6. Support Vector Machines

Whereas a lot of our previous algorithms arose out of statistics, the support vector machine arose from computer science. Vladimir Vapnik developed the ideas for the support vector machine in his PhD thesis in the 1960s in his native Russia. Unfortunately, the computing power was not present there at that time to implement his ideas. Later he emigrated to the U.S. to work for Bell Labs. He submitted several papers to top conferences including NIPS but his papers were rejected. Vapnik's persistence paid off, he went on to win numerous awards including in 2017 the IEEE von Neumann Medal. As of this writing he has over 180K citations on Google Scholar.

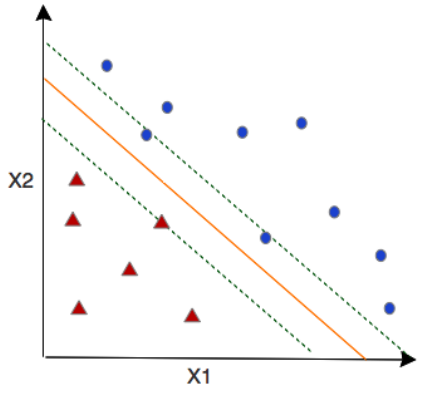

Figure 6.1: Separating Hyperplane

6.1 Overview

SVM is a very popular algorithm because it performs well in a variety of scenarios. The most common scenario is binary classification but it can also be extended to multiclass classification and even regression. The key ideas are summarized in Figure 6.1 where we see two classes, represented by red triangles and blue dots, divided by a separating hyperplane, represented by the orange line. Although visualized in 2 dimensions representing two predictors for illustrative purposes, we should think of the separating line as being a hyperplane in multidimensional space for the more common occurence of having multiple predictors. Notice that the separating line has margins on either side, in green. The goal is to find a margin that separates the classes which is not any wider or narrower than necessary. We see instances actually on the margin, these are called the *support vectors* from which the algorithm gets its name. As we see later, classification is simply a matter of determining which side of the margin an instance falls on, designated by class +1 and class −1.

6.2 SVM in R

One of the nice things about SVM is that it does multi-class classification. On the iris data set, we were able to do multiclass classification as shown below, with 95% accuracy. Notice the confusion matrix that was output by the table() function. Most of the observations are on the diagonal, only 4 observations out of the test set were misclassified. The implementation we are using in the example below is from the e1071 package.

Code 6.2.1 — SVM. Iris Data Set

```
library(e1071)

svm3 <- svm(Species~., data=train, kernel="linear",
        cost=10, scale=TRUE)

pred <- predict(svm3, newdata=test)
table(pred, test$Species)
pred         setosa versicolor virginica
  setosa         27          0         0
  versicolor      0         24         4
  virginica       0          0        24
mean(pred==test$Species)
[1] 0.9493671

# plot the support vectors
plot(svm3, test, Petal.Width ~ Petal.Length,
     slice = list(Sepal.Width = 3, Sepal.Length = 4))
```

Figure 6.2 shows a plot for the test data, but similar results would be seen using the train data or on the entire data set. The plot is a nice feature of the svm function. Note the color coding for the 3 classes. The limitations of this type of svm plot are that you can only plot 2 dimensions and the data must be quantitative.

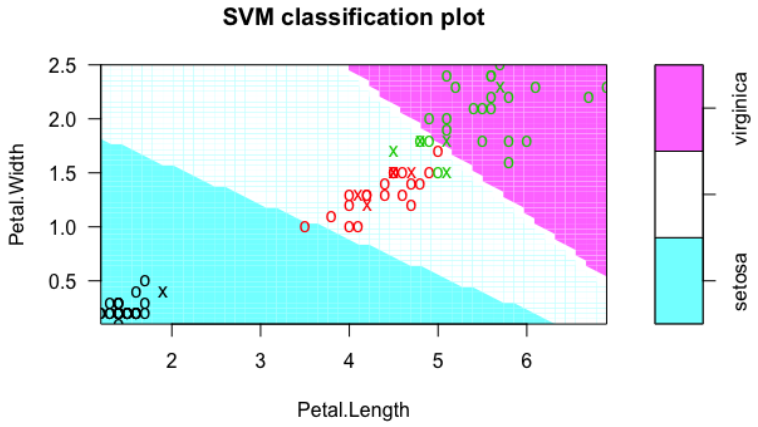

Figure 6.2: SVM Plot

6.3 The Algorithm

The standard svm uses a linear decision boundary defined by:

$$\mathbf{w}^T\mathbf{x} + b \tag{6.1}$$

To classify new observations, data will be assigned to the $+1$ class on one side of the decision boundary and to the -1 class on the other.

$$\hat{y} = sign(\mathbf{w}^T\mathbf{x}_{new} + b) \tag{6.2}$$

The decision bounary is defined by the orthogonal vector $\mathbf{w}$. For binary classifiers, w is assumed to point toward the positive class. The margin is the perpendicular distance from the decision boundary to the closest points (support vectors) on either side. The classification rule will be:

$$if\ (\mathbf{w}^T\mathbf{x}_{new} + b) > 0\ then\ y = +1 \qquad if\ (\mathbf{w}^T\mathbf{x}_{new} + b) < 0\ then\ y = -1 \tag{6.3}$$

Taking advantage of the fact that y will be $+1$ or -1 we can rewrite the two equations above as one:

$$y_i(\mathbf{w}\cdot\mathbf{x}_{new} + b) - 1 \geq 0 \tag{6.4}$$

Note that in the above, the equation will be equal to zero for support vectors.

We can visualize this solution geometrically. For a new point x_{new} we can visualize a vector from the origin to this point which we will call u. Is u on the positive or negative side? We can project u onto w to get the direction and length, and then classify according to this decision rule:

$$w\cdot u + b \geq 0 \tag{6.5}$$

If the above is true, it will be classified as the positive class, otherwise it will be classified as the negative class.

6.4 Mathematical Foundations

The goal is to learn the hyperplane from the training data, making the margin as wide as it can be, given the data. The width of the margin is the difference of two support vectors, one on each margin, normalized by the unit vector of w:

$$(x_+ - x_-) \cdot \frac{w}{||w||} \tag{6.6}$$

The positive support vector has width 1-b while the negative support vector has width 1+b. Subtracting these gives 2, so the quantity to optimize is:

$$max\ \frac{2}{||w||} \equiv\ max\ \frac{1}{||w||} \equiv\ min\ ||w|| \equiv\ min\ \frac{1}{2}||w||^2 \tag{6.7}$$

With the final step above selected for mathematical convenience. The quantity to minimize is constrained by the support vectors:

$$y_i(w \cdot x_i + b) - 1 = 0 \tag{6.8}$$

The above notation for the support vectors takes advantage of the fact that y is either positive or negative 1.

Lagrangian multipliers, denoted by alpha below, allow us to put together our optimization expression with the constraints:

$$L\ =\ \frac{1}{2}||w||^2\ -\ \sum_i \alpha_i \left[y_i(w \cdot x_i + b) - 1\right] \tag{6.9}$$

To find w, we take the partial derivative with respect to w and set to zero:

$$\frac{\partial L}{\partial w}\ =\ w - \sum_i \alpha_i y_i x_i = 0 \qquad \Rightarrow \qquad w\ =\ \sum_i \alpha_i y_i x_i \tag{6.10}$$

The equation for w above tells us that w is a linear sum of all the samples. For most samples, the alpha will be 0. The alpha will be non-zero only for those on the margin. Next we need to find b by taking the partial derivative of the Lagrangian equation with respect to b.

$$\frac{\partial L}{\partial b}\ =\ -\sum_i \alpha_i y_i = 0 \qquad \Rightarrow \qquad b = -\sum_i \alpha_i y_i \tag{6.11}$$

Now plug in the solution for w back into equation 6.0:

$$L = \frac{1}{2}(\sum_i \alpha_i y_i x_i)(\sum_j \alpha_j y_j x_j)\ -\ (\sum_i \alpha_i y_i x_i)(\sum_j \alpha_j y_j x_j)\ -\ \sum_i \alpha_i y_i b\ +\ \sum_i \alpha_i \tag{6.12}$$

We can simplify the above by combining terms and eliminating the next-to-last term because b is a constant and the alphas will be zero.

$$L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{6.13}$$

So the optimization depends on the dot product of pairs of samples. This turns out to be a quadratic optimization problem which is solved by existing software solutions.

6.4.1 Slack Variables and the C Hyperparameter

If the data cannot be neatly separated by the margins, it may be possible to get good results by letting a few observations fall on the wrong side of the margin. These are called **slack variables**. The term C controls how much impact slack variables are allowed to have on the determination of the optimal w. Larger values of C result in larger margins and more support vectors. Smaller values of C will have lower bias, higher variance than larger values.

$$\epsilon_i \geq 0, \sum_n \epsilon_i \leq C \tag{6.14}$$

Allowing for slack variables can also generalize better to test data because it will not have overfit the training data. It is always desireable to perform a little worse on the training data if performance on test data can be improved. Previously we defined support vectors as instances on the margin. When we allow for slack variables, the support vectors become instances either on the margin or on the wrong side of the margin given their class. C is the "cost" hyperparameter that can be optimized experimentally as shown in the following Check Your Understanding. The SVM algorithm has some hyperparameters to tune so we need to set aside data for that. For a linear SVM we need to specify the cost parameter. The optimal hyperparameters are usually found by experimenting on a validation data set that is separate from the test set. The following code shows how to divide your data into 3 sets, train, test, and validation. After setting a seed as usual, we set up a variable, spec, to hold our specifications for the percent of data we want in each group. Then we sample using the cut() function. We have not used this function before; cut(x) divides the range of x into intervals and codes the values in x according to which interval they fall. The leftmost interval corresponds to level one, the next leftmost to level two and so on. We also see the use of the cumsum(), cumulative sum function, which returns a vector whose elements are the cumulative sums.

```
# train-test-validate split
set.seed(1234)
spec <- c(train=.6, test=.2, validate=.2)
i <- sample(cut(1:nrow(df),
                nrow(df)*cumsum(c(0,spec)), labels=names(spec)))
train <- df[i=="train",]
test <- df[i=="test",]
vald <- df[i=="validate",]
```

Check Your Understanding 6.1 — SVM on the Titanic Data. Try the following:

- Load the titanic3.csv data (available on the github) into variable df.
- Subset the data to only columns pclass, survived, sex, age. Make sure that survived and pclass are factors.
- Check for NAs. Replace NAs in the age column with the average age. Use complete.cases() to get rid of remaining NAs.
- Divide into 60% train, 20% test, and 20% validation as shown in the code above.

Now let's try SVM:

- Build an SVM model on the train set using cost=10 and kernel="linear".
- Evaluate on the test set. What is your accuracy?
- The cost parameter is a hyperparameter that determines how much impact slack variables are allowed to have. Larger values of cost result in larger margins. Smaller values for cost will make the model move toward lower bias, higher variance. Try various values for cost as shown in the code below.

```
# experiment with cost hyperparameter
set.seed(1234)
tune.out <- tune(svm, survived~., data=vald, kernel="linear",
         ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune.out)
best_model <- tune.out$best.model
summary(best_model)
pred <- predict(best_model, newdata=test)
acc_svm2 <- mean(pred==test$survived)
```

Some explanation of the code above:

- Look at the summary of tune.out. It tells you the best cost was 0.01.
- Notice that you don't have to rerun the model with this best cost. The algorithm will extract the best model for you.
- This best model is then used to make predictions on the test data. The results should be a little better than the first svm.
- Note that we tuned the hyperparameters on the validation set. If we tuned it on the training set, we would probably overfit that data and not do well on the test data. If we tuned it on the entire data set that would be against good principles because we are letting the algorithm see test data. By tuning on a validation set we avoid these problems. We had about 260 observations for the validation set. If we had more data we would have more confidence in our hyperparameters.

6.5 Kernel Methods and the Gamma Hyperparameter

The discussion above assumed a linear decision boundary. However, data is often not linearly seperable. Kernel functions can be used to map the linear function to another form so that the data is linearly seperable. In Figure 6.3 below we see on the first line data that is not linearly separable. The bottom part of the figure illustrates the mapping of the data to a higher dimension so that it is now linearly separable. This visualizes a polynomial kernel.

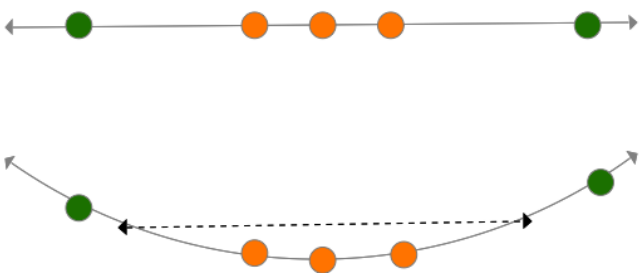

Figure 6.3: Mapping to a Polynomial Kernel

Figure 6.4 illustrates a radial kernel in which the decision boundary is roughly circular. Points are classified according to whether they are inside or outside this radial boundary.

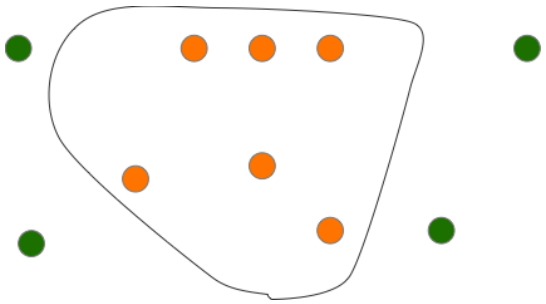

Figure 6.4: Mapping to a Radial Kernel

Recall that our liner svm relied on the inner product of observations $x_i \cdot x_j$. This can be combined with a non-linear kernel function $K(x_i \cdot x_j)$. The two most common kernels are the polynomial kernel: $(x_i \cdot x_j)^d$ and the radial kernel: $exp(-\gamma(x_i \cdot x_j)^2)$. The gamma parameter controls the bias-variance tradeoff. A larger gamma can overfit, having low bias and high variance. A smaller gamma could have higher bias and lower variance.

6.6 SVM Regression

SVM linear, polynomial, and radial kernels can be used for classification or regression. The first example in this chapter was for classification, now we turn our attention to a regression problem. The full notebook online shows the Boston housing data set in the `MASS` package. We want to predict median home value for a neighborhood given various parameters such as the number of rooms, and indicators of the nature of the neighborhood. First we perform linear regression and get an mse=37.5 and correlation of the predicted versus target values of .8. The root mse is 6.12 which tells us we would be off on the house estimate by about 6,120 dollars.

Next we try SVM regression on the same train and test data, shown in the code below. The svm() function looks similar to other R functions with some additional parameters. Recall from the discussion about slack variables, that the C variable controls how much we will allow variables to violate the decision boundary. The value C, cost=10, was chosen arbitrarily but later we will discuss a cross-validation technique to find the best C. In the svm() call below we selected a linear kernel and chose not to scale the data. When

should you scale the data? If you have one predictor with very large values compared to the other predictors, it will dominate calculating the distance between the separating hyperplane and the support vectors. In this example, setting scale=TRUE resulted in slightly worse performance, however the algorithm ran much faster. The correlation for the SVM regression was .76, a little worse than linear regression, and the mse was 45, also worse than linear regression. The linear regression model slightly outperformed the SVM model, and the lm() model also has the advantage of being highly interpretable.

Code 6.6.1 — SVM Regression. Boston Housing.

```
library(e1071)
svm_fit1 <- svm(medv~., data=train, kernel="linear", cost=10,
     scale=FALSE)
summary(svm_fit1)
svm_pred1 <- predict(svm_fit1, newdata=test)
cor(svm_pred1, test$medv)
mse_svm1 <- mean((svm_pred1 - test$medv)^2)
```

In the code above we just arbitrarily picked a cost=10 value. There is a tune() function in R that tries to find optimal hyperparameters for the svm using a grid search. This involves trying all the suggested vaues for the hyperparameters in a cross-validation scheme. The output of summary() is shown below the code. Look down the column to find the lowest error. It seems to be at cost=0.1.

Code 6.6.2 — Tuning Hyperparameters. Finding the best C.

```
tune_svm1 <- tune(svm, medv~., data=train, kernel="linear",
          ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune_svm1)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 28.22127

- Detailed performance results:
   cost    error dispersion
1 1e-03 45.37743   28.59339
2 1e-02 29.26508   18.06280
3 1e-01 28.22127   16.40897
4 1e+00 28.42638   16.52989
5 5e+00 28.42807   16.52849
6 1e+01 28.43127   16.53240
7 1e+02 28.43401   16.55991
```

Notice in the output that there really is not much difference in the error from cost=0.1 to cost=100. The cost was lower at 0.1 and when we used this value in the R notebook online, the performance was similar, with a slightly worse correlation but a slightly better mse than the first svm model. However, it is still not as good as the linear regression model. You can extract the best model suggested by tune() with `tuned$best.model`.

Cost is a hyperparameter. What is a hyperparameter? Whereas the parameters of a model are internal to the model and can be estimated from data, the *hyperparameters* are external to the model and are often optimized experimentally.

6.6.1 Radial Kernel SVM

Next we try a radial kernel SVM. The radial kernel has an additional hyperparameter, gamma, which controls the shape of the hyperplane boundary. Smaller gammas give sharper peaks in higher dimensions whereas larger gammas give more rounded peaks. Higher gammas allow the decision point to be influenced highly by points close to the decision boundary, making peaks and valleys in the decision bounary. Higher gammas may overfit. This suggests that smaller gammas tend toward lower bias and higher variance and that larger gammas increase bias and lower variance.

Code 6.6.3 — SVM Regression. Radial Kernel.

```
svm_fit2 <- svm(medv~., data=train, kernel="radial",
      cost=1, gamma=1, scale=FALSE)
svm_pred2 <- predict(svm_fit2, newdata=test)
cor(svm_pred2, test$medv)
mse_svm2 <- mean((svm_pred2 - test$medv)^2)
```

The online notebook demonstrates trying the tune() function again to get the best hyperparameters but in this example, the svm model got close but never performed better than the linear regression model. One limitation in this example is that the hyperparameters were tuned on a small validation set of about 100 observations.

Check Your Understanding 6.2 — Kernel Methods on the Titanic Data. Try the following:

- Continuing with the same data as Check Your Understanding 6.1, build a model with kernel="polynomial", gamma=1, and cost=1.
- Evaluate on the test data and compare to the linear model.
- Build a model with kernel="radial", gamma=1 and cost=1.
- Evaluate on the test data and compare to the linear and polynomial models.
- Try variations of parameters on the validation data, as shown in the code below.
- Evaluate on the test data, using the best model, and compare to the earlier models.

```
set.seed(1234)
tune.out <- tune(svm, survived~., data=vald, kernel="radial",
                 ranges=list(cost=c(0.1,1,10,100,1000),
                             gamma=c(0.5,1,2,3,4)))
```

■

6.7 Summary

Because only a few instances determine the margins, SVM is robust to observations that are far from the hyperplane. SVM classifiers share this in common with logistic regression. For points away from the hyperplane, the loss function is zero in SVM and very small for logistic regression. This is why logistic regression and SVM often give similar results. When classes are well separated, SVM tends to perform better than logistic regression. When the classes overlap, logistic regression may perform better. SVM can be used for both regression and classification. Furthermore, either SVM regression or SVM classification can use linear, polynomial, or radial kernels. This makes SVM a highly versatile algorithm that will perform well in a variety of applications.

6.7.1 New Terminology in this Chapter

The new terminology in this chapter mainly involves aspects of the SVM algorithm:

- separating hyperplane
- margins
- support vectors
- slack variables
- kernels: linear, polynomial, radial
- hyperparameters

6.7.2 Quick Reference

The e1071 package provides the implementation of SVM we used in this chapter. The svm() function performs classification if the target is a factor and regression otherwise. The kernel can be linear, polynomial, or radial. We can also choose to scale the data or not with the SCALE parameter.

Reference 6.7.1 SVM Classification or Regression

```
# classification or regression depends on target type
svm_model <- svm(formula, data=train, kernel="linear",
    cost=10, scale=TRUE)
```

Reference 6.7.2 Divide into Train/Test/Validation Sets

```
set.seed(1234)
spec <- c(train=.6, test=.2, validate=.2)
i <- sample(cut(1:nrow(df),
                nrow(df)*cumsum(c(0,spec)), labels=names(spec)))
train <- df[i=="train",]
test <- df[i=="test",]
vald <- df[i=="validate",]
```

Reference 6.7.3 SVM Tuning

```
set.seed(1234)
# tuning a linear svm
tune_svm1 <- tune(svm, medv~., data=vald, kernel="linear",
          ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune_svm1)
# tuning a radial svm
tune.out <- tune(svm, medv~., data=vald, kernel="radial",
                 ranges=list(cost=c(0.1,1,10,100,1000),
                             gamma=c(0.5,1,2,3,4)))
# extract best model
tune.out$best_model
```

6.7.3 Labs

Problem 6.1 — SVM Classification on the Glass Data. Try the following:

- Load the glass data from package mlbench.
- Check for NAs and run str() on the data.
- Divide the data into 60/20/20 train/test/validate data.
- Run naive Bayes on the training data and evaluate on the test data.
- Run an SVM linear model on the training data and evaluate on the test data.
- Tune the hyperparameters on the validation set, and evaluate on the best model.
- Run an SVM polynomial model on the training data and evaluate on the test data.
- Run an SVM radial model on the training data and evaluate on the test data.
- Tune the hyperparameters on the validation set, and evaluate on the best model.
- Compare all the models you created in terms of accuracy on the test data.

Problem 6.2 — SVM Advantages and Disadvantages. A friend is comparing a naive Bayes model and a logistic regression model and getting fairly good results. Would you recommend SVM? What questions would you ask about the data before making your recommendation?

6.7.4 Going Further

This chapter used the svm() function in package `e1071`. This function is an interface into the well-known C++ implementation by Chang and Lin called libsvm. Read more about their work here: `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`

An in-depth tutorial on SVM for pattern recognition by Burges and Burges from the *Data Mining and Knowledge Discovery* journal is available here: https://www.microsoft.com/en-us/research/