# STAT 621 Lecture Notes
# Support Vector Machines

The Support Vector Machine (SVM) is another method for classification in supervised learning. The objective is the same as for logistic regression, LDA, etc. That is, we want to find a rule that tells us class a response should fall into, based on the values of a set of features measured on that response. Our discussion of SVM follows Chapter 9 of the text *An Introduction to Statistical Learning* (AISL), by G. James, D. Witten, T. Hastie and R. Tibshirani. Figures taken from this text are with permission from the authors.

## Separating Hyperplanes

Suppose we have observations $(Y_i, \boldsymbol{x}_i)$ where $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})$ is a vector of feature or predictor values, and $Y_i$ is a *binary* response. We discuss the method in the context of binary responses, but we will try to generalize to multinomial responses at the end. Specifically, we will assume that $Y_i \in \{-1, 1\}$. First a quick definition.

A **Hyperplane:** In $p$ dimensions, a hyperplane is a flat subspace with dimension $p - 1$.

- So what would a hyperplane be if $p = 2$? How would we express it? $p = 3$?

So in $p$ dimensions we can express the hyperplane as all points $\boldsymbol{X}$ such that

$$f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p = 0$$

Moreover, for any point $\boldsymbol{x}^*$, the value of this linear combination indicates where this point lies in relation to the hyperplane. That is, Hyperplanes can be used for classification in the following manner. Given training data $(Y_i, \boldsymbol{x}_i)$ for $i = 1, \ldots, n$, suppose we are able to find a hyperplane that perfectly separates the training data into two response classes. That is, for all points on one side of the plane, $Y_i = 1$, and for all points on the other side, $Y_i = -1$. More specifically,

Note what this implies about the product of $Y_i$ with $f(\boldsymbol{x}_i)$.

This separating hyperplane then gives us a classification rule. Given a new $\boldsymbol{x}^*$, first one computes $f(x^*) = \beta_0 + \beta_1 x_1^*, \ldots + \beta_p x_p^*$. Then we'd classifiy $Y^*$ according to

In addition, the magnitude of $f(x^*)$ informs us about how far $x^*$ falls from that classification boundary (in a moment we'll impose some conditions so that it represents the perpindicular distance). The closer $f(x^*)$ is to zero, the closer it is to the hyperplane. The farther way a point is from the boundary, in other words the greater the magnitude of $f(\boldsymbol{x}^*)$, the more confident we probably are in that classification.

## Maximal Margin Classifier

It's worth noting that if there exists a separating hyperplane, so that responses in the training set can be perfectly partitioned, then there also exists an infinite number of separating hyperplanes. The *Maximal Margin Classifier* uses the classification rule that is defined by the separating hyperplane that is *farthest* from the training data. We do the following.

- Compute the $\perp$ distance from each training observation to a given separating hyperplane.
- The smallest $\perp$ distance out of the $n$ points is called the *margin*.
- The maximal margin hyperplane (MMH) is the one with the largest margin.
- The maximal margin classifier (MMC) is the rule that classifies $Y$ according to the MMH.

The idea is displayed in the following figure, taken from AISL.

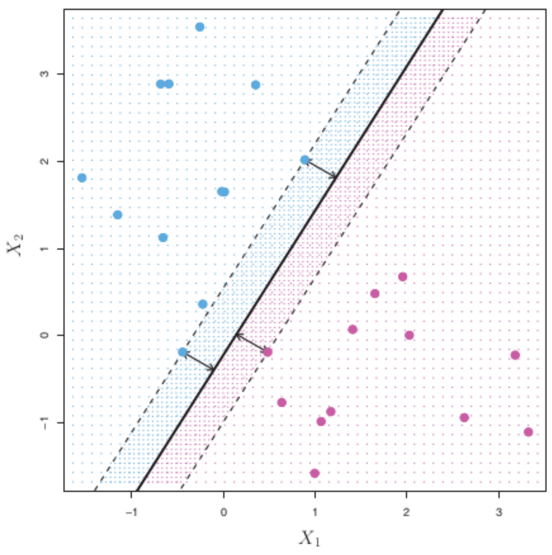


**FIGURE 9.3.** *There are two classes of observations, shown in blue and in purple. The maximal margin hyperplane is shown as a solid line. The margin is the distance from the solid line to either of the dashed lines. The two blue points and the purple point that lie on the dashed lines are the support vectors, and the distance from those points to the hyperplane is indicated by arrows. The purple and blue grid indicates the decision rule made by a classifier based on this separating hyperplane.*

In the above figure, there are three points that are the same distance from the MMH. Each of these points is actually a $p$-dimensional vector, in this case $(x_1, x_2)$. These points are called the *support vectors*. A little explanation:

There is still a big issue here. Namely assuming that a separating hyperplane exists, how do we actually find the MMH? We need to solve for the coefficients $\beta_0, \ldots, \beta_p$ that maximize $M=$ the margin width. This is an optimization problem. Let's write it out below.

The MMC seems like a reasonable tool, provided that a separating hyperplane does exist for a data set. This isn't always the case however. Sketch an example:

Even if a separating hyperplane does exist, a classification procedure based on this can be problematic. These tend to be very sensitive to certain observations, and may also tend to overfit the training set. The figure below presents one of these situations.

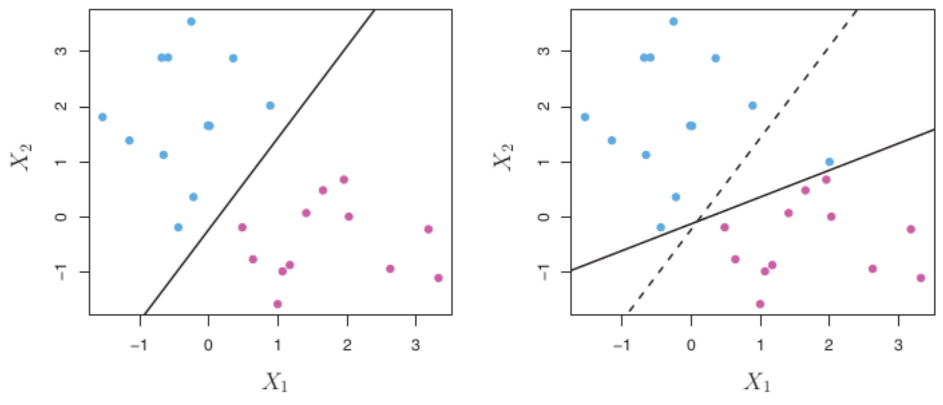


**FIGURE 9.5.** Left: *Two classes of observations are shown in blue and in purple, along with the maximal margin hyperplane.* Right: *An additional blue observation has been added, leading to a dramatic shift in the maximal margin hyperplane shown as a solid line. The dashed line indicates the maximal margin hyperplane that was obtained in the absence of this additional point.*

## Support Vector Classifiers

The Support Vector Classifier (SVC) tries to overcome some of these problems by basing classification on hyperplanes that may not fit perfectly. That is not all of the $Y$'s in the training set must be classified correctly. The SVC allows some of the data points to fall inside the margin, or even on the wrong side of the hyperplane.

Here the optimization problem is as follows. Remember $M$ is the width of the margin and let $C$ be a constant. Then we find

$$\max_{\beta_0,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n} M$$

subject to

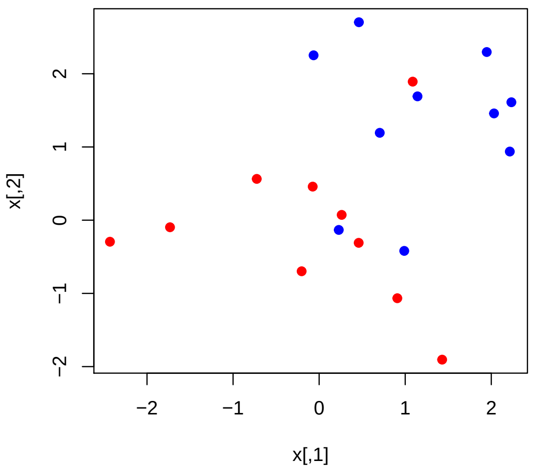
- $\sum_{j=1}^{p} \beta_j^2 = 1$
- $Y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i)$
- $\epsilon_i \geq 0$ and $\sum \epsilon_i \leq C$

Some details: $\epsilon$'s, $C$, Bias-Variance tradeoff...

Once the optimization problem is solved, classification proceeds as before. Given a new value $\boldsymbol{x}^*$, $Y^*$ is classified as $\pm 1$ by the sign of $f(x^*)$. For the SVC, only those points on or in violation of the margin will affect the hyperplane, and therefore the classifier. Such points are again called *support vectors*. No other points influence the classifier. For this reason, SVC is seen as more robust than other classification methods like logistic regression and LDA.

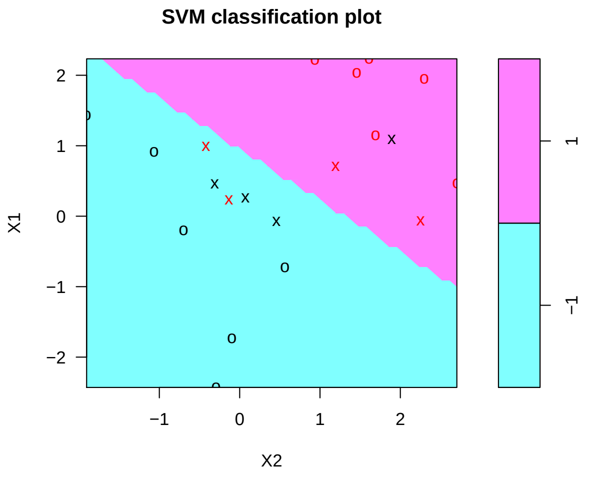
**Example:** Let's try out the SVC on some simulated data. I'll use the `svm` function in the `e1071` library. First simulate some data with $p$=2 and plot.

```
x = matrix(rnorm(40), 20, 2)
y = rep(c(-1, 1), c(10, 10))
x[y == 1,] = x[y == 1,] + 1
plot(x, col = y + 3, pch = 19)

dat=data.frame(x, y = as.factor(y)) # make y factor
head(dat)
              X1         X2  y
    1 -0.72489861  0.5637505 -1
    2  1.08690703  1.8920839 -1
    3 -0.20332471 -0.6978732 -1
         ......  ETC.

library(e1071)
```



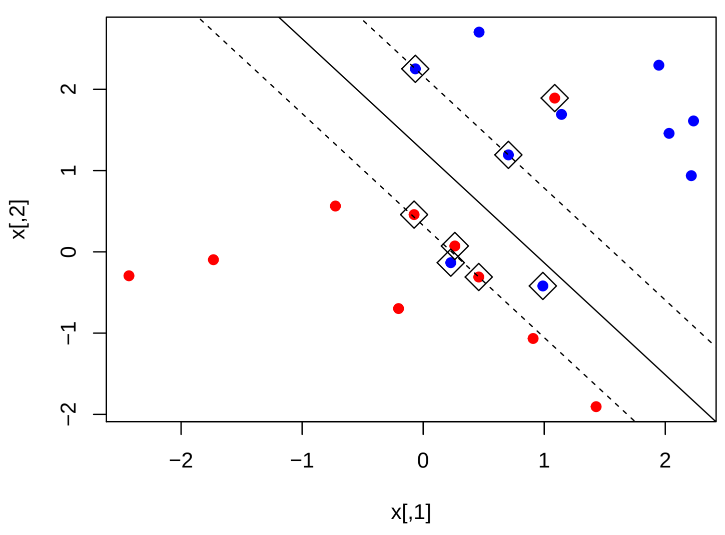
Now fit the model and look at some output.

```
> svmfit=svm(y~., data=dat, kernel="linear", cost=10, scale=FALSE)  #linear gives svc model, scale F don't std. X's
> summary(svmfit)
Call: svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
\Number of Support Vectors:  8
 ( 4 4 )
Number of Classes:  2
Levels:
 -1 1

> svmfit$index          # indices of support vectors
[1]  2  6  8  9 11 12 16 17

> plot(svmfit, dat)     # aargh -- can't change the colors!
```



A better plot. This will only work with $p = 2$

```
# plot bdry and id the sv's
beta = drop(t(svmfit$coefs)%*%x[svmfit$index,])  # get coeffs: b0 + b1*x1 + b2*x2 = 0
beta0 = svmfit$rho
plot(x, col = y + 3, pch = 19)
abline(beta0 / beta[2], -beta[1] / beta[2])   # plot boundary and margins
abline((beta0 - 1) / beta[2], -beta[1] / beta[2], lty = 2)
abline((beta0 + 1) / beta[2], -beta[1] / beta[2], lty = 2)
points(dat[svmfit$index,1],dat[svmfit$index,2], pch=5, cex=2)   # the sv's
```



Finally we can use the fitted model to predict some new responses, or the original data.

```
> newdat=data.frame(X1=c(-0.5,0,1.5),X2=c(0,-2,2.5))
> predict(svmfit, newdat)
     1  2  3
    -1 -1  1


> table(pred=predict(svmfit), obs=dat$y)
          obs
   pred -1   1
    -1   9   2
     1   1   8
```

## Support Vector Machines

The hyperplane approaches we've discussed so far only consider linear boundaries for separating response classes. As the figure below shows, in some cases, these just don't work.

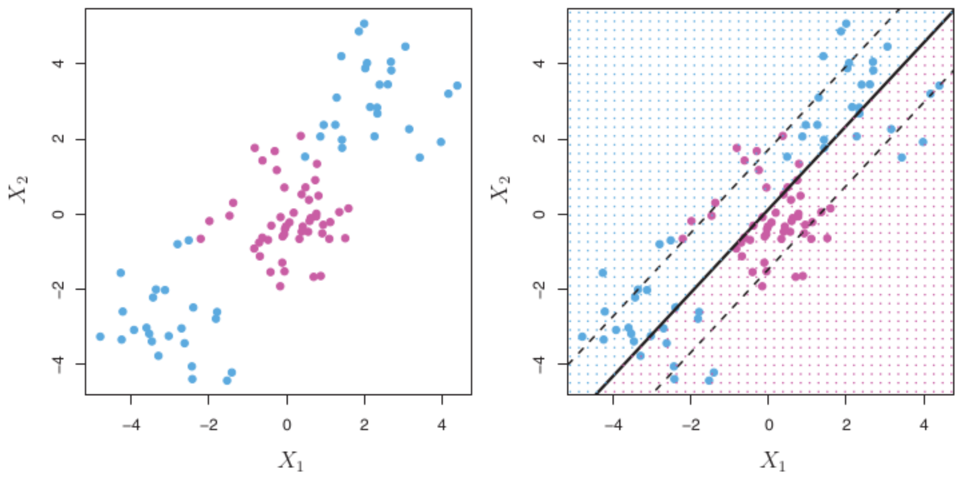


**FIGURE 9.8.** Left: *The observations fall into two classes, with a non-linear boundary between them.* Right: *The support vector classifier seeks a linear boundary, and consequently performs very poorly.*

The Support Vector Machine (SVM) is a classification method that replaces the hyperplane with a nonlinear decision boundary. We'll just go over the main idea here, more details are in AISL and ESL.

First recall the inner product of two $(p \times 1)$ vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$,

$$\langle \boldsymbol{x}, \boldsymbol{x}' \rangle = \sum_{j=1}^{p} x_j x_j'$$

Then next, note that the SVC with linear decision boundary can be written as

$$f(\boldsymbol{x}) = \beta_0 + \sum_{i \in S} \alpha_i \langle \boldsymbol{x}, \boldsymbol{x}_i \rangle$$

where $\beta_0$ and $\alpha_1, \ldots \alpha_n$ are coefficients to be estimated and $X$ is the set of indicies of those points that are support vectors. The SVM essentiall replaces the inner product above with a kernel function $K(\boldsymbol{x}, \boldsymbol{x}_i)$ describing a nonlinear boundary. Here are a couple of examples,

- Degree $d$ polynomial kernel

- Radial kernel

The decision boundary for SVM is then

$$f(\boldsymbol{x}) = \beta_0 + \sum_{i \in S} \alpha_i K(\boldsymbol{x}, \boldsymbol{x}_i)$$

Two examples are shown in the figure below.

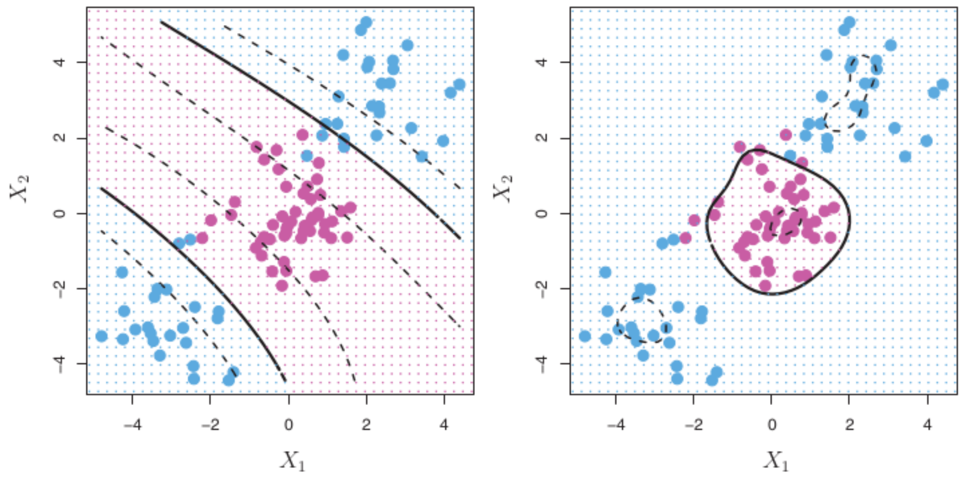
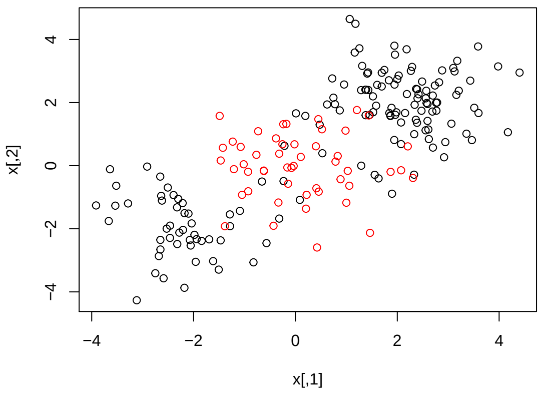


**FIGURE 9.9.** Left: *An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from Figure 9.8, resulting in a far more appropriate decision rule.* Right: *An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.*

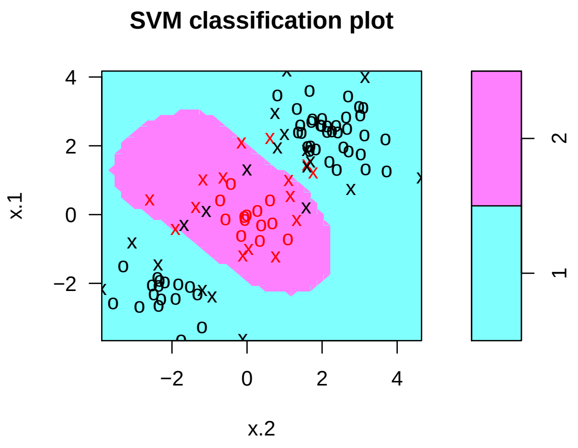
Note that the SVM classifier is local, in the sense that when making a classification at a new point $\boldsymbol{x}^*$, those points nearby (in Euclidean terms) will have the most effect, and those far away will have very little effect.

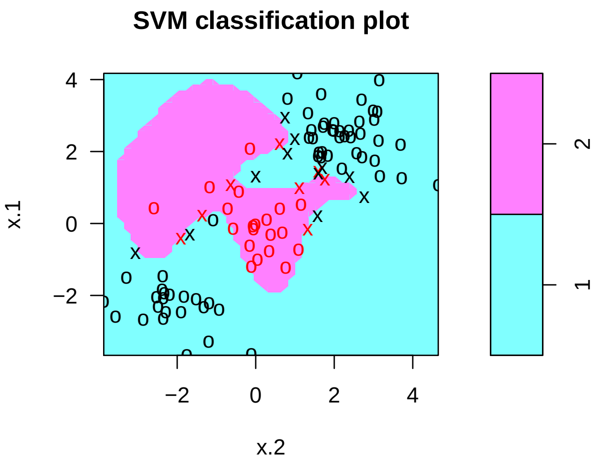
**Example:** Again we will use simulated data. We'll fit a SVM classifier using the radial kernel.

```
set.seed(1)
x=matrix(rnorm(200*2), ncol=2)
x[1:100,]=x[1:100,]+2
x[101:150,]=x[101:150,]-2
y=c(rep(1,150),rep(2,50))
dat=data.frame(x=x,y=as.factor(y))
plot(x, col=y)
```



```
# fit model on training set
train=sample(200,100)
svmfit=svm(y~., data=dat[train,], kernel="radial",  gamma=1, cost=1)  #gamma specifies parameter in kernel fctn
plot(svmfit, dat[train,])

summary(svmfit)

    Parameters:
        SVM-Type:  C-classification
        SVM-Kernel:  radial
        cost:  1

       Number of Support Vectors:  35
       ( 15 20 )
       Number of Classes:  2
       Levels:
        1 2

# change cost and refit
svmfit=svm(y~., data=dat[train,], kernel="radial",gamma=1,cost=1e5)
plot(svmfit,dat[train,])
```





The `tune` function will make search for a model with lowest prediction error over a grid a parameter values. Below we search for the best values of

```
> set.seed(1)
> # tune does grid search over parameter range
> # finds combo with lowest prediction error
> tune.out=tune(svm, y~., data=dat[train,], kernel="radial", ranges=list(cost=c(0.1,1,10,100,1000),gamma=c(0.5,1,2,3,4)
> summary(tune.out)

Parameter tuning of svm:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    1   0.5

- best performance: 0.09

- Detailed performance results:
    cost gamma error dispersion
1  1e-01   0.5  0.20 0.14142136
2  1e+00   0.5  0.09 0.08755950
3  1e+01   0.5  0.10 0.08164966
4  1e+02   0.5  0.11 0.09944289
5  1e+03   0.5  0.14 0.13498971
6  1e-01   1.0  0.11 0.09944289
7  1e+00   1.0  0.10 0.08164966
8  1e+01   1.0  0.09 0.07378648
9  1e+02   1.0  0.14 0.12649111
10 1e+03   1.0  0.14 0.12649111
11 1e-01   2.0  0.18 0.12292726
12 1e+00   2.0  0.10 0.08164966
13 1e+01   2.0  0.12 0.09189366
14 1e+02   2.0  0.19 0.12866839
15 1e+03   2.0  0.18 0.13165612
16 1e-01   3.0  0.22 0.13165612
17 1e+00   3.0  0.10 0.08164966
18 1e+01   3.0  0.16 0.09660918
19 1e+02   3.0  0.15 0.11785113
20 1e+03   3.0  0.18 0.13165612
21 1e-01   4.0  0.26 0.11737878
22 1e+00   4.0  0.10 0.08164966
23 1e+01   4.0  0.16 0.11737878
24 1e+02   4.0  0.16 0.11737878
25 1e+03   4.0  0.19 0.12866839

> table(true=dat[-train,"y"], pred=predict(tune.out$best.model,newdata=dat[-train,]))
    pred
true  1  2
   1 68 10
   2  1 21
```