CLASSIFICATION METHODS FOR SUPPORT VECTOR MACHINES

JULIA ANDRONOWITZ
B.S., Mathematics

May 2023

Abstract

The purpose of this thesis is to give an introduction to the concept of Support Vector Machines in Machine Learning. We will first outline the idea of classification, including the maximal margin classifier and the support vector classifier. Examples of each will be given using programming languages such as R and Python. Then, we will move onto support vector machines and the use of kernels with example data. We will implement the techniques previously described in a real data set and finish by discussing applications of SVMs and examining the documentation of the support vector machine modules in R and Python.

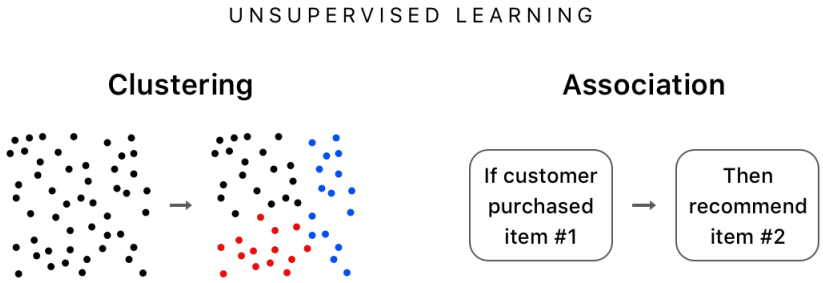
University of Connecticut
Department of Mathematics
Advisor: Dr. Jeremy Teitelbaum

1 INTRODUCTION

The first discoveries regarding machine learning date back to the 1950s. Known as the "Turing Test", mathematician Alan Turing attempted to discern if a computer could fool a human into thinking it is also a human [7]. In the years to follow, computer codes turned into learning programs that evolved the more times the code was run. In 1957, the first neural network was created which simulated a human brain's thought process. Ten years later, a basic pattern recognition algorithm was created called the "nearest neighbor" algorithm. Come the 1990's, accessibility to computers and advances in computers has exponentially increased. Scientists begin using a data-driven approach as large amounts of data are available, which allows computer algorithms to analyze data and learn from their results. Fast forward a few decades and we now have deep learning, survival analysis, and unsupervised learning.

Machine learning algorithms fall into one of two categories: supervised or unsupervised learning. Essentially, unsupervised learning arises when each observation does not have an associated response. This allows the machine to find certain patterns among the data and draw conclusions. Types of unsupervised learning include clustering and association. Clustering problems involve grouping the data based on the features. It is unknown how many groups are present in the data set when we begin, and so we rely on the machine to distinguish between groups. For example, we may want to group different animals based on their hunting/gathering methods and what they consume. Perhaps some animals tend to graze, others tend to scavenge, and others hunt. The algorithm would try to split the data into these three groups. In contrast, association rule learning problems focus on generalized trends between the groups. These types of trends are widely applicable to large portions of the data. For example, if one animal tends to graze on grass and other plants, the machine might suggest that another animal who grazes on grass will also eat other plants. Another common example is that people who buy one product are likely to buy another product [3].

In contrast, supervised learning uses data that is well-labeled to teach our model and either infer or predict. Inference problems aim for a better understanding of the relationship between the response variable and the features. On the other hand, prediction problems relate to developing a model that accurately fits the response variable to the predictors. Each observation in a data set has associated predictors and a response variable. Predictors are the input variables and can go by a variety of names such as variables, independent variables or features [5]. Predictors usually



Source: Data Driven Investor

correspond to the variables $x_1, x_2, ..., x_n$ where each x_n is an input variable. The response variable is the dependent variable or what we are trying to measure, usually denoted as y. Let's take a look at some penguin data, shown in Table 1.

Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Delta 15 N (o/oo)	Delta 13 C (o/oo)	Species
50.2	18.7	198	3775	9.39305	-24.25255	Gentoo
39.5	17.4	186	3800	8.94956	-24.69454	Adelie
44.9	13.8	212	4750	8.11238	-26.20372	Chinstrap
52.2	17.1	228	5400	8.36701	-25.89834	Chinstrap
50.8	19	210	4100	9.98044	-24.68741	Gentoo
42.5	20.7	197	4500	8.67538	-25.13993	Adelie

Table 1: Penguin Data

In this example, the culmen length, culmen depth, flipper length, body mass, Delta 15 N and Delta 13 C are the predictors. Based on these variables, we want the model to predict which species the penguin is. So, each body measurement is a predictor and the species is the response variable. In supervised learning, we often have training and testing groups for our data. Training and test groups are a common practice in machine learning to both classify existing data points and assess the underlying accuracy of the model on known data. Training data is data used to teach our model how to estimate the response variable. Once the model is trained, we can apply the algorithm to the testing data which the model has not previously seen. Since we already have the response variable for this set in the original data, we can compare

the model's predicted y-values, denoted $\hat{y}$, with the actual y-values. In machine learning, it is common to see the data split into about 80% training and 20% testing data, but these values can be manually specified in the code [2]. Suppose we collect the data for the ten students at the end of the year. Then, we have all their test scores for the year. We take 8 students and train the model based on these observations. Then, we use the remaining two students as the test set.

Supervised learning can be categorized into either regression or classification problems. Generally, this has to do with the fact that data is either qualitative (categorical) or quantitative (numerical). Regression typically uses a quantitative response variable. In regression, we aim to create a model that uses features to accurately predict the response variable. We may want to see how a population's access to clean drinking water impacts their average life expectancy. Linear regression can be used in this case to see whether a correlation between the two variables exist and how strong that correlation is. We can also look at how accurate the model is at predicting the response variable as well as investigate whether the relationship is linear or non-linear. On the other hand, classification is common with a quantitative response variable. In these types of problems, the data is grouped into specific categories based on the given features. The model then is able to predict the category of an unknown data point. Similarly, we can also use tools to predict the accuracy of the model and determine if the relationship is linear or non-linear.

Developed in the 1990s, support vector machines are an approach to classification problems now widely used among data scientists. In the following paper, we will discuss what a support vector machine is, how one can be implemented among a data set and practical applications of SVMs.

2 HYPERPLANES

We first begin by defining a hyperplane. Specifically, a hyperplane is "a flat affine subspace of dimension $p-1$" in a p-dimensional space [5]. In the two-dimensional space, a hyperplane looks like a straight line bisecting the data points into two distinct groups. We call this data linearly separable, as the data can be separated into two distinct groups by the hyperplane. In three dimensions, we see the formation of a plane. In higher dimensions, it can be hard to visualize the hyperplane, but the concept still applies.

Mathematically, a hyperplane in $\mathbb{R}^2$ is the equation $0 = \beta_0 + \beta_1 x_1 + \beta_2 x_2$. Note that this is the equation of a one-degree polynomial, or a straight line. In higher dimensional spaces, a hyperplane has the form

$$0 = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

We can generalize the equation of a hyperplane to be

$$f(x) = \beta_0 + \beta \cdot x$$

where $f(x) = 0$ and β is a non-zero vector $\beta = (\beta_1, \beta_2, ..., \beta_p)$ in $\mathbb{R}^p$ in which the dot product of β and x is computed. For any $x = (x_1, x_2)^T$ where the equation holds, we say the point is on the hyperplane. However, points that do not lie on the hyperplane will have

$$0 \neq \beta_0 + \beta \cdot x$$

Moreover, the points that have $0 > \beta_0 + \beta \cdot x$ will lie on one side of the hyperplane while the points $0 < \beta_0 + \beta \cdot x$ will lie on the other. In this, we can see the concept of linearly separability. In a data set that is linearly separable and each observation is in one of two distinct groups, the hyperplane will bisect the data points in such a way that every point in one group is greater than zero while every point in the alternate group is less than zero.

To examine a hyperplane in two dimensions, we generate two distinct groups of data using *R*.

```
n <- 200
x1 <- matrix(rnorm(n*2, 1,0.3), ncol=2)
x2 <- matrix(rnorm(n*2,-1,0.3), ncol=2)
x <- rbind(x1,x2)
```

We first set $n = 200$. This will be the length of our data set. Then, we use the function **rnorm** to generate a set of data points from the normal distribution with mean 1 and standard deviation 0.3. The matrix function transforms the 400 randomly generated points into a [200 x 2] matrix. The same is done for a second group, instead with mean -1 and standard deviation 0.3. The **rbind** function combines these two groups into one vector, namely x.

```
plot(x,xlab="",ylab="")
abline(h=0,v=0,col="gray",lty="dotted")
abline(a=0,b=-1,col="black",lty="dashed")
```

Now, we plot the function. The **abline** functions add lines at $x = 0$ and $y = 0$ as well as add one such separating hyperplane. Figure 2 shows the output.

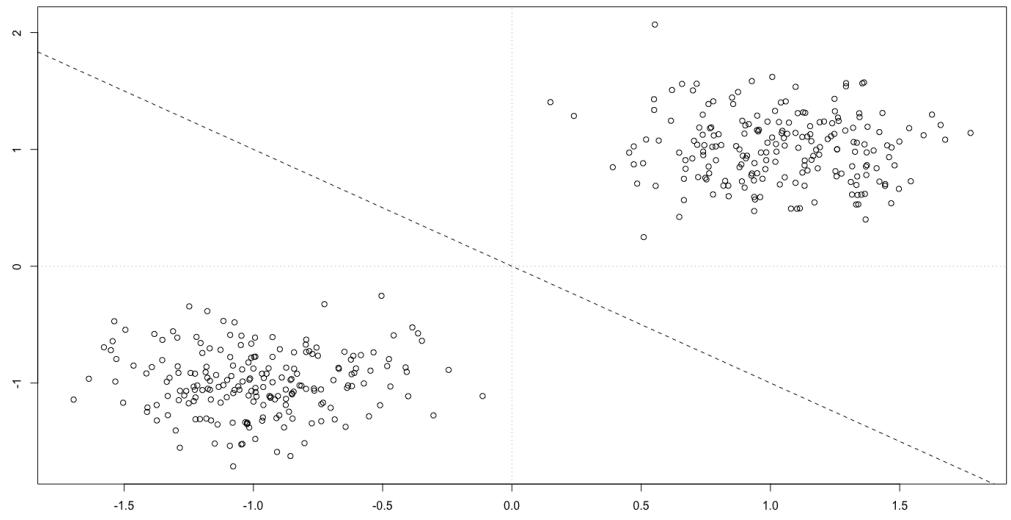


Figure 2: Randomly Generated Clustered Data
with Separating Hyperplane in $\mathbb{R}^2$

We see that there is a clear separation between the data points and a hyperplane that bisects the data. The equation of the hyperplane in this example is defined by $y = -x$. In the form $f(x) = \beta_0 + \beta \cdot x$, we have $f(x) = 0$, $\beta_0 = 0$, and $\beta = (1, 1)$ to give $0 = 0 + x_1 + x_2$ or $x_1 = -x_2$.

Notice that we can rotate the line slightly in either direction and still have separating hyperplane. So long as the line does not pass through the points that the gray line intersects, there exist infinitely many such separating hyperplanes. An example is shown in red in Figure 3.

Similarly, in a three-dimensional space the hyperplane partitions the data. A plane separates the set into two distinct groups. An example can be generated with Python.

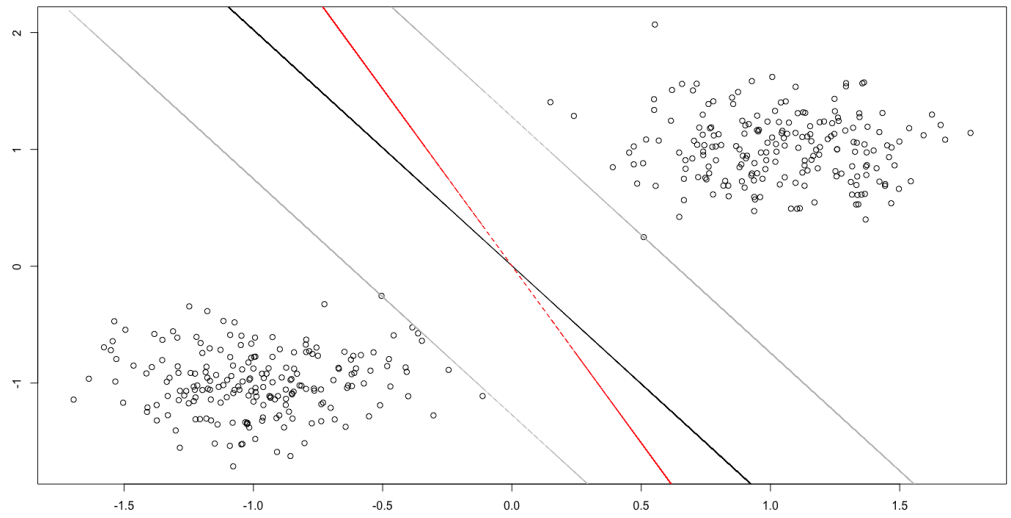


Figure 3: Randomly Generated Clustered Data with Multiple Hyperplanes in R^2

```
data1a = np.random.normal(-1,0.3,size=(10,1))
data1b = np.random.normal(-0.5,0.3,size=(10,1))
data1c = np.random.normal(0,0.1,size=(10,1))

data2a = np.random.normal(1,0.3,size=(10,1))
data2b = np.random.normal(0.5,0.3,size=(10,1))
data2c = np.random.normal(0,0.1,size=(10,1))
```

These functions set up our two data sets. The a, b, c values correspond to the x, y, z coordinates for each set. A randomly generated sample from the normal distribution is used. We then create the graph by setting up the axes and writing the equation for the hyperplane that will bisect the data before plotting the points.

```
x = np.linspace(-1, 1, 10)
y = np.linspace(-1, 1, 10)
```

```
x, y = np.meshgrid(x, y)
eq = 1 * x + 0.15 * y

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(x, y, eq, color='green',alpha=0.3)

ax.scatter(data1a,data1b,data1c, color='black')
ax.scatter(data2a,data2b,data2c, color='black')

plt.show()
```

The output is shown in Figure 4.

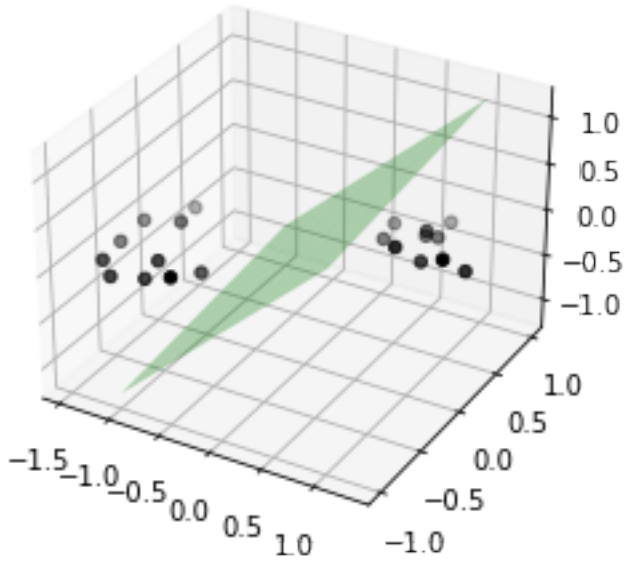


Figure 4: Randomly Generated Clustered Data
with Separating Hyperplane in $\mathbb{R}^3$

In this example, we see one set of points is to the left of the hyperplane while the other lies on the right. The equation of the hyperplane is $z = x + 0.15y$. In the form $f(x) = \beta_0 + \beta \cdot x$, we have $f(x) = 0$, $\beta_0 = 0$, and $\beta = (1, 0.15, -1)$ to give $0 = 0 + x_1 + 0.15x_2 - x_3$ or $x_3 = x_1 + 0.15x_2$. Like in the two-dimensional case,

there is an infinite number of separating hyperplanes in three dimensions. By slightly rotating the plane shown in Figure 4 in either direction, we can visualize the concept of numerous other planes that would bisect the same data set.

3 MAXIMAL MARGIN CLASSIFIER

The use of hyperplanes to distinguish between two classes is central to the maximal margin classifier. Also called the optimal margin classifier, this method aims to find the separating hyperplane that maximizes the distance between the two sets of points. That is, if the plane were to rotate slightly in either direction, it would be closer to the set of data points. Our depiction of the maximal margin classifier draws upon those presented in [5] and [9].

In the above example, we can classify each set of points to determine what side of the hyperplane they lie on. So, if $y_i = 1$ then $0 > \beta_0 + \beta \cdot x$ and if $y_i = -1$ then $0 < \beta_0 + \beta \cdot x$. Each y_i becomes the class label for that observation. Class labels are the identifiers for the data points and distinguish which class the observation belongs to. Introducing the following starred lines to our code produces colors in the graph according to the respective group:

```
n <- 200

x1 <- matrix(rnorm(n*2, 1,0.3), ncol=2)
x2 <- matrix(rnorm(n*2,-1,0.3), ncol=2)
x<- rbind(x1,x2)

y <- c(rep(1,n), rep(-1,n)) #**

plot(x,xlab="",ylab="",col=(3-y)) #**
abline(h=0,v=0,col="gray",lty="dotted")
abline(a=0,b=-1,col="black",lty="dashed")
```

The fifth line assigns a vector of 200 negative ones followed by 200 positive ones to the variable y. Combined with the adjusted plot function containing the color option $(3 - y)$, the first 200 points are plotted blue and the last 200 points are plotted red.

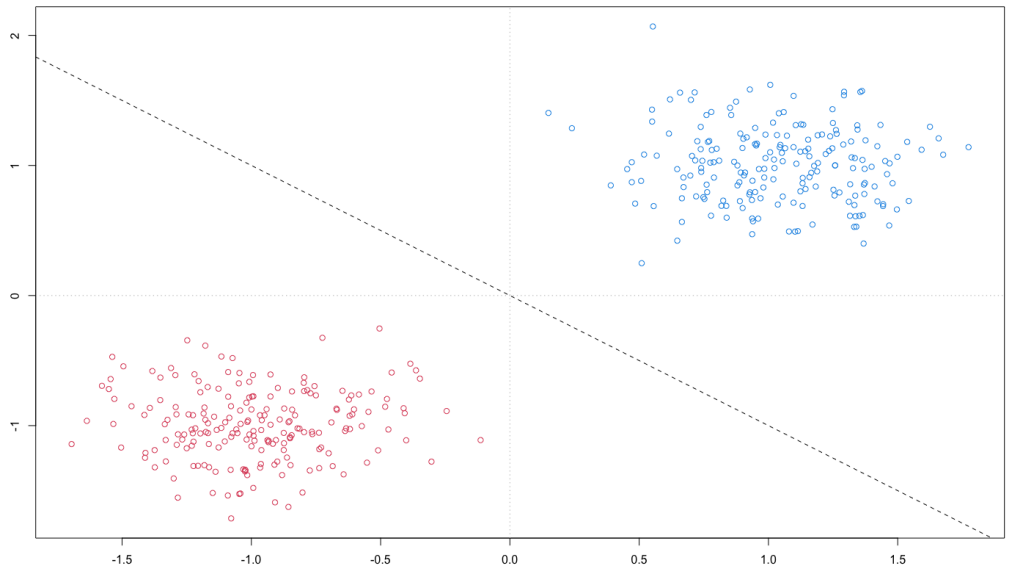


Figure 5: Randomly Generated Data with Associated Class Label Coloring

This gives out two groups shown in Figure 5. We need to find the closest blue and red points such that shifting the hyperplane in any such way results in a smaller margin for either group. These points are known as the support vectors, as they are the only points that end up formulating the hyperplane. At this point, we know that points with $0 > \beta_0 + \beta \cdot x$ will lie on one side of the hyperplane and have a class label y_i of 1, while the points $0 < \beta_0 + \beta \cdot x$ lie on the other and have a class label $y_i = -1$. Combining these two constraints, we can produce one inequality that correctly classifies each point

$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) > 0$$

Suppose we look at a few points with a separating hyperplane of $0 = f(x_1, x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$, pictured in Figure 6. We assume the point (a_1, a_2) in the set of the positive class labels is the closest point to the set of points with a negative class label. Then, to compute the perpendicular distance d from the point (a_1, a_2) to $f(x_1, x_2) = 0$, we first find a normal vector to f. Through calculus, the gradient of the hyperplane calculated by (dx_1, dx_2) gives the normal vector $\vec{n} = \langle \beta_1, \beta_2 \rangle$ to f.

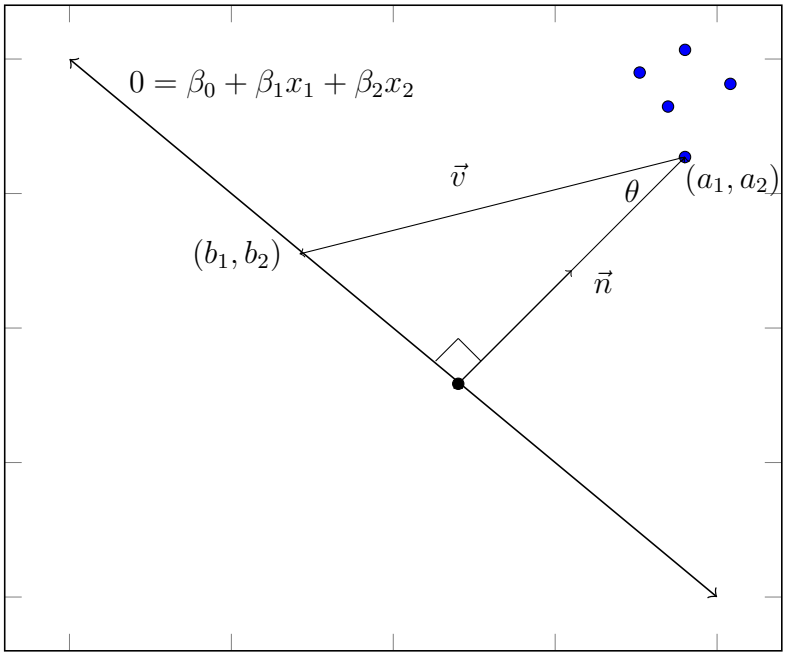


Figure 6: Diagram of Hyperplane and Support Vector

Let (b_1, b_2) be any point on the hyperplane. Then $\vec{v} = (a_1 - b_1, a_2 - b_2)$. The distance d equals $||\vec{v}|| \cos\theta$ through trigonometry. Then we have

$$\begin{aligned}
\vec{v} \cdot \vec{n} &= ||\vec{v}||||\vec{n}|| \cos\theta \\
\vec{v} \cdot \vec{n} &= d \cdot ||\vec{n}|| \\
d &= \frac{\vec{v} \cdot \vec{n}}{||\vec{n}||} \\
&= \frac{(a_1 - b_1, a_2 - b_2) \cdot \langle \beta_1, \beta_2 \rangle}{\sqrt{\beta_1^2 + \beta_2^2}} \\
&= \frac{\beta_1 a_1 - \beta_1 b_1 + \beta_2 a_2 - \beta_2 b_2}{\sqrt{\beta_1^2 + \beta_2^2}} \\
&= \frac{\beta_1 a_1 + \beta_2 a_2 - (\beta_1 b_1 + \beta_2 b_2)}{\sqrt{\beta_1^2 + \beta_2^2}}
\end{aligned}$$

Since (b_1, b_2) lies on the hyperplane, we have $\beta_0 + \beta_1 b_1 + \beta_2 b_2 = 0$ and $\beta_1 b_1 + \beta_2 b_2 =$

$-\beta_0$. Thus,

$$\begin{aligned} d &= \frac{\beta_1 a_1 + \beta_2 a_2 - (\beta_1 b_1 + \beta_2 b_2)}{\sqrt{\beta_1^2 + \beta_2^2}} \\ &= \frac{\beta_1 a_1 + \beta_2 a_2 - (-\beta_0)}{\sqrt{\beta_1^2 + \beta_2^2}} \\ &= \frac{\beta_1 a_1 + \beta_2 a_2 + \beta_0}{\sqrt{\beta_1^2 + \beta_2^2}} \end{aligned}$$

Note that the numerator is exactly the equation f. In fact, there exist infinitely many such functions, as we could simply multiply f by a constant. Suppose we have the equivalent hyperplane $0 = C\beta_1 a_1 + C\beta_2 a_2 + C\beta_0$ for any $C \in \mathbb{R}$. Then,

$$\begin{aligned} d &= \frac{C\beta_1 a_1 + C\beta_2 a_2 + C\beta_0}{\sqrt{(C\beta_1)^2 + (C\beta_2)^2}} \\ &= \frac{C(\beta_1 a_1 + \beta_2 a_2 + \beta_0)}{\sqrt{C^2(\beta_1^2 + \beta_2^2)}} \\ &= \frac{C(\beta_1 a_1 + \beta_2 a_2 + \beta_0)}{C\sqrt{\beta_1^2 + \beta_2^2}} \\ &= \frac{\beta_1 a_1 + \beta_2 a_2 + \beta_0}{\sqrt{\beta_1^2 + \beta_2^2}} \end{aligned}$$

Notice that we are left with the same distance d, even with a different equation for the hyperplane. By imposing the condition $\beta_1^2 + \beta_2^2 = 1$, we restrict the above to just one possible hyperplane. So, we are left with

$$d = \frac{f(a_1, a_2)}{\sqrt{\beta_1^2 + \beta_2^2}}$$

We aim to find the greatest distance, or margin, by maximizing the coefficients to the function f of the closest point(s) to the hyperplane. Thus, we have the basic idea for the maximal margin classifier. We seek to maximize the margin M to find the best separating hyperplane given by

$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq M$$

provided that

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{1}$$

Though the maximal margin classifier is not a support vector machine quite yet, we may use the SVM module in the "e1071" library in R to determine the equation of the line.

```
library(e1071)
dat <- data.frame (x = x, y = as.factor (y))
svmfit <- svm ( y~., data = dat , kernel="linear", cost = 1000,
            scale = FALSE)
plot(svmfit,dat)
```

Once downloading the package, we assign the x and y variables to one dataframe. We convert the y variable into a factor since class label is a binary value. In the **svm** function, the dataframe we just defined is the basis of the calculations. We assign the kernel as linear with a cost of 1000, though other options exist for non-separable data and non-linear decision boundaries which we will get to in later pages. We plot the function and see the output in Figure 7.

As we see, the separating hyperplane is illustrated clearly albeit not pictured exactly linear. Details of the **svm** function can be found using the following:

```
summary(svmfit)
svmfit$index
svmfit$rho
svmfit$coefs
```

The **str** function tells us that **svmfit** is a list of 30 variables, some of which we defined in the function and others that were calculated. Particularly of interest are the **index**, **rho**, and **coefs** variables, which give the output in Figure 8. The **rho** variable is the intercept of the separating hyperplane, and the **coefs** variable is an array of the coefficients of the supporting vectors. The **index** variable tells us the

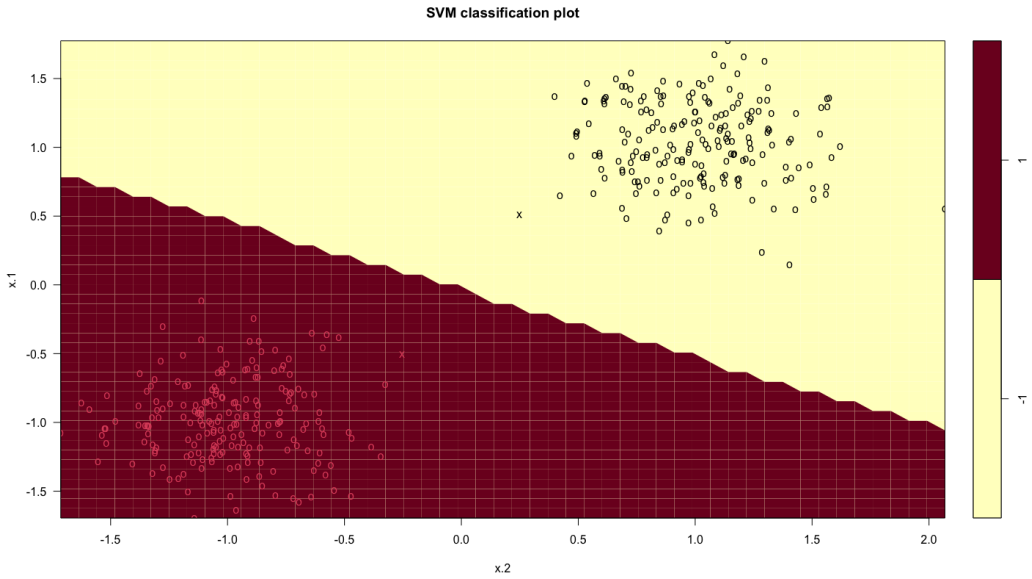


Figure 7: Maximal Margin SVM Module Output in *R*

```
> svmfit$rho
[1] 0.002291883
```

```
> svmfit$index
[1]  92 309
```

```
> svmfit$coefs
             [,1]
[1,]  1.561796
[2,] -1.561796
```

Figure 8: Selected Output of SVM Function

indices of the observations that the classifier depends on, or the support vectors. In this case, it is data points 92 and 309. If we were to rotate the hyperplane in any direction, the margin from these two points to the black line would inevitably become smaller. Visually, we can see these two points as those that the supporting hyperplanes pass through in Figure 9.

To find the equation of the line, we use the hyperplane function as outlined in [10]:

```
hyperplane <- function(P,data,x,z=0) {
  alphas <- -1*P$coefs
  svs <- data[P$index,]
  c <- P$rho - z
```

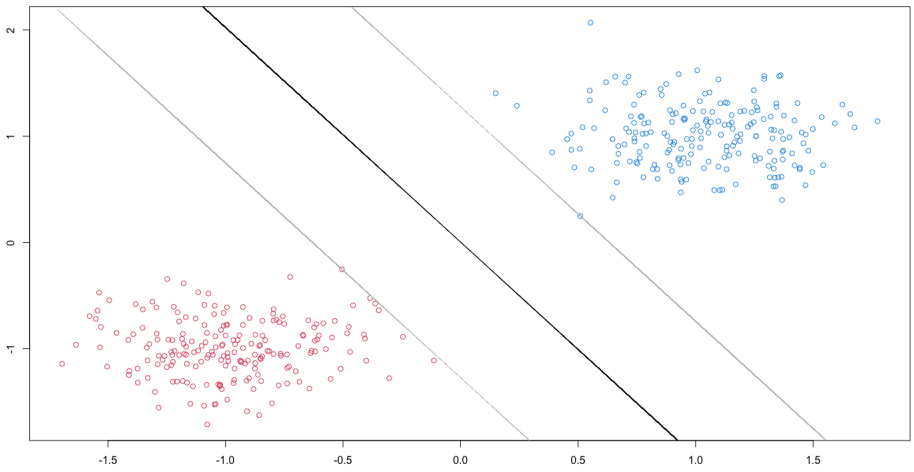


Figure 9: Maximal Margin Classifier Output in *R*

```
  a <- sum(t(alphas)*data[P$index,][1])
  b <- sum(t(alphas)*data[P$index,][2])
  (-c-a*x)/b
}
```

In this function, **P** is the **svm** function which we labeled *svmfit*. The **data** is our dataframe containing the observations and class labels. **x** is the column vector of observations. We set **z**=0 for now, but this will later be shown to move the hyperplane vertically by a set constant.

Inside the function, the **alphas** are assigned to the coefficients of the hyperplane and multiplied by negative one. The **svs** variable takes the index of the support vectors and finds the exact point values in the data. **c** refers to the intercept and can be adjusted by the variable **z**. The variables **a** and **b** take the x and y coordinates, respectively, of the support vectors and multiply by their corresponding coefficient, as calculated by taking the transpose of the **alphas** variable, then summed. In Python, the **hyperplane** function looks slightly different due to the differences in the corresponding module, but the concept remains the same [10].

```
def hyperplane(P,x,z=0):
"""Given an SVC object P and an array of vectors x, computes the
↪  hyperplane wx+b=z"""
```

```
alphas = P.dual_coef_
svs = P.support_vectors_
c = P.intercept_[0]-z
a = np.sum(alphas.T*svs,axis=0)[0]
b = np.sum(alphas.T*svs,axis=0)[1]
return (-c-a*x)/b
```

The R code for the plot referenced in Figure 9 is below and pulls information from the **hyperplane** function.

```
plt0 <- hyperplane(svmfit,dat,x,0)
plt1 <- hyperplane(svmfit,dat,x,1)
plt2 <- hyperplane(svmfit,dat,x,-1)

plot(x,xlab='',ylab='')
lines(x,plt0,col='black')
lines(x,plt1,col='gray',lty='dashed')
lines(x,plt2,col='gray',lty='dashed')
```

4 SUPPORT VECTOR CLASSIFIER

What happens when data is not linearly separable? This occurs when the two groups of data have overlapping points. In this case, we cannot form a separating hyperplane as some points will be classified on the wrong side of the line. For non-linearly separable data, we introduce the support vector classifier. Much like the maximal margin classifier, we form a hyperplane between the groups of points in a data set. This classification method is also known as the soft-margin classifier since some points will not be classified on the correct side of the hyperplane unlike the maximal margin classifier. We will introduce the topic of costs, which essentially allow a certain number of observations to be classified incorrectly, then discuss the code in R and Python.

The following code in R can be used to plot non-linearly separable data with randomly generated points, shown in Figure 10. We use the **set.seed()** function to replicate the randomly generated data. The process is the same as with linearly separable data, except we now narrow the gap between the means of the two groups which

allows some overlap. Notice that we can no longer draw a straight line bisecting the two groups.

```
# setting initial number of observations
n <- 200

# calculating x-variable using two randomly generated sets with
↪  overlap
set.seed(100)
svcdatax1 <- matrix(rnorm(n*2, 0.5,0.3), ncol=2)
svcdatax2 <- matrix(rnorm(n*2,-0.5,0.3), ncol=2)
svcdatax <- rbind(svcdatax1,svcdatax2)

# assigning labels to both classes
svcdatay <- c(rep(-1,n), rep(1,n))

# plotting the data
plot(svcdatax, col=(3-svcdatay),xlab="",ylab="",pch=16)
```

We can add a line to visualize a possible hyperplane by running the following line of code after we have constructed the graph.

```
abline(0,-1,col="gray",lty="dashed")
```

As we can see, there is no possible linear hyperplane that separates the data. Some blue points are intermingled with the red and vice versa. To still construct a machine that attempts to classify the data, we introduce the topic of costs. In the maximal margin classifier, we aimed to find the greatest distance (margin) between the closest points of the two sets. Since there is no concrete margin in non-linearly separable data, we must introduce what is called a slack variable, denoted as ϵ_i for each observation x_i. For variables that are correctly classified, the corresponding observation's slack variable is 0. For points that are incorrectly classified, the slack variable takes on a value greater than 0 and incurs some cost for the maximization function. Since the slack variable is essentially a measure of how far a point lies on the wrong side of the margin, we constrict the sum of all slack variables to be less than a specified

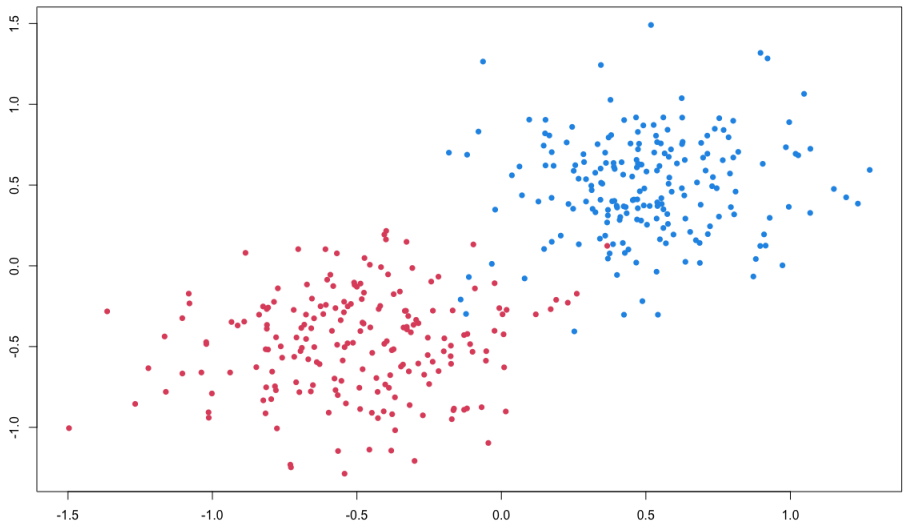


Figure 10: Randomly Generated Non-Linearly Separable Data

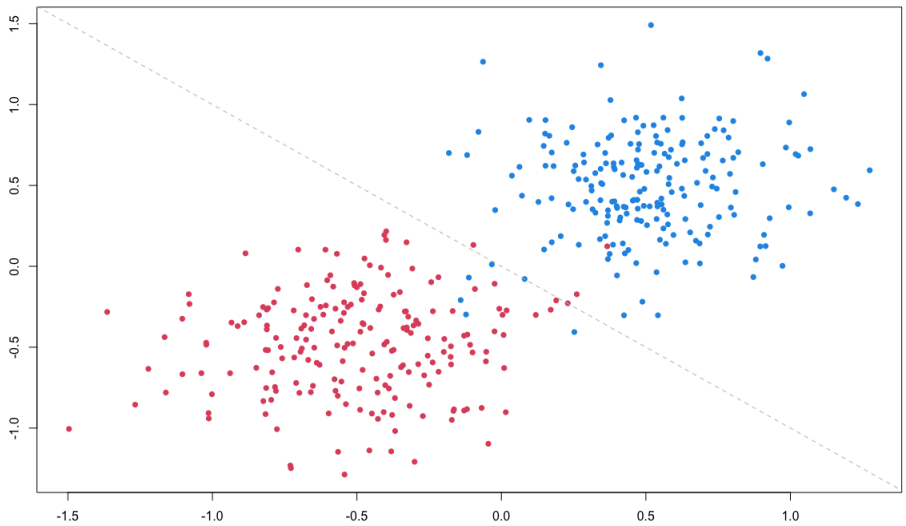


Figure 11: Randomly Generated Non-Linearly Separable Data With Line

cost C. As we will see later, a user-defined cost can result in over- or under-fitting of the model. For a support vector classifier, the maximization of M is subject to

$$\sum_{j=1}^{p} \beta_j^2 = 1$$

$$\sum_{i=1}^{n} \epsilon_i \leq C, \epsilon_i \geq 0$$

Let us introduce the concept of inner products and their role in the support vector classifier as outlined in [5] and [4]. The inner products of two vectors is defined as

$$\langle a, b \rangle = \sum_{i=1}^{r} a_i b_i$$

for two r-vectors a and b. Following the work produced in [5], the support vector classifier can be rewritten as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

and the solution revolves around the inner products. Futhermore, it is known that each α_i is non-zero only for the support vectors. The support vector classifier is simplified even more, as we can ignore points that are not support vectors. The solution then becomes

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

provided that

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{2}$$

$$\sum_{i=1}^{n} \epsilon_i \leq C, \epsilon_i \geq 0 \tag{3}$$

where S is the set of indices of the support vectors.

We now implement a support vector classifier in *R*. Using the same **svm** module as in the maximal margin classifier, we can find the best hyperplane to classify the data given a specified cost, in this case a value of 1000.

```
# creating data frame
svcdata <- data.frame (x = svcdatax, y = factor(svcdatay))

# computing svm function
svmfit <- svm (y~., data = svcdata , kernel="linear", cost =
↪  1000, scale = FALSE)

# looking at statistics
summary(svmfit)
svmfit$index
svmfit$rho
svmfit$coefs

# plotting svm function graph
plot(svmfit,svcdata)

# calculating the hyperplane and supporting hyperplanes
plt0 <- hyperplane(svmfit,svcdata,svcdatax,0)
plt1 <- hyperplane(svmfit,svcdata,svcdatax,1)
plt2 <- hyperplane(svmfit,svcdata,svcdatax,-1)

# plotting the data with hyperplanes
plot(svcdatax,col=(3-svcdatay),xlab="",ylab="",pch=16)
lines(svcdatax,plt0,col='black')
lines(svcdatax,plt1,col='gray')
lines(svcdatax,plt2,col='gray')
```

This produces the result in Figure 12, graphically representing observations with slack variables greater than zero. Red points either inside the margin illustrated with the gray supporting hyperplanes or on the right side of the black hyperplane will have non-zero slack variables. Similarly, blue points inside the margin or on the left side of the hyperplane will have non-zero slack variables.

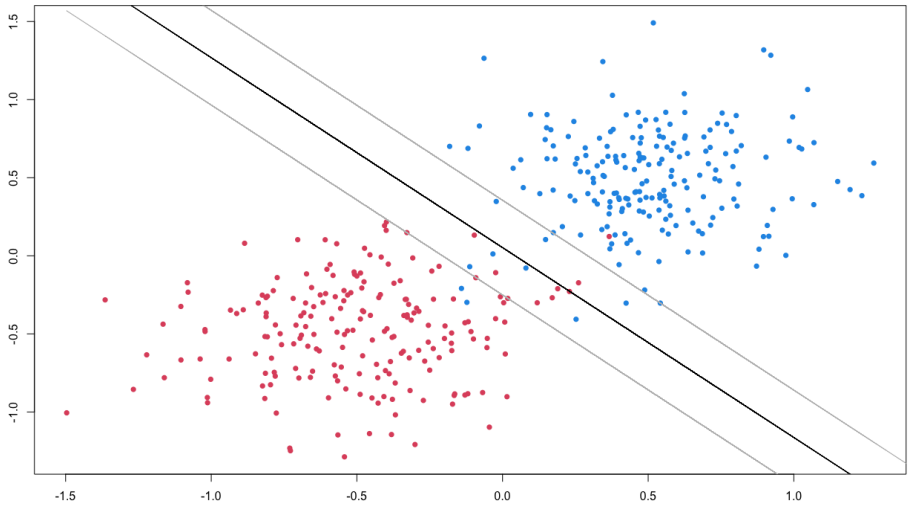


Figure 12: Randomly Generated Non-Linearly Separable Data With Line

Let us look at what happens with different values of the cost variable.

```
# using cost of 1
svmfit1 <- svm (y~., data = svcdata , kernel="linear", cost = 1,
↪  scale = FALSE)
summary(svmfit1)
    # results in 36 support vectors

# using cost of 100
svmfit100 <- svm (y~., data = svcdata , kernel="linear", cost =
↪  100, scale = FALSE)
summary(svmfit100)
    # results in 21 support vectors

# using cost of 1000
svmfit1000 <- svm (y~., data = svcdata , kernel="linear", cost =
↪  1000, scale = FALSE)
summary(svmfit1000)
    # results in 20 support vectors
```

```
# using cost of 10000
svmfit10000 <- svm (y~., data = svcdata , kernel="linear", cost =
↪  10000, scale = FALSE)
summary(svmfit10000)
        # results in 20 support vectors
```

We see that as the cost increases, there are fewer support vectors. That is, fewer observations are classified incorrectly. Shouldn't it be the opposite? We see in 3 that the sum of the slack variables has to be less than or equal to the cost. A higher cost allows for greater slack variables, meaning more observations can lie outside the margin. This discrepancy lies in the underlying code. In both the R and Python modules for the support vector machine, the cost is inversely proportional to the C we defined in 3.

The difference is a reference to the "dual problem", as there are two approaches to solving this problem. One approach, which we have previously outlined, fixes the variable C and the sum of β_j^2 in order to minimize the margin. We are left with solving

$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq M(1 - \epsilon_i)$$

subject to the conditions on β and ϵ. We could also solve this problem by switching our conditions and minimization as outlined in [4]. If we fix the margin to be 1, we aim to minimize $||\beta||$. The new problem becomes

$$\min ||\beta|| \text{ subject to } \begin{cases} y_i(\beta x_i + \beta_0) \geq 1 - \epsilon_i \forall i, \\ \epsilon_i \geq 0, \sum \epsilon_i \leq \text{constant} \end{cases} \tag{4}$$

which is equivalent to

$$\begin{cases} \min_{\beta,\beta_0} \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N} \epsilon_i \\ \text{subject to } y_i(\beta_i x_i^T + \beta_0) \geq 1 - \epsilon_i, \epsilon_i \geq 0 \end{cases} \tag{5}$$

The constant defined in 4 is replaced with the cost parameter of the Lagrange function in 5. This cost parameter is the same as the variable defined in both the R and Python **svm** modules.

We saw that choosing different costs affects the number of support vectors. A smaller cost results in a lower penalty for each misclassified data point. As such, the resulting margin is large. When the cost is larger, misclassified data points have a bigger impact on the model, so the resulting margin is smaller to minimize these points [11]. How does one go about determining the correct cost though? Luckily, the **svm** function can be used in a **tuning** function that tests various costs for the correlating model accuracy. We use the following code to determine the best model:

```
# tune the function to find the best model with various costs
tune <- tune(svm,y~.,data=svcdata,
             ranges = list(cost = 2^(-2:10)), tunecontrol =
              ↪  tune.control(sampling = "fix"))

tune$best.parameters
  # we see a cost of 0.25 gives the best model
tune$best.performance
  # this is the model error
```

Thus, we have produced the best support vector classifier for our data, illustrated in Figure 13.

5 SUPPORT VECTOR MACHINES

Previously, we have only worked with data involving two classes. The maximal margin classifier is used for linearly separable data, and the support vector classifier is used for non-linearly separable data. When there are more than two classes, a support vector machine is used. A linear hyperplane is not appropriate for multi-class data, so we introduce the concept of a kernel.

Recall the inner products from the support vector classifier. The kernel is a generalized representation of the inner product of two observations

$$K(x_i, x_i')$$

where K is some function that measures the similarity between two observations [5]. In the support vector classifier, the kernel is a linear function calculated using Pearson correlation. For non-linear data, this function K can be replaced with another form. Common quantities include

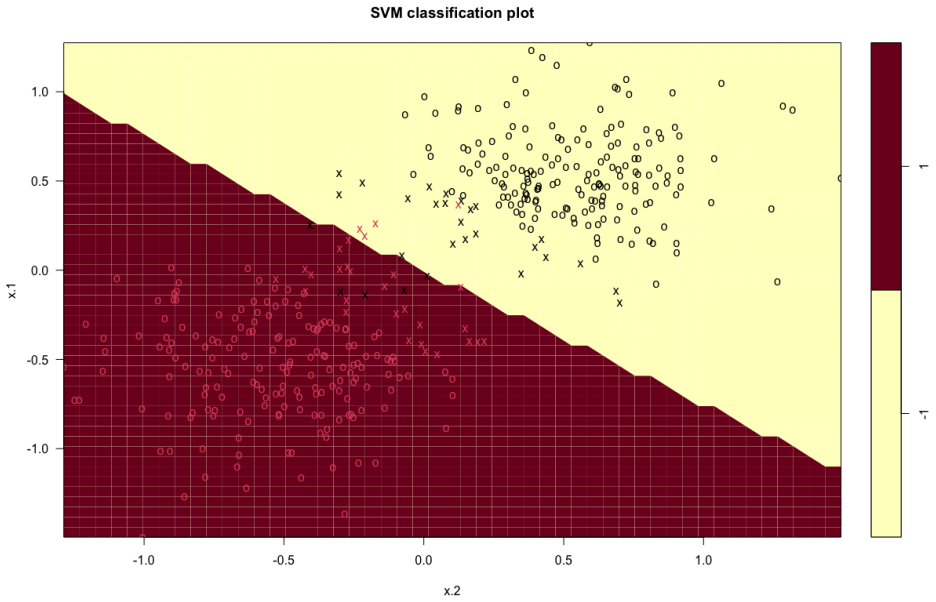


Figure 13: Final SVC Model with $C = 0.25$

K	Form
linear	$\sum_{j=1}^{p} x_{ij}x_{i'j}$
polynomial	$(1 + \sum_{j=1}^{p} x_{ij}x_{i'j})^d$
radial	$\exp\left(-\gamma \sum_{j=1}^{p}(x_{ij} - x_{i'j})^2\right)$

We will focus on the radial kernel. The radial kernel is a popular first choice as it performs well in multi-class data with a non-linear relationship. In addition to the cost parameter, we now have the gamma parameter. Common choices for the cost parameter range from 2^{-5} to 2^{15} and for the gamma parameters range from 2^{-15} to 2^3. One can input these ranges into the **tune** function to determine the best performing values for the model.

To implement this in R, we first create randomly generated multi-class data with the following code:

```
# setting initial number of observations in each set
n <- 200

# calculating x-variable using three randomly generated sets
↪  with overlap
set.seed(1000)

svmdatax1a <- matrix(rnorm(n, 0.25,0.3), ncol=1)
svmdatax1b <- matrix(rnorm(n,0.75,0.3), ncol=1)
svmdatax1 <- cbind(svmdatax1a,svmdatax1b)

svmdatax2 <- matrix(rnorm(n*2,-0.5,0.4), ncol=2)

svmdatax3a <- matrix(rnorm(n,1.5,0.3), ncol=1)
svmdatax3b <- matrix(rnorm(n,1,0.3), ncol=1)
svmdatax3 <- cbind(svmdatax3a,svmdatax3b)

svmdatax <- rbind(svmdatax1,svmdatax2,svmdatax3)

# assigning labels to both classes
svmdatay <- c(rep(1,n), rep(2,n), rep(3,n))

# plotting the data
plot(svmdatax, col=(11-svmdatay),xlab="",ylab="",pch=16)
```

Data can look like that in Figure 14.

What happens if we change the cost for a non-linear dataset? We try below with different values and compare.

```
# using different costs
svmfit.00001 <- svm (y~., data = svmdata , kernel="radial", cost
↪  = 0.00001, scale = FALSE)
svmfit.001 <- svm (y~., data = svmdata , kernel="radial", cost =
↪  0.001, scale = FALSE)
svmfit1 <- svm (y~., data = svmdata , kernel="radial", cost = 1,
↪  scale = FALSE)
```

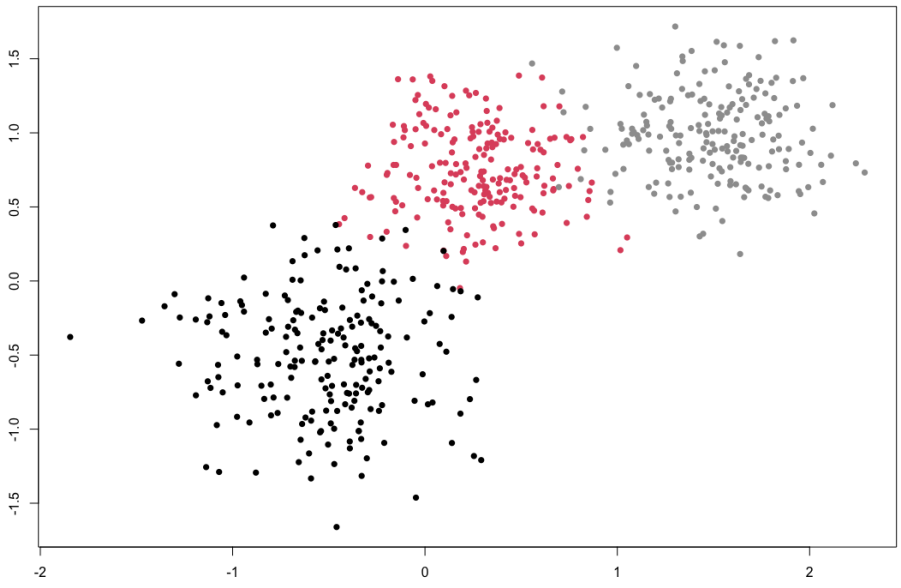


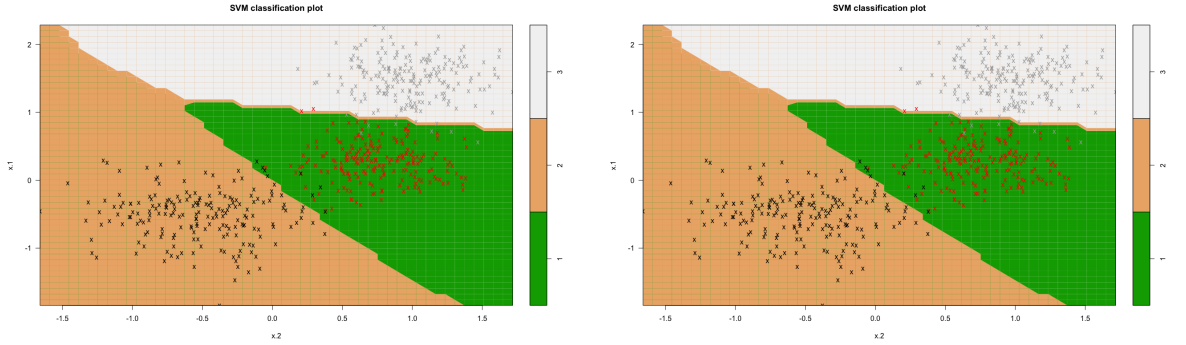
Figure 14: Randomly Generated Multi-Class Data

```
svmfit1000 <- svm (y~., data = svmdata , kernel="radial", cost =
↪  1000, scale = FALSE)
svmfit100000 <- svm (y~., data = svmdata , kernel="radial", cost
↪  = 100000, scale = FALSE)
svmfit10000000 <- svm (y~., data = svmdata , kernel="radial",
↪  cost = 10000000, scale = FALSE)

# plots of different costs
plot(svmfit.00001,svmdata,color.palette =
↪  terrain.colors,symbolPalette = c("red","black","darkgray"))
plot(svmfit.001,svmdata,color.palette =
↪  terrain.colors,symbolPalette = c("red","black","darkgray"))
plot(svmfit1,svmdata,color.palette = terrain.colors,symbolPalette
↪  = c("red","black","darkgray"))
plot(svmfit1000,svmdata,color.palette =
↪  terrain.colors,symbolPalette = c("red","black","darkgray"))
```

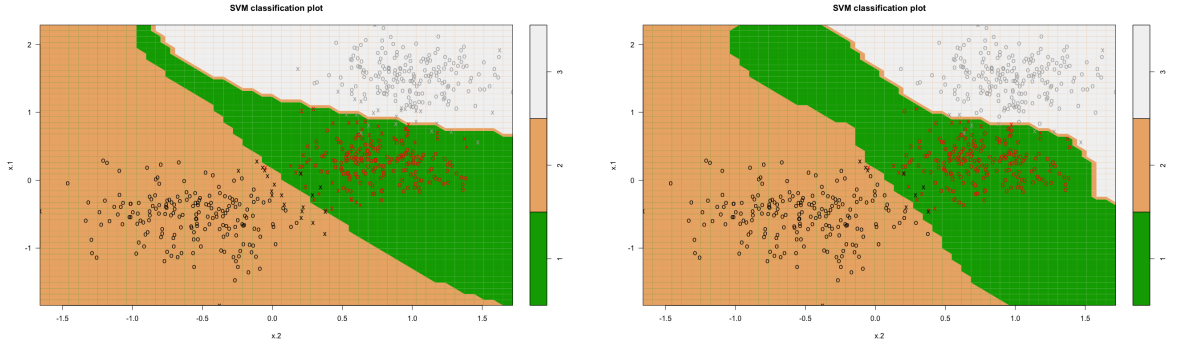
```
plot(svmfit100000,svmdata,color.palette =
 ↪  terrain.colors,symbolPalette = c("red","black","darkgray"))
plot(svmfit10000000,svmdata,color.palette =
 ↪  terrain.colors,symbolPalette = c("red","black","darkgray"))
```

The code produces the graphs in Figure 15. Initially, with a low cost every point is a support vector. In a dataset with 600 observations, 600 support vectors is not ideal. This makes sense, as a smaller cost means a lower penalty for each misclassified data point. As we increase the cost parameter, the number of support vectors decreases because the cost of being misclassified is greater. At the highest cost of 10,000,000 there are only 35 support vectors. We see a trade-off between the number of observations that influence the support vector machine and the associated cost. Zooming in, the support vectors in 15 are labeled with an "x".

$C = 0.00001$ $C = 0.001$



$C = 1$ $C = 1000$



$C = 100000$ $C = 10000000$

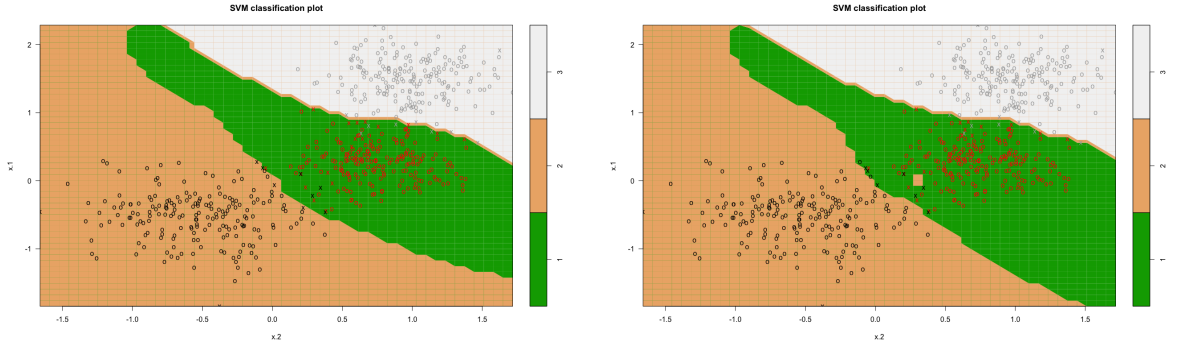


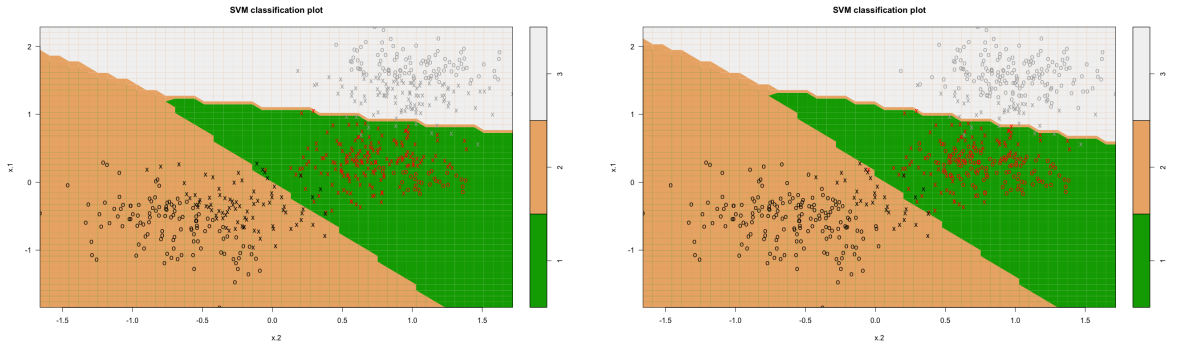
Figure 15: Experimentation with Different Costs in Non-Linear Data

What happens if we change the gamma value? We try below with different values and compare.

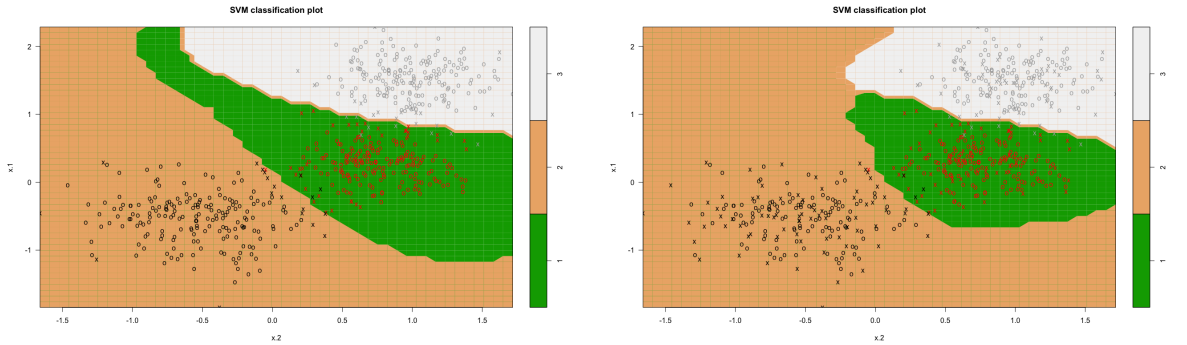
```
# using different gammas
svmfit.00001g <- svm (y~., data = svmdata , kernel="radial", cost
↪  = 1, gamma = 0.00001, scale = FALSE)
svmfit.001g <- svm (y~., data = svmdata , kernel="radial", cost =
↪  1, gamma = 0.001, scale = FALSE)
svmfit1g <- svm (y~., data = svmdata , kernel="radial", cost = 1,
↪  gamma = 1, scale = FALSE)
svmfit1000g <- svm (y~., data = svmdata , kernel="radial", cost =
↪  1, gamma = 1000, scale = FALSE)

# plots of different gammas
plot(svmfit.00001g,svmdata,color.palette =
↪  terrain.colors,symbolPalette = c("red","black","darkgray"))
plot(svmfit.001g,svmdata,color.palette =
↪  terrain.colors,symbolPalette = c("red","black","darkgray"))
plot(svmfit1g,svmdata,color.palette =
↪  terrain.colors,symbolPalette = c("red","black","darkgray"))
plot(svmfit1000g,svmdata,color.palette =
↪  terrain.colors,symbolPalette = c("red","black","darkgray"))
```

The code produces the graphs in Figure 16. From [8], the "gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'." With the lowest gamma value of 0.01, the support vector machine mostly consists of straight lines as the influence of individual support vectors reaches farther out. As we increase the gamma value, the machine becomes more specialized to the dataset. We see the gamma value of 1000 yields a model with 600 out of the 600 points being support vectors. Each point has only a small radius of influence, leading to an extremely localized model with margins closely following the class boundaries. Choosing an appropriate gamma value helps protect against under- and over-fitting the model.

$\gamma = 0.01$ $\gamma = 0.1$



$\gamma = 1$ $\gamma = 10$



$\gamma = 100$ $\gamma = 1000$

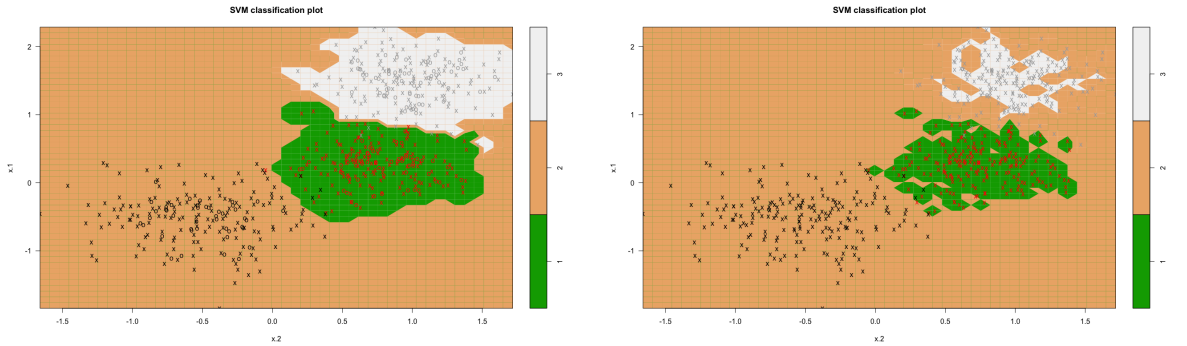


Figure 16: Experimentation with Different Gammas in Non-Linearly Separable Data

Using the tuning parameter, we can determine the best values for the cost and gamma parameters based on the lowest error. This helps increase the accuracy and validity of the model.

```
# using tune function to find the best model
tune <- tune(svm,y~.,data=svmdata,
    ranges = list(gamma = 2^(-2:10), cost = 2^(-2:10)),
    ↪  tunecontrol = tune.control(sampling = "fix"))

tune$best.parameters
tune$best.performance
```

The output is a gamma value of 0.5 and a cost of 0.25. Using these values in the **svm** function, we have

```
# creating data frame
svmdata <- data.frame (x = svmdatax, y = factor(svmdatay))

# computing svm function
svmfit <- svm (y~., data = svmdata , kernel="radial", cost =
↪  0.25, gamma=0.5, scale = FALSE)

# looking at statistics
summary(svmfit)
svmfit$index
svmfit$rho
svmfit$coefs

# plotting svm function graph
plot(svmfit,svmdata,color.palette = terrain.colors,symbolPalette
↪  = c("red","black","darkgray"))
```

The final model using the best parameters determined by the **tune** function is presented in Figure 17.

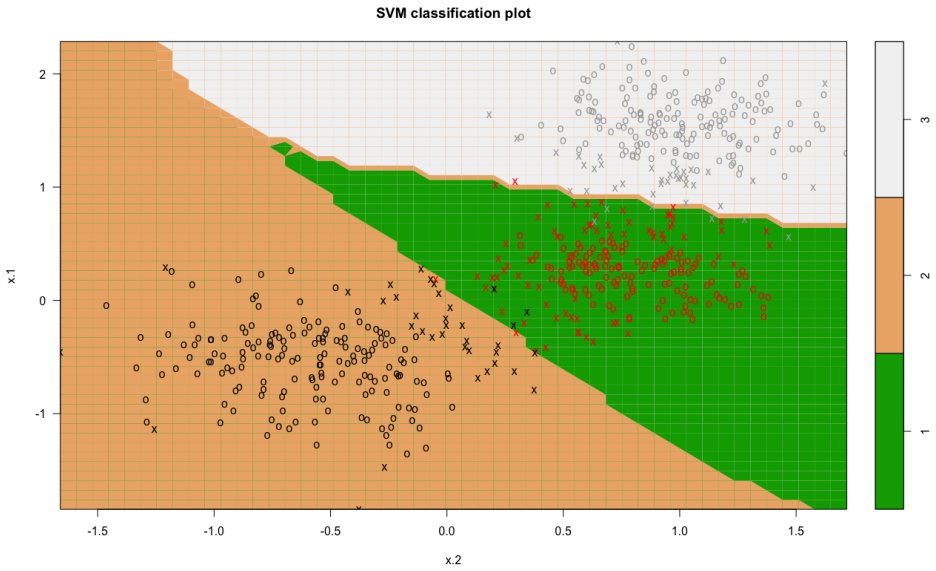


Figure 17: Final SVM Model with $C = 0.25$ and $\gamma = 0.5$

6 EXAMPLE

Let us now look at support vector machines in a real dataset. We will be using what is widely known as the Penguin Dataset, collected by Dr. Kristen Gorman and The Palmer Station, Antarctica LTER. This dataset was originally published in 2014 and includes 7 variables, namely: species, culmen length (mm), culmen depth (mm), flipper length (mm), body mass (g), island, and sex. We transition now into the Python SVM modules from the scikit-learn package to anaylze the Penguin data.

6.1 DATA EXPLORATION

We start with importing the necessary modules and functions, as well as some initial data exploration.

```
### importing modules
```

```
import numpy as np
from numpy.random import default_rng
from bokeh.io import output_notebook, show
from bokeh.plotting import figure
from bokeh.models import CategoricalColorMapper
from sklearn.svm import SVC
rng = default_rng(5)
output_notebook()
import pandas as pd

### importing functions

def hyperplane(P,x,z=0):
    """Given an SVC object P and an array of vectors x, computes
    ↪  the hyperplane wx+b=z"""
    alphas = P.dual_coef_
    svs = P.support_vectors_
    c = P.intercept_[0]-z
    a = np.sum(alphas.T*svs,axis=0)[0]
    b = np.sum(alphas.T*svs,axis=0)[1]
    return (-c-a*x)/b

def pts(P):
    """Given an SVC object P, returns the two closest points in
    ↪  the associated reduced convex hulls."""
    alphas = P.dual_coef_[0]
    svs = P.support_vectors_
    plus_indices = np.where(alphas>0)
    minus_indices = np.where(alphas<=0)
    alphas = alphas.reshape(-1,1)
    pluspt =
    ↪  np.sum(alphas[plus_indices]*svs[plus_indices],axis=0)/np.sum(alphas[plus_
    minuspt =
    ↪  np.sum(alphas[minus_indices]*svs[minus_indices],axis=0)/np.sum(alphas[min
    return pluspt, minuspt

### importing dataset
```

```
df = pd.read_csv('/Users/juliaandronowitz/Desktop/thesis/thesis/
↪  Data/penguins_size.csv')

### initial exploration of data

df.head()
df.info()
df.isna().sum()
df.describe()

### a user-defined function to view unique values

def rstr(df): return df.shape, df.apply(lambda x: [x.unique()])
rstr(df)
```

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	MALE
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	FEMALE
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	FEMALE

Figure 18: Output of df.head() Function

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   culmen_length_mm   342 non-null    float64
 3   culmen_depth_mm    342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                334 non-null    object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

```
species              0
island               0
culmen_length_mm     2
culmen_depth_mm      2
flipper_length_mm    2
body_mass_g          2
sex                 10
dtype: int64
```

Figure 19: Output of df.info() and df.isna().sum() Functions

```
((344, 7),
                      species                       island  \
 0  [Adelie, Chinstrap, Gentoo]  [Torgersen, Biscoe, Dream]

                                    culmen_length_mm  \
 0  [39.1, 39.5, 40.3, nan, 36.7, 39.3, 38.9, 39.2...

                                     culmen_depth_mm  \
 0  [18.7, 17.4, 18.0, nan, 19.3, 20.6, 17.8, 19.6...

                                    flipper_length_mm  \
 0  [181.0, 186.0, 195.0, nan, 193.0, 190.0, 180.0...

                                          body_mass_g                     sex
 0  [3750.0, 3800.0, 3250.0, nan, 3450.0, 3650.0, ...  [MALE, FEMALE, nan, .]  )
```

Figure 20: Output of rstr(df) Function

	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g
count	342.000000	342.000000	342.000000	342.000000
mean	43.921930	17.151170	200.915205	4201.754386
std	5.459584	1.974793	14.061714	801.954536
min	32.100000	13.100000	172.000000	2700.000000
25%	39.225000	15.600000	190.000000	3550.000000
50%	44.450000	17.300000	197.000000	4050.000000
75%	48.500000	18.700000	213.000000	4750.000000
max	59.600000	21.500000	231.000000	6300.000000

Figure 21: Output of df.describe() Function

From Figure 18, we can see the general structure of the dataset. In Figure 19, **df.info()** tells us the name, length, and type of each variable and **df.isna().sum()** tells us how many missing values are in each variable. Figure 20 shows us the unique values for each variable, which is useful in finding the species names. They are Adelie, Chinstrap, and Gentoo. Figure 21 describes the basic statistics for the numerical predictors. We see that the data looks clean with no outliers.

6.2 CLEANING THE DATA

We first want to drop the missing values. This can be acheived with

```
df = df.dropna()
```

Next, we are going to need labels for each of our penguin species. These labels will be used when we graph the functions. We also want to color these labels accordingly. Red will be Adelie, blue is Chinstrap, and green is Gentoo.

```
labels = []
x = 0
for i in df['species']:
    if i == 'Adelie':
        x = 0
    elif i == 'Chinstrap':
        x = 1
    elif i == 'Gentoo':
        x = 2
    labels.append(x)

colors = ['red','blue','green']
penguin_colors = np.array([colors[i] for i in labels])
```

We then want to create a **numpy** array of the desired variables which will work with the SVM module. We will be removing the species (the class label), island, and sex predictors.

```
data = df.drop(columns=['species','island','sex'])
data = data.to_numpy()
```

6.3 DATA ANALYSIS

We are then going to plot the raw data points for each set of variables. This will help us determine which predictors will be the most useful in the support vector machine.

```
# plot of culmen length and culmen depth
cl_cd=figure(title='Penguin Data: culmen length vs. culmen
↪  depth')
cl_cd.scatter(x=data[:,0],y=data[:,1],color=penguin_colors)
show(cl_cd)

# plot of culmen length and flipper length
cl_fl=figure(title='Penguin Data: culmen length vs. flipper
↪  length')
cl_fl.scatter(x=data[:,0],y=data[:,2],color=penguin_colors)
show(cl_fl)

# plot of culmen length and body mass
cl_bm=figure(title='Penguin Data: culmen length vs. body mass')
cl_bm.scatter(x=data[:,0],y=data[:,3],color=penguin_colors)
show(cl_bm)

# plot of culmen depth and flipper length
cd_fl=figure(title='Penguin Data: culmen depth vs. flipper
↪  length')
cd_fl.scatter(x=data[:,1],y=data[:,2],color=penguin_colors)
show(cd_fl)

# plot of culmen depth and body mass
cd_bm=figure(title='Penguin Data: culmen depth vs. body mass')
cd_bm.scatter(x=data[:,1],y=data[:,3],color=penguin_colors)
show(cd_bm)

# plot of culmen depth and body mass
fl_bm=figure(title='Penguin Data: flipper length vs. body mass')
fl_bm.scatter(x=data[:,2],y=data[:,3],color=penguin_colors)
show(fl_bm)
```

The associated plots are located in Figure 22.

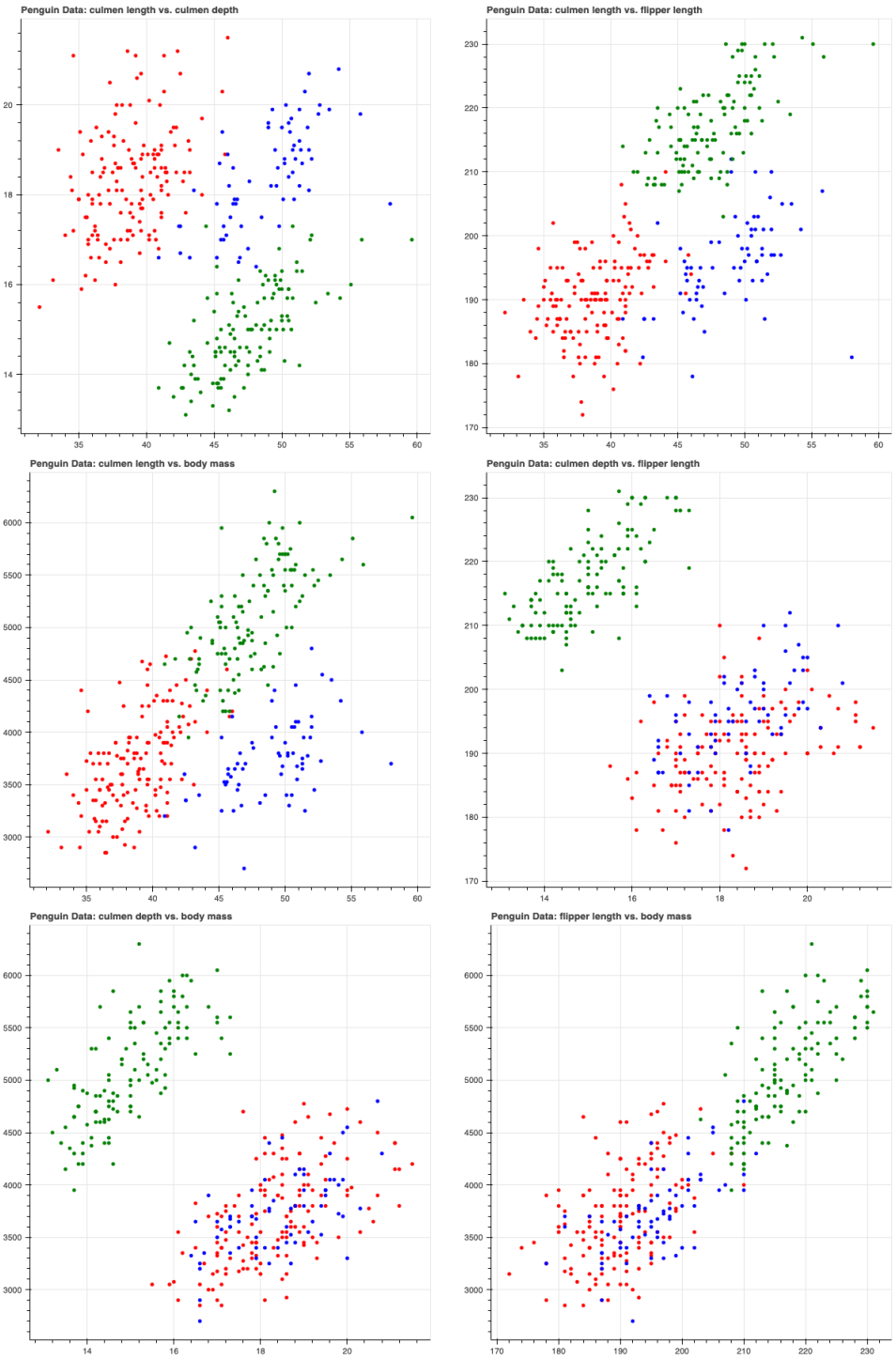


Figure 22: Raw Data Points of the Penguins Dataset

We see the top three plots look the best in separating the three species. The bottom three graphs have some overlap between the red and blue points, or Adelie and Chinstrap, with the green points, the Gentoo, more clearly distinguishable from the others. We will perform four different classifications to see which performs the best. Three will be using the variables associated with the top three graphs - culmen length vs. culmen depth, culmen length vs. flipper length, and culmen length vs. body mass - and the last will be a classifier with all four predictors.

6.4 CULMEN LENGTH VS. CULMEN DEPTH

```
# selecting columns
data_cl_cd = data[:,[0,1]]

# fitting model
fit_cl_cd = SVC(kernel = 'rbf', C=1000,
↪  gamma='scale').fit(data_cl_cd,labels)

# prediction
fit_cl_cd.predict(data_cl_cd)
print('Classifier yields accuracy of
↪  {:2f}%'.format(fit_cl_cd.score(data_cl_cd,labels)))
```

6.5 CULMEN LENGTH VS. FLIPPER LENGTH

```
# selecting columns
data_cl_fd = data[:,[0,2]]

# fitting model
fit_cl_fd = SVC(kernel = 'rbf', C=1000,
↪  gamma='scale').fit(data_cl_fd,labels)

# prediction
fit_cl_fd.predict(data_cl_fd)
print('Classifier yields accuracy of
↪  {:2f}%'.format(fit_cl_fd.score(data_cl_fd,labels)))
```

6.6 CULMEN LENGTH VS. BODY MASS

```
# selecting columns
data_cl_bm = data[:,[0,3]]

# fitting model
fit_cl_bm = SVC(kernel = 'rbf', C=1000,
↪  gamma='scale').fit(data_cl_bm,labels)

# prediction
fit_cl_bm.predict(data_cl_bm)
print('Classifier yields accuracy of
↪  {:2f}%'.format(fit_cl_bm.score(data_cl_bm,labels)))
```

6.7 ALL 4 PREDICTORS

```
# fitting model
fit_all = SVC(kernel = 'rbf', C=1000,
↪  gamma='scale').fit(data,labels)

# prediction
fit_all.predict(data)
print('Classifier yields accuracy of
↪  {:2f}%'.format(fit_all.score(data,labels)))
```

6.8 RESULTS

We see the support vector machine using the culmen length and culmen depth as the only predictors had the highest accuracy of 97.6%. The next highest performing model contains the culmen length and flipper depth, followed by the model with all 4 predictors, and lastly the culmen length and body mas with respective accuracies of 95.5%, 87.1%, and 82.9%.

7 APPLICATIONS

One such example, as presented in *Intro to Statistical Learning with Applications in R* [5] is the use of support vector machines in genetic expression data. The *Khan* dataset in the $ISLR2$ package of R contains expression measurements for 2,308 genes from tissue samples of patients with one of four types of small round blue cell tumors. The data is then split up into training and test groups. There are 63 observations in the training set and 20 observations in the test set. It is not feasible to visually plot the data on a graph, as there is a very large number of features relative to the number of observations. However, due to the large number of features, it is easy to find a linearly separating hyperplane that predicts the type of cell tumor. As such, the SVM approach outlined in the textbook yields no data points that are misclassified in the training set and two test set errors. In this case, a support vector machine was used to classify and predict cancer types based on gene expression data. In fact, SVMs are a popular choice in machine learning approaches to detecting cancer. Several studies have shown SVMs perform with great accuracy. Among those include the prediction of breast cancer using SVM and an extremely randomized trees classifier [1]. Support vector machines have also been used in multi-class lung cancer classification [6].

8 ANALYSIS OF R AND PYTHON SVM MODULES AND DOCUMENTATION

- supporting documentation for each
- ease of use
- notes on function parameters

References

[1] Ganjar Alfian, Muhammad Syafrudin, Imam Fahrurrozi, Norma Latif Fitriyani, Fransiskus Tatas Dwi Atmaji, Tri Widodo, Nurul Bahiyah, Filip Benes, and Jongtae Rhee. Predicting breast cancer from risk factors using svm and extra-trees-based feature selection method. *Computers*, 11(136), 2022.

[2] Kirsten Barkved. The difference between training data vs. test data in machine learning short history of machine learning – every manager should read. *obviously.ai*, 2022.

[3] Geeks for Geeks. Supervised and unsupervised learning, 2022.

[4] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2nd edition, 2008. ISBN 978-0387848570.

[5] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R*. Springer, 2nd edition, 2021. ISBN 978-1461471370.

[6] Yong Mao, Xiaobo Zhou, Daoying Pi, Youxian Sun, and Stephen T. C. Wong. Multiclass cancer classification by using fuzzy support vector machine and binary decision tree with gene selection. *Journal of Biomedicine and Biotechnology*, 2005.

[7] Bernard Marr. A short history of machine learning – every manager should read. *Forbes*, 2016.

[8] scikit-learn Developers. Rbf svm parameters. *scikit learn*, 2023.

[9] Jeremy Teitelbaum. Support vector machines, 2021.

[10] Jeremy Teitelbaum. Support vector machines lab, 2021.

[11] Soner Yildrim. Hyperparameter tuning for support vector machines — c and gamma parameters. *Towards Data Science*, 2020.