarrow \widehat{\pi}_k$, $\mu_k \leftarrow \widehat{\mu}_k$,
$\mathbf{\Sigma} \leftarrow \widehat{\mathbf{\Sigma}} = \frac{\sum_{k\in\mathcal{C}}(n_k-1)\widehat{\mathbf{\Sigma}}_k}{\sum_{k\in\mathcal{C}}(n_k-1)}$ (**pooled cov**),

When $\mathcal{C} = \{0,1\}$, LDA admits
$\widehat{\delta}(\mathbf{x}) = \mathbf{1}\left((\widehat{\mu}_1 - \widehat{\mu}_0)^T\widehat{\mathbf{\Sigma}}^{-1}(\mathbf{x} - \widehat{\mu}_a) + \log(\widehat{\pi}_1/\widehat{\pi}_0) > 0\right)$,
where $\widehat{\mu}_a = (\widehat{\mu}_1 + \widehat{\mu}_0)/2$.

Classification Errors

- For any $\mathbf{w} \in \mathbb{R}^p$, $(\mathbf{w}^T\mathbf{X}|Y=k) \sim N(\mathbf{w}^T\mu_k, \mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w})$.

Test Error of $\delta(\mathbf{X}) = I(\mathbf{w}^T(\mathbf{X}-\mu_a) > 0)$, $\qquad \mu_a = (\mu_1+\mu_0)/2$

$$R(\mathbf{w}) \equiv P(\delta(\mathbf{X}) \neq Y) = 1 - \Phi\Big(\frac{\mathbf{w}^T\mu_d}{(\mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w})^{1/2}}\Big),$$

where $\mu_d = (\mu_1-\mu_0)/2$.

Optimal classifier: $\mathbf{w}^* \propto \boldsymbol{\Sigma}^{-1}\mu_d$.

Logistic regression

- Fit $\log(\frac{p(\mathbf{x})}{1-p(x)}) = \beta_0 + \beta^T \mathbf{x}$. ★does not model dist($\mathbf{X}$).

- Prediction: $\widehat{\delta}(\mathbf{x}) = \mathbf{1}(\widehat{\beta}_0 + \widehat{\beta}^T \mathbf{x} > 0)$ (linear classifier).

- Compared with LDA, logist reg is more robust against model misspecification, but not as efficient under the correct model.

Reuters Newswires data consists of 11228 short newswires and their topics (46 topics) published in 1986.

It is included in R package `Keras`, frequently used as a testbed for text analysis.

```
library(keras)    #install.packages("keras") , install this first
max_words = 1000        #most frequent 1000 words will be used as x
reuters <- dataset_reuters(num_words = max_words, test_split = 0.2)
  #11228 short newswires and their topics published in 1986. 20% used as testing.
x_train <- reuters$train$x      #extract the training data
y_train <- reuters$train$y      #extract response (topic of newswires)
x_test <- reuters$test$x
y_test <- reuters$test$y
x_train[[1]];                   #take a peek on the first news data
[1]   1   2   2   8  43  10 447   5  25 207 270   5   2 111  16 369 186  90
##total 87 items, show the vocabulary in the top 1000 words of the first training artic
y_train[1:20]                   #a peek of topics for first 20 articles
# [1]  3  4  3  4  4  4  4  3  3 16  3  3  4  4 19  8 16  3  3 21
        #shape the data into matrix format
 tokenizer <- text_tokenizer(num_words = max_words)
 x_train <- sequences_to_matrix(tokenizer, x_train, mode = 'binary')
 x_test  <- sequences_to_matrix(tokenizer, x_test, mode = 'binary')
 dim(x_train)                    #take a peak of data dim
#[1] 8982 1000
 dim(x_test)
#[1] 2246 1000
x_train[1:2, 1:10]              #take a peek of actual data
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    0    1    1    0    1    1    1    1    1     1
[2,]    0    1    1    0    1    1    0    1    1     1
```

```
y_train = to_categorical(y_train, 46)  #put response as categorical data
y_test  = to_categorical(y_test,  46)
y_train[1:2,]                    #a peek of responses for first 2 articles after coding
        [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]    0    0    0    1    0    0    0    0    0     0     0     0     0     0
[2,]    0    0    0    0    1    0    0    0    0     0     0     0     0     0
................
     [,41] [,42] [,43] [,44] [,45] [,46]
[1,]     0     0     0     0     0     0
[2,]     0     0     0     0     0     0
############ logitic regression for classication ################
x_train1 = data.frame(x_train)              #create data.frame to pass along
x_test1  = data.frame(x_test)
y_train4 = reuters$train$y; y_train4[y_train4 != 4] = 0   #topic 4 data
y_train4 = as.factor(y_train4)
y_test4 = reuters$test$y; y_test4[y_test4 != 4] = 0

logit4 = glm(y_train4 ~ ., data=x_train1, family="binomial") #fitting logistic reg
logit.test4 = predict(logit4,x_test1)   #predict the data

logit.test4 = (logit.test4 > 0)         #classify as 0 and 1
mean(4*logit.test4 == y_test4)          #proportion of correction prediciton
#[1] 0.8824577
```

The correct prediction rate is 0.8825, compared with prior $1 - \pi \approx 0.79$. The following computes the true positive and true negative rate. It begins with the set of true positives, indexed by "ind".

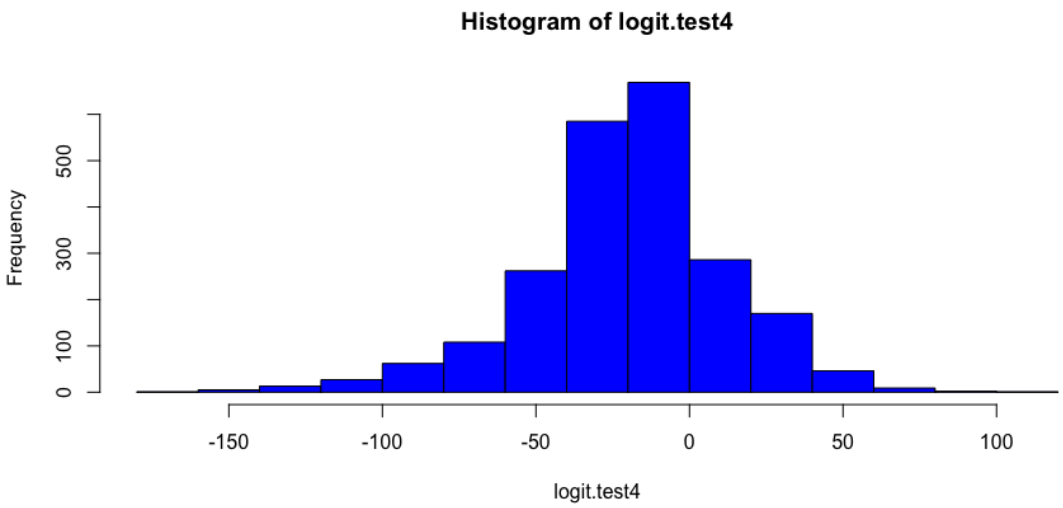
```
ind = (y_test4==4); mean(ind)            #cases with topic 4 and its prop = 0.2110419
mean(mean(4*logit.test4[ind] == y_test4[ind]))     #true positive
mean(mean(4*logit.test4[!ind] == y_test4[!ind]))   #true negative
```

Results: 0.7637131 (**sensitivity**) and 0.9142212 (**specificity**). Note that

$$0.2110419 * 0.7637131 + (1 - 0.2110419) * 0.9142212 = 0.8824577,$$

i.e., prior * sensitivity + (1-prior)*specificity = correct prediction rate.

To increase sensitivity, decrease threshold and specificity decreases as a result. Similarly, to increase specificity, increase the threshold. To determine the order of magnitude, it is helpful to look at the distribution `hist(logit.test4,col="blue")` before choosing a number. e.g. if using `logit.test4 = (logit.test4 > -10)`, sensitivity = 0.8797, specificity = 0.7737.

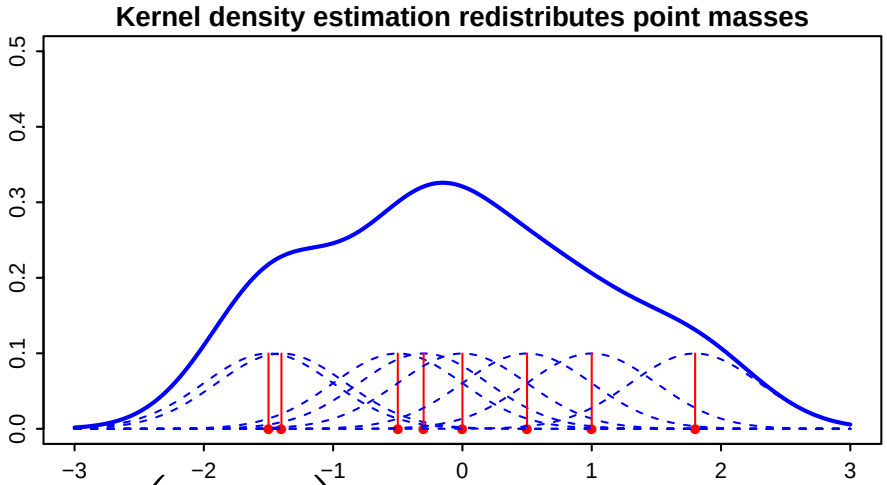


5.2 Kernel Density Classifiers and Naive Bayes

Kernel Density Classifiers

Kernel Density Classifier: $\text{argmax}_{k\in\mathcal{C}}(\widehat{\pi}_k \widehat{f}_k(\mathbf{x}))$ or

$$\widehat{\delta}^{KD}(\mathbf{x}) = \text{argmax}_{k\in\mathcal{C}} \left\{ \sum_{i:\ y_i=k} K\left(\frac{\mathbf{x}_i - \mathbf{x}}{h}\right) \right\}.$$



Kernel Density Estimator (KDE):

Given i.i.d. samples $\{\mathbf{z}_i\}_{i=1}^n$

$$\widehat{g}(\mathbf{z}) = \frac{1}{n}\sum_{i=1}^{n} h^{-p} K\left(\frac{\mathbf{z}_i - \mathbf{z}}{h}\right)$$

where K is a **kernel function** and h is the **bandwidth**.

R ★density

Kernel Density Estimation

- If K is a PDF $\Rightarrow \widehat{g}$ is a PDF.
- **Gaussian kernel**: $K(z) = (2\pi)^{-p/2} e^{-\|\mathbf{z}\|_2^2/2}$ **(comm used)**.
- KDE redistributes point mass at $\mathbf{z}_i$ smoothly to $h^{-p} K(\frac{\mathbf{z}_i - \mathbf{z}}{h})$.
- Small bandwidth $h \Longrightarrow$ smaller bias $\Longrightarrow$ bigger variance
- Optimal choice $h \asymp n^{-\frac{1}{2s+p}}$, with MSE $\asymp O(n^{-\frac{2s}{2s+p}})$ if $g(\cdot)$ has s derivatives **—curse of dimensionality**.
- Unimportant in choice of K, but important in h.
- **Rule of Thumb**: $h = 1.06\sigma n^{-1/5}$ for univ. Gaussian kernel.

Naive Bayes Classifiers

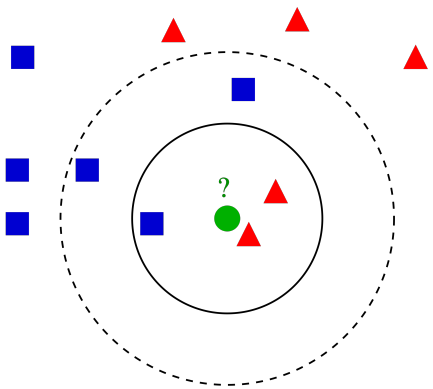
- ★ Nonparametric estimation of high-d density is difficult **(COD)**;
- ★ Yet, their marginal density is easy to estimate.
- ★ This motivates the **Naive Bayes Classifier**

$$\widehat{\delta}^{NB}(\mathbf{x}) = \text{argmax}_{k \in \mathcal{C}} \left\{ \widehat{\pi}_k \prod_{j=1}^{p} \widehat{f}_{kj}(x_j) \right\},$$

which **naively** estimates f_k by product of its marginal dist.

- ★ $\widehat{\delta}^{NB}$ handles both continuous and discrete features, and works **surprisingly well** in many complex problems!

5.3 Nearest Neighbor Classifiers



Nearest Neighbor Classifiers

Let $d(\mathbf{z}, \mathbf{w})$ be a distance metric in the feature space, e.g.

$$d(\mathbf{z}, \mathbf{w}) = \|\mathbf{z} - \mathbf{w}\|_q = \Big(\sum_{j=1}^{p} |z_j - w_j|^q \Big)^{1/q}, \qquad e.g.\ q = 1, 2$$

k-Nearest Neighbor: Let $\mathcal{N}_k(\mathbf{x}) = \{i : \text{k smallest in } \{d(\mathbf{x}_i, \mathbf{x})\}_{i=1}^n\}$. Output the prediction via **majority vote**:

$$\widehat{\delta}(\mathbf{x}) = \text{argmax}_{j \in \mathcal{C}} |\{i \in \mathcal{N}_k(\mathbf{x}) : y_i = j\}|.$$

- ★ $d(\cdot, \cdot)$ defines the **proximity** and k controls the **locality**.
- ★ k-NN suffers from the **curse of dim**, even 1-NN is not close, but works well in a **low-dim manifold**.
- ★ Computation intensive

5.4 Classification Trees and Ensemble Classifiers

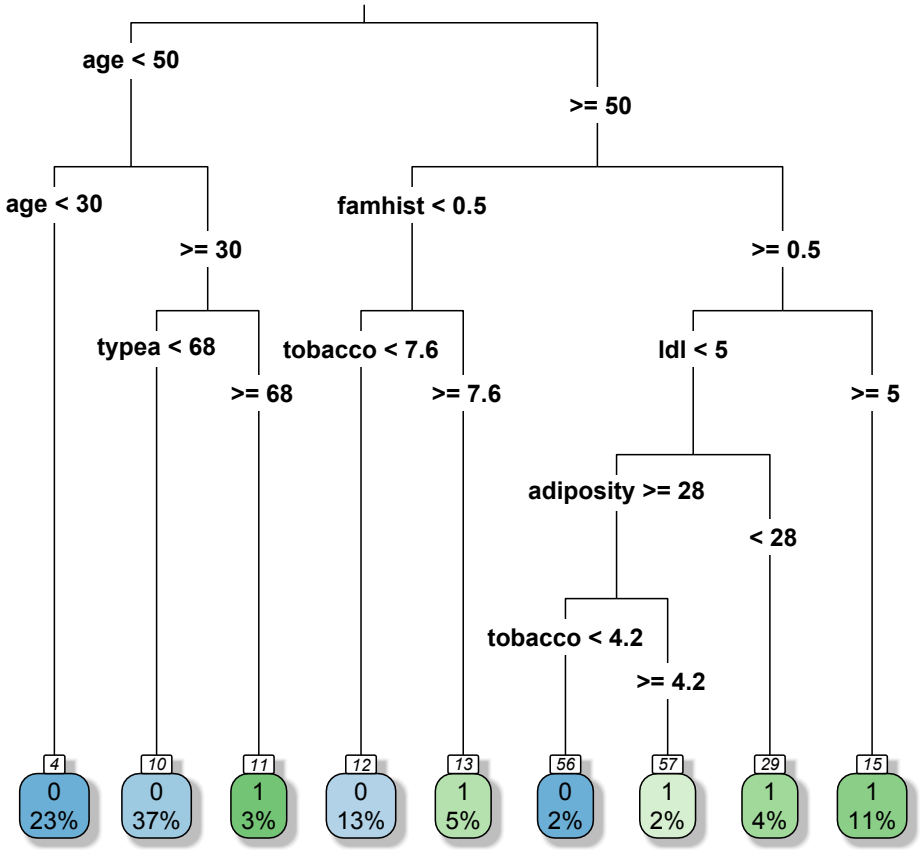
Classification Trees

A **Classification Tree** on South Africa heart disease data: the leaves are predictions for features falling into those regions.

R package ★`rpart`

$n = 462$

1. **sbp**: systolic blood pressure
2. **tobacco**: cumulative tobacco (kg)
3. **ldl**: low densiity lipoprotein cholesterol
4. **adiposity**
5. **famhist**: family history of heart disease (Present, Absent)
6. **typea**: type-A behavior
7. **obesity**
8. **alcohol**: current alcohol consumption
9. **age**: age at onset
10. **chd**: coronary heart disease



★1 = “yes”, 0 = “no” by majority vote; percentage = % of data

Classification and Regression Trees (CART)

A **Classification Tree** partitions $\mathbb{R}^p$ into **rectangular boxes**. Let $\mathcal{R}(\mathbf{x})$ be the rectangle containing $\mathbf{x}$. Prediction: majority vote

$$\text{argmax}_{k \in \mathcal{C}} |\{i : \mathbf{x}_i \in \mathcal{R}(\mathbf{x}), y_i = k\}|.$$

CART: a two-step procedure *(Breiman, Friedman, Olshen, Stone, 1984, 45K)*

★Step 1: **Grow** the tree by recursive bi-partition

1. Choose a rectangle $\mathcal{R}$ (i.e. a node in the tree) to be split.
2. Choose a feature $j \in [p]$ and a threshold $t \in \mathbb{R}$. $p(n-2)$ choices
3. Output $\mathcal{R}_1(j,t) = \{\mathbf{x} \in \mathcal{R} : x_j \leq t\}$, $\mathcal{R}_2(j,t) = \{\mathbf{x} \in \mathcal{R} : x_j > t\}$.

Choosing j and t: make $\mathcal{R}_1$ and $\mathcal{R}_2$ as **"pure"** as possible.

Stop splitting a region if it contains less than $n_{\min}$ samples or only one class.

★Step 2: **Prune** the tree using some penalty on complexity.

Remarks

- **Impurity measures**: Gini index or cross-entropy. Given a node region R, let $p_k = \frac{1}{|R|}\sum_{\mathbf{x}_i \in R} I(y_i = c_k)$.

$$\mathrm{GI}(R) = \sum_k p_k(1-p_k), \quad \mathrm{CE}(R) = -\sum_k p_k \log(p_k).$$

- For given proposals $R_1(j,t)$ and $R_2(j,t)$, find

$$(j,t)^{\mathrm{opt}} = \mathrm{argmin}_{j,t}\left[\frac{|R_1(j,t)|}{|R|}\mathrm{GI}(R_1(j,t)) + \frac{|R_2(j,t)|}{|R|}\mathrm{GI}(R_2(j,t))\right],$$

- Adding a penalty α to each node, the chosen tree minimizes sum of impurity and penalty over all terminals. Choose α by CV.

Bootstrap aggregating

Classification trees are **interpretable** but highly **instable**.

Ensemble learning: a variety of tools for promoting **stability**.

Bootstrap aggregating (Bagging), *(Breiman, 1996, 23K)*

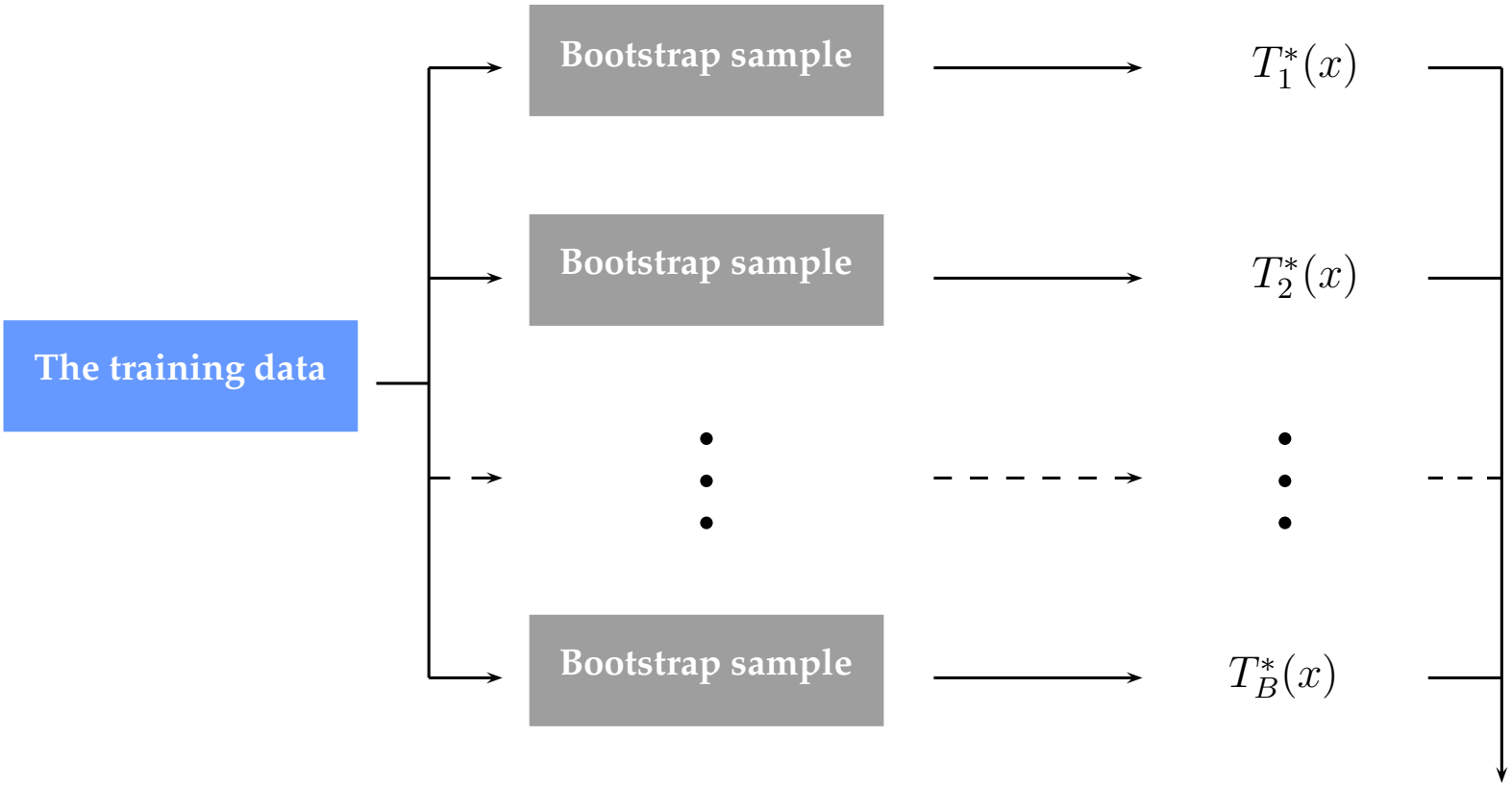
Let $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be the samples.

1. Draw B i.i.d. **bootstrap** samples $\{\mathcal{S}^b\}_{b=1}^B$ from $\mathcal{S}$.
2. On each $\mathcal{S}^b$, fit a classifier $\widehat{\delta}^b$ (e.g. classification tree).
3. Output the majority vote of $\{\widehat{\delta}^b\}_{b=1}^B$.

Bagging for regression: output the average $\frac{1}{B}\sum_{b=1}^B \widehat{y}^b$.

★Breiman (1996) suggested $B = 50$.

Illustration of Bagging



Final Classifier

$$\widehat{y}^{\text{bag}}(X) = \arg\max_{c_k \in \mathcal{C}} \sum_{b=1}^{B} I\left(T_b^*(x) = c_k\right)$$

Random Forests

In Bagging, correlation among samples hinders variance reduction.

Random Forest: use random dropouts to decorrelate

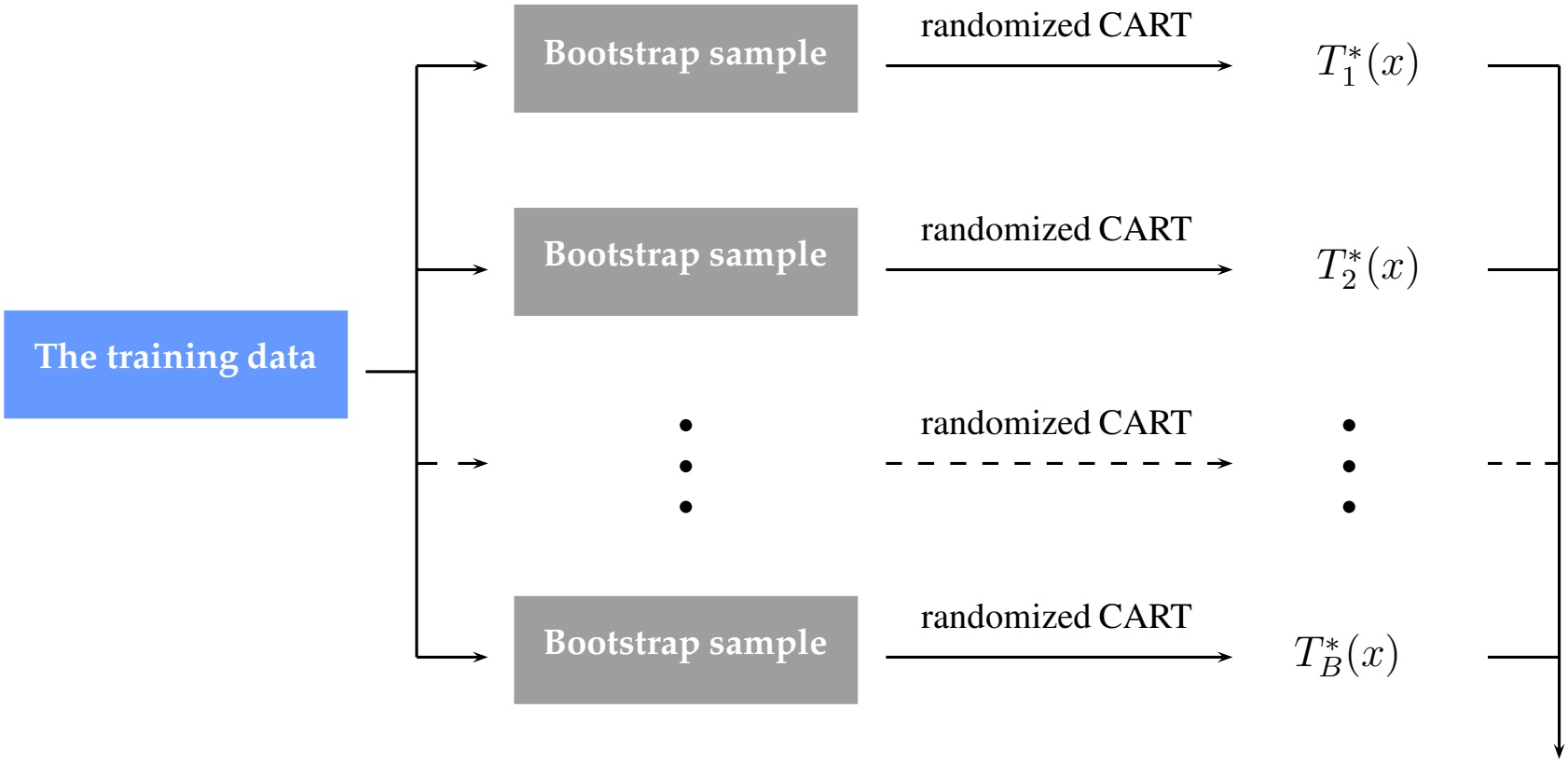
From Bagging to Random Forest *(Breiman, 2001, 56K)*

- Step 2 in Bagging: fit a tree $\widehat{\delta}^b$ on each $\mathcal{S}^b$.
- Step 2 in Random Forest for constructing $\widehat{\delta}^b$: use only m **randomly-selected** features (out of p features).

Breiman (2001) suggested $m = \lfloor \sqrt{p} \rfloor$ for classification and $p/3$ for regression

R ★ `randomForest`

Illustration of Random Forest



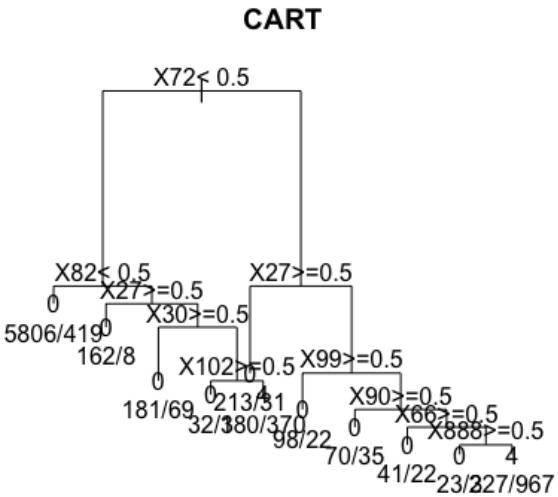
Final Classifier

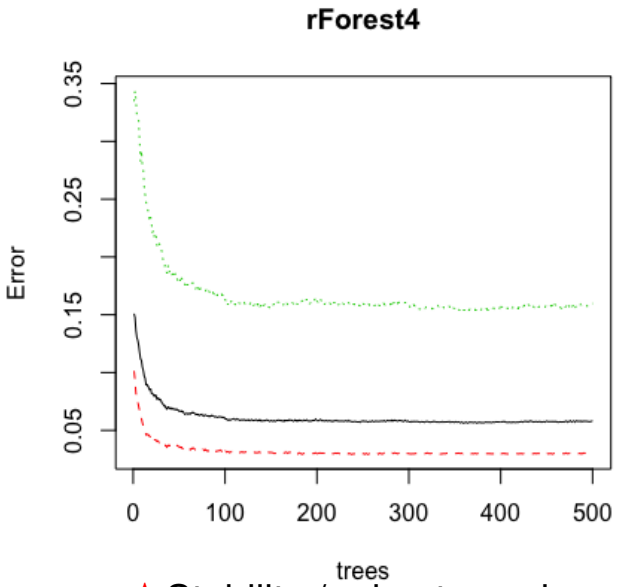
$$\widehat{y}^{\mathrm{RF}}(X) = \arg\max_{c_k \in \mathcal{C}} \sum_{b=1}^{B} I\Big(T_b^*(x) = c_k\Big)$$

Reuters Newswires: 11228 short newswires and their topics (46 topics) in 1986.

```
##################### fitting CART and Random Forest with 46 topics#####################
y_train1 <- as.factor(reuters$train$y)          #using 46 topics
y_test1  <- as.factor(reuters$test$y)
x_train1 = data.frame(x_train)                  #create data.frame to pass along
x_test1  = data.frame(x_test)

####install.packages("rpartt")                  #installking "rpartt" package
library(rpart)
cart = rpart(y_train1 ~ ., data=x_train1)       #fitting CART to the data
mean(predict(cart,type="class")==y_train1)      # 1 - training errors
##[1] 0.6032064
cart.test = predict(cart,x_test1,type="class")  #predict the data
mean(cart.test == y_test1)                      #proportion of correct prediction
##[1] 0.5837044

###install.packages("randomForest")             #installking "randomForest" package
library(randomForest)
rForest = randomForest(y_train1 ~ ., data=x_train1) #fitting randomForest
mean(predict(rForest,type="class")==y_train1)       # 1 - training errors
##[1] 0.7812291
rForest.test = predict(rForest,x_test1,type="class") #predict the data
mean(rForest.test == y_test1)                        #proportion of correct pred
##1] 0.7720392

#################### fitting CART and Random Foest with topic 4 only #################
y_train4 = reuters$train$y; y_train4[y_train4 != 4] = 0   #topic 4 data
```

```
y_train4 = as.factor(y_train4)
y_test4 = reuters$test$y; y_test4[y_test4 != 4] = 0       #topic 4 data

cart4 = rpart(y_train4 ~ ., data=x_train1)                #Fitting CART
cart.test4 = predict(cart4,x_test1,type="class")          #predict the data
mean(cart.test4 == y_test4)                               #proportion of correction prediciton
#[1] 0.8726625

rForest4 = randomForest(y_train4 ~ ., data=x_train1)     #Fitting Random Forest
rForest.test4 = predict(rForest4,x_test1,type="class")   #predict the data
mean(rForest.test4 == y_test4)                           #proportion of correction prediciton
##[1] 0.9394479

par(mfrow=c(1,2)); par(xpd = TRUE);
plot(cart4, compress = TRUE, main="CART");          #plotting CART partition
text(cart4, use.n = TRUE)
plot(rForest4);                                     #plot of random forest errors
```





★Improvements by Random Forest are evidenced.

★Stability / robustness important

Boosting

Boosting is also an **ensemble learning** algorithm: (*Freund & Schapire, 1995, 19K*)

- it requires a **weak classifier** that is easily trained on **weighted data**;
- it outputs a **weighted** majority vote using $\sum_{m=1}^{M} \alpha_m \widehat{\delta}_m(\mathbf{x})$;
- $\widehat{\delta}_m$ (experts) and α_m (credibility) are obtained **sequentially**.

Idea of Boosting: Let $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be the samples and $w_1 = \cdots = w_n = 1/n$.

For $m = 1, 2, \cdots, M$:

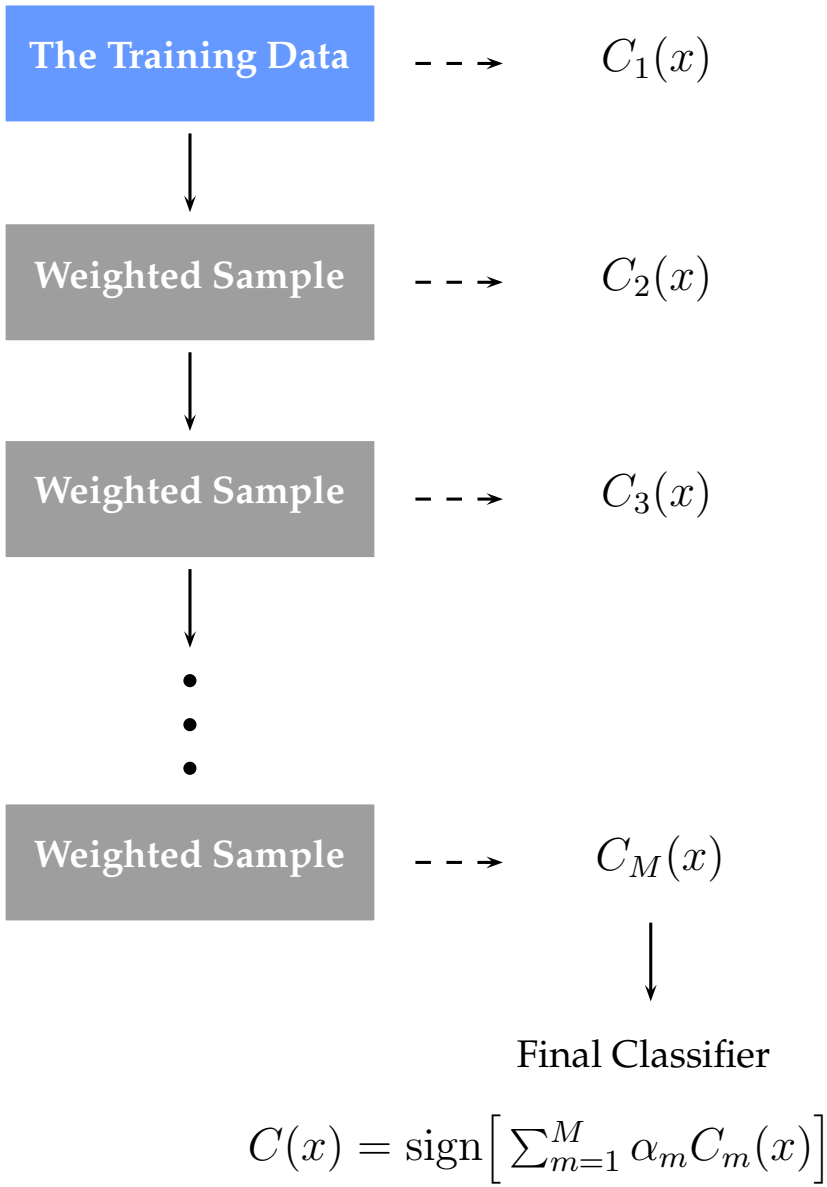
1. Train a classifier $\widehat{\delta}_m$ to min the **weighted error** $\sum_{i=1}^n w_i \mathbf{1}(y_i \neq \widehat{\delta}_m(\mathbf{x}_i))$.
2. Compute α_m using a decreasing function of the weighted error.
3. Update $\{w_i\}_{i=1}^n$ by assigning more weights to $\{i : \ y_i \neq \widehat{\delta}_m(\mathbf{x})\}$.

Illustration of AdaBoost Algorithm

Binary label: $Y = \pm 1$.

Adaptive Boosting uses

- $\alpha_m = \log(\frac{1-err_m}{err_m})$ with
  err_m is the weighted error;
- update $w_i = w_i' / \sum_{j=1}^n w_j'$ with
  $w_i' = w_i \exp[\alpha_m \mathbf{1}(y_i \neq \widehat{\delta}_m(\mathbf{x}_i))]$



$$C(x) = \text{sign}\Big[\sum_{m=1}^M \alpha_m C_m(x)\Big]$$

Relation with Forward Additive model

Minimize sequentially **exponential loss** $L(f) = \sum_{i=1}^n e^{-y_i f(\mathbf{x}_i)}$ using additive model $f(\mathbf{x}) = \sum_{m=1}^M \beta_m C_m(\mathbf{x})$.

- ★ Given $f_{m-1}(\mathbf{x}) = \sum_{k=1}^{m-1} \beta_k C_k(\mathbf{x})$, by $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m C_m(\mathbf{x})$, we have

$$L(f) = \sum_{i=1}^n w_i^{(m)} \underbrace{\exp\left(-y_i \beta_m C_m(\mathbf{x}_i)\right)}_{\exp(-\beta_m) I(y_i = C_m(x_i)) + \exp(\beta_m) I(y_i \neq C_m(x_i))}, \qquad w_i^{(m)} = \exp(-y_i f^{(m-1)}(\mathbf{x}_i)).$$

- ★ Given β_m, we have $\widehat{C}_m(\mathbf{x}) = \operatorname{argmin}_{C_m(\mathbf{x})} \sum_{i=1}^n w_i^{(m)} I(y_i \neq C(\mathbf{x}_i))$

  weighted error

- ★ Given C_m, we have $\beta^{(m)} = \frac{1}{2} \log\left(\frac{\sum_{i=1}^n w_i^{(m)} I(y_i = C_m(\mathbf{x}_i))}{\sum_{i=1}^n w_i^{(m)} I(y_i \neq C_m(\mathbf{x}_i))}\right) = \frac{\alpha_m}{2}$.

- ★ $w_i^{(m+1)} = w_i^{(m)} \exp\left(-\frac{\alpha_m}{2} \underbrace{y_i \widehat{C}_m(\mathbf{x}_i)}_{1 - 2I(y_i \neq C_m(\mathbf{x}_i))}\right) \propto w_i^{(m)} \exp[\alpha_m \mathbf{1}(y_i \neq \widehat{C}_m(\mathbf{x}_i))]$

Gradient Boosting

Goal: Find sequentially $f_m(\mathbf{x}) = f_{m-1}(x) + \beta_m C_m(\mathbf{x})$, $C_m \in \mathcal{C}$ to min

$$L_n(f) = \sum_{i=1}^{n} L(f_m(\mathbf{x}_i), y_i).$$

Basic idea: Find β_m and $C_m \in \mathcal{C}$ to min "residuals" *(Breiman, 97)*

1. Given $f_{m-1}(x)$, compute pseudo-residuals:

$$r_{im} = -\left[\frac{\partial L(y_i, t)}{\partial t}\right]_{t=f_{m-1}(x_i)}.$$

2. Find $C_m \in \mathcal{C}$ using the training data $\{(\mathbf{x}_i, r_{im})\}_{i=1}^n$
3. Find γ_m to minimize $\sum_{i=1}^n L(f_{m-1}(\mathbf{x}_i) + \gamma C_m(\mathbf{x}_i), y_i)$

Gradient tree boosting

Base Learner: trees w/ J_m disjoint regions: $C_m(\mathbf{x}) = \sum_{j=1}^{J_m} b_{mj} I_{R_{mj}}(\mathbf{x})$

Modified: Instead of using one γ, use γ_{jm} so that

$$f_m(\mathbf{x}) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{mj} I_{R_{mj}}(\mathbf{x}), \qquad \gamma_{jm} = \text{argmin}_\gamma \sum_{\mathbf{x}_i \in R_{mj}} L(f_m(\mathbf{x}_i) + \gamma, y_i)$$

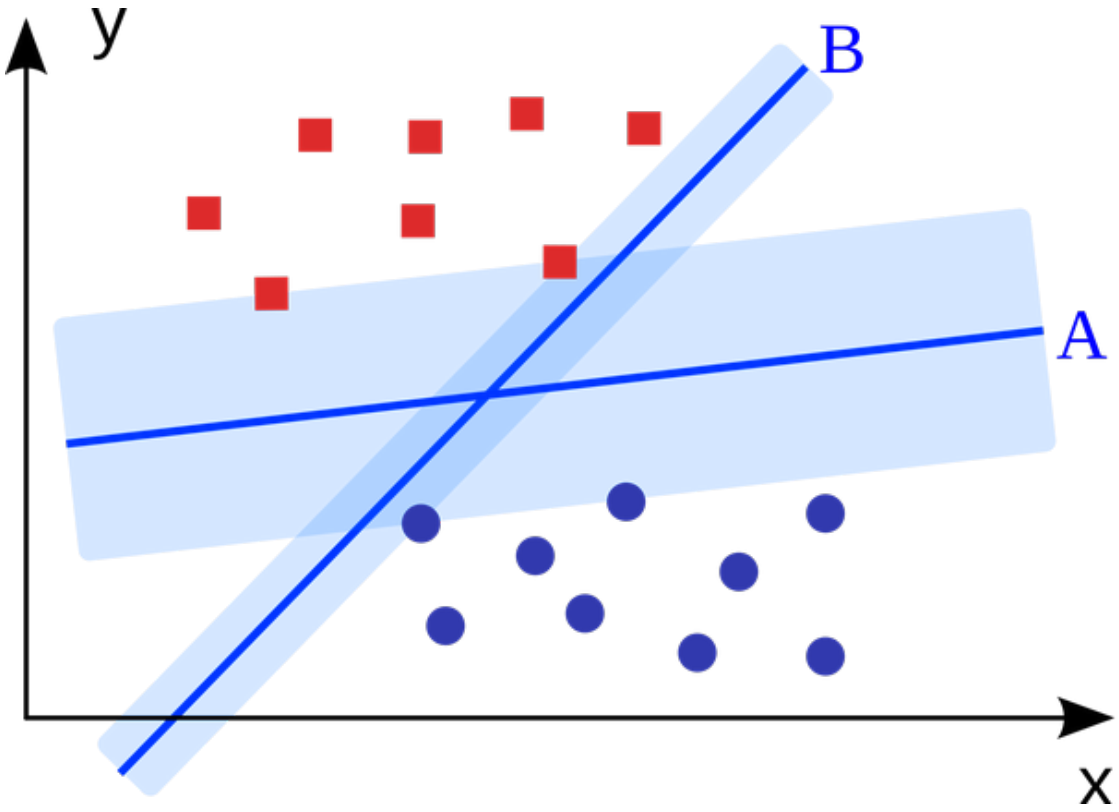
Regression: r_{mi} is a residual, $C_m(\mathbf{x})$ is the fitted tree to $\{(\mathbf{x}_i, r_{im})\}_{i=1}^n$.

See MART in Algorithm 12.3 for details (*Friedman, 2001*). ★xgboost

5.5 Support Vector Machines

Motivation for linear SVM

Which linear classifier do you prefer? A or B? wider margin



★A has wider margin, stability

A framework for margin maximization

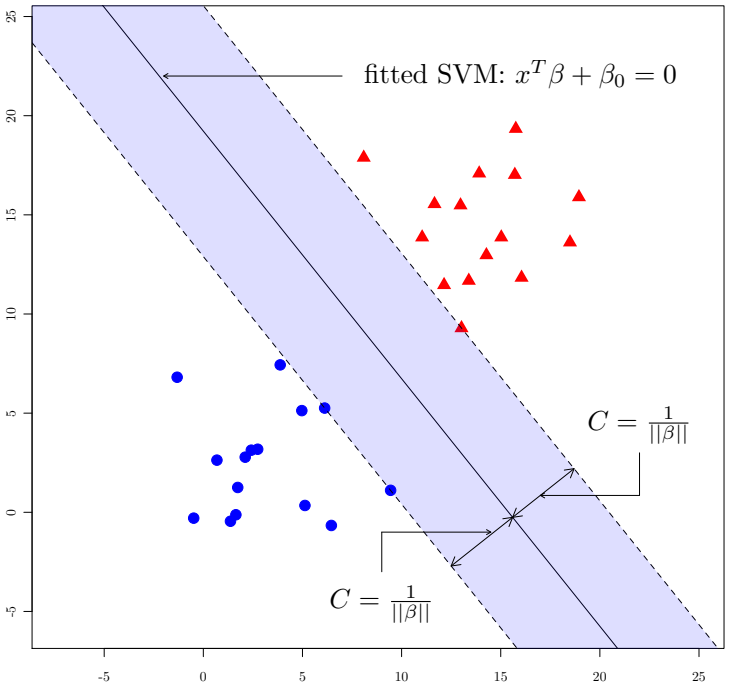
★ Training data: $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \{1, -1\}$

Linear classifier / hyperplane: $\{\mathbf{x} : \beta_0 + \mathbf{x}^\top \beta = 0, \text{and } \|\beta\|_2 = 1\}$

Linear separability if $y_i(\beta_0 + \mathbf{x}_i^\top \beta) > 0, \forall i$

SVM: find a linear classifier with largest margin / confidence

$$\begin{aligned} \max_{\beta_0, \beta : \|\beta\|_2 = 1} \quad & C \\ \text{s.t.} \quad & y_i(\beta_0 + \mathbf{x}_i^\top \beta) \geq C, \\ & \forall 1 \leq i \leq n \end{aligned}$$



How about non-seperable data?

SVM: ★ ξ_i: slack variables ★ $B > 0$: tuning parameter

$$\max_{\beta_0, \beta, \{\xi_i\}: \|\beta\|_2 = 1} \quad C,$$

$$\text{s.t.} \quad y_i(\beta_0 + \mathbf{x}_i^\top \beta) \geq C(1 - \xi_i), \qquad \forall 1 \leq i \leq n,$$

$$\xi_i \geq 0, \qquad \forall 1 \leq i \leq n,$$

$$\sum_{i=1}^{n} \xi_i \leq B$$

■ $B = 0 \Longrightarrow$ separable case

Equivalent formulation

$$\min_{\beta_0,\beta} \quad \frac{1}{n}\sum_{i=1}^{n}\left[1 - y_i(\beta_0 + \mathbf{x}_i^\top \beta)\right]_+ + \lambda\|\beta\|_2^2,$$

★ $(1-t)_+$ is often called hinge loss.

★ λ and B are **one-to-one**.

R `svm` in ★`e1071`

★ allow other basis: e.g. kernel SVM.

Proof. Two constraints become $\xi_i \geq [1 - y_i(\gamma_0 + \mathbf{x}_i^\top \gamma)]_+$, where $\gamma = \beta/C$. Hence $\|\gamma\| = 1/C$. The problem reduces to $\min \|\gamma\|^2, \quad s.t. \ \sum_{i=1}^n [1 - y_i(\gamma_0 + \mathbf{x}_i^\top \gamma)]_+ \leq B$. The conclusion follows by Lagrange multiplier method.

Kernel SVM

RKHS $\mathcal{H}_{\mathcal{K}}$ defined by a kernel $K(\mathbf{x}, \mathbf{x}')$

$$\Longrightarrow \quad \min_{f \in \mathcal{H}_K} \quad \frac{1}{n} \sum_{i=1}^{n} [1 - y_i f(\mathbf{x}_i)]_+ + \lambda \|f\|^2_{\mathcal{H}_K}.$$

Solution to kernel SVM is given by the representer's theorem:

$$\widehat{f}(\mathbf{x}) = \widehat{\beta}_0 + \sum_{i=1}^{n} \widehat{\beta}_i K(\mathbf{x}, \mathbf{x}_i), \qquad \text{with}$$

$$\min_{\beta_0, \beta} \quad \frac{1}{n} \sum_{i=1}^{n} \Big[1 - y_i(\beta_0 + \sum_{l=1}^{n} \beta_l K(\mathbf{x}_i, \mathbf{x}_l))\Big]_+ + \lambda \beta^T \mathbf{K} \beta,$$

★ $\mathbf{K} = [K(\mathbf{x}_i, \mathbf{x}_l)]$ is the kernel matrix

Why Hinge Loss?

Numerical advantage: easy to compute gradient

Statistical explanation: Let $f^*(\mathbf{x}) = \text{argmin}_f \, \mathrm{E}\left([1 - Yf(\mathbf{X})]_+\right)$.

★ population minimizer corresponds to Bayes rule:

$$\text{sign}(f^*(x)) = \text{sign}(p_+(\mathbf{x}) - p_-(\mathbf{x}))$$

where $p_+(\mathbf{x}) = \Pr(Y = 1 | \mathbf{X} = \mathbf{x})$ and $p_-(\mathbf{x}) = \Pr(Y = -1 | \mathbf{X} = \mathbf{x})$.

★ **without** estimating high-dim conditional class prob.

Extension: Fisher consistent loss

★ Let $\phi(t)$ be a general loss function

★ Risk: $\mathrm{E}\left(\phi(Yf(\mathbf{X}))\right)$ and conditional risk: $\mathrm{E}\left[\phi(Yf(\mathbf{X})) \mid \mathbf{X}\right]$

Why? error rate $= n^{-1}\sum_{i=1}^{n} I\{Y_i \neq \mathrm{sgn}(f(X_i))\} = n^{-1}\sum_{i=1}^{n} I\{Y_i f(X_i) < 0\}$, ★use surrogate $\phi(\cdot)$

Fisher consistent for ϕ if

$$\mathrm{sign}(f^*(\mathbf{x})) = \mathrm{sign}(p_+(\mathbf{x}) - p_-(\mathbf{x})) \quad \text{for any} \quad \mathbf{x},$$

where $f^*(\mathbf{x}) = \mathrm{argmin}_f \quad \mathrm{E}\left[\phi(Yf(\mathbf{X})) \mid \mathbf{X} = \mathbf{x}\right]$.

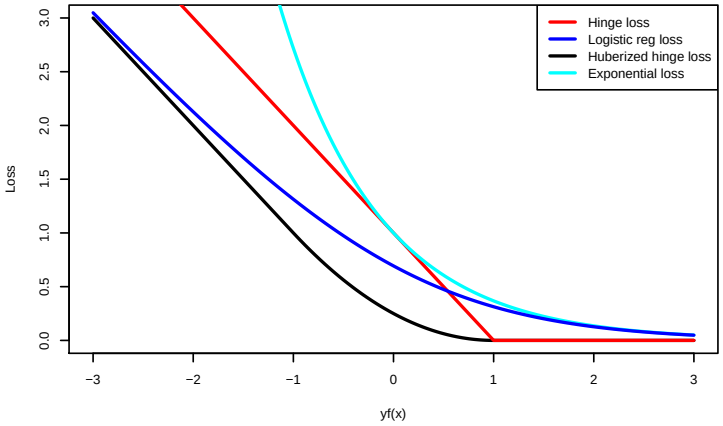
Examples: ■logistic regression loss $\log(1 + e^{-t})$

■ hinge loss $(1 - t)_+$ ■ exponential loss e^{-t}

General large margin classifier

Formulation: for a Fisher consistent ϕ

$\min_{f \in \mathcal{H}_K} \frac{1}{n} \sum_{i=1}^n \phi(y_i f(\mathbf{x}_i)) + \lambda \|f\|^2_{\mathcal{H}_K}$



Solution: $\widehat{f}(\mathbf{x}) = \beta_0 + \sum_{i=1}^n \beta_i K(\mathbf{x}, \mathbf{x}_i)$

and $\widehat{\beta}_0, \widehat{\beta}$ minimize

$$\min_{\beta_0, \beta} \quad \frac{1}{n} \sum_{i=1}^n \phi\left(y_i (\beta_0 + \sum_{l=1}^n \beta_l K(\mathbf{x}_i, \mathbf{x}_l)) \right) + \lambda \beta^T \mathbf{K} \beta$$

5.6 Sparse Classifiers

Sparse SVM and large margin classier

ℓ_1 **SVM**: replace ℓ_2- by ℓ_1-penalty

$$\min_{\beta_0,\beta} \quad \frac{1}{n}\sum_{i=1}^{n}\left[1 - y_i(\beta_0 + \mathbf{x}_i^T\beta)\right]_+ + \lambda\|\beta\|_1$$

Sparse large margin classifier: For concave $p_\lambda(\cdot)$

$$\min_{\beta_0,\beta} \quad \frac{1}{n}\sum_{i=1}^{n}\phi\left(y_i(\beta_0 + \mathbf{x}_i^T\beta)\right) + \sum_j p_\lambda(|\beta_j|).$$

★ $\phi^{\text{logit}}(t) = \log(1 + e^{-t}) \Longrightarrow$ penalized logistic regression.

★ $\phi^{\text{SVM}}(t) = (1 - t)_+ \Longrightarrow$ sparse SVM

Feature augmentation

Nonlinear features: $z_j = \log(\frac{f_{2j}(x_j)}{f_{1j}(x_j)})$, **optimal** using only j^{th} feature; **easily** learnable from data. (KDE w/ ROB)

Logistic reg: $\log\left(\frac{p(\mathbf{x})}{1-p(\mathbf{x})}\right) = a + \beta_1 z_1 + \cdots + \beta_p z_p$

- ★ $a = 0, \beta_1 = \cdots = \beta_p = 1 \Longrightarrow$ **Naive Bayes** $\Longrightarrow$ improvements of NB
- ★ can be combined with original features:

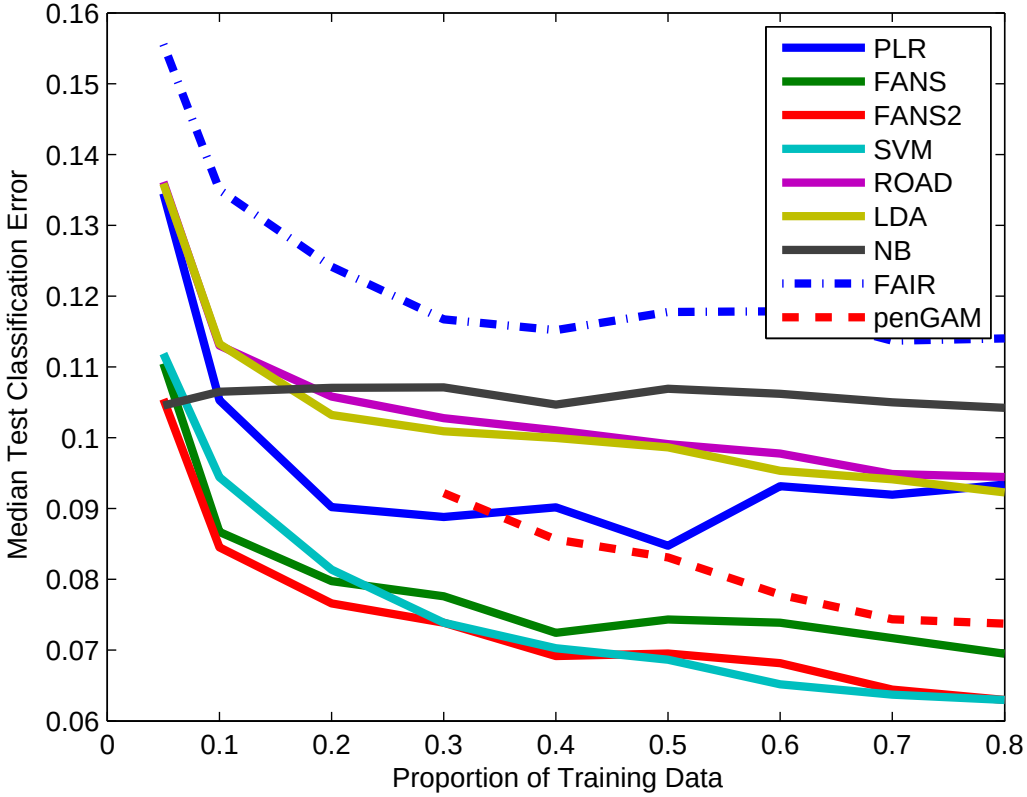
$$\log\left(\frac{p(\mathbf{x})}{1-p(\mathbf{x})}\right) = a + \beta_1 Z_1 + \cdots + \beta_p Z_p + \alpha_1 X_1 + \cdots + \alpha_p X_p$$

- ★ applied to other large-martin losses

Email classification

E-mail data: $n = 4601$, 40% spam; $p = 57$ predictors (UCI)

Results: different split ratios and methods



Penalized additive logistic regression

Generalized additive model for logistic regression:

$$\underbrace{f(\mathbf{X})}_{\text{log-odds function}} = \beta_0 + \sum_{j=1}^{p} f_j(X_j) \qquad \text{and} \qquad P(Y=1|\mathbf{X}) = \frac{e^{f(\mathbf{X})}}{1+e^{f(\mathbf{X})}}$$

Basis expansion using B-splines:

$$f_j(X_j) = \sum_{m=1}^{M_j} \beta_{jm} B_m(X_j).$$

Penalized generalized additive model (Pen-GAM):

$$\text{argmin}_\beta \frac{1}{n} \sum_{i=1}^{n} \log(1+e^{-y_i f(\mathbf{x}_i)}) + \lambda \sum_{j=1}^{p} \sqrt{M_j} \|\beta^{(j)}\|_2$$

★ $\|\beta^{(j)}\|_2$: group-lasso penalty