Spatial Data Analysis with Support Vector Machines

Richard E. Plant

September 15, 2020

Additional topic to accompany Spatial Data Analysis in Ecology and Agriculture using R, Second Edition

http://psfaculty.plantsciences.ucdavis.edu/plant/sda2.htm

Chapter and section references are contained in that text, which is referred to as SDA2.

1. Introduction

To the practitioner, the term “support vector machines” conjures up such a complete blank that the name is an invitation to think that whatever they are, they must be really cool and powerful. Support vector machines, which we will abbreviate SVM from now on, were such a phenomenon in the early 2000’s that in his discussion of an implementation of SVM in R, Meyer (2018) begins with a citation of a paper titled “Hype or Hallelujah?” (Bennett and Campbell, 2000). It is probably fair to say that neither Hype nor Hallelujah is exactly the case, but that SVM are a very useful addition to the practitioner’s toolkit. The easiest way to begin the discussion of SVM is by parsing out the name. Of the three mysterious words the easiest is the middle one, “vector”. If we think of a data set such as any of the four data sets of SDA2, we can envision the main data as comprised of data fields, which are the columns of the csv file containing the data, and consisting of a collection of data records, which are the rows of the csv file.

For ease of visualization, let’s start with the training data set that we used in our Additional Topic on the K nearest neighbor method to introduce that method. It consists of three data fields: QUDO, which has a value of 0 or 1 depending on whether a blue oak (Quercus douglasii) is absent or present at the site; MAT.n, the normalized mean annual temperature; and Precip.n, the normalized mean annual precipitation. QUDO is the variable to be classified. In the general case will denote this variable Y. As with the Additional Topic on the K nearest neighbor method, we will refer to the Yi as the class labels. The data fields MAT.n and Precip.n are the attributes on which the classification is to be based. These are called the predictors and will in the general case be denoted Xi, i = 1, … p. For this particular example p = 2. The training data set is created by randomly selecting 25 records each of both QUDO = 1 and QUDO = 0 from the full data set. Here are the first six members.
Figure 1. Representation of the KNN training data as vectors in a vector space, showing in particular the arrow representing the vector characterizing the 45th point.

Fig. 1 shows a plot in the data space of the values of QUDO. Also shown in the figure is an arrow representing the point \((X_{45,1}, X_{45,2})\). This is the representation of the data space as a vector space, and each point (i.e., data record) in that space as a vector. This is the sense of the word “vector” in “support vector machines.” As a final note before we move on, we will see later that the SVM method is very sensitive to the scale of the predictor data, which is a reason to normalize the training data values.
2. Separating hyperplanes

Now we turn to the word “support.” For this we will use a simple made-up example shown in Fig. 2. There are \( n = 20 \) data records, \( p = 2 \) predictors \( X_1 \) and \( X_2 \), and two class labels \( Y_1 \) and \( Y_2 \). Moreover, the regions containing \( Y_1 \) and \( Y_2 \) are completely separate in the data space so that a line can be drawn between them. For ease of exposition we will take the class labels to be \( Y_1 = 1 \) and \( Y_2 = -1 \) (rather than 0). If the number of attributes \( p \) was equal to 3 and the class labels were completely separable like this then they would be separated by a plane in a 3 dimensional vector space, and if \( p \) was greater than three then the class labels would be separated by a hyperplane in an \( n \) dimensional vector space. This cannot be visualized but it can be conceived of. Because this latter is the general case, the special cases \( p = 2 \) and \( p = 3 \) use the same terminology and the lines in Fig. 2 are called separating hyperplanes.

![Figure 2](image)

Figure 2. An example showing data that can be separated into two distinct classes. Three separating hyperplanes are shown.

As shown in Fig. 2, there is more than one separating hyperplane. We would like to find the “best” one, in the sense that it is the greatest possible distance from both collections of class labels. This is called the optimal separating hyperplane. The optimal separating hyperplane is as far as possible from a line parallel to it and touching the nearest class label of the \( Y_1 \), and
similarly it is as far as possible from the parallel line touching the nearest $Y_2$ (Fig. 3) This
hyperplane avoids to the greatest extent possible the chance that other data will be misclassified.
As shown in Fig. 3, we can deduce from geometry that there are two possible candidates for the
optimal separating hyperplane. It must be the centerline of one of two “streets,” either one whose
“curbs” (the dashed lines) touch two red points and one green one (Fig. 3a), or touch two green
points and one red one (Fig. 3b). These “curbs,” taken together, are called the geometric margin.
Notice from Fig. 3 that the optimal separating hyperplane depends only on the data points (i.e.,
vectors) touching the dashed lines (the “curbs”). If any of these three vectors were moved, the
geometric margin would move and so would the optimal separating hyperplane. These vectors
therefore “support” the optimal separating hyperplane, and so are called “support vectors.” Two
terms down and one to go.

![Figure 3. Two possible optimal separating hyperplanes](image)

3. The support vector classifier

Understanding the meaning of the third term, “machine,” will take a little more effort. To begin,
consider the separating hyperplanes represented in Fig. 3a and 3b by the solid lines. As usual we
follow the convention established in SDA2 of denoting data attributes with upper case letters,
saving the lower case for geographic coordinates. We can write the equation for a line in the vector space generally as

\[ a_0 + a_1X_1 + a_2X_2 = 0. \]  

(1)

The three specific data points (which are also, remember, vectors) that define the geometric margin in Fig. 3a are \( X_2, X_5, \) and \( X_{13} \). Similarly, in Fig. 3b the points are \( X_2, X_5, \) and \( X_{15} \). We have to be careful here with subscripts. Following the convention of linear regression (SDA2, Appendix A), we will denote the specific values of a data vector \( X_i \) using the pair \( (X_{i,1}, X_{i,2}) \). We have already used this notation in Section 1. Thus for example the data vector \( X_{41} \) has coordinates \( (X_{41,1}, X_{41,2}) \).

As stated in Section 2, the separating hyperplane is the center line of the “street” whose “curbs” are the dashed lines. The optimal separating hyperplane is the one whose distance \( d \) from the center line to the curb is the largest (the widest “street”). To carry out this calculation we need to know how to express the distance from a line to a point. In our notation, the distance from a line defined by the equation \( a_0 + a_1X_1 + a_2X_2 = 0 \) to a point \( X_i \) is expressed as

\[ d_i = \frac{|a_0 + a_1X_{i,1} + a_2X_{i,2}|}{\sqrt{a_1^2 + a_2^2}}. \]  

(2)

It is evident that the \( a_i \) are only defined within an arbitrary constant, since if the equation \( a_0 + a_1X_{i,1} + a_2X_{i,2} = 0 \) defines a line then so does the product \( c(a_0 + a_1X_{i,1} + a_2X_{i,2}) = 0 \) for any multiplicative constant \( c \). Therefore to make the computation in Equation (2) easier we will establish for the line defining the separating hyperplane the constraints

\[ Y_i(a_0 + a_1X_{i,1} + a_2X_{i,2}) \geq 1, \]  

with equality holding on the lines passing through the support vectors (i.e., the “curbs” of the “street”). Inequality (3) specifies that \( a_0 + a_1X_{i,1} + a_2X_{i,2} = 1 \) on the geometric margin (dashed line) on the positive (green) side of the “street” and \( a_0 + a_1X_{i,1} + a_2X_{i,2} = -1 \) on the margin on the negative side. Therefore, on the margin the numerator in Equation (2) is equal to 1 and the equation becomes \( d_i = (a_1^2 + a_2^2)^{-1/2} \). In order to maximize this distance, we must therefore solve the following minimization problem:
minimize $a_1^2 + a_2^2$ \hspace{1cm} (4a)

subject to the constraints

$$Y_i(a_0 + a_1 X_{i,1} + a_2 X_{i,2}) \geq 1, \quad i = 1, \ldots, n.$$ \hspace{1cm} (4b)

This is a quadratic programming problem, and the solution is called a support vector classifier (there is one further step to reach the definition of a support vector machine). It goes without saying that there is nothing special about $p = 2$ (recall that $p$ is the number of predictors $X$ used in the classification, and as such is the dimension of the vector space defined by these predictors), and that everything we have done generalizes to higher values of $p$.

Although the general form of Expression (4a) is that of a quadratic programming problem, the constraints given in Expression (4b) do not fit directly into the classical quadratic programming formalism, and are generally approached using a technique called Lagrange multipliers, discussed in SDA2, Appendix A.4. Solving the problem this way would take too much explanation to be worth the effort, so instead we will simply compute the distances $d_1$ and $d_2$ in Fig. 3 and see which is larger. In Figs. 3a the “curbs” of the optimal separating hyperplane “street” pass through the points $X_2$ and $X_{13}$ on the negative (red) side and through $X_5$ on the positive (green) side. This leads to the three equations

$$a_0 + a_1 X_{2,1} + a_2 X_{2,2} = -1$$
$$a_0 + a_1 X_{13,1} + a_2 X_{13,2} = -1.$$ 
$$a_0 + a_1 X_{5,1} + a_2 X_{5,2} = 1$$ \hspace{1cm} (5)

We can express this in matrix formalism as

$$\begin{bmatrix} 1 & X_{2,1} & X_{2,2} \\ 1 & X_{13,1} & X_{13,2} \\ 1 & X_{5,1} & X_{5,2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}$$ \hspace{1cm} (6)

which has the from $AX = Y$. The R function `solve()` solves matrix equations such as this. Here is the code.

```r
> A <- with(XY, matrix(c(1,1,1,X1[2],X1[13],X1[5], + X2[2],X2[13],X2[5]), nrow = 3, ncol = 3))
> Y <- c(-1,-1,1)
> print(a <- solve(A,Y))
[1]  0.1610459 -8.6178258  9.6437668
```
[1] 1
[1] 0.07732004

The values of the vector a computed in the code above, which correspond to \(a_0, a_1,\) and \(a_2\) in Equation (6), are then used to compute the value of \(d_1\) as well as the slope and intercepts to draw the lines in Fig. 3a. Here is the first bit of code (remember that \(a[1]\) corresponds to \(a_0\) etc.)

> #Draw Fig. 3a
> with(XY, plot(X1, X2, xlim = c(0,1), ylim = c(0,1), pch = 16, col = color))
> legend(0.0, 1.0, c("Y = 1", "Y = -1"), pch = 16, col = c("green", "red"))
> lines(c(0,1), c(bred, (bred+m)), lty = 3)

In the same way, we now compute \(d_2\) in Fig. 3b and compare the two.

> A <- with(XY, matrix(c(1,1,1,X1[2],X1[15],X1[5],
+ X2[2],X2[15],X2[5]), nrow = 3, ncol = 3))
> Y <- c(-1,1,1)
> print(a <- solve(A,Y))
[1] 0.3870426 -8.8375367  8.9638607
[1] 1
[1] 0.07944196

Since \(d_2\) is larger, this defines our optimal separating hyperplane. Using Equation (2) we can compute the distance of any data point from the separating hyperplane and we can use this as an indication of confidence in the correctness of the classification.

It is clear from Figs. 2 and 3 that the data in this example are very special in the sense that they can be completely separated by a straight line (a separating hyperplane). Data like these are said to be linearly separable, and real data such as those displayed in Fig. 1 clearly do not fit this criterion. In the next section we begin the process of generalization by maintaining the separating hyperplane but allowing some points to “cross the line”. Before moving on, however, we make one final observation. Using the notation of matrix theory, we can write

\[
a_1^2 + a_2^2 = \begin{bmatrix} a_1 & a_2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = a'a ,
\]

where the prime denotes the transpose. The quantity on the right-hand side is called the inner product of \(a\) with itself, and thus Expression (4a) can be written

\[
\text{minimize } a'a ,
\]

(7)
so that the quantity to be minimized depends only on the inner product of $a$ with itself, and not on the specific components of $a$ by themselves. The property that the support vectors depend only on an inner product turns out to be true of support vector machines in general, and it is important in solving the optimization problems they engender.

4. The incompletely separated case

The example in Sections 2 and 3 is completely separable, that is, we can construct a hyperplane that completely separates the two classes. Let’s now add five more data records to the original twenty, and put them on the wrong side of the separating hyperplane. In order to put them exactly where I wanted them, I fiddled with them a bit. Here is the code.

```r
> set.seed(3)
> XYn <- data.frame(X1 = runif(5), X2 = runif(5))
> XYn$color <- "green"
> XYn$color[with(XYn, which(X2 > X1))] <- "red"
> XYn$Y <- -1
> XYn$Y[with(XYn, which(X2 > X1))] <- -1
> XY2 <- merge(XY, XYn, all = TRUE)
```

I had to adjust two of the points from their original position to get them exactly where I wanted them. The function `merge()`, used in the third from the last line to merge the two data frames, reorders the data records, so I had to make these adjustments after the merger. Fig. 4 shows the result.
Figure 4. Augmented version of the data in Fig. 2 showing some points that are either inside the geometric margin or on the wrong side of the separating hyperplane. The point $X_{11}$ is on the correct side of the hyperplane but inside the margin, and the remaining identified points are on the wrong side and increasingly distant from the hyperplane.

It is evident from Fig. 4 that there is no straight line that will completely separate the regions so that one contains all green points and one contains all red points. We must therefore change our strategy to one where we accept that some points will be misclassified, and try to minimize the number of misclassified points. The way that this is accomplished is to associate with each data point $X_i$ a non-negative slack variable denoted $s_i$ whose value is zero if the point $X_i$ is on the correct side of the geometric margin. If the point is within the margin or on the wrong side of the separating hyperplane then we will define $s_i$ as the distance from point $X_i$ to the separating hyperplane. We then require that the sum of these slack variables be less than some preassigned value $M$.

Our new objective is then to choose $a_0$, $a_1$, and $a_2$ to

$$\text{minimize } a'a$$

subject to the constraints

$$Y_i (a_0 + a_1 X_{i,1} + a_2 X_{i,2}) \geq 1 - s_i, \text{ } i = 1, ..., p,$$

$$s_i \geq 0, \sum_{i=1}^p s_i \leq M,$$

for a predetermined constant $M$. The constraints (9b) are sometimes called “soft” constraints. The larger the value of $M$, the more willing we are to accept incorrectly classified data records. Thus $M$ represents the amount of misclassification that we can accept. If $M$ is low then we are only willing to accept a small amount, and if $M$ is larger than we are willing to accept a greater level of misclassification. In practice, one computes the solution with various values of $M$ and compares the results to determine the preferred one.

Finding the optimal separating hyperplane for the data of Fig. 4 is too difficult to do graphically, so we will do it using R. The R function $\text{svm()}$ is used to compute support vector machines. It is part of the package $\text{e1071}$ (Meyer et al., 2017). Before applying it to the data set $XY_2$ of this section, we will use $\text{svm()}$ to compute the optimal separating hyperplane for the data set $XY$ of
Section 3 as a check. Like many R functions, svm() has lots of optional arguments, but we will take a minimalist approach at first.

```r
> library(e1071)
> XY$YF <- as.factor(XY$Y)
> svm.model <- svm(YF ~ X1 + X2, data = XY, kernel = "linear", cost = 1000)
```

Like the function rpart() (Therneau et al., 2015) described in Chapter 9 of SDA2, svm() by default computes a regression solution if the response variable is numeric and a classification if it is a factor. Since our Y is numeric, svm() would compute a regression solution if we used the original Y. One can override this by inserting the argument `type = "C-classification"`, but this can cause other problems down the line, so I feel that it is better to create a new data field that is the factor form of the original. In this case we call the new data field YF. The first two arguments of the function svm() are analogous to those of functions like lm() for linear regression (SDA2, Chapter 8) and rpart() for recursive partitioning. Familiarity with at least one of these functions will be assumed. The value "linear" of the third argument, kernel, basically tells svm() to use the inner product $a'a$ of Section 2, as opposed to something more complicated (the kernel will be defined in Section 5). The final argument, cost, is related to the value of $M$ in the constraints (9b), but it represents the cost of misclassification, so that higher values of cost correspond to lower values of $M$.

The function svm() returns an svm object that has a lot of information (use `str(svm.model)` and `summary(svm.model)` to see some of it). For our purposes in this section, the two most important objects are the computed support vectors and the classification.

```r
> svm.model$SV
  X1     X2
5  1.2354528 -0.7496833
15  0.7503001  1.2823212
2  -0.6397475 -0.9494843
```

The support vectors are correctly specified. Their component values have been scaled; this can be avoided if desired by adding the argument `scale = FALSE`. Similarly to the function predict() that works with objects from linear models, the function svm.predict() computes the class labels for objects computed by svm(). Because of polymorphism (SDA2, Section 2.5) predict() recognizes the appropriate implementation by the class of its first argument.

```r
> print(svm.pred <- predict(svm.model, XY))
   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
1 -1 -1  1 -1  1 -1  1  1  1 -1 -1  1  1  1 -1  1  1  1 -1  1
Levels: -1 1
```
> which(XY$YF != svm.pred)
integer(0)

The R workhorse function `plot()` also has a polymorphic implementation for `svm` objects. In addition to the `svm` object, one must specify the data source and the axes to plot (in the rather unconventional fashion `ordinate ~ abscissa`). The optional argument `grid` controls the resolution of the plot. Here is the code.

> plot(svm.model, data = XY, X2 ~ X1, grid = 200) #Fig. 5a

Fig. 5a shows the resulting plot. The X’s are the support vectors and the O’s are all the other points. The optimal separating hyperplane and the support vectors are the same as those shown in Fig. 3b.

We set the `cost` value quite high in this computation. The default value of `cost` is 1. Let’s see what happens when we use this value.

> svm.model <- svm(YF ~ X1 + X2, data = XY, kernel = "linear")
> svm.pred <- predict(svm.model, XY)
> svm.model$SV
   X1  X2
 3  0.06181455 0.64495517
 5 -1.23545283 -0.74968333
10 -1.72439691 -0.48440368
15  0.75030013 1.28232124
 2 -0.63974748 -0.94948430
 8  0.36918583 -0.33190354
13  0.46084404 0.07131533
> which(as.factor(XY$Y) != svm.pred)
integer(0)
> plot(svm.model, data = XY, X2 ~ X1, grid = 200) #Fig. 5b

Figure 5. Output of the call to the function `plot()`, a) with `cost` set to 1000; b) with `cost` set to 1. The X’s denote support vectors and the O’s denote other data points.
The model still predicts all of the class labels correctly, but there are a lot more support vectors. Also, Fig. 5b shows that the optimal separating hyperplane has moved. To understand what is going on here we need to introduce a second concept of the margin, called the *functional margin*. Returning to the equations we started with, but using our notation for the inner product, we can rewrite Equation (2) as

$$d_i = \frac{Y_i (a_0 + a'X_i)}{(a'a)^{1/2}},$$

where we have used the fact that $Y_i = \pm 1$ on the margin. The quantity in the numerator, $Y_i (a_0 + a'X_i)$, is called the *functional margin*, and we can see that because $Y_i (a_0 + a'X_i) > 0$ on either side of the separating hyperplane, finding the optimal separating hyperplane is equivalent to maximizing the functional margin, normalized by $a$, subject to constraints of the form of Expression (9b). In this context, if we define the kernel $K(u,v)$ as

$$K(u,v) = u'v,$$

then the functional margin can be written as $Y_i (a_0 + K(a,X_i))$. Note that from Equation (11) the kernel $K(a,X)$ is linear in $X$. This is the source of the argument `kernel = "linear"` in the calls we have made to `svm()`.

Now we move on to the effect of the `cost = 1` argument represented by Expressions (9) and the result shown in Fig. 5b. In brief, when we move from the optimization problem characterized by Expressions (4) to that characterized by Expressions (9), and we allow some of the slack variables in Expression (9b) to be positive, we permit the optimal separating hyperplane to move in such a way that it can pick up support vectors that are inside the margin. Comparing Figs. (5a) and (5b) indicates that in this case more support vectors have been picked up in the blue region, and this has caused the optimal separating hyperplane to rotate in the direction of these vectors. In general, changing the value of the cost function causes the optimal separating hyperplane to move as support vectors defining that hyperplane are added or removed.

Now we can move on to the data set $XY2$ shown in Fig. 4, which is not linearly separable. Here is the code I used initially to compute the classification.

```r
> XY2$YF <- as.factor(XY2$Y)
> svm.model <- svm(YF ~ X1 + X2, data = XY2, kernel = "linear",
+   cost = 1000, scale = FALSE)
```
> svm.model$SV
X1        X2
 1 0.06178627 0.3403490
 4 0.20168193 0.2672207
13 0.57285336 0.6516738
19 0.76984142 0.8273733
 2 0.16804153 0.3043941
 7 0.32773432 0.5776099
11 0.38494235 0.4046009
14 0.60210067 0.6309793

> svm.pred <- predict(svm.model, XY2)
> which(XY2$YF != svm.pred)
[1] 4 7

> plot(svm.model, data = XY2, X2 ~ X1, grid = 200, xlim = c(0,1),
+     ylim = c(0,1)) #Fig. 6a

The points $X_4$ and $X_7$ are incorrectly classified. I set the argument `scale = FALSE` in the call to `svm()` and extended the axis limits in the call to `plot()` so that the points could be more easily visualized in Fig. 6a. Again, the colors of the regions specify the classification, support vectors are represented by an $X$, other vectors are represented by an $O$, vectors colored red have the value 1, and vectors colored black have the value -1.

![SVM classification plot](image)

(a) ![SVM classification plot](image)
(b)

Fig. 6. Classification plots for the data set XY2; a) cost = 1000, b) cost = 100

Remember that the support vectors should be thought of as supporting the kernel defined by Equation (11), because these are the vectors that have nonzero slack variables and therefore they contribute to the functional margin $Y_i(a_0 + K(a, X_i))$.

Let’s see what happens when we reduce the value of `cost` from 1000 to 100.
<pre>&gt; svm.model$SV
   X1        X2
1  0.06178627 0.3403490
4  0.20168193 0.2672207
5  0.20597457 0.4820801
13 0.57285336 0.6516738
18 0.71761851 0.7942399
2  0.16804153 0.3043941
7  0.32773432 0.5776099
11 0.38494235 0.4046009
14 0.60210067 0.6309793
17 0.68702285 0.4935413

&gt; svm.pred <- predict(svm.model, XY2)
&gt; which(as.factor(XY2$YF) != svm.pred)
[1]  4  7 14

As shown in Fig. 6b, the optimal separating hyperplane shifts slightly, some new support vectors are added, and one more point is misclassified.

5. The bias-variance tradeoff

You might be wondering why one would ever not set the value of cost to be a high one, since it results in fewer misclassified points. The answer is that the value of cost, or equivalently of M in expression (9b), is directly connected to the bias-variance tradeoff as it relates to support vector machines. If you are not familiar with the bias-variance tradeoff, it is discussed in SDA2, Section 8.2, and in many other sources as well (e.g., James et al., 2013). The basic idea is that any curve fitting or classification problem constructs a model of the data based on statistical properties of these data. For example, in linear regression one constructs a regression model. In support vector machine classification the model is the classification regions, represented, for example, by the different colored regions in Fig. 6. Points in the brown region are classified as $Y = 1$, and points in the yellow region are classified as $Y = -1$.

In this context, the (fallacious) logic of greatly increasing the cost value is that a higher value of this argument will lead to a more accurate classification. The fallacy is that, while this is true for the data used to fit the SVM model, it means that when the model is used to predict the values of data records not used to construct the model, the prediction might not be as accurate because the model is too tightly fit to the training data. This is the consequence of the bias-variance tradeoff. In the case of linear regression there is a very simple decomposition of the total error of the form $(Total\ Error)^2 = Bias^2 + Variance^2$ (see SDA2, Section 8.2). There is no corresponding simple formula with SVMs (Valenti and Dietterich, 2004), but the interpretation is the same. In case you
are familiar with linear regression or have read SDA2 Section 8.2, let me briefly allude to that. Suppose we are trying to use linear regression to fit data that does not adhere to a straight line very well. In this situation, as one adds higher order terms to the model the prediction of the data can become increasingly accurate. Conversely, if there is only, say, a first order predictor the model might not be very accurate at all. This inaccuracy is the bias, and one says that the model underfits the data. On the other hand, as more terms variables are added to the regression model, the model begins to overfit the data. A consequence of this overfitting is that the variance of the predicted Y values increases. This is an expression of the bias-variance tradeoff for regression models. A consequence of overfitting the data is that when the model is used to predict the values of data not used in its creation, the prediction may be very poor.

Returning to the case of support vector machines, consider first the model used to create Fig. 6a, which has a high value of cost. This model has only three support vectors. These are the only data that are used to construct the model. Although it is not a correct analogy, as a mnemonic you can think of this as being analogous to a linear regression with only a first order term. In the case of the support vector machine, the margin is relatively wide, and inaccurate predictions generally result from the model being a relatively poor fit to the training data. The model used to create Fig 6b, on the other hand, has a greater number of support vectors and as a consequence a narrower margin. This is somewhat analogous to a linear regression that includes more high order terms. Inaccurate predictions in this model tend to come from data values not well represented by the training data set.


The support vector classifier as we have developed it in the previous sections will not work very well for data such as shown in Fig. 7. This data set is generated with the following code.

```r
> XY <- data.frame(X1 = runif(20), X2 = runif(20))
> XY$Y <- 1
> XY$Y[which(abs(XY$X2 - 0.5) < 0.25)] <- -1
> XY$color <- "green"
> XY$color[with(XY, which(Y < 0))] <- "red"
```
Figure 7. A data set that could not be effectively classified with a linear kernel.

This data set is analogous to one that in linear regression would be fit by including an $X_2^2$ term in the regression model, and the approach in SVM is also analogous. Specifically, we can generalize the kernel $K(u,v) = u'v = (u_1v_1 + u_2v_2)$ given in Equation (11) by using a polynomial kernel. We will write the general form for a polynomial kernel as

$$K(u,v) = (\gamma u'v + c_0)^d$$

(12)

Here $d$ is the degree of the polynomial and $\gamma$ and $c_0$ are coefficients. The symbols are chosen to approximately match the corresponding arguments in the function $\text{svm}()$. To classify the data in Fig. 7 we will choose the polynomial degree $d = 2$. This means that our classification will be based on

$$K(u,v) = \left( \gamma \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + c_0 \right)^2$$

(13)

The problem of the classification is then to maximize the functional margin defined by the kernel of Equation (13) subject to the “soft” constraints of Expression (9b). From the discussion in Section 5, this involves working with the functional margin $Y_i \left( a_0 + K(a_i, X_i) \right)$, which in this case is $Y_i \left( a_0 + \gamma ((a_{i,1}X_{i,1} + a_{i,2}X_{i,2}) + c_0)^2 \right)$. This functional margin contains quadratic terms in the $X_i$, which permit nonlinear margins. While the foregoing is a very rough and ready explanation, it
does provide a workable idea of what is going on. For those who are interested, further details are provided in the references discussed in Section 8.

In the example shown in Fig. 7, it is clear that an $X_2^2$ term will come in very handy. Fig. 8a shows the classification regions using a linear kernel. The classification has a 40% misclassification rate. The classification using a quadratic polynomial kernel (Fig. 8b) has a misclassification rate of 0. Increasing the polynomial degree to 3 (Fig. 8c) actually increases the misclassification rate to 25%. The other type of kernel function that we will consider is the radial basis kernel. In \texttt{svm()} this function is expressed as (Meyer et al, 2017)

$$K(u, v) = \exp(-\gamma|u - v|^2),$$  \hspace{1cm} (13)

where $\exp()$ is the exponential function. Hsu et al. (2016) recommend the radial basis function as the logical first choice when carrying out a real-world analysis, pointing out that other kernel functions can be considered approximately as special cases of this function. Indeed, in our example the radial basis function works as well as the quadratic polynomial, with a 0 misclassification rate (Fig. 8d).

Figure 8. Classification regions for (a) linear; (b) polynomial degree 2; (c) polynomial degree 3; (d) radial basis functions.
Next let’s examine real data, the KNN blue oak training and test data. The training data were loaded into the system at the beginning and generated Fig. 1. We will start by using the normalized data $MAT.n$ and $Precip.n$.

\[
\text{oaks.Train.n} \leftarrow \text{data.frame}(\text{QUDO = as.factor(Set2.Train$QUDO)}, \right.
\text{MAT.n = Set2.Train$MAT.n, Precip.n = Set2.Train$Precip.n}) \\
\text{oaks.svm.n} \leftarrow \text{svm(QUADO ~ MAT.n + Precip.n, data = oaks.Train.n,} \\
\text{kernel = "radial", cost = 1000, scale = FALSE)} \\
\text{plot(oaks.svm.n, data = oaks.Train.n, Precip.n ~ MAT.n, grid = 200,} \\
\text{xlim = c(\text{round}(2*\text{min(oaks.Train.n$MAT.n))/2)}, \\
\text{ylim = c(\text{round}(2*\text{max(oaks.Train.n$Precip.n))/2)},} \\
\text{oaks.pred.n} \leftarrow \text{predict(oaks.svm.n, oaks.Train.n)} \\
\text{length(which(oaks.pred.n != oaks.Train.n$QUDO))/ length(oaks.Train.n$QUDO)} \\
\text{[1] 0.06}
\]

![SVM classification plots](image)

Figure 9. Classification regions generated by $\text{svm()}$ for the Set2.U $QUADO$ training data: X’s represent support vectors, O’s represent other vectors, black represents $QUADO = 0$, red represents $QUADO = 1$. a) normalized training data, $\text{scale = FALSE}$; b) non-normalized data, $\text{scale = TRUE}$, c) non-normalized data, $\text{scale = FALSE}$.

Fig. 9a shows the results of the application of $\text{svm()}$ to the normalized training data, which produces a 12% misclassification rate. Hsu et al. (2016) point out the importance of scaling the data, primarily because it avoids attributes with larger values dominating those with smaller values. We can see this with the $QUADO$ data set. Fig. 9b shows the results of applying $\text{svm()}$ to the non-normalized, scaled data. Here is the result.

\[
\text{oaks.pred.s} \leftarrow \text{predict(oaks.svm.s, oaks.Train)} \\
\text{length(which(oaks.pred.s != oaks.Train$QUDO))/ length(oaks.Train$QUDO)} \\
\text{[1] 0.06}
\]
The algorithm produces very similar, but actually slightly better, results. Fig. 9c shows the results of applying the algorithm to non-normalized, unscaled data.

```r
> oaks.pred.u <- predict(oaks.svm.u, oaks.Train)
> length(which(oaks.pred.u != oaks.Train$QUDO))/ length(oaks.Train$QUDO)
[1] 0
```

As can be seen in the figure, the Precip data, which has a magnitude about fifty times bigger than that of MAT, dominates the result. The same result is obtained if the prediction is made based on Precip alone (try it!). As a result, the prediction is very accurate, because it simply classifies everything according to the unique attribute that defines it, snaking around the precipitation data. In Exercise (2) you are asked to test this predictive model on the KNN test data set.

### 7. Application to a real problem

Now that we know enough about support vector machines to begin thinking about using them, let’s see how the method can be applied to a real problem. The problem is to classify the Sierra Nevada and Coast Range of California for the presence or absence of blue oaks, and to draw a map showing the boundaries between these classification regions. Because the regions are geographically distinct, we can construct the two classification boundaries separately. We will start with the Sierra Nevada, reading the data from the created folder on the SDA2 website. As a check on its robustness, we will use the SVM model generated by the Sierra Nevada data for both the Sierra Nevada and the Coast Range, and in the Exercises compare the result with an SVM model generated using Coast Range data.

```r
> library(sf)
> data.Set2S.sf <- st_read("created\set2sierra.shp")
```

The object `data.Set2S.sf` is an sf object, and the function `svm()` does not accept these, so the first thing we will do is to strip the geometry, so that we have a simple data frame, and then generate the factor valued attribute `QUDOF` for use in `svm()`.

```r
> data.Set2S <- st_set_geometry(data.Set2S.sf, NULL)
> class(data.Set2S)
[1] "data.frame"
> data.Set2S$QUDOF <- as.factor(data.Set2S$QUDO)
```

The next step is to generate the predictor. To accomplish this we will carry out tenfold cross-validation. This is discussed in SDA2 Section 6.3.3 and also by Hastie et al. (2009) and James et al. (2013), and is used in our Additional Topic on the K nearest neighbor method. In that topic
we created a homemade function `cross.validate()` to carry out the operation because we wanted to use a homemade K nearest neighbor function. In the case of SVM, however, we are using the function `svm()` from the `e1071` package. The function `tune()` from the same package does tenfold cross validation, and we will use it. The name “tune” refers to the parameter `cost`, which is considered as a tuning parameter, but we also want to explore different predictor attributes $X_i$. The Additional Topic on the comparison of supervised classification methods contains a discussion of variable selection for SVM predictor data. Here we use a sort of “ecologically informed stepwise selection” to generate a reasonable model for the classification region, and simultaneously use `tune()` to tune for the best value of `cost`. If this were a real project instead of a demonstration, much more care would be given to generating the best possible model, but in this case the only goal is to demonstrate the procedure, so less care will be taken.

Here is an edited version of three of the steps of the “stepwise selection” process, the first one, an intermediate one, and the best one.

```R
> summary(tune(svm , QUDOF ~ CoastDist,
+    data = data.Set2S, kernel = "radial",
+    ranges =list(cost=c(1, 10, 25, 50, 100, 1000))))

Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
  cost
  50
- best performance: 0.1900231

> summary(tune(svm , QUDOF ~ CoastDist + Elevation + Precip + JuMin,
+    data = data.Set2S, kernel = "radial",
+    ranges =list(cost=c(1, 10, 25, 50, 100, 1000))))

Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
  cost
  1000
- best performance: 0.133481

> summary(tune(svm , QUDOF ~ CoastDist + Elevation + Precip
+    + JuMin + ET + Permeab,
+    data = data.Set2S, kernel = "radial",
+    ranges =list(cost=c(1, 10, 25, 50, 100, 1000))))

Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
  cost
  10
- best performance: 0.1215973
```
Because cross validation involves random numbers, your results will be slightly different from mine.

We now use the selected attributes, *CoastDist, Elevation, Precip, JuMin, ET* and *Permeab*, as our predictors to create an SVM model.

```r
sierra.svm <- svm(QUDOF ~ CoastDist + Elevation + Precip + JuMin + ET + Permeab, data = data.Set2S, kernel = "radial", cost = 10, scale = TRUE)
```

The next step is to draw a map showing the boundary between the predicted *QUDO* and non-*QUDO* regions. I did this using code lifted from SDA2 Sections 1.1 and 12.6.1. In the interest of saving space I will not duplicate the description here, but rather refer to the original text. The mapping process is then repeated for the Coast Range, using the same SVM model for both the Coast Range and the Sierra Nevada (see Exercise 3). The resulting boundaries are shown in Fig. 10. Fig. 11 shows the actual distribution of blue oaks. This figure is taken from the Additional Topic on map data exploration with leaflet and mapview. This little demonstration is not meant to be an actual analysis, but it does indicate the direction that such an analysis could take. A second step in this direction is taken in Exercise 4.
Figure 10. Predicted blue oak suitability boundaries. Both predictions are made using the Sierra SVM model.

Figure 11. Actual blue oak distribution

8. Further reading

The main source for this topic discussion is James et al. (2013). Chang and Lin (2018) provide pointers to a wealth of very practical information, especially Hsu et al. (2016). Meyer (2018) is another excellent source that focuses on the use of the `svm()` function. Although the discussion here considers only dichotomous supervised classification problems, SVM can also be applied to classification problems involving more than two classes, as well as to continuous data, where it is called support vector regression. SVM is applied to more than two classes in the Additional Topic on the comparison of supervised classification methods. The three sources cited here provide a good place to begin for each of these topics. Although it is certainly not obvious, there turns out to be a deep theoretical connection between the SVM method and logistic regression (see SDA2 Chapter 8 for a discussion of logistic regression). This connection is explored by James et al. (2013, Section 9.5).
9. Exercises

1) The package *kernlab* contains the function *ksvm()* that provides an alternative to *svm()* for the computation of support vectors. Reanalyze the data sets XY and XY2 of Sections 3 and 4 using this function.

2) Repeat the analysis of Section 6 on the KNN test data.

3) The Coast Range boundary in Fig. 10 was created using the SVM model for the Sierra Nevada. Repeat the stepwise selection method of Section 7 using the Coast Range data and draw a map comparing boundaries created by the two models. *Note:* I found it very difficult to generate a decent map and in the end resorted to trial and error in selection attributes.

4) Use the *leaflet* package described in the Additional Topic on map data exploration with *leaflet and mapview* to draw a map of the Sierra Nevada in which the correctly identified blue oak locations are shown in blue and the incorrectly identified locations are shown in red.

10. References


