Support Vector Machines

D2K Course Staff

3/18/2019

Introduction

In the module that covered Linear Discriminant Analysis and Naive Bayes, we briefly covered separating hyper-planes as a method of classification. Both methods use probability distributions in order to determine linearly separating hyperplanes with slightly different sets of assumptions. But what if our data is not linearly separable? What if our data is noisy and we have many data points that break the rules of our separating hyperplane? Additionally, what if our data is separable, but not by a **linear** plane in p-dimensions?

Support vector machines give us the tools to deal with these different scenarios, and have been shown to work better than more traditional techniques like LDA and NB in most use cases. The basic idea of support vector machines begins with defining a hyperplane

$$\{x : f(x) = x^T\beta + \beta_0 = 0\}$$

where $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$ and $||\beta|| = 1$. We then determine the classification of a data point based on the function

$$G(x) = sign[x^T\beta + \beta_0]$$

where $G(x)$ gives us the label for new data. Furthermore, $f(x)$ actually gives us the distance from a point x to the hyperplane $f(x) = x^T\beta + \beta_0 = 0$.

The concept of a support vector machine begins with the idea of finding the **margin** that *maximizes* the separation between the two classes. Naturally, that lends itself to an optimization problem where we define the margin to be M units away from the hyperplane, and thus have a total width of $2M$.

Under the assumption that we can separate the two classes, we can find a function $f(x) = x^T\beta + \beta_0$ where $y_i f(x_i) > 0 \ \ \forall i$. Essentially, this is saying there exists a margin line on the same side as the y_i in any feature space. The question remains, how wide do we make the large? We recall the optimization problem we introduced in the previous paragraph and define it as

$$\max_{\beta,\beta_0,||\beta||=1} M$$

$$\textit{subject to } y_i(x_i^T\beta + \beta_0) \geq M,\ i = 1, ..., N$$

or more conveniently as

$$\max_{\beta,\beta_0} ||\beta||$$

$$\textit{subject to } y_i(x_i^T\beta + \beta_0) \geq 1,\ i = 1, ..., N$$

where we get rid of the constraint on $||\beta||$ and $M = \frac{1}{||\beta||}$. But what if our two classes aren't perfectly separable by our maximized margins? How can we account for the outliers? The solution, and what puts the "support vector" in SVM's is the inclusion of slack variables in the optimization problem. These slack variables allow us to train our classifier to pick up points that are misclassified by the margin and re-classify them as the correct label. These points that lie on the incorrect side of the margin are considered our "support vectors". We define the slack variables $\xi = (\xi_1, ..., \xi_N)$ and the new optimization as

$$\max_{\beta,\beta_0} \frac{1}{2}||\beta||^2 + C\Sigma_{i=1}^{N}\xi_i$$

$$\textit{subject to } \xi_i \geq 0,\ y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i,\ \forall i$$

where C is the "cost" parameter. This parameter determines how "wiggly" the SVM is, meaning how far it will reach into the other classes region to re-classify the point. If the cost parameter is small, the margin will be much larger, and if the cost parameter is big, the margin will be smaller. Thus, if we use too small of a cost value for SVM's, we are likely to overfit to our training data and get bad results when predicting on future data. The cost parameter should be tuned in the model training process using cross-validation.

For computational purposes, we use a Lagrangian multiplier to solve the above optimization problem. What this allows us to do is express the optimization problem as a single differentiable function. The primal function, which we minimize, takes the form

$$L_P = \frac{1}{2}||\beta||^2 + C\Sigma_{i=1}^{N}\xi_i - \Sigma_{i=1}^{N}\alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] - \Sigma_{i=1}^{N}\mu_i\xi_i$$

and the dual, which we maximize, takes the form

$$L_D = \Sigma_{i=1}^{N}\alpha_i - \frac{1}{2}\Sigma_{i=1}^{N}\Sigma_{i'=1}^{N}\alpha_i\alpha_i'y_iy_i'x_i^Tx_i'$$

where both formulations result in the same solution. If you want to look more into primals and duals in optimization theory there are plenty of resources, but this paper from Stanford's David Knowles gives a good overview: https://cs.stanford.edu/people/davidknowles/lagrangian_duality.pdf.

In practice, we always solve the dual as it allows for easier computation. Furthermore, they allow for us to apply non-linear kernels to SVM's. But, more on that later. For now, we know the solution of the primal and dual problem yields

$$\hat{\beta} = \Sigma_{i=1}^{N}\hat{\alpha}_iy_ix_i$$

where $\hat{\alpha}_i$ is non-zero for observations that do not meet the constraints specified above. These observations are the **support vectors**. We can determine the class label of a new data point using the same decision function we outlined above, but with the new β coefficient.

Support Vector Machines (Linear)

As we mentioned above, non-linear kernels can be applied to SVM's, however, we will first cover the implementation of linear SVM's (we covered all of the math in the introduction). We use the `e1071` package to implement SVM's in R, although other packages like `caret` are also available for use.

We use the good ol' `iris` data set to demonstrate our implementation.

```r
# Load libraries
library(e1071)

# Load data
```

```r
data(iris)
summary(iris)
```

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
 Median :5.800   Median :3.000   Median :4.350   Median :1.300  
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
       Species  
 setosa    :50  
 versicolor:50  
 virginica :50  
```

Of course, we need to chunk our data into training/validation, query, and a test set in order to tune, compare and asses our final model's predictive performance.

```r
# Set seed for reproducibility
set.seed(123)

# Create training/test index
tr_idx <- sample(1:nrow(iris), nrow(iris) * 0.6, replace = FALSE)

# Create training/validation set
iris_train <- iris[tr_idx, ]

iris_test <- iris[-tr_idx, ]

# Create query set from test set
qr_idx <- sample(1:nrow(iris_test), nrow(iris_test) * 0.5, replace = FALSE)

iris_query <- iris_test[qr_idx, ]

iris_test <- iris_test[-qr_idx, ]
```

Now that we have chunked our data, we can proceed in tuning the cost parameter for the linear SVM on our training data using cross-validation.

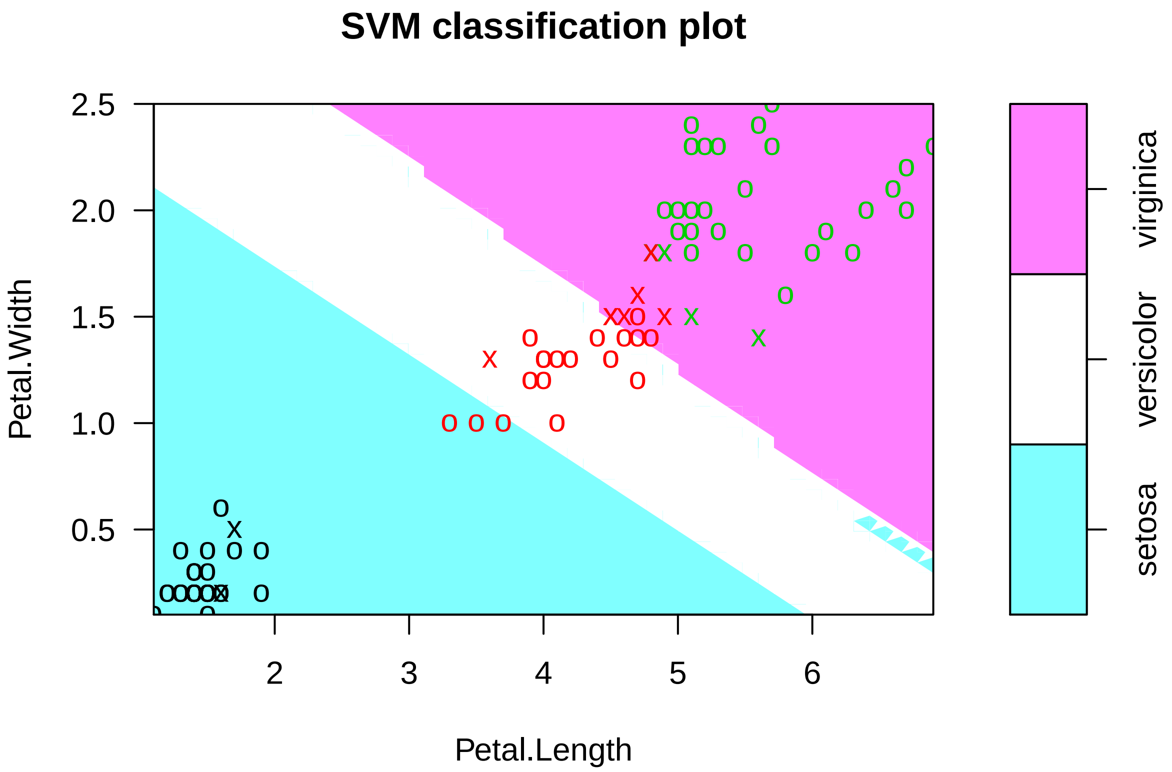
```r
# Set seed for reproducibility
set.seed(123)

# Create tune control, set number of validations, training
# size, and bootstraps
svm_tune <- tune.control(cross = 10, fix = 2/3, nboot = 5)

# Tune model using cross-validation
svm_cv <- tune(svm, Species ~ ., data = iris_train, kernel = "linear",
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)),
    tunecontrol = svm_tune)

# Best model
```

```r
best_svm <- svm_cv$best.model
best_svm
```

```
Call:
best.tune(method = svm, train.x = Species ~ ., data = iris_train,
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)),
    tunecontrol = svm_tune, kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  5
      gamma:  0.25

Number of Support Vectors:  15
```

We can see that our best models chooses a cost parameter value of 5. Ignore the gamma values as that is only relevant when we are using non-linear kernels. We can also visualize what this model looks like in terms of classifiying the 3 flower species.

```r
# Plot best model against two variables
plot(best_svm, iris_train, Petal.Width ~ Petal.Length, slice = list(Sepal.Width = 3,
    Sepal.Length = 4))
```



In the plot above you can see the 3 different outcome labels and their respective regions delineated by color. Furthermore, the support vectors are denoted as "x" while all other data points are denoted by "o". Keep in mind that this is only looking at the data across 2 out of the 4 dimensions of p.

We can then use our best model and predict on the query set and report our prediction accuracy.

```r
# Predict on query set
svm_y_query <- predict(best_svm, iris_query)

# Show misclassified values in a confusion matrix
query_conf_mat <- table(predict = svm_y_query, truth = iris_query$Species)
query_conf_mat
```

```
            truth
predict      setosa versicolor virginica
  setosa          8          0         0
  versicolor      0         13         0
  virginica       0          2         7
```

```r
# Report misclassification error
svm_error <- 1 - mean(svm_y_query == iris_query$Species)
svm_error
```

```
[1] 0.06666667
```

We can see that the SVM performs quite well on predicting new data. This is not surprising as we know that the `iris` data is linearly separable from previous modules.

Kernel Functions

Recall that we were able to define the optimization problem for SVM's by the dual form

$$L_D = \Sigma_{i=1}^N \alpha_i - \frac{1}{2}\Sigma_{i=1}^N \Sigma_{i'=1}^N \alpha_i \alpha_i' y_i y_i' x_i^T x_i'$$

or equivalently

$$L_D = \Sigma_{i=1}^N \alpha_i - \frac{1}{2}\Sigma_{i=1}^N \Sigma_{i'=1}^N \alpha_i \alpha_i' y_i y_i' \langle h(x_i), h(x_i')\rangle$$

where we define the kernel function to be $K(x, x') = \langle h(x), h(x')\rangle$. In the first example, the kernel function was just the identity, however, we can incorporate different transformations in order to differentiate what our decision boundary and associated margins look like in p-dimensions. We highlight 3 of the most common Kernel functions below:

- Radial basis: $K(x, x') = exp(-\gamma||x - x'||^2)$
- d^{th} -Degree polynomial: $K(x, x') = (1 + \langle x, x'\rangle)^d$
- Neural network (sigmoid): $K(x, x') = tanh(\kappa_1\langle x, x'\rangle + \kappa_2)$

Radial Basis Kernel

We show the implementation of a Radial Basis SVM in R using the `e1071` package as we have used previously. It is essentially the same set up as before, except that we need to change the `kernel` argument in the `tune` function and specify a vector of values for γ to tune over.

```r
# Set seed for reproducibility
set.seed(123)

# Tune model using cross-validation
```

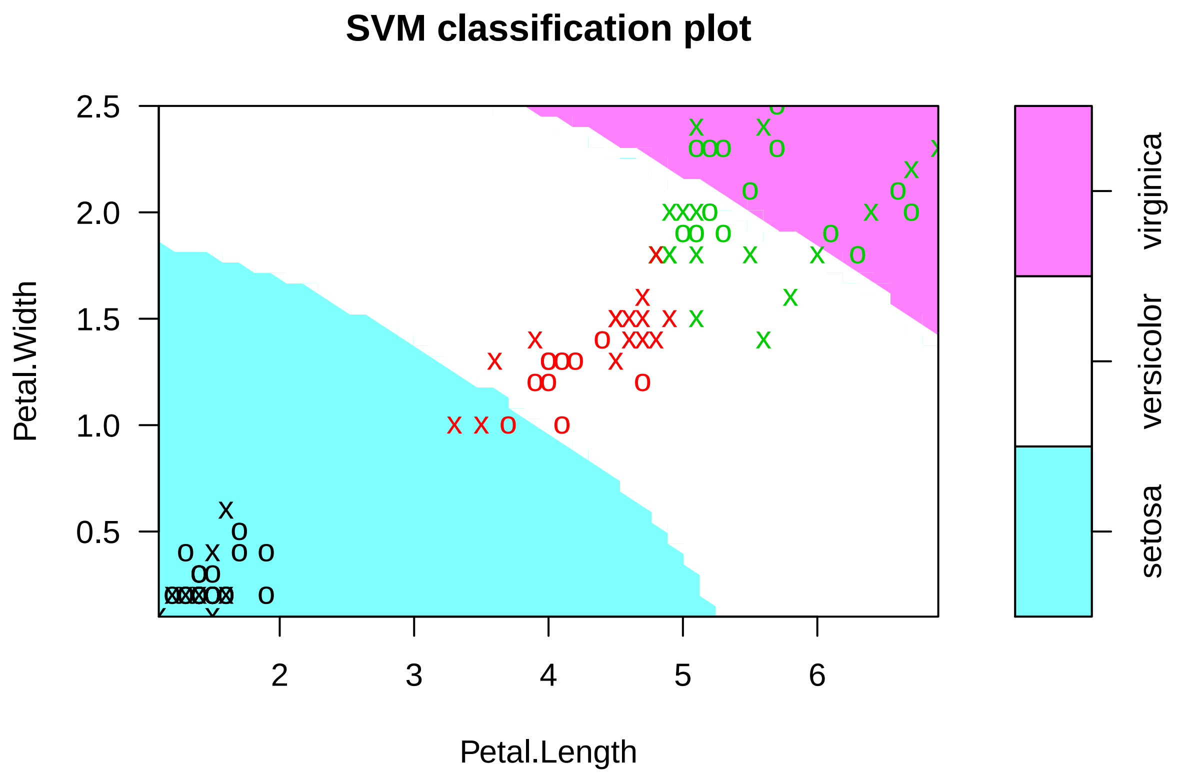
```r
svm_rad_cv <- tune(svm, Species ~ ., data = iris_train, kernel = "radial",
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100),
        gamma = c(0.5, 1, 2, 3, 4)), tunecontrol = svm_tune)

# Best model
best_rad_svm <- svm_rad_cv$best.model
best_rad_svm
```

```

Call:
best.tune(method = svm, train.x = Species ~ ., data = iris_train,
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100),
        gamma = c(0.5, 1, 2, 3, 4)), tunecontrol = svm_tune,
    kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.5

Number of Support Vectors:  41
```

This time we should pay attention to the value of γ that our cross-validation process selects. The γ value influences how much a single point in the training data has, a low-value of γ is a big influence, and high-value of γ is a small influence.

We can visualize what the Radial basis SVM looks like below.

```r
# Plot best model against two variables
plot(best_rad_svm, iris_train, Petal.Width ~ Petal.Length, slice = list(Sepal.Width = 3,
    Sepal.Length = 4))
```



And report our predictive accuracy on the query set…

```
# Predict on query set
svm_rad_y_query <- predict(best_rad_svm, iris_query)

# Show misclassified values in a confusion matrix
rad_query_conf_mat <- table(predict = svm_rad_y_query, truth = iris_query$Species)
rad_query_conf_mat
```

```
            truth
predict       setosa versicolor virginica
  setosa           8          0         0
  versicolor       0         13         0
  virginica        0          2         7
```

```
# Report misclassification error
svm_rad_error <- 1 - mean(svm_rad_y_query == iris_query$Species)
svm_rad_error
```

```
[1] 0.06666667
```

It seems to perform the same as the linear SVM on the query set.

Polynomial Kernel

For the d-Degree polynomial kernel we have the same set of hyperparameters, but also add the degree of polynomial term d.

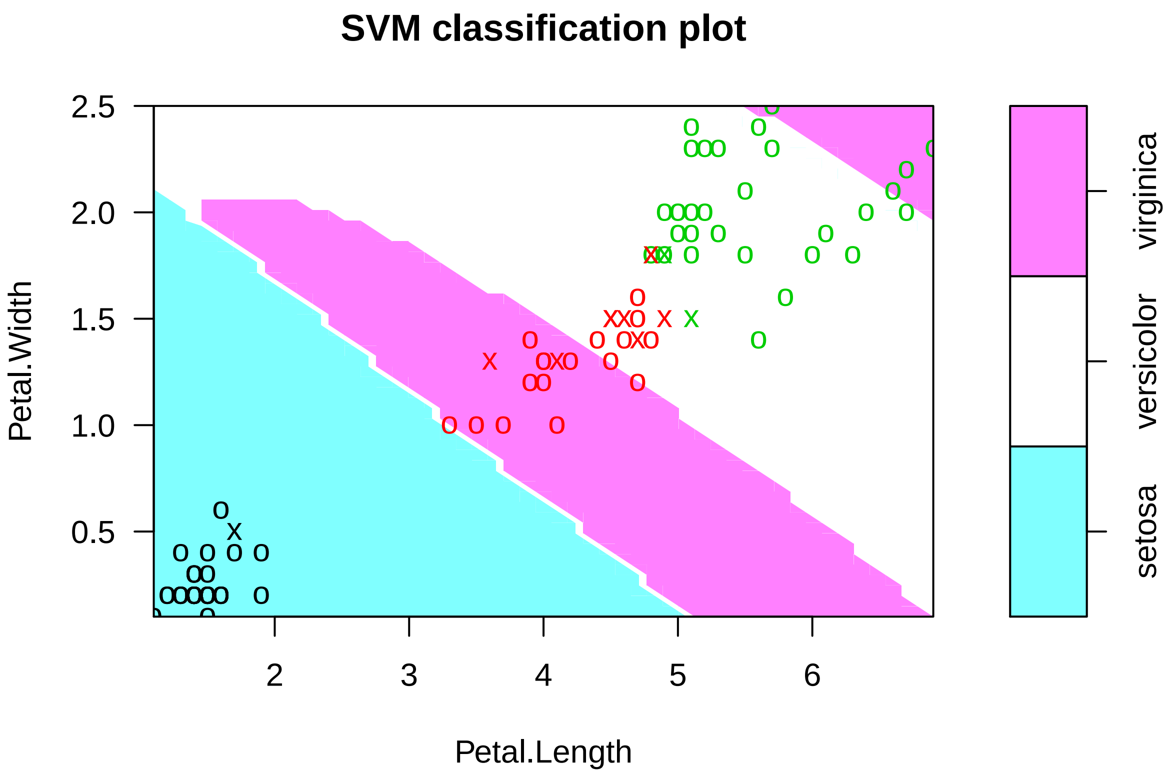
```
# Set seed for reproducibility
set.seed(123)
```

```r
# Tune model using cross-validation
svm_poly_cv <- tune(svm, Species ~ ., data = iris_train, kernel = "polynomial",
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100),
        gamma = c(0.5, 1, 2, 3, 4), degree = c(0, 1, 2, 3, 4,
            5)), tunecontrol = svm_tune)

# Best model
best_poly_svm <- svm_poly_cv$best.model
best_poly_svm
```

```

Call:
best.tune(method = svm, train.x = Species ~ ., data = iris_train,
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100),
        gamma = c(0.5, 1, 2, 3, 4), degree = c(0, 1, 2, 3, 4,
            5)), tunecontrol = svm_tune, kernel = "polynomial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  10
     degree:  3
      gamma:  0.5
     coef.0:  0

Number of Support Vectors:  12
```

We can see that the optimal value of d is 3. This means that our separating hyperplane is a cubic function. Again, we can visualize our best performing model on the training data.

```r
# Plot best model against two variables
plot(best_poly_svm, iris_train, Petal.Width ~ Petal.Length, slice = list(Sepal.Width = 3,
    Sepal.Length = 4))
```



And evaluate it on the query set…

```r
# Predict on query set
svm_poly_y_query <- predict(best_poly_svm, iris_query)

# Show misclassified values in a confusion matrix
poly_query_conf_mat <- table(predict = svm_poly_y_query, truth = iris_query$Species)
poly_query_conf_mat
```

```
            truth
predict      setosa versicolor virginica
  setosa          8          0         0
  versicolor      0         13         0
  virginica       0          2         7
```

```r
# Report misclassification error
svm_poly_error <- 1 - mean(svm_poly_y_query == iris_query$Species)
svm_poly_error
```

```
[1] 0.06666667
```

Interestingly, we get the exact same results using the polynomial kernel.

Neural Net (Sigmoid) Kernel

Finally, we will use the Neural Net kernel, which is just the sigmoid function, as our final model comparison. We will need to tune both the cost and γ parameters using cross-validation, just as we have done with other kernels.

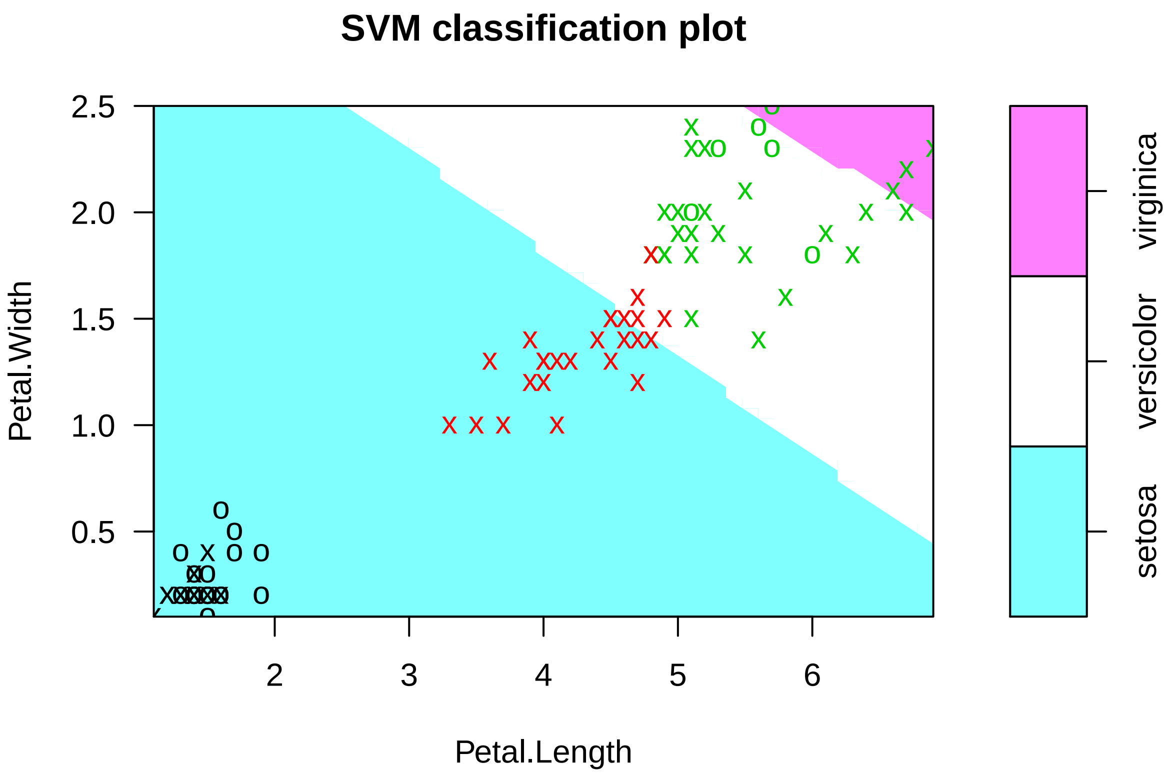
```r
# Set seed for reproducibility
set.seed(123)
```

```r
# Tune model using cross-validation
svm_nn_cv <- tune(svm, Species ~ ., data = iris_train, kernel = "sigmoid",
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100),
        gamma = c(0.5, 1, 2, 3, 4)), tunecontrol = svm_tune)

# Best model
best_nn_svm <- svm_nn_cv$best.model
best_nn_svm
```

```

Call:
best.tune(method = svm, train.x = Species ~ ., data = iris_train,
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100),
        gamma = c(0.5, 1, 2, 3, 4)), tunecontrol = svm_tune,
    kernel = "sigmoid")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  sigmoid
       cost:  0.1
      gamma:  1
     coef.0:  0

Number of Support Vectors:  65
```

In this model, we see that the optimal values of cost and γ are 0.1 and 1 respectively. Let us visualize this model on the training data.

```r
# Plot best model against two variables
plot(best_nn_svm, iris_train, Petal.Width ~ Petal.Length, slice = list(Sepal.Width = 3,
    Sepal.Length = 4))
```



And evaluate on the query set…

```r
# Predict on query set
svm_nn_y_query <- predict(best_nn_svm, iris_query)

# Show misclassified values in a confusion matrix
nn_query_conf_mat <- table(predict = svm_nn_y_query, truth = iris_query$Species)
nn_query_conf_mat
```

```
            truth
predict      setosa versicolor virginica
  setosa          8          0         0
  versicolor      0          7         0
  virginica       0          8         7
```

```r
# Report misclassification error
svm_nn_error <- 1 - mean(svm_nn_y_query == iris_query$Species)
svm_nn_error
```

```
[1] 0.2666667
```

We can see that the neural net kernel performs the worst out of all of the kernels we have tried thus far. As always, it is important to try many different types of models on unseen data in order to evaluate which ones perform best.

Best performing model on test set

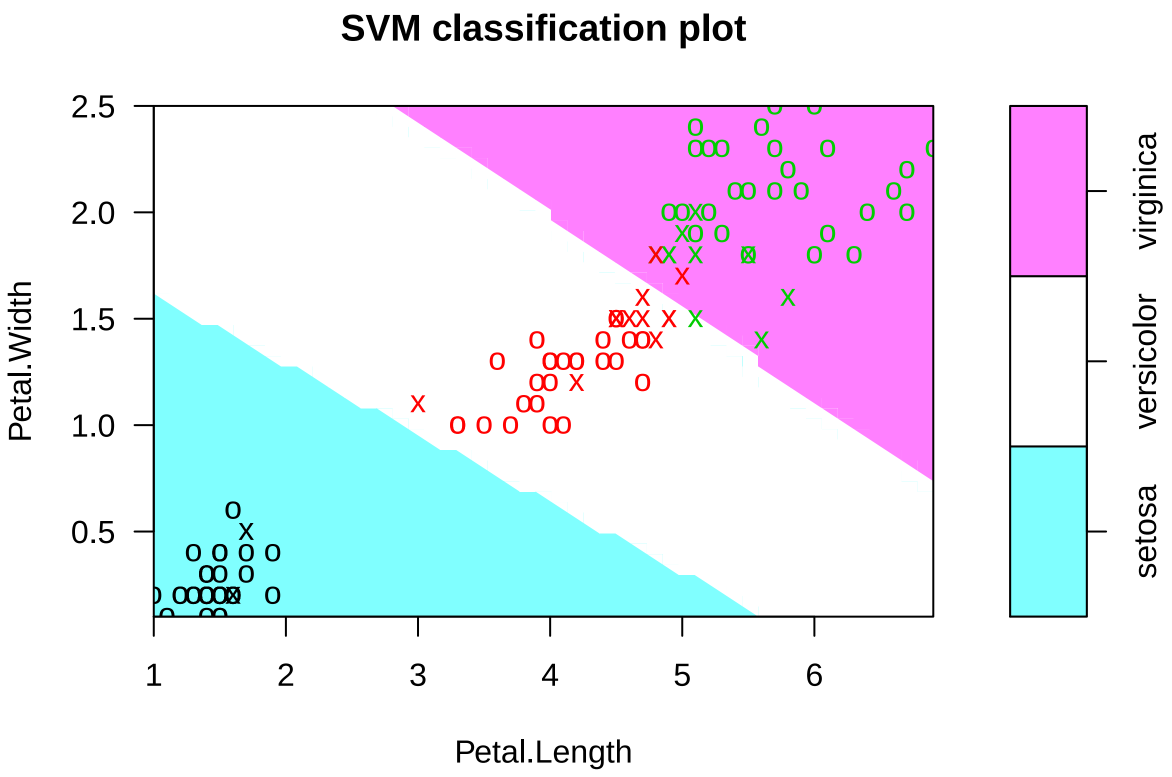
Since the linear, radial basis, and polynomial model all performed the same on the query set, we can elect to use the simplest model and select the linear SVM. We retrain the model on the combined training and query set, and then report our final test accuracy on our final chunk of unseen data. First let's see what our final model looks like.

```r
# Combine training and query sets
iris_train_query <- data.frame(rbind(iris_train, iris_query))

# Set seed for reproducibility
set.seed(123)

# Tune finalmodel using cross-validation
final_svm_cv <- tune(svm, Species ~ ., data = iris_train_query,
    kernel = "linear", ranges = list(cost = c(0.001, 0.01, 0.1,
        1, 5, 10, 100)), tunecontrol = svm_tune)

# Best model
best_final_svm <- final_svm_cv$best.model
best_final_svm
```

```

Call:
best.tune(method = svm, train.x = Species ~ ., data = iris_train_query,
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)),
    tunecontrol = svm_tune, kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.25

Number of Support Vectors:  26
```

And visualizing...

```r
# Plot best model against two variables
plot(best_final_svm, iris_train_query, Petal.Width ~ Petal.Length,
    slice = list(Sepal.Width = 3, Sepal.Length = 4))
```



And reporting our final test error…

```
# Predict on query set
svm_y_test <- predict(best_final_svm, iris_test)

# Show misclassified values in a confusion matrix
test_conf_mat <- table(predict = svm_y_test, truth = iris_test$Species)
test_conf_mat
```

```
            truth
predict      setosa versicolor virginica
  setosa          9          0         0
  versicolor      0          9         1
  virginica       0          1        10
```

```
# Report misclassification error
svm_test_error <- 1 - mean(svm_y_test == iris_test$Species)
svm_test_error
```

```
[1] 0.06666667
```

We can see that our linear SVM predicts the classes of the test set with very high accuracy (~93.3%). This makes sense since we know our data is linearly separable, but in real life cases it is important to use a rigorous data science pipeline that assesses and compares many different types of models to ensure that the final reported test accuracy is as high as possible.

References

Hastie, Trevor, et al. The Elements of Statistical Learning Data Mining, Inference, and Prediction. Springer, 2009.