Support Vector Machine

Arnab Maity

NCSU Statistics ~ 5240 SAS Hall ~ amaity[at]ncsu.edu

Contents

Introduction

The support vector machine (SVM) is a family of classification rules that contain both parametric (e.g., linear) and nonparametric (e.g., kernel based) methods. It is often considered as one of the ready-to-use classifiers. It can be viewed as a generalization of linear decision boundaries for classification. In particular, SVM produces nonlinear boundaries by constructing a linear boundary in a large, transformed version of the feature space. We will mainly discuss three approaches: maximal margin classifier, the support vector classifier, and the support vector machine.[1]

[1] People often loosely refer to these methods collectively as *support vector machines*.

Maximal margin classifier

We know that linear discriminant analysis and logistic regression both estimate linear decision boundaries (using different approaches) in classification problems. The separating hyperplane classifiers construct linear decision boundaries that *explicitly try to separate* the data into different classes as well as possible. They provide the basis for support vector classifiers.

Let us take a look at the following artificially generated dataset. The dataset contains two predictors, X_1 and X_2, and observations come from two classes (blue circles and orange triangles); see Figure 1. The two classes are well separated, and a straight line can be used for classification. This situation is called *linearly separable*. Formally, in a linearly separable case, there exists $\beta_0, \beta_1, \beta_2$ such that the line

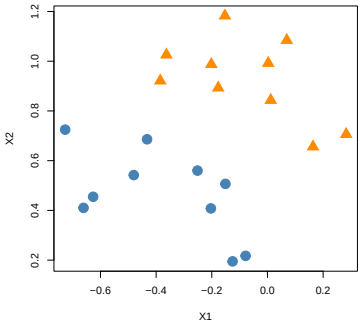


Figure 1: Simulated two class data with support vectors and separating hyperplane (line) highlighted.

$$\beta_0 + X_1\beta_1 + X_2\beta_2 = 0$$

perfectly separate the two classes. If we code the response as $Y = -1$ and $Y = 1$ for the two classes, respectively, then for a data point with predictor values (X_{i1}, X_{i2}), we have the relation

$$Y_i = -1 \text{ if } \beta_0 + X_{i1}\beta_1 + X_{i2}\beta_2 < 0; \;\; Y_i = +1 \text{ otherwise.}$$

In other words,

$$Y_i = sign(\beta_0 + X_{i1}\beta_1 + X_{i2}\beta_2).$$

In general, with p predictors, $X_{i1}, \ldots, X_{ip}$, we will use a linear combination of the form $\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p$. This is called a *hyperplane*, and thus the name *separating hyperplane*. One crucial property of a separating hyperplane is that,

$$Y_i(\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p) > 0, \; i = 1, \ldots, n.$$

When such a separating hyperplane exists, we can construct a natural classifier: for a new observation $\boldsymbol{x} = (x_1, \ldots, x_p)$, the predicted

class is

$$\widehat{Y} = sign(\beta_0 + x_1\beta_1 + \ldots + x_p\beta_p).$$

In other words, a test observation is assigned a class depending on which side of the hyperplane it is located.

One possible was to find such a separating hyperplane is to minimize the distance of misclassified points to the decision boundary. If a response $Y_i = 1$ is misclassified, then $\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p < 0$, and the opposite for a misclassified response with $Y_i = -1$. Also note that the magnitude of $\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p$ tells us how far the observation $(X_{i1}, \ldots, X_{ip})$ is located from the boundary. Thus we can minimize

$$-\sum_{i\in\mathcal{M}} Y_i(\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p)$$

with respect to $\beta_0, \ldots, \beta_p$, where $\mathcal{M}$ indexes the set of misclassified points. It can be shown that the quantity above is nonnegative and proportional to the distance of the misclassified points to the decision boundary defined by $\beta_0 + X_1\beta_1 + \ldots + X_p\beta_p = 0$.

However, when the problem is linearly separable, there are infinitely many such separating hyperplanes, see Figure 2. In this perfectly separable case, given any separating hyperplane, we can define the *margin* as the minimal distance from the observations to the hyperplane.[2] The optimal classification rule is the line that *maximizes the margin* around the separating line. Such a classifier is called *the maximal margin classifier*. Formally, we consider the optimization problem:

$$\max M \quad \text{subject to} \begin{cases} Y_i(\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p) \geq M, \ i = 1, \ldots, n, \\ \beta_1^2 + \ldots + \beta_p^2 = 1, \end{cases}$$

with respect to $\beta_0, \ldots, \beta_p$. The two conditions ensure that, when $M > 0$, each observation will remain on the correct side of the boundary and is at least distance M from the boundary. We seek the largest such M and associated parameters.

It can be shown that the optimization problem above is equivalent to the following optimization problem:

$$\min (\beta_1^2 + \ldots + \beta_p^2) \quad \text{subject to } Y_i(\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p) \geq 1, \ i = 1, \ldots, n,$$

with respect to $\beta_0, \ldots, \beta_p$. To understand the geometry behind the optimization problem above, we note that for perfectly separable data, $Y_i(\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_{ip}) > 0$ for each i. The constraint $Y_i(\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p) \geq 1$ implies $\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p \geq 1$ if $Y_i = 1$, and that $\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p \leq -1$ if $Y_i = -1$. Thus the constraints above define an empty slab or margin around the linear decision boundary of thickness $1/(\beta_1^2 + \ldots + \beta_p^2)^{1/2}$. Hence

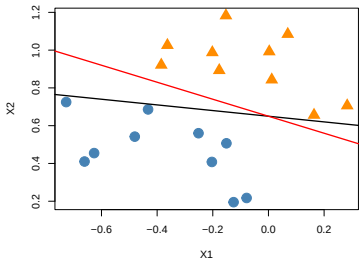


Figure 2: Examples of multiple separating hyperplanes for the same data.

[2] Formally, for a given separating hyperplane, define M_i = distance between the hyperplane and i-th training point. The `margin` is defined as $M = min(M_1, \ldots, M_n)$. The optimal classification rule (maximal separating hyperplane) maximizes the margin.

we choose $\beta_0, \beta_1, \ldots, \beta_p$ to maximize its thickness, or equivalently minimize $(\beta_1^2 + \ldots + \beta_p^2)$. The boundaries of this empty slab/margin are determined by the points that exactly satisfies the condition

$$\beta_0 + X_1\beta_1 + \ldots + X_p\beta_p = \pm 1.$$

Once we obtain the estimators $\widehat{\beta}_0, \widehat{\beta}_1, \ldots, \widehat{\beta}_p$, the optimal separating hyperplane is $\widehat{f}(\boldsymbol{x}) = \widehat{\beta}_0 + x_1\widehat{\beta}_1 + \ldots + x_p\widehat{\beta}_p$. Thus our classification rule is as follows: for a new observation $\boldsymbol{x} = (x_1, \ldots, x_p)$,

$$\widehat{Y} = sign\{\widehat{f}(\boldsymbol{x})\}.$$

In Figure 3, the optimal separating line is shown as the solid red line, the closest points to the line are circled, and the separation between the classes is shown using the dashed black lines. Essentially, the separating line corresponding to the maximal margin classifier represents the middle line of the widest space that we can fit between the two classes. Although none of the training observations fall in the margin (by construction), this will not necessarily be the case for test observations. The intuition is that a large margin on the training data will lead to good separation on the test data.

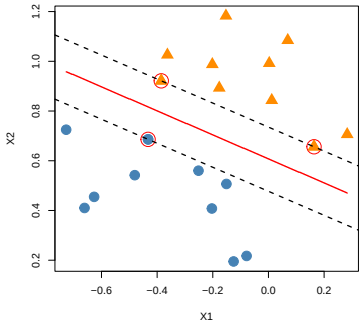


Figure 3: Simulated two class data with support vectors and separating hyperplane (line) highlighted.

Notice that there are a few points (circled) that are closest, and equidistant, to the red separating line. These three points lie along the dashed lines indicating the width of the margin, that is, these points satisfy the condition

$$Y_i(\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p) = 1,$$

exactly. These points are called the *support vectors* for this problem. It can be shown that *only the support vectors are enough to define the optimal classification rule fully*. If we move the support vectors, the optimal separating line also changes. A movement to any of the other observations would not affect the separating hyperplane, provided that the observation's movement does not cause it to cross the boundary set by the margin. This property of the maximal margin classifier is important in development of support vector classifier and support vector machine.

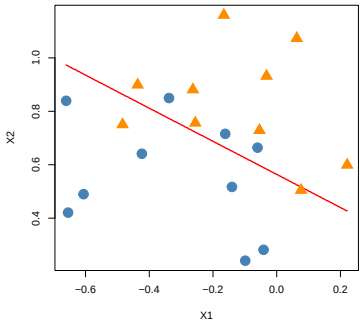


Figure 4: Simulated two-class data with separating hyperplane (line) highlighted for the non-separable case.

Support vector classifiers

When the two classes are not linearly separable (e.g., Figure 4), we will not be able to find a line that entirely separates the groups, that is, the maximal margin classifier can not be computed. Thus, the two classes cannot be classified exactly. We can, however, generalize the ideas to develop a classification rule that *almost* separates the classes. To do so, we allow a few points to fall on the wrong side

of the margin or separating hyperplane. Such a classifier is called a *support vector classifier* or a *soft-margin classifier*.[3]

Even if the classification problem is completely separable, we still may consider such soft margin classifier for robustness. A separating hyperplane based classifier will perfectly classify all of the training data (training error = 0), which can lead to sensitivity to individual observations. An example is shown in Figure 5. The addition of a single observation in the right-hand panel of Figure Figure 5 leads to a dramatic change in the maximal margin hyperplane. Thus we might want to consider a classifier based on a hyperplane that does not perfectly separate the two classes, in the interest of greater robustness to individual observations, and better classification of most of the training observations – it could be worthwhile to misclassify a few training observations in order to do a better job in classifying the remaining observations.

In support vector classifier, each data point i is given a *slack variable* e_i that allow individual data points to be on the wrong side of the margin or the separating hyperplane. The slack variables e_i quantifies where the i-th observation is located relative to the hyperplane and the margin:

- If $e_i = 0$ then the i-th data point is on the correct side of the margin;
- If $e_i > 0$ then the i-th observation is on the wrong side of the margin (the data point has *violated the margin*),
- If $e_i > 1$ then it is on the wrong side of the hyperplane.

The support vector classifiers then attempt to maximize the margin such that $\sum_i e_i \leq L$, for a pre-specified constant L.[4] Formally, we solve the optimization problem:

$$\max \quad M$$

with respect to $\beta_0, \ldots, \beta_p$ and $e_1, \ldots, e_n$, subject to the constraints

$$Y_i(\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p) \geq M(1 - e_i), i = 1, \ldots, n,$$

$$\beta_1^2 + \ldots + \beta_p^2 = 1,$$

$$e_i \geq 0, e_1 + \ldots + e_n \leq L,$$

where L is a nonnegative tuning parameter.

Conceptually, the value e_i in the constraint $Y_i(\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p) \geq M(1 - e_i)$ is the proportional amount by which the quantity $f(\mathbf{X}_i) = \beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p$ is on the wrong side of its margin. Hence by bounding the sum $e_1 + \ldots + e_n$, we bound the total proportional amount by which $f(\mathbf{X}_i)$ fall on the wrong side of their margin.

[3] The term "soft margin" is coined because the margin can be violated by some of the training observations.

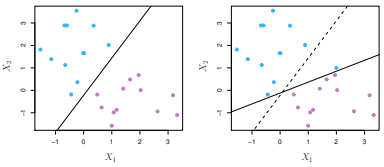


Figure 5: Two classes of observations are shown in blue and in purple, along with the maximal margin hyperplane (left panel). An additional blue observation has been added, leading to a dramatic shift in the maximal margin hyperplane shown as a solid line (right panel). The dashed line indicates the maximal margin hyperplane that was obtained in the absence of this additional point.

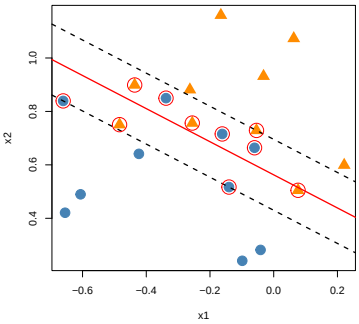


Figure 6: Simulated two-class data with support vectors and separating hyperplane (line) highlighted for the non-separable case.

[4] The constant L controls the number and severity of the violations to the margin and to the hyperplane that can be tolerated by the classifier.

Misclassifications occur when $e_i > 1$, so bounding $e_1 + \ldots + e_n$ at a value L bounds the total number of training misclassifications at L. Specifying $L = 0$ gives us the maximal margin classifier, if it exists. In contrast, as L increases we become more tolerant of violations to the margin, and so the margin will widen. In practice, we need to choose L via cross-validation.

As before, we can write an equivalent optimization problem:

$$\min \ (\beta_1^2 + \ldots + \beta_p^2)$$

subject to the constraints

$$Y_i(\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p) \geq 1 - e_i, i = 1, \ldots, n,$$

$$e_i \geq 0, e_1 + \ldots + e_n \leq L.$$

This is the usual way the support vector classifier is defined for the nonseparable case.

Once we obtain the estimators $\widehat{\beta}_0, \widehat{\beta}_1, \ldots, \widehat{\beta}_p$, the estimated hyperplane is $\widehat{f}(\boldsymbol{x}) = \widehat{\beta}_0 + x_1\widehat{\beta}_1 + \ldots + x_p\widehat{\beta}_p$, and our classification rule is as follows: for a new observation $\boldsymbol{x} = (x_1, \ldots, x_p)$,

$$\widehat{Y} = sign\{\widehat{f}(\boldsymbol{x})\}.$$

The classification boundary, and the two margins are

$$\widehat{f}(\boldsymbol{x}) = 0, \ \text{ and } \ \widehat{f}(\boldsymbol{x}) = \pm 1,$$

respectively.

As with the maximal margin classifier, *the classifier is affected only by the observations that lie on the margin or violates the margin*. Data points that lies strictly on the correct side of the margin does not affect the support vector classifier at all. In this case, data points that fall directly on the margin, or on the wrong side of the margin for their class, are known as *support vectors*. The circled points in Figure 4 are the support vectors for the the classifier shown using the red line.[5]

Let us now revisit the wines data. We can use the `svm()` function in the `e1071` library to implement support vector classifier[6]. We will only use two classes to begin with (1 and 2) and two covariate, `Alcohol` and `Proline`, so that we can plot the results. Support vector classifiers however can be implemented for much larger number of predictors.[7] While it is technically not needed, we often standardive the predictors beforehand – `svm()` does standardization by default. We perform standardization manually so that we can compare the coefficients of the resulting hyperplane.

[5] Since support vector classifier is based only on a small subset of the training set – the support vectors – it is quite robust to the behavior of data points that are far away from the hyperplane.

[6] There are many other functions such as `ksvm` in the `kernlab` library that perform SVMs.

[7] The argument type = "C-classification" specifies that we want to perform the support vector classification.

```r
# Read wine data
wines <- read.table("data/Wines.txt", header = TRUE)
# Two class (1 and 2) data
wine_twoclass <- wines[wines$Class < 3, ]
wine_twoclass$Class <- as.factor(wine_twoclass$Class)
# Standardize the predictors (excluding Class)
wine_twoclass[,-1] <- scale(wine_twoclass[,-1],
                            center = TRUE, scale = TRUE)
# SVC with linear boundary
sv.wine <- svm(Class ~ Proline + Alcohol,
               data = wine_twoclass,
               type = "C-classification",
               kernel = "linear",
               cost = 1)
sv.wine
```

```
##
## Call:
## svm(formula = Class ~ Proline + Alcohol, data = wine_twoclass, type = "C-classification",
##     kernel = "linear", cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  18
```

The argument `kernel = "linear"` ensures that we are using support vector classifier. Later, when we learn *support vector machine*, we will specify different nonlinear kernels. The parameter `cost` takes the role of L, however, the `svm()` function uses a different mathematical formulation than what we discuss above. Thus `cost` is not exactly same as L. The main concepts, however, remain the same. When the `cost` argument is small, then the margins will be wide and many support vectors will be on the margin or will violate the margin. When the cost argument is large, then the margins will be narrow and there will be few support vectors on the margin or violating the margin. The case `cost` $= \infty$ corresponds to the perfectly separable case.

The coefficients of the separating hyperplane can be extracted using the `coef()` function.

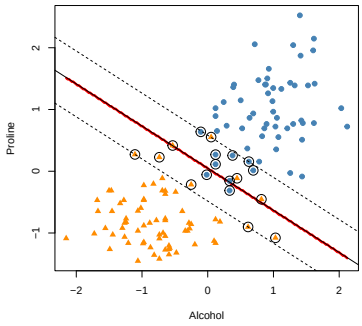


Figure 7: Classification of wine data using support vector classifier.

```
beta_hat <- coef(sv.wine)
beta_hat
```

```
## (Intercept)     Proline     Alcohol
## -0.09306929  1.87645694  1.28020122
```

Thus, the hyperplane is

$$-0.093 + 1.876 * \text{Proline} + 1.28 * \text{Alcohol} = 0.$$

The lines indicating the two margins are

$$-0.093 + 1.876 * \text{Proline} + 1.28 * \text{Alcohol} = 1,$$

and

$$-0.093 + 1.876 * \text{Proline} + 1.28 * \text{Alcohol} = -1,$$

respectively. The points that are on the margin or violate the margin are support vectors.

Here, we have used the default value of `cost=1`. However, it is very important to choose `cost` using cross-validation for better optimization. We can use `caret` to choose `cost`.

```
# Set up repeated cv option
set.seed(1001)
tr <- trainControl(method = "repeatedcv",
                   number = 5, repeats = 10)
# Tuning grid
tune_grid <- expand.grid(cost = exp(seq(-5,3,len=30)))
# Train the model
sv_caret <- train(Class ~ Proline + Alcohol,
                  data = wine_twoclass,
                  method = "svmLinear2",
                  tuneGrid = tune_grid,
                  trControl = tr)
```

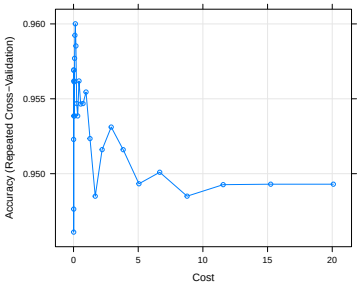


Figure 8: Results from repeated CV using support vector classifier on two-class wines data.

```
# Best C
sv_caret$bestTune
```

```
##         cost
## 12 0.1400834
```

```
# Final fit
wine_sv_final <- svm(Class ~ Proline + Alcohol,
                     data = wine_twoclass,
                     type = "C-classification",
```

```
                      kernel = "linear",
                      cost = sv_caret$bestTune$cost)
wine_sv_final
```

```
##
## Call:
## svm(formula = Class ~ Proline + Alcohol, data = wine_twoclass, type = "C-classification",
##     kernel = "linear", cost = sv_caret$bestTune$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1400834
##
## Number of Support Vectors:  30
```

To obtain the predicted classes, we can use the `predict()` function.[8]

[8] See documentation of `predict.svm()`.

```
pred.class <- predict(object = wine_sv_final,
                      newdata = wine_twoclass,
                      type = "response")
err <- klaR::errormatrix(true = wine_twoclass$Class,
                  predicted = pred.class,
                  relative = TRUE)
round(err, 3)
```

```
##        predicted
## true        1     2 -SUM-
##   1     0.966 0.034 0.034
##   2     0.042 0.958 0.042
##   -SUM- 0.600 0.400 0.038
```

More than two classes

Suppose that we have $K > 2$ classes. One simple way to extend SVM to this situation is to compute all pair-wise classifiers, that is, compute all $C_2^K = K(K-1)/2$ classification rules. This is often called *one-versus-one* approach. Given a test observation, we classify it using each of the C_2^K classifiers, and record the number of times that the test observation is assigned to each of the K classes. The final classification is performed taking a *majority vote*.

Another approach is to compare class k with the remaining classes together, i.e., class k vs. not class k. This is called the *one-versus-all* approach. Let $\widehat{\beta}_{0k}, \ldots, \widehat{\beta}_{pk}$ denote the parameters that result from fitting

an SVM comparing the k-th class (coded as $+1$) to the others (coded as -1). Given a test observation, x, we assign the observation to the class for which $\widehat{\beta}_{0k} + x_1\widehat{\beta}_{1k} + \ldots + x_p\widehat{\beta}_{pk}$ is largest, as this amounts to a high level of confidence that the test observation belongs to the kth class rather than to any of the other classes.

The `svm()` function uses *one-versus-one* approach. Let us use the full `wines` data with three classes, but with two predictors `Alcohol` and `Proline`. We show support vector classifier with `cost` chosen by cross-validation for demonstration.

```
# Pre-process wines data
wines$Class <- as.factor(wines$Class)
wines[, -1] <- scale(wines[,-1],
                     center = TRUE, scale = TRUE)
# Train the model
set.seed(1001)
tr <- trainControl(method = "repeatedcv",
                   number = 5, repeats = 10)
tune_grid <- expand.grid(cost = exp(seq(-5,3,len=30)))
sv_caret <- train(as.factor(Class) ~ Proline + Alcohol,
                  data = wines,
                  method = "svmLinear2",
                  tuneGrid = tune_grid,
                  trControl = tr)
```

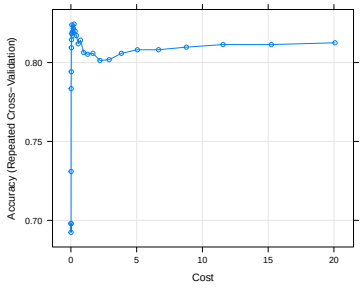


Figure 9: CV results for three class wines data.

```
# SVC with optimal cost
sv.wine <- svm(Class ~ Proline + Alcohol,
               data = wines,
               type = "C-classification",
               kernel = "linear",
               cost = sv_caret$bestTune$cost)
```

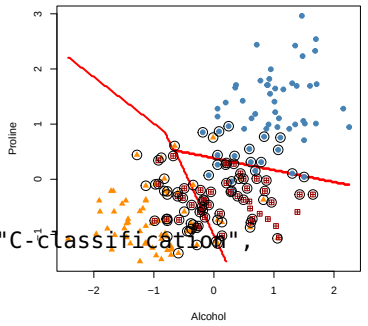


Figure 10: Classification of wine data using support vector classifier with all three classes.

```
sv.wine
```

```
##
## Call:
## svm(formula = Class ~ Proline + Alcohol, data = wines, type = "C-classification",
##     kernel = "linear", cost = sv_caret$bestTune$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
```

```
##        cost:  0.243219
##
## Number of Support Vectors:  90
```

Support vector machines

The support vector classifier described so far finds linear boundaries in the input feature space. Often linear effects of the covariates $\mathbf{X}$ is not enough for a classification problem. Thus we might want to incorporate nonlinear terms (e.g., square or cubic terms). For example, if we have two covariates X_1 and X_2. So we might include X_1, X_2, X_1^2, X_2^2 and X_1X_2 in our classifier. If we run the support vector classifier, the decision boundary would be a quadratic polynomial. In general, we can incorporate other nonlinear transformations $h_1(\mathbf{X}), \ldots, h_M(\mathbf{X})$ as features. We can make the procedure more flexible by enlarging the feature space using basis expansions such as polynomials or splines. Generally linear boundaries in the enlarged space achieve better training-class separation, and translate to nonlinear boundaries in the original space. Once the basis functions $h_m(\boldsymbol{x}), m = 1, \ldots, , M$ are selected, the procedure is the same as before. We fit the support vector classifier using input features $h_1(\mathbf{X}_i), h_2(\mathbf{X}_i), \ldots, h_M(\mathbf{X}_i)$, $i = 1, \ldots, N$, and produce the (nonlinear) function

$$\widehat{f}(\boldsymbol{x}) = \widehat{\beta}_0 + h_1(\boldsymbol{x})\widehat{\beta}_1 + \ldots + h_M(\boldsymbol{x})\widehat{\beta}_M.$$

The classifier is $\widehat{Y} = sign(\widehat{f}(\boldsymbol{x}))$ as before.

As an example, the following code includes natural cubic spline terms with four degrees of freedom of `Alcohol` and `Proline` with `cost` $=$ 0.2. The decision boundaries are quadratic, as shown in Figure 11.

```
library(splines)
sv.wine <- svm(Class ~ ns(Proline, df=4) + ns(Alcohol, df=4),
               data = wines,
               type = "C-classification",
               kernel = "linear",
               cost = 0.2)
```

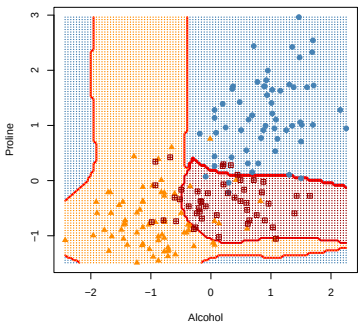


Figure 11: Classification of wine data using support vector classifier with natural cubic splines.

The core idea is that even though the two classes are not separable in the original feature space, they may be separable in transformed, and/or expanded feature space. For example, consider a simulated data in Figure 12, panels (a) and (b). The dataset has two classes and two predictors, X_1 and X_2. Panel (a) shows the original data. Even though we see the separation between two classes visually, fitting a linear support vector classifier is not ideal here. Panel (b) plots the

transformed predictors X_1^2 and X_2^2. In the transformed feature space, the two classes are linearly separable. Specifically, we see that there is a linear combination of the form $aX_1^2 + bX_2^2 + c = 0$ that separates the two classes.

Panels (c) – (e) in Figure 12, show another example. Panel (c) shows the two original features. Panel (d) shows transformed predictors X_1^2 and X_2^2, but they do not linearly separate the classes. It turns out we need a third predictor X_1X_2 to linearly separate the classes, see Panel (e). Thus we see that there is a linear combination of the form $aX_1^2 + bX_2^2 + cX_1X2 + d = 0$ that separates the two classes. In general, as we mention above, we can include other nonlinear transformations $h_1(\mathbf{X}), \ldots, h_M(\mathbf{X})$ as features in support vector classifiers.

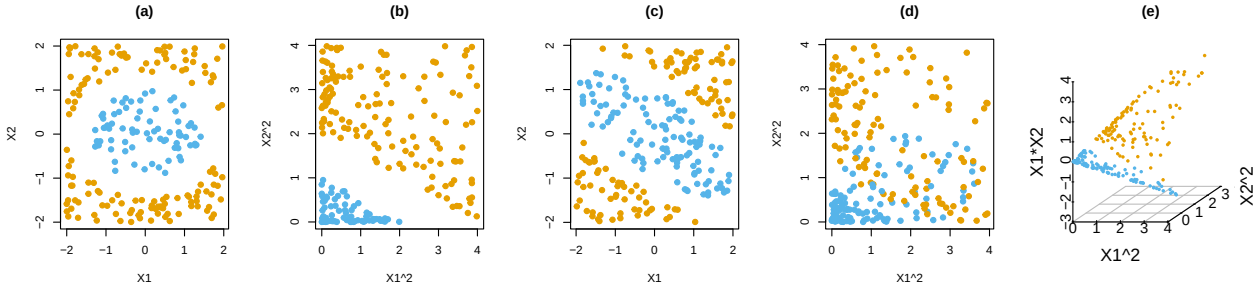


Figure 12: Simulated example of transformed and enlarged feature space.

Support vector machines (SVM) generalize support vector classifiers by including nonlinear features in a specific way that allows us to add many such features as well as a high number of variables. The dimension of the enlarged space is allowed to get very large, infinite in some cases. Without going into mathematics, SVM does so using the so called *kernel trick*, that is, by specifying a *kernel function* that controls which nonlinear features to include in the classifier. To see this, let us briefly look into how support vector classifier computes the classifier, that is, how the optimization is done. Define $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^T$. It turns out that

$$\widehat{\boldsymbol{\beta}} = \sum_{i=1}^{n} \widehat{\alpha}_i Y_i \mathbf{X}_i,$$

for some weights $\widehat{\alpha}_i$. Thus the solution to the support vector classifier problem can be represented as

$$\widehat{f}(\boldsymbol{x}) = \widehat{\beta}_0 + \boldsymbol{x}^T\boldsymbol{\beta} = \widehat{\beta}_0 + \sum_{i=1}^{n} \widehat{\alpha}_i \boldsymbol{x}^T\mathbf{X}_i\, Y_i = \widehat{\beta}_0 + \sum_{i=1}^{n} \widehat{\alpha}_i \langle \boldsymbol{x}, \mathbf{X}_i \rangle\, Y_i.$$

To estimate $\widehat{\beta}_0$ and $\widehat{\alpha}_1, \ldots, \widehat{\alpha}_n$, it can be shown that we only need the all *pair-wise* inner products of the training data $\langle \mathbf{X}_i, \mathbf{X}_{i'} \rangle$. Many of

the resulting solutions $\widehat{\alpha}_i$ are zero. The observations for which $\widehat{\alpha}_i$ are nonzero are called the *support vectors*.

To summarize, in representing the linear classifier $f(x)$, and in computing its coefficients, all we need are inner products: $\langle \mathbf{X}_i, \mathbf{X}_{i'} \rangle, i, i' = 1, \ldots, n$, and $\langle x, \mathbf{X}_i \rangle$.

Therefore, for general nonlinear features

$$h(\mathbf{X}_i) = [h_1(\mathbf{X}_i), h_2(\mathbf{X}_i), \ldots, h_M(\mathbf{X}_i)]^T,$$

the classifier $\widehat{f}(x)$ can be computed using the inner products: $\langle h(\mathbf{X}_i), h(\mathbf{X}_{i'}) \rangle$ and $\langle h(x), h(\mathbf{X}_i) \rangle$. In fact, we need not specify the transformation $h(x)$ at all, but require only knowledge of the *kernel function*

$$K(x, x') = \langle h(x), h(x') \rangle,$$

that computes inner products in the transformed space.

In general, the kernel function $K(\cdot, \cdot)$ should be a symmetric positive (semi-) definite function. Some popular choices for $K(\cdot, \cdot)$ in the SVM are

$$\text{Linear} : K(x, x') = \langle x, x' \rangle,$$

$$d\text{-th degree polynomial} : K(x, x') = (1 + \langle x, x' \rangle)^d,$$

$$\text{Radial basis} : K(x, x') = exp(-\gamma ||x - x'||^2),$$

$$\text{Neural network} : K(x, x') = tanh(\kappa_1 \langle x, x' \rangle + \kappa_2).$$

For example, using the "linear" or "quadratic" ($d = 2$ degree polynomial) kernel will result in a linear or quadratic classification boundaries, respectively. On the other hand, using a "radial basis kernel" captures other nonlinear features. As an example, Figure 13 shows three classification rules corresponding to linear, quadratic and radial kernels based on a simulated data set.

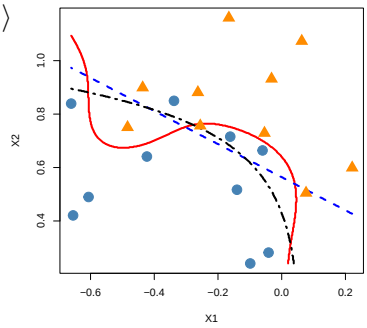


Figure 13: Simulated two-class data with linear (blue dashed), quadratic (black dash-dotted) and radial-basis based (red solid) classification rules.

To see the correspondence between kernels and original features, consider for example a feature space with two inputs X_1 and X_2, and a polynomial kernel of degree 2. Then

$$K(\mathbf{X}, \mathbf{X}') = (1 + \langle \mathbf{X}, \mathbf{X}' \rangle)^2 = 1 + 2X_1X_1' + 2X_2X_2' + (X_1X_1')^2 + (X_2X_2')^2 + 2X_1X_1'X_2X_2'.$$

Therefore, $M = 6$, and with the choice $h_1(\mathbf{X}) = 1, h_2(\mathbf{X}) = \sqrt{2}X_1, h_3(\mathbf{X}) = \sqrt{2}X_2, h_4(\mathbf{X}) = X_1^2, h_5(\mathbf{X}) = X_2^2, h_6(\mathbf{X}) = \sqrt{2}X_1X_2$, we have $K(\mathbf{X}, \mathbf{X}') = \langle h(\mathbf{X}), h(\mathbf{X}') \rangle$.

Let us now revisit the wines data. Figure 14 shows the decision boundaries using the radial basis kernel SVM. Here we set $\gamma = 1$. In practice, we need to explore a few values of γ to select an optimal value based on test performance.

```
sv.wine <- svm(Class ~ Proline + Alcohol,
               data = wines,
               type = "C-classification",
               kernel = "radial",
               gamma = 1,
               cost = 0.2)
```

```
sv.wine
```

```
##
## Call:
## svm(formula = Class ~ Proline + Alcohol, data = wines, type = "C-classification",
##     kernel = "radial", gamma = 1, cost = 0.2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.2
##
## Number of Support Vectors:  114
```

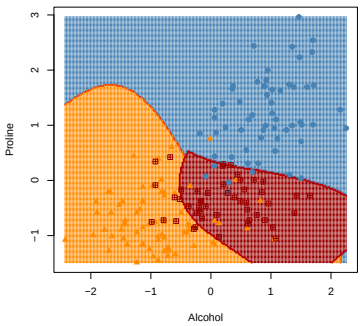


Figure 14: Classification of wine data using radial kernel SVM.

In general, non-linear kernels are better in capturing nonlinear boundaries compared to linear methods like LDA, Logistic regression etc. Figure 15 shows two examples, the left panel uses a polynomial kernel of degree 3, and right panel uses a radial basis kernel. In both these cases, linear support vector classifiers would not work well.

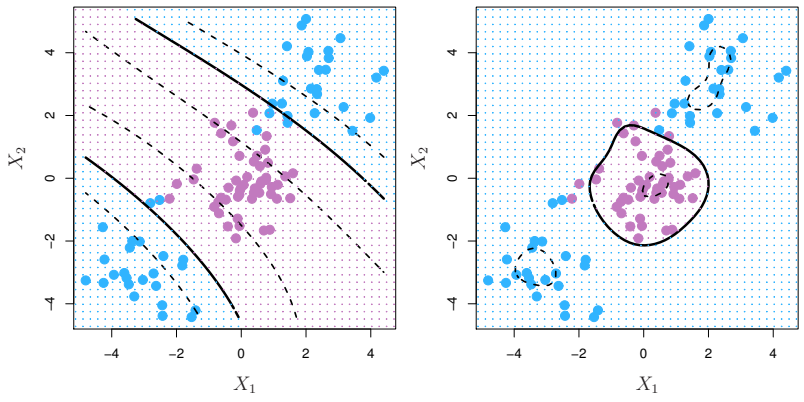


Figure 15: Left: An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from Figure 9.8, resulting in a far more appropriate decision rule. Right: An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary. Figure and caption taken from the textbook.

One advantage of using a kernel rather than simply enlarging the feature space using transformation of the original features is ease of computation. Using kernels, we only need to compute inner products

distinct pairs of observations $\langle \mathbf{X}_i, \mathbf{X}_{i'} \rangle$. This can be done without explicitly working in the enlarged feature space. This is important because in many applications of SVMs, the enlarged feature space is so large that computations are intractable. For some kernels, e.g., radial kernel, the feature space is implicit and infinite-dimensional, so it is not feasible to specify the entire feature space using explicit basis functions.

SVMs and the Curse of Dimensionality

Even though some publications claim that SVMs have some edge on the curse of dimensionality over other methods, such claims may not be true.

> In the early literature on support vectors, there were claims that the kernel property of the support vector machine is unique to it and allows one to finesse the curse of dimensionality. Neither of these claims is true. – *Elements of Statistical Learning* by Hastie et al.

Consider the example a feature space with two inputs X_1 and X_2, and a polynomial kernel of degree 2. Then

$$K(\mathbf{X}, \mathbf{X}') = (1 + \langle \mathbf{X}, \mathbf{X}' \rangle)^2 = 1 + 2X_1X_1' + 2X_2X_2' + (X_1X_1')^2 + (X_2X_2')^2 + 2X_1X_1'X_2X_2'.$$

Therefore, $M = 6$, and with the choice $h_1(\mathbf{X}) = 1, h_2(\mathbf{X}) = \sqrt{2}X_1, h_3(\mathbf{X}) = \sqrt{2}X_2, h_4(\mathbf{X}) = X_1^2, h_5(\mathbf{X}) = X_2^2, h_6(\mathbf{X}) = \sqrt{2}X_1X_2$, we have $K(\mathbf{X}, \mathbf{X}') = \langle h(\mathbf{X}), h(\mathbf{X}') \rangle$. This kernel is not incorporating a fully general inner product in the space of powers and products. For example, all terms of the form $2X_jX_j'$ are given equal weight, and the kernel cannot adapt itself to concentrate on subspaces. If the number of features p were large, but the class separation occurred only in linear effects of X_1 and X_2, this kernel would not easily find the structure. Thus it would suffer from having many dimensions to search over. Thus to get a better performing kernel, we need prior knowledge about the proper subspace in which the classes are separated, and build a kernel based on that. However, such knowledge is unlikely to be readily available – a major goal of adaptive methods is to discover such structure.

Chapter 12.3.4 of *Elements of Statistical Learning* by Hastie, Tibshirani and Friedman, presents a simulation study with two classes, where a hyperplane cannot separate the classes in the original feature space, and consequently the support vector classifier performs poorly. The polynomial support vector machine makes a substantial improvement in test error rate, but is found to be adversely affected by noise features. Higher degree polynomial kernels performed

much worse compared to 2nd degree polynomial kernel in this setting, indicating that SVM can be very sensitive to the choice of kernel function.

SVM as penalized method and Support Vector Regression

Suppose we define $f(\boldsymbol{x}) = \beta_0 + h_1(\boldsymbol{x})\beta_1 + \ldots + h_M(\boldsymbol{x})\beta_M$, where $h_m(\boldsymbol{x})$ are transformations of the original functions. Then it can be shown that the support vector classifier problem with $h_1(\boldsymbol{x}), \ldots, h_M(\boldsymbol{x})$ as predictors is equivalent to solving the following penalized problem:

$$min \sum_{i=1}^{n}[1 - Y_i f(\mathbf{X}_i)]_+ + \frac{\lambda}{2}\sum_{m=1}^{M}\beta_m^2,$$

where t_+ denotes the positive part, that is,$t_+ = tI(t > 0)$, and λ is a penalty parameter. The loss function above is also called the *hinge loss*. The optimization problem has the typical form of `loss + penalty` that we have encountered in our discussion of regularized regression.

The penalty above is the ridge penalty. Some recent works[9] replaces the ridge penalty with the lasso and elastic net penalty to obtain sparse solution of the coefficients. The `sparseSVM` library performs such a procedure in R. For the two-class wines data, let us run the support vector classifier based on all 13 predictors.

[9] Yi, C. and Huang, J. (2017) Semismooth Newton Coordinate Descent Algorithm for Elastic-Net Penalized Huber Loss Regression and Quantile Regression, Journal of Computational and Graphical Statistics, 547-557.

```
# Set up repeated cv option
set.seed(1001)
tr <- trainControl(method = "repeatedcv",
                   number = 5, repeats = 10)
# Tuning grid
tune_grid <- expand.grid(cost = exp(seq(-5,3,len=30)))
# Train the model
sv_caret <- train(Class ~ .,
                  data = wine_twoclass,
                  method = "svmLinear2",
                  tuneGrid = tune_grid,
                  trControl = tr)
# Final model
wine_sv_final <- svm(Class ~ .,
                     data = wine_twoclass,
                     type = "C-classification",
                     kernel = "linear",
                     cost = sv_caret$bestTune$cost)
sv_caret$bestTune
```

```
##         cost
## 11 0.1063118
```

```
beta_hat <- coef(wine_sv_final)
beta_hat
```

```
## (Intercept)     Alcohol       Malic         Ash       Alcal          Mg
## -0.12034879  0.56737384  0.19086682  0.42391421 -0.49320791  0.03062720
##      Phenol        Flav        Nonf       Proan       Color         Hue
## -0.05963129  0.04504281 -0.09314091 -0.06333361  0.21875381 -0.11699293
##         Abs     Proline
##  0.30659144  0.72825654
```

Notice that the standard support vector classifier does not set any of the small coefficients to exactly zero. Recall that we have already standardized the two-class wines data. So we suspect predictors with smaller values of coefficients (e.g., `Mg`, `Phenol`, `Flav` etc) may not contribute to the classification rule as much as other variables like `Alcohol`, `Proline`, `Ash`, `Alcal`, which have relatively large coefficients.

Now let us run the same classifier with lasso penalty. We will use CV to choose λ. By default, the R function uses 10-fold CV.

```
library(sparseSVM)
set.seed(1001)
X <- as.matrix(wine_twoclass[,-1])
y <- wine_twoclass$Class
# Cross validation to choose lambda
spr.cv <- cv.sparseSVM(X = X, y = y, alpha = 1)
spr.cv$lambda.min
```

```
## [1] 0.08655872
```

```
plot(spr.cv)
abline(v = log(spr.cv$lambda.min))
```

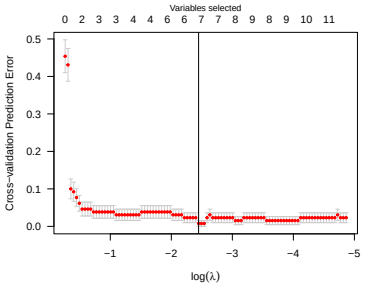


Figure 16: 10-fold CV results for sparse support vector classifier.

```
# Final fit
beta_sparse <- coef(spr.cv,
                    lambda = spr.cv$lambda.min)
cbind(beta_sparse, beta_hat)
```

```
##             beta_sparse    beta_hat
## (Intercept) -0.04816313 -0.12034879
## Alcohol      0.59791703  0.56737384
## Malic        0.00210073  0.19086682
```

```
## Ash          0.07456517  0.42391421
## Alcal       -0.02877570 -0.49320791
## Mg           0.00000000  0.03062720
## Phenol       0.00000000 -0.05963129
## Flav         0.07439179  0.04504281
## Nonf         0.00000000 -0.09314091
## Proan        0.00000000 -0.06333361
## Color        0.00000000  0.21875381
## Hue          0.00000000 -0.11699293
## Abs          0.16252337  0.30659144
## Proline      0.88202057  0.72825654
```

Note that quite a few variables have exactly zero coefficients in the final fit.

SVM can be applied to regression problems with quantitative response as well. Let us start with a linear regression problem

$$E(Y_i|\mathbf{X}_i) = f(\mathbf{X}_i),$$

where

$$f(\mathbf{X}_i) = \beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p = \beta_0 + \mathbf{X}_i^T\boldsymbol{\beta}.$$

Support vector regression solves the following problem:

$$min \sum_{i=1}^{n} V_\epsilon(Y_i - f(\mathbf{X}_i)) + \frac{\lambda}{2}\sum_{j=1}^{p}\beta_j^2,$$

where the loss function $V_\epsilon(\cdot)$ has the form

$$V_\epsilon(r) = \begin{cases} 0 \text{ if } |r| < \epsilon, \\ |r| - \epsilon \text{ otherwise.} \end{cases}$$

Thus this loss function ignores errors of size less than ϵ, see Figure 17. It can be shown that the solution function has the form

$$\widehat{f}(\boldsymbol{x}) = \sum_{i=1}^{n}(\widehat{\alpha}_i^* - \widehat{\alpha}_i)\langle \boldsymbol{x}, \mathbf{X}_i\rangle + \widehat{\beta}_0,$$

where $\widehat{\alpha}_i^*$ and $\widehat{\alpha}_i$ are constants. Typically only a subset of the values $(\widehat{\alpha}_i^* - \widehat{\alpha}_i)$ are nonzero, and the associated data values are called the support vectors. As was the case in the support vector classification, the solution depends on the input values only through the inner products $\langle \mathbf{X}_i, \mathbf{X}_{i'}\rangle$. Thus we can generalize the methods to richer spaces by defining an appropriate inner product, and specifying the corresponding kernel function.

In R, the `svm()` functions can perform regression as well. Here we have two parameters to tune: ϵ and `cost`. This can be done using usual methods such as CV. Here we demonstrate support vector

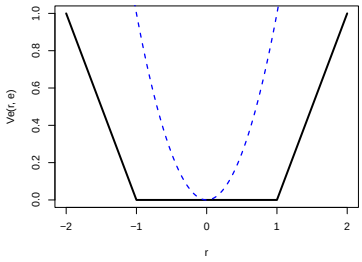


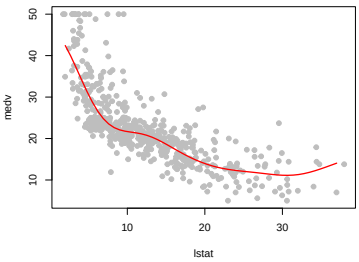
Figure 17: The loss function for support vector regression for epsilon value 1. The blue dashed line is the usual squared error loss.

regression for a specific value of $\epsilon = 0.1$ and `cost = 1`, using `Boston` data. We use radial kernel with $\gamma = 1$.

```
library(ISLR2)
svr <- svm(medv ~ lstat, data = Boston,
           type = "eps-regression",
           kernel = "radial", gamma = 1,
           cost = 1, epsilon = 0.1
           )
summary(svr)
```

```
##
## Call:
## svm(formula = medv ~ lstat, data = Boston, type = "eps-regression",
##     kernel = "radial", gamma = 1, cost = 1, epsilon = 0.1)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##     epsilon:  0.1
##
##
## Number of Support Vectors:  406
```

```
# Prediction
xnew <- data.frame(lstat = seq(2, 37, len=51))
pred <- predict(svr, newdata = xnew)
# Plot
plot(medv ~ lstat, data = Boston, pch=19, col = "gray")
lines(xnew$lstat, pred, col = "red", lwd=2)
```



While the example above involves only one predictor, support vector regression can accommodate multiple predictors.

Overall, SVMs are quite useful in practice, and extend beyond what we discussed in this chapter. For example, the penalized loss function formulation, as described in the classification and regression context can be used for *any* convex loss function with *any* kernel function. This enables us to use kernel methods in function estimation in linear, generalized linear (e.g., logistic), and other (e.g., least absolute deviation) regression models.