ISTANBUL TECHNICAL UNIVERSITY - SCIENCE AND LETTERS FACULTY

MATHEMATICS ENGINEERING PROGRAM

GRADUATION PROJECT

EMRE KÜÇÜK - 090140318

SUBMISSION DATE: 18.05.2019

ADVISOR: Assoc. Prof. Dr. Atabey Kaygun

MAY 2019

Index

A Study on Statlog Australian Credit Approval Dataset

May 18, 2019

1 Introduction

Many institutions use different statistical and machine learning algorithms on a prepared dataset in order to obtain efficient and useful results. In this thesis, we are going to apply some statistical and machine learning algorithms to a given dataset and criticize their results.

Dataset contains confidential information about different people who acquired credit from a bank, and whether they are paid their credit back or not. Dataset can be accessed via this link, and we will try to build a solid model for making a classification for credit approvals.

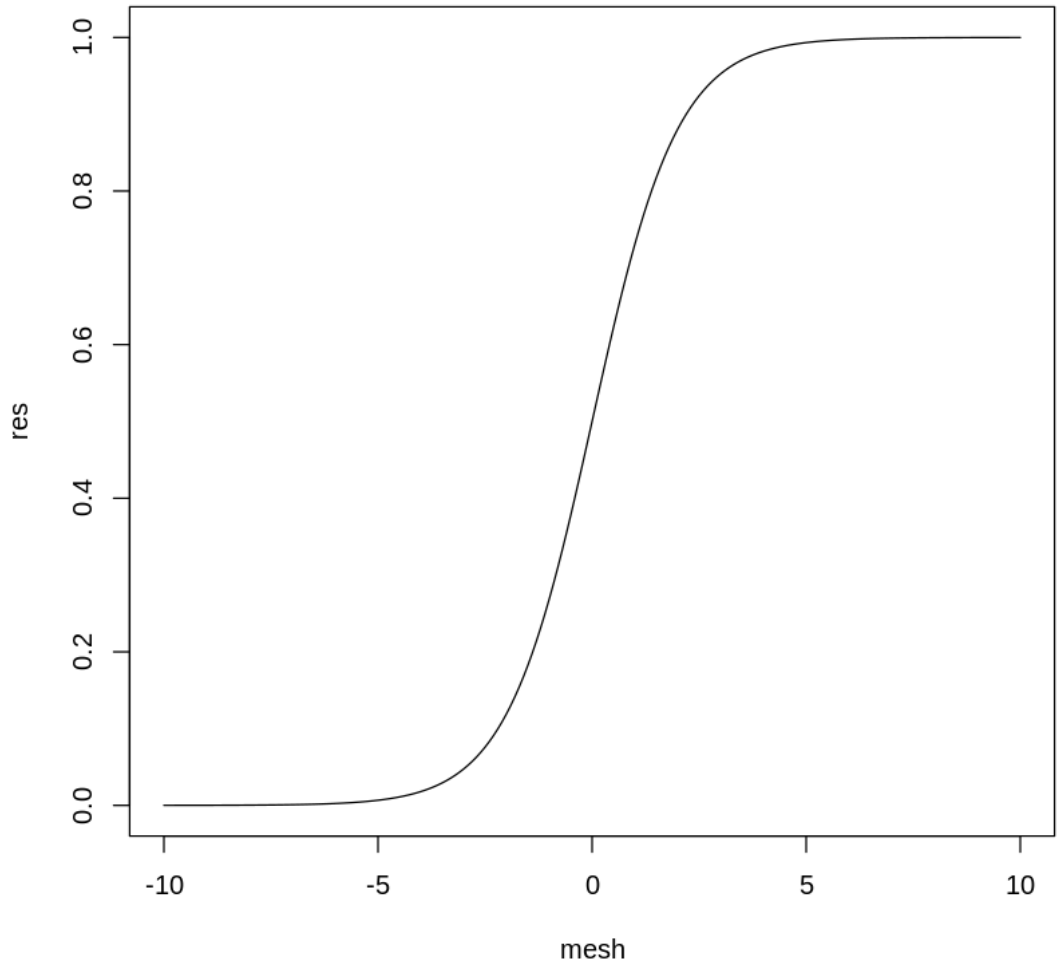
"**Classification problem**" considers the about giving predictions on a defined discrete result. Each instance belongs to one element of result set, and we will try to arrange dataset properly for making predictions by scaling them, applying statistical algortihms etc. Then, we will test algorithms such as logistic regression and support vector machine to find out if they are accurate or not; then we will make evaluations about outputs.

1.1 Logistic Regression

Logistic regression is a classification method for finding an answer to "classification" problem.

For logistic regression problems, the response variable has to be binary: 0 or 1, or YES or NO. The usual way of converting numerical values to binary classes is to interpret it as probabilities which takes values between 0 and 1. In converting probabilities we use cetain functions. The sigmoid function is most commonly used.

```
In [1]: sigmoid <- function(x) {
            1/(1+exp(-x))
        }
        mesh <- seq(-10, 10, 0.1)
        res <- sigmoid(mesh)
        plot(mesh, res, type="l")
```



Passing parameters to a function returns a probability value. Since sigmoid function is bounded between 0 and 1, it is wise to choose for calculating probability. Usually selecting a $P(Y = y|X = x_{(1)}, \ldots, x_{(n)}) \geq 0.5$ as a label and other values as another label. This concludes the binary classification problem.

Theoretical explanations below are taken from CMU 36-402, Undergraduate Advanced Data Analysis, Lecture 13, Logistic Regression.

Solving a classification problem is hard with linear regression. One way to solve might be selecting a linear function for making classification, but it is not useful due to "diminishing returns" problem and being unbounded from function's nature. On the other hand, logaritmic transformation might be useful, but, logaritmic functions are bounded from one direction.

Lastly, a useful concept might be making nuance adjustments as called "**logistic transformation**", which can be made as $\log(\frac{p}{1-p})$. This adjustment makes possible to have reasonable results.

This method is called as logistic regression. We can accomplish this as solving the equation below for $P(x)$.

$$\log(\frac{P(x)}{1-P(x)}) = \beta_0 + x.\beta$$

Solving for $P(x)$, we conclude:

$$P(x;b;w) = \frac{e^{\beta_0 + x.\beta}}{1 + e^{\beta_0 + x.\beta}} = \frac{1}{1 + e^{-(\beta_0 + x.\beta)}}$$

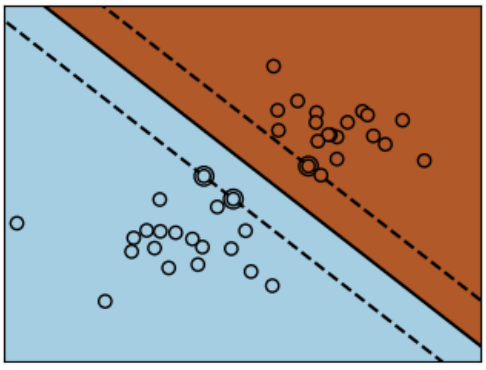
For reducing classification error rate, it should be predicted $Y = 1$ when $P \geq 0.5$ and $Y = 0$ when $P < 0.5$. This means that whenever $\beta_0 + x.\beta$ is greater than zero Y should be classified as 1 and 0 on other conditions. As a conclusion, logistic regression returns a "**linear classifier**".

$$\beta_0 + x.\beta = 0$$

can be considered as the "**decision boundary**", which is a point or line if x is one dimensional or two, and so on.

1.2 Support Vector Machine

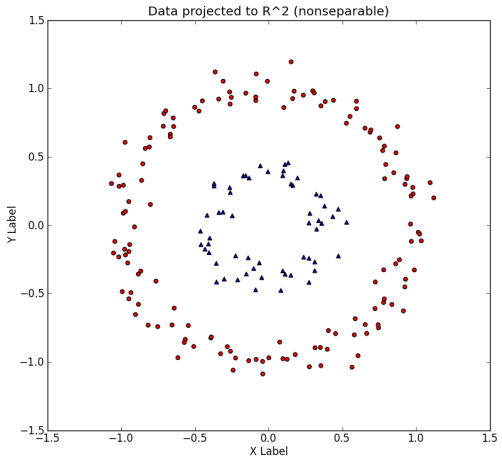
Support Vector Machine (SVM) is also a classification algorithm. Name comes from a term called as "support vector" which defines a classifier from using it. Support vector machine uses hyperplanes for making classifications. But on a space, there can be infinitely many hyperplanes, and SVM tries to calculate most efficient and accurate hyper plane to use as a classifier. On a given space, for example $\mathbb{R}^n$, algorithm tries to find the hardest points to classify. After that, by using these points, generates support vectors in order to create a classifier, then by using these support vectors SVM creates hyperplanes and tries to maximizing margin between these hyperplanes. In the middle of those hyperplanes, algorithm obtains most efficient hyperplane. Easiest way to apply or show this is on using linear separators:

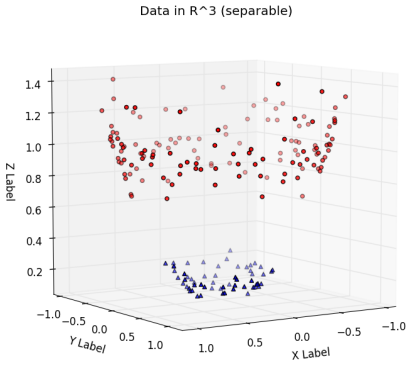


Taken from scikit-learn.org

Dotted lines are called support vectors, black line is hyperplane. There can be infinitely many hyperplane, but when we are choosing we need to do this by maximizing the margin between our support vectors so that we can use the best classifier.

SVM also has a kernel "**trick**". One can change kernel to be used in order to making classifications possible. For example:





Taken from towardsdatascience.com

A non-separable data can be made separable by usage of different kernels.

But, how does SVM makes as best prediction as possible? Let us dive in to some theory. Theoretical explanations below taken from Computer Image Processing and Analysis (E161) and R. Berwick, An Idiot's guide to Support vector machines (SVMs).

From linear algebra, we simply know that a hyperplane can be written as

$$w^T * x + \beta_0 = 0$$

Any vector $x = x_1, \ldots, x_{(n)}$ in a given dataset belongs a certain class which defined as $y = y_1, \ldots, y_{(n)}$. Thus, we need to define a condition for making classifications. For talking binary classification on a linearly separable dataset on $\mathbb{R}^2$, we can say that there are two classes, and each data point must belong either one. Define data points as (x_i, y_i) where x_i is the feature, y_i is the class. and two hyper planes such as

$$H_1 : w.x_i + b = +1$$

$$H_2 : w.x_i + b = -1,$$

assuming $y_i = +1$ for $w.x_i + b > +1$ and $y_i = -1$ for $w.x_i + b < -1$. These support vectors are created from critical points, which means changing any of their position would affect the position of the vectors.

SVM is pursuing the idea of maximizing the margin between these support vectors, also called as "**street**".

Maximizing the margin of street is a problem that concerns optimization, which called as "Constrained Optimization Problem"; to minimize $||w||$.

Kernel Trick If data is not linearly separable, "**Kernel Mapping**" is also a solution. Simply, it is possible to map samples to higher dimensions:

$$x \to \phi(x)$$

where $\phi(x)$ is linearly separable. Decision number on new space becomes as:

$$f(x) = \phi(x)^T w + b = \sum_{j=1}^{m} \alpha_j y_j \phi(x_j)$$

where

$$w = \sum_{j=1}^{m} \alpha_j y_j \phi(x_j)$$

Definition: Kernel function is a function that taking two vector and returns an inner product.

Let x_i, x_j be two vectors and $\phi(x_i)$ and $\phi(x_j)$ be their images, a kernel function is represented as:

$$K(x_1, x_2) = \phi(x_1)^T \phi(x_2)$$

Since kernel function returns a result of inner product, dimension of space does not matter.

Kernel space's "learning algorithm" can be derived by changing all inner products of the original space to new kernel:

$$f(x) = \phi(x)^T w + b = \sum_{j=1}^{m} \alpha_j y_j K(x, x_j) + b$$

And parameter b can be found from any support vector x.

2 Experiment

First we are importing necessary libraries and dataset. Then we are assigning dataset to a variable "myData".

```
In [3]: library(rpart)
        library(rpart.plot)
        library(e1071)
        library(caret)
        library(ROSE)
        library(foreign)
        library(nnet)
```

```
In [4]: myData <- read.csv("http://archive.ics.uci.edu/ml/machine-learning-databases/
statlog/australian/
```

As explained on the web page of the dataset, because of confidentiality, column names, instance names and other charactheristics of features are obscured except type of feature (either continuous or categorical) and dependend variable V15 should be predicted. We are going to investigate other characteristics of features. V1, V4, V5, V6, V8, V9, V11, V12 are categorical features, other features are continuous features. Categorical features are also encoded with numbers.

```
In [5]: head(myData)
```

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15
1	22.08	11.460	2	4	4	1.585	0	0	0	1	2	100	1213	0
0	22.67	7.000	2	8	4	0.165	0	0	0	0	2	160	1	0
0	29.58	1.750	1	4	4	1.250	0	0	0	1	2	280	1	0
0	21.67	11.500	1	5	3	0.000	1	1	11	1	2	0	1	1
1	20.17	8.170	2	6	4	1.960	1	1	14	0	2	60	159	1
0	15.83	0.585	2	8	8	1.500	1	1	2	0	2	100	1	1

2.1 Data Normalizaiton, Feature Selection and Sampling

Adjusting dataset for categorical variables. Here R's factor function is used in order to tell R kernel that given certain columns should be treated as categorical features. The reason we are doing this is since categorical features are encoded as numbers, by default R will try to behave those features continuously and make calculations by that assumption. R would not assume those features if categorical features would encoded as strings.

```
In [6]: myData$V1 <- factor(myData$V1)
        myData$V4 <- factor(myData$V4)
        myData$V5 <- factor(myData$V5)
        myData$V6 <- factor(myData$V6)
        myData$V8 <- factor(myData$V8)
        myData$V9 <- factor(myData$V9)
        myData$V11 <- factor(myData$V11)
        myData$V12 <- factor(myData$V12)
        myData$V15 <- factor(myData$V15)
```

For an easier implementation, we created a vector called numerical_features and categorical_features. It contains names of numerical features and categorical features by only themselves.

```
In [7]: numerical_features <- c("V2","V3","V7","V10","V13","V14")
        categorical_features <- c("V1","V4","V5","V6","V8","V9","V11","V12")
```

After factorization, we are going to split the dataset to train and test parts. Here we are taking random row number with using nrow. Then creating a train and test samples from dataset with %75 train to %25 test ratio.

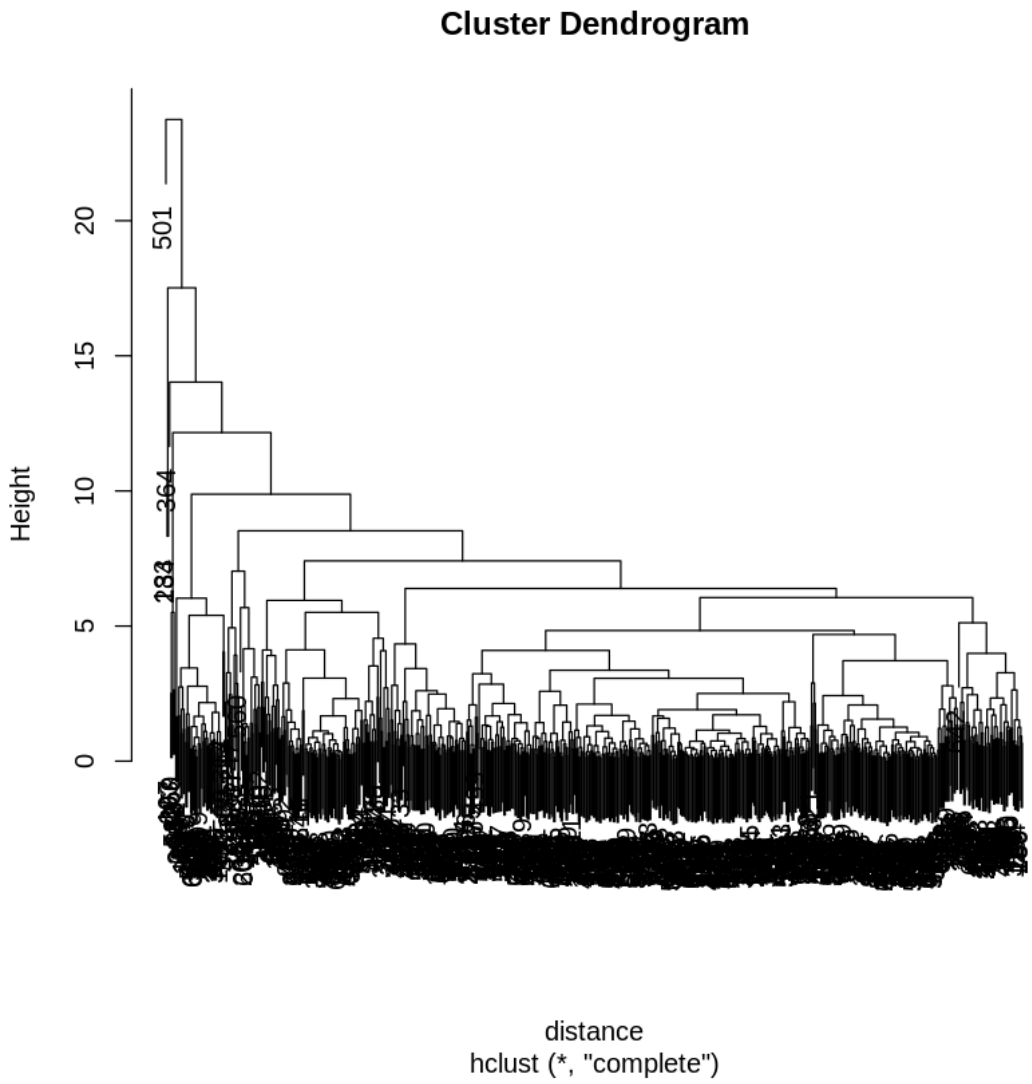
```
In [8]: n <- nrow(myData)
        test <- sample(1:n, n*0.25)
        train <- -test
```

In order to make accurate predictions, we must normalize the dataset if range of magnitude differs. Here we are scaling our data in order to make more accurate predictions. If we do not apply this operation, columns with bigger numbers might have an effect so that columns with smaller numerical values might not be considered for algorithms. This might create a problem if a column with a smaller number have vast amount of effect on predictions.

```
In [9]: myData[,numerical_features] <- scale(myData[,numerical_features])
```

Other than normalization problem, some of categorical features might create a problem. For example, 5th column is categorical but there are 14 different categories. It is better to check if 5th column is important, or able to prun it with hierarchical clustering; because there might some problems occur whilst splitting data into train and test. Let us take a look at dendogram.

```
In [128]: distance <- dist(myData[,numerical_features])
          cluster <- hclust(distance)
          plot(cluster, label = myData$v15)
```



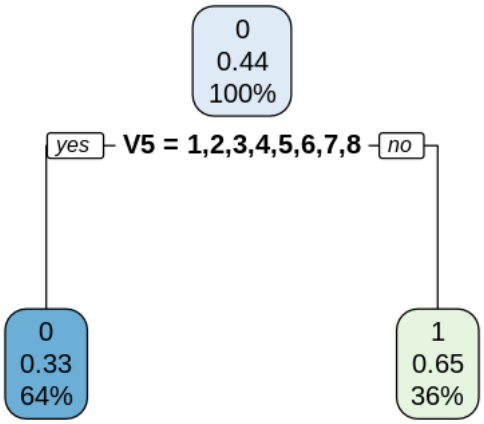

```
In [11]: sd <- split(myData$V5,cutree(cluster, 4))
         for(i in 1:4) {
             print(table(sd[i]))
         }
```

```
  1   2   3   4   5   6   7   8   9  10  11  12  13  14
 52  30  59  51  10  54  38 144  64  25  78   3  40  38

 1  2  3  4  5  6  7  8  9 10 11 12 13 14
 0  0  0  0  0  0  0  1  0  0  0  0  1  0

 1  2  3  4  5  6  7  8  9 10 11 12 13 14
```

```
 0  0  0  0  0  0  0  1  0  0  0  0  0  0

 1  2  3  4  5  6  7  8  9 10 11 12 13 14
 1  0  0  0  0  0  0  0  0  0  0  0  0  0
```

We can conclude from the reults that majority of instances will take part on cluster 1, so cutting tree on fourth level might not be a valuable operation. Now let us take a look at decision tree if we can observe anything significant from entropy values.

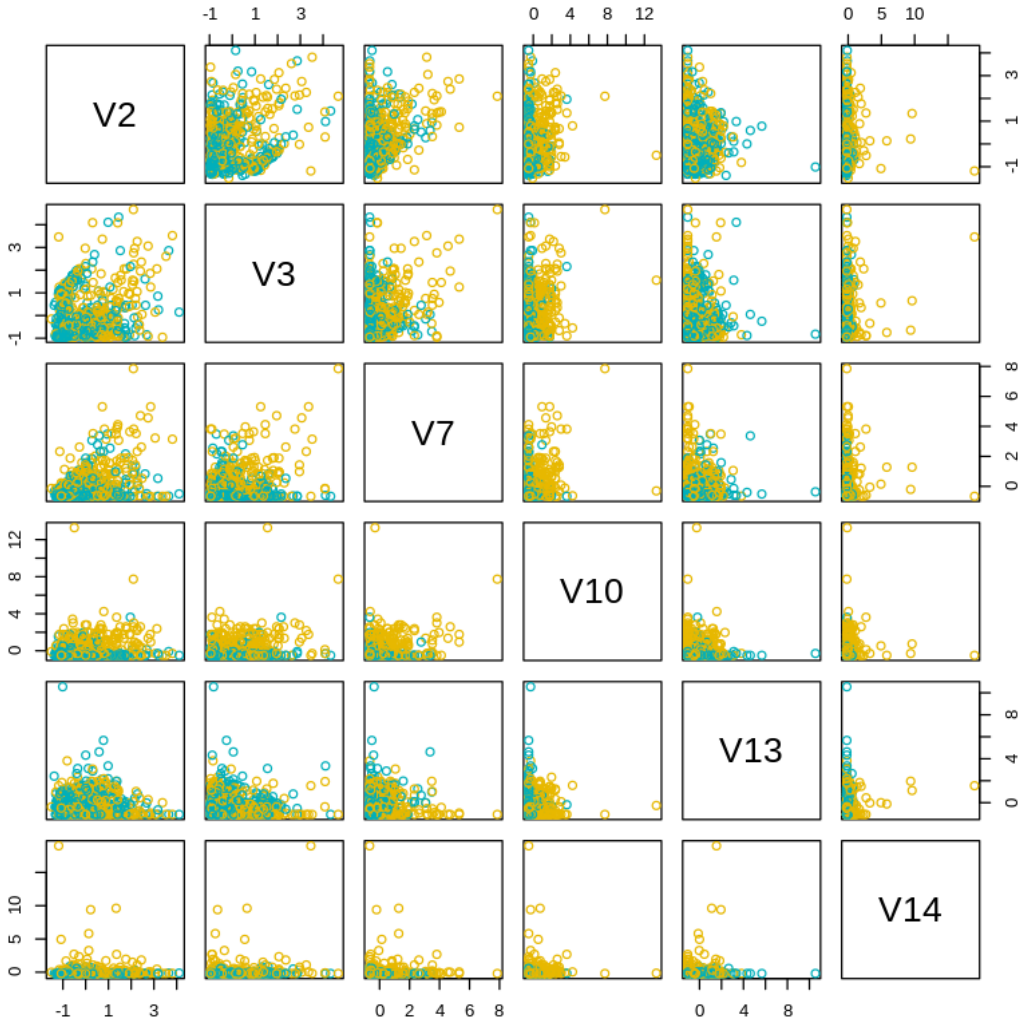
```
In [12]: treeModel.model <- rpart(V15 ~ V5, data=myData, method='class')
         rpart.plot(treeModel.model)
```

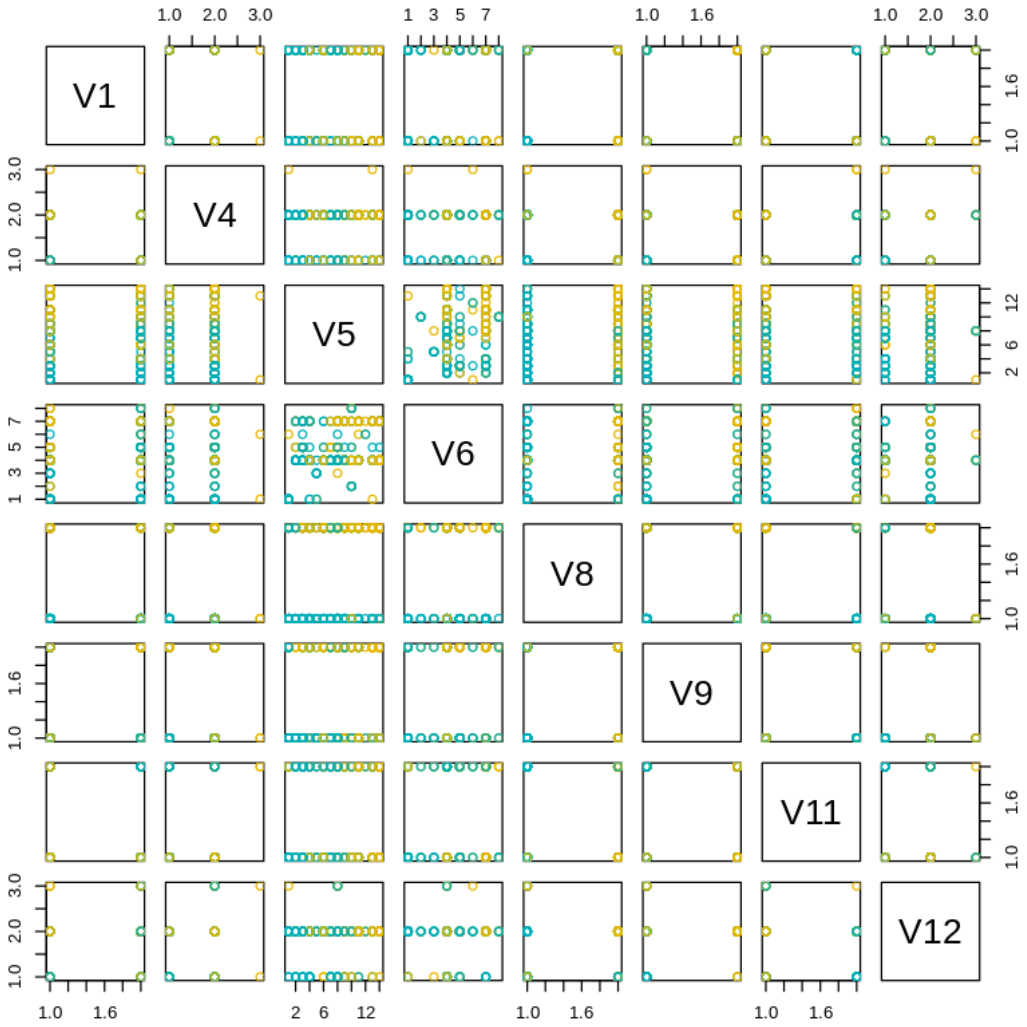


As we can see, decision tree could not give satisfying results for evaluation on feature V5. Because we can see that 0.33 of the data if %64 accurate and 0.65 of data is %36 accurate. These values are not reliable enough for consideration.

Now let us take a look from a visual perspective to our data. We are going to use R's pairs function in order to determine if there is something significant between features as pairs.

```
In [13]: my_cols <- c("#00AFBB", "#E7B800")
         pairs(myData[,numerical_features], col=my_cols[myData$V15])
         pairs(myData[,categorical_features], col=my_cols[myData$V15])
```





From these graphs, we can conclude that finance dataset is applicable and safe for regression functions. We might need to arrange features, maybe subtracting certain ones if we would observe some linear pattern. These kinds of linearity means that features who behaves as such have non-colinear behavior.

2.2 Contingency matrix

From visual methods, making comments about categorical data is rather difficult. We do not know weights on each possible values and can not tell their behaviour by looking graphics. Instead, we are going to look at contingency matrix on some features. Apparently, calculating all contingency tables for all features is not feasible so we are going to look several features' contingency tables. After this, we will look at chi-square tests to see if each selected features are independent from each other or not.

In [142]:
```
table(myData$V8, myData$V12)
chisq.test(myData$V8, myData$V12, simulate.p.value = TRUE)

table(myData$V8, myData$V5)
chisq.test(myData$V8, myData$V5, simulate.p.value = TRUE)

table(myData$V9, myData$V11)
chisq.test(myData$V9, myData$V11)
```

```
   1   2   3
0 36 285   8
1 21 340   0
```

```
Pearson's Chi-squared test with simulated p-value (based on 2000
replicates)

data:  myData$V8 and myData$V12
X-squared = 15.336, df = NA, p-value = 0.0004998
```

```
   1  2  3  4  5  6  7  8  9 10 11 12 13 14
0 41 20 39 31  6 24 18 73 28 10 20  1 12  6
1 12 10 20 20  4 30 20 73 36 15 58  2 29 32
```

```
Pearson's Chi-squared test with simulated p-value (based on 2000
replicates)

data:  myData$V8 and myData$V5
X-squared = 73.222, df = NA, p-value = 0.0004998
```

```
   0   1
0 217 178
1 157 138
```

```
Pearson's Chi-squared test with Yates' continuity correction

data:  myData$V9 and myData$V11
```

```
X-squared = 0.13724, df = 1, p-value = 0.711
```

From chi-square tests, we can say that at third test, by looking X-squared value and p-value, so we accept the null hyphothesis, which means those features are independent from each other. Test solution is opposite on first two tests, which means they are dependent. To be more explanatory, V12 and V5 are dependent to V8.

2.3 Covariance Matrix

Now let us take a look at covariance matrix. We should observe that result of matrix have a value 1 on diagonal axis and not 1 on others.

```
In [129]: # Numerical features:
          # V2, V3, V7, V13, V14
          cov(myData[,numerical_features])
```

	V2	V3	V7	V10	V13	V14
V2	1.00000000	0.2013152	0.39278787	0.18557383	-0.07715894	0.01853870
V3	0.20131524	1.0000000	0.29890156	0.27120674	-0.22234629	0.12312115
V7	0.39278787	0.2989016	1.00000000	0.32232967	-0.07638852	0.05134493
V10	0.18557383	0.2712067	0.32232967	1.00000000	-0.11980806	0.06369244
V13	-0.07715894	-0.2223463	-0.07638852	-0.11980806	1.00000000	0.06560887
V14	0.01853870	0.1231212	0.05134493	0.06369244	0.06560887	1.00000000

2.4 Logistic Regression

We will use R's glm function for creating a generalized linear model. Parameters are placed as:

Feature to predict ~ Features that will be used for predicting (adding them with addition means using them together), data to use, family.

After that, we simply test our model and make predictions; then print it on a confusion matrix. Below, we are testing our dataset for all features on logistic regression.

```
In [36]: model <- glm(V15 ~ . , data=myData[train,], binomial)
         oddsratio <- exp(predict(model,myData[test,]))
         predicted <- ifelse(oddsratio > 0.7,1,0)
         confusionMatrix(table(predicted,real=myData[test,15]))
```

```
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred

Confusion Matrix and Statistics

         real
predicted  0  1
        0 88  4
        1 14 66
```

```
               Accuracy : 0.8953
                 95% CI : (0.8397, 0.9368)
    No Information Rate : 0.593
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.7879
 Mcnemar's Test P-Value : 0.03389

            Sensitivity : 0.8627
            Specificity : 0.9429
         Pos Pred Value : 0.9565
         Neg Pred Value : 0.8250
             Prevalence : 0.5930
         Detection Rate : 0.5116
   Detection Prevalence : 0.5349
      Balanced Accuracy : 0.9028

       'Positive' Class : 0
```

Using all features gave a good result. Now let us take a look at anova and summary of model that includes all features to see if we can make any deductions.

After executing the cell above, we came across a warning that says:

Warning message: "glm.fit: fitted probabilities numerically 0 or 1 occurred"

This warning means logistic regression acquired a probability either exactly ONE or ZERO. So R is warning us in case of making over fitting.

```
In [33]: anova(model)
         summary(model)
```

	Df	Deviance	Resid. Df	Resid. Dev
NULL	NA	NA	517	714.3585
V1	1	0.03223381	516	714.3263
V2	1	11.75240978	515	702.5739
V3	1	9.54742828	514	693.0264
V4	2	12.57345747	512	680.4530
V5	13	85.84356258	499	594.6094
V6	7	6.84781907	492	587.7616
V7	1	21.87411565	491	565.8875
V8	1	197.07862224	490	368.8089
V9	1	19.32813512	489	349.4807
V10	1	5.04040619	488	344.4403
V11	1	1.37543596	487	343.0649
V12	2	10.88824204	485	332.1767
V13	1	5.83671496	484	326.3399
V14	1	17.23341401	483	309.1065

```
Call:
glm(formula = V15 ~ ., family = binomial, data = myData[train,
    ])

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4619  -0.3062  -0.1205   0.4591   3.2022

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   -4.49470    1.05377  -4.265    2e-05 ***
V11           -0.08313    0.34722  -0.239  0.81078
V2             0.14810    0.18268   0.811  0.41752
V3            -0.19938    0.16128  -1.236  0.21638
V42            0.51299    0.37297   1.375  0.16900
V43           14.19758 1023.60423   0.014  0.98893
V52            5.09061    2.44231   2.084  0.03713 *
V53            4.44189    2.35201   1.889  0.05895 .
V54            4.23610    2.24173   1.890  0.05880 .
V55            0.25699    2.24326   0.115  0.90879
V56            4.47178    2.31202   1.934  0.05310 .
V57            4.71659    2.34789   2.009  0.04455 *
V58            4.62698    2.27924   2.030  0.04235 *
V59            5.44701    2.31905   2.349  0.01883 *
V510           6.18767    2.56691   2.411  0.01593 *
V511           4.65402    2.28358   2.038  0.04155 *
V512           3.32147    4.34320   0.765  0.44442
V513           5.83584    2.37887   2.453  0.01416 *
V514           7.83342    2.51088   3.120  0.00181 **
V62           -4.54778    3.24751  -1.400  0.16140
V63            1.18367    2.23884   0.529  0.59702
V64           -2.97156    2.14409  -1.386  0.16577
V65           -3.45244    2.22952  -1.549  0.12150
V67           -0.72115    2.48415  -0.290  0.77159
V68           -2.38023    2.16438  -1.100  0.27145
V69           -6.25786    2.82279  -2.217  0.02663 *
V7             0.22136    0.18490   1.197  0.23123
V81            3.78872    0.40479   9.360  < 2e-16 ***
V91            0.71305    0.42217   1.689  0.09122 .
V10            0.44779    0.29768   1.504  0.13250
V111          -0.01509    0.31340  -0.048  0.96160
V122          -0.26169    0.52245  -0.501  0.61644
V123           3.62320    1.13231   3.200  0.00138 **
V13           -0.42914    0.17042  -2.518  0.01180 *
V14            3.44539    1.11492   3.090  0.00200 **
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 714.36  on 517  degrees of freedom
Residual deviance: 309.11  on 483  degrees of freedom
AIC: 379.11

Number of Fisher Scoring iterations: 15
```

From anova table, we can observe that deviance values of V1, V11 are significantly low, this means these features have equally small amounts of effect on model. Therefore we can subtract these two columns.

We can observe that using factor function creates one-hot encoded feature columns on background. We can see that some categories of V12 are reliable and should be considered whilst making predictions, with V8's first category. In addition, we can see that estimate values of feature V6 are relatively low and high on standard error.

Before changing our model, let us try to change the classification ratio to see if the accuracy will be increased.

```
In [44]: model <- glm(V15 ~ . , data=myData[train,], binomial)
         oddsratio <- exp(predict(model,myData[test,]))
         predicted <- ifelse(oddsratio > 0.5,1,0)
         confusionMatrix(table(predicted,real=myData[test,15]))
```

```
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred

Confusion Matrix and Statistics

         real
predicted  0  1
        0 84  3
        1 18 67

               Accuracy : 0.8779
                 95% CI : (0.8194, 0.9228)
    No Information Rate : 0.593
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.7553
 Mcnemar's Test P-Value : 0.00225

            Sensitivity : 0.8235
            Specificity : 0.9571
         Pos Pred Value : 0.9655
         Neg Pred Value : 0.7882
             Prevalence : 0.5930
         Detection Rate : 0.4884
```

```
   Detection Prevalence : 0.5058
      Balanced Accuracy : 0.8903

       'Positive' Class : 0
```

Changing ratio declined the accuracy. Now we will remove some features to observe differences on accuracy.

```
In [67]: model <- glm(V15 ~ . - V6, data=myData[train,], binomial)
         oddsratio <- exp(predict(model,myData[test,]))
         predicted <- ifelse(oddsratio > 0.7,1,0)
         confusionMatrix(table(predicted,real=myData[test,15]))
```

```
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred

Confusion Matrix and Statistics

         real
predicted  0  1
        0 88  3
        1 14 67

               Accuracy : 0.9012
                 95% CI : (0.8465, 0.9414)
    No Information Rate : 0.593
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.8002
 Mcnemar's Test P-Value : 0.01529

            Sensitivity : 0.8627
            Specificity : 0.9571
         Pos Pred Value : 0.9670
         Neg Pred Value : 0.8272
             Prevalence : 0.5930
         Detection Rate : 0.5116
   Detection Prevalence : 0.5291
      Balanced Accuracy : 0.9099

       'Positive' Class : 0
```

Subtracting V6 increased accuracy a little bit. 0.9012 accuracy is a great ratio. We are going to take a look at precision of numerical features.

```
In [73]: model <- glm(V15 ~ V2 + V3 + V7 + V10 + V13 + V14, data=myData[train,], binomial)
         oddsratio <- exp(predict(model,myData[test,]))
         predicted <- ifelse(oddsratio > 0.7,1,0)
         confusionMatrix(table(predicted,real=myData[test,15]))
```

```
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred

Confusion Matrix and Statistics

         real
predicted  0  1
        0 80 13
        1 22 57

               Accuracy : 0.7965
                 95% CI : (0.7285, 0.854)
    No Information Rate : 0.593
    P-Value [Acc > NIR] : 1.138e-08

                  Kappa : 0.5868
 Mcnemar's Test P-Value : 0.1763

            Sensitivity : 0.7843
            Specificity : 0.8143
         Pos Pred Value : 0.8602
         Neg Pred Value : 0.7215
             Prevalence : 0.5930
         Detection Rate : 0.4651
   Detection Prevalence : 0.5407
      Balanced Accuracy : 0.7993

       'Positive' Class : 0
```

From result of the model above, we can conclude that categorical features have a significant effect on predictions, and they should be emphasized. Below we are going to look numerical model's anova table and summary.

```
In [70]: anova(model)
         summary(model)
```

	Df	Deviance	Resid. Df	Resid. Dev
NULL	NA	NA	517	714.3585
V2	1	11.7838443	516	702.5747
V3	1	9.5430836	515	693.0316
V7	1	45.8632337	514	647.1684
V10	1	81.0050763	513	566.1633
V13	1	0.1897627	512	565.9735
V14	1	33.0455169	511	532.9280

```
Call:
glm(formula = V15 ~ V2 + V3 + V7 + V10 + V13 + V14, family = binomial,
    data = myData[train, ])

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.8653  -0.7896  -0.6808   0.8448   1.8441

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.27653    0.14224   1.944 0.051888 .
V2          -0.02429    0.11683  -0.208 0.835283
V3           0.01772    0.11478   0.154 0.877338
V7           0.76582    0.16771   4.566 4.96e-06 ***
V10          1.30421    0.21837   5.972 2.34e-09 ***
V13         -0.08993    0.11674  -0.770 0.441103
V14          2.38945    0.62143   3.845 0.000121 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 714.36  on 517  degrees of freedom
Residual deviance: 532.93  on 511  degrees of freedom
AIC: 546.93

Number of Fisher Scoring iterations: 7
```

From results of anova and summary values, we can see that V7, V10 and V14 are reliable, also lack of deviance tells us that V13 is not necessary for consideration. Below, we will construct our model from using only reliable features.

```
In [74]: model <- glm(V15 ~ V7 + V10 + V14, data=myData[train,], binomial)
         oddsratio <- exp(predict(model,myData[test,]))
         predicted <- ifelse(oddsratio > 0.7,1,0)
         confusionMatrix(table(predicted,real=myData[test,15]))
```

```
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred

Confusion Matrix and Statistics

          real
predicted  0  1
        0 80 15
        1 22 55

               Accuracy : 0.7849
                 95% CI : (0.7159, 0.8438)
    No Information Rate : 0.593
    P-Value [Acc > NIR] : 7.814e-08

                  Kappa : 0.5612
 Mcnemar's Test P-Value : 0.3239

            Sensitivity : 0.7843
            Specificity : 0.7857
         Pos Pred Value : 0.8421
         Neg Pred Value : 0.7143
             Prevalence : 0.5930
         Detection Rate : 0.4651
   Detection Prevalence : 0.5523
      Balanced Accuracy : 0.7850

       'Positive' Class : 0
```

Using only reliable numerical features can give pretty good solution too. If the dataset would be bigger, it might be useful to use only these reliable features in order to obtain faster calculations, but it is definitely preferable to include categorical variables. Lastly, for showing untrustable features, we are going to create a model using V2, V3 and V13 and observe its accuracy.

```
In [76]: model <- glm(V15 ~ V2 + V3 + V13, data=myData[train,], binomial)
         oddsratio <- exp(predict(model,myData[test,]))
         predicted <- ifelse(oddsratio > 0.7,1,0)
         confusionMatrix(table(predicted,real=myData[test,15]))
```

```
Confusion Matrix and Statistics

          real
predicted  0  1
        0 40 12
        1 62 58

               Accuracy : 0.5698
```

```
               95% CI : (0.4922, 0.6449)
    No Information Rate : 0.593
    P-Value [Acc > NIR] : 0.7583

                  Kappa : 0.1985
 Mcnemar's Test P-Value : 1.226e-08

            Sensitivity : 0.3922
            Specificity : 0.8286
         Pos Pred Value : 0.7692
         Neg Pred Value : 0.4833
             Prevalence : 0.5930
         Detection Rate : 0.2326
   Detection Prevalence : 0.3023
      Balanced Accuracy : 0.6104

       'Positive' Class : 0
```

Clearly, result is not good and should not be applied.

2.5 Support Vector Machine

Now let us try to implement Support Vector Machine (SVM) algorithm.

First, we will try to implement SVM for all features to see if categorical features.
We are creating a model from R's SVM function. After that we are making predictions, showing our results on a confusion matrix after testing the predictions.

```
In [91]: model <- svm(V15 ~ ., data=myData[train,])
         predictions <- predict(model, myData[test,])
         confusionMatrix(table(predictions, myData[test,15]))
```

```
Confusion Matrix and Statistics


predictions  0  1
          0 85  4
          1 17 66

               Accuracy : 0.8779
                 95% CI : (0.8194, 0.9228)
    No Information Rate : 0.593
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7542
 Mcnemar's Test P-Value : 0.008829

            Sensitivity : 0.8333
```

```
            Specificity : 0.9429
         Pos Pred Value : 0.9551
         Neg Pred Value : 0.7952
             Prevalence : 0.5930
         Detection Rate : 0.4942
   Detection Prevalence : 0.5174
      Balanced Accuracy : 0.8881

       'Positive' Class : 0
```

SVM gave results just as good as logistic regression on all data. A model that has 0.8779 can be considered as a successful model if over fitting has not been made. After this result, we can compare results from all features and only numerical features to have an idea of how categorical features are beneficial, whilst measuring the goodness of numerical features.

```
In [97]: model <- svm(V15 ~ V2 + V3 + V7 + V10 + V13 + V14, data=myData[train,])
         predictions <- predict(model, myData[test,])
         confusionMatrix(table(predictions, myData[test,15]))
```

```
Confusion Matrix and Statistics


predictions  0  1
          0 89 17
          1 13 53

               Accuracy : 0.8256
                 95% CI : (0.7605, 0.8791)
    No Information Rate : 0.593
    P-Value [Acc > NIR] : 4.938e-11

                  Kappa : 0.6354
 Mcnemar's Test P-Value : 0.5839

            Sensitivity : 0.8725
            Specificity : 0.7571
         Pos Pred Value : 0.8396
         Neg Pred Value : 0.8030
             Prevalence : 0.5930
         Detection Rate : 0.5174
   Detection Prevalence : 0.6163
      Balanced Accuracy : 0.8148

       'Positive' Class : 0
```

Numerical features also makes creating a solid model possible since confusion matrix has 0.8256 accuracy value.

Now, let us go further and make predictions with using different kernels.

```
In [99]: model <- svm(V15 ~ ., data=myData[train,], kernel="polynomial")
         predictions <- predict(model, myData[test,])
         confusionMatrix(table(predictions, myData[test,15]))
```

```
Confusion Matrix and Statistics


predictions   0   1
          0 102  55
          1   0  15

               Accuracy : 0.6802
                 95% CI : (0.605, 0.7492)
    No Information Rate : 0.593
    P-Value [Acc > NIR] : 0.01144

                  Kappa : 0.2444
 Mcnemar's Test P-Value : 3.305e-13

            Sensitivity : 1.0000
            Specificity : 0.2143
         Pos Pred Value : 0.6497
         Neg Pred Value : 1.0000
             Prevalence : 0.5930
         Detection Rate : 0.5930
   Detection Prevalence : 0.9128
      Balanced Accuracy : 0.6071

       'Positive' Class : 0
```

Using a different kernel does not supposed to be a useful idea all the time. From the code snippet above, we see that using a polynomial kernel on all features created a worse model than radial kernel (Default kernel on SVM is radial kernel). Some other kernels:

```
In [102]: model <- svm(V15 ~ ., data=myData[train,], kernel="linear")
          predictions <- predict(model, myData[test,])
          confusionMatrix(table(predictions, myData[test,15]))
```

```
Confusion Matrix and Statistics


predictions  0  1
          0 86  3
```

```
         1 16 67

               Accuracy : 0.8895
                 95% CI : (0.8329, 0.9322)
    No Information Rate : 0.593
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7776
 Mcnemar's Test P-Value : 0.005905

            Sensitivity : 0.8431
            Specificity : 0.9571
         Pos Pred Value : 0.9663
         Neg Pred Value : 0.8072
             Prevalence : 0.5930
         Detection Rate : 0.5000
   Detection Prevalence : 0.5174
      Balanced Accuracy : 0.9001

       'Positive' Class : 0
```

```
In [103]: model <- svm(V15 ~ ., data=myData[train,], kernel="sigmoid")
          predictions <- predict(model, myData[test,])
          confusionMatrix(table(predictions, myData[test,15]))
```

```
Confusion Matrix and Statistics


predictions  0  1
          0 86  4
          1 16 66

               Accuracy : 0.8837
                 95% CI : (0.8261, 0.9275)
    No Information Rate : 0.593
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.7654
 Mcnemar's Test P-Value : 0.01391

            Sensitivity : 0.8431
            Specificity : 0.9429
         Pos Pred Value : 0.9556
         Neg Pred Value : 0.8049
             Prevalence : 0.5930
         Detection Rate : 0.5000
```

```
Detection Prevalence : 0.5233
   Balanced Accuracy : 0.8930

    'Positive' Class : 0
```

Other kernels seem to be just as good as radial kernel.

3 Analysis

3.1 Distribution of Algorithms

In this section, we are going to test each algorithm for same train and test data splits for several time and take a look at their distributions.

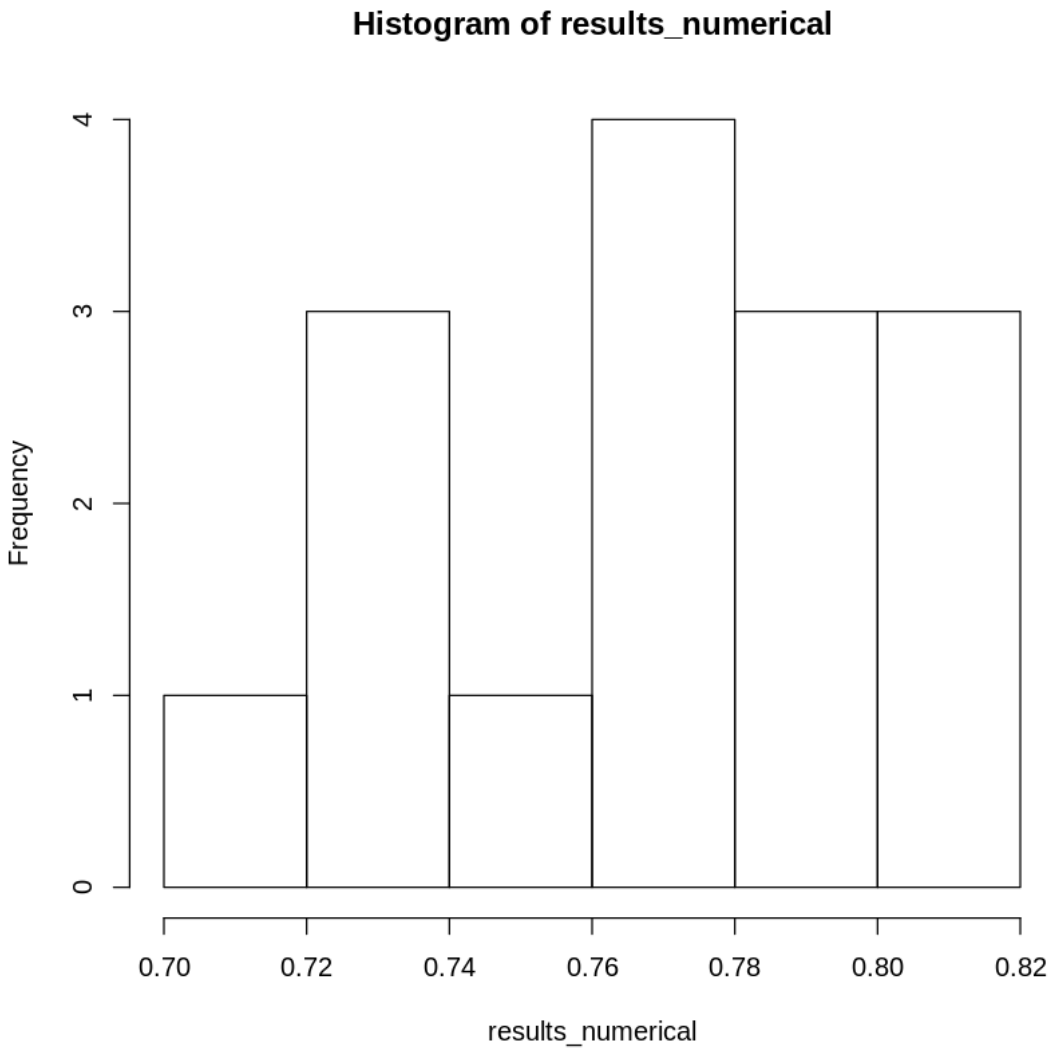
We are going to create random train and test groups, and test each algorithm with same groups. Then we will pass each result to an array and check what their distribution looks like, then we are going to evaluate results.

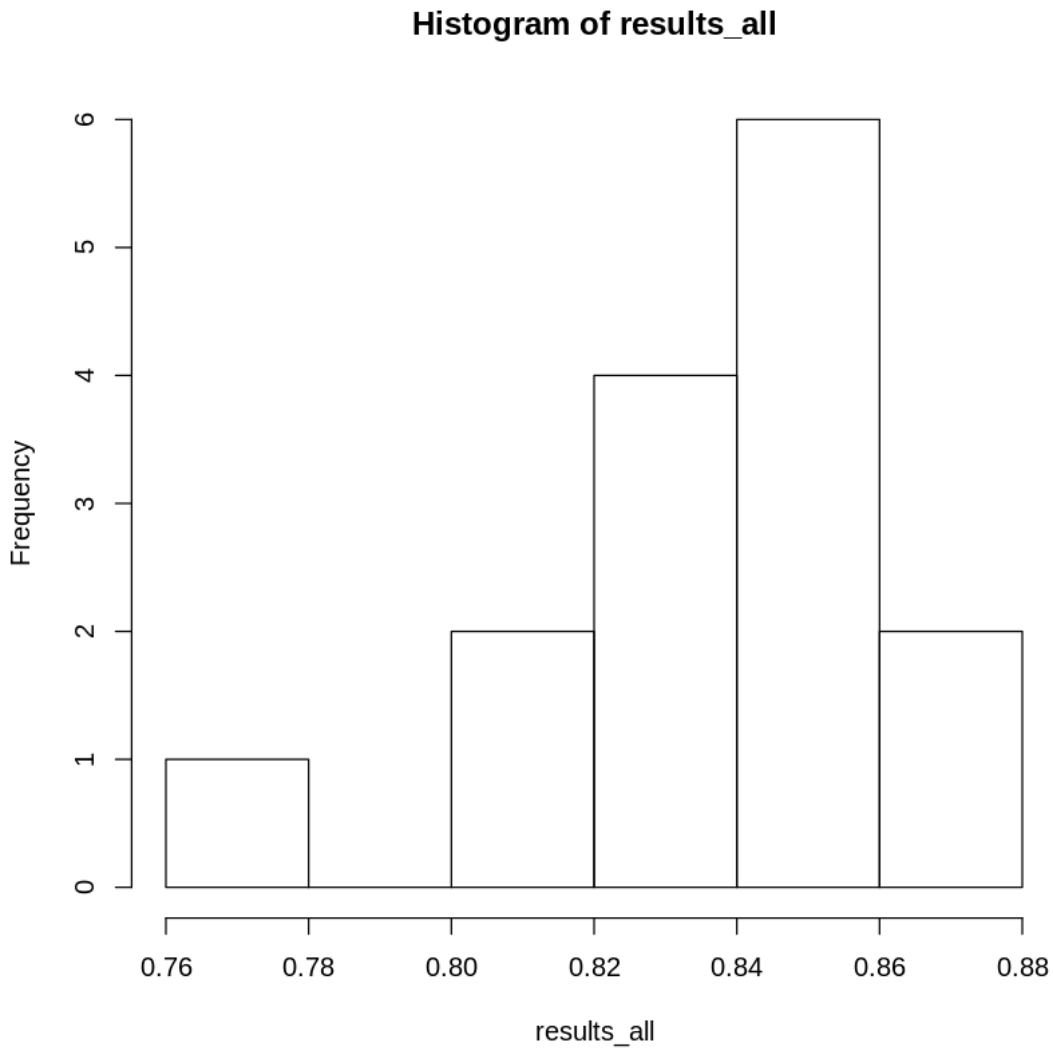
3.1.1 Logistic Regression

```
In [41]: results_numerical <- c()
         results_all <- c()
         for(i in 1:15){
             n <- nrow(myData)
             test <- sample(1:n, n*0.25)
             train <- -test
             model_numerical <- glm(V15 ~ V2 + V3 + V7 + V10 + V13 + V14, data=myData[train,],
             oddsratio_numerical <- exp(predict(model_numerical,myData[test,]))
             predicted_numerical <- ifelse(oddsratio_numerical > 0.75,1,0)
             cm_numerical <- confusionMatrix(table(predicted_numerical,real=myData[test,15]))
             results_numerical <- c(results_numerical, cm_numerical$overall['Accuracy'])


             model_all<- glm(V15 ~ .,  data=myData[train,], binomial)
             oddsratio_all <- exp(predict(model_all,myData[test,]))
             predicted_all <- ifelse(oddsratio_all > 0.75,1,0)
             cm_all <- confusionMatrix(table(predicted_all,real=myData[test,15]))
             results_all <- c(results_all, cm_all$overall['Accuracy'])


         }

         hist(results_numerical)
         hist(results_all)
```

```
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
```

```
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred
```


Histogram of results_numerical
Frequency
0
1
2
3
4
0.70
0.72
0.74
0.76
0.78
0.80
0.82
results_numerical

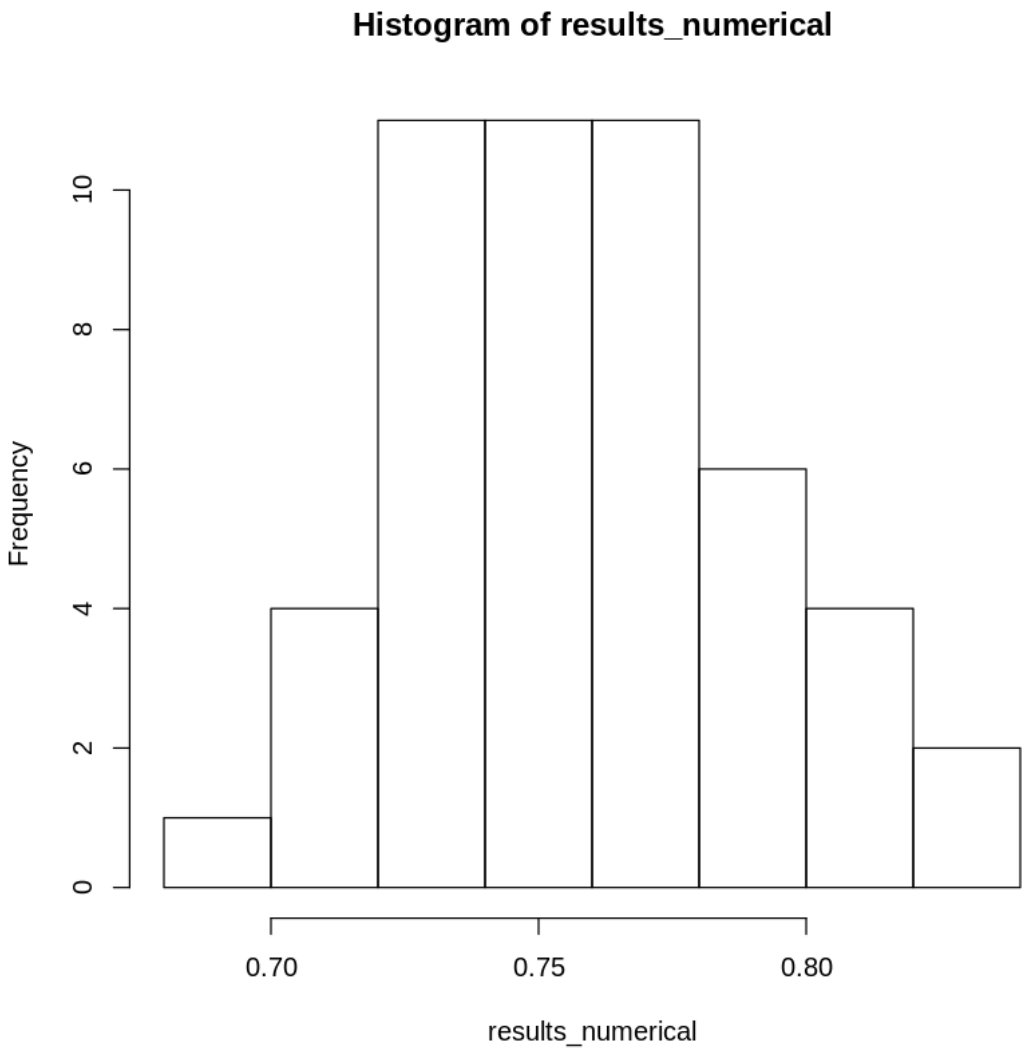


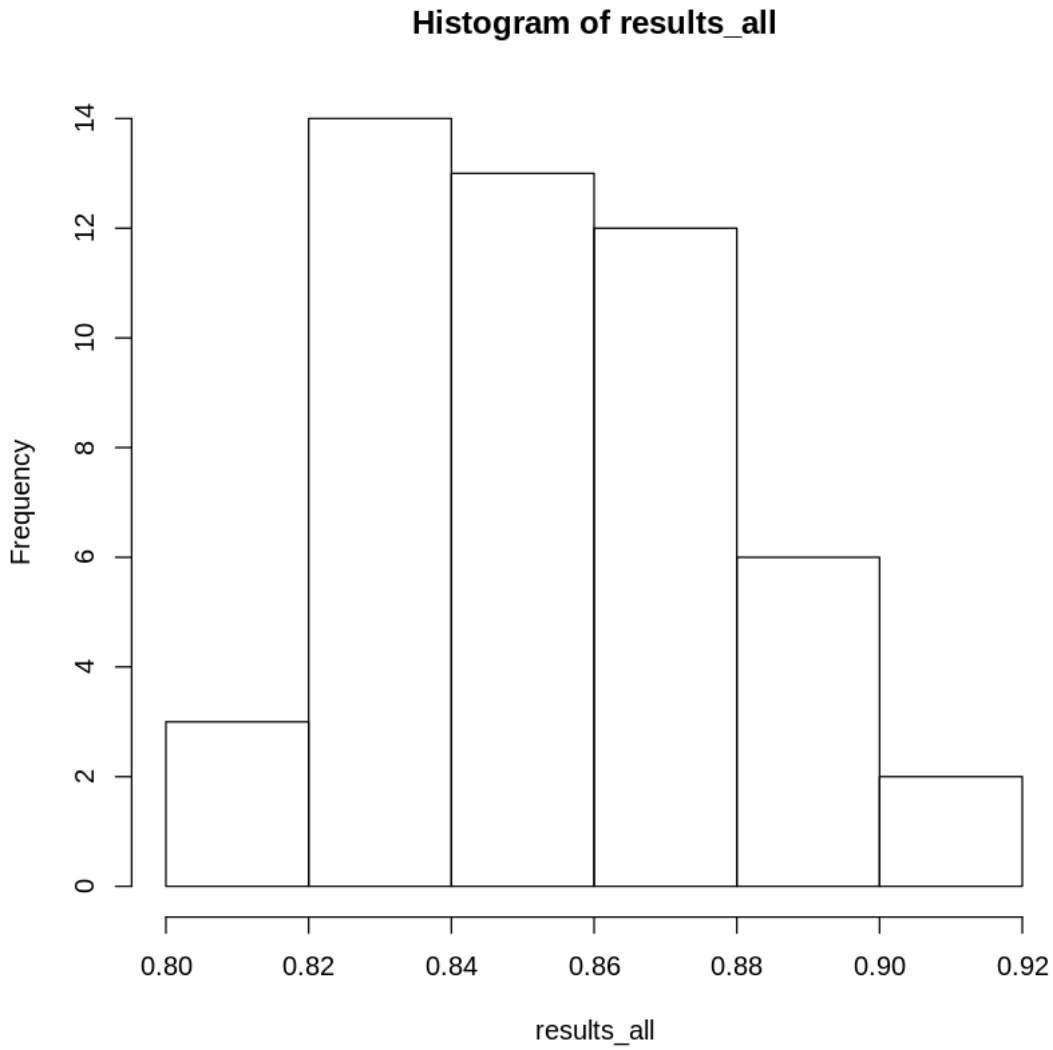
Results of numerical features show us that there are more than one model to converge, so this might create problem for some specific occasions. For example, for a certain random dataset, model might converge to very different solutions even for the smallest nuances. Results for all features looks more consistent on histogram.

3.1.2 Support Vector Machine

```
In [127]: results_numerical <- c()
          results_all <- c()
          for(i in 1:50){
              n <- nrow(myData)
              test <- sample(1:n, n*0.25)
              train <- -test
              model_numerical <- svm(V15 ~ V2 + V3 + V7 + V10 + V13 + V14, data=myData[train,]
```

```
    predictions_numerical <- predict(model_numerical, myData[test,])
    cm_numerical <- confusionMatrix(table(predictions_numerical, myData[test,15]))
    results_numerical <- c(results_numerical, cm_numerical$overall['Accuracy'])

    model_all <- svm(V15 ~ ., data=myData[train,])
    predictions_all <- predict(model_all, myData[test,])
    cm_all <- confusionMatrix(table(predictions_all, myData[test,15]))
    results_all <- c(results_all, cm_all$overall['Accuracy'])


}

hist(results_numerical)
hist(results_all)
```





We can observe that SVM is more reliable upon convergence, its shape looks like t-distribution as should be.

4 Conclusion

On this thesis, we have worked on a finance dataset and tested logistic regression and SVM algorithms. We investigated the nature of dataset, features' specifications and behaviours; and we tried to derive relations and meanings in between. Also we tested logistic regression and SVM's nature, studied specialties on a classification problem and came across several different conclusions about their methodology.

To take this study one step further, deep learning could be discussed; by its nature, deep learning could be an efficient approach for handling this certain classification problem, but we did not surround that on this thesis.

Bibliography

1. Berwick, R. (2011). "An idiot's guide to support vector machines (SVMs)," in 6.034 Artificial Intelligence-Recitations.

2. Quinlan R. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

3. Wang, R. (2016). "Introduction to Support Vector Machines" in http://fourier.eng.hmc.edu/e161/lectures/svm/svm.html

4. (2013). Chapter 12 Logistic Regression https://www.stat.cmu.edu/~cshalizi/uADA/13/lectures/ch12.pdf

5. Figure 1 (2019), Linearly separable data example on support vector machines algorithm, https://scikit-learn.org/stable/auto_examples/svm/plot_svm_margin.html

6. Figure 2 (2017), Kernel Applications on Support Vector Machines, https://cdn-images-1.medium.com/max/1000/0*ngkO1BblQXnOTcmr.png