Support Vector Classifier

Suppose now that the classes overlap in the feature space. We still classify based on the sign of $f(X) = \beta_0 + X^T\beta$ and still maximize the margin M of the hyperplane $f(X) = 0$, but allow some points to be on the wrong side of the margin as well as the hyperplane. Let $\epsilon_1, \ldots, \epsilon_n$ denote **slack variables**. We modify the constraint $Y_i f(X_i) \geq M$ for all $i = 1, \ldots, n$ as

$$Y_i f(X_i) \geq M(1 - \epsilon_i), \quad \epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C.$$

- "boundary" and "margin" are used interchangeably
- $\epsilon_i = 0$: correct side of the margin
- $\epsilon_i > 0$: wrong side of the margin (violates the margin)
- $\epsilon_i > 1$: wrong side of the hyperplane (misclassification)
- Think of C as a **budget** for margin violation. If $C = 0$, all $\epsilon_i = 0$. If $C > 0$, # misclassifications $\leq C$.
- Large C = more tolerant of violations.

The **support vector classifier** is obtained by solving the same optimization problem as in the maximal margin classifier but with the new constraints. As before, upon taking $M = 1/||\beta||$, we get the following equivalent problem:

$$\min_{\beta_0,\beta} \frac{1}{2}||\beta||^2$$

subject to

$$Y_i f(X_i) \geq 1 - \epsilon_i,\ \epsilon_i \geq 0,\ i = 1, \ldots, n,\ \sum_{i=1}^{n} \epsilon_i \leq C.$$

- **Support vectors**: The points that lie either on the margin or violate the margin. Only these affect the classifier.
- The observations that lie on the correct side of the margin do not affect the classifier.
- Slab (margin) boundaries: $Yf(X) = 1$; margin $= 1/||\beta||$.

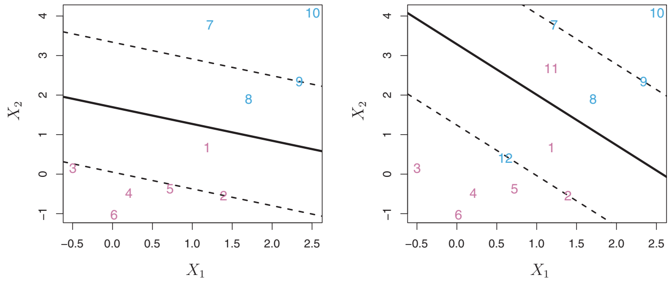

FIGURE 9.6. Left: *A support vector classifier was fit to a small data set. The hyperplane is shown as a solid line and the margins are shown as dashed lines.* Purple observations: *Observations* $3, 4, 5,$ *and* 6 *are on the correct side of the margin, observation* 2 *is on the margin, and observation 1 is on the wrong side of the margin.* Blue observations: *Observations* 7 *and* 10 *are on the correct side of the margin, observation* 9 *is on the margin, and observation* 8 *is on the wrong side of the margin. No observations are on the wrong side of the hyperplane.* Right: *Same as left panel with two additional points,* 11 *and* 12*. These two observations are on the wrong side of the hyperplane and the wrong side of the margin.*

Source: ISL

How to Choose C?

C is a *tuning* parameter that controls the bias-variance tradeoff.

- Larger C = more tolerant of margin violations = wider margin = more support vectors = potentially lower variance (because the classifier is determined only by the support vectors which are larger in number) but higher bias (because the training error is larger due to the larger number of margin violations)
- C can be chosen in the usual manner using a cross-validation or a validation-set approach.

Note: Support vector classifier has a linear decision boundary because of which it may not work well when a non-linear decision boundary is called for.

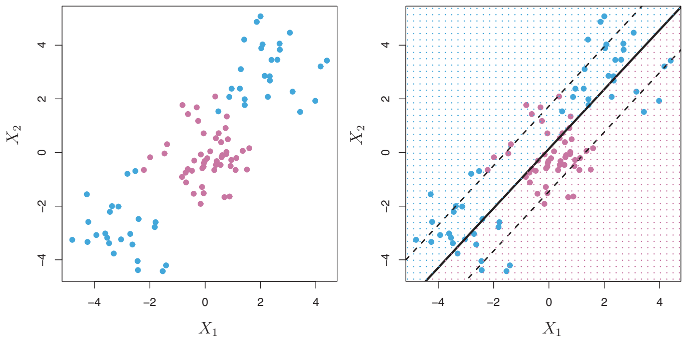

FIGURE 9.8. Left: *The observations fall into two classes, with a non-linear boundary between them.* Right: *The support vector classifier seeks a linear boundary, and consequently performs very poorly.*

Source: ISL

Classification with Non-Linear Boundaries

Option 1: Fit a support vector classifier but instead of using just the original features $X_1, \ldots, X_p$, we enlarge the feature space using functions of predictors, e.g., polynomials in X_j and interactions of the form $X_j * X_l$, $j \neq l$, and use all the features in the enlarged space as predictors.

- Decision boundary is linear in the enlarged space but non-linear in the original space
- There are many ways to enlarge the feature space. Unless done carefully, we may end up with a prohibitively large number of features, making the computations problematic.

Option 2: Use a **support vector machine** which also involves fitting a support vector classifier in an enlarged space but done so in a computationally efficient manner.

Support Vector Machine (SVM)

Inner product of two feature vectors X_i and X_l is:

$$\langle X_i, X_l \rangle = X_i^T X_l = \sum_{j=1}^{p} X_{ij} X_{lj}.$$

Note: It can be seen that in an SV classifier, the X_i play a role in the optimization problem and the solution $\hat{f}(X) = \hat{\beta}_0 + X^T \hat{\beta}$ only through the inner products involving them. In particular, we have $\hat{\beta} = \sum_{i=1}^{n} \hat{\alpha}_i Y_i X_i$, where the coefficient $\hat{\alpha}_i$ is positive for a support vector and is zero otherwise. Thus, we can write

$$\hat{f}(X) = \hat{\beta}_0 + X^T \left(\sum_{i=1}^{n} \hat{\alpha}_i Y_i X_i \right) = \beta_0 + \sum_{i=1}^{n} \hat{\alpha}_i Y_i \langle X, X_i \rangle.$$

To generalize, we can replace the inner product $\langle X_i, X_l \rangle$ by a *kernel function* $K(X_i, X_l)$, leading to the estimated function as

$$\hat{f}(X) = \hat{\beta}_0 + \sum_{i=1}^{n} \hat{\alpha}_i Y_i K(X, X_i).$$

This generalization is called an **SVM**. It fits an SV classifier in the transformed feature space. By definition, K is a symmetric, positive (semi)-definite function. We can think of the kernel K as a *measure of dissimilarity*. Two popular choices for K in the SVM literature are:

d**th degree polynomial**: $K(X, X') = (1 + \langle X, X' \rangle)^d$

Radial basis: $K(X, X') = \exp(-\gamma ||X - X'||^2)$, $\gamma > 0$.

Taking $K(X, X') = \langle X, X' \rangle$ (or equivalently $d = 1$ in the polynomial kernel) gives the SV classifier. This is a *linear* kernel whereas the other kernels are non-linear.

SVMs with $Q\,(> 2)$ Classes

So far our focus was on **binary classification**. The concept of separating hyperplanes does not extend naturally to more than two classes. Two common approaches to deal with this:

One-versus-one classification: There are $\binom{Q}{2}$ pairs of classes. Fit an SVM for each pair. Then, classify a test point X as follows: Obtain its $\binom{Q}{2}$ classifications, one from each fit, and assign it to the most frequent class.

One-versus-all classification: Fit Q SVMs, each time comparing one class (coded as $+1$) with the remaining $Q - 1$ classes (coded as -1). For a test point X, compute $\hat{f}(X)$ from each fit. Assign it to the class for which $\hat{f}(X)$ is largest.

SV Classifier vs Logistic Regression

SV classifier: $\text{sign}\{f(X)\}$, where $f(X) = \beta_0 + X^T\beta$. The margin (width) is $1/||\beta||$ and its boundaries are $Yf(X) = 1$. An observation does not violate the margin if $Yf(X) \geq 1$. The optimization problem for fitting it can be rewritten as

$$\min_{\beta_0,\beta} \left\{ \sum_{i=1}^{n} \max[0, 1 - Y_i f(X_i)] + \lambda ||\beta||^2 \right\},$$

where $\lambda\,(> 0)$ is a tuning parameter. This is the familiar "loss (or error) + penalty" formulation that leads to regularized estimation. The penalty here is same as the ridge penalty.

Loss function, $\max[0, 1 - Yf(X)]$: It equals zero if $Yf(X) \geq 1$,i.e., the observation does not violate the margin, and equals $1 - Yf(X)$ if $Yf(X) < 1$, i.e., the observation violates the margin — **hinge loss function**. It tends to be similar to the loss function $(-\log L)$ used in logistic regression.

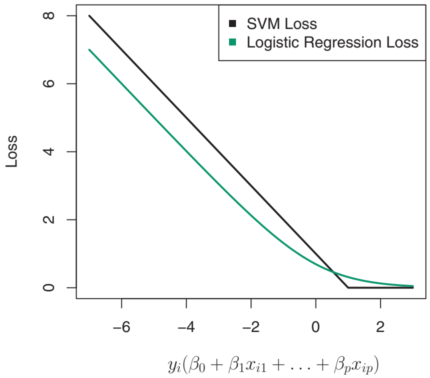

FIGURE 9.12. *The SVM and logistic regression loss functions are compared, as a function of $y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})$. When $y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})$ is greater than 1, then the SVM loss is zero, since this corresponds to an observation that is on the correct side of the margin. Overall, the two loss functions have quite similar behavior.*

Source: ISL

Takeaways

- Maximal margin hyperplane — classes are linearly separable
- SV classifier — linear decision boundary (works like regularized logistic regression)
- SVMs with non-linear kernels lead to non-linear decision boundaries

Boosting Decision Trees

Bagging: Create multiple copies of the original dataset using bootstrap, fit a separate tree to each dataset, and combine to create a single predictive model.

Boosting: Grow the trees *sequentially* — grow each tree using information from previously grown trees. Each tree in the sequence is “weak” by itself but together they produce a “powerful committee” — a general idea that works in other contexts as well.

- Does not involve bootstrap sampling
- Learns slowly

We will consider boosting only in the context of regression trees. Similar ideas apply for classification trees (see Chapter 10 of *Elements of Statistical Learning, 2nd edition*, for details).

Algorithm 8.2 *Boosting for Regression Trees*

```
1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all $i$ in the training set.
2. For $b = 1, 2, \dots, B$, repeat:
    (a) Fit a tree $\hat{f}^b$ with $d$ splits ($d+1$ terminal nodes) to the training data $(X, r)$.
    (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
            $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$    (8.10)
    (c) Update the residuals,
            $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$    (8.11)
3. Output the boosted model,
            $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$    (8.12)
```

Source: ISL

- Given the current model, we fit a tree to the *residuals* from the model rather than the outcome Y. Then, add the new tree into the fitted function to update the residuals.
- Each of these trees can be rather small — controlled by d.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well.
- The shrinkage parameter λ slows down this process further, allowing more and different shaped trees.
- A *slow learning approach* generally performs well.
- See ESLII for the objective function being minimized using a *gradient boosting algorithm.* Allows loss functions other than the squared error and misclassification error, and performs *subsampling* — at each iteration we sample a fraction η of the training observations (without replacement) and grow the next tree using that subsample. In practice, $\eta = 0.5$ is typical.

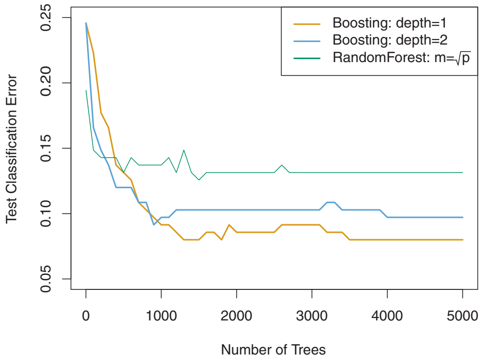

FIGURE 8.11. *Results from performing boosting and random forests on the 15-class gene expression data set in order to predict* cancer *versus* normal. *The test error is displayed as a function of the number of trees. For the two boosted models,* $\lambda = 0.01$. *Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant. The test error rate for a single tree is 24 %.*

Source: ISL

Three Tuning Parameters in Boosting

# trees B: **Boosting can overfit if B is too large** — unlike bagging and random forest. We can use cross-validation to select B.

Shrinkage parameter λ: A small positive # that controls the rate at which boosting learns. A very small value can require a very large B to achieve good performance. Typical values are 0.01 or 0.001, but the right choice depends on the problem.

# of splits d (or terminal nodes $d + 1$): It controls the complexity of the trees. Often, $d = 1$ works well — each tree is a *stump*, consisting of a single split. More generally, d is called the *interaction depth* as it controls the interaction order of the boosted model, since d splits can involve at most d variables. In practice, $d \approx 5$ is often good enough.

What Else?

- Regularization
- Kernels
- Deep learning
- $\cdots$

Note: *Proof is in the pudding* — apply all methods you know and pick the one that has the smallest test error.