Topic X:
Classification Methods

Wei You

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Fall, 2023

Introduction

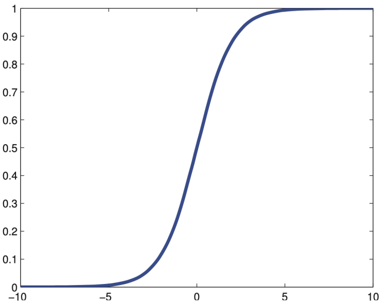
Suppose we are given a set of data $\{(\boldsymbol{X}_i, Y_i), 1 \leq i \leq n\}$

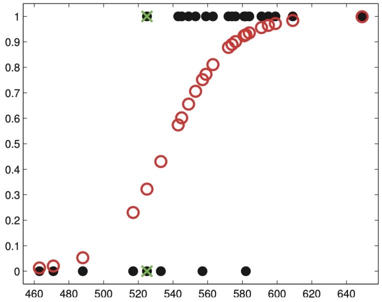
- $\boldsymbol{X} \in \mathbb{R}^p$: predictors;
- Y: class label, taking value in a discrete set $\mathcal{C}$.

Classification rule

A classification rule is a mapping $\delta : \mathbb{R}^p \to \mathcal{C}$ such that a label $\delta(\boldsymbol{X})$ is assigned to each data point $\boldsymbol{X}$.

Logistic Regression as a Classifier

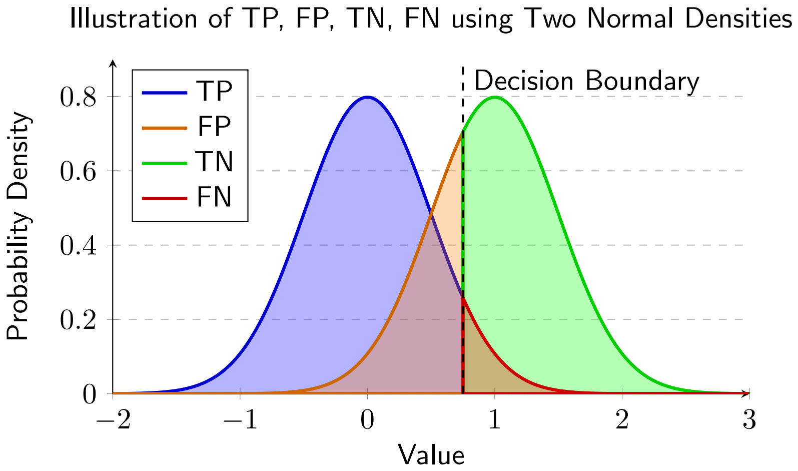




Prediction of the success probability: $\mathbb{P}(Y = 1|x) = \mu = \frac{1}{1+\exp(-\eta)}$.

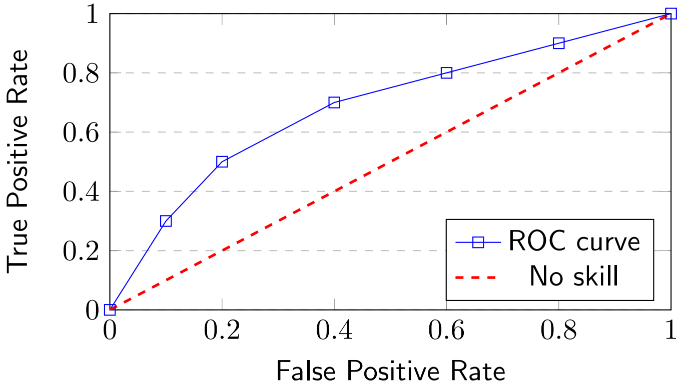
Classification/decision rule: $\widehat{Y} = 1 \iff \mathbb{P}(Y = 1|x) > 0.5$.

Receiver operating characteristic (ROC) curve



ROC curve

The ROC curve is a plot of the true positive rate (probability of detection, TPR = TP/(TP+FN)) against the false positive rate (probability of false alarm, FPR = FP/(FP+TN)) for the different possible cutoff points.



Bayes Rule

Misclassification error

Expected 0-1 loss

$$R(\delta) = \mathbb{P}(Y \neq \delta(\boldsymbol{X})) = \mathbb{E}_{\boldsymbol{X},Y}[\mathbb{1}(Y \neq \delta(\boldsymbol{X}))].$$

Bayes classifier

A Bayes rule is defined as $\delta^*(\boldsymbol{X}) = \arg\max_{c \in \mathcal{C}} \mathbb{P}(Y = c \mid \boldsymbol{X})$.

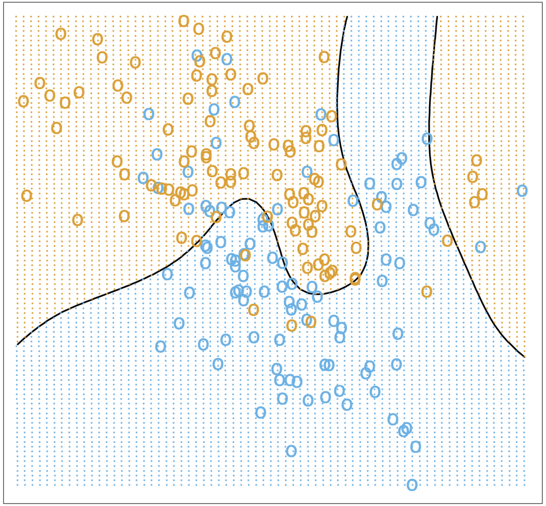
The Bayes rule minimizes the misclassification error

Let's prove it under the simple case of $\mathcal{C} = \{0, 1\}$.

Let $\eta(i) = \mathbb{P}(Y = i \mid \boldsymbol{X})$ for $i = 0, 1$.

$$\begin{aligned}
R(\delta) &= \mathbb{E}_{\boldsymbol{X},Y}[\mathbb{1}(Y \neq \delta(\boldsymbol{X}))] \\
&= \mathbb{E}_{\boldsymbol{X}}\left[\mathbb{E}_{Y|\boldsymbol{X}}[\mathbb{1}(Y \neq \delta(\boldsymbol{X}))]\right] \\
&= \mathbb{E}_{\boldsymbol{X}}\left[\mathbb{E}_{Y|\boldsymbol{X}}[\mathbb{1}(Y = 0, \delta(\boldsymbol{X}) = 1) + \mathbb{1}(Y = 1, \delta(\boldsymbol{X}) = 0]\right] \\
&= \mathbb{E}_{\boldsymbol{X}}[\mathbb{P}(Y = 0 \mid \boldsymbol{X})\mathbb{1}(\delta(\boldsymbol{X}) = 1) + \mathbb{P}(Y = 1 \mid \boldsymbol{X})\mathbb{1}(\delta(\boldsymbol{X}) = 0)] \\
&= \mathbb{E}_{\boldsymbol{X}}[\eta(0)\mathbb{1}(\delta(\boldsymbol{X}) = 1) + (1 - \eta(0))(1 - \mathbb{1}(\delta(\boldsymbol{X}) = 1))] \\
&= \mathbb{E}_{\boldsymbol{X}}[(2\eta(0) - 1)\mathbb{1}(\delta(\boldsymbol{X}) = 1) + 1 - \eta(0)].
\end{aligned}$$

Bayes Rule



Bayes Rule

The major obstacle in applying the Bayes rule directly is that $\mathbb{P}(Y = c \mid \boldsymbol{X} = \boldsymbol{x})$ is in general not readily available.

The goal is to model/approximate the Bayes rule as closely as possible, given a finite amount of data.

Let's start with the model-based methods.

Bayes Rule

Recall the Bayes formula

$$\mathbb{P}(Y = c_k \mid \boldsymbol{X} = \boldsymbol{x}) = \frac{\pi_k f_k(\boldsymbol{x})}{f(\boldsymbol{x})},$$

where $\pi_i = \mathbb{P}(Y = c_k)$, $f_k(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{X} = \boldsymbol{x} \mid Y = c_k)$ and $f(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{X} = \boldsymbol{x}) = \sum_k f_k(\boldsymbol{x})$. Hence, modeling the posterior label probability is equivalent to modeling the conditional distribution f_k of $\boldsymbol{x}$ given the label and the class probabilities π_k.

- Exact formula for π_c and $f_c(\boldsymbol{X})$ is not available.
- π_c can be estimated using n_c/n.
- $f_c(\boldsymbol{X})$ can be estimated using kernel density estimator, see Section 12.2 of Fan et al. (2020).

Quadratic Discriminant Analysis

The linear and quadratic discriminant analysis directly models the conditional distribution as a multivariate normal distribution: $\boldsymbol{X} \mid Y = c_k \sim N(\boldsymbol{\mu}_k, \Sigma_k)$.

Bayes rule for QDA

$$\arg\max_k \delta^{\mathsf{qda}}(\boldsymbol{x})$$

where

$$\delta^{\mathsf{qda}}(\boldsymbol{x}) = \log \pi_k - \frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T\Sigma_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k).$$

- The Bayes rule in QDA is a quadratic function of $\boldsymbol{x}$.
- Except for some constants, the QDA classifies a point according to its Mahalanobis distance $(\boldsymbol{x}-\boldsymbol{\mu}_k)^T\Sigma_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)$ to the centroids $\boldsymbol{\mu}_k$.

The QDA approximates the Bayes rule by substituting the parameters $\boldsymbol{\mu}_k, \Sigma_k$ with the estimates

$$\begin{aligned}
\widehat{\pi}_k &= \frac{n_k}{n}, \\
\widehat{\boldsymbol{\mu}}_k &= \frac{1}{n_k} \sum_{Y_i = c_k} \boldsymbol{X}_i, \\
\widehat{\Sigma}_k &= \frac{1}{n_k - 1} \sum_{Y_i = c_k} (\boldsymbol{X}_i - \widehat{\boldsymbol{\mu}}_k)^T (\boldsymbol{X}_i - \widehat{\boldsymbol{\mu}}_k).
\end{aligned}$$

QDA

$$\arg\max_k \left\{ \log \widehat{\pi}_k - \frac{1}{2} \log |\widehat{\Sigma}_k| - \frac{1}{2} (\boldsymbol{x} - \widehat{\boldsymbol{\mu}}_k)^T \widehat{\Sigma}_k^{-1} (\boldsymbol{x} - \widehat{\boldsymbol{\mu}}_k) \right\}$$

Linear Discriminant Analysis

The linear discriminant analysis (LDA) is a special case of the QDA with the additional homogeneous assumption

$$\Sigma_k = \Sigma, \text{ for all } c_k \in \mathcal{C}.$$

As a result, the normalizing constants and the quadratic termsm, $\boldsymbol{x}^\top \Sigma^{-1} \boldsymbol{x}$, is independent of the class label c_k.

Bayes rule for LDA

$$\arg\max_k \delta^{\mathsf{lda}}(\boldsymbol{x})$$

where

$$\delta^{\mathsf{lda}}(\boldsymbol{x}) = \log \pi_k + \boldsymbol{x}^T \Sigma^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \boldsymbol{\mu}_k^T \Sigma^{-1} \boldsymbol{\mu}_k.$$

The Bayes rule is a linear function of $\boldsymbol{x}$, hence the name linear discriminant.

Linear Discriminant Analysis

The LDA estimate the covariance matrix by the pooled sample variance:

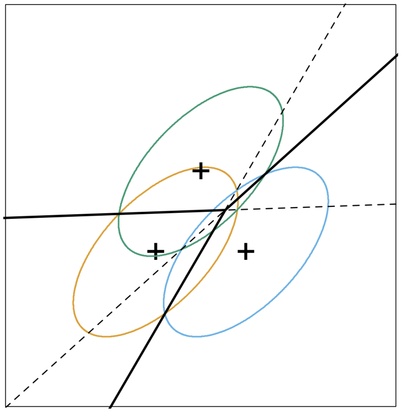
$$\widehat{\Sigma} = \frac{1}{\sum_{k=1}^{K}(n_k - 1)} \sum_{k=1}^{K} (n_k - 1)\widehat{\Sigma}_k$$

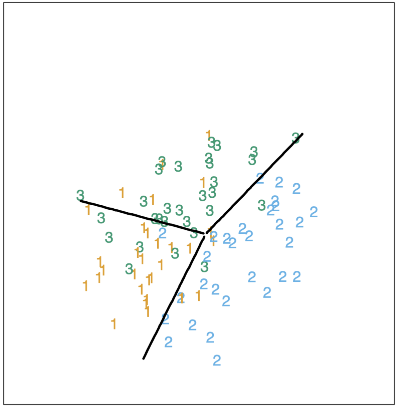
LDA

$$\arg\max_k \left\{ \log \widehat{\pi}_k + \boldsymbol{x}^T \widehat{\Sigma}^{-1} \widehat{\boldsymbol{\mu}}_k - \frac{1}{2} \widehat{\boldsymbol{\mu}}_k^T \widehat{\Sigma}^{-1} \widehat{\boldsymbol{\mu}}_k \right\}$$

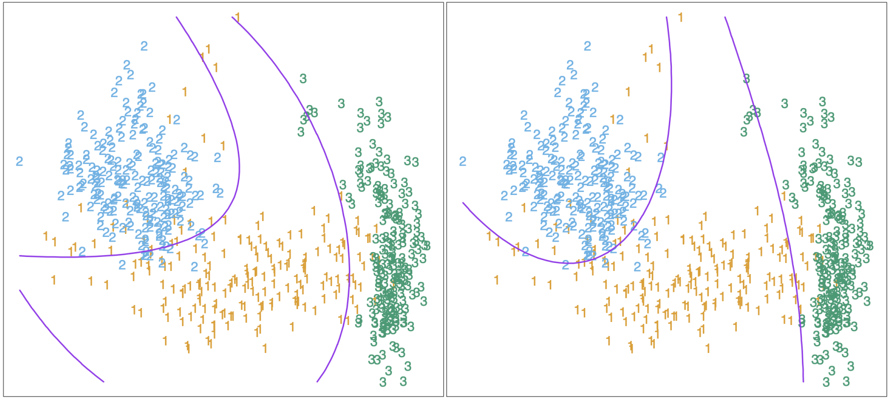
LDA is also referred to as Fisher's discriminant analysis.

Example: LDA





Example: (Left) LDA with quadridic boundary v.s. (right) QDA



These two are quite similar, with QDA generally preferred. But QDA is more computationally expensive, needing to estimate the covariance matrix for each class.

Regularized Discriminant Analysis

When p is reasonably large, $\Sigma \in \mathbb{R}^{p\times p}$ cannot be accurately estimated for smaller sample size, and the calculation of $\widehat{\Sigma}^{-1}$ can be unstable. For example, when $p > n$, the covariance matrix is not full rank.

The regularized discriminant analysis (RDA) propose to use the shrinkage estimator

$$\widehat{\Sigma}^{\text{rda}}(\gamma) = \gamma\widehat{\Sigma} + (1-\gamma)\frac{\text{tr}(\widehat{\Sigma})}{p}I, \quad 0 \le \gamma \le 1,$$

$$\widehat{\Sigma}_k^{\text{rda}}(\alpha) = \alpha\widehat{\Sigma}_k + (1-\alpha)\widehat{\Sigma}^{\text{rda}}(\gamma), \quad 0 \le \alpha \le 1.$$

- In practice, (α, γ) are chosen from the data by cross-validation.

Connection to Logistic Regression

Consider $\mathcal{C} = \{0, 1\}$. The log-odds under the LDA model is

$$\begin{aligned}\text{logodds}(\boldsymbol{x}) &= \log \frac{\mathbb{P}(Y = 1 \mid \boldsymbol{X} = \boldsymbol{x})}{\mathbb{P}(Y = 0 \mid \boldsymbol{X} = \boldsymbol{x})} \\ &= \log \frac{\pi_1}{\pi_0} + \boldsymbol{x}^T \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_0)^T \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) \\ &= \beta_0 + \boldsymbol{\beta}^T \boldsymbol{x},\end{aligned}$$

where $\beta_0 = \log \frac{\pi_1}{\pi_0} - \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_0)^T \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$ and $\boldsymbol{\beta} = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$.

- LDA corresponds to the linear logistic regression model with a specific $(\beta_0, \boldsymbol{\beta})$, which is different from that obtained from the logistic regression.
- Difference: LDA explicitly models $\mathbb{P}(\boldsymbol{X} \mid Y)$ as normal, whereas logistic regression does not specify (or care about) it at all.
    - Logistic regression is more robust for non-Guassian distributions.
    - LDA is more efficient under normal assumption.

The Nearest Neighbor Classifier

The nearest neighbor classifier is a localized classification algorithm in the predictor space. The idea is that predictors that are close to each other are more likely to share the same label.

Closeness is defined by a distance metric, e.g.

$$d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_q = (\sum_{j=1}^{p} |x_j - x'_j|^q)^{1/q},$$

- Manhattan distance: l_1 norm for $q = 1$.
- Euclidean distance: l_2 norm for $q = 2$.
- Hamming distance: for $q = 0$, i.e. $d(\boldsymbol{x}, \boldsymbol{x}') = \sum_{j=1}^{p} \mathbb{1}(x_j \neq x'_j)$.

The Nearest Neighbor Classifier

k nearest neighbours

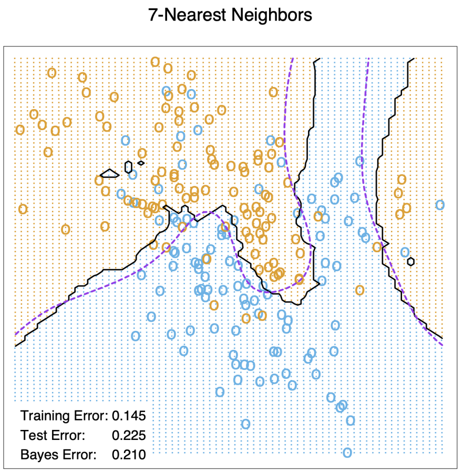
For any predictor $\boldsymbol{x}$, define $\mathcal{N}_k(\boldsymbol{x})$ as the k-nearest neighbours of $\boldsymbol{x}$ measured using a given distance metric d.

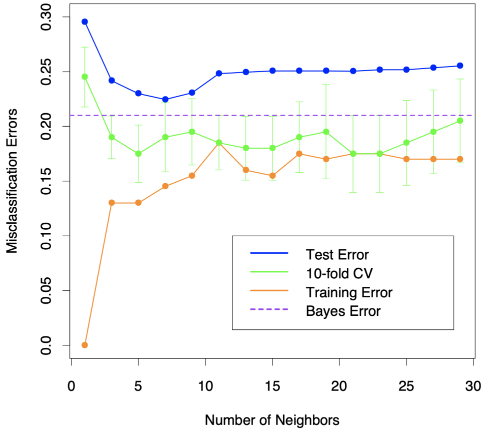
k-NN classifier

The nearest neighbor classifier predicts the class label according to the majority vote of the neighbors

$$\operatorname*{arg\,max}_{c_j \in \mathcal{C}} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} \mathbb{1}(Y_i = c_j).$$

Example: k-NN approximating Bayes rule.





[Figures from ESL Chapter 13]

Remarks – k-NN

Advantages

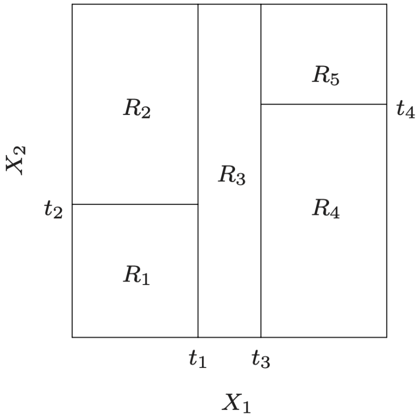
- Easy to program.
- Requires no training at all.
- Can learn complex target functions.

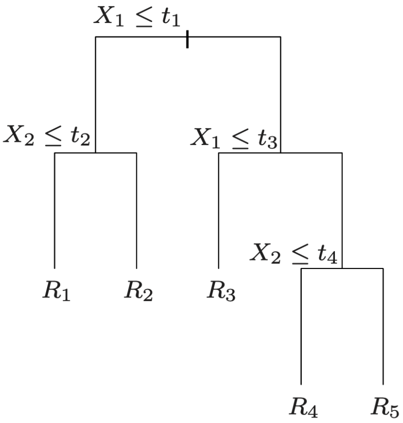
Disadvantages

- Slow at query time. Need to go through the entire data set in the worst case.
- Easily fooled by irrelevant attributes.
- Curse-of-dimensionality. Nearest neighbors can be far away.

Classification Trees

A <u>tree-structured classifier</u> is constructed by recursively partitioning the predictor space.





Classification Trees

Similar to k-NN, the idea of classification tree is that predictors in the same leaf should be more likely to share the same label.

Given the partition regions (the leaves) $\{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_S\}$, and a predictor $\boldsymbol{x}$, the predicted label is again the majority vote

$$\widehat{y} = \arg\max_{c_k \in \mathcal{C}} \sum_{\boldsymbol{X}_i \in \mathcal{R}(\boldsymbol{x})} \mathbb{1}(Y_i = c_k).$$

Here $\mathcal{R}(\boldsymbol{x})$ is the leaf that contain $\boldsymbol{x}$.

- Partitioned regions may not be rectangle.
- Trees may have multiway splits. However, binary splits are preferred: (1) multiway splits fregments the training data too quickly; (2) a multiway split can be achieved using binary splits.

CART

Classification And Regression Tree (CART) is a popular tree-based method for regression and classification.

- CART uses binary splits.
- CART uses one splitting variable in each split

Decision stump

A decision stump takes the form of

$$\mathcal{R}_1(j,t) = \{\boldsymbol{X} \in \mathcal{R} : X_j \leq t\}, \quad \mathcal{R}_2(j,t) = \{\boldsymbol{X} \in \mathcal{R} : X_j > t\}.$$

In particular, the partitions are rectangles.

- CART splits each leaf by myopically minimize the “impurity” after splitting.

CART uses greedy search for the optimal split (j, t) that minimizes node impurity.

Impurity functions

Let $\mathcal{R}$ be the node to be split into two regions. Define the proportion of label k in a region $\mathcal{R}$ as

$$p_k = \frac{1}{|\mathcal{R}|} \sum_{\boldsymbol{X}_i \in \mathcal{R}} \mathbb{1}(Y_i = c_k), \quad k = 1, 2, \dots, p.$$

- Gini impurity $\mathrm{GI}(\mathcal{R}) = \sum_k p_k(1 - p_k)$.
- Cross-entropy $\mathrm{CE}(\mathcal{R}) = -\sum_k p_k \log(p_k)$.

Notice that for small values of p_k, the Gini impurity is much less than the cross-entropy (p_k^2 versus $-p_k \log(p_k)$). For cases where data are imbalanced, cross-entropy is preferred.

Optimal split

$$(j,t)^{\mathsf{opt}} = \arg\min_{j,t}\left[\frac{|\mathcal{R}_1(j,t)|}{|\mathcal{R}|}F(\mathcal{R}_1(j,t)) + \frac{|\mathcal{R}_2(j,t)|}{|\mathcal{R}|}F(\mathcal{R}_2(j,t))\right],$$

where $F = GI$ or CE is a impurity function and

$$\mathcal{R}_1(j,t) = \{\boldsymbol{X} \in \mathcal{R} : X_j \le t\}, \quad \mathcal{R}_2(j,t) = \{\boldsymbol{X} \in \mathcal{R} : X_j > t\}.$$

If we split each current leaf until all leaves contain exactly one observation, then a fully-grown tree is obtained.

We need to decide when to stop growing the tree.

- If a tree is too deep[1], we overfit.
- If a tree is too shallow, we underfit.

CART prune the fully grown tree to a smaller one.

- A positive penalty term α is assigned to each leaf.
- The chosen tree minimizes the sum of the impurity and penalty over all leaves.
- The parameter α is chosen by cross-validation.

[1]The depth of a tree is the maximum length of the path from the root node to a leaf.

Regression Tree

Classification tree can be easily adapted to regression problems.

- For a set of data $\{(\boldsymbol{X}_i, Y_i), 1 \le i \le n\}$, a regression problem asks for a function f such that $f(\boldsymbol{X}_i) \approx Y_i$.

- Consider the following parametric function

$$f(x) = \sum_{k=1}^{S} \beta_k \mathbb{1}(x \in \mathcal{R}_k)$$

  where $\mathcal{R}_k$ are leaves of a decision tree.
- The impurity of a leaf can be the mean square error.
- The label of a leaf can be the sample mean.

As before, regression tree uses greedy search heuristics for a good partition.

- Start with a root $\mathcal{R} = \mathbb{R}^p$.
- For each leaf $\mathcal{R}$,
  - Split the leaf according to

$$(j,t)^{\mathsf{opt}} = \arg\min_{j,t} \left[\frac{|\mathcal{R}_1(j,t)|}{|\mathcal{R}|} F(\mathcal{R}_1(j,t)) + \frac{|\mathcal{R}_2(j,t)|}{|\mathcal{R}|} F(\mathcal{R}_2(j,t)) \right],$$

    where

$$F(\mathcal{R}) = \frac{1}{|\mathcal{R}|} \sum_{\boldsymbol{X}_i \in \mathcal{R}} (Y_i - \widehat{\beta})^2, \quad \text{and} \quad \widehat{\beta} = \frac{1}{|\mathcal{R}|} \sum_{\boldsymbol{X}_i \in \mathcal{R}} Y_i$$

    and

$$\mathcal{R}_1(j,t) = \{\boldsymbol{X} \in \mathcal{R} : X_j \le t\}, \quad \mathcal{R}_2(j,t) = \{\boldsymbol{X} \in \mathcal{R} : X_j > t\}.$$

- The regression tree prediction is then

$$\widehat{y} = \widehat{f}(x) = \sum_{k=1}^{S} \widehat{\beta}_k \mathbb{1}(x \in \mathcal{R}_k), \quad \text{with} \quad \widehat{\beta}_k = \frac{1}{|\mathcal{R}_k|} \sum_{\boldsymbol{X}_i \in \mathcal{R}_k} Y_i$$

Remarks – Classification Trees

- Tree-based classifier enjoys great interpretability.
- Although CART is computational efficient, it is based on greedy approach with limited theoretical guarantees.
- Classification trees are notorious for their instability, i.e., a small change in the training set could greatly distort the tree.
    - This can be mitigated if we "combine" multiple trees to form a final classification decision. Such techniques are called emsemble learning.

Bagging

Bootstrap **agg**regat**ing** (Bagging) is a general technique for improving unstable predictive algorithms.

Bootstrap

Given a data set $Z_n = \{(\boldsymbol{X}_i, Y_i), i = 1, \ldots, n\}$, a bootstrap sample is drawn uniformly with replacement from Z_n.

Bagging starts with

- a training data $Z_n = \{(\boldsymbol{X}_i, Y_i), i = 1, \dots, n\}$; and
- a learning algorithm that maps Z_n to a prediction outcome $g(Z_n)$.

Bagging

1. Generate bootstrap samples $Z_n^{(*b)} = \{(\boldsymbol{X}_i^{(*b)}, Y_i^{(*b)}), i = 1, \dots, n\}, b = 1, \dots, B$.
2. Apply the learning algorithm to produce bootstrap predictions $g(Z_n^{(*b)})$.
3. Aggregate the bootstrap predictions using the majority vote:

$$\widehat{y}^{\text{bag}} = \arg\max_{c_k \in \mathcal{C}} \sum_{b=1}^{B} \mathbb{1}(g(Z_n^{(*b)}) = c_k).$$

- Bagging is a simple **meta algorithm** on top of a base algorithm.

The Bagging idea can be combined with virtually any statistical learning method.

- Regression models: use averaging.
  Example: Linear regression: $\widehat{f}(x) = \boldsymbol{x}^T \widehat{\boldsymbol{\beta}}$, then

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

- Classification methods: use majority vote, e.g., k-NN.

Random Forests

Bagging uses randomization, namely bootstrap, to generation a collection of classification trees.

Random Forest adds another layer of randomization on top of Bagging by using only a randomly selected subset of predictors to construct each tree.

- The correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated.
- The idea is to reduce the correlation between trees, which can further enhance the variance reduction.

Random forests

1. Generate bootstrap samples $Z_n^{(*b)} = \{(\boldsymbol{X}_i^{(*b)}, Y_i^{(*b)}), i = 1, \dots, n\}, b = 1, \dots, B$.
2. Apply a randomized version of CART for each bootstrap sample, to obtain a random forest $\{T_b^* : b = 1, \dots, B\}$.
   a. Before each splitting step, randomly select m variables.
   b. Find the best splitting using only the selected variables.
   c. Repeat until the tree is grown.
3. For each query $\boldsymbol{X} = \boldsymbol{x}$, let $T_b^*(\boldsymbol{x})$ be the prediction of the bth tree. The random forest prediction is

$$\widehat{y}^{\text{RF}} = \underset{c_k \in \mathcal{C}}{\arg\max} \sum_{b=1}^{B} \mathbb{1}(T_b^*(\boldsymbol{x}) = c_k).$$

- Compare with Bagging: RF uses a randomized tree growing algorithm (Step 2). This is sometimes called "feature bagging."

Boosting

The boosting problem asks whether we can build a learning algorithm of arbitrary accuracy (strong learner), using weak learning algorithms (weak learner) whose performance is only slightly better than random guessing.

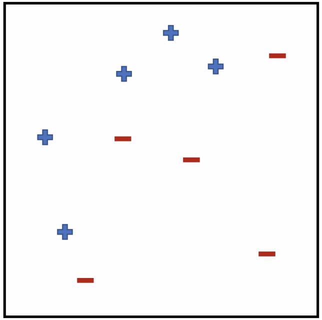
- The weak learners can be any classification method such as decision stumps, CART, RF, etc.

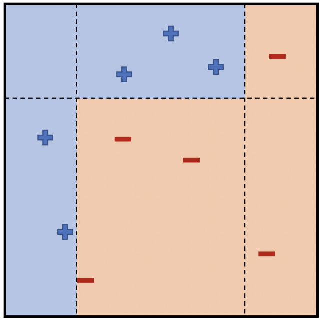
Adaptive Boosting

Adaptive **Boost**ing (AdaBoost) (Freund and Schapire, 1996)

- Build base learners sequentially using reweighted data;
  - inflate the weight of incorrectly classified observations
  - shrink the weight of correctly classified observations
  - **Idea**: encourage the future classifiers to focus on previously misclassified data.
- Assign credibility/weight to the base learners based on their error rate;
- Make decision based on weighted majority votes.

Example: Decision stumps and AdaBoost.





AdaBoost

1. Initialize the observation weights $w_i = 1/n, i = 1, \ldots n$. Code the class label $\{-1, 1\}$.
2. for $m = 1, \ldots, M$
   a. Fit a classifier $C_m(\boldsymbol{x})$ aiming to minimize the weighted misclassification error
   $$\sum_{i=1}^{n} w_i \mathbb{1}(Y_i \neq C_m(\boldsymbol{X}_i)).$$
   b. Compute the weighted misclassification error of the mth classifier
   $$\text{err}^{(m)} = \sum_{i=1}^{n} w_i \mathbb{1}(Y_i \neq C_m(\boldsymbol{X}_i)) / \sum_{i=1}^{n} w_i.$$
   c. Compute the weight of the mth classifier
   $$\alpha_m = \frac{1}{2} \log \frac{1 - \text{err}^{(m)}}{\text{err}^{(m)}}.$$
   d. Update weights
   $$w_i \leftarrow w_i \cdot \exp(\alpha_m \cdot \mathbb{1}(Y_i \neq C_m(\boldsymbol{X}_i)))$$
3. Output the final prediction as $\widehat{Y} = \text{sign}\left(\sum_{m=1}^{M} \alpha_m C_m(\boldsymbol{x})\right)$.

Derivation of AdaBoost

For a classification problem with $Y \in \{-1, 1\}$ and a generic rule sign$(f(\boldsymbol{x}))$, consider the expected exponential loss

$$L(f) = \mathbb{E}[\exp(-Yf(\boldsymbol{X}))] = \mathbb{E}_{\boldsymbol{X}}[\mathbb{P}(Y = 1 \mid \boldsymbol{X})e^{-f(\boldsymbol{X})} + \mathbb{P}(Y = -1 \mid \boldsymbol{X})e^{f(\boldsymbol{X})}].$$

It is easy to show that the minimizer of $L(f)$ is

$$f^*(\boldsymbol{x}) = \frac{1}{2}\log\left(\frac{\mathbb{P}(Y = 1 \mid \boldsymbol{X})}{\mathbb{P}(Y = -1 \mid \boldsymbol{X})}\right).$$

Moreover, the Bayes rule is $\delta^*(\boldsymbol{x}) = \text{sign}(f^*(\boldsymbol{x}))$.

We can approximates the Bayes rule by minimizing the empirical exponential loss

$$L_n(f) = \frac{1}{n}\sum_{i=1}^{n}\exp(-Y_i f(\boldsymbol{X}_i)) \approx L(f),$$

Derivation of AdaBoost

Minimizing $L(f)$ directly is as hard as obtaining the Bayes rule. Let's approximate!

1. Consider a restricted functional space

$$\left\{ f(\boldsymbol{x}) : f(\boldsymbol{x}) = \sum_{m=1}^{M} \beta_m C_m(\boldsymbol{x}) \right\}.$$

    - $\beta_m > 0$ and $C_m(\boldsymbol{x})$ is a classifier (the weak learner).

2. Use a iterative greedy search to approximate the minimizer.

- Set initial value $f^{(0)} = 0$.
- Given $f(\boldsymbol{x}) = f^{(m-1)}(\boldsymbol{x})$, update $f^{(m)}(\boldsymbol{x}) = f^{(m-1)}(\boldsymbol{x}) + \alpha^{(m)} C_m(\boldsymbol{x})$ by

$$
\begin{aligned}
(\alpha^{(m)}, C_m(\boldsymbol{x})) &= \underset{\alpha, C(\boldsymbol{x})}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \exp\left\{-Y_i [f^{(m-1)}(\boldsymbol{x}) + \alpha C(\boldsymbol{x})]\right\} \\
&= \underset{\alpha, C(\boldsymbol{x})}{\arg\min} \sum_{i=1}^{n} w_i^{(m)} \exp\{-Y_i \alpha C(\boldsymbol{x})\},
\end{aligned}
$$

  where $w_i^{(m)} = \frac{1}{n} \exp\left\{-Y_i f^{(m-1)}(\boldsymbol{x})\right\}$.
- (AdaBoost Step 2.a) Fix α, the minimizer $C(\boldsymbol{x})$ is

$$
\begin{aligned}
C_m(\boldsymbol{x}) &= \underset{C(\boldsymbol{x})}{\arg\min} \sum_{i=1}^{n} w_i^{(m)} e^{-\alpha} \mathbb{1}(Y_i = C(X_i)) + w_i^{(m)} e^{\alpha} \mathbb{1}(Y_i \neq C(X_i)) \\
&= \underset{C(\boldsymbol{x})}{\arg\min} \sum_{i=1}^{n} w_i^{(m)} \mathbb{1}(Y_i \neq C(X_i))
\end{aligned}
$$

- (AdaBoost Step 2.b and 2.c) Given C_m above, the minimizer is $\alpha^{(m)} = \alpha_m/2$ for α_m in AdaBoost.
- (AdaBoost Step 2.d) Plug $f^{(m)}(\boldsymbol{x}) = f^{(m-1)}(\boldsymbol{x}) + \frac{\alpha_m}{2} C_m(\boldsymbol{x})$ into $w_i^{(m+1)} = \frac{1}{n} \exp\left\{-Y_i f^{(m)}(\boldsymbol{x})\right\}$, we have $w_i^{(m+1)} = w_i^{(m)} \cdot \exp(\alpha_m \cdot \mathbb{1}(Y_i \neq C_m(\boldsymbol{X}_i)))$ after dropping a constant factor $e^{-\alpha_m/2}$ independent of i.

Performance Guarantee of AdaBoost

Training error can decay exponentially fast

Theorem (Freund and Schapire, 1997)

Let $1-\gamma_m$ be the misclassification error of C_m, then the misclassification error of AdaBoost is at most $\prod_{m=1}^{M}\sqrt{1-4\gamma_m^2} \leq \exp(-2\sum \gamma_m^2)$.

Remarks – Bagging, Random Forest and Boosting

- Bagging and random forests are perfect for parallel computing, because each base learner is completely decoupled with each other in terms of computation. This is not true for AdaBoost.
- Bagging, random forests and boosting are all ensemble learning methods.
- Bagging reduces variance significantly but may slightly increase the bias.
- The weak learners usually has low variance and high bias. In compared with the weak learners, boosting can significantly reduce bias but will increase the variance. So boosting usually works better for simple learners such as decision stumps, which are one-level decision trees.
- Multi-class AdaBoost, J. Zhu, H. Zou, S. Rosset, T. Hastie, 2009.

Hyperplane in $\mathbb{R}^p$

Let $\boldsymbol{x} \in \mathbb{R}^p$ (p-dimensional real space) with $\boldsymbol{x} = (x_1, x_2, \dots, x_p)$.

Hyperplane

Consider all $\boldsymbol{x}$ satisfying

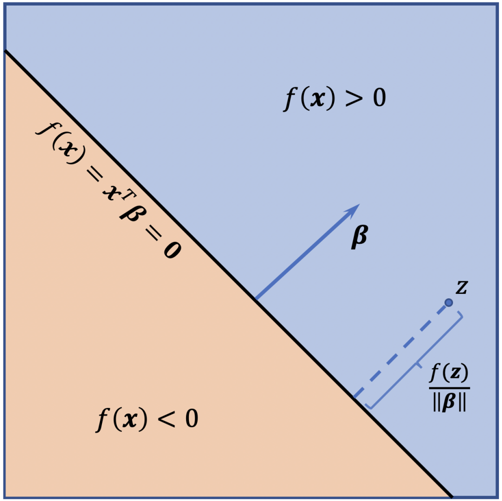
$$f(\boldsymbol{x}) = \beta_0 + x^T \boldsymbol{\beta} = \beta_0 + \beta_1 x_1 + \dots \beta_p x_p = 0.$$

All such $\boldsymbol{x}$ defines a <u>hyperplane</u>.

- $f(\boldsymbol{x}) > 0 \Leftrightarrow \boldsymbol{x}$ is on the side of the hyperplane pointed by $\boldsymbol{\beta}$.
- $f(\boldsymbol{x}) < 0 \Leftrightarrow \boldsymbol{x}$ is on the other side of the hyperplane.
- For any $\boldsymbol{z} \in \mathbb{R}^p$, the signed distance of $\boldsymbol{z}$ to the hyperplane is

$$(\langle \boldsymbol{z}, \boldsymbol{\beta} \rangle + \beta_0)/\|\boldsymbol{\beta}\|_2) = f(\boldsymbol{z})/\|\boldsymbol{\beta}\|_2.$$

  $\| \cdot \|_2$ is the Euclidean norm in $\mathbb{R}^p$.
- Hence, if $\|\boldsymbol{\beta}\|_2 = 1$, then $f(\boldsymbol{z})$ is the signed distance of $\boldsymbol{z}$ to the hyperplane defined by $f(\boldsymbol{x}) = 0$.


$f(\boldsymbol{x}) > 0$
$f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{\beta} = 0$
$\boldsymbol{\beta}$
z
$\frac{f(\boldsymbol{z})}{\|\boldsymbol{\beta}\|}$
$f(\boldsymbol{x}) < 0$

Support Vector Machine

Let's focus on the classification problem with $Y \in \{-1, 1\}$. Consider the simpler case where the training data is linearly separable.

Linear separability

The training data $\{(\boldsymbol{X}_i, Y_i) : i = 1, 2 \ldots, n\}$ is linearly separable, if there exists a hyperplane[a] such that

$$Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta}) \geq 0, \quad \forall i.$$

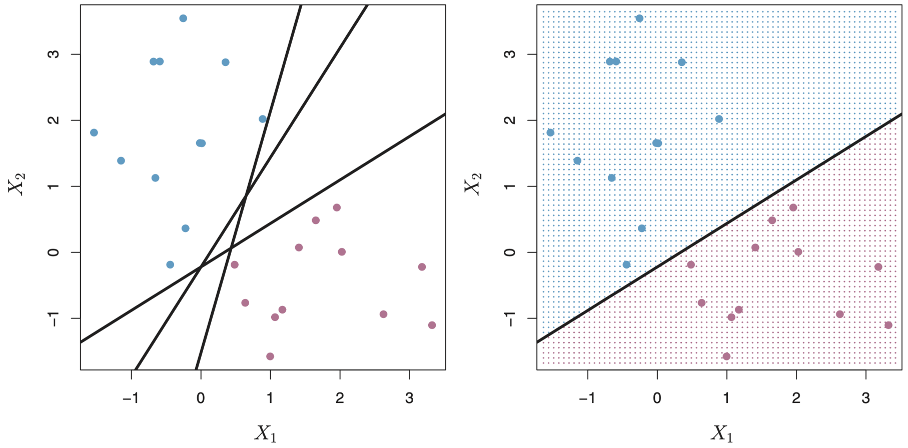
[a] A hyperplane is defined by $\{\boldsymbol{x} : \beta_0 + \boldsymbol{x}^T \boldsymbol{\beta} = 0\}$, where $\|\beta\|_2 = 1$ is a unit vector.

- For linearly separable data, the hyperplane gives a perfect classifier:

$$\widehat{y} = \text{sign}(\beta_0 + \boldsymbol{x}^T \boldsymbol{\beta}).$$

- This is called a separating hyperplane.

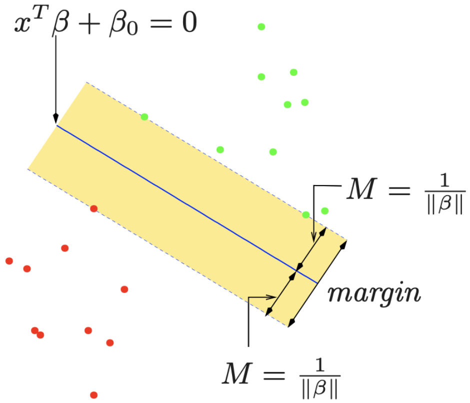
- The separating hyperplane is not unique, can we find a "best" one?



Margin

The margin is defined as the smallest distance from the training data to the hyperplane.

- Hyperplanes with larger margin are more robust.
- The Support Vector Machine (SVM) finds the hyperplane that maximizes the margin.
- The seperating hyperplane such that the minimum distance of any training point to the hyperplane is the largest.



Recall that the signed distance from a point $\boldsymbol{X}_i$ to a hyperplane specified by $(\beta_0, \boldsymbol{\beta})$ is $(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta}) / \|\beta\|_2$.

Consider only $\boldsymbol{\beta}$ such that $\|\beta\|_2 = 1$. Then $Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta})$ is the unsigned distance from the point $\boldsymbol{X}_i$ to the hyperplane.

Support vector machine

The SVM problem is formulated as

$$\begin{aligned} \max_{\beta_0, \boldsymbol{\beta}, \|\boldsymbol{\beta}\|_2 = 1} \quad & C, \\ \text{s.t.} \quad & Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta}) \geq C, \quad \forall i. \end{aligned}$$

For any feasible solution $\beta_0, \boldsymbol{\beta}$, perform a change of variable $\gamma_0 = \beta_0 / C, \boldsymbol{\gamma} = \boldsymbol{\beta}/C$, then $C\|\boldsymbol{\gamma}\|_2 = \|\boldsymbol{\beta}\|_2 = 1 \Rightarrow$ maximizing C is equivalent to minimizing $\|\boldsymbol{\gamma}\|_2$.

SVM is equivalent to (<u>with an abuse of notation $\gamma_0 = \beta_0, \boldsymbol{\gamma} = \boldsymbol{\beta}$</u>)

Alternative formulation of SVM

$$\min_{\beta_0, \boldsymbol{\beta}} \quad \frac{1}{2}\|\boldsymbol{\beta}\|_2^2,$$
$$\text{s.t.} \quad Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta}) \geq 1, \quad \forall i.$$

SVM – the Name

Apply Lagrangian multiplier for contrained optimization

$$\mathcal{L}(\alpha, \boldsymbol{\beta}) = \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 - \sum_{i=1}^{n} \alpha_i \left[Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta}) - 1 \right].$$

The Karush-Kuhn-Tucker (KKT) conditions for optimality is

$$\begin{aligned}
\text{(stationarity)} \quad & \boldsymbol{\beta} = \sum_{i=1}^{n} \alpha_i Y_i \boldsymbol{X}_i, \quad 0 = \sum_{i=1}^{n} \alpha_i Y_i, \\
\text{(primal/dual feasibility)} \quad & Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta}) \geq 1, \quad \alpha_i \geq 0, \quad \forall i, \\
\text{(complementary slackness)} \quad & \alpha_i \left[Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta}) - 1 \right] = 0, \quad \forall i.
\end{aligned}$$

Support vectors

The points $(Y_i, \boldsymbol{X}_i)$ with $\alpha_i > 0$ are called the support vectors, because they on the boundary of the margin, i.e.,

$$\alpha_i \left[Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta}) - 1\right] = 0 \Rightarrow \left[Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta}) - 1\right] = 0.$$

- Support vectors support the margin in the sense that a slight change in their location will result in a change of the margin.
- The optimal solution $\boldsymbol{\beta}$ depends only on support vectors. Robustness!

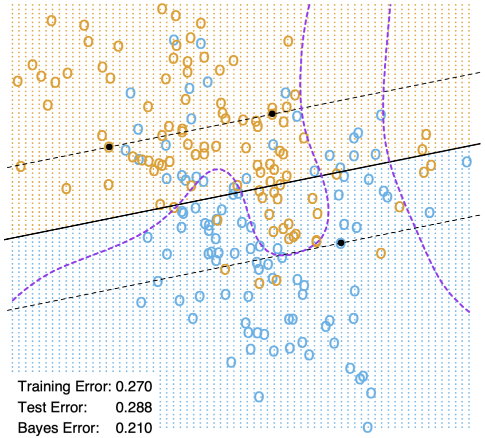
$$\boldsymbol{\beta} = \sum_{i=1}^{n} \alpha_i Y_i \boldsymbol{X}_i.$$

- The SVM classifier is then given by the sign of a linear function

$$\widehat{y} = \text{sign}(\widehat{\beta}_0 + \boldsymbol{x}^T \widehat{\boldsymbol{\beta}}) = \text{sign}\left(\widehat{\beta}_0 + \sum_{i=1}^{n} \widehat{\alpha}_i Y_i \boldsymbol{x}^T \boldsymbol{X}_i\right).$$

There are two potential issues with the barebone SVM:

- Data may not be linearly separable $\Rightarrow$ SVM with soft margin.
- Linear boundaries are not flexible enough $\Rightarrow$ Kernel SVM.



SVM for the Non-Separable Case

When data are not linearly separable, we allow some training data to be on the wrong side of the hyperplane. In general, we have SVM with soft margin:

$$\begin{aligned} \max_{\beta_0, \boldsymbol{\beta}, \|\beta\|_2=1} \quad & C, \\ \text{s.t.} \quad & Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta}) \geq C(1 - \xi_i), \quad \forall i, \\ & \xi_i \geq 0, \sum_i \xi_i \leq B. \end{aligned}$$

- ξ_i are the slack variables.
- B is a tuning parameter.

The SVM classifier is then given by the sign of a linear function

$$\widehat{y} = \text{sign}(\widehat{\beta}_0 + \boldsymbol{x}^T \widehat{\boldsymbol{\beta}}).$$

Remarks

ξ_i indicates the location of the observation with respect to the margin/hyperplane.

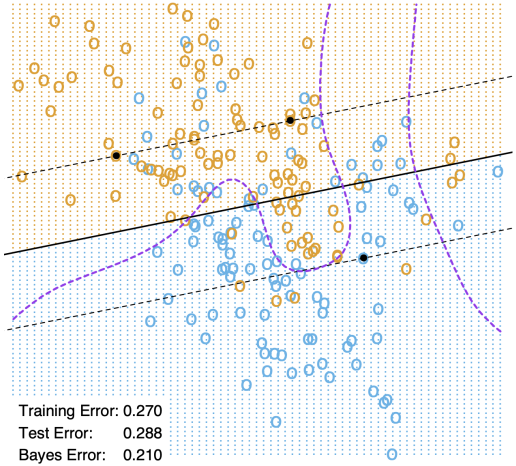
- $\xi_i = 0$ on the correct side of the margin;
- $\xi_i > 0$ on the wrong side of the margin;
- $\xi_i > 1$ on the wrong side of the hyperplane.

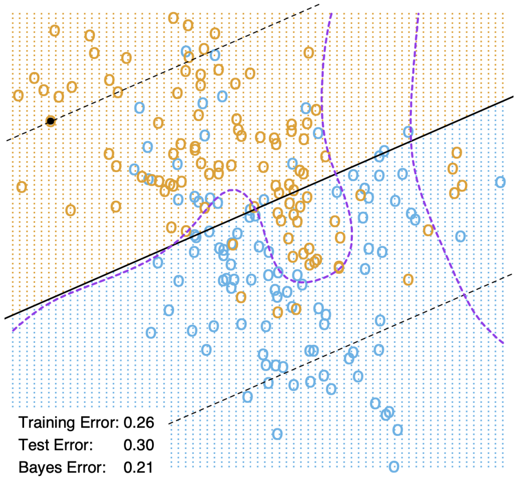
B is the "budget" for voilations of the margin.

- $B = 0$ then no violation allowed. A classifier exist only if the data is separable.
- The larger B is, the more violation allowed.
- No more than B observations can be on the wrong side of the hyperplane.
- The half-width of the margin C in creases as B increases.

The support vectors

- Similar to the separable case, we can analyze the KKT conditions and find the <u>support vectors</u>.
- All vectors that are on the wrong side of the margin (including those who are exactly on the margin) are now support vectors.
- Only the support vectors affect the support vector classifier.
- Larger $B \Rightarrow$ larger margin $\Rightarrow$ more support vectors $\Rightarrow$ larger bias, lower variance, hence more robust.


Training Error: 0.270
Test Error: 0.288
Bayes Error: 0.210

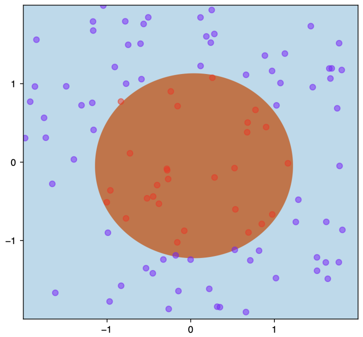

Training Error: 0.26
Test Error: 0.30
Bayes Error: 0.21

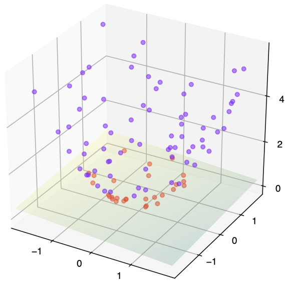
Nonlinear Boundary

In practice, we may have nonlinear boundaries. Just as in the polinomial regression, we can <u>enlarge the feature space</u> from the original inputs to the polynomials of them.

We can directly apply SVM on the enlarged the feature space.

Example: Quadradic boundary. $(X_1, X_2) \rightarrow (X_1, X_2, X_1^2, X_2^2)$.





A separating hyperplane in the enlarged feature space $(X_1, X_2, X_1^2, X_2^2, X_1X_2)$ is expressed by the equation:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 = 0.$$

However, the caveat is that the dimension of the enlarged feature space grows really fast.

- Suppose we have $\boldsymbol{X} \in \mathbb{R}^m$ and the degree of polynomial features are at most d.
- Then the dimension of the enlarged feature space is

$$\binom{d+m-1}{d} = \frac{(d+m-1)!}{d!(m-1)!}.$$

- For $m = 100$ and $d = 6$, this is about 1.6 billion.

The "kernel trick" can help!

SVM with Nonlinear Boundary

Recall the Karush-Kuhn-Tucker (KKT) conditions in the separable case

$$\boldsymbol{\beta} = \sum_{i=1}^{n} \alpha_i Y_i \boldsymbol{X}_i, \quad 0 = \sum_{i=1}^{n} \alpha_i Y_i,$$

$$Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta}) \geq 1, \quad \alpha_i \geq 0, \quad \forall i,$$

$$\alpha_i \left[Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta}) - 1 \right] = 0, \quad \forall i.$$

Key observation 1

- Plugging $\boldsymbol{\beta} = \sum_{i=1}^{n} \alpha_i Y_i \boldsymbol{X}_i$ into complementary slackness

$$\alpha_i \left[Y_i(\beta_0 + \sum_{j=1}^{n} \alpha_j Y_j \boldsymbol{X}_i^T \boldsymbol{X}_j) - 1 \right] = 0.$$

- The calculation of SVM depends on $\boldsymbol{X}$ only through the $\underline{\text{inner products}}$ $\boldsymbol{X}_i^T \boldsymbol{X}_j$.

Key observation 2

- The SVM classifier depends on $\boldsymbol{X}$ only through the inner product of the new observation and the features $\boldsymbol{x}^T \boldsymbol{X}_j$.

$$\widehat{y} = \text{sign}(\widehat{\beta}_0 + \boldsymbol{x}^T \widehat{\boldsymbol{\beta}}) = \text{sign}\left(\widehat{\beta}_0 + \sum_{i=1}^{n} \widehat{\alpha}_i Y_i \boldsymbol{x}^T \boldsymbol{X}_i\right).$$

Similar observations holds for SVM with soft margin (the nonseparable case).

- For complete details using dual SVM formulation, read Section 12.2.1 of ESL.
- SVM is usually calculated from its dual formulation, which is a convex optimization problem.

Only the inner product in the feature space is relevant in computing the linear support vector classfier.

This grants us an alternative way to think about enlarging the feature space. If we have the inner product,

- We are not required to explicitly write the enlarged features space, which can be very large in some applications.
- The computation complexity may be greatly reduced. Because we now only need to calculate $\binom{n}{2}$ distinct pairs of inner products.
- It becomes much easiler to deal with implicit and infinite dimensional feature space.

The Kernel Trick

- The inner product is a bivariate function $\langle \cdot, \cdot \rangle : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is the feature space.
- It can be generalized to kernel functions $K(x, z)$.
- The kernelized SVM classifier is then

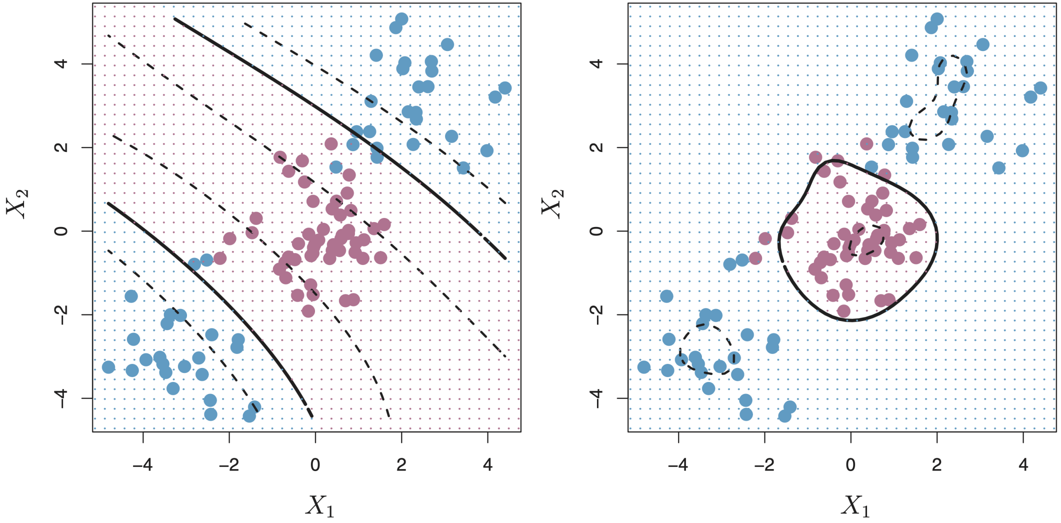
$$\widehat{y} = \text{sign}\left(\widehat{\beta}_0 + \sum_{i=1}^{n} \alpha_i Y_i K(\boldsymbol{x}, \boldsymbol{X}_i) \right).$$

- The enlarged feature space is a space of kernel functions $\boldsymbol{x}_i \to K(\boldsymbol{x}, \boldsymbol{x}_i)$, which can have infinite dimension.

Example: Commonly used kernels

- Linear kernel:
$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle = \boldsymbol{x}_i^T \boldsymbol{x}_j.$$
- Polynomial kernel of degree up to d:
$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1 + \boldsymbol{x}_i^T \boldsymbol{x}_j)^d.$$
  - This is equivalent to enlarging the feature space to include all polinomials with degree up to d.
- Polynomial kernel of degree exactly d:
$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^T \boldsymbol{x}_j)^d.$$
- Gaussian (radial basis function) kernel:
$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2).$$
  - **Local behavior**: The kernel decreases exponentially fast in the distance of two feature vectors. Points faraway play have little effect in classification.
  - The corresponding feasture space is implicit and infinite-dimensional.

Example: Polynomial kernel of degree 3 (left) versus Gaussian radial kernel (right).



"Loss + Penalty" Formula

This "loss+penalty" formula is commonly seen in statistical learning models

$$\min_f \left\{ \sum_i L(Y_i, f(\boldsymbol{X}_i)) + \lambda P(f) \right\}.$$

- Ridge regression: square error and l_2 penalty

$$\sum_i (Y_i - \boldsymbol{X}_i^T \boldsymbol{\beta}))^2 + \lambda \|\boldsymbol{\beta}\|_2^2$$

- Lasso regression: square error loss and l_1 penalty

$$\sum_i (Y_i - \boldsymbol{X}_i^T \boldsymbol{\beta}))^2 + \lambda \|\boldsymbol{\beta}\|_1$$

We now show that SVM can also be written in this form!

SVM in "Loss + Penalty" Form

$$\begin{aligned} \max_{\beta_0, \boldsymbol{\beta}, \|\beta\|_2=1} \quad & C, \\ \text{s.t.} \quad & Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta}) \geq C(1-\xi_i), \quad \forall i, \\ & \xi_i \geq 0, \sum_i \xi_i \leq B. \end{aligned}$$

Let $\gamma_0 = \gamma_0/C, \boldsymbol{\gamma} = \boldsymbol{\beta}/C$, then $C\|\boldsymbol{\gamma}\|_2 = 1$. The constraints are then

$$\xi_i \geq 1 - Y_i(\gamma_0 + \boldsymbol{X}_i^T \boldsymbol{\gamma}).$$

Combine this with the non-negative constraints of ξ_i, we have

$$\xi_i \geq \left[1 - Y_i(\gamma_0 + \boldsymbol{X}_i^T \boldsymbol{\gamma})\right]_+ .$$

Here $x_+ = \max\{0, x\}$ is the positive part of x.

- The optimal choice of ξ_i is

$$\xi_i = \left[1 - Y_i(\gamma_0 + \boldsymbol{X}_i^T \boldsymbol{\gamma})\right]_+ .$$

Plugging into the upper bound for the sum of ξ_i's, we have

$$\sum_i \left[1 - Y_i(\gamma_0 + \boldsymbol{X}_i^T \boldsymbol{\gamma})\right]_+ \le B.$$

Thus, we have the equivalent problem

$$\begin{aligned} \min_{\gamma_0, \boldsymbol{\gamma}} \quad & \frac{1}{2}\|\boldsymbol{\gamma}\|_2^2, \\ \text{s.t.} \sum_i & \left[1 - Y_i(\gamma_0 + \boldsymbol{X}_i^T \boldsymbol{\gamma})\right]_+ \le B. \end{aligned}$$

Apply Lagrangian Multiplier method,

$$\min_{\beta_0,\boldsymbol{\beta}} \frac{1}{n} \sum_i \left[1 - Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta})\right]_+ + \lambda \|\boldsymbol{\beta}\|_2^2.$$

- $(1-t)_+$ is called the hinge loss.
- $\frac{1}{n}\sum_i \left[1 - Y_i(\beta_0 + \boldsymbol{X}_i^T \boldsymbol{\beta})\right]_+$ is the empirical hinge loss.
- $\lambda\|\boldsymbol{\beta}\|_2^2$ is the l_2 penalty.
- There is a one-to-one correspondence between B and λ.

Logistic regression: If $Y \in \{-1, 1\}$, the logistic loss (also called binomial deviance)

$$\sum_i \log(1 + e^{-Y_i \boldsymbol{X}_i^T \boldsymbol{\beta}})$$

- Regularized logistic regression:

$$\sum_i \log(1 + e^{-Y_i \boldsymbol{X}_i^T \boldsymbol{\beta}}) + \lambda \|\boldsymbol{\beta}\|_2^2, \quad \sum_i \log(1 + e^{-Y_i \boldsymbol{X}_i^T \boldsymbol{\beta}}) + \lambda \|\boldsymbol{\beta}\|_1$$

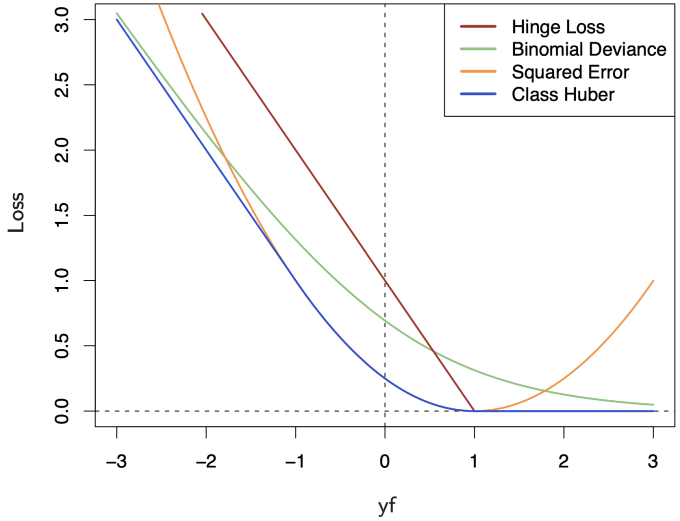
AdaBoost: exponential loss

$$\frac{1}{n} \sum_{i=1}^n \exp(-Y_i f(\boldsymbol{X}_i))$$

- Regularized AdaBoost:

$$\frac{1}{n} \sum_{i=1}^n \exp(-Y_i f(\boldsymbol{X}_i)) + \lambda \|\boldsymbol{\alpha}\|_1$$

Comparing the Loss Functions



Sparse Support Vector Machine

Sparsity is important under high dimensional data.

- **Noise acumulation**: unimportant features will compromise classification accuracy, because the stochastic error in the estimation of the parameters can accumulate.
- **Interpretability**: classifiers with too many parameters are hard to interpret.

We have seen that the Lasso (l_1) penalty can encourage sparsity. Starting from the loss + penalty formula for SVM, we replace the l_2 penalty by the l_1 penalty

l_1 SVM

$$\min_{\beta_0,\boldsymbol{\beta}} \frac{1}{n}\sum_i \left[1 - Y_i(\beta_0 + \boldsymbol{X}_i^T\boldsymbol{\beta})\right]_+ + \lambda\|\boldsymbol{\beta}\|_1.$$

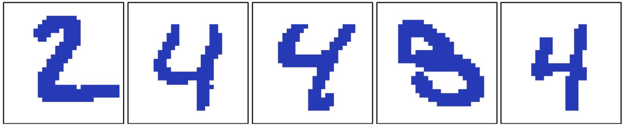
One-Versus-One Approach

- For $K > 2$ classes.
- Run a SVM on each of $\binom{K}{2}$ pairs of classes.
- For each new observation, count the number of times the SVMs classify it to class k.
- Classify it to the majority vote.

One-Versus-All Approach

- For $K > 2$ classes.
- Run a SVM on class k (re-labeled as $+1$) versus all others (re-label ed as -1): compute
$$f^k(\boldsymbol{x}) = \widehat{\beta}_0^k + \boldsymbol{x}^T \widehat{\boldsymbol{\beta}}^k.$$
- For each new observation, classify it to the k with the largest $f^k(\boldsymbol{x})$. (Note that the larger $f^k(\boldsymbol{x})$, the more likely $\boldsymbol{x}$ belongs to k.)

Example: Hand-writing recognition.



Reference

- The Element of Statistical Learning. Section 4.3-4.5 , 9.2, 10.1-10.6, 12.2-12.3, 13.3.