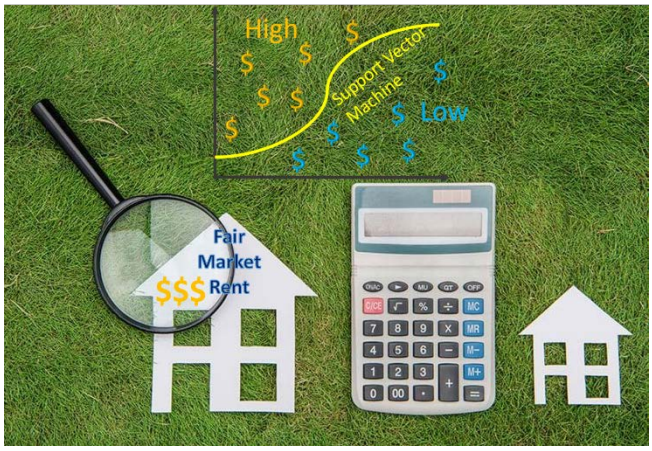


DETERMINING HOUSING AFFORDABILITY BASED ON FAIR MARKET RENT USING SVM METHODOLOGY

ABSTRACT

Employing the Support Vector Machine (SVM) methodology to classify the Housing Affordability Data System (HADS) dataset to determine the housing affordability based on the Fair Market Rent

Tanaya Nandanwar

CIS 9660 Data Mining

Table of Contents

Introduction

For my final project, I have chosen to implement Support Vector Machine or SVM as the classification algorithm for my Housing Affordability Data System (HADS) dataset. Here, I aim to classify the housing affordability based on the Fair Market Rent. The Housing Affordability Data System (HADS) is a set of housing-unit level datasets that measures the affordability of housing units and the housing cost burdens of households, relative to area median incomes, poverty level incomes, and Fair Market Rents.[1] I would use a few variables from this dataset to determine the affordability of houses in the US based on the Fair Market Rent (FMR).

Basic concept of SVM

Let us consider two features – x and y. Based on x and y, we aim to classify the data into either of two classes – red or blue. Refer Figure1 for the labeled training data. An SVM would take these data points and outputs a best hyperplane that divides the red and blue points. In Figure 2, we can see a line as the best hyperplane separating the points. The blue points lie to the left side of the hyperplane while red points lie to the right side. Thus, the position of points respective to the hyperplane can determine its class – red or blue.

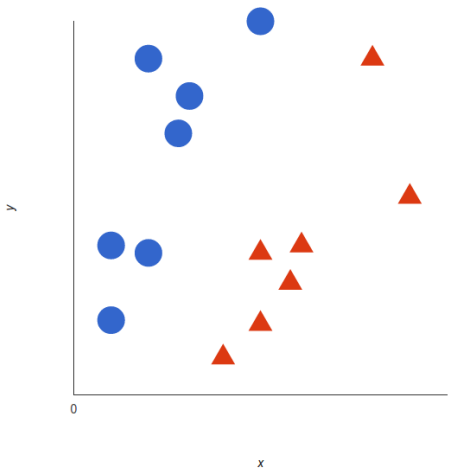


Figure 1: Our labeled data [2]

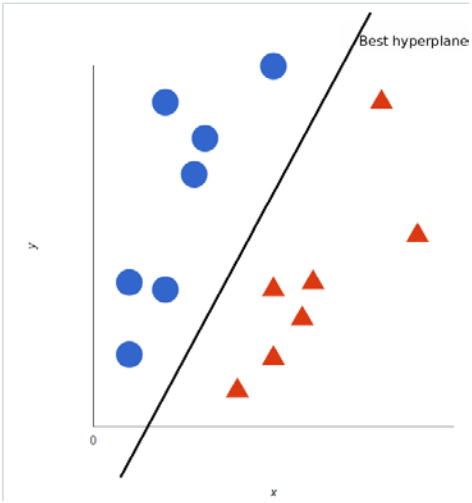


Figure 2: In 2D, the best hyperplane is simply a line [3]

A best hyperplane also maximizes its distance from the nearest elements of each class. This can be well demonstrated in the adjacent Figure 3.

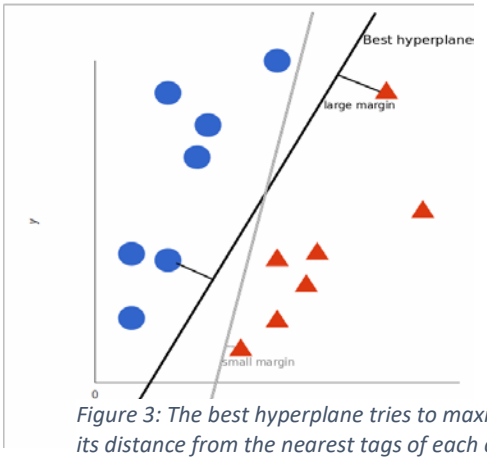


Figure 3: The best hyperplane tries to maximize its distance from the nearest tags of each class [4]

[1] Source: https://www.huduser.gov/portal/datasets/hads/HADS_doc.pdf
[2,3,4] Source: https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

Now consider a distribution of points where the hyperplane cannot correctly distinguish between all the points. Sometimes it is worthwhile to misclassify a few training observations rather than over fit the data. This helps in better classifying the future observations. The extent, to which we seek a margin (Margin is as shown by dotted lines in the adjacent figure) where we allow a few misclassifications, can be set by a tuning parameter called as *cost or C*. The larger the value of C the smaller is the margin and smaller the misclassification tolerance and vice-versa.

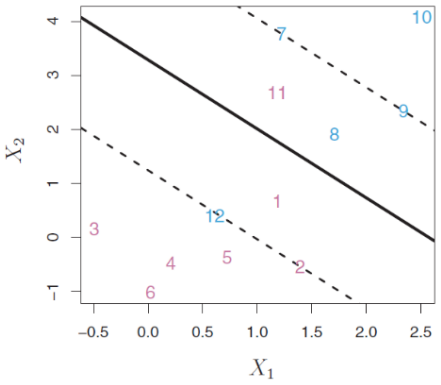


Figure 4: Position of data points relative to the margin and the hyperplane [4]

Some of the points lie on the wrong side of hyperplane and some on the wrong side of the dotted margin or some on the wrong side of both the margin as well as the hyperplane. Points 3, 4, 5, 6, 10 lie on the correct side of the hyperplane and margin. Points 2, 7, 9 lie on the correct side of hyperplane, but on the margin. Points 11, 12 lie on the wrong side of margin and hyperplane.

The points, which lie on the margin and within it, are known as *support vectors*. An addition or deletion of points to support vectors might drastically affect the best hyperplane because these points lie closest to it. In addition, we can note that C controls the bias-variance tradeoff of the support vector classifiers.

Walking through an example – Support Vector Classifier*

I will use the e1071 library which contains the svm() function to fit a support vector classifier for any given data. To demonstrate the use of svm function, I will first create a few observations, which belong to either of two classes. I will create a two dimensional x-matrix that will be used as an input to the svm() function. The expected output will be y, which has a value of either 1 or -1.[5]

I have set the seed to make the dataset replicable to the one in ISL text. Then I created the x matrix of random data points. The y matrix is the expected class of the data point as 1 or -1.

```
> 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
```

Let us check the plot of the data points. The red points show the class=1 and blue points class=-1

```
> plot(x, col =(3-y))
```

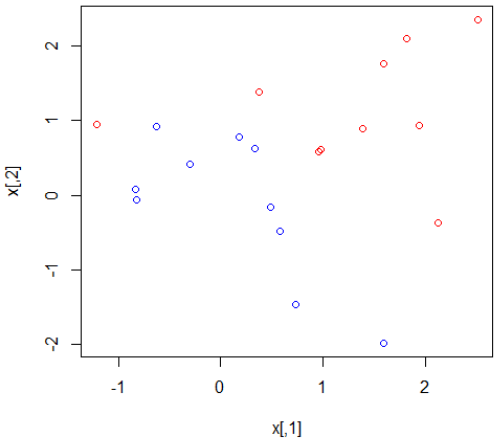


Figure 5: Plotting the labelled data

Before using svm() function, we need to integrate all the data in a single dataframe. For svm() function to classify correctly we should convert the y as a factor datatype. I used the following code to accomplish that. To use the svm() function, we have to first include the e1071 library in our workspace. Hence, I used the command library(e1071). In the svm() function, kernel="linear" specifies we will use support vector classifier while cost can be used to specify

---

* The R code used in this section is taken from the ISLR Text, An Introduction to Statistical Learning, 359-362

[4] ISLR Text, An Introduction to Statistical Learning, Figure 9.6, 346

[5] ISLR Text, An Introduction to Statistical Learning, 9.6.1 Support Vector Classifier, 359

the tolerance of violations to the margin. Scale= FALSE helps not to scale each feature to mean zero and standard deviation of one.

```
> dat=data.frame(x=x, y=as.factor (y))
> library (e1071)
> svmfit =svm(y~., data=dat , kernel ="linear", cost=10, scale=FALSE)
```

Let us now check the support vector classifier obtained above:

```
> plot(svmfit , dat)
```

The classification space is nicely highlighted in pink and blue. We have only one misclassified point, which is highlighted in red in the adjacent figure. The support vectors are shown as crosses and the remaining points are shown as circles. The support vectors can be identified as follows:

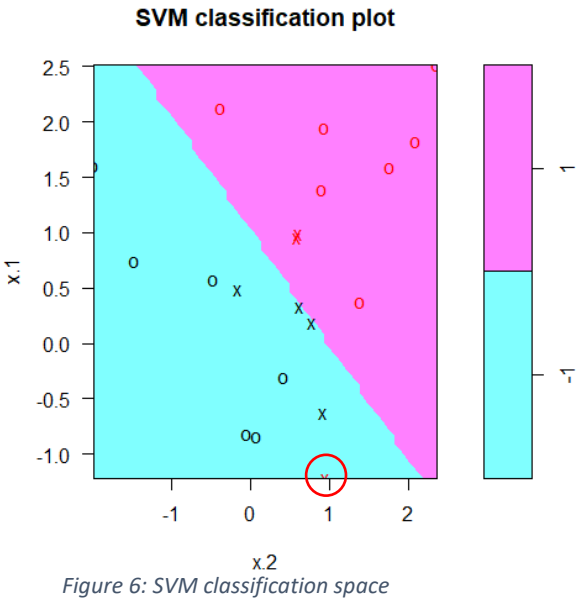


Figure 6: SVM classification space

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

An information about the support vector classifier can be obtained using the summary() function. We can note again that there are 7 support vectors. 4 belonging to one class and 3 to the other.

```
> summary (svmfit )

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


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.5

Number of Support Vectors:  7

 ( 4 3 )


Number of Classes:  2

Levels:
 -1 1
```

Let us now try to reduce the value of cost to 0.1. This means we are allowing a wider margin. This leads to an increase in the number of support vectors. We can note the increased number of support vectors as below:

```
> svmfit =svm(y~., data=dat , kernel ="linear", cost=0.1, scale=FALSE)
> plot(svmfit , dat)
> svmfit$index
 [1]  1  2  3  4  5  7  9 10 12 13 14 15 16 17 18 20
```

The increased number of support vectors are also shown in the below classification space as crosses:

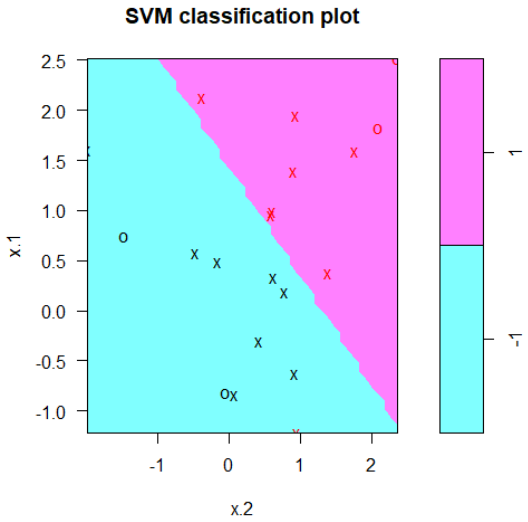


Figure 7: Increased number of support vectors when cost is reduced

We can perform a ten-fold cross-validation on our models of interest using the tune() function. We can also specify a list of cost parameters. Using the below command we wish to compare our SVM models:

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

The error rates for different models can be found out using the summary() function: We can see that the model performs best when cost=0.1. We have a minimal error of 0.1 with this parameter. The best model can be obtained as follows:

```
> bestmod =tune.out$best.model
```

Following is the summary of the best performing model:

```
> summary (tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.1

- Detailed performance results:
   cost error dispersion
1 1e-03  0.35  0.4116363
2 1e-02  0.35  0.4116363
3 1e-01  0.10  0.2108185
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
```

```
> 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
      gamma:  0.5

Number of Support Vectors:  16

 ( 8 8 )


Number of Classes:  2

Levels:
 -1 1
```

Now I will generate a test dataset and try to predict the class for each data point. The input to the svm model will be xtest. ytest is the expected output. I then converge the xtest and ytest to a single dataframe, with y as factor datatype for svm to work correctly.

```
> 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))
```

I used the predict() function to predict the class of the above generated data points. I have used the best model obtained above from the cross validation to make the prediction. Further, I used the table() function to display a confusion matrix of the results.

```
> ypred=predict (bestmod ,testdat )
> table(predict =ypred , truth= testdat$y )
       truth
predict -1 1
     -1  8 3
     1   0 9
```

We get 3 misclassifications out of 20 observations!

Let us check the classification rate with cost=0.01. We now have one additional misclassified point because by reducing the value of cost, we have increased our margin and more tolerant of some misclassifications!

```
> 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  8 4
     1   0 8
```

Support Vector Machine

We saw that linearly separable data was easy to work with. Now consider that we have nonlinear data as shown in the below Figure 8. It is not possible to segregate the data using a linear decision boundary.

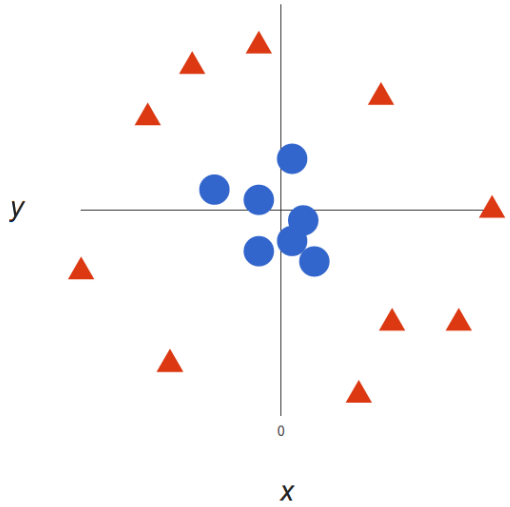


Figure 8: Nonlinear dataset [6]

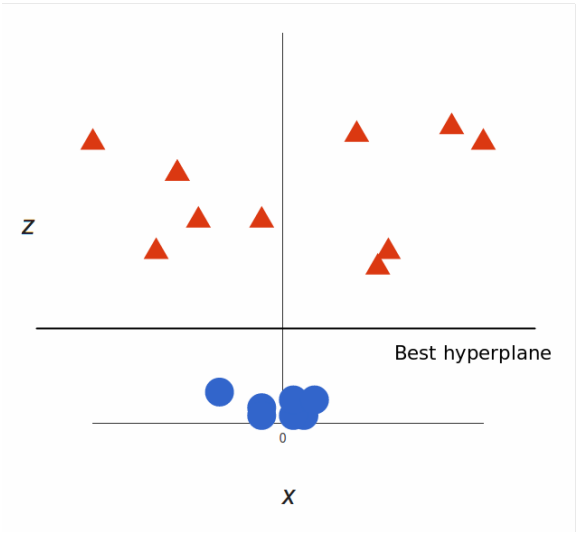


Figure 9: Adding the z- dimension to get two separated groups [7]

Therefore, we introduce a new z dimension. Refer Figure 9; we observe that the two classes of data are now separable. The best hyperplane lies parallel to the x-axis with z equal to some non-zero value. Going back to our normal view, we see that the best separating hyperplane looks like a circle, as shown in the adjacent figure 10.

However, mapping our data space to higher dimensions can get computationally complicated and time consuming. We have a better solution to this! SVM can work by computing the dot product between the vectors, which is greatly computationally efficient.

This dot product function is known as the *kernel* function. Thus, the support vector machine (SVM) is an extension of the support vector classifier that results from enlarging the feature space in a specific way, using kernels.[9]

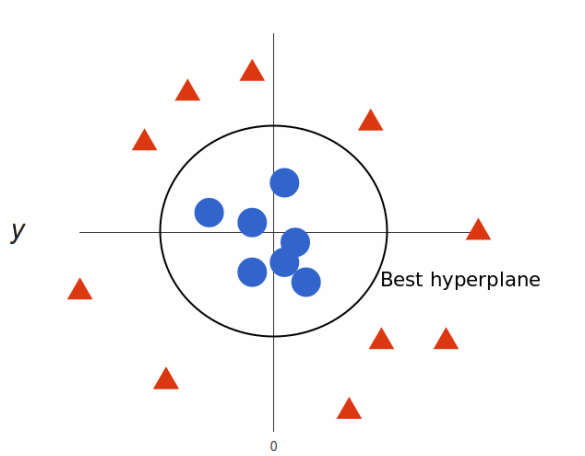


Figure 10: Original view with the separating hyperplane [8]

Walking through an example: Support Vector Machine**

In the e1071 library, we can pass the nonlinear data to the svm() function, but we use a different value for the kernel parameter. In this case, we would specify kernel="radial", which is a radial basis kernel. A radial basis kernel is one example of non-linear kernel. A radial kernel can be of the following form:

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

[Taken from the ISLR Text [10]]

Where x_i, $x_{i'}$ are two data samples with i rows, p is the number of parameters.
The γ parameter as seen in the above formula, can be passed to the svm() function with radial kernel as gamma=<value>

** The R code used in this section is taken from the ISLR Text, An Introduction to Statistical Learning, 359-362

[6,7,8] Source: https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

[9] ISLR Text, An Introduction to Statistical Learning, 9.3.2 The Support Vector Machine, 350

[10] ISLR Text, An Introduction to Statistical Learning, 9.3.2 The Support Vector Machine, 352

Let us now generate some data with a nonlinear separation boundary. I will first generate an x matrix with 200 rows and 2 columns. Then, I generated the y matrix with 200 rows and values 1, 2 in it.
All the x and y data will now be merged into a single dataframe with y as a factor datatype.

```
> 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))
> dat[1:4,]
       x.1      x.2 y
1 1.373546 2.409402 1
2 2.183643 3.688873 1
3 1.164371 3.586588 1
4 3.595281 1.669092 1
```

Let us verify if the data is non-linear by plotting it using the below command:

```
> plot(x, col=y)
```

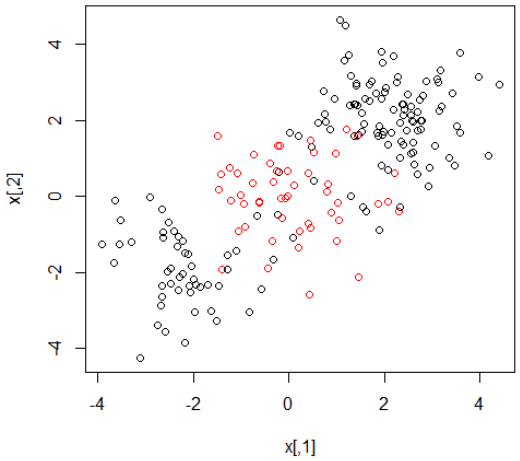


Figure 11: Nonlinear data space

I will now create a variable 'train' to randomly select rows from the dataset as training data and the remaining data rows would be the testing dataset.

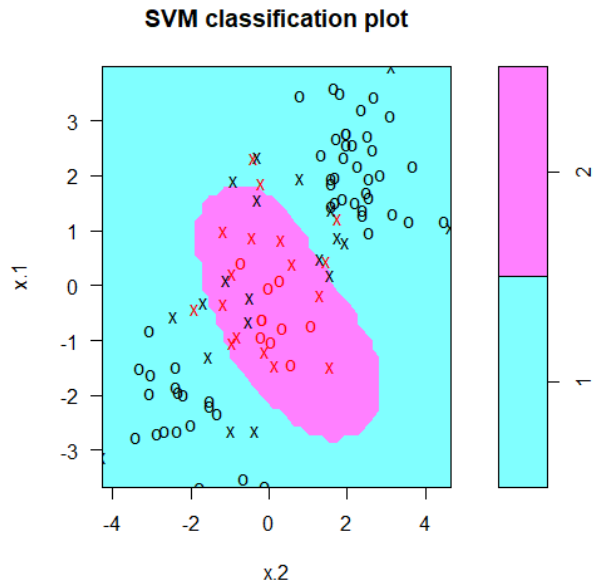
```
> train=sample (200 ,100)
```

We now pass the training data to the svm() function by specifying kernel as radial and $\gamma=1$.

```
> svmfit =svm(y~., data=dat[train,], kernel ="radial", gamma =1, cost=1)
```

Let us check the classification space obtained. From the Figure 12, we can see that the decision boundary is non-linear, as we expected it to be!

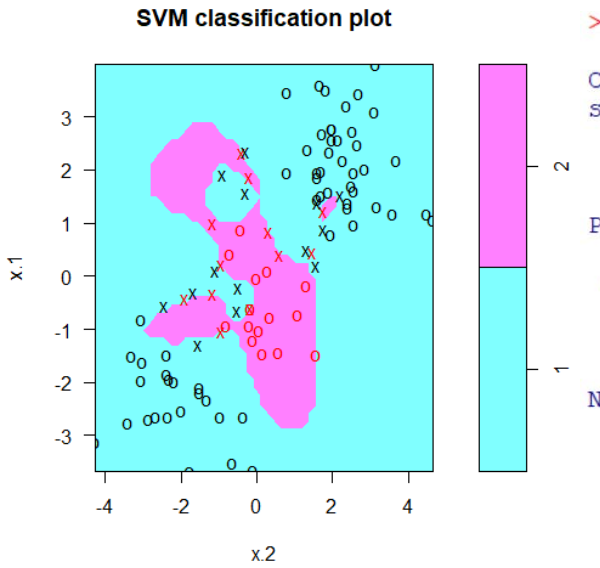
```
> plot(svmfit , dat[train ,])
```




```
> 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
      gamma:  1

Number of Support Vectors:  37

 ( 17 20 )


Number of Classes:  2

Levels:
 1 2
```

Figure 12: Classification space with radial kernel

Now suppose we increase the value of cost to 100000. Let us check the classification plot and how the number of support vectors change. We see that increasing cost can reduce the number of errors. It also results in an irregular decision boundary. However, reducing the number of errors might come at the cost of overfitting the data.




```
> summary (svmfit )

Call:
svm(formula = y ~ ., data = dat[train, ], kernel = "radial",
    gamma = 1, cost = 1e+05)


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

Number of Support Vectors:  26

 ( 12 14 )
```

Figure 13: Classification space with radial kernel and increased cost

Let us try to perform cross validation to find the best model and determine an optimal value of cost and gamma. We would be using tune() function as used with support vector classifiers.

We found out that the best values of cost and gamma are 1 and 2 respectively. The error rate is lowest for this model, which is equal to 0.12.

```
> 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 validation

- best parameters:
 cost gamma
    1     2

- best performance: 0.12

- Detailed performance results:
   cost gamma error dispersion
1  1e-01   0.5  0.27 0.11595018
2  1e+00   0.5  0.13 0.08232726
3  1e+01   0.5  0.15 0.07071068
```

Now let us predict the outcome on the test data. We will use predict() function to predict the outcome and then use table function to display the confusion matrix. The error rate comes out to be equal to 39%. The algorithm misclassifies most of the class '2' data.

```
> pred=predict(tune.out$best.model,newx=dat[-train,])
> table(true=dat[-train,"y"], pred=pred)
    pred
true  1  2
   1 56 21
   2 18  5
> (18+21)/(56+21+18+5)
[1] 0.39
```

Applying SVM methodology to Stock Range Data

I wish to determine if the stock is a high or low risk depending on the range data from the previous day and the day before. Lag1range represents the range data from previous day while Lag2range represents the range data from two days before.

```
> TPL=read.csv("TPL_Project.csv")
> TPL(head)
Error in TPL(head) : could not find function "TPL"
> head(TPL)
  TPLrange Lag1range Lag2range
1 1.100000  1.050001  0.199999
2 1.080000  1.100000  1.050001
3 0.299999  1.080000  1.100000
4 0.200001  0.299999  1.080000
5 0.450000  0.200001  0.299999
6 0.600001  0.450000  0.200001
```

I first loaded the stock range data to R workspace. . I found the median value of the range data and created another variable y where range data above the median value is classified as "HighRisk" while the data below median value is classified as "LowRisk".

```
> y.TPL= TPL[,1]
> median(y.TPL)
[1] 2.049999
> y.TPL[y.TPL>2.049999]="HighRisk"
> y.TPL[y.TPL<=2.049999]="LowRisk"
> y.TPL[1:4]
[1] "LowRisk" "LowRisk" "LowRisk" "LowRisk"
```

Before using svm() function, we should convert the y.TPL data as factor datatype. In addition, we should remove the TPLrange column, as we no longer need it for the prediction purpose.

```
> y.TPL=as.factor(y.TPL)
> TPL=TPL[,-1]
```

I then concatenated the Lag1range, Lag2range, y.TPL into a single dataframe.

```
> TPL=data.frame(TPL,y=y.TPL)
> head(TPL)
  Lag1range Lag2range       y
1  1.050001  0.199999 LowRisk
2  1.100000  1.050001 LowRisk
3  1.080000  1.100000 LowRisk
4  0.299999  1.080000 LowRisk
5  0.200001  0.299999 LowRisk
6  0.450000  0.200001 LowRisk
```

I set up the training data sample space by randomly choosing half of the data sample row numbers:

```
> train=sample(nrow(TPL),(nrow(TPL)/2))
> head(train)
[1] 2033  139  352 2881 2934 1360
```

I tried to plot the training data points. We see that this is a linear data space with two classes but there is certain region where both the red and black points mingle together. I created the color of each class by converting the categories "HighRisk" and "LowRisk" to numbers 2 and 3 respectively. HighRisk is represented by red color and LowRisk is represented as black color.

```
> col=TPL$y
> col=ifelse(col=="HighRisk",2,3)
> col=as.factor(col)
> summary(col)
   2    3
1503 1507
```

Then, I used the plot() command:

```
> plot(TPL[train,1],TPL[train,2],col=col[train])
```

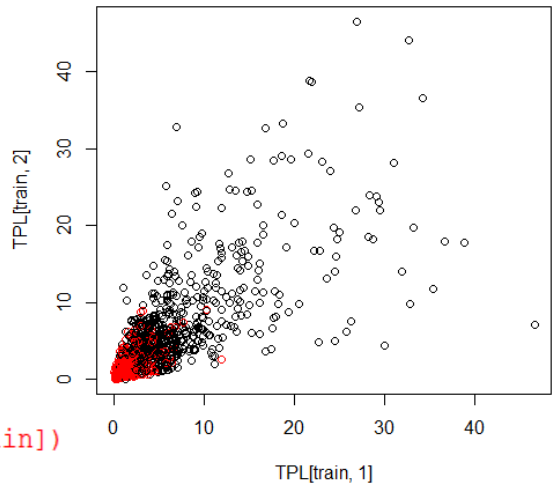


Figure 14: TPL Stock Range Training Dataset

I used cross validation to choose the best performing model and included a list of range for determining the best cost parameter out of those values.

```
> tune.out=tune(svm, y~., data=TPL[train,], kernel ="linear",
+ ranges =list(cost=c(0.001,0.01,0.1 ,1 ,10 ,100 ,1000))
+ )
```

The best model was obtained and the optimum value of cost was equal to 10. The number of support vectors was 591 as seen from the below summary.

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

Call:
best.tune(method = svm, train.x = y ~ ., data = TPL[train,
    ], ranges = list(cost = c(0.001, 0.01, 0.1, 1, 10, 100,
    1000)), kernel ="linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.5

Number of Support Vectors:  591

 ( 296 295 )


Number of Classes:  2

Levels:
 HighRisk LowRisk
```

I moved on to plot the classification space of this dataset. The LowRisk region is a small region in pink at the bottom left corner of the graph.

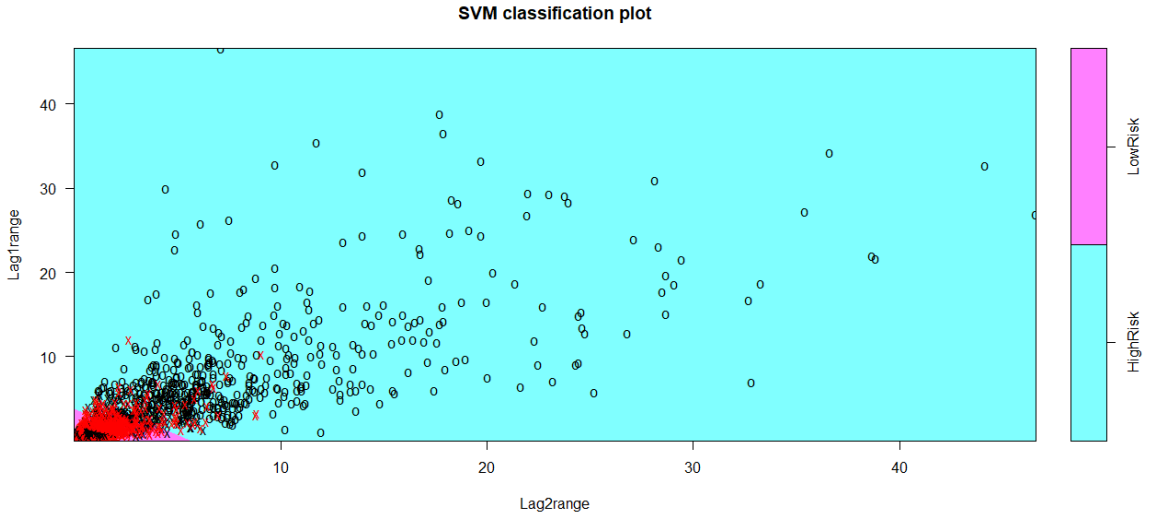


Figure 15: Classification Space and Data Points for TPL Stock Range Data

I then used the test dataset to predict the risk using the above model. I displayed the confusion matrix using the table function. Using SVM, I got a correct classification rate = 84.05%.

```
> ypred=predict(bestmod,TPL[-train,])
> table(predict=ypred,truth=TPL[-train,'y'])
          truth
predict    HighRisk LowRisk
  HighRisk      568      55
  LowRisk       185     697
```

Correct classification rate:

```
> (568+697)/(568+55+185+697)
[1] 0.8405316
```

Applying SVM methodology to Housing Affordability Data System (HADS) Dataset

The Housing Affordability Data System (HADS) dataset [11] is based on American Housing Survey (AHS) national files over past few years. This dataset includes a number of parameters describing housing-unit level information. It aims to measure the affordability of housing units relative to various features such as area median income, poverty level income and Fair Market Rent. For this project, I am going to classify Fair Market Rent as 'High' or 'Low' based on certain features from the HADS dataset. I am going to use the SVM methodology as described in previous sections for the classification problem. A description of features to be used as an input to the svm() function is as follows[12] :

Variable	Description
AGE1	Age of head of household
REGION	Census Region
BEDRMS	# of bedrooms in unit
BUILT	Year unit was built
VALUE	Current market value of unit
ROOMS	# of rooms in unit
ZINC2	Household Income
ZSMHC	Monthly housing costs
UTILITY	Monthly utility cost
COSTMED	Housing cost at Median interest
TOTSAL	Total Wage Income
BURDEN	Housing cost as a fraction of income

Response	Description
FMR	Fair market rent (average)

I began by importing the training dataset to the R workspace:

```
> FMR_Train=read.csv("FMR_Train.csv")
> FMR_Train[1:4,]
  AGE1 REGION BEDRMS BUILT  VALUE ROOMS  ZINC2 ZSMHC  UTILITY   COSTMED
1   82    '1'      2  2006  40000     6  18021   533 169.0000  615.1567
2   50    '3'      4  1980 130000     6 122961   487 245.3333 1058.9885
3   53    '3'      4  1985 150000     7  27974  1405 159.0000 1068.0252
4   67    '3'      3  1985 200000     6  32220   279 179.0000 1411.7002
  TOTSAL     BURDEN  FMR
1      0 0.35491926  956
2 123000 0.04752726 1100
3  28000 0.60270251 1100
4      0 0.10391061  949
```

[11] Source: https://catalog.data.gov/dataset/housing-affordability-data-system-hads
[12] Source: https://www.huduser.gov/portal/datasets/hads/HADS_doc.pdf

I converted the actual FMR values to classes as 'High' or 'Low' and stored it in y-variable. If FMR>2000, then y="High" otherwise y='Low'.
To accomplish this, I created a vector y with same data structure as FMR variable of the dataframe. I converted the FMR values to categories - 'High' and 'Low', using adjacent commands. Finally, I converted 'y' to factor datatype because svm() function would need it in that format.

```
> y=FMR_Train$FMR
> y[FMR_Train$FMR>2000]='High'
> y[FMR_Train$FMR<=2000]='Low'
> y=as.factor(y)
> summary(y)
High  Low
 309  326
```

I added 'y' to the FMR_Train dataframe and removed FMR column from the dataframe because we do not need to include it for the SVM computation.

```
> FMR_Train=data.frame(FMR_Train,y)
> FMR_Train=FMR_Train[,-13]
> FMR_Train[1:4,]
  AGE1 REGION BEDRMS BUILT  VALUE ROOMS  ZINC2 ZSMHC  UTILITY   COSTMED
1   82    '1'      2  2006  40000     6  18021   533 169.0000  615.1567
2   50    '3'      4  1980 130000     6 122961   487 245.3333 1058.9885
3   53    '3'      4  1985 150000     7  27974  1405 159.0000 1068.0252
4   67    '3'      3  1985 200000     6  32220   279 179.0000 1411.7002
  TOTSAL     BURDEN   y
1      0 0.35491926 Low
2 123000 0.04752726 Low
3  28000 0.60270251 Low
4      0 0.10391061 Low
```

I moved on to run the SVM function on this dataframe. I used radial kernel because the above data would not have a linear decision boundary owing to 12 features. I performed cross validation to determine the best model and used a range of values for cost and gamma variables.

```
> FMR_tune.out=tune(svm, y~., data=FMR_Train, kernel ="radial",
+ ranges =list(cost=c(0.1,1,10,100,1000),
+ gamma=c(0.001,0.01,0.1,1)))
```

I tried to run a summary of the above svm() function with cross validation. We see that the best performing model uses cost=1000 and gamma=0.01.

```
> summary(FMR_tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
 1000  0.01

- best performance: 0.08665675

- Detailed performance results:
    cost gamma      error dispersion
1  1e-01 0.001 0.48199405 0.06776319
2  1e+00 0.001 0.14335317 0.04246169
3  1e+01 0.001 0.10711806 0.04217817
4  1e+02 0.001 0.10868056 0.05306211
5  1e+03 0.001 0.11966766 0.05166862
6  1e-01 0.010 0.14332837 0.04229888
7  1e+00 0.010 0.11024306 0.04484697
8  1e+01 0.010 0.10399306 0.04630767
9  1e+02 0.010 0.09454365 0.04301164
10 1e+03 0.010 0.08665675 0.04240712
```

The best model can be accessed as shown below. I tried to re-verify the values of cost and gamma for the best model. We can also notice the number of support vectors from the below summary:

```
> FMR_bestmod=FMR_tune.out$best.model
> summary(FMR_bestmod)

Call:
best.tune(method = svm, train.x = y ~ ., data = FMR_Train,
    ranges = list(cost = c(0.1, 1, 10, 100, 1000), gamma = c(0.001,
        0.01, 0.1, 1)), kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1000
      gamma:  0.01

Number of Support Vectors:  133

 ( 66 67 )


Number of Classes:  2

Levels:
 High Low
```

I then tried to plot the SVM classification space. However, I got the following error message.

```
> plot(FMR_bestmod,FMR_Train)
Error in plot.svm(FMR_bestmod, FMR_Train) : missing formula.
```

I googled the error message and checked for the answer in a blog [13] . I found out that my data has multiple columns (variables). The above function works fine if there are only three columns in a dataframe; of which one column is the response. I have to be specific against which two variables could I plot the classification space. I chose to use ROOMS and UTILITY for now. I modified my plot command as follows:

```
> plot(FMR_bestmod,FMR_Train,UTILITY~ROOMS)
```

I got the following graph. However, this seems to be incorrect as it shows only the 'High' classification space.

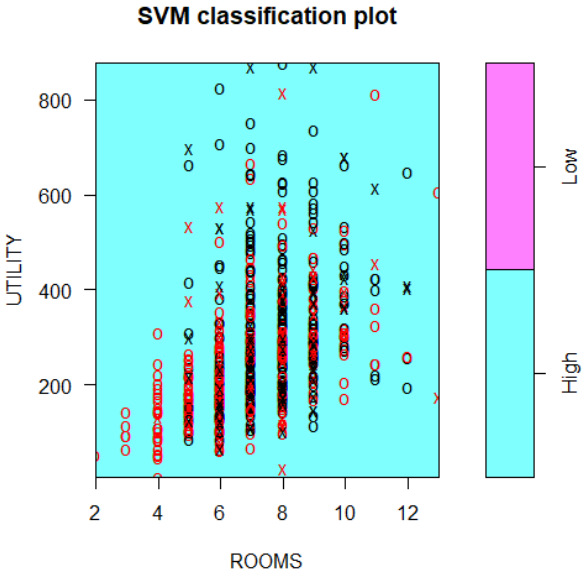


Figure 16: An incorrect SVM classification Space for HADS Dataset

I tried to research on this issue. I checked the r documentation on the plot.svm function[14] . I found out an argument 'slice', which needs to be used if there are, more than two variables in the dataset. 'Slice' argument is used to pass a list of named values for dimensions we want to hold constant. In our case, UTILITY and ROOMS are used for the plot. We need to specify the values of other dimensions, which are

[13] Source: https://stackoverflow.com/questions/25716998/error-in-plot-formula-missing

[14] https://www.rdocumentation.org/packages/e1071/versions/1.7-0/topics/plot.svm

to be held constant. I created a variable 'sl' which contains a list of values to be held constant and to be passed to the 'slice' argument in the plot() function. I got the classification space as shown in Figure 17 below

sl=list(AGE1=58,BEDRMS=4,BUILT=1985,VALUE=550000,ZINC2=314209,ZSMHC=2306,COSTMED=3468.426,TOTSAL=0,BURDEN=0.088069)

```
> sl=list(AGE1=58,BEDRMS=4,BUILT=1985,VALUE=550000,ZINC2=314209
> plot(FMR_bestmod,FMR_Train,UTILITY~ROOMS,slice=sl)
```

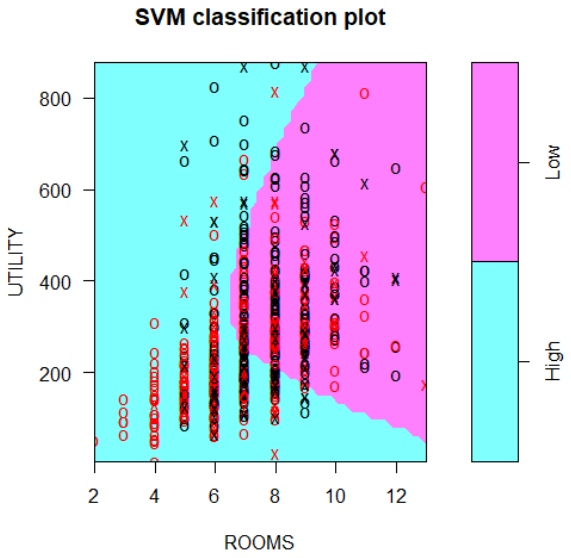


Figure 17: Correct SVM Classification Space for HADS dataset

I tried to apply the FMR_bestmod to our test dataset. Before that, I had to import the test dataset to R workspace.

```
> FMR_Test=read.csv("FMR_Test.csv")
> FMR_Test[1:4,]
  AGE1 REGION BEDRMS BUILT  VALUE ROOMS  ZINC2 ZSMHC  UTILITY  COSTMED
1   53    '1'      4  1960 680000     8 167274  1253 327.6667 4378.581
2   57    '1'      4  1970 680000    11 138610  5512 220.6667 4421.581
3   54    '1'      4  1980 200000     6  75187  1339 138.5000 1383.867
4   35    '1'      3  1950 380000     6  96974  1816 290.6667 2593.530
  TOTSAL     BURDEN  FMR
1 160800 0.08988845 2280
2 109000 0.47719501 2280
3  50000 0.21370716 2280
4  97000 0.22472003 2058
```

I converted the FMR values in the above table to classes – 'High' or 'Low' and stored it in y1 variable; same as we did it for the training dataset.

```
> y1=FMR_Test$FMR
> y1[FMR_Test$FMR>2000]='High'
> y1[FMR_Test$FMR<=2000]='Low'
> y1=as.factor(y1)
> summary(y1)
High  Low
 100  100
```

Integrating 'y1' into the FMR_Test dataframe and removing the FMR column from the dataframe because we would not need FMR to be included in the SVM computation.

```
> FMR_Test=data.frame(FMR_Test,y1)
```

```
> FMR_Test=FMR_Test[,-13]
> FMR_Test[1:4,]
  AGE1 REGION BEDRMS BUILT  VALUE ROOMS  ZINC2 ZSMHC  UTILITY  COSTMED
1   53    '1'      4  1960 680000     8 167274  1253 327.6667 4378.581
2   57    '1'      4  1970 680000    11 138610  5512 220.6667 4421.581
3   54    '1'      4  1980 200000     6  75187  1339 138.5000 1383.867
4   35    '1'      3  1950 380000     6  96974  1816 290.6667 2593.530
  TOTSAL     BURDEN   y1
1 160800 0.08988845 High
2 109000 0.47719501 High
3  50000 0.21370716 High
4  97000 0.22472003 High
```

I could now apply the SVM model to the above test dataset. I used predict() function for the prediction.

```
> pred=predict(FMR_bestmod,FMR_Test)
```

Using table() function, I displayed the confusion matrix as follows:

```
> table(true=FMR_Test[,'y1'],pred=pred)
      pred
true   High Low
  High   87  13
  Low     5  95
```

The correct classification rate was found out to be equal to 91% with SVM.

```
> (87+95)/(87+13+5+95)
[1] 0.91
```

Comparing performance of SVM with other methodologies

1. Stock Range Data

I. Using the k-nn methodology

I went further and checked how the k-nn algorithm performs on the stock range dataset. Here, is how our stock range dataframe looked like:

```
> head(TPL)
  Lag1range Lag2range       y
1  1.050001  0.199999 LowRisk
2  1.100000  1.050001 LowRisk
3  1.080000  1.100000 LowRisk
4  0.299999  1.080000 LowRisk
5  0.200001  0.299999 LowRisk
6  0.450000  0.200001 LowRisk
```

I randomly selected a few observations to be used for the training dataset. The rest of the observations would be used as the test dataset:

```
> train=sample(3010,1505)
```

I set the train and test dataset as follows:

```
> TrainX.TPL=TPL[train,1:2]
> TrainY.TPL=TPL[train,3]
> TestX.TPL=TPL[-train,1:2]
> TestY.TPL=TPL[-train,3]
```

I used the k-nn function and displayed the confusion matrix using the table() function:

```
> knn.pred_TPL=knn(TrainX.TPL,TestX.TPL,TrainY.TPL,25)
```

```
                TestY.TPL
knn.pred_TPL HighRisk LowRisk
    HighRisk      614      108
    LowRisk       131      652
```

I found out the correct classification rate using k-nn to be 84.12%.

```
> (614+652)/(614+108+131+652)
[1] 0.841196
```

II. Using the Naïve Bayes methodology

We already had the train and test dataset available from the k-nn method. Therefore, I passed the TrainX and TrainY data to the Naïve Bayes classifier.

```
> classifier_TPL=naiveBayes(TrainX.TPL,TrainY.TPL)
```

I predicted the classes of response variable using the predict() function and used the table() function to display the confusion matrix.

```
> table(predict(classifier_TPL,TestX.TPL),TestY.TPL)
           TestY.TPL
            HighRisk LowRisk
  HighRisk       514       35
  LowRisk        231      725
```

The correct classification rate using this methodology was 82.33%. This is very close to the rate with k-nn method.

```
> (514+725)/(514+35+231+725)
[1] 0.8232558
```

III. Using the logistic regression methodology

I ran the glm() function where I used the argument family='binomial' to specify that I will use logistic regression. I passed all the variables except y as the predictor variables and y would be the response variable.

```
> glm.fit_Stock=glm(y~.,data=TPL,family=binomial)
```

I then used the predict() function to output the estimates of probabilities of the risk to be 'HighRisk' or 'LowRisk'. I created a vector glm.forecast_TPL to convert the above probabilities to response forecasts. I copied the data structure of y i.e. the response variable for glm.forecast_TPL. For probabilities greater than 0.5, I replaced the value as 'HighRisk' in the forecast vector, whereas for probabilities less than equal to 0.5, I replaced the value 'LowRisk'.

```
> glm.probs_TPL=predict(glm.fit,type='response')
> glm.probs_TPL[1:10]
           1            2            3            4            5
4.604394e-03 1.169447e-03 1.907138e-01 5.890024e-03 1.369609e-02
           6            7            8            9           10
2.418912e-11 3.675343e-04 6.045707e-03 6.891496e-03 1.401448e-03
> glm.forecast_TPL = TPL$y
> glm.forecast_TPL[glm.probs_TPL>0.5]="HighRisk"
> glm.forecast_TPL[glm.probs_TPL<=0.5]="LowRisk"
> summary(glm.forecast_TPL)
HighRisk  LowRisk
    1440     1570
```

Let us check the confusion matrix.

```
> table(glm.forecast_TPL,TPL$y)

glm.forecast_TPL HighRisk LowRisk
        HighRisk      743     697
        LowRisk       760     810
```

The correct classification rate using logistic regression was 51.59%, which is extremely low, and the least when compared to other methods.

```
> (743+810)/(743+697+760+810)
[1] 0.5159468
```

I have summarized the correct classification rates using different methodologies for stock range data as follows:

Method	Correct Classification Rate
k-nn	84.12%
Naïve Bayes	82.33%
Logistic Regression	51.59%
SVM	84.05%

Thus, k-nn, SVM and Naïve Bayes perform very close to each other on the stock range data with a good correct classification rate. However, logistic regression does not seem to predict the output very well!

2. HADS Data

I. Using the k-nn methodology

I will now check how the k-nn algorithm performs on the HADS dataset.
I opened the classification package using the below command:

```
> library(class)
```

I set the train and test dataset as follows, where all the variables except the response variable was included in the TrainX and TestX, while the response variable was stored in the TrainY and TestY variable:

```
> TrainX=FMR_Train[,1:12]
> TrainY=FMR_Train[,13]
> TestX=FMR_Test[,1:12]
> TestY=FMR_Test[,13]
```

I ran the knn() function, however I got the following error and warning messages:

```
> knn.pred = knn(TrainX,TestX,TrainY,25)
Error in knn(TrainX, TestX, TrainY, 25) :
  NA/NaN/Inf in foreign function call (arg 6)
In addition: Warning messages:
1: In knn(TrainX, TestX, TrainY, 25) : NAs introduced by coercion
2: In knn(TrainX, TestX, TrainY, 25) : NAs introduced by coercion
```

I searched for the error messages and found in a blog that – "Check if there are any factor variables. If any convert it into numeric." [15] .The REGION variable was of factor datatype and I converted it to numeric datatype before using the knn() function. I could then run the knn() function successfully.

```
> TrainX$REGION=as.numeric(TrainX$REGION)
> TestX$REGION=as.numeric(TestX$REGION)
> knn.pred = knn(TrainX,TestX,TrainY,25)
```

I displayed the confusion matrix using the table function. I found out the correct classification rate = 73%, which extremely less than that with SVM (91%).

```
> table(knn.pred,TestY)
        TestY
knn.pred High Low
    High   65  19
    Low    35  81
> knn_corrClassificationRate = (65+81)/(65+19+35+81)
> knn_corrClassificationRate
[1] 0.73
```

II. Using the Naïve Bayes methodology

I went ahead to check how Naïve Bayes worked on the HADS dataset. We already had the train and test dataset available from the k-nn method. I passed the TrainX and TrainY data to the Naïve Bayes classifier.

```
> classifier=naiveBayes(TrainX,TrainY)
```

I predicted the classes of response variable using the predict() function and used the table() function to display the confusion matrix.

```
> classifier=naiveBayes(TrainX,TrainY)
> table(predict(classifier,TestX),TestY)
      TestY
       High Low
  High   76  10
  Low    24  90
```

The correct classification rate using this methodology was 83%. This is a significant improvement than the k-nn methodology, but SVM still performed the best amongst the three methods we had employed until now.

[15] https://discuss.analyticsvidhya.com/t/how-to-resolve-error-na-nan-inf-in-foreign-function-call-arg-6-in-knn/7280

```
> nb_corrClassificationRate = (76+90)/(76+10+24+90)
> nb_corrClassificationRate
[1] 0.83
```

III. Using Logistic Regression methodology

I ran the glm() function where I used the argument family='binomial' to specify that I will use logistic regression. I passed all the variables except y1 as the predictor variables and y1 is the response variable.

```
> glm.fit=glm(y1~.,data=FMR_Test,family=binomial)
```

I used the predict() function to output the estimates of probabilities of the FMR to be 'High' or 'Low'. I created a vector glm.forecast to convert the above probabilities to response forecasts. I copied the data structure of y1 i.e. the response variable for glm.forecast. For probabilities greater than 0.5, I replaced the value as 'High' in the forecast vector, whereas for probabilities less than equal to 0.5, I replaced the value 'Low'.

```
> glm.probs=predict(glm.fit,type='response')
> glm.probs[1:4]
          1           2           3           4
0.004604394 0.001169447 0.190713788 0.005890024

> glm.forecast=FMR_Test$y1
> glm.forecast[glm.probs<=0.5]="High"
> glm.forecast[glm.probs>0.5]="Low"
> summary(glm.forecast)
High  Low
 104   96
```

I then displayed the confusion matrix as follows.

```
> table(glm.forecast,FMR_Test$y1)

glm.forecast High Low
        High   96   8
        Low     4  92
```

Thus, the correct forecast rate with logistic regression method is 94%. It is interesting to note that this methodology performed better than the SVM!

```
> lr_corrClassificationRate = (96+92)/(96+8+4+92)
> lr_corrClassificationRate
[1] 0.94
```

I have summarized the results for all the four methodologies used as follows:

Method	Correct Classification Rate
k-nn	73%
Naïve Bayes	83%
Logistic Regression	94%
SVM	91%

To summarize for HADS dataset, SVM performs well, however logistic regression seems to make the best prediction with the highest correct classification rate of 94%. k-nn performs the worst and only makes 73% correct predictions.

Conclusion

Thus, SVM algorithm worked well for both, the stock range data to predict the risk level of stock, as well as for the HADS dataset to classify the Fair Market Rent according to various housing-unit level based parameters. To sum up the project: A support vector machine helped us classify linearly separable data. The decision boundary was a straight line in this case. For non-linearly separable data, we used the kernel trick to transform the data and based on the transformations, SVM could find an optimal separating hyperplane between the data points. In this case, the best hyperplane was not a straight line but a non-linear decision boundary. The kernel trick saves a lot of computational complexity and time. The tuning parameters such as cost and gamma helped to control the number and severity of errors to the margin we could tolerate.