Introduction to Support Vector Machines

William Morgan

30 March, 2018

Outline

- Purpose and Intuition
- Separating Hyperplanes and the Maximal Margin Classifier
- Support Vector Classifiers
- Support Vector Machines with 2 classes

Purpose and Intuition

- Classification technique used for separating observations into one of two classes
  - This can be extended to $K > 2$ classes using One-vs-One or One-vs-All classification
- Construct a decision boundary that divides the predictor space into two halves

Hyperplanes - Definition

- **Definition:**
  - A **hyperplane** of a p-dimensional space is a flat subset with dimension p-1
  - For a 2-dimensional space, a **hyperplane** will just be a line; in 3 dimensions we have a 2-dimensional plane
- More formally:
  - Let $\beta \neq 0$ be a vector in $\mathbb{R}^n$ and let $a \in \mathbb{R}$
  - A **hyperplane** H is then defined as the set:

$$H = \{x \in \mathbb{R}^n \mid \beta \cdot x = a\}$$

Hyperplanes - Definition

- A hyperplane in $\mathbb{R}^2$ can be described with the following equation:

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

- We say that two sets D and E in $\mathbb{R}^2$ are **separated** by the hyperplane H if we have

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 \geq 0, \forall \quad d \in D$$
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 \leq 0, \forall \quad e \in E$$

Hyperplanes - Separation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 < 0 \quad [1]$$

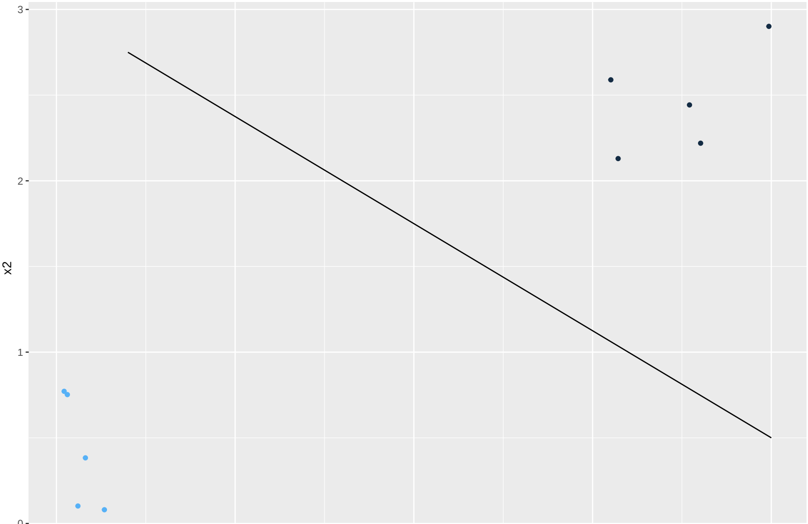
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0 \quad [2]$$

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0 \quad [3]$$

- When we have an observation $x = (x_1, x_2)$ that satisfies the first or third equation, we can get an idea of what side of the hyperplane it lies on
- If we have [2] then x is simply a point on the hyperplane

Hyperplanes - Example

- The hyperplane $-3 + \frac{1}{4}X_1 + X_2 = 0$. The set of points (x_1, x_2) for which this equation is greater than 0 are colored black, where those less than 0 are colored blue



The Maximal Margin Classifier

- Usually if we can define one hyperplane that separates our data, we can define many
- An intuitive choice for the *best* hyperplane is the one that is furthest from our training observations
  - Think of it as being the boundary that creates the most distance between the two groups

The Maximal Margin Classifier

- For each hyperplane, we can find the distance to the closest observation
  - This is known as the **margin**
  - The points that are closest to each side of the hyperplane are called **support vectors**
- Hence, the **maximal margin classifier** will be the hyperplane for which the margin is the largest

The Maximal Margin Classifier

- Can lead to overfitting in high-dimensional cases
- Usually unrealistic to assume that data is linearly separable
  - Even if it is, the margin might be small and sensitive to new observations
- Need to come up with less rigid solution to the classification problem

Support Vector Classifiers

- Support Vector Classifiers loosen the requirement that all training observations lie on the correct side of a separating hyperplane
- We allow some observations to be misclassified for two reasons:
  - Allow our model to be more robust to individual observations (overfitting)
  - "Widen" our margin to allow for better classification on testing sets
- For these reasons Support Vector Classifiers are also known as *Soft Margin Classifiers*

Support Vector Classifiers - Setup

- $i = 1, ..., n$ observations in the training set
- $j = 1, ..., p$ predictors
- $y_i \in \{-1, 1\}$ classes
- Our hyperplane will be defined as:

$$\beta_0 + \beta_1 X_1 + ... + \beta_p X_p = 0$$

Support Vector Classifiers - Setup

- A support vector classifier can then be described as the solution to the optimization problem:

$$
\begin{aligned}
\max_{\beta,\epsilon} \quad & M \\
\text{s.t.} \quad & \textstyle\sum_{j=1}^{p} \beta_j^2 = 1, \\
& y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq M(1 - \epsilon_i) \\
& \epsilon_i \geq 0, \forall i \in 1, ..., n \\
& \textstyle\sum_{i=1}^{n} \epsilon_i \leq C
\end{aligned}
$$

Support Vector Classifiers - Setup

- M is the **margin** and is a positive number
- $\sum_{j=1}^{p} \beta_j^2 = 1$ constrains the size of each β_j
- $y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq M(1 - \epsilon_i)$ ensures that each observation will fall on the correct side of the hyperplane, except for maybe a couple
  - Note that a hyperplane $a_o + a_1 x_1 + a_2 x_2 = 0$ is still a hyperplane when it is multiplied by a scalar $y(a_0 + a_1 x_1 + a_2 x_2) = 0$
  - Also recall that y_i is either 1 or -1; y_i will give us an indication of which side of the hyperplane the observation is (above or below)

Support Vector Classifiers - Setup

- Think of all the ϵ_i as individual tolerances to violations of the hyperplane
  - When $\epsilon_i < 1$, the observation will be within the margin (but still classified correctly)
  - In the case of $\epsilon_i > 1$, our observation will be misclassified
  - $\epsilon_i = 0$ implies that the observation is at least M distance away from the margin

Support Vector Classifiers - Setup

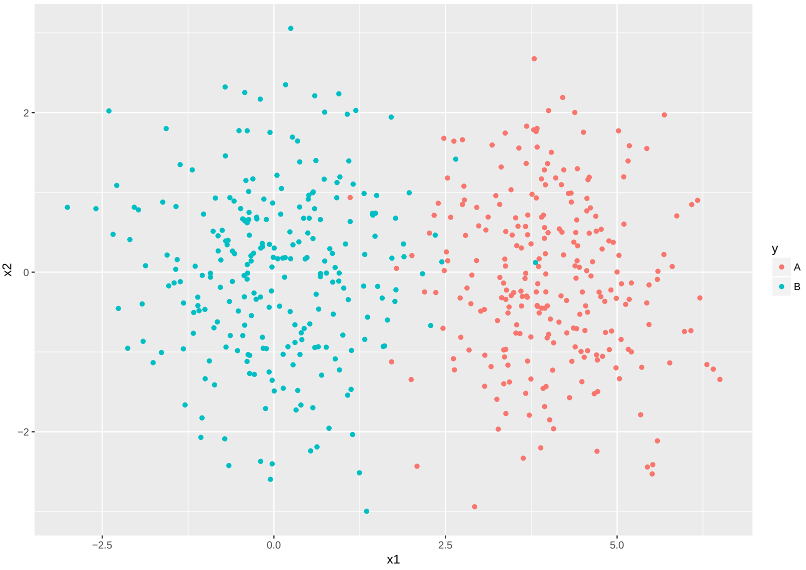
- In total, we choose to only allow C amount of error
- $C = 0$ implies that there is no tolerance at all, which is equivalent to the maximal margin classifier problem
- Larger and larger values of C increase the margin, allowing for less accuracy but lower variance
- C can be tuned using cross-validation

Support Vector Classifiers - Example

```r
# Set up toy data to work with
set.seed(1)
dat <- tibble(
  x1 = rnorm(500),
  x2 = rnorm(500),
  y = factor(c(rep('A',250), (rep('B',250))))        # Y is
)

# Add a little of separability between the classes
dat$x1[dat$y == "A"] <-  dat$x1[dat$y == "A"] + 4
```

Support Vector Classifiers - Example



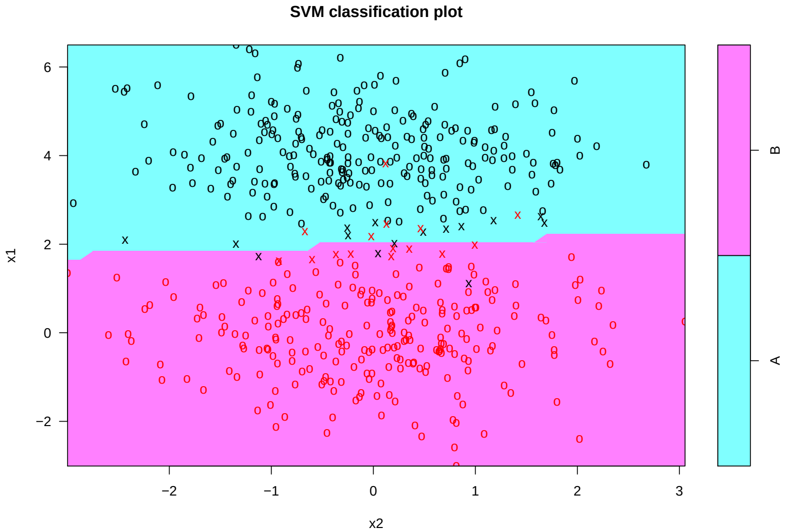
Support Vector Classifiers - Example

```
library("e1071")
svc <- svm(y ~ ., data = dat, kernel = 'linear',
           cost = 10, scale = FALSE)
```

Support Vector Classifiers - Example

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.5
##
## Number of Support Vectors:  30
##
##  ( 15 15 )
##
##
## Number of Classes:  2
```

Support Vector Classifiers - Example



Support Vector Classifiers - Tuning

- How do we decide on a value for our cost parameter C?
- `e1071::tune()` allows us to specify a range of values that it will test using 10-fold cross-validation

```
svc_cv <-  tune(svm,y ~.,
                    data = dat,
                    kernel = 'linear',
                    ranges = list(cost = c(.001, .01, .1, 1, 5,
```

Support Vector Classifiers - Tuning

- Extract the best performing model by grabbing the `best.model` component of `svc_cv`

```
best_model <-  svc_cv$best.model
```

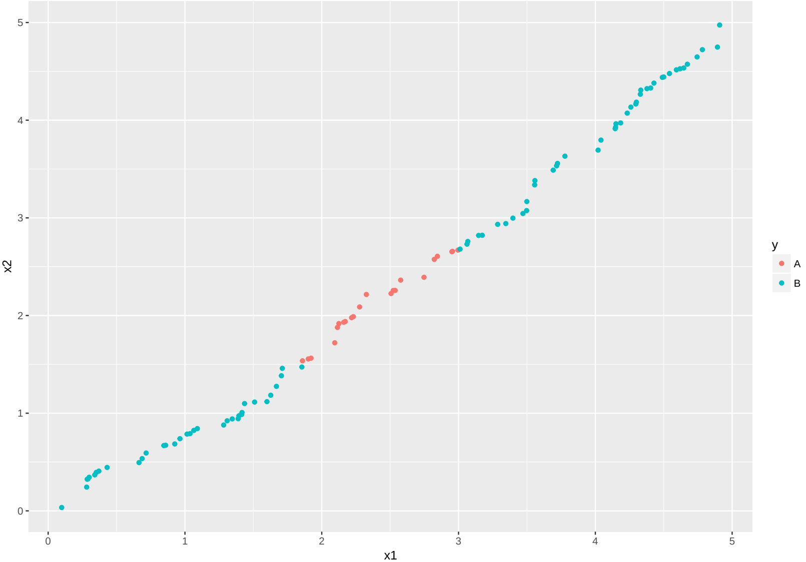
Support Vector Classifiers - Prediction

```r
predictions <- predict(best_model, test)
table(predictions, truth = test$y)
```

```
##            truth
## predictions   A   B
##           A 243   5
##           B   7 245
```

Non-linear Boundaries

What happens when we can't separate our data with a linear boundary?



Non-linear Boundaries

- Enlarge our feature space using functions of the predictors
  - $X \mapsto \phi(X)$
- Before we fit the SVC with p predictors:

$$X_1, X_2, ..., X_p$$

- We could have fit the SVC with any amount of predictors to allow our boundary to deal with the non-linearity:

$$X_1, X_2, ..., X_p, X_1^2, X_2^2, ..., X_p^2$$

Non-linear Boundaries - Computation

- The optimization problem described earlier is solved using the *inner products* of the support vectors
  - *inner products* can be thought as a measure of similarity between two observations
  - The inner product between two points x_1, x_2 is denoted $\langle x_1, x_2 \rangle$

Non-linear Boundaries - Computation

- Transforming our feature space $X \mapsto \phi(X)$ could potentially make the computation much more difficult
  - instead of $\langle x_1, x_2 \rangle$ we have $\langle \phi(x_1), \phi(x_2) \rangle$
  - could be super time consuming depending on ϕ
- We can simplify this computation by knowing the **kernel function**

Non-linear Boundaries - Kernels

- **Kernels** generalize inner products by allowing us to use different metrics of similarity
- SVCs have linear kernels
- A **kernel function** is usually written:

$$K(x_1, x_2)$$

- **Support Vector Machines** transform the feature space using kernels

Support Vector Machines - Kernels

- A **support vector machine** is a combination of a support vector classifier with a non-linear kernel
- There are 3 very popular kernels:

$$K(x_1, x_2) = (1 + \sum_{j=1}^{p} x_{1j}x_{2j})^d \quad [1]$$

$$K(x_1, x_2) = exp(-\gamma \sum_{j=1}^{p} (x_{1j} - x_{2j})^2) \quad [2]$$

$$K(x_1, x_2) = tanh(\gamma \sum_{j=1}^{p} x_{1j}x_{2j} + \delta) \quad [3]$$

Support Vector Machines - Kernels

- [1] is called a *polynomial kernel* of degree d
- [2] is the *radial basis function* and is very good at describing points close to a training observation
- [3] is the *hyperbolic tangent* function

Support Vector Machines - Example

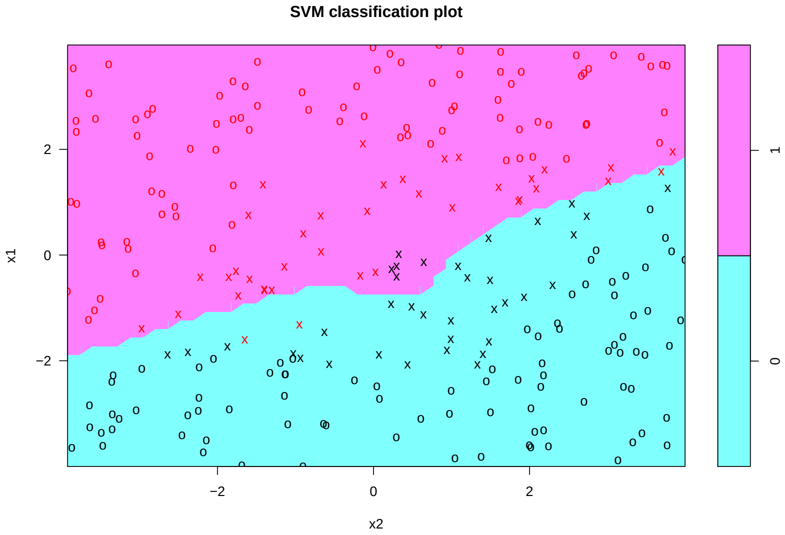
- We use an artificial data set made to work well with a polynomial kernel

Support Vector Machines - Example

- SVM with a polynomial kernel of degree 3

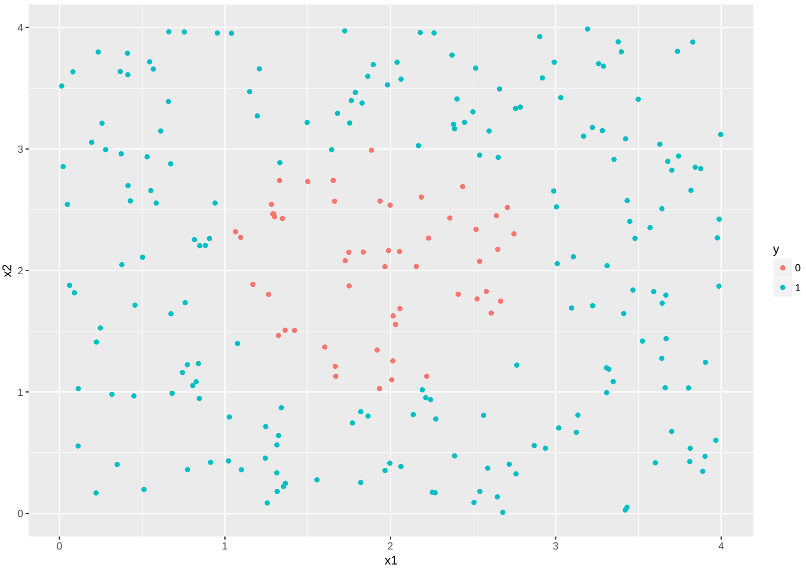
```
svm_poly <-  svm(y ~.,
                 data = poly_dt,
                 kernel = "polynomial")
```

Support Vector Machines - Example



Support Vector Machines - Example

- New data to predict:

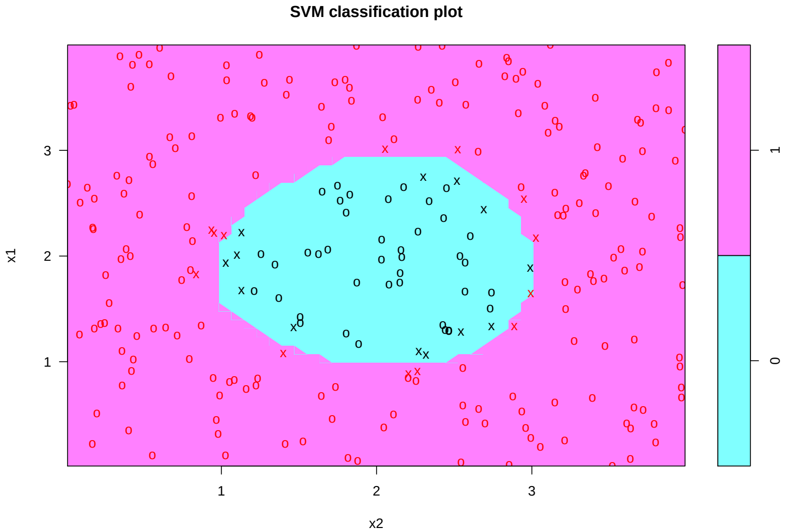


Support Vector Machines - Example

- SVM with radial kernel

```
svm_rad <-  svm(y ~ .,
                data = rad_dt,
                kernel = "radial",
                cost = 10)
```

Support Vector Machines - Example



Support Vector Machines - Tuning

- These non-linear kernel functions have new parameters that you have to choose yourself
  - Let CV do it for you!

```
tune(svm, y ~ .,
     data = train,
     kernel = 'radial',
     ranges = list(cost = c(.1, 1, 10, 100, 1000),
                   gamma = c(.5, 1, 2, 3, 4)))
```

Support Vector Machines - Evaluating

- Basic Workflow:
  - Define training and testing sets
  - Decide on a model (predictors, kernel, classes)
  - Use cross-validation to find hyperparameters with best training results (optional)
  - Evaluate best model using test set (ROC Curves, Classification Error, etc.)