Support Vector Machines

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Founding Director, Center for Data Science
Associate Director, Center for Biomedical Informatics and Health AI
Rutgers University – New Jersey Medical School

2025-11-13

Support Vector Machines

Support vector machines or **SVM**s are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.

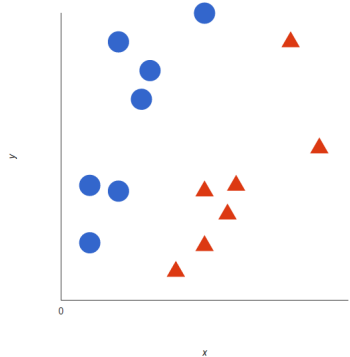
The goal is to find a classifier from an optimized *decision boundary* or *"separating hyperplane"* between two classes.

Material for this lecture was obtained and adapted from:

- https://www.datacamp.com/community/tutorials/support-vector-machines-r
- *The Elements of Statistical Learning*, Hastie, et al., Springer
- https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-r/amp/

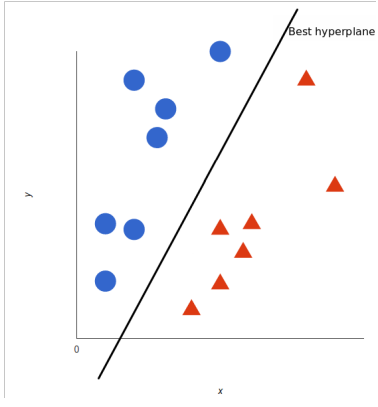
Support Vector Machines–Linear Data

Let's imagine we have two tags: *red* and *blue*, and our data has two features: x and y. We can plot our training data on a plane:



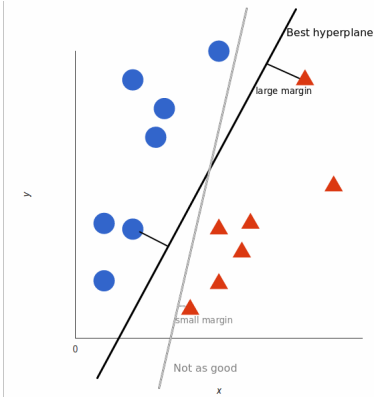
Support Vector Machines

An **SVM** identifies the **decision boundary** or **hyperplane** (two dimensions: line) that best separates the tags:



Support Vector Machines

But, what exactly is the best hyperplane? For SVM, it's the one that maximizes the margins from the data from both tags:



A Look into SVM Methodology

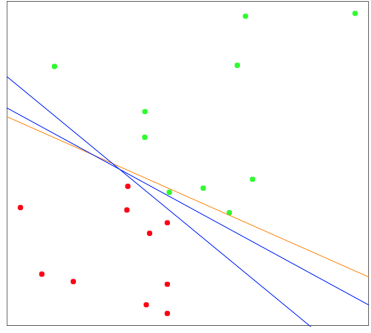


FIGURE 4.14. *A toy example with two classes separable by a hyperplane. The orange line is the least squares solution, which misclassifies one of the training points. Also shown are two blue separating hyperplanes found by the* perceptron learning algorithm *with different random starts.*

A Look into SVM Methodology

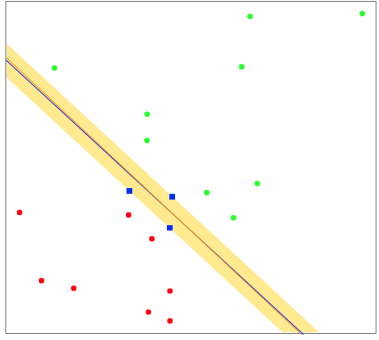
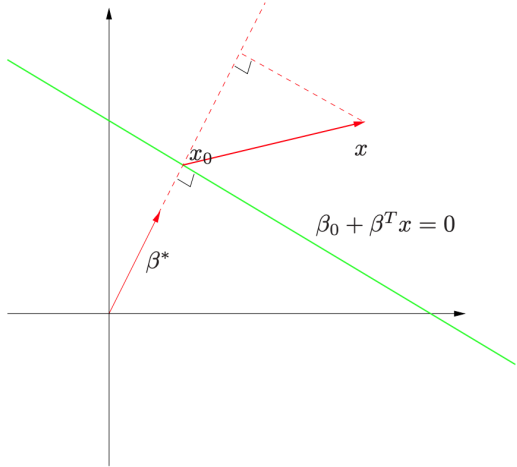


FIGURE 4.16. *The same data as in Figure 4.14. The shaded region delineates the maximum margin separating the two classes. There are three support points indicated, which lie on the boundary of the margin, and the optimal separating hyperplane (blue line) bisects the slab. Included in the figure is the boundary found using logistic regression (red line), which is very close to the optimal separating hyperplane (see Section 12.3.3).*

A Look into SVM Methodology



A Look into SVM Methodology

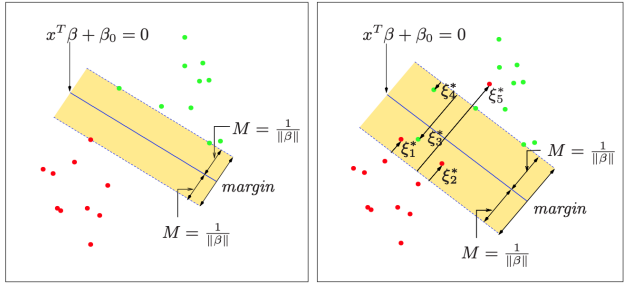


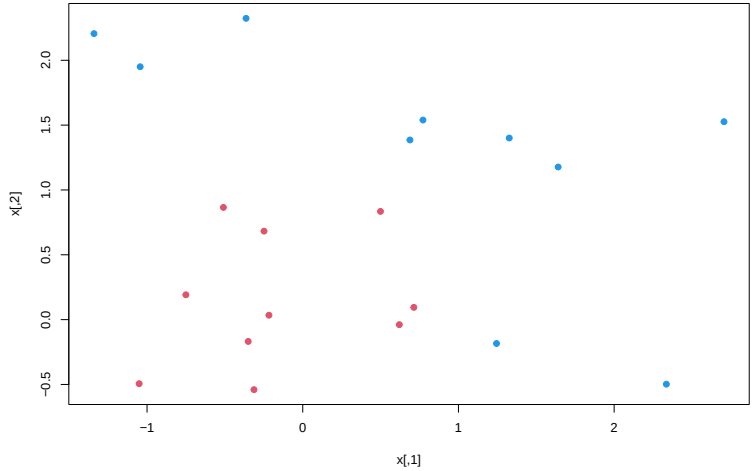
FIGURE 12.1. *Support vector classifiers. The left panel shows the separable case. The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width* $2M = 2/\|\beta\|$. *The right panel shows the nonseparable (overlap) case. The points labeled* ξ_j^* *are on the wrong side of their margin by an amount* $\xi_j^* = M\xi_j$; *points on the correct side have* $\xi_j^* = 0$. *The margin is maximized subject to a total budget* $\sum \xi_i \leq$ *constant. Hence* $\sum \xi_j^*$ *is the total distance of points on the wrong side of their margin.*

Support Vector Machines in R

First generate some data in 2 dimensions, and make them a little separated:

```
set.seed(10111)
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)
```

Support Vector Machines in R



Support Vector Machines in R

We will use the **e1071** package which contains the svm function and make a dataframe of the data, turning y into a factor variable.

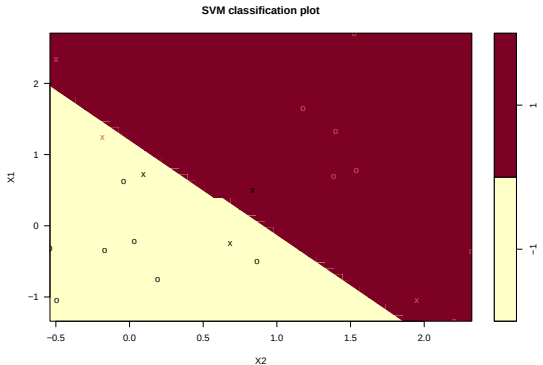
```
library(e1071)
dat = data.frame(x, y = as.factor(y))
svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
print(svmfit)
```

```
##
## 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:  6
```

Support Vector Machines in R

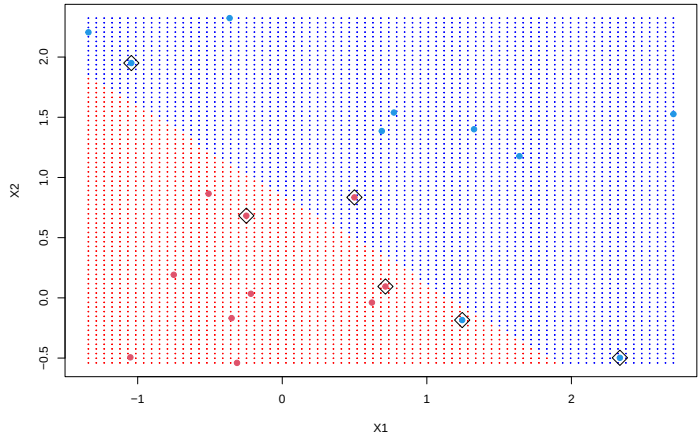
There's a plot function for SVM that shows the decision boundary

```
plot(svmfit, dat)
```



Support Vector Machines in R

Or plotting it more cleanly:



Support Vector Machines in R

Unfortunately, the svm function is not too friendly, in that you have to do some work to get back the linear coefficients. The reason is probably that this only makes sense for linear kernels, and the function is more general. So let's use a formula to extract the coefficients more efficiently. You extract β and β_0, which are the linear coefficients.

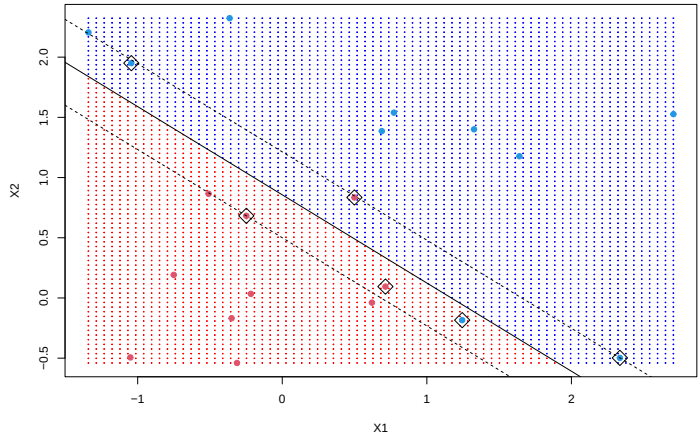
```
beta = drop(t(svmfit$coefs)%*%x[svmfit$index,])
beta0 = svmfit$rho
```

Now you can replot the points on the grid, then put the points back in (including the support vector points). Then you can use the coefficients to draw the decision boundary using a simple equation of the form:

$$\beta_0 + x_1\beta_1 + x_2\beta_2 = 0$$

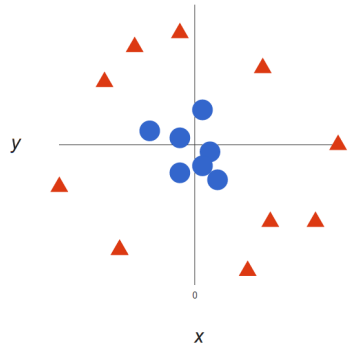
Support Vector Machines in R

Now plotting the lines on the graph:



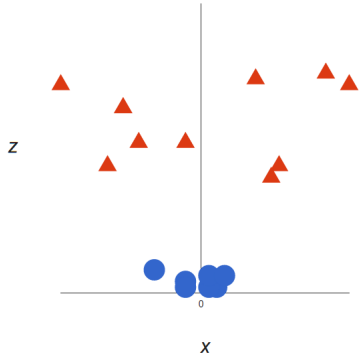
Support Vector Machines–Non-Linear Data

The prior examples were easy because the data were linearly separable. Often things aren't that simple. Look at this case:



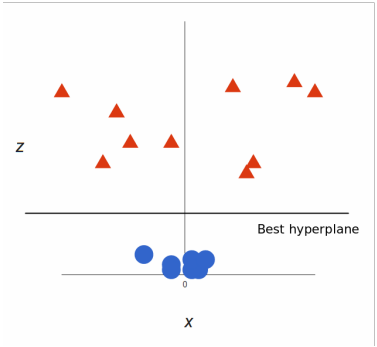
Support Vector Machines

Here we will add a third dimension: z = x2 + y2 (you'll notice that's the equation for a circle!), and plot x and z.



Support Vector Machines

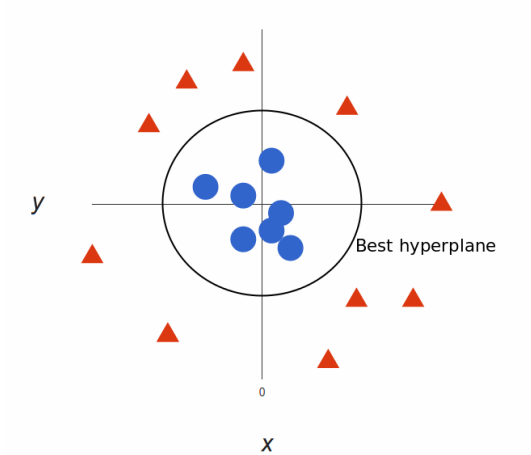
What can SVM do with this? Let's see:



That's great! Note that since we are in three dimensions now, the hyperplane is a plane parallel to the x axis at a certain z (let's say $z=1$).

Support Vector Machines

What's left is mapping it back to two dimensions:



The “Kernel Trick”

Transformations are computationally expensive, but SVM doesn't need the actual vectors, it only needs the dot products.

We can tell SVM to do its thing using the new dot product—we call this a **kernel function.**

The “Kernel Trick”

This often called the *“kernel trick”*, which enlarges the feature space for a non-linear boundary between the classes.

Common types of kernels: *linear*, *polynomial*, and *radial basis* kernels.

Simply, these kernels transform our data to pass a linear hyperplane and thus classify our data.

Support Vector Machines in R: Non-linear SVM

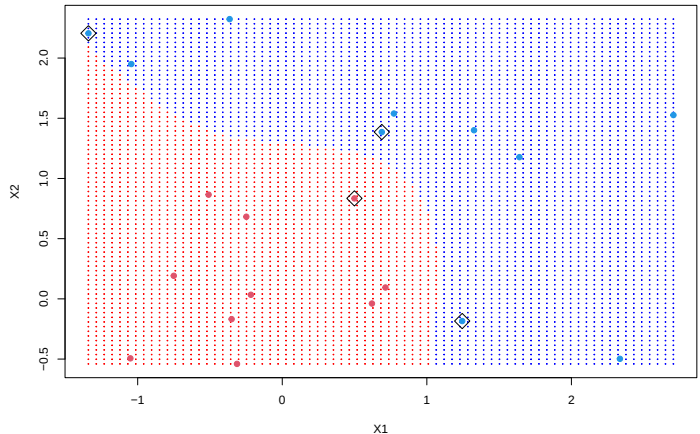
Now let's apply a non-linear (polynomial) SVM to our prior simulated dataset.

```
dat = data.frame(x, y = as.factor(y))
svmfit = svm(y ~ ., data = dat, kernel = "polynomial", cost = 10, scale = FALSE)
print(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "polynomial", cost = 10,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  4
```

Support Vector Machines in R: Non-linear SVM

Plotting the result:



Support Vector Machines in R: Non-linear SVM

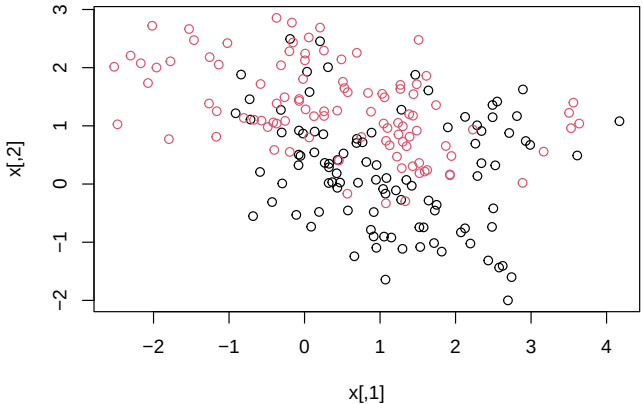
Here is a more complex example from *Elements of Statistical Learning*, where the decision boundary needs to be non-linear and there is no clear separation.

```
#download.file(
#  "http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/ESL.mixture.rda",
#  destfile='ESL.mixture.rda')
rm(x,y)
load(file = "ESL.mixture.rda")
attach(ESL.mixture)
names(ESL.mixture)
```

```
## [1] "x"        "y"        "xnew"     "prob"     "marginal" "px1"      "px2"
## [8] "means"
```

Support Vector Machines in R: Non-linear SVM

Plotting the data:



Support Vector Machines in R: Non-linear SVM

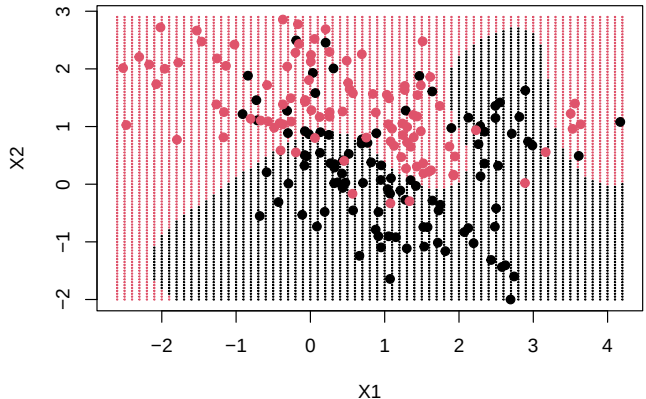
Now make a data frame with the response y, and turn that into a factor. We will fit an SVM with radial kernel.

```
dat = data.frame(y = factor(y), x)
fit = svm(factor(y) ~ ., data = dat, scale = FALSE, kernel = "radial", cost = 5)
print(fit)
```

```
##
## Call:
## svm(formula = factor(y) ~ ., data = dat, kernel = "radial", cost = 5,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  5
##
## Number of Support Vectors:  103
```

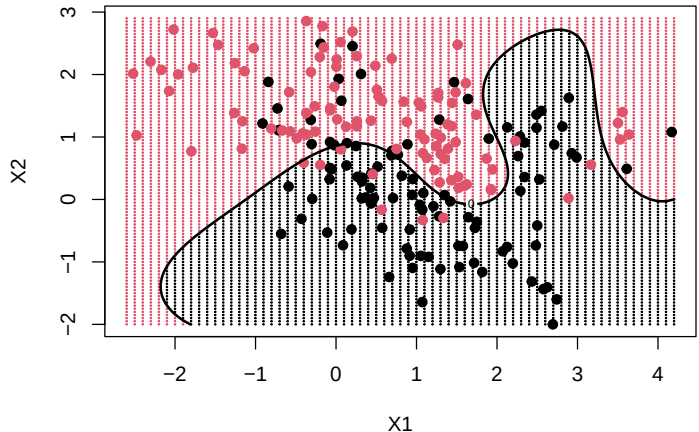
Support Vector Machines in R: Non-linear SVM

It's time to create a grid and predictions. We use `expand.grid` to create the grid, predict each of the values on the grid, and plot them:



Support Vector Machines in R: Non-linear SVM

Plotting with a contour:



Advantages and Disadvantages of SVMs

Advantages:

- **High Dimensionality:** SVM is an effective tool in spaces where dimensionality is large.
- **Memory Efficiency:** Only a subset of the training points are used, so just these points need to be stored in memory.
- **Versatility:** Class separation is often highly non-linear. The ability to apply new kernels allows flexibility for the decision boundaries.

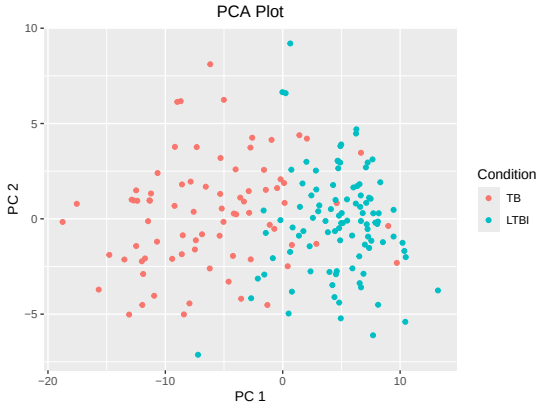
Advantages and Disadvantages of SVMs

Disadvantages:

- **Kernel Selection:** SVMs are very sensitive to the choice of the kernel parameters.
- **High dimensionality:** When the number of features for each object exceeds the number of training data samples, SVMs can perform poorly.
- **Non-Probabilistic:** The classifier works by placing objects above/below a classifying hyperplane, there is no direct probabilistic interpretation.

SVM Nanostring

Remember the PCA dimension reduction of the TB Nanostring dataset. The points are colored based on TB status.



SVM Nanostring

Now try an SVM on the PCs of the Nanostring data:

```
# use only the first 10 PCs
dat = data.frame(y = pca_reduction$Condition, pca_reduction[,1:2])
fit = svm(y ~ ., data = dat, scale = FALSE, kernel = "linear", cost = 10)
print(fit)
```

```
##
## 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:  52
```

SVM Nanostring

We can evaluate the predictor with a `Confusion matrix`:

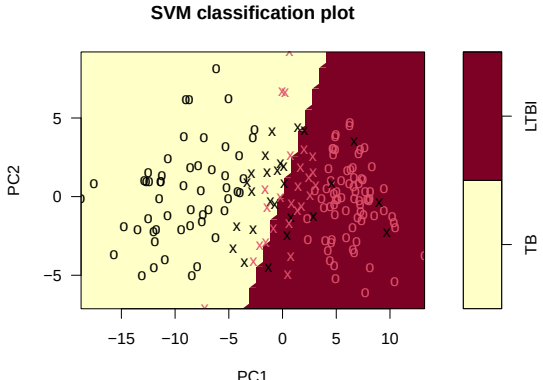
```
library(caret)
confusionMatrix(dat$y,predict(fit,dat))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction TB LTBI
##       TB   69   10
##       LTBI  8   92
##
##                Accuracy : 0.8994
##                  95% CI : (0.8457, 0.9393)
##     No Information Rate : 0.5698
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7955
##
##  Mcnemar's Test P-Value : 0.8137
##
##             Sensitivity : 0.8961
##             Specificity : 0.9020
##          Pos Pred Value : 0.8734
##          Neg Pred Value : 0.9200
##              Prevalence : 0.4302
```

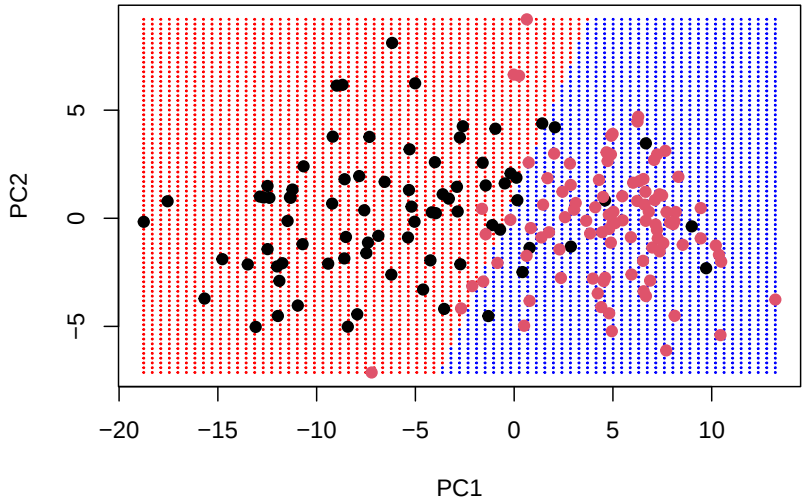
SVM Nanostring

Plotting Nanostring data:

```
plot(fit,dat,PC2~PC1)
```



SVM Nanostring



Session Info

```
sessionInfo()
```

```
## R version 4.5.1 (2025-06-13)
## Platform: aarch64-apple-darwin20
## Running under: macOS Sequoia 15.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;  LAPACK version 3.12.1
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/Denver
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] e1071_1.7-16    lubridate_1.9.4 forcats_1.0.0   stringr_1.5.1
##  [5] dplyr_1.1.4     purrr_1.1.0     readr_2.1.5     tidyr_1.3.1
##  [9] tibble_3.3.0    tidyverse_2.0.0 caret_7.0-1     lattice_0.22-7
## [13] ggplot2_3.5.2
##
## loaded via a namespace (and not attached):
## [1] gtable_0.3.6       xfun_0.52          recipes_1.3.1
```