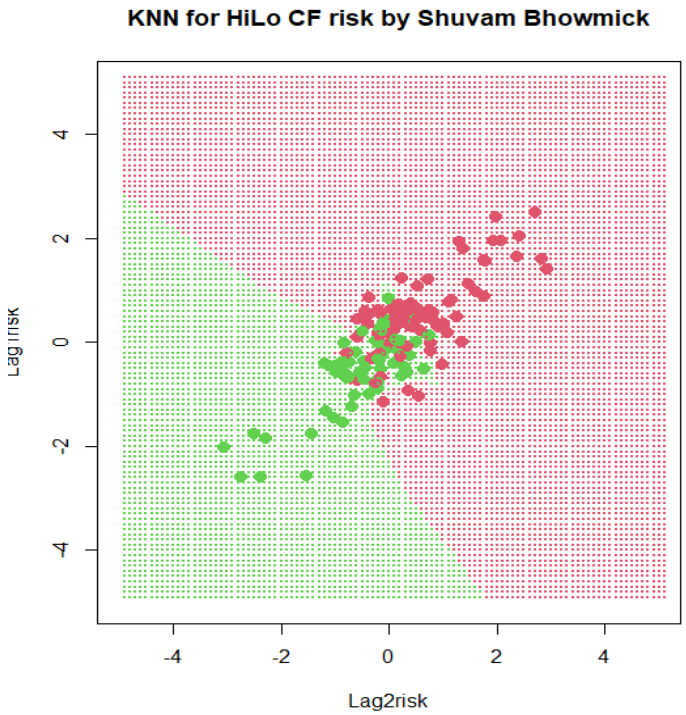
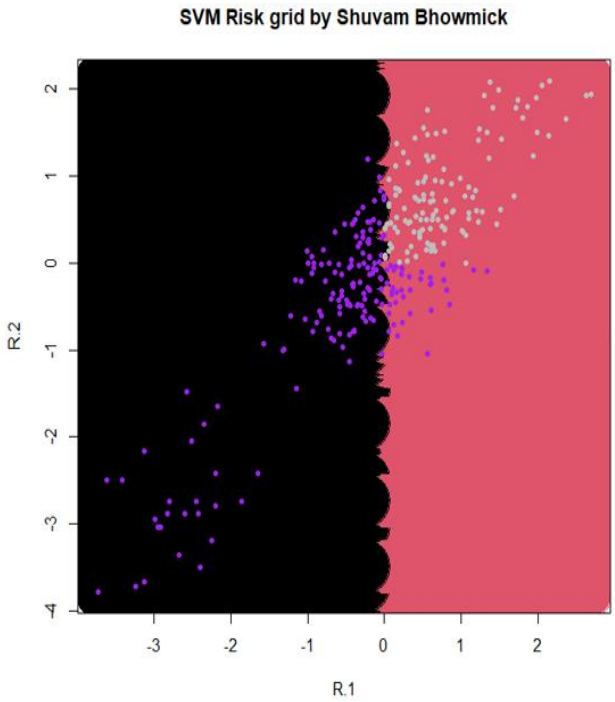
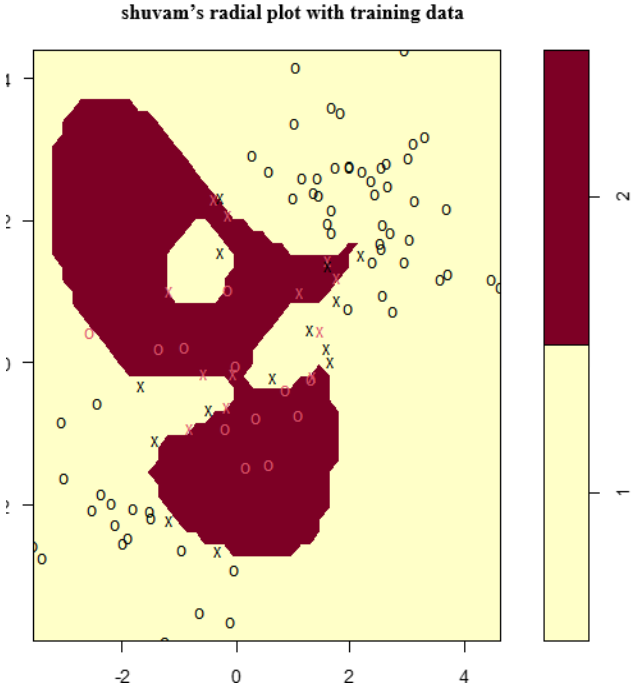
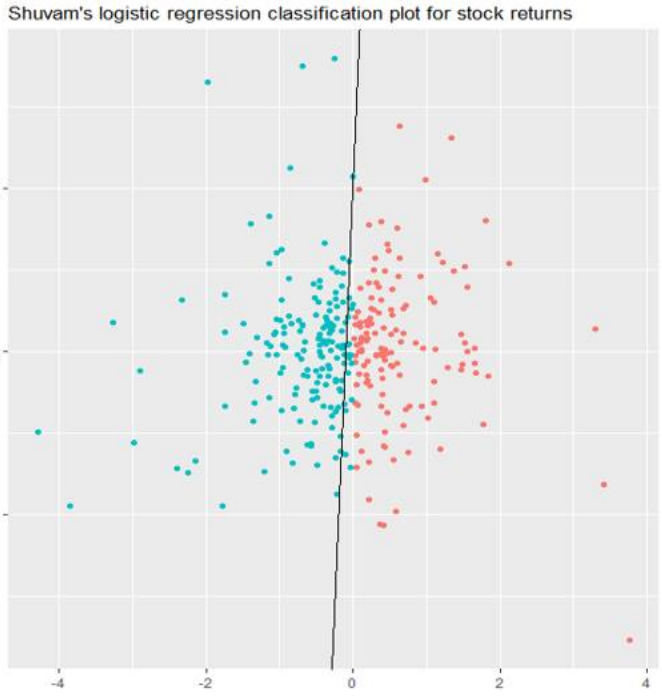
Unleashing the Power of Statistics on Stock Data: A Comprehensive Approach to Classification using Naive Bayes, KNN, Logistic Regression, and SVM









Shuvam Bhowmick

Table of Contents

1. In this paper, I will be working with stock data of the manufacturing company CFindustries who are well known for its innovative fertilization products. Interestingly, this company spent most of its life being a non-stock company, working closely with regional agricultural supply cooperatives. The original name of CFindustries was the "Central Farmers Fertilizer Company[1]" founded in 1946. CFindustries become a stock company in 2002 after 56 years of trade. I will now explain why we took two different transformations of the price return and risk data.

First transformation : For the return stock data, I worked with the percentage change of the adjusted daily closing price. By doing this, we can see how the return percentage changes over time between two consecutive adjusted closing prices. This gives us a clearer understanding of what the average return might be since we can see the average percent change of the stock. From the picture below, you can see how Column "I" contains a formula that subtracts the adjusted close of the previous day from the adjusted close of the current observation. It then multiplies the subtracted value by 100 and divides that number by the previous adjusted close price.

Second transformation: For the risk stock data, I used the daily price range (High – Low) transformation. This gives us an idea of how the stock moves daily. In other words, we compute the daily volatility of the stock to understand the risk. It is useful to take the logarithm of the range because logs respond well to the problem of skewness that can come about from large daily swings. The logs can reduce some of the larger variation that comes with large daily swings. By using the log of the percent change of the stock, we account for outliers or unusual price movements. From the picture below, you can see how column "J" contains the formula High (Column C) – Low (Column D) to get the daily range. We then took the logarithm (excel using 'ln') of the daily range values in column J.

	A	B	C	D	E	F	G	H	I	J	K
1	Date	Open	High	Low	Close	Adj Close	Volume		Cfret "=100*(F3-F2)/F2"	Range "=C2 - D2"	Cfrisk "=Ln(J2)"
2	1/3/2006	3.05	3.08	3.02	3.06	2.270684	2272500			0.06	-2.813410717
3	1/4/2006	3.06	3.14	3.06	3.112	2.30927	2567500		1.699311749	0.08	-2.525728644
4	1/5/2006	3.12	3.248	3.114	3.23	2.396832	5217500		3.791761033	0.134	-2.009915479
5	1/6/2006	3.22	3.22	3.16	3.204	2.37754	5087000		-0.804895796	0.06	-2.813410717
6	1/9/2006	3.204	3.32	3.204	3.28	2.433935	3454500		2.371989535	0.116	-2.154165088
7	1/10/2006	3.246	3.296	3.224	3.29	2.441357	2359500		0.3049383	0.072	-2.63108916
8	1/11/2006	3.29	3.29	3.24	3.258	2.417611	2560500		-0.972655781	0.05	-2.995732274
9	1/12/2006	3.25	3.316	3.23	3.26	2.419095	2819000		0.061382911	0.086	-2.453407983
10	1/13/2006	3.262	3.284	3.254	3.274	2.429484	1849000		0.429458124	0.03	-3.506557897
11	1/17/2006	3.244	3.268	3.172	3.256	2.416127	2442500		-0.549787527	0.096	-2.343407088
12	1/18/2006	3.24	3.318	3.21	3.316	2.460649	1538500		1.842701149	0.108	-2.225624052
13	1/19/2006	3.32	3.322	3.268	3.306	2.45323	1172500		-0.301505822	0.054	-2.918771232
14	1/20/2006	3.312	3.318	3.178	3.19	2.367152	1943500		-3.508761918	0.14	-1.966112856
15	1/23/2006	3.194	3.276	3.192	3.246	2.408706	1171000		1.755442827	0.084	-2.47693848
16	1/24/2006	3.25	3.324	3.24	3.298	2.447294	3657000		1.602021999	0.084	-2.47693848
17	1/25/2006	3.3	3.308	3.22	3.26	2.419095	11327500		-1.152252243	0.088	-2.430418465
18	1/26/2006	3.266	3.366	3.266	3.352	2.487363	2149500		2.822047088	0.1	-2.302585093
19	1/27/2006	3.35	3.42	3.342	3.404	2.52595	1289500		1.551321621	0.078	-2.551046452
20	1/30/2006	3.4	3.45	3.354	3.38	2.508142	1613500		-0.705002078	0.096	-2.343407088
21	1/31/2006	3.38	3.424	3.37	3.402	2.524467	2094500		0.650880213	0.054	-2.918771232
22	2/1/2006	3.384	3.384	3.35	3.356	2.490332	2454500		-1.352166616	0.034	-3.381394754
23	2/2/2006	3.344	3.356	3.266	3.3	2.448777	7376500		-1.668653015	0.09	-2.407945609
24	2/3/2006	3.296	3.38	3.276	3.376	2.505172	2558000		2.302986348	0.104	-2.26336438
25	2/6/2006	3.376	3.418	3.37	3.416	2.534856	2713000		1.184908661	0.048	-3.036554268
26	2/7/2006	3.418	3.418	3.376	3.388	2.514077	2793000		-0.819730983	0.042	-3.170085661
27	2/8/2006	3.386	3.462	3.37	3.448	2.5586	2350000		1.770948145	0.092	-2.385966702
28	2/9/2006	3.446	3.558	3.44	3.5	2.597187	1597500		1.508129446	0.118	-2.137070655

BEFORE AFTER

[1] "History." CF Industries. Accessed December 15, 2022. https://www.cfindustries.com/who-we-are/history

I created lagged stock data to show the return and risk data in multiple columns. Before I show that process, I will mention that I started by standardizing the return and risk data. For this, we subtract the mean of the entire return/risk column from each observation and divided it by the standard deviation.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Date	Cfret	Cfrisk		CFret(standardized)	Cfrisk(standardized)						Cfret	Cfrisk	
2	1/4/2006	1.699311749	-2.525728644		0.53323379	-2.855734003					SamAvg	0.133522	-0.08604	
3	1/5/2006	3.791761033	-2.009915479		1.245822671	-2.251959076					SamStd	2.936405	0.854314	
4	1/6/2006	-0.804895796	-2.813410717		-0.319580406	-3.192474578								
5	1/9/2006	2.371989535	-2.154165088		0.762315907	-2.420807611								
6	1/10/2006	0.3049383	-2.63108916		0.058376394	-2.979061676								
7	1/11/2006	-0.972655781	-2.995732274		-0.376711493	-3.40588748								
8	1/12/2006	0.061382911	-2.453407983		-0.024567008	-2.771080479								

Formula in column E : (B2 - L2)/ L3

Formula in column F : (C2 - M2) / M3

After the standardization, I created the lagged stock data by making three copies of each column (Cfrisk and Cfret). We then created time-lagged versions of these columns by deleting the first entry of the second column and the first two entries of the third column. The first column remained unchanged. I renamed the first two columns as lag2 and lag1.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Date	Lag2ret	Lag1ret	Cfret			Lag2risk	Lag1risk	Cfrisk						
2	1/4/2006	0.53323379	1.245822671	-0.319580406			-2.855734003	-2.251959076	-3.192474578						
3	1/5/2006	1.245822671	-0.319580406	0.762315907			-2.251959076	-3.192474578	-2.420807611						
4	1/6/2006	-0.319580406	0.762315907	0.058376394			-3.192474578	-2.420807611	-2.979061676						
5	1/9/2006	0.762315907	0.058376394	-0.376711493			-2.420807611	-2.979061676	-3.40588748						
6	1/10/2006	0.058376394	-0.376711493	-0.024567008			-2.979061676	-3.40588748	-2.771080479						
7	1/11/2006	-0.376711493	-0.024567008	0.100781934			-3.40588748	-2.771080479	-4.003824338						
8	1/12/2006	-0.024567008	0.100781934	-0.232702642			-2.771080479	-4.003824338	-2.642321101						
9	1/13/2006	0.100781934	-0.232702642	0.582065412			-4.003824338	-2.642321101	-2.504452491						
10	1/17/2006	-0.232702642	0.582065412	-0.148149681			-2.642321101	-2.504452491	-3.315802251						
11	1/18/2006	0.582065412	-0.148149681	-1.240388835			-2.504452491	-3.315802251	-2.200686781						
12	1/19/2006	-0.148149681	-1.240388835	0.55234937			-3.315802251	-2.200686781	-2.798623638						

We copied over the CFret columns to get three copies of it. Then we deleted the first entry and shifted cells upward. We deleted two entries and shifted cells upward for the last CFret column.

Next, we delete the third column labeled Cfret and Cfrisk because those columns hold the most current return/risk values. The purpose of the lagged stock data is to use day-before-yesterday and yesterday return/risk values to determine the forecast of "today's"(deleted column which will be used for testing) return. To make things a bit simpler, we will use the if function that will convert the last column into a categorical factor of being either high or low. With this, we can use the previous two days of values to determine whether "todays" return will be either high or low.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Date	Lag2ret	Lag1ret	Cfret					Date	Lag2risk	Lag1risk	Cfrisk
2	1/4/2006	0.53323379	1.245822671	LoRet					1/4/2006	-2.855734003	-2.251959076	LoRisk
3	1/5/2006	1.245822671	-0.319580406	HiRet					1/5/2006	-2.251959076	-3.192474578	LoRisk
4	1/6/2006	-0.319580406	0.762315907	HiRet					1/6/2006	-3.192474578	-2.420807611	LoRisk
5	1/9/2006	0.762315907	0.058376394	LoRet					1/9/2006	-2.420807611	-2.979061676	LoRisk
6	1/10/2006	0.058376394	-0.376711493	LoRet					1/10/2006	-2.979061676	-3.40588748	LoRisk
7	1/11/2006	-0.376711493	-0.024567008	HiRet					1/11/2006	-3.40588748	-2.771080479	LoRisk
8	1/12/2006	-0.024567008	0.100781934	LoRet					1/12/2006	-2.771080479	-4.003824338	LoRisk
9	1/13/2006	0.100781934	-0.232702642	HiRet					1/13/2006	-4.003824338	-2.642321101	LoRisk

Cfret Column Function : `=IF(D2<0,"LoRet", "HiRet")` **Same function for Cfrisk just different column

2. For part 2, I will draw a random sample of size n=300 without replacement from CFindustries stock returns data set. This sample will come from 300 randomly selected observations from a total of 4246 trading days. In other words, 300 rows of the return dataset will be pulled randomly from 4246 rows. I will do the same procedure for the CFindustries risk data set.

```
CFstock = read.csv("CFriskF.csv")
CFstockRisk = read.csv("CFriskF.csv")
CFstockRet = read.csv("CFretF.csv")
CFstockRet300 = CFstockRet[sample(4246, 300),]
CFstockRisk300 = CFstockRisk[sample(4246,300),]
```

Now that we have our subsets of risk and return data, I can go through the steps of SVM while explaining the methods being used for classification (reference appendix for ISLM textbook steps)

We will be working with two classes : High and Low returns. We will be classifying the lagged stock data. As I mentioned before, the purpose of the lagged stock data is to use day-before-yesterday and yesterday return values for predicting the forecast of "today's" return. With the SVM method, we will train the model using the lagged training data and test the accuracy of predictions using the test data.

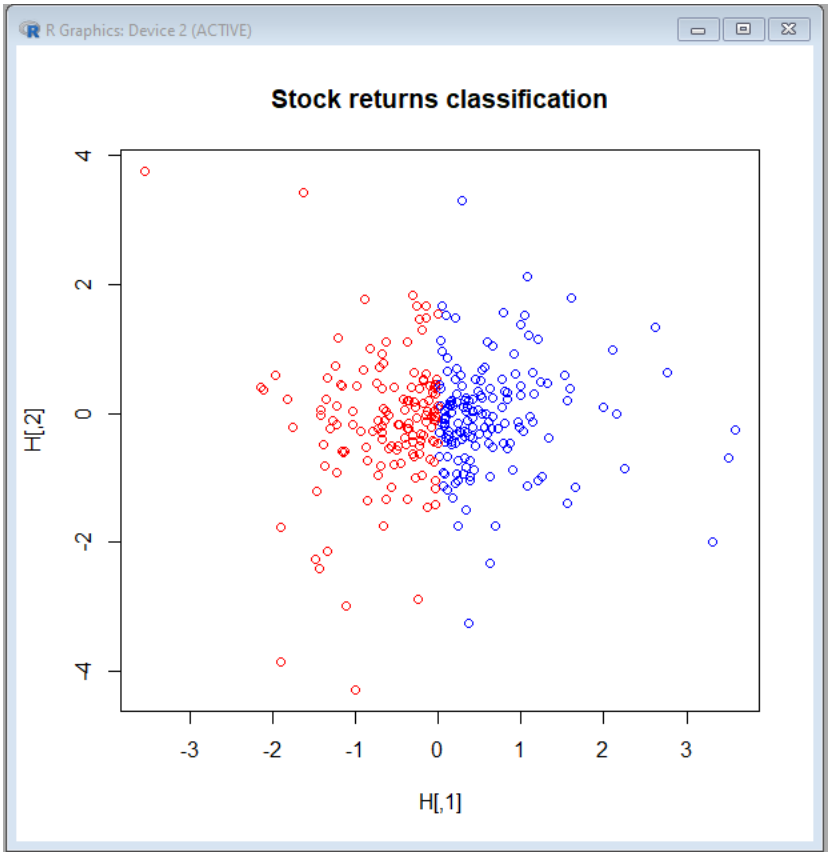
Create a variable than contains 300 sample observations from the stock return dataset of 4246 observations. I will plot the classification and see if it is linearly separable.

```
> head(CFstockRets300[,-1])
         Lag2ret     Laglret Cfret
291  -1.33600951  0.55070105 HiRet
4001  0.07501293 -0.09766977 LoRet
919  -0.26368474 -0.07034668 LoRet
3241  1.14945322  0.30523525 LoRet
2777 -0.56848421 -1.13788632 LoRet
1471  0.96224554 -0.14314606 LoRet
```

```
> head(CFstockRisk300N[,-1])
       Lag2risk   Laglrisk Cfrisk
256  -3.1169303 -2.1677119 LoRisk
3677  0.5908217 -0.2517479 LoRisk
1445  0.7730172  1.0766706 HiRisk
3960  0.7490451  0.4167801 HiRisk
1189 -0.2549143 -0.5531735 LoRisk
2706 -0.5779894 -0.5779894 HiRisk
```

CFstockRets300 and CFstockRisk300N is the sample of 300 observations from the excel data put into a data frame. I then used only the lagged stock data columns and created another matrix called H. I will use this second matrix to show the classification plot on the next page.

```
H = (matrix(as.numeric((CFstockRets300[,2:3])), ncol = 2 ))
plot(H,col=ifelse(H[,1]<0, "red" , "blue"))
```

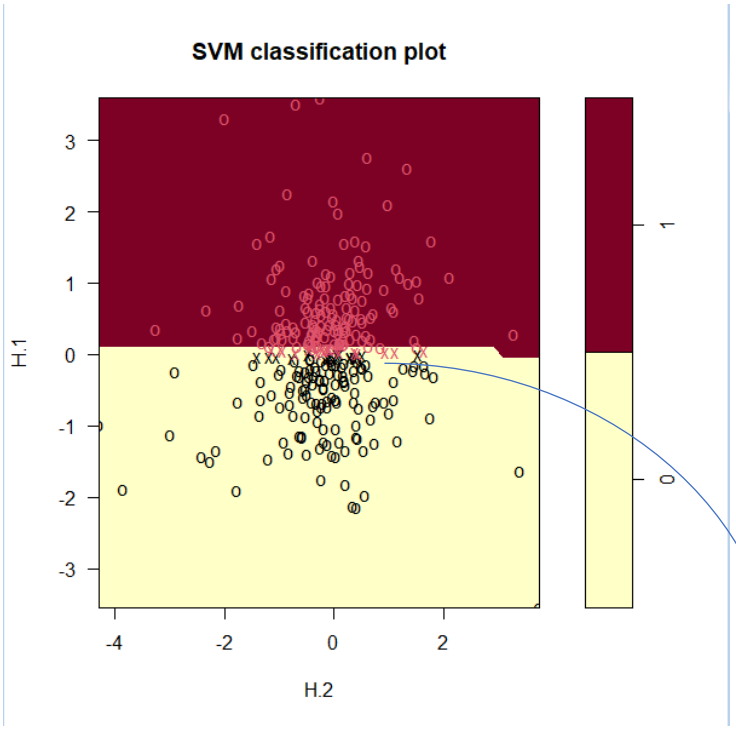


The blue points represent high-return data points or lagged stock returns that have returns greater than zero. The red points represent low-return data points. It looks like this plot is linearly separable. We must test whether a linear boundary or a non-linear boundary will produce better separations of data. First, let's add some linear support vector classifiers and test the classification.

```
F = as.numeric(matrix(H[,1]>0))
dat = data.frame(H=H, F=as.factor(F))
svmfit = svm(F~., data=dat, cost=10,kernel = "linear", scale= FALSE)
plot(svmfit, dat)
```

LINEAR SEPERATION

Shuvam Bhowmick's Stock SVM plot C = 10




```
> summary(svmfit)

Call:
svm(formula = F ~ ., data = dat, cost = 10, kernel = "linear", scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10

Number of Support Vectors:  36

 ( 18 18 )


Number of Classes:  2

Levels:
 0 1
```

The Linear kernel is using 36 support vectors:

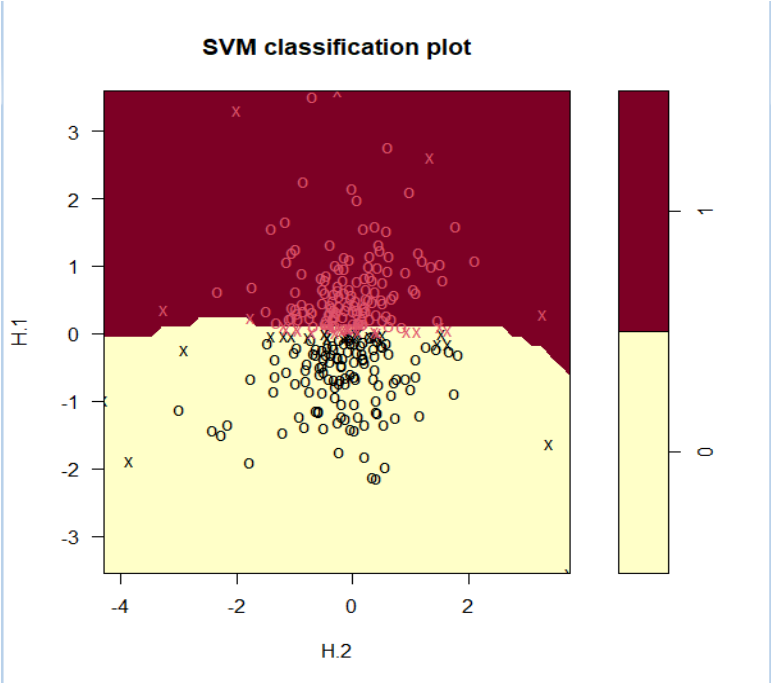
```
> svmfit$index
 [1]  11  26  53  54  74  76 103 166 167 181 217 228 273   6  27  42  65 139
[19] 221 229 230 241 267 274 276
```

The points marked as "X" are support vectors and the black/red circles are stock observations. We have a large cost parameter of 10 with many support vectors. Is this optimal? We will answer that question in part 3. For now, we will look at the classification plots for another kernel

```
F = as.numeric(matrix(H[,1]>0))
svmfit = svm(F~., data=dat, cost=10,kernel = "radial", scale= FALSE)
dat = data.frame(H=H, F=as.factor(F))
plot(svmfit, dat)
```

RADIAL SEPERATION

Shuvam Bhowmick's Stock SVM plot C = 10



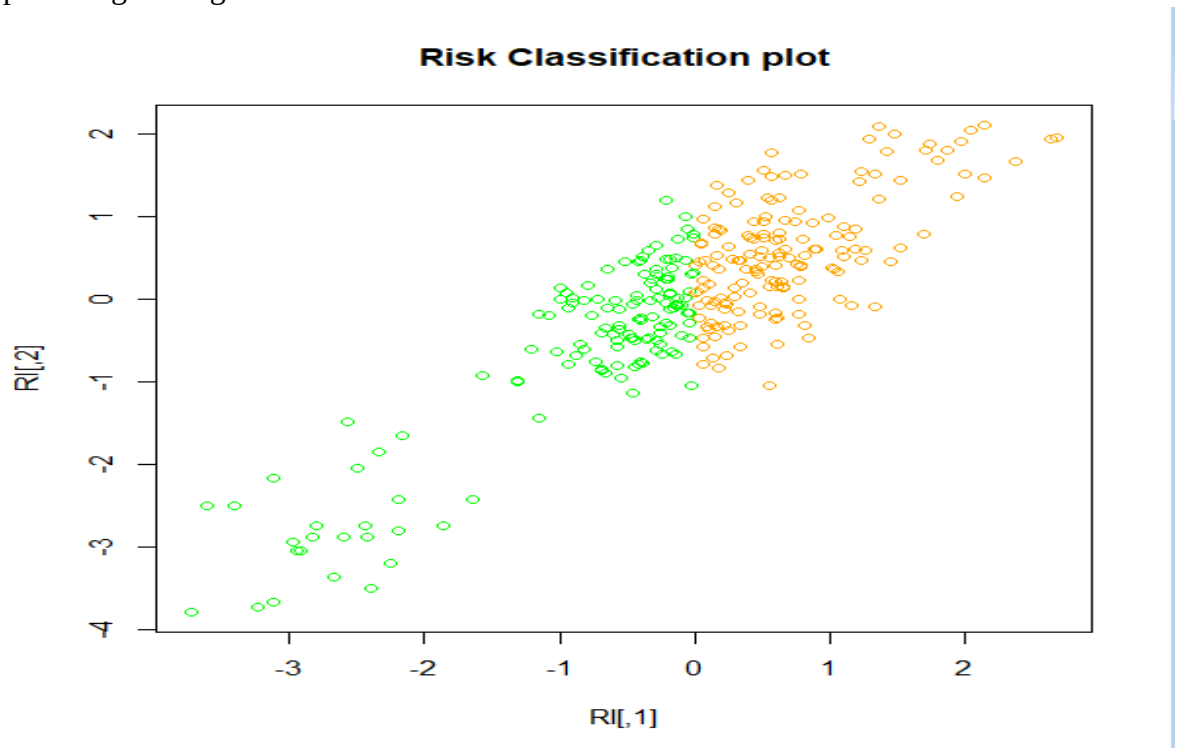

```
> summary(svmfit)

Call:
svm(formula = F ~ ., data = dat, cost = 10, kernel = "radial", scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  52

 ( 26 26 )


Number of Classes:  2

Levels:
 0 1
```

The svm() function is using the radial kernel. The number of support vector machines has increased from before even though the cost parameter is still C=10. The radial kernel has increased the width of the margin.

Now, that we have created SVM plots for the stock return data, we will know do the same for the stock risk data. I will begin by setting up my matrix the same way I did for the return data (using cbind()).

```
data.matrix(CFstockRisk300N)

RI = (matrix(as.numeric((MatrixRISK[,2:3])), ncol=2))

plot(RI,col = ifelse(RI[,1]<0, "green", "orange"))
```

I will now plot the data where the red points represent high risk and the green points represent low risk. Yesterday's percentage change of risk is on the y-axis, while the day-before yesterday's percentage change is on the x-axis.

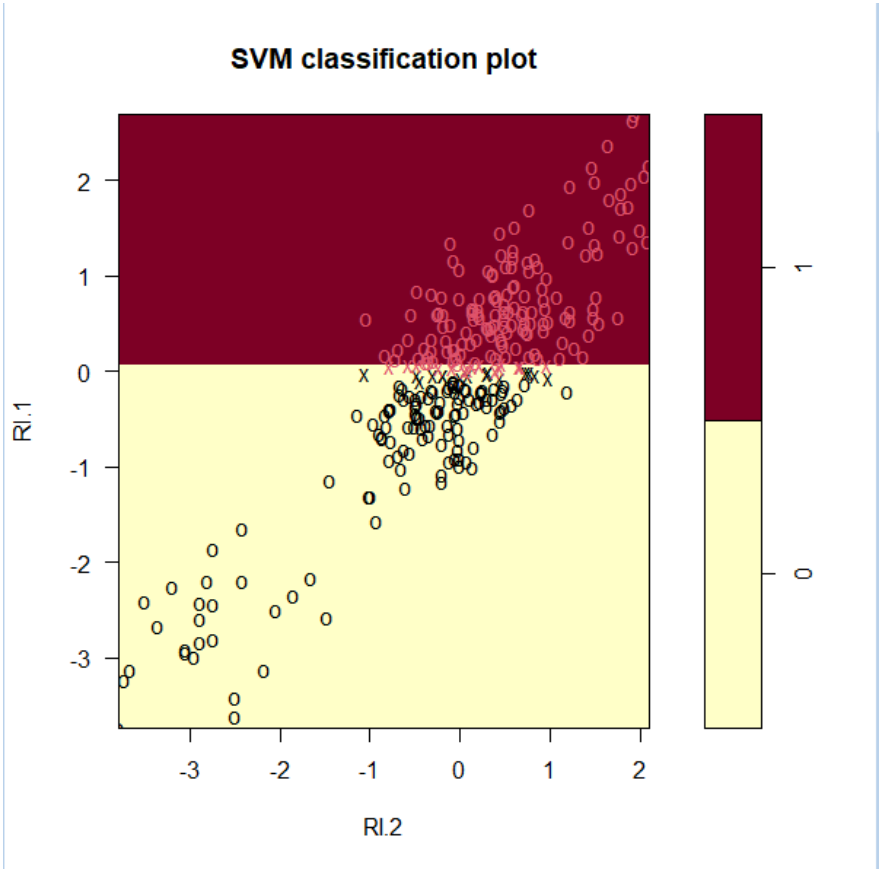


This plot looks very similar to the return data plot. However, there might be a sharper linear separation here.

```
OP = as.numeric(matrix(RI[,1]>0))
datRisk = data.frame(RI =RI , OP = as.factor(OP))
svmfit = svm(OP~., data=datRisk,kernel="linear", cost=10,scale=FALSE)
plot(svmfit,datRisk)
```

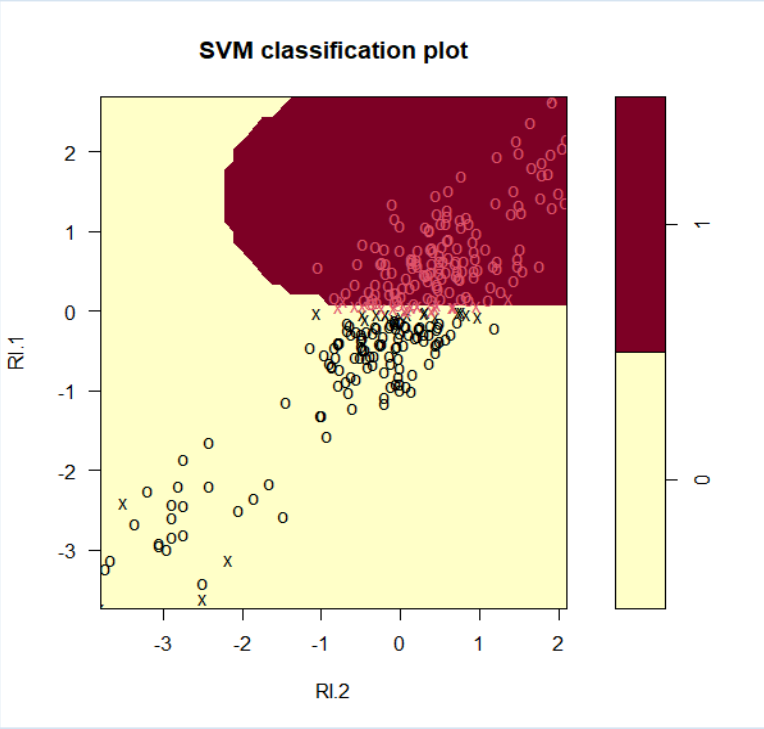
The OP variable is creating the classification for the svm function. We are using a linear kernel with a cost of 10 which should give us a smaller width margin.

Shuvam Bhowmick's Stock Risk SVM plot C = 10 LINEAR SEPERATION




```
> summary(svmfit)

Call:
svm(formula = OP ~ ., data = datRisk, kernel = "linear", cost = 10, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10

Number of Support Vectors:  32

 ( 16 16 )


Number of Classes:  2

Levels:
 0 1
```

There are 32 support vector classifiers with a decent sized margin. It looks like there is a good spread of support vectors on both sides of the classification boundary(bottom yellow represents low risk, and red represents high risk). There are a few high-risk points leaning towards the low-risk side. Let's look at a radial kernel and see if that helps reduce the classification errors.




```
> summary(svmfit)

Call:
svm(formula = OP ~ ., data = datRisk, kernel = "radial", cost = 10, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  38

 ( 20 18 )


Number of Classes:  2

Levels:
 0 1
```

There was an increase in support vectors. Changing from a linear kernel to a non-linear kernel did not change the classification. It looks like the radial kernel is a bit more conservative in terms of the space used for the red classification (high risk). This may suggest that the probability of risk being classified as high is smaller than it is for low.

In general, the definition of high risk can depend on how risk adverse a person is. However, everyone can agree if a return is high or not by comparing previous returns. Using the svm method, it also looks like the linear kernel performs just as well as the radial kernel. Using a linear or non-linear decision boundary does not make a difference in terms of how correct the classifications are. However, when we used the radial (non-linear) decision boundary, the program used more support vectors which maximizes the margin.

3.

The "cost parameter[2]" in SVM maximizes the trade-off between achieving a low error rate on the training data and allowing the model to be more flexible to generalize new data. A high-cost value will result in a model with low error on the training data but wouldn't generalize well to new data. A low-cost value will result in a more flexible model that may have a higher error on the training data but better generalization of new data. The cost parameter is determined through cross-validation, where the model is trained on a subset of data and tested on the remaining data.

Let's test the cost parameter by using the built-in tune() function (available in the library e1071) to perform cross-validation. By default, the tune() function performs ten-fold cross-validation using a range of cost parameters. This will set us up for testing which cost parameter works best.

```
+ > tune.out = tune(svm, F~., data=dat, kernel="linear", ranges = list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  100

- best performance: 0.01

- Detailed performance results:
   cost      error dispersion
1 1e-03 0.43000000 0.15432049
2 1e-02 0.15333333 0.09189366
3 1e-01 0.05000000 0.04779070
4 1e+00 0.01333333 0.02330686
5 5e+00 0.01333333 0.02330686
6 1e+01 0.01333333 0.02330686
7 1e+02 0.01000000 0.02249829
```

Based on the tune.out() function, the cost parameter that performs the best is Cost=100. This allows for fewer errors on training data but may not work well with classifying new data.

[2] Tumminello, Aurora. "Statistical Models Part II." Chapter 11 Support Vector Machines. https://bookdown.org/aurora_tumminello/statistics_lab/support-vector-machines.html

The cost parameter of 100 gives us the lowest cross-validation error rate so we will store this model for later use.

```
> bestmod = tune.out$best.model
> summary(bestmod)

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


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  100

Number of Support Vectors:  15

 ( 8 7 )


Number of Classes:  2

Levels:
 0 1
```

```
> RETPREDICT = predict(bestmod, dat)
> table(predict = RETPREDICT, truth = dat$F)
       truth
predict   0   1
      0 142   0
      1   0 158
```

For the classification of the return data, there are no errors. Let's see if the same holds for the risk data set.

```
> tune.outRisk = tune(svm, OP~., data=datRisk, kernel="linear", ranges = list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))
> summary(tune.outRisk)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  100

- best performance: 0.006666667

- Detailed performance results:
  cost       error dispersion
1 1e-03 0.413333333 0.11243654
2 1e-02 0.153333333 0.04216370
3 1e-01 0.053333333 0.07062333
4 1e+00 0.013333333 0.02330686
5 5e+00 0.010000000 0.01610153
6 1e+01 0.013333333 0.01721326
7 1e+02 0.006666667 0.01405457
```

The tune() function says that the best cost parameter function to use is 100 based on the given range (0.001, 0.01,1,5,10,100)). Let's store that model with the best cost function and determine its performance on the training data set.

```
> bestmodRisk = tune.outRisk$best.model
> Riskpred = predict(bestmodRisk, datRisk)
> table(predict = Riskpred, truth = datRisk$OP)
       truth
predict   0   1
      0 140   2
      1   0 158
```

There are 2 classification errors for the risk training data, while the return data had 0. The model predicted low risk(0) when 2 observations were actually high risk. This could mean that it is harder to classify the risk levels of the stock using the SVM() method then it is to classify return levels. Since 298 out of 300 risk points were classified correctly, we can neglect the classification errors. However, if we worked with a bigger sample size, the classification errors will be larger in most cases.

After observing the classification performance, let us now look at how the model performs at predicting new data. I will take a brand new sample of 300 observations of risk and return data points from the 4246 observations. I will then split it in half. 150 observations will be used to train the model for prediction of the second half.

First, let's create the training and testing data set.

```
> intrain = createDataPartition(y=CFstockRet300Sample$Cfret, p=0.5, list = FALSE)
> trainingR = CFstockRet300Sample[intrain,]
> testingR = CFstockRet300Sample[-intrain,]
> anyNA(CFstockRet300Sample)
[1] FALSE
> dim(trainingR)
[1] 150   4
> dim(testingR)
[1] 150   4
```

```
trainingR[["Cfret"]] = factor(trainingR[["Cfret"]])
```

Cfret
LoRet
HiRet
HiRet
LoRet
LoRet
HiRet

There are 150 observations for each of the subsets. Cfret refers to what we will be predicting and it must be factorized.

```
trctrl = trainControl(method = "repeatedcv", number = 10, repeats = 3)
svm_Linear = train(Cfret~. , data = trainingR, method = "svmLinear", trControl = trctrl, preProcess = c("center", "scale"), tuneLength = 10)
```

Now we control the training process by setting the cross-validation attributes. We will iterate the cross-validation method 10 times and repeat the entire process 3 times. A 10-fold Cross-validation[3] repeated 3 times refers to dividing the training data into 10 parts and performing cross-validation 3 times on each of those partitions. This can be useful for model selection and evaluation because it allows the model performance to be estimated multiple times, which can help to reduce the variability of the estimate and provide a more reliable assessment of model performance. Let's see how the model performed when using the testing set.

```
> test_pred_Ret = predict(svm_Linear, newdata = testingR)
```

```
> confusionMatrix(table(test_pred_Ret, testingR$Cfret))
Confusion Matrix and Statistics

             
test_pred_Ret HiRet LoRet
        HiRet    32    33
        LoRet    42    43

               Accuracy : 0.5
                 95% CI : (0.4174, 0.5826)
    No Information Rate : 0.5067
    P-Value [Acc > NIR] : 0.5968

                  Kappa : -0.0018

 Mcnemar's Test P-Value : 0.3556

            Sensitivity : 0.4324
            Specificity : 0.5658
         Pos Pred Value : 0.4923
         Neg Pred Value : 0.5059
             Prevalence : 0.4933
         Detection Rate : 0.2133
   Detection Prevalence : 0.4333
      Balanced Accuracy : 0.4991

       'Positive' Class : HiRet
```

The model was able to predict the outcomes of the testing data with an accuracy of 50%. This is quite low so let's adjust the tuning parameter.

[3] Kuhn, Max. "The Caret Package." 5 Model Training and Tuning, March 27, 2019. https://topepo.github.io/caret/model-training-and-tuning.html

```
grid = expand.grid(C = c(0,0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.25,1.5,1.75,2,5))
svm_Linear_Grid = train(Cfret~., data = trainingR, method = "svmLinear",trControl = trctrl, preProcess = c("center", "scale"), tuneGrid = grid, tuneLength = 10)
```

```
> svm_Linear_Grid
Support Vector Machines with Linear Kernel

150 samples
  3 predictor
  2 classes: 'HiRet', 'LoRet'

Pre-processing: centered (3), scaled (3)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 135, 134, 134, 134, 135, 135, ...
Resampling results across tuning parameters:

  C     Accuracy   Kappa
  0.00       NaN         NaN
  0.01  0.5066667   0.00000000
  0.05  0.4826786  -0.03639969
  0.10  0.4669048  -0.07027220
  0.25  0.4704762  -0.06327388
  0.50  0.4727183  -0.05845403
  0.75  0.4822421  -0.03940641
  1.00  0.4822421  -0.03940641
  1.25  0.4822421  -0.03940641
  1.50  0.4822421  -0.03940641
  1.75  0.4822421  -0.03940641
  2.00  0.4822421  -0.03940641
  5.00  0.4820833  -0.04006600

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 0.01.
```

The final C parameter for the model was chosen to be C = 0.01 which gives us an accuracy of 51%. The accuracy did not improve much after tuning the parameter C.

```
> confusionMatrix(table(test_pred_grid, testingR$Cfret))
Confusion Matrix and Statistics


test_pred_grid HiRet LoRet
          HiRet      0      0
          LoRet     74     76

               Accuracy : 0.5067
                 95% CI : (0.4239, 0.5892)
    No Information Rate : 0.5067
    P-Value [Acc > NIR] : 0.5327

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.0000
            Specificity : 1.0000
         Pos Pred Value :    NaN
         Neg Pred Value : 0.5067
             Prevalence : 0.4933
         Detection Rate : 0.0000
   Detection Prevalence : 0.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : HiRet
```

Since the code is the same for the risk data set, I will just show the accuracy and tuning of the risk svm model.

```
> intrainRisk = createDataPartition(y=CFstockRisk300Sam$Cfrisk, p=0.5, list=FALSE)
> trainingRisk = CFstockRisk300Sam[intrainRisk,]
> testingRisk = CFstockRisk300Sam[-intrainRisk,]
> trainingRisk[["Cfrisk"]] = factor(trainingRisk[["Cfrisk"]])
> svm_Linear = train(Cfrisk~., data = trainingRisk, method = "svmLinear", trControl = trctrlRisk, preProcess = c("center", "scale"), tuneLength = 10)
> test_pred_Risk = predict(svm_Linear , newdata = testingRisk)
> confusionMatrix(table(test_pred_Risk, testingRisk$Cfrisk))
Confusion Matrix and Statistics


test_pred_Risk HiRisk LoRisk
       HiRisk     67     18
       LoRisk     17     48

               Accuracy : 0.7667
                 95% CI : (0.6907, 0.8318)
    No Information Rate : 0.56
    P-Value [Acc > NIR] : 1.106e-07

                  Kappa : 0.5257

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.7976
            Specificity : 0.7273
         Pos Pred Value : 0.7882
         Neg Pred Value : 0.7385
             Prevalence : 0.5600
         Detection Rate : 0.4467
   Detection Prevalence : 0.5667
      Balanced Accuracy : 0.7624

       'Positive' Class : HiRisk
```

The model was 76% accurate at predicting risk on the testing data after training. Let's adjust the C parameter to try and increase the accuracy of the model.

```
> svm_Linear_Grid_Risk
Support Vector Machines with Linear Kernel

150 samples
  2 predictor
  2 classes: 'HiRisk', 'LoRisk'

Pre-processing: centered (2), scaled (2)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 134, 135, 136, 135, 135, 134, ...
Resampling results across tuning parameters:

  C        Accuracy   Kappa
  0.00000        NaN        NaN
  0.00001  0.5601389  0.0000000
  0.00100  0.5601389  0.0000000
  0.01000  0.6601984  0.2463016
  0.05000  0.7027976  0.3707378
  0.10000  0.7023413  0.3713248
  0.25000  0.7058929  0.3898197
  0.50000  0.7061508  0.3934660
  0.75000  0.7017063  0.3859578
  1.00000  0.6990278  0.3823899
  1.25000  0.6923413  0.3694667
  1.50000  0.6945635  0.3747150
  1.75000  0.6945635  0.3747150
  2.00000  0.6991667  0.3837603
  5.00000  0.7056944  0.3979505

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 0.5.
```

The final value for the model was C = 0.5 . This is where we achieve the highest accuracy.

Let's see how the prediction table looks using C value of 0.5

```
> test_pred_grid_risk = predict(svm_Linear_Grid_Risk, newdata = testingRisk)
> confusionMatrix(table(test_pred_grid_risk,testingRisk$Cfrisk))
Confusion Matrix and Statistics


test_pred_grid_risk HiRisk LoRisk
             HiRisk     67     18
             LoRisk     17     48

               Accuracy : 0.7667
                 95% CI : (0.6907, 0.8318)
    No Information Rate : 0.56
    P-Value [Acc > NIR] : 1.106e-07

                  Kappa : 0.5257

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.7976
            Specificity : 0.7273
         Pos Pred Value : 0.7882
         Neg Pred Value : 0.7385
             Prevalence : 0.5600
         Detection Rate : 0.4467
   Detection Prevalence : 0.5667
      Balanced Accuracy : 0.7624

       'Positive' Class : HiRisk
```

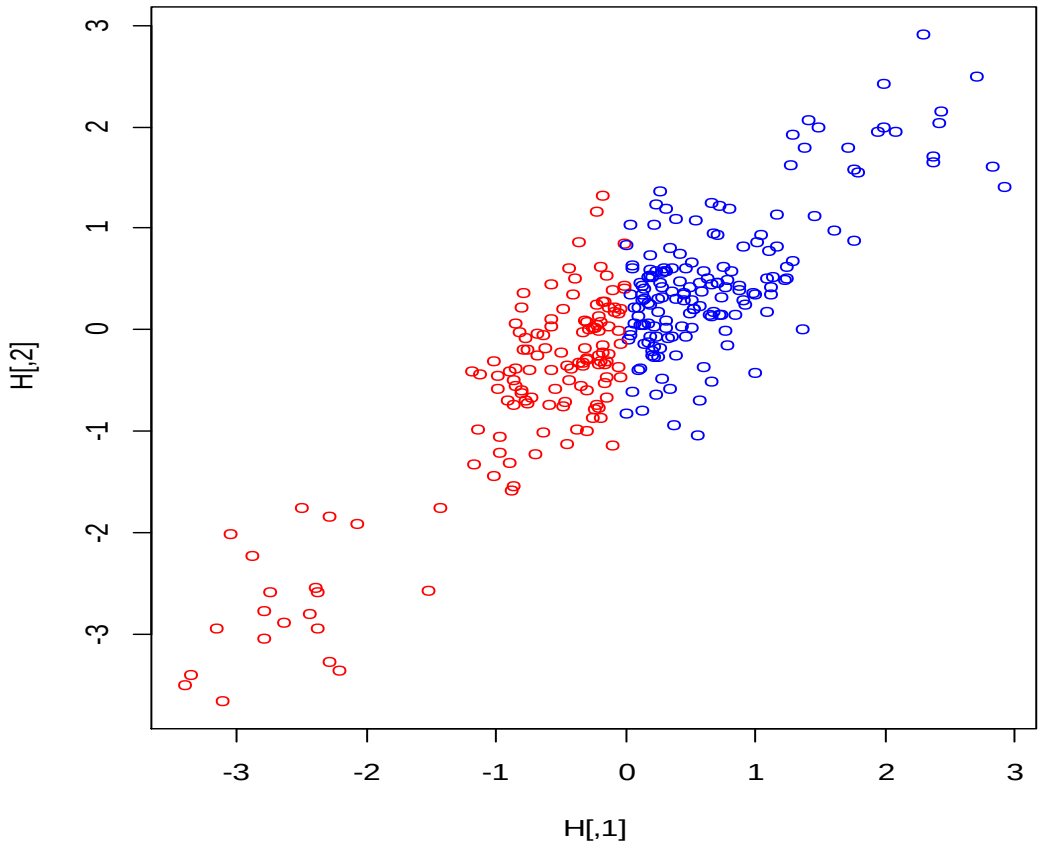
The best accuracy of 76% was achieved when we used C=0.5257. Overall, the model was able to predict the classification of new data with higher accuracy for risk than return. This is quite odd since we used a higher Cost parameter value for the risk model, yet it performed better with the new data than the return model. In general, I believe it is easier for the model to predict the risk level of the stock compared to the previous two days than it is to predict the returns based on previous two days' returns.

4. I will now create step-by-step classification plot for the SVM methodology without using the SVM software generated plots like before.

n=300 return data set

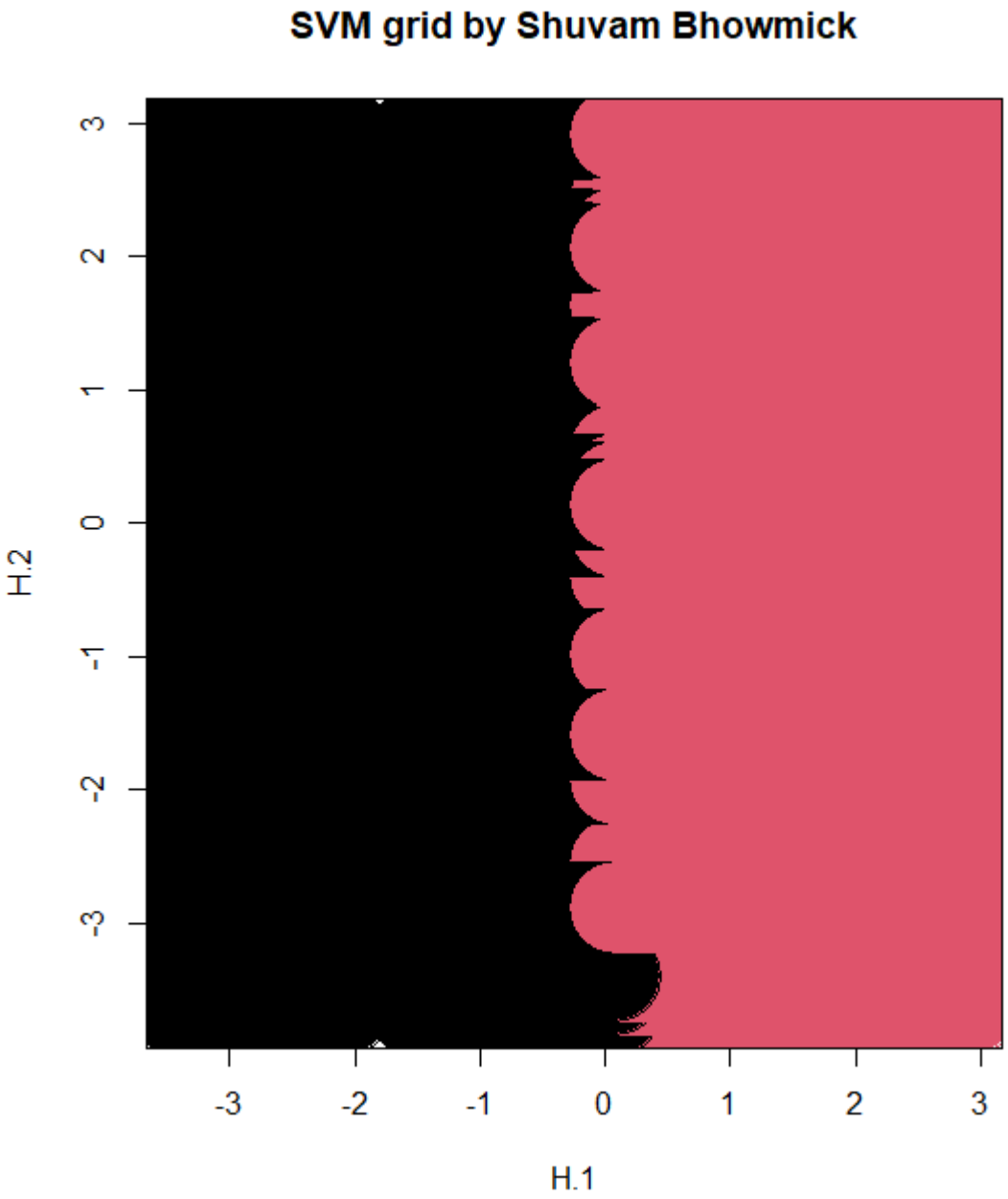
We will start off with just plotting the points,

```
H = (matrix(as.numeric((MatrixRET[,2:3])), ncol=2))
L = ifelse(H[,1]>0 , "blue" , "red")
plot(H, col = L)
```

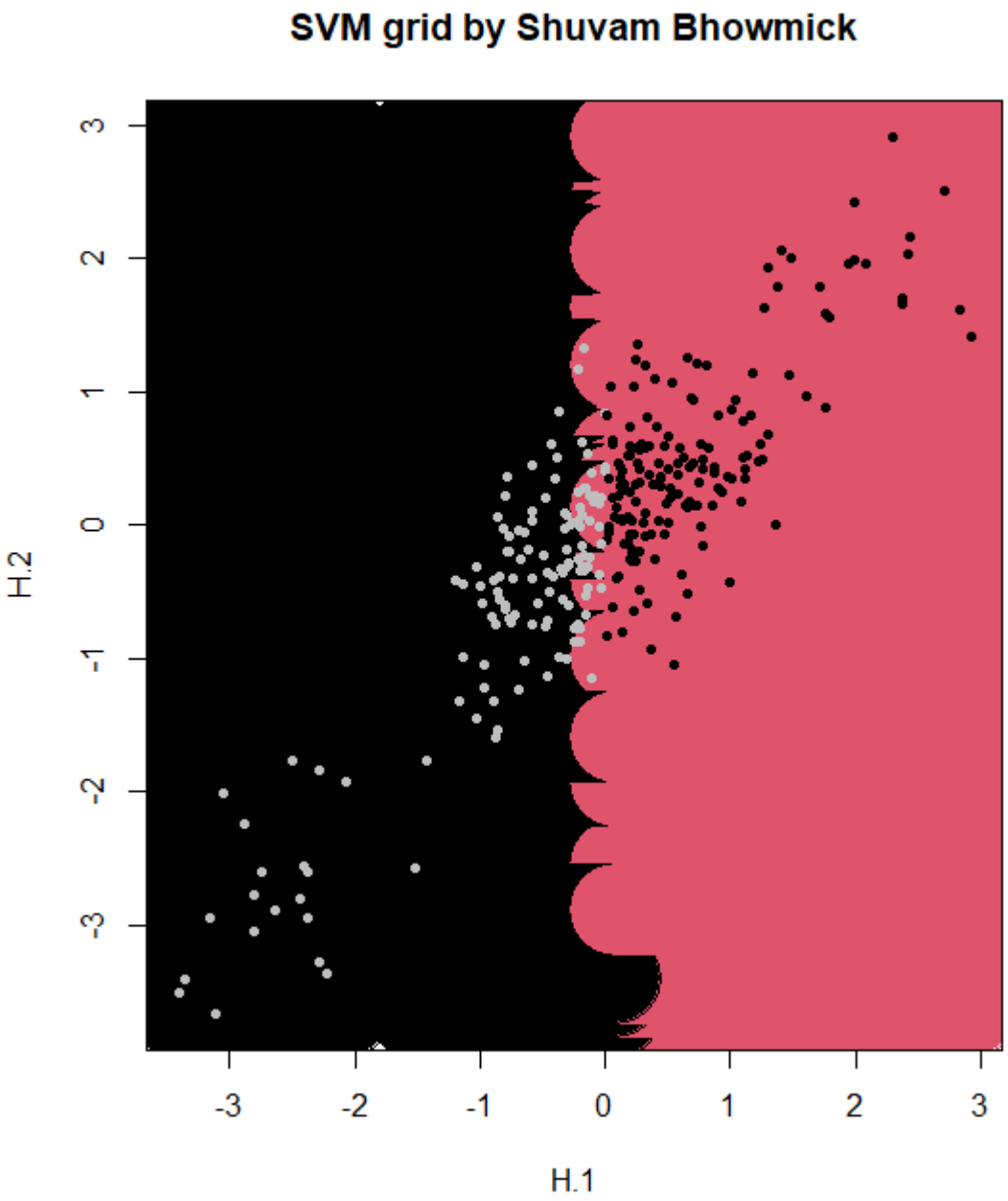


We will now need to implement a grid.

```
xgrid = expand.grid(H.1 = H[,1] , H.2 = H[,2] )
ygrid = predict(svmfit, xgrid)
plot(xgrid, col=as.numeric(ygrid), pch=20, cex=10)
```



```
svmfit = svm(F~., data=dat, cost=10,kernel = "linear", scale= FALSE)
L = ifelse(H[,1]>0 , "black" , "grey")
plot(xgrid, col=as.numeric(ygrid), pch=20, cex=10, main="SVM grid by Shuvam Bhowmick")
points(H, col = L, pch=20)
```

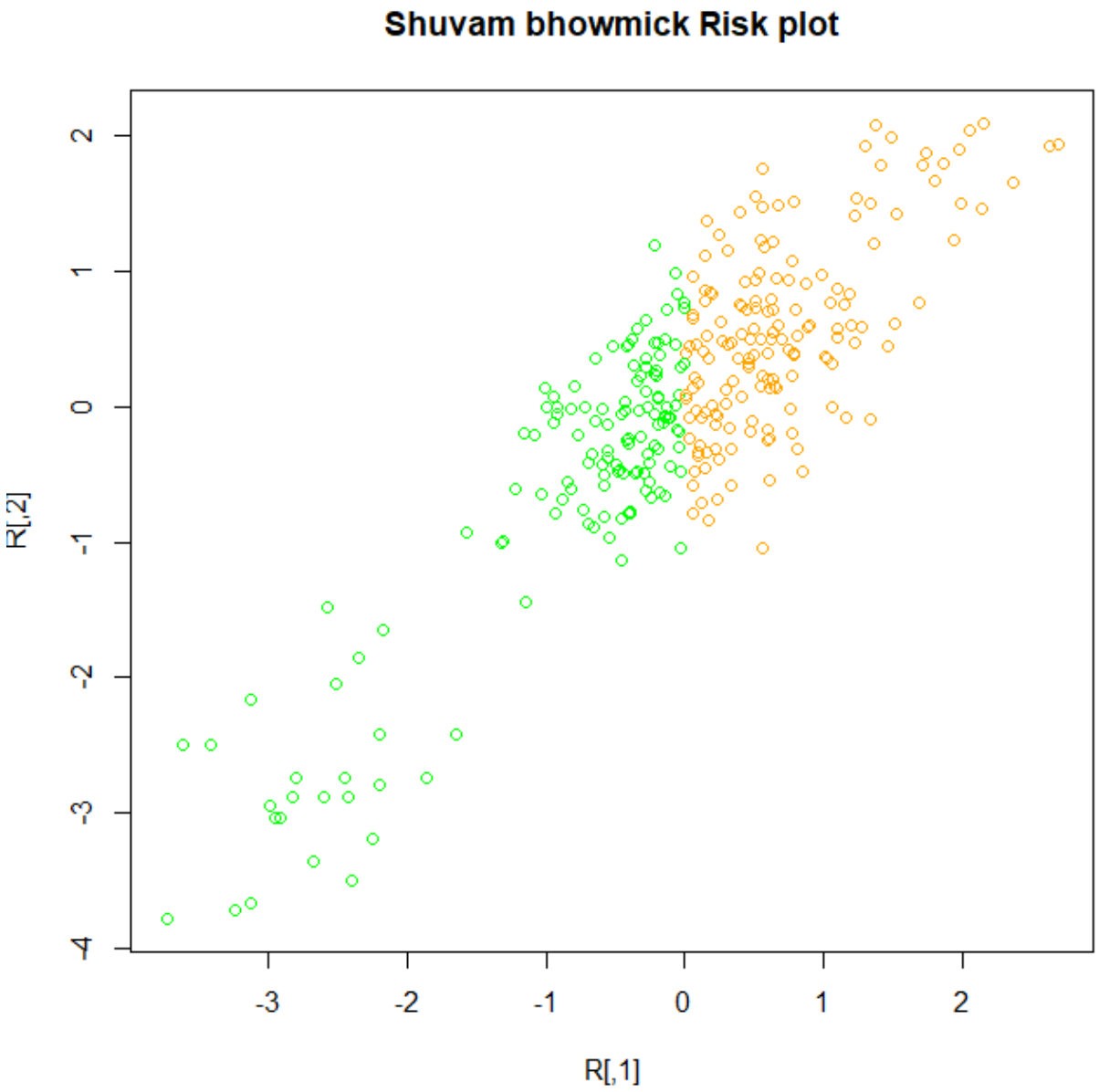


The grey points residing in the black part of the graph represents low return data based on the x and y-axis. The black points on the red side represent high return data. It makes sense that the predict() function in the code would pick a linear boundary. However, the boundary seems to spike inward and outward instead of being a straight line.

n=300 risk data set

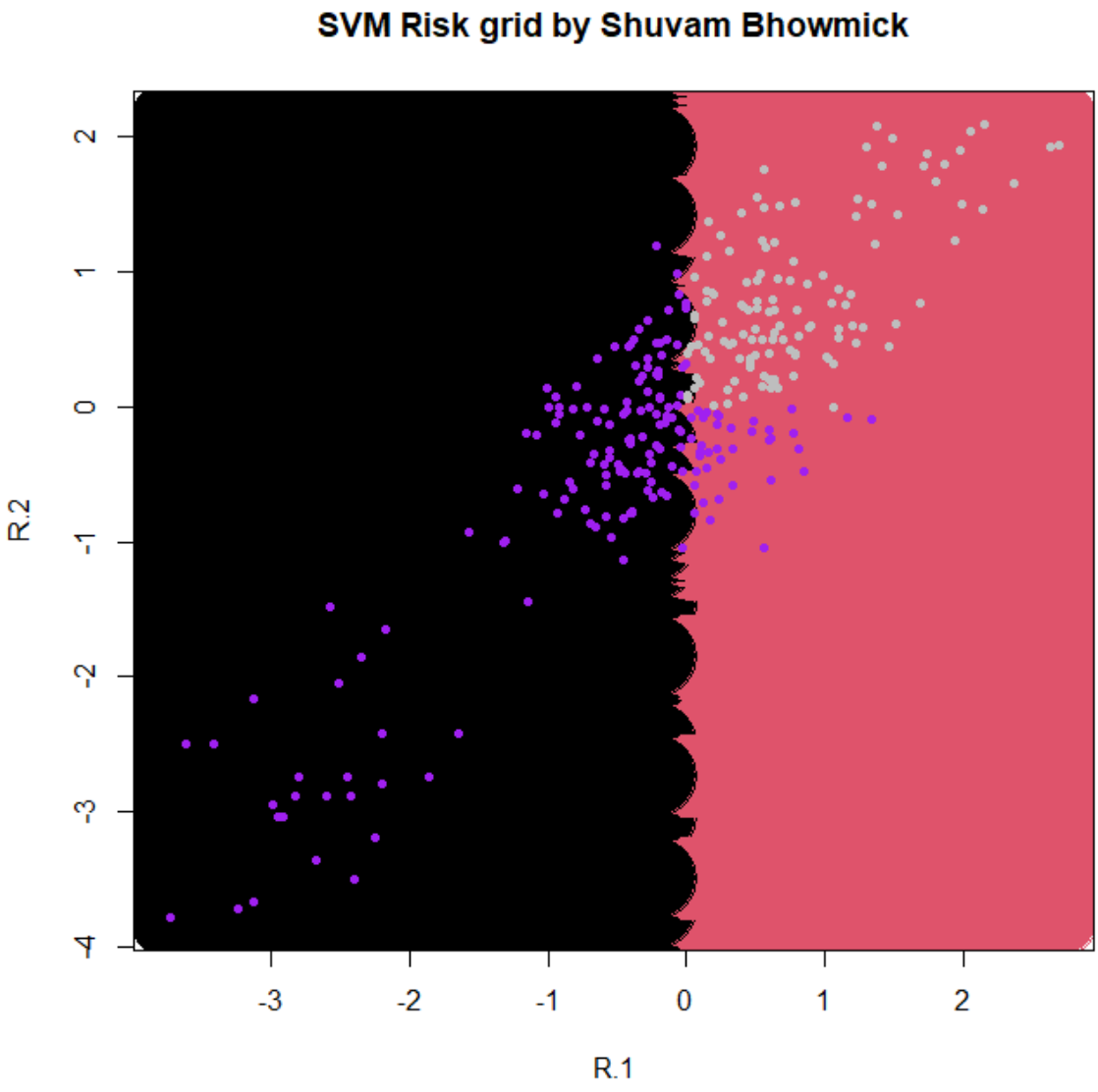
We will start off with just plotting the points of the matrix while using the condition of the first column being less than 1. This indicates orange points

```
R = (matrix(as.numeric((MatrixRisk[,2:3])),ncol=2))
plot(R, col = ifelse(R[,1] < 0 , "orange", "green"))
```



Green points represent low risk, while the red points represent high risk. We will now create an SVM classification plot using a somewhat linear boundary.

```
svmfit = svm(OPN~., data=datRiskN, kernel = "linear", cost=10, scale=FALSE)

xgrid = expand.grid(R.1 = R[,1], R.2 = R[,2])
ygrid = predict(svmfit, xgrid)
plot(xgrid,col=as.numeric(ygrid),pch=20, cex=10)
plot(xgrid,col=as.numeric(ygrid),pch=20, cex=10, main="SVM Risk grid by Shuvam Bhowmick")
GL = ifelse(R[,2]>0&R[,1]>0, "grey", "purple")
points(R, col=GL, pch=20)
```



As you can see, there are more classification errors for the risk plot than the return plot when we use a linear boundary. But overall, most of the purple points (or low risk points) are on the correct side of the classification while most the grey points are on the correct side of classification as well.

5. In the final part of this project, I will prepare a comparative study of knn, naïve Bayes, logistic regression ,and SVM using my stock and risk observations. Before I do that, I want to compare these methodologies and explain their significance.

KNN is a non-parametric method that uses a distance metric to find the "k-nearest neighbors[4]"of a point and predicts the labeling of the neighbors. The method is sensitive to the choice of k and distance metric. Naïve Bayes is a probabilistic method that makes predictions based on the Bayes theorem, which states that the probability of the label and the likelihood of the features given in the label. This method is efficient, but it makes an unrealistic assumption that the features are independent of each other. Logistic regression is a parametric method that uses a logistic function to model the relationship between the dependent variable and the independent variables. One downfall of the logistic regression method is that it can only model binary classification problems and assumes a linear relationship between the dependent and independent variables. SVM is a non-parametric method that uses a kernel function to map the data into a higher-dimensional space, where it finds the hyperplane that maximally separates the two classes. This method is effective in high dimensional spaces and can handle non-linear boundaries but can be sensitive to the cost parameter. Each of these methods have their advantages and disadvantages so the specific characteristic of the data and requirements of the task will determine which ones work best.

Knn Method

First, I will prepare my stock return and risk data so I can draw predictions and display a classification plot.

```
> CFH = CFstockRet300[,-1]
> table(CFH[,3])

HiRisk LoRisk
   165    135
> head(CFH)
         Lag2risk   Laglrisk Cfrisk
3788  0.443286485  0.2944485 HiRisk
536   0.352502224  0.3804844 HiRisk
3286  0.380484360  0.3043245 HiRisk
3747 -0.145944615  0.5356318 HiRisk
29   -2.381124818 -2.5945377 LoRisk
3662 -0.009685664  0.4078123 LoRisk
```

[4] "K-Nearest Neighbors." k-Nearest Neighbors - Python Tutorial. https://pythonbasics.org/k-nearest-neighbors/

```
> CFG = CFstockRets300[,-1]
> table(CFG[,3])

HiRet LoRet
  167   133
> head(CFG)
         Lag2ret     Lag1ret Cfret
291  -1.33600951  0.55070105 HiRet
4001  0.07501293 -0.09766977 LoRet
919  -0.26368474 -0.07034668 LoRet
3241  1.14945322  0.30523525 LoRet
2777 -0.56848421 -1.13788632 LoRet
1471  0.96224554 -0.14314606 LoRet
```

Now, I will use the first 150 observations as my training set and the final 150 observations as my testing set for both the return and risk data. I will use the k=11 parameter for the knn function

```
> predRet.knn = knn(CFH[1:150,1:2],CFH[151:300,1:2],CFH[1:150,3],11)
> table(predRet.knn, CFH[151:300,3])

predRet.knn HiRisk LoRisk
     HiRisk     69     16
     LoRisk     14     51
```

		Actual	
		HiRisk	LoRisk
Forecasts	HiRisk	69	16
	LoRisk	12	51

The forecast was correct 120/150 = 80% of the time for the risk data. Let's do the same thing for the return data set.

```
> predRet.knn = knn(CFG[1:150,1:2],CFG[151:300,1:2],CFG[1:150,3],11)
> table(predRet.knn, CFG[151:300,3])

predRet.knn HiRet LoRet
      HiRet    53    35
      LoRet    32    30
```

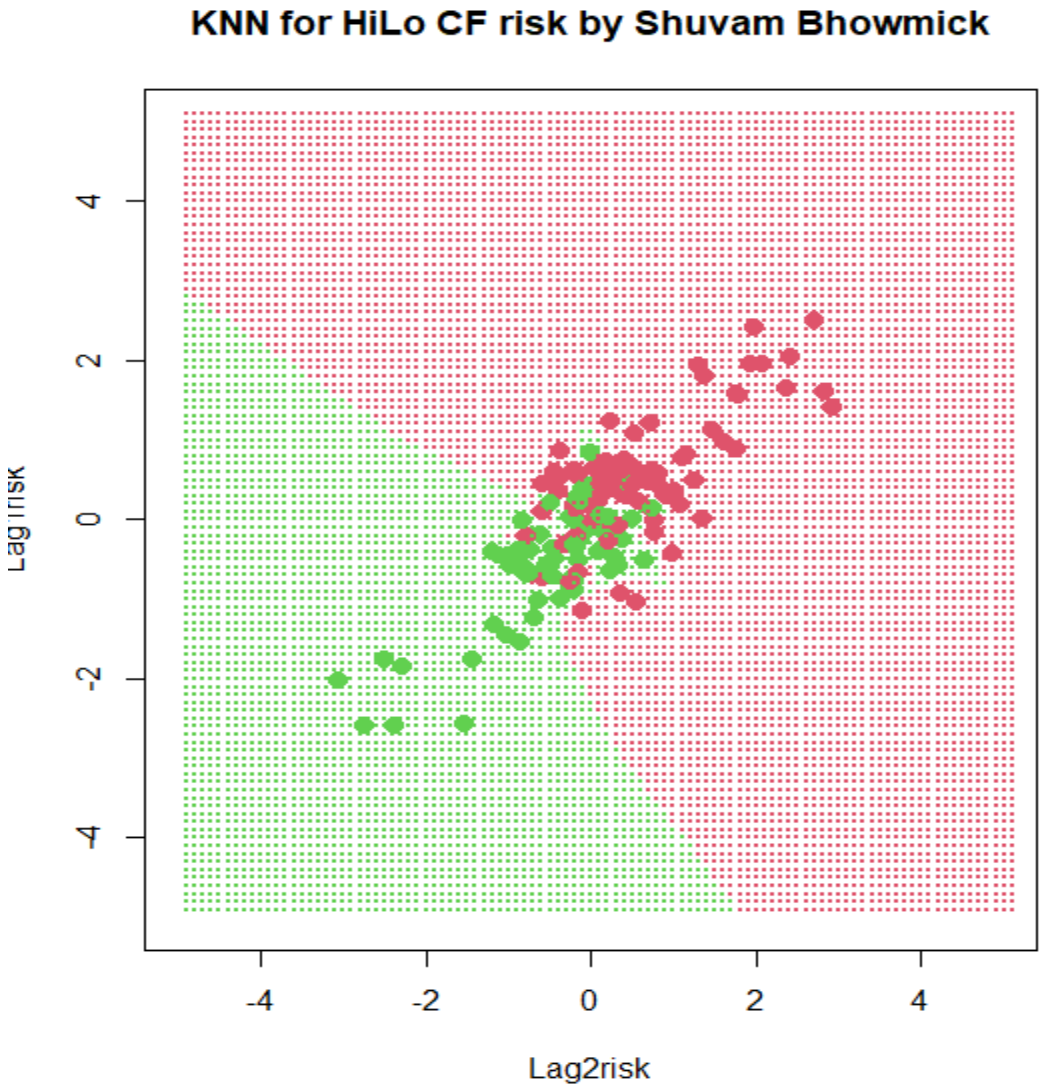
		Actual	
		HiRet	LoRet
Forecasts	HiRet	53	35
	LoRet	32	30

The forecast was correct 83/150 = 55% of the time.

The forecast for risk is more accurate than the forecast for returns.

```
plot(CFH[1:150,1:2], col = 1+unclass(factor(CFH[1:150,3])),pch=20, cex=2,xlim=c(-5,5),ylim=c(-5,5))
pred.knn = knn(CFH[1:150,1:2],stGrid, CFH[1:150,3],1)
points(stGrid, col=1+unclass(pred.knn), pch=20, cex=0.3, xlim=c(-5,5), ylim=c(-5,5))
title(main="KNN for HiLo CF risk by Shuvam Bhowmick")
```

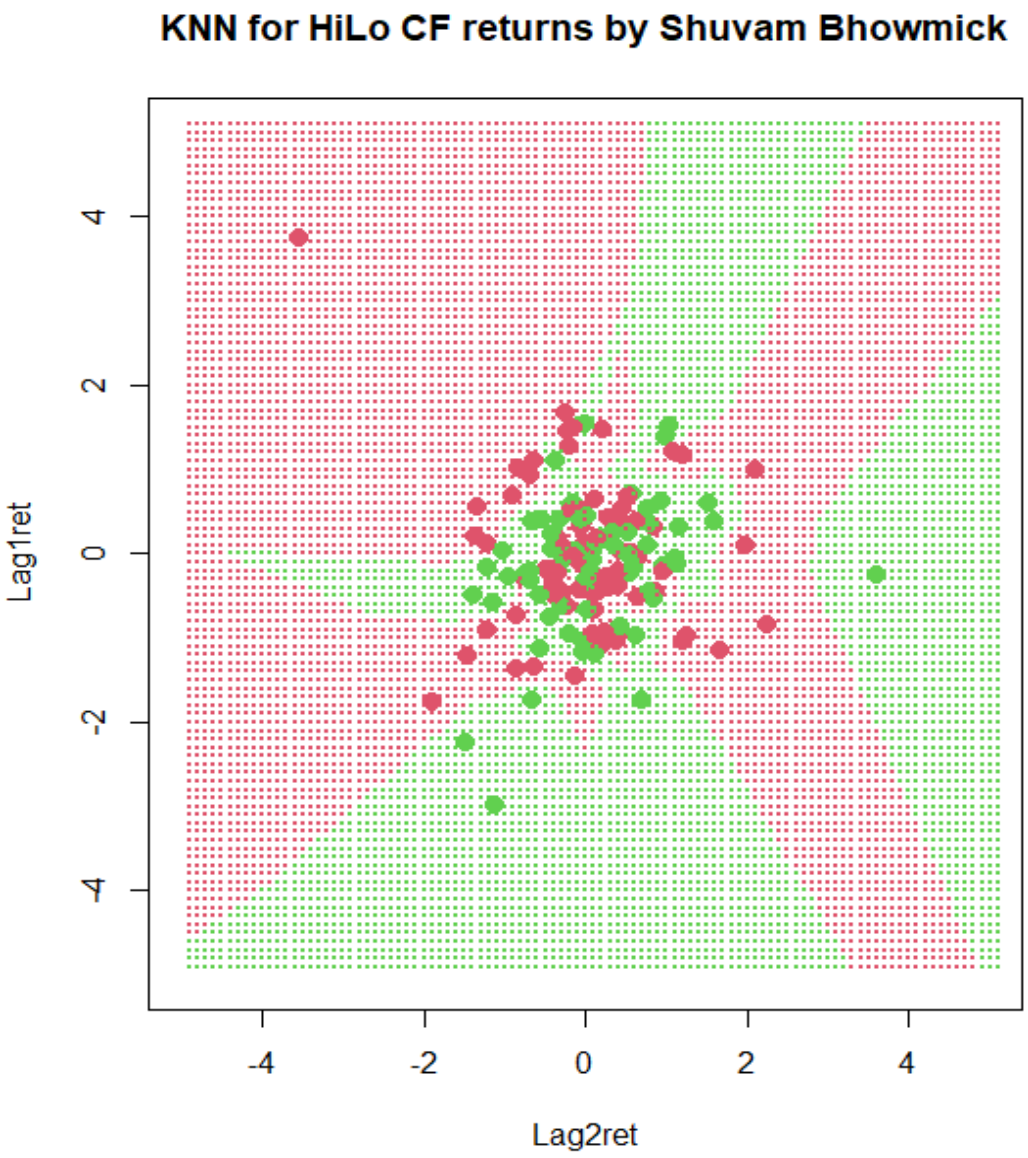
Classification plot for the risk Data set :



The knn classification plot created a diagonal separation between the high risk points and the low risk points.

Classification plot for the return Data set :

```
plot(CFG[1:150,1:2], col = 1+unclass(factor(CFG[1:150,3])),pch=20, cex=2,xlim=c(-5,5),ylim=c(-5,5))
pred.knn = knn(CFG[1:150,1:2],stGrid, CFG[1:150,3],1)
points(stGrid, col=1+unclass(pred.knn), pch=20, cex=0.3, xlim=c(-5,5), ylim=c(-5,5))
title(main="KNN for HiLo CF returns by Shuvam Bhowmick")
```



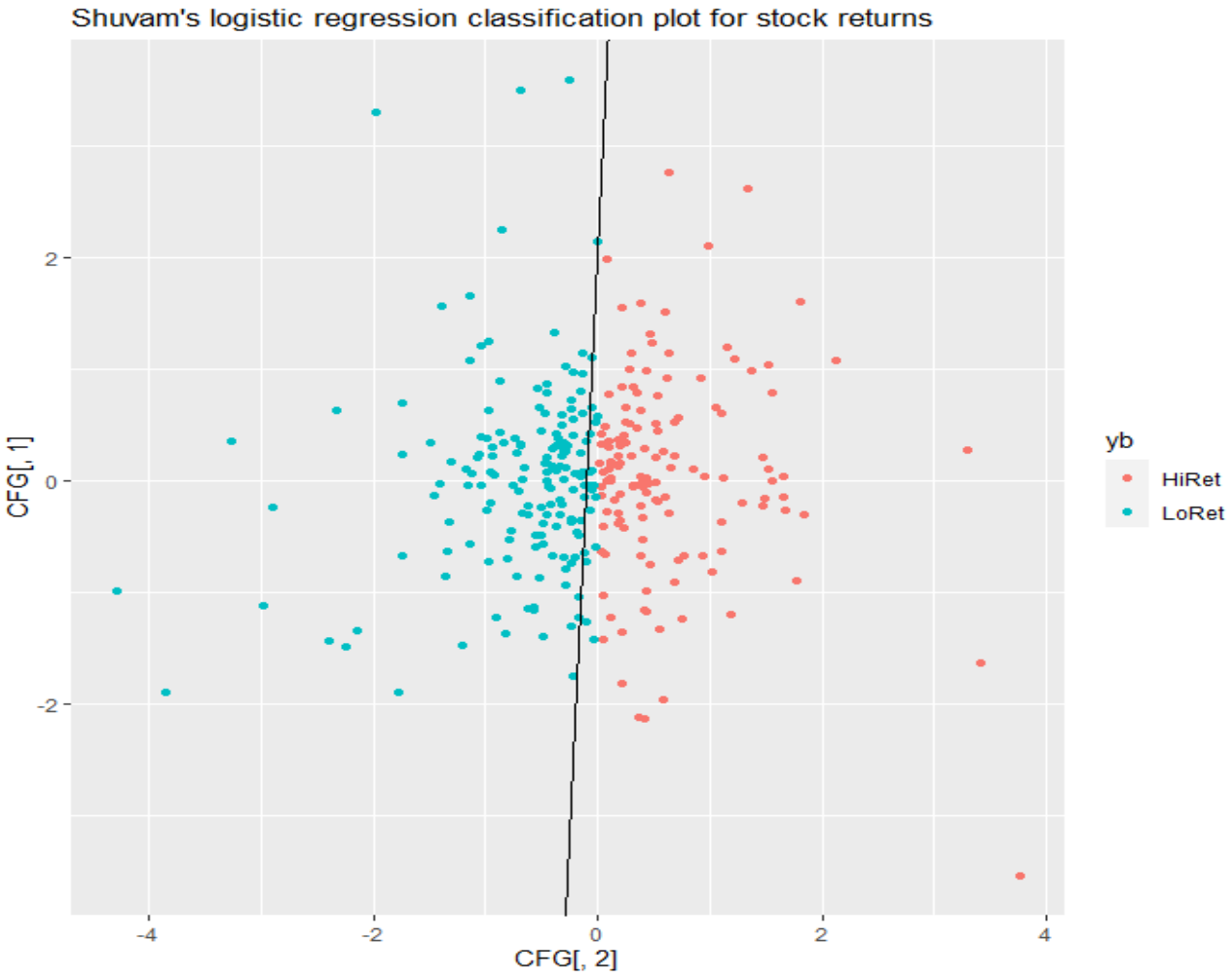
The separation between the data isn't very clear here. The clustering of points confused the knn function. Looks like the risk data was easier to classify than the return data based when using the KNN method.

Logistic Regression

Return Data Set

I will begin by creating a classification plot using my lag1 and lag2 predictors. I will use two libraries “dplyr” and “ggplot2” to create the plot.

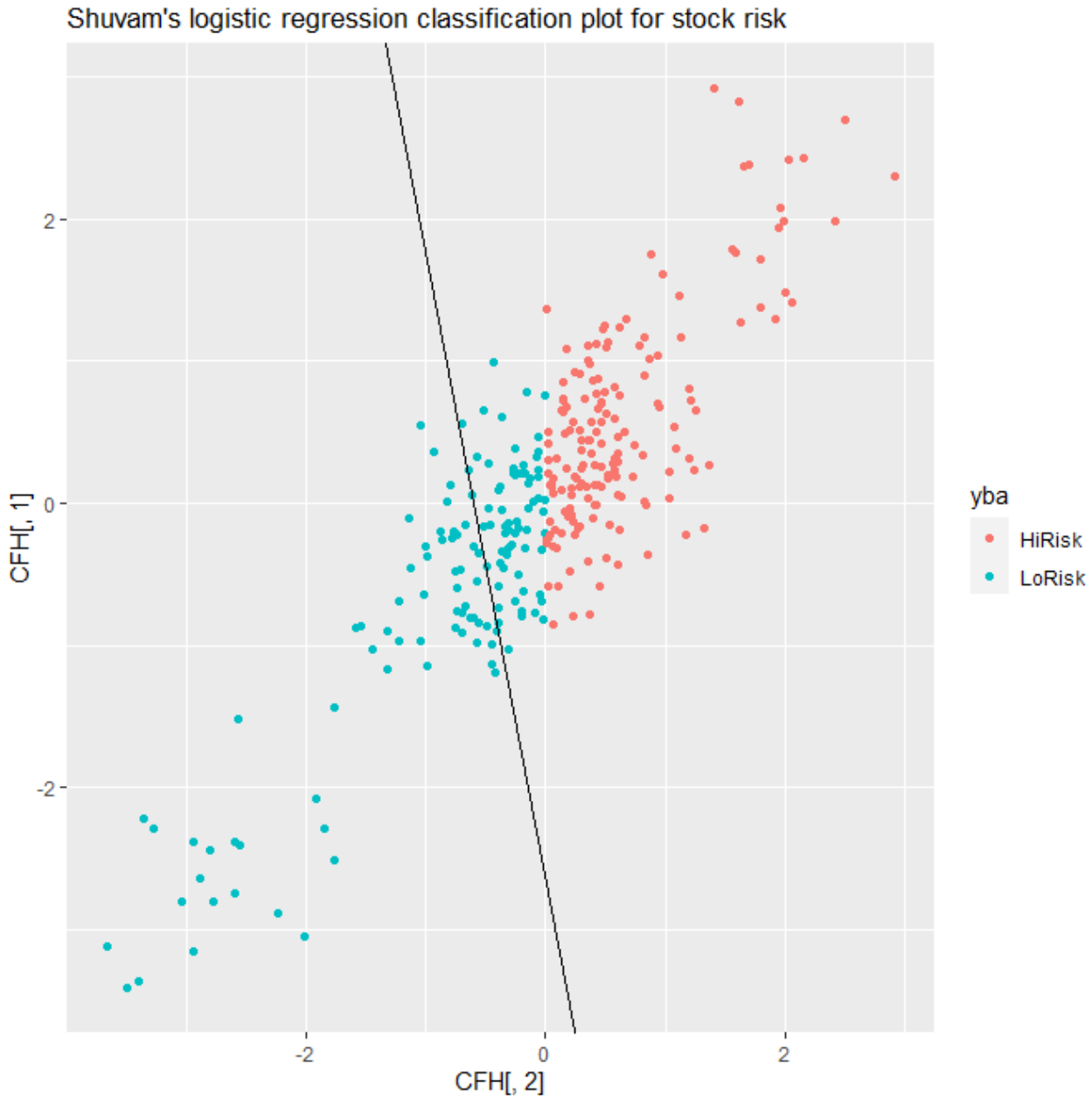
```
> library(dplyr)
> train_ind = sample(1:nrow(CFG), 0.8*nrow(CFG))
> train = CFG[train_ind,]
> test = CFG[-train_ind,]
> yb = as.factor(ifelse(CFG[,2] > 0, "HiRet", "LoRet"))
> model=glm(yb~CFG[,1] + CFG[,2], family = "binomial")
ggplot(data = model.frame(model), aes(x=CFG[,2], y=CFG[,1], color=yb)) + geom_point() + geom_abline(intercept = model$coefficients[1], slope=model$coefficients[2])
```



Explanation of code : the model function fits the data into a binomial regression which I then used for classification plot. I entered the lag1 and lag2 parameters into the ggplot functional arguments. I also created the linear separation line using the geom_abline() function.

Risk Data Set

```
> yba = as.factor(ifelse(CFH[,2] > 0, "HiRisk", "LoRisk"))
> model = glm(yba~CFH[,1] + CFH[,2], family = "binomial")
ggplot(data = model.frame(model), aes(x=CFH[,2], y=CFH[,1], color=yba)) + geom_point()
geom_abline(intercept = model$coefficients[1], slope=model$coefficients[2])
```



The decision boundary is much more slanted in the risk data set than the return. It also looks there are more misclassifications in the risk data set.

Predictions for Stock Risk and Return

```
library(dplyr)
train_ind = sample(1:nrow(CFG), 0.8*nrow(CFG))
train = CFG[train_ind,]
test = CFG[-train_ind,]
```

```
> model=glm(ybkh~CFG[1:60,1] + CFG[1:60,2], family = "binomial", data=train)
Warning messages:
1: glm.fit: algorithm did not converge
2: glm.fit: fitted probabilities numerically 0 or 1 occurred
> ybk = data.frame(yb)
> head(ybk)
     yb
1 HiRet
2 LoRet
3 LoRet
4 HiRet
5 LoRet
6 LoRet
> ybkh = ybk[1:60,]
> head(ybk[1:60,]
```

The data is split where 240 observations are training and 60 are testing. I will use 240 observations to predict the 60 observations.

I did the same procedure for the risk data but changed the prediction parameter and data rows to match the risk data.

```
> ybkhi = data.frame(yba)
> head(ybkhi)
     yba
1 HiRisk
2 HiRisk
3 HiRisk
4 HiRisk
5 LoRisk
6 HiRisk
```

```
predictions.logistic = predict(model, newdata = CFGN[151:300,3])
```

```
> table(predictions.logistic, CFGN[151:300,3])

predictions.logistic HiRisk LoRisk
             HiRisk     78     24
             LoRisk     15     33
```

For the Risk data , the logistic regression predictions were correct 111/150 = 74% of the time.

```
> table(predictions.logistic.Returns, CFG[151:300,3])

predictions.logistic.Returns HiRet LoRet
                      HiRet     50     37
                      LoRet     35     28
```

For the returns data, the logistic regression predictions were correct 78/150 = 52% of the time

Naïve Bayes

For this section, I am going to use the bayes classifier and predict function for n=300 randomly selected observations. First, I will use the naiveBayes function to train the data using the first 150 observations. Afterwards, I will use the predict function to classify the other 150 observations (we can call this the test set).

Return Data Predictions

```
> classifier.bayes = naiveBayes(CFG[1:150, 1:3], CFG[1:150,3])
> predict.bayes=predict(classifier.bayes, CFG[151:300,1:2])
> table(predict.bayes,CFG[151:300,3])

predict.bayes HiRet LoRet
        HiRet    73    55
        LoRet    12    10
```

Forecast is classified 83/150 = 55% correctly

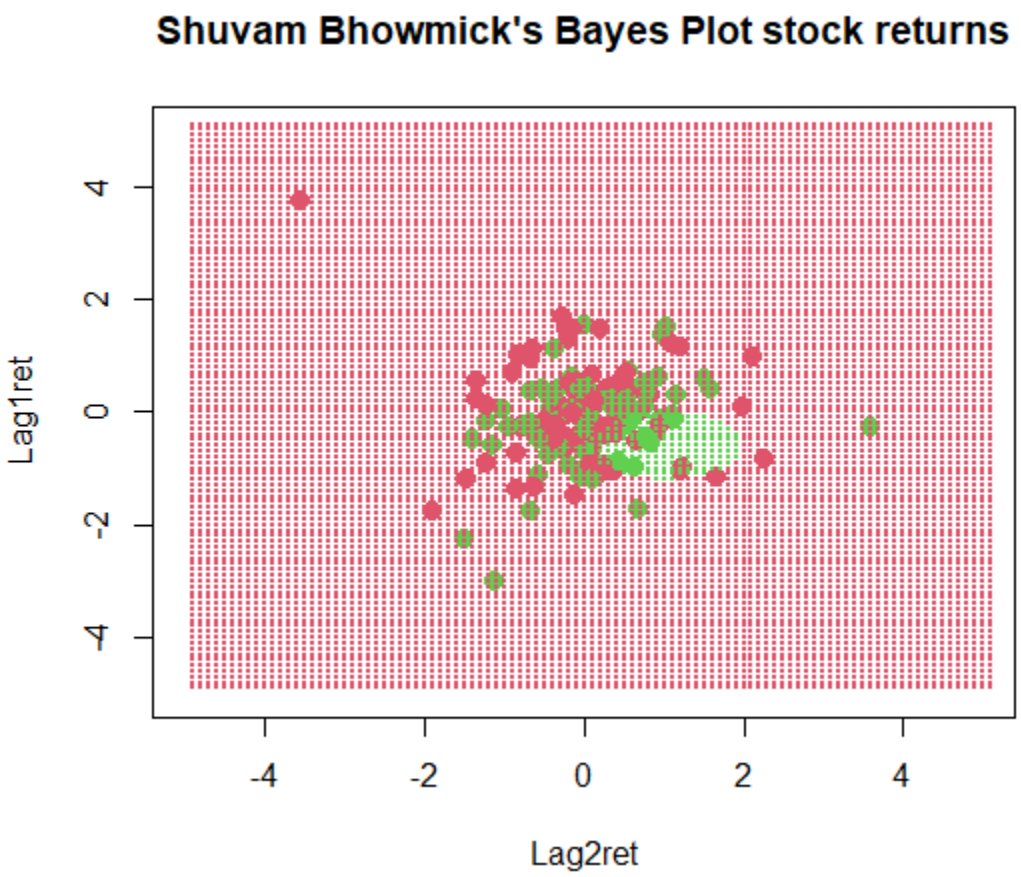
Risk Data Predictions

```
> classifier.bayes = naiveBayes(CFH[1:150, 1:3], CFH[1:150,3])
> predict.bayes=predict(classifier.bayes, CFH[151:300,1:2])
> table(predict.bayes,CFH[151:300,3])

predict.bayes HiRisk LoRisk
       HiRisk     69     17
       LoRisk     14     50
```

Forecast is classified 119/150 = 79% correctly

Return Data Classification Plot Naïve Bayes

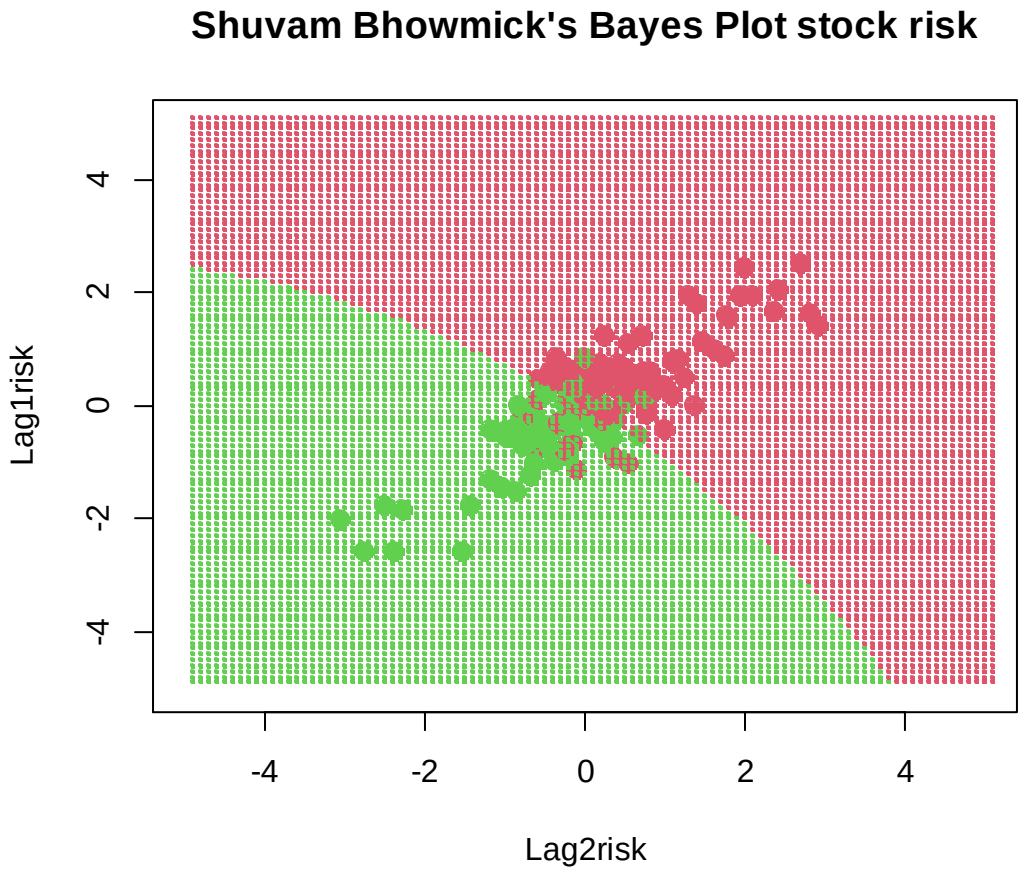
```
colnames(stGridNB) = colnames(CFG[,1:2])
classifier.bayes = naiveBayes(CFG[1:150, 1:2], CFG[1:150,3])
pred.stGridNB = predict(classifier.bayes, stGridNB)
plot(CFG[1:150, 1:2], col=1+unclass(factor(CFG[1:150,3])), cex=2,pch=20,xlim=c(-5,5),ylim=c(-5,5),main="Shuvam Bhowmick's Bayes Plot stock returns")
points(stGridNB, col=1+unclass(pred.stGridNB),pch=20,cex=0.3,xlim=c(-5,5), ylim=c(-5,5))
```



The classification is very unclear since the high return (green) points and the low return (red) points are clustered together. I'm not sure why the naïve bayes decided to cluster the points instead of separating them down the diagonal like svm. One conclusion that could be made is if yesterday's return ("Lag2ret") were low than today's returns would also be low given that there are more red points on the left side of the graph than the right.

Risk Data Classification Plot Naïve Bayes

```
classifier.bayes = naiveBayes(CFH[1:150, 1:2], CFH[1:150,3])
pred.stGridNB = predict(classifier.bayes, stGridNB)

plot(CFH[1:150, 1:2], col=1+unclass(factor(CFH[1:150,3])), cex=2,pch=20,xlim=c(-5,5),ylim=c(-5,5),main="Shuvam Bhowmick's Bayes Plot stock risk")
points(stGridNB, col=1+unclass(pred.stGridNB),pch=20,cex=0.3,xlim=c(-5,5), ylim=c(-5,5))
```



The Naïve Bayes classification method worked better for the risk data than the return data. We can see a clear boundary between the low risk and high-risk points here.

CFindustries Return Data (Using training data to classify new data)

SVM	Predicted	Low Return	Actual High Return
	Low Return	32	33
	High Return	42	43
			75/150 = 50% accurate
KNN		53	35
		32	30
			83/150 = 55% accurate
Logistic Regression		50	37
		35	28
			78/150 = 52% accurate
Naïve Bayes		73	55
		12	10
			83/150 = 55% accurate

** For svm we correctly predicted 32 low return data points, and 43 high return data points which gives us an accuracy rate of 50%. The other numbers are misclassifications**

CFindustries Risk Data (Using training data to classify new data)

SVM	Predicted	Actual: Low Risk	Actual: High Risk
	Low Risk	67	18
	High Risk	17	48
			115/150 = 77% accurate
KNN		69	16
		12	51
			120/150 = 80% accurate
Logistic Regression		78	24
		15	33
			111/150 = 74% accurate
Naïve Bayes		69	17
		14	50
			79/150 = 79% accurate

** For svm we correctly predicted 67 low risk data points, and 48 high risk data points which gives us an accuracy rate of 77% **

Overall, the returns were harder to forecast/classify compared to the risk of the CFindustries stock. The data suggests that it is easier to predict risk than it is to predict returns. Some research has suggested that predicting stock risk may be more difficult because it involves assessing the potential losses and "volatility"[5] of a stock, which can be affected by a wide range of internal and external factors. On the other hand, predicting stock return may be more straightforward because it is typically measured by the change in the stock price over a given period, which can be more easily quantified and modeled. Predicting stock risk and stock returns using classification methods such as Support Vector Machines (SVM), Naive Bayes, Logistic Regression, and K-Nearest Neighbors (KNN) involves different approaches and assumptions. SVM and Logistic Regression are models that can be used for binary classification, where the goal is to predict whether a stock will have high or low risk or return. On the other hand, Naive Bayes is a probabilistic model that can be used for multiclass classification, where the goal is to predict the specific class or category that a stock belongs to based on its risk or return characteristics. KNN is a non-parametric method that can be used for both binary and multiclass classification, and it makes predictions based on the similarity of the stock to its nearest neighbors in the feature space. Overall, the choice of classification method will depend on the specific characteristics of the stock dataset and the goals of the analysis.

[5] Finra.org, https://www.finra.org/investors/investing/investing-basics/risk

Appendix :

This Lab in broken up into two parts. In the first part, we will look at the Support Vector Classifiers and then Support Vector Machines. The concepts in both labs are essential in understanding the mechanisms behind SVM along with some of its parameters.

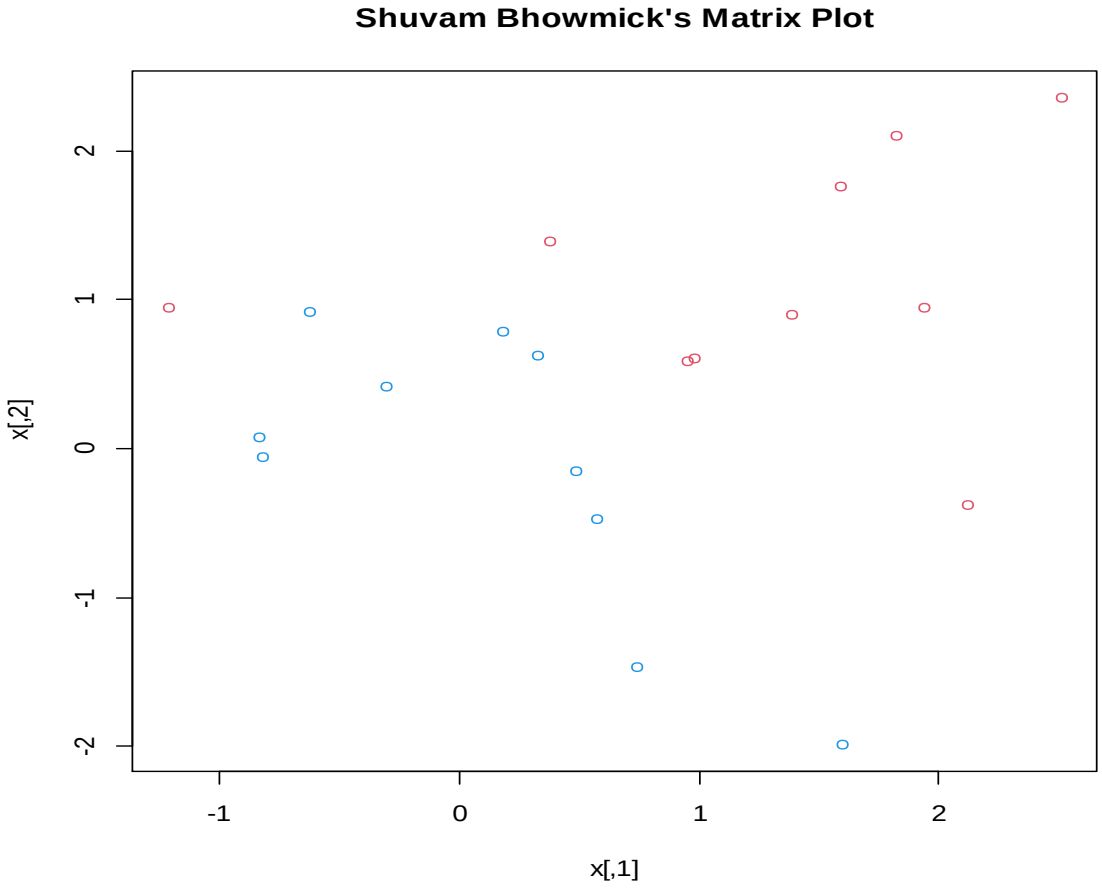
PART ONE : SUPPORT VECTOR CLASSIFIERS

I am going to present how to use the svm() function to fit the support vector classifier for a given value of a cost parameter. We are going to use the function on a two-dimensional example so we can plot the decision boundary.

First, I will generate observations and then check for a linear separation for the matrix plot.

```
set.seed(1)
x=matrix(rnorm(20*2), ncol=2)
y=c(rep(-1,10),rep(1,10))
x[y==1,]= x[y==1,] + 1
```

```
plot(x, col=(3-y), main = "Shuvam Bhowmick's Matrix Plot")
```



Looks like the classes for the plot are not linearly separable. For the svm() function to perform classification , I must encode the response as a factor variable using a dataframe.

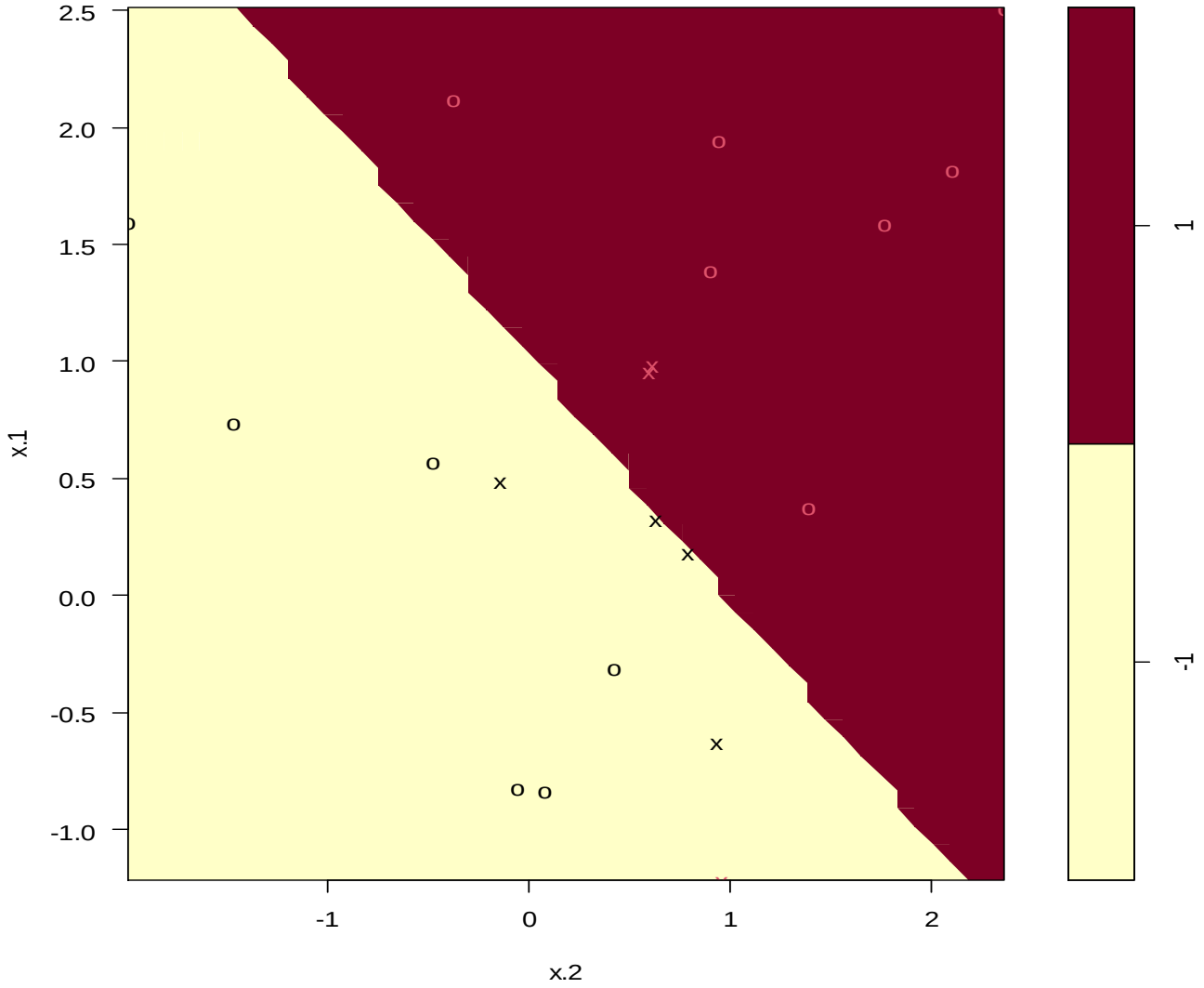
```
dat=data.frame(x=x, y=as.factor(y))
 library(e1071)

svmfit = svm(y~., data = dat , kernel = "linear", cost = 10, scale = FALSE)
```

The argument scale = FALSE tells the svm function not to scale each feature to have mean zero or standard deviation one. Depending on preferences, one might prefer scale=TRUE. A cost argument allows us to specify the cost of a violation to the margin. When the cost argument is small, then the margins will be wide and many support vectors will be on the margin or will violate the margin. When the cost argument is large, then the margins will be narrow and there will be few support vectors on the margin. We want to tune the C parameter so that we reduce overfitting by allowing some samples inside the margin but not eliminate the large margin properties that are beneficial for accurate classification. Now we will plot the support vector classifier obtained :

```
plot(svmfit, dat, main = "Shuvam Bhowmicks SVM Plot")
```

The two arguments , “svmfit” and “dat” are the output of the call to svm(), as well as the data used in the call to svm().

Shuvam Bhowmick’s SVM plot C = 10



The yellow region is assigned to the feature space of the -1 class while, the red region is assigned to the feature space +1. The decision boundary between the two classes is linear since we used the kernel = "linear" parameter in the svmfit function. The first feature is plotted on the y-axis while the second feature is plotted on the x-axis. Features define the classification characteristics. The support vectors are plotted as crosses and the remaining observations are plotted as circles.

```
> svmfit$index
[1]  1  2  5  7 14 16 17
```

We can identify the support vectors using the svmfit$index command. There are 7 support vectors in our plot.

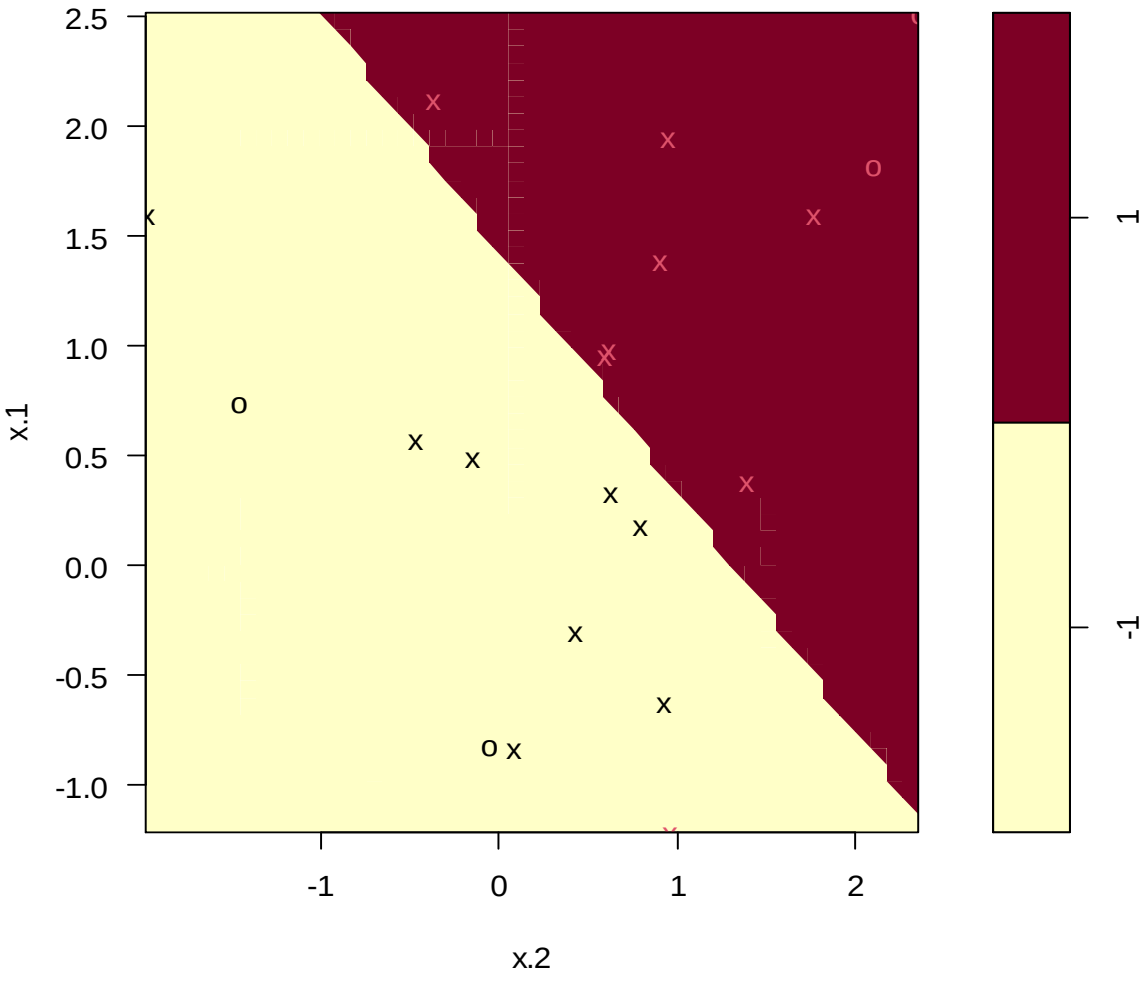
Here is the summary of the svmfit function:

```
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10

Number of Support Vectors:  7

 ( 4 3 )


Number of Classes:  2

Levels:
 -1 1
```

Now we will plot the linear kernel using a cost parameter of 0.1 instead of 10.

```
svmfit = svm(y~., data=dat, kernel = "linear", cost = 0.1, scale = FALSE)
plot(svmfit, dat)
```

Shuvam Bhowmick's SVM plot C = 0.1




```
> svmfit$index
 [1]  1  2  3  4  5  7  9 10 12 13 14 15 16 17 18 20
```

Because we used a smaller value for the cost parameter, we obtain a larger number of support vectors, since the margin is now wider. The symbols "x" indicates the support vectors. Based on the index function, there are 16 support vectors instead of 7.

The e1071 library includes a built-in function, tune(), to perform cross-validation. By default, tune() performs ten-fold cross-validation. The following arguments in the tune() function refers to the comparison of SVMs with a linear kernel while using a range of the cost parameter.

```
set.seed(1)
tune.out = tune(svm, y~., data=dat, kernel = "linear", ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
```

We can access the cross-validation errors for each of these models using the summary() command.

```
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.05

- Detailed performance results:
   cost error dispersion
1 1e-03  0.55  0.4377975
2 1e-02  0.55  0.4377975
3 1e-01  0.05  0.1581139
4 1e+00  0.15  0.2415229
5 5e+00  0.15  0.2415229
6 1e+01  0.15  0.2415229
7 1e+02  0.15  0.2415229
```

We can see that cost = 0.1 results in the lowest cross-validation error rate. The turne() function stores the best model obtained which we will access with the following command.

```
> bestmod=tune.out$best.model
> summary(bestmod)

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


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1

Number of Support Vectors:  16

 ( 8 8 )


Number of Classes:  2

Levels:
 -1 1
```

The predict() function is used to predict the class label on a set of test observations, at any given value of the cost parameter. Let's begin by generating a test data set.

```
xtest=matrix(rnorm (20*2) , ncol=2)
ytest=sample (c(-1,1), 20, rep=TRUE)
xtest[ytest==1,]= xtest[ytest==1,] + 1
testdat=data.frame(x=xtest , y=as.factor(ytest))
```

Now, we will predict the class labels of these test observations. We will then use the best model obtained through cross-validation in order to make predictions.

```
> ypred = predict(bestmod, testdat)
> table(predict = ypred, truth = testdat$y)
       truth
predict -1 1
     -1  9 1
     1   2 8
```

Based on the bestmod cost value of 0.1, 17 test observations are correctly classified.

Let's use cost = 0.01 to see what happens.

```
> svmfit = svm(y~., data=dat, kernel = "linear", cost = .01, scale = FALSE)
> ypred = predict(svmfit, testdat)
> table(predict = ypred, truth=testdat$y)
       truth
predict -1  1
     -1 11  6
     1   0  3
```

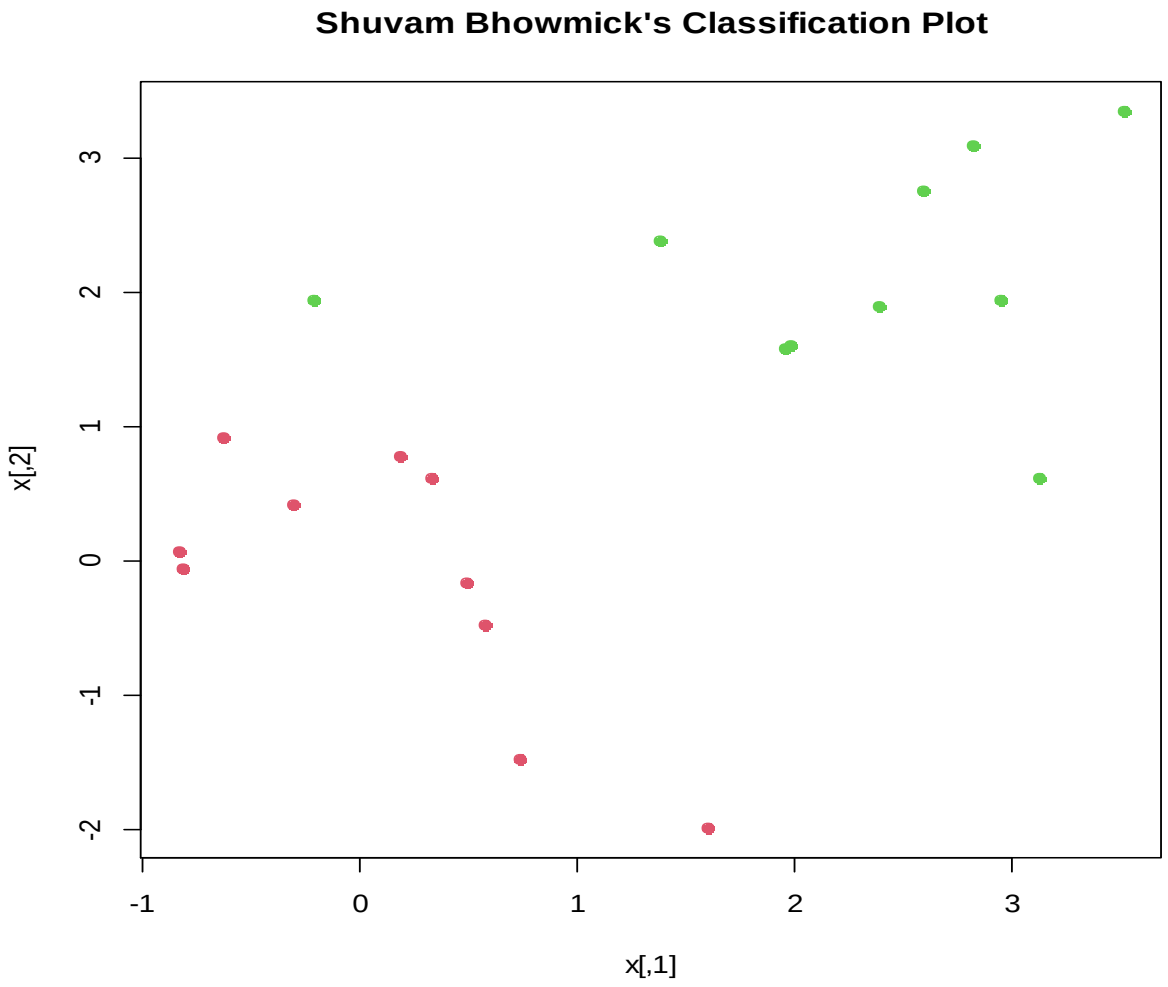
In this case only 14 were correctly classified which is 3 less than before.

Let’s consider a situation in which the classes in our simulated data are linearly separable.

```
x[y == 1. ] = x[y==1. ] + 0.5
```

*This Line of code further separates the two classes as shown in the plot.

```
plot(x, col = (y+5)/2, pch=19, main = "Shuvam Bhowmick's Classification Plot")
```



Because of the modifications, the plot is somewhat linearly separable.

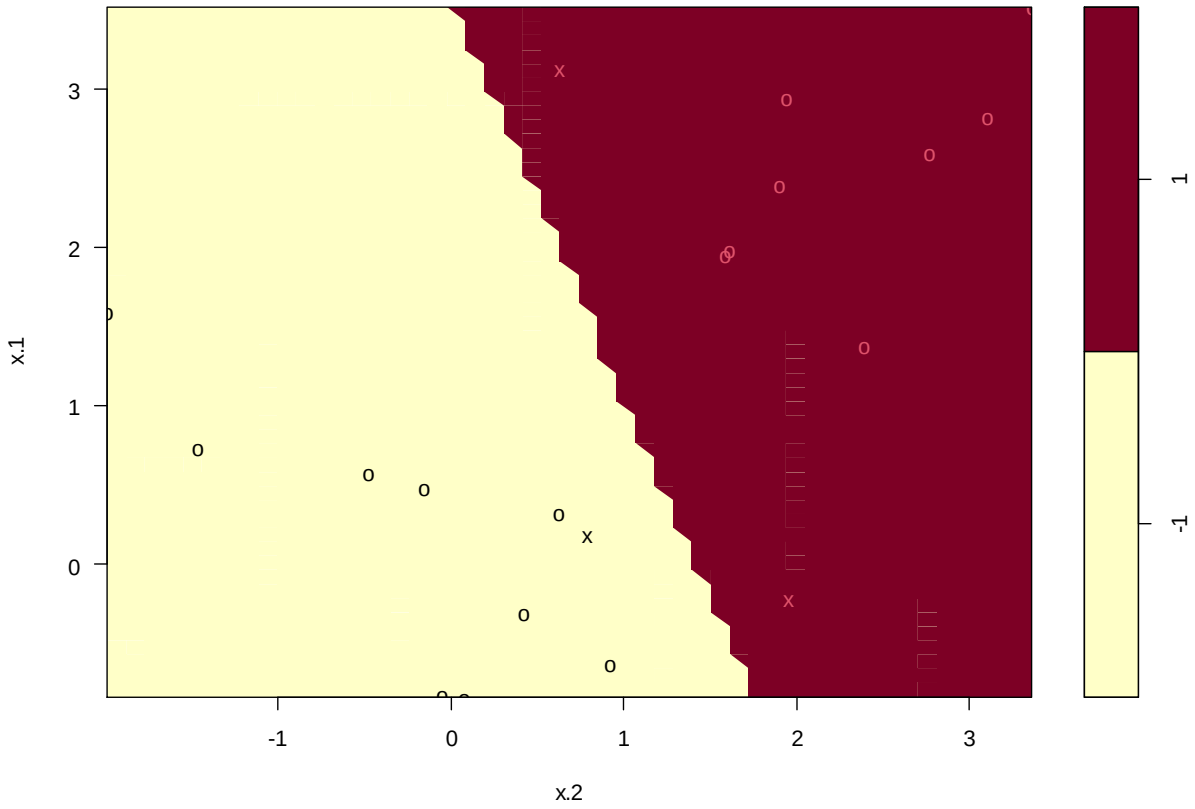
Now, we fit the support vector classifier and plot the resulting hyperplane, using a very large value of cost so that no observations are misclassified. The cost argument equal 10^5

```
dat=data.frame(x=x,y=as.factor(y))
```

```
>  svmfit=svm(y~., data=dat , kernel ="linear", cost=1e5)
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1e+05)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1e+05

Number of Support Vectors:  3

 ( 1 2 )


Number of Classes:  2

Levels:
 -1 1
```

Only 3 support vectors were used. Now, let's plot to see where the support vectors and observations are located.

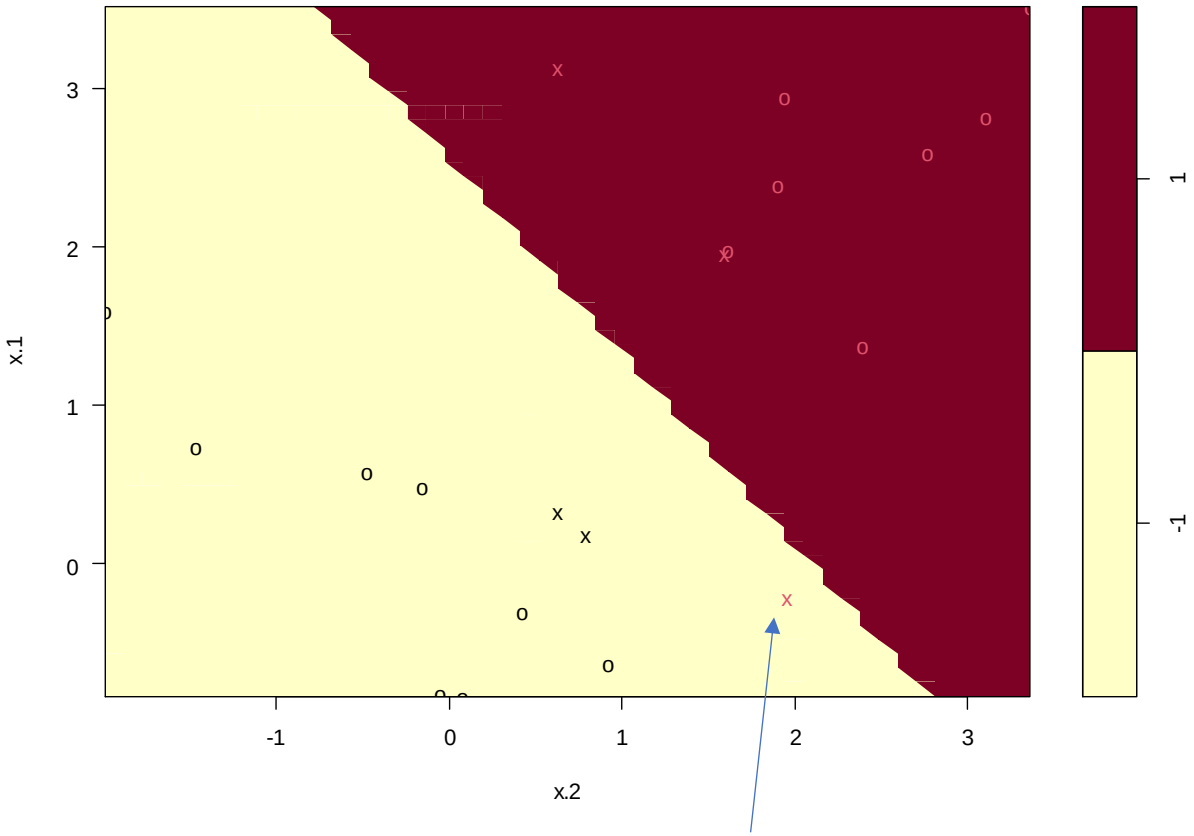
Shuvam Bhowmick's SVM Classification plot C = 10^5



The margins are very narrow since the observations that are not support vectors, indicated as circles, are very close to the decision boundary. However, not a single training error occurred here. All the red points are on the red side of the classification while all the black points are on the yellow side. Let's see the plot with a smaller value of cost.

```
> svmfit=svm(y~., data=dat , kernel ="linear", cost=1)
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1)


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

Number of Support Vectors:  5

 ( 2 3 )


Number of Classes:  2

Levels:
 -1 1
```

Shuvam Bhowmick's SVM plot C = 10^5



Using cost=1, there is one training observation that was misclassified.

The margin here is much wider and we are making use of 7 support vectors. This model will perform better on test data than the model with cost = 10^5 because of the wider margin and extra support vectors. Its unfortunate that one observation is misclassified but we have to introduce some bias to decrease the variance that will occur in the test data predictions.

PART 2 SUPPORT VECTOR MACHINES

We will continue using the Svm() function to fit data but this time with new arguments. We will fit an SVM with two different non-linear kernels : polynomial and radial. We will also use the degree argument to specify a degree for the polynomial kernel. For the radial kernel , we will use the gamma argument to specify the value of γ (omega) . Degree and Gamma are like the C parameter in that they adjust the functions of the decision boundary to better separate two classes in an SVM plot. The two non-linear decision boundary functions are :

1.

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d.$$

d = degrees

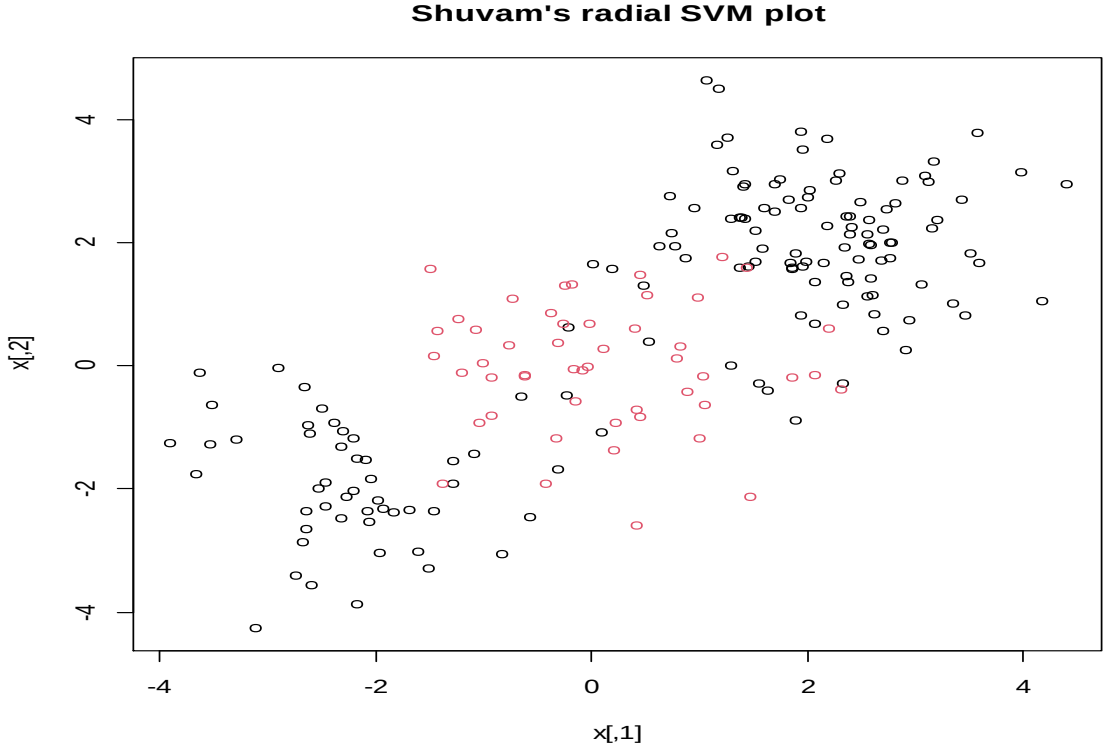
2.

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2).$$

γ = Omega

We will begin by generating some data with a non-linear class boundary

```
set.seed(1)
x = matrix(rnorm(200*2), ncol = 2)
x[1:100,]=x[1:100,]+2
x[101:150 ,]=x[101:150,]-2
y=c(rep(1,150) ,rep(2,50))
 dat=data.frame(x=x,y=as.factor(y))   plot(x, col = y, main = "Shuvam's radial SVM plot")
```

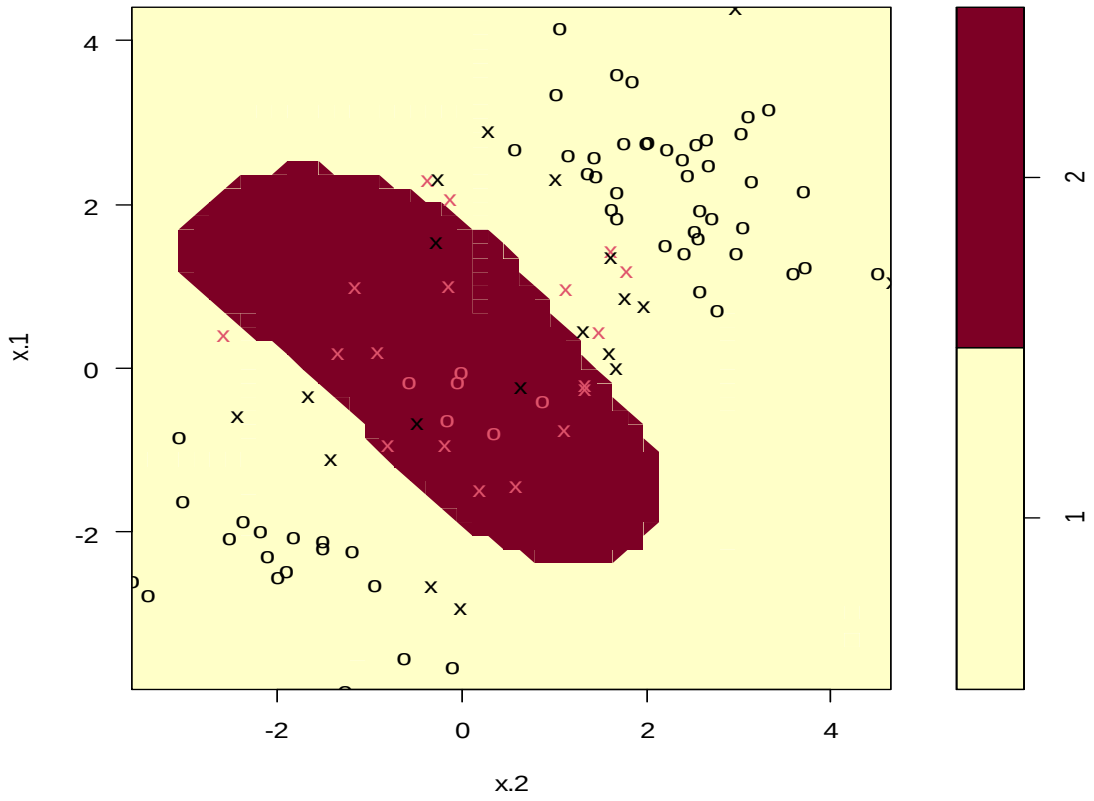


Based on the plot, it's clear that the decision boundary will be non-linear.

The data is randomly split into training and testing groups. We then fit the training data using the svm() function with a radial kernel and $\gamma = 1$

```
train=sample (200,100)
svmfit=svm(y~., data=dat[train ,], kernel ="radial", gamma=1, cost = 1)
plot(svmfit, dat[train,], main = "shuvam's radial plot with training data")
```

shuvam's radial plot with training data



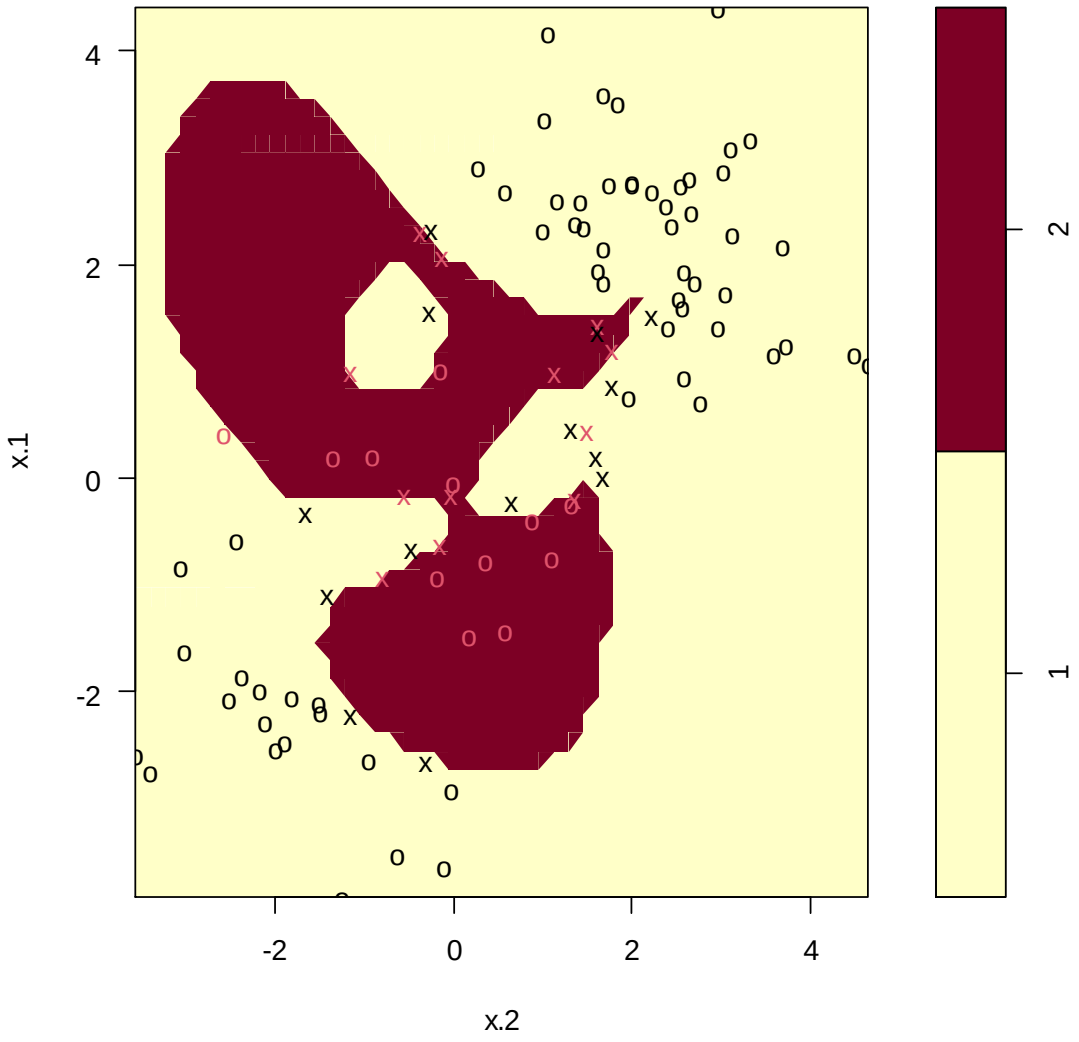

```
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat[train, ], kernel = "radial", gamma = 1, cost = 1)


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

Number of Support Vectors:  37

 ( 18 19 )


Number of Classes:  2

Levels:
 1 2
```

The summary() function can be used to obtain some information about the SVM fit:

We can see the decision boundary is oval shaped and non-linear. In the plot, we also see a fair number of training errors. If we increase the value of cost, we can reduce the number of training errors. However, this comes at a price of a more irregular decision boundary that seems to be at risk of overfitting the data.

```
svmfit = svm(y~. , data = dat[train,], kernel = "radial", gamma = 1, cost = 1e5)
plot(svmfit, dat[train,], main = "shuvam's radial svm plot with c = 10^5")
```

shuvam's radial plot with training data



The plot almost looks like artwork! The shape of the decision boundary is much more irregular compared to the plot with C = 1. However, this plot seems to have less training data errors.

We can perform cross-validation using tune() to select the best choice of γ and cost for an SVM with a radial kernel.

```
tune.out = tune(svm, y~., data=dat[train,], kernel = "radial",
ranges = list(cost=c(0.1, 1, 10, 100, 1000), gamma = c(0.5,1,2,3,4)))
```

```
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross vali

- best parameters:
 cost gamma
    1     2

- best performance: 0.13

- Detailed performance results:
    cost gamma error dispersion
1  1e-01   0.5  0.24 0.20655911
2  1e+00   0.5  0.14 0.10749677
3  1e+01   0.5  0.15 0.09718253
4  1e+02   0.5  0.18 0.12292726
5  1e+03   0.5  0.17 0.14944341
6  1e-01   1.0  0.24 0.20655911
7  1e+00   1.0  0.14 0.09660918
8  1e+01   1.0  0.18 0.12292726
9  1e+02   1.0  0.19 0.16633300
10 1e+03   1.0  0.19 0.13703203
11 1e-01   2.0  0.24 0.20655911
12 1e+00   2.0  0.13 0.11595018
13 1e+01   2.0  0.19 0.17919573
14 1e+02   2.0  0.17 0.15670212
15 1e+03   2.0  0.20 0.14142136
16 1e-01   3.0  0.24 0.20655911
17 1e+00   3.0  0.15 0.10801234
18 1e+01   3.0  0.16 0.15776213
19 1e+02   3.0  0.19 0.12866839
20 1e+03   3.0  0.22 0.17511901
21 1e-01   4.0  0.24 0.20655911
22 1e+00   4.0  0.17 0.14944341
23 1e+01   4.0  0.16 0.15776213
24 1e+02   4.0  0.20 0.13333333
25 1e+03   4.0  0.21 0.17919573
```

Best choice of parameters involves

Cost = 1 and gamma = 2

We can view the test set predictions for this model by applying the predict() function to the data. To do this we subset the dataframe dat using -train as an index set.

```
> table(true = dat[-train,"y"], pred = predict(tune.out$best.model,newx=dat[-train,]))
    pred
true  1  2
   1 59 15
   2 20  6
```

35% of the observations are misclassified

Works Cited

1. “History.” CF Industries. https://www.cfindustries.com/who-we-are/history

2. Tumminello, Aurora. “Statistical Models Part II.” Chapter 11 Support Vector Machines. https://bookdown.org/aurora_tumminello/statistics_lab/support-vector-machines.html

3. Kuhn, Max. “The Caret Package.” 5 Model Training and Tuning, March 27, 2019. https://topepo.github.io/caret/model-training-and-tuning.html

4. “K-Nearest Neighbors.” k-Nearest Neighbors - Python Tutorial. https://pythonbasics.org/k-nearest-neighbors/

5. Finra.org, https://www.finra.org/investors/investing/investing-basics/risk

6. James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. “Chapter 9 : Support Vector Machines.” Essay. In An Introduction to Statistical Learning: With Applications in R. Boston: Springer, 2021. https://hastie.su.domains/ISLR2/ISLRv2_website.pdf