Machine Learning – CS -584

Class Project - Report and Analysis

Rental Bike Analysis and Prediction

Done

By

Bhasheyam Krishnan - A23080078

Hemanth Balakrishna - A20385274

Content

1. Introduction:

Bicycle sharing frameworks are new age of customary bicycle rentals where entire process from enrolment, rental and return has turned out to be programmed. Through these frameworks, client can undoubtedly lease a bicycle from a position and return at another position. At present, there are about more than 500 bicycle sharing projects the world over which is made from more than 500 thousand bikes. Aside from fascinating certifiable uses of bicycle sharing frameworks, the attributes of information being created by these frameworks make them appealing for the exploration. Restricted to other transport administrations, for example, transport or metro, the span of travel, take off and entry position is expressly recorded in these frameworks. This component transforms bicycle sharing framework into a virtual sensor arrange that can be utilized for detecting portability in the city. Consequently, it is normal that most of remarkable occasions in the city could be identified through checking this information. We can visualise the data to understand the trend. To have more clear image we need some of the machine learning and datamining techniques.

2. Dataset Processing:

We took dataset from UCI Machine Learning Repository. The data is about bike rental business in Portugal throughout the year, which consist of data about the time, hour, day of the month, Season, Day (Holiday, working) and climate details (Temperature, Wind, etc..) of the instance (rented bike). The Process consist of Clearing Outliers, Scaling, Factorizing and Classification of the data.

2.1. Dataset:

We used two dataset one is the observation made on hour basis and another on day basis. We used both the dataset for prediction and classification analysis to find the better model.

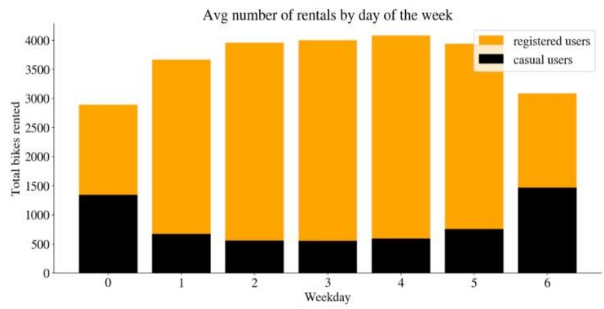
Dataset have three target features Casual, Registered and Count. Count is the sum of registered and casual. These target features are numeric which is type of predicting

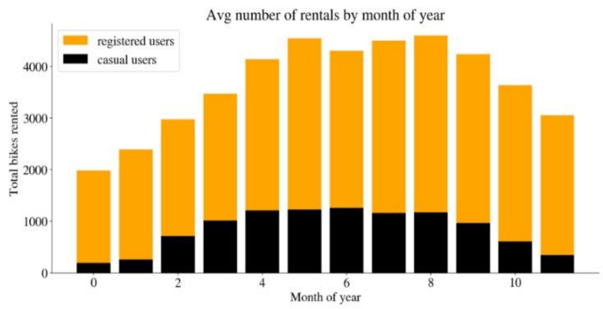
Hour data:

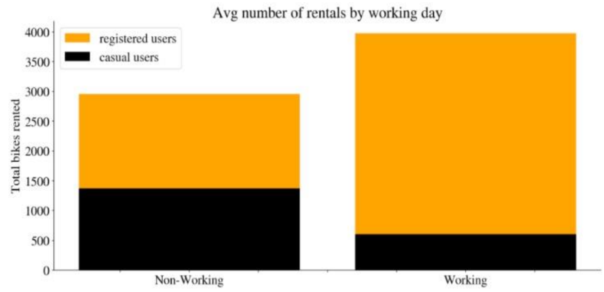
```
 [1] "instant"    "dteday"     "season"     "yr"         "mnth"       "hr"         "holiday"    "weekday"    "workingday"
[10] "weathersit" "temp"       "atemp"      "hum"        "windspeed"  "casual"     "registered" "cnt"
```

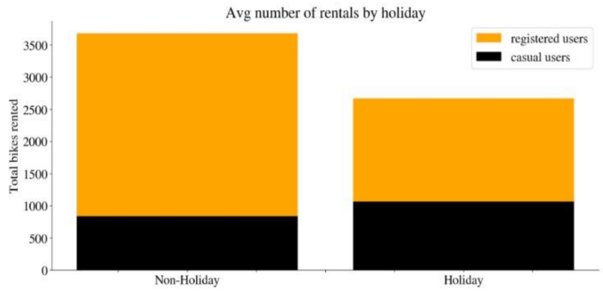
Day data:

```
 [1] "instant"    "dteday"     "season"     "yr"         "mnth"       "holiday"    "weekday"    "workingday" "weathersit"
[10] "temp"       "atemp"      "hum"        "windspeed"  "casual"     "registered" "cnt"
```





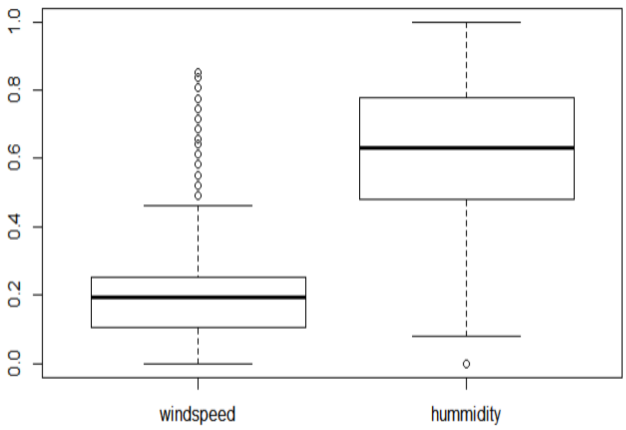


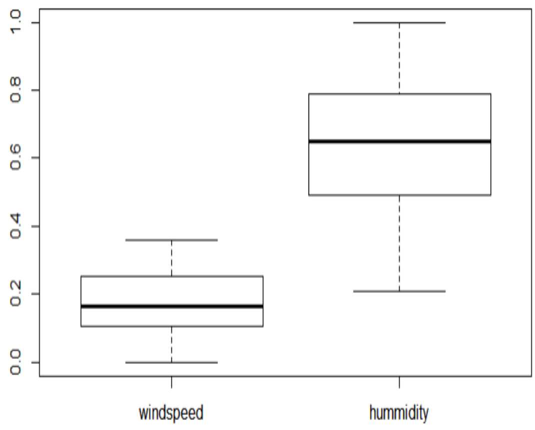


To perform classification, we have included one more feature "count class". Data is generated according to the Count feature, for classifying Register "register class" column is created according to register column.

2.2. Outliers:

The outliers are identified in the Humidity and Windspeed as the features can have peak at some instance and that is not the usual observations so those are observed and removed. Below is before and after removal of outliers





2.3. Data and Technology:

Hour data -- 17379 Rows and 17 Features

Days data – 731 Rows 16 Features

Technology – R, Rmarkdown

Packages – OneR , MLR, Class, ggplot, rpart, fancyplot.

3. Prediction and Classification:

3.1. Baseline:

We **calculated** base line for prediction (regression) and Classification. For Prediction we took the mean of the target classes and for the classification we took the majority class as favourite class those accuracies are given in the below table.

For Regression:

For the Regression problem we considered the mean of the feature as predicted values but the accuracy is very low(20 %) .

For Classification:

Baseline	Registered		Count	
Sample	Train	Test	Train	Test
Classification	65.69	64.54	67.09	67.67

Target = Register, Count, Count class

Count class levels = True, False class (High, low).

Train : Test = 80 : 20

Sampling = Cross - Validation

3.2. Algorithms:

Linear Regression:

Y = B0 + B1 * X + B2 * X1 + . . .

Logistic Regression:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

Decision Tree:

Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision are made using Gini Index, Entropy and Information gain values.

Boosting with Tuning:

1. Initialize the outcome
2. Iterate from 1 to total number of trees
   2.1 Update the weights for targets based on previous run (higher for the ones misclassified)
   2.2 Fit the model on selected subsample of data
   2.3 Make predictions on the full set of observations
   2.4 Update the output with current results considering the learning rate
3. Return the final output.

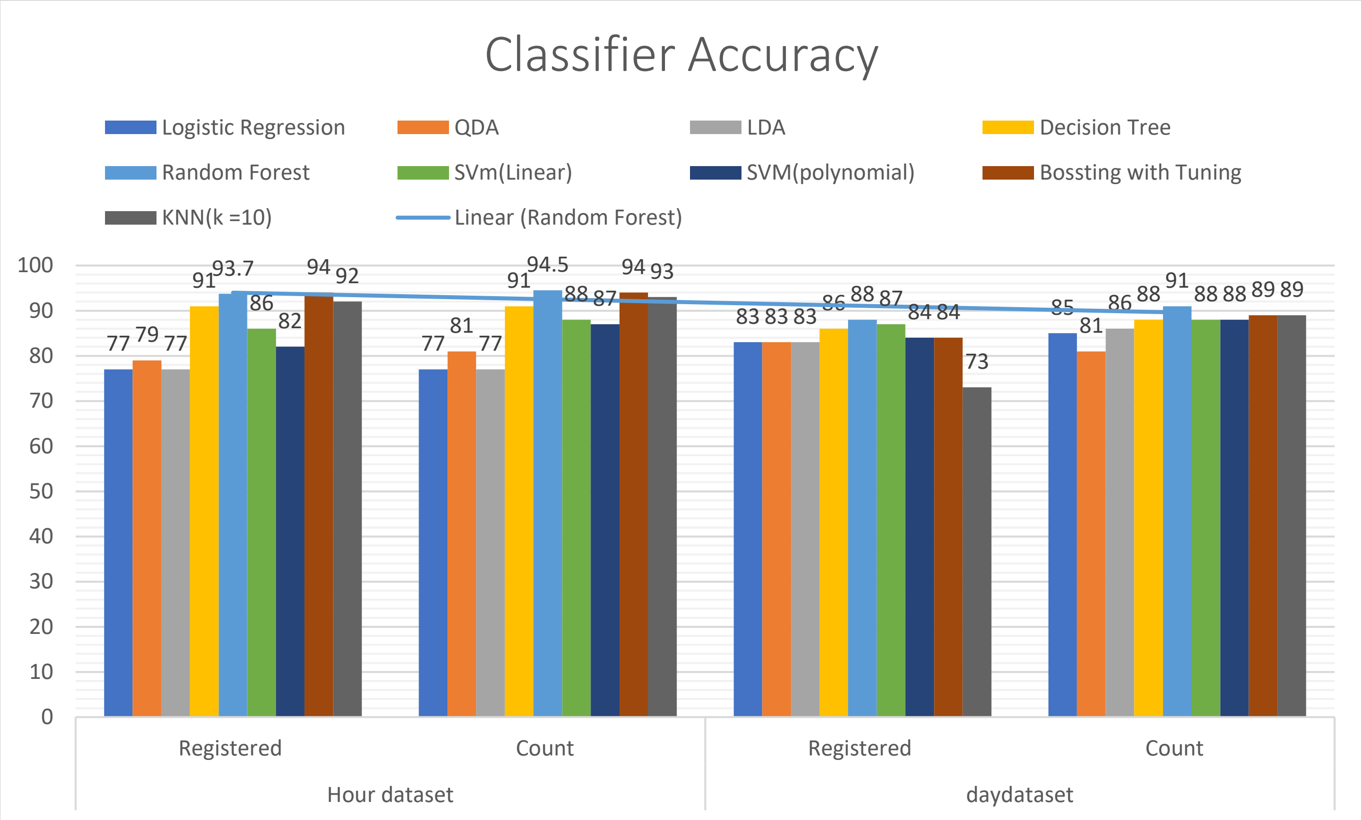
Random Forest:

1) Randomly select "k" features from total "m" features.
   Where k << m
2) Among the "k" features, calculate the node "d" using the best split point.
3) Split the node into daughter nodes using the best split.
4) Repeat 1 to 3 steps until "l" number of nodes has been reached.
5) Build forest by repeating steps 1 to 4 for "n" number times to create "n" number of trees.

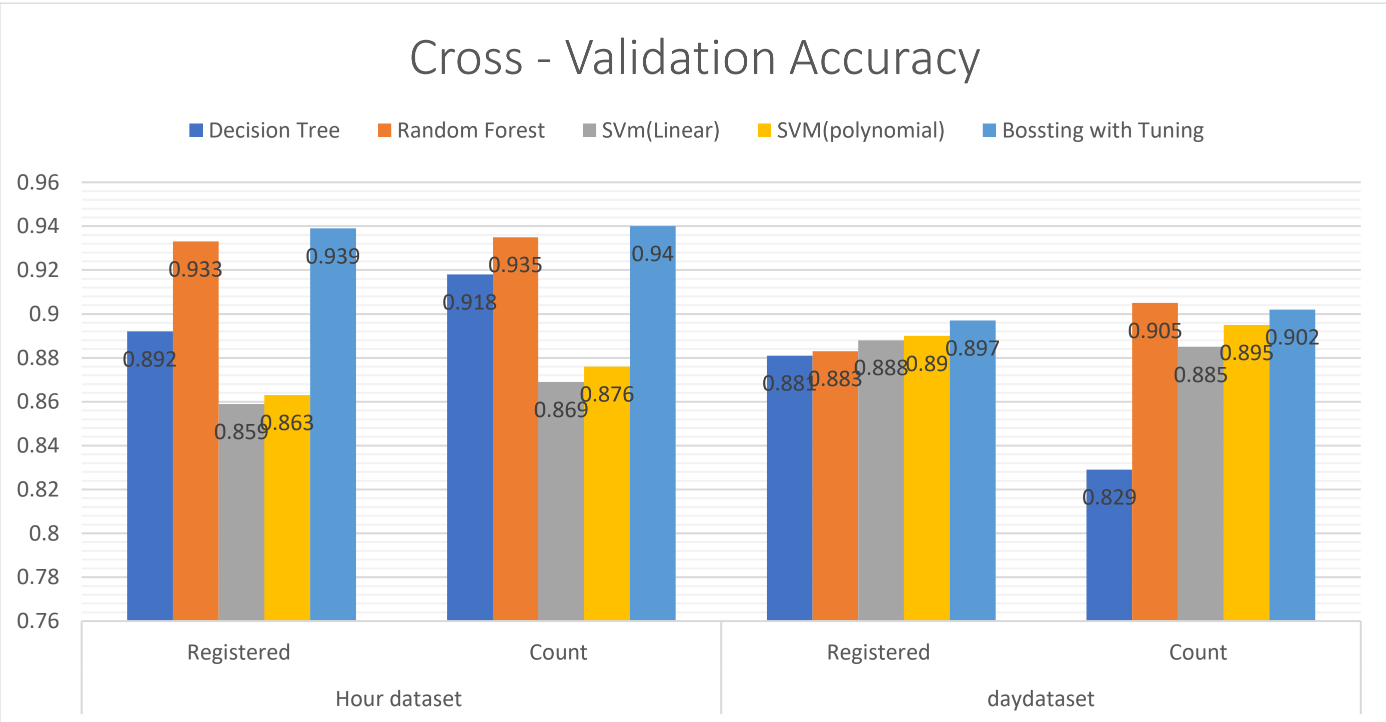
SVM:

$$\vec{w} := \sum_j \alpha_j c_j \vec{d}_j, \quad \alpha_j \geq 0,$$

4. Statistic Summary:



Cross- Validation Accuracy



Predict Accuracy – Linear model Stats:

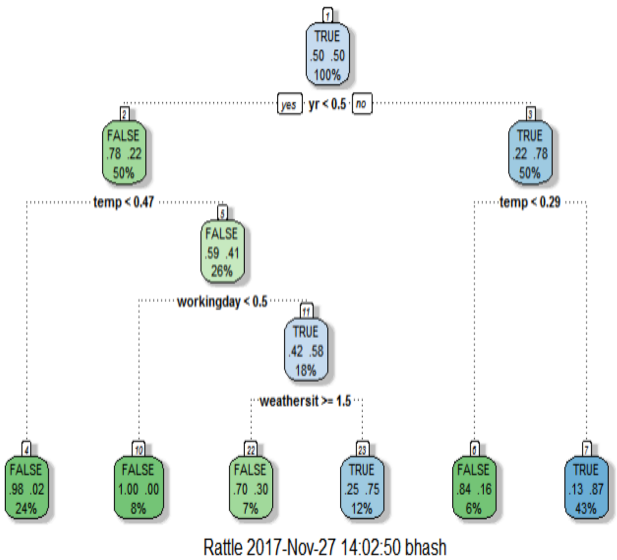
Target	Accuracy	Adj - R2	Accuracy	Adj - R2
Casual	68.3	0.43	82	0.693
Register	55.7	0.336	90.7	0.812
Count	61.52	0.388	90.2	0.792

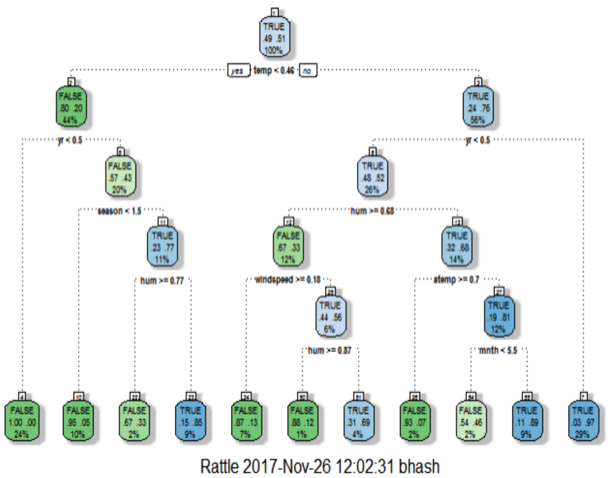
Accuracy Stats - Classifiers:

Classification	Accuracy				Cross - Validation Accuracy			
	Hour dataset		day dataset		Hour dataset		day dataset	
	Registered	Count	Registered	Count	Registered	Count	Registered	Count
Logistic Regression	77	77	83	85	NA	NA	NA	NA
QDA	79	81	83	81	NA	NA	NA	NA
LDA	77	77	83	86	NA	NA	NA	NA
Decision Tree	91	91	86	88	0.892	0.918	0.881	0.829
Random Forest	93.7	94.5	88	91	0.933	0.935	0.883	0.905
SVm(Linear)	86	88	87	88	0.859	0.869	0.888	0.885
SVM(polynomial)	82	87	84	88	0.863	0.876	0.89	0.895
Bossting with Tuning	94	94	84	89	0.939	0.94	0.897	0.902
KNN(k =10)	92	93	73	89	NA	NA	NA	NA

Decision tree:

Following are the decision tree of Register and Count Targets





4.1. Observations and findings:

- Linear regression is poor fit for the given datasets though day dataset produces an 90% accuracy, the amount data we have is less seems to be a over fit. Likewise, we can observer k – means performance is poor in day dataset we lesser data so cluster formation not so effective.

- Random forest tuning and Boosting with tuning performs the best for the dataset with accuracy of 94% for the hour dataset.

- We gave more importance to classification as number prediction is not important as we do in climat prediction or stock prediction.

5. Recommendation Systems & Interface:

From the models we have we can see boosting, decision tree models results are better and have accuracy above 90%. So, developing an application or system which can predict the upcoming hour count or registration. This dataset features are easy obtain as they don't need any human interaction. Say like temperature, humidity, windspeed, time stamp.

From the classification we can calculate the class, this system is will be helpful know the trend and most importantly we have track how good the processing going. Say for example we expected 50 plus registers but only 20 came. So, this show some change in the trend, or else we may assume it to be a normal case.

Day Prediction can be used to see the prediction for the upcoming days and decide accordingly, so the resources will be utilised properly and keep track on the Business.

6. Related work:

Some of the related works are:

- Data Analysis of Bike Rental System in Washington, D.C.(hyperlink)
- And divvy bike share Chicago. For the comparison we took an analysis done on the same dataset. (Hyperlink)

6.1. Performance Comparison:

- Washington D.C analysis they tried to find the change in season or month. They didn't concentrate much on day or time stamp. Moreover they have concentrated on prediction and regression and didn't try classification
- In the second Evaluation, we follow some of the techniques they followed but some of our model out performed their model as they have diluted the data which in turn increased the features and dimension of the dataset. Higher the dimension higher the variation and error.

6.2. Evaluation:

- The highest accuracy for the classification model is 89% for the extra regression tree model, we have the high accuracy of 94% in Boosting with tuning algorithm and random forest. As we mentioned before this due to the more number of features. Also they have not used any sampling method so we can cant completely trust the model with one sample and test.
- Regression both had similar result for the day dataset of 80% whereas the linear model of hour dataset results is poor.

- There linear model is better when they tried to find transformed linear regression. We didn't concentrate more on Regressions.

7. Future work:

- To develop an interacting model, which can be updating the model according the input obtained in day to day process. Also, Stats of improvement and down in the Process.
- Using Time series type of algorithm to predict the count and Registers to get more accurate result
- We have classified the use of the bike rental, for the same year time analysis the use public transport and variation in use of the gas in that area would give insights of the other dimensions, also more accuracy predictions

8. Conclusion:

The sparseness of our initial dataset proved to be insufficient for any modelling techniques. Expanding our observation size as well as predictor space greatly decreased the overall MSE of our models. Linear regression did not prove to be accurate enough models to predicting the count. In the end, our random forests and Boosting approach to predict the count and Register gave us our most accuracy on the testing set. With this type of model, we can gain a better idea on how the various features within the dataset effect the Count and Register feature.

Using this analysis, we tired to give the best result to classify the hour or day, but as we mentioned in future work we need inputs from transport and use of car or gas to bring more accurate and appropriate model.

9. Appendix

9.1. Reference:

1) https://www.analyticsvidhya.com/learning-paths-data-science-business-analytics-business-intelligence-big-data/learning-path-r-data-science/
2) https://www.rdocumentation.org/packages/base/versions/3.4.1/topics/library
3) https://cran.r-project.org/web/packages/mlr/vignettes/mlr.html
4) https://rstudio-pubs-static.s3.amazonaws.com/86328_7ffa1e4fb4964ec9b0458abb6a0c75c7.html
5) **http://rstudio-pubs-static.s3.amazonaws.com/275540_535a85b3ca0840dd8cfb1b7ed93fe320.html**

9.2. Code:

Bi-cycle Analysis R code

Bhasheyam and Hemanth

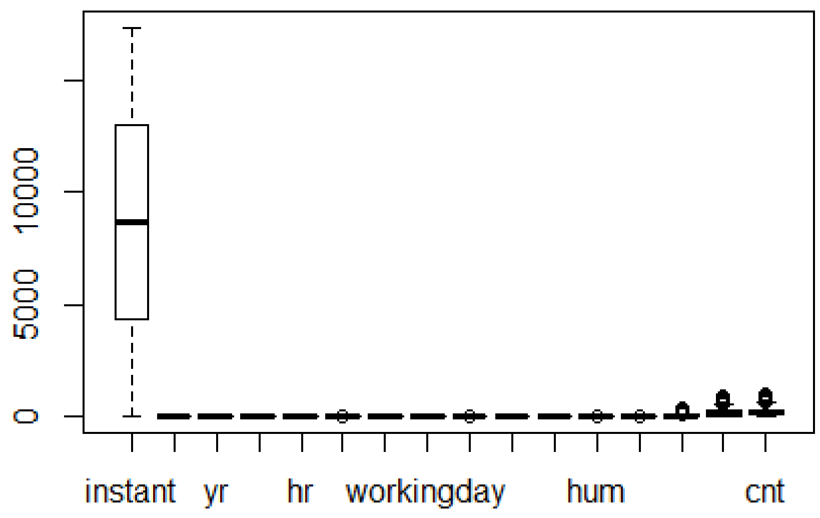
18 October 2017

Read the Data

```
datacy=read.csv("B:/MS/Fall-2017/ML/Project/Bycycle/Data/hour.csv")
dim(datacy)

## [1] 17379    17

fix(datacy)
```

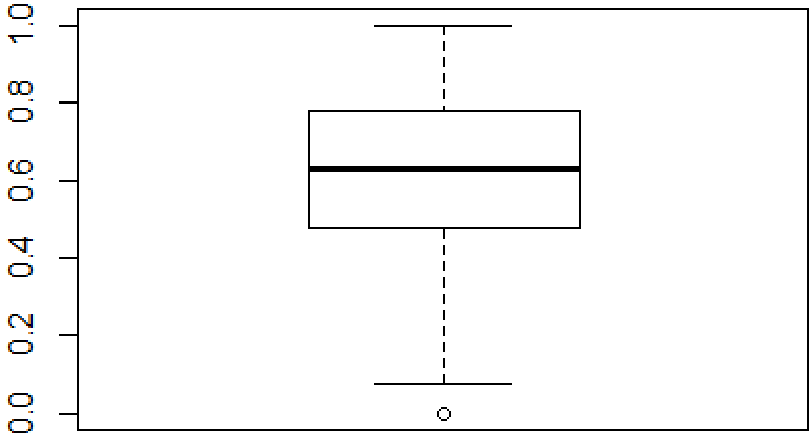
Outliers are removed from the data

```
boxplot(Filter(is.numeric,datacy))
```

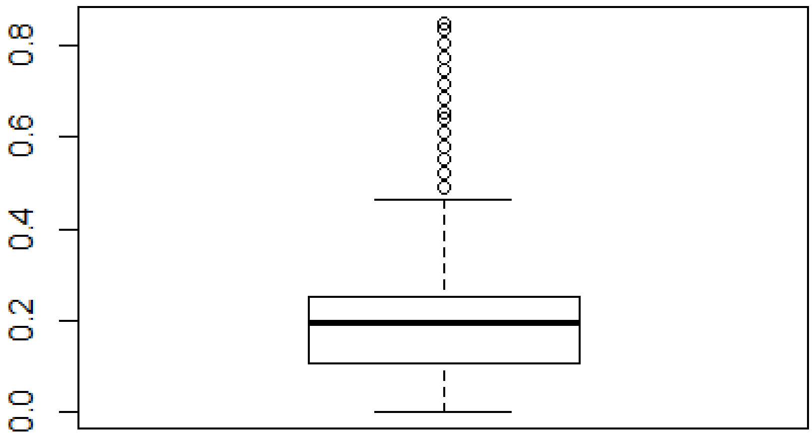


from the above we are able to see hum, windspeed, has some outliers

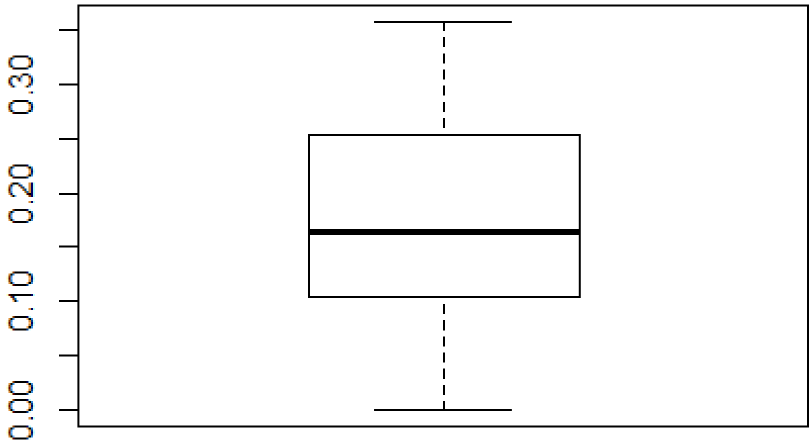
```
boxplot(datacy$hum)
```




```
boxplot(datacy$windspeed)
```



```
changed = datacy$windspeed[datacy$windspeed<0.38]
boxplot(changed)
```

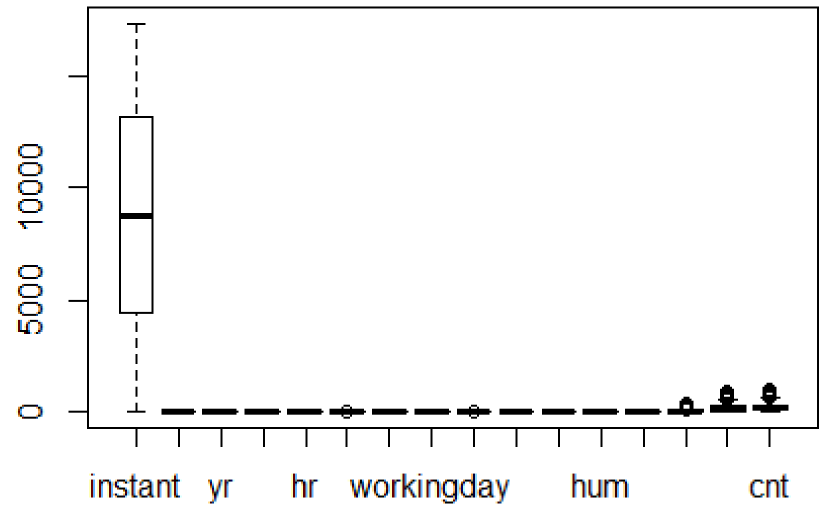


After Removing the Outliers:

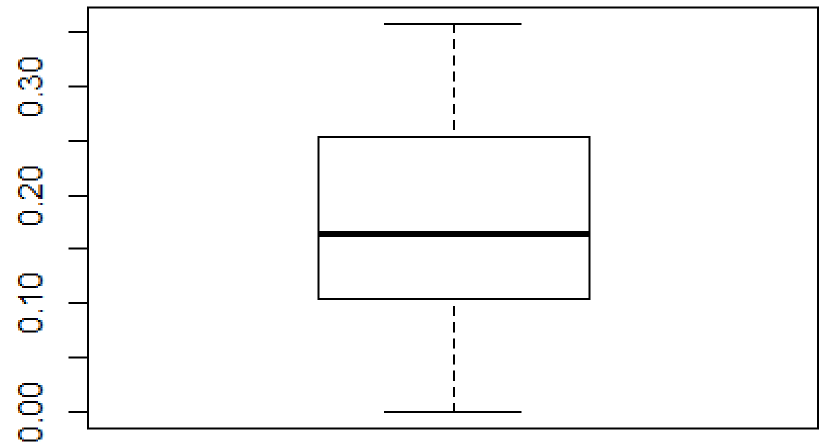
```
dim(datacy)

## [1] 17379    17

datacycle = datacy[datacy$windspeed < 0.37 & datacy$hum > 0.20,]
dim(datacycle)

## [1] 15972    17
```

1392 Instance are removed as they have outliers

```
boxplot(Filter(is.numeric,datacycle))
```




```
boxplot(datacycle$windspeed)
```



to find the better model and learning of the data

Lets Introduced new column as Contclass

analysis 1 - for the Count

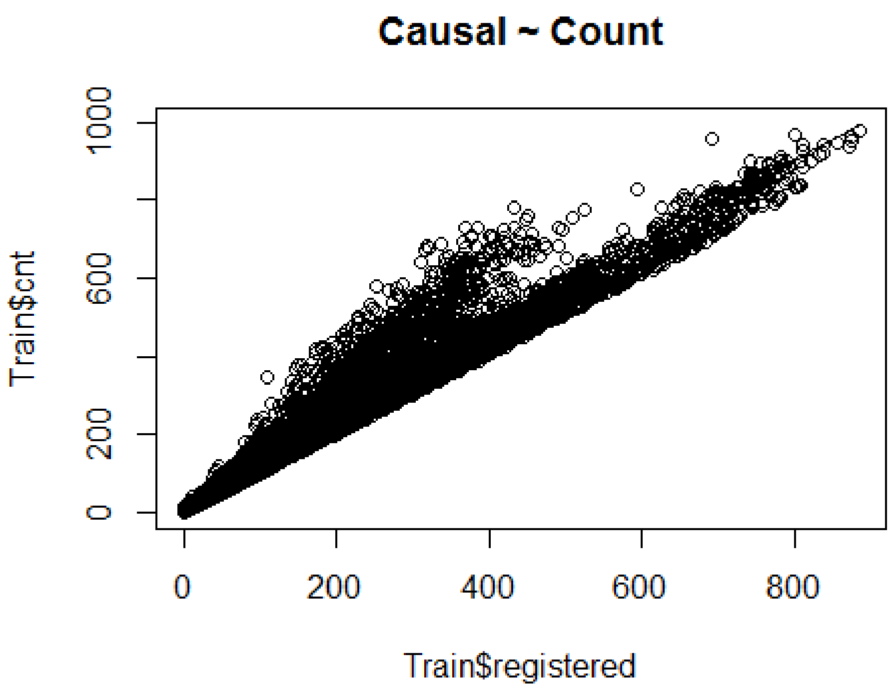
```
contclass = True -> High
contclass = False -> Low

datacycle$countclass = apply(datacycle, 1, function(x)  x[17] > mean(datacycle$cnt ))
set.seed(123)
jumble = runif(nrow(datacycle))
datacycle = datacycle[ordered(jumble),]
sampleindex = sample(2, nrow(datacycle),replace = TRUE, prob = c(0.80, 0.20))
Train = datacycle[sampleindex == 1,]
Test = datacycle[sampleindex == 2,]
dim(Train)

## [1] 12802    18

dim(Test)

## [1] 3170   18

tdata = Train[3:14]
tdata = cbind(tdata,Train[18])

library(OneR)

## Warning: package 'OneR' was built under R version 3.4.2

m =  optbin(tdata)
mod = OneR(m)
summary(mod)

##
## Call:
## OneR.data.frame(x = m)
##
## Rules:
## If atemp = (-0.001,0.559] then countclass = FALSE
## If atemp = (0.559,1]      then countclass = TRUE
##
## Accuracy:
## 8589 of 12802 instances classified correctly (67.09%)
##
## Contingency table:
##           atemp
## countclass (-0.001,0.559] (0.559,1]   Sum
##      FALSE         * 5762      1919  7681
##      TRUE            2294    * 2827  5121
##      Sum             8056      4746 12802
## ---
## Maximum in each column: '*'
##
## Pearson's Chi-squared test:
## X-squared = 1201.5, df = 1, p-value < 2.2e-16
```

```
predictmod = predict(mod,Test)
eval_model(predictmod, Test)
```

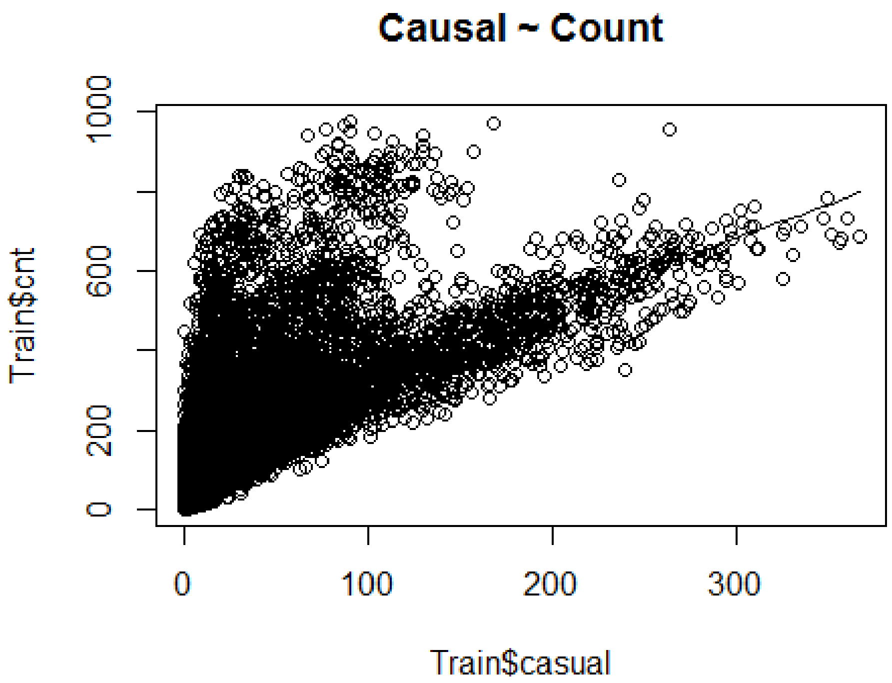
```
##
## Confusion matrix (absolute):
##           Actual
## Prediction FALSE TRUE  Sum
##      FALSE  1456  576 2032
##      TRUE    449  689 1138
##      Sum    1905 1265 3170
##
## Confusion matrix (relative):
##           Actual
## Prediction FALSE TRUE  Sum
##      FALSE  0.46 0.18 0.64
##      TRUE   0.14 0.22 0.36
##      Sum    0.60 0.40 1.00
##
## Accuracy:
## 0.6767 (2145/3170)
##
## Error rate:
## 0.3233 (1025/3170)
##
## Error rate reduction (vs. base rate):
## 0.1897 (p-value < 2.2e-16)
```

LINEAR REGRESSION

```
scatter.smooth(x=Train$registered, y=Train$cnt, main="Causal ~ Count")
```



```
scatter.smooth(x=Train$casual, y=Train$cnt, main="Causal ~ Count")
```




```
##Partioning the dataset to registered users & causal users
x_data = subset(datacycle, select = -c(cnt, countclass))
y_data = subset(datacycle, select = cnt)
x_data = subset(x_data, select = -registered)
x_data = subset(x_data, select = -casual)

## Partioning
#casual
ytrain_casual = Train['casual']
ytest_casual = Test['casual']

xtrain_casual = subset(Train, select = -c(casual, registered, cnt))
xtest_casual = subset(Test, select = -c( casual, countclass, cnt))

x_train = subset(Train, select = -c(registered, cnt, instant, dteday, countclass))
x_test = subset(Test, select = -c(registered, cnt, instant, dteday, countclass))

lmMod <- lm(x_train$casual~. ,  data=x_train)
summary(lmMod)

##
## Call:
## lm(formula = x_train$casual ~ ., data = x_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -95.503 -20.447  -3.669  13.467 274.076
##
## Coefficients:
```

```
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  21.218726   2.128127   9.971  < 2e-16 ***
## season        1.253834   0.543757   2.306  0.02113 *
## yr            9.552786   0.636656  15.005  < 2e-16 ***
## mnth         -0.001053   0.170719  -0.006  0.99508
## hr            1.252732   0.048068  26.061  < 2e-16 ***
## holiday     -10.080291   1.968485  -5.121 3.09e-07 ***
## weekday       0.793014   0.159019   4.987 6.22e-07 ***
## workingday  -34.683961   0.704845 -49.208  < 2e-16 ***
## weathersit    2.661940   0.565574   4.707 2.55e-06 ***
## temp         55.194901  11.520253   4.791 1.68e-06 ***
## atemp        54.504404  13.020840   4.186 2.86e-05 ***
## hum         -71.488740   2.059297 -34.715  < 2e-16 ***
## windspeed     9.064393   3.450285   2.627  0.00862 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 35.81 on 12789 degrees of freedom
## Multiple R-squared:  0.4548, Adjusted R-squared:  0.4543
## F-statistic: 888.9 on 12 and 12789 DF,  p-value: < 2.2e-16
```

```r
casual_predict = predict(lmMod, x_test)
actuals_preds <- data.frame(cbind(actuals=x_test$casual, predicteds=casual_predict))
cor(actuals_preds)
```

```
##              actuals predicteds
## actuals    1.0000000  0.6836118
## predicteds 0.6836118  1.0000000
```

```r
head(actuals_preds)
```

```
##       actuals predicteds
## 13830     108  102.85721
## 15474      10   33.16796
## 16664       2   16.13314
## 12007     258   97.24347
## 10159      12   41.93040
## 15749      29   22.77337
```

68.3% test accuracy for causal users

```r
xtrain_reg = subset(Train, select = -c(casual, cnt, instant, dteday, countclass))
xtest_reg = subset(Test, select = -c(casual, cnt, instant, dteday, countclass))

lmMod_reg <- lm(xtrain_reg$registered~. ,  data=xtrain_reg)
reg_predict = predict(lmMod_reg, xtest_reg)
actuals_preds_reg <- data.frame(cbind(actuals=xtest_reg$registered, predicteds=reg_predict))
cor(actuals_preds_reg)
```

```
##              actuals predicteds
## actuals    1.0000000  0.5572793
## predicteds 0.5572793  1.0000000
```

55.7% test accuracy for registered users

Transformed Linear Regreession

```
Train_cpy = Train
Test_cpy = Test
```

Convert the cnt to Log

```
Train_cpy$log_cnt = log(Train$cnt)
Test_cpy$log_cnt = log(Test$cnt)
xtrain_log = subset(Train_cpy, select = -c(registered, casual, cnt, instant, dteday, countclass))
xtest_log = subset(Test_cpy, select = -c(casual, registered, cnt, instant, dteday, countclass))

lmMod_log <- lm(xtrain_log$log_cnt~. ,  data=xtrain_log)
log_predict = predict(lmMod_log, xtest_log)
actuals_preds_log <- data.frame(cbind(actuals=xtest_log$log_cnt, predicteds=log_predict))
cor(actuals_preds_log)

##            actuals predicteds
## actuals    1.0000000  0.6891219
## predicteds 0.6891219  1.0000000
```

68.91% accuracy

creating train and test task for the classifier analysis

Drop features

```
library(mlr)

## Warning: package 'mlr' was built under R version 3.4.2

## Loading required package: ParamHelpers

## Warning: package 'ParamHelpers' was built under R version 3.4.2

traintaskf = makeClassifTask(data = Train  ,target = "countclass")

## Warning in makeTask(type = type, data = data, weights = weights, blocking =
## blocking, : Empty factor levels were dropped for columns: dteday

traintask  = makeClassifTask(data = Train  ,target = "countclass" , positive = "TRUE")

## Warning in makeTask(type = type, data = data, weights = weights, blocking =
## blocking, : Empty factor levels were dropped for columns: dteday
```

```
traintask = dropFeatures(task = traintaskf, features = c("dteday","instant","cnt","registered","casua
l"))
testtaskf = makeClassifTask(data = Test  ,target = "countclass")
```

```
## Warning in makeTask(type = type, data = data, weights = weights, blocking =
## blocking, : Empty factor levels were dropped for columns: dteday
```

```
testtask = dropFeatures(task = testtaskf, features = c("dteday","instant","cnt","registered","casual")
)
```

classifir -1 QDA - Quadratic Discriminant Analysis

```
cycleqda<- makeLearner("classif.qda", predict.type = "response")
cyclequdatrain = train(cycleqda, task = traintask)
qdapredict = predict(cyclequdatrain, testtask)
table(Test$countclass, qdapredict$data$response)
```

```
##
##        FALSE TRUE
##   FALSE  1609  296
##   TRUE    302  963
```

Here the Accuracy is 81 %

classifier - 2 Logistic Regression

```
cyclelr =  makeLearner("classif.logreg", predict.type = "response")
cyclequdatrain = train(cyclelr, task = traintask)
logrpredict = predict(cyclequdatrain, testtask)
table(Test$countclass, logrpredict$data$response)
```

```
##
##        FALSE TRUE
##   FALSE  1603  302
##   TRUE    412  853
```

Accuracy is 77%

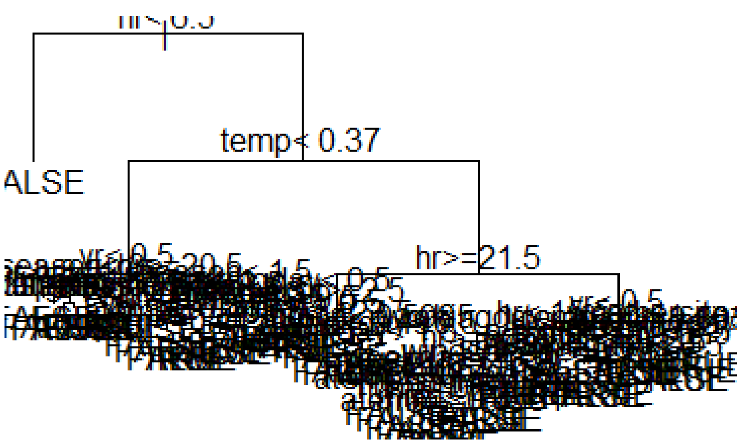
classifier 3 - Desicion tree

```
cycletree =  makeLearner("classif.rpart", predict.type = "response")


treecv =  makeResampleDesc("CV",iters = 10L)


param = makeParamSet(
makeIntegerParam("minsplit",lower = 10, upper = 20),
```

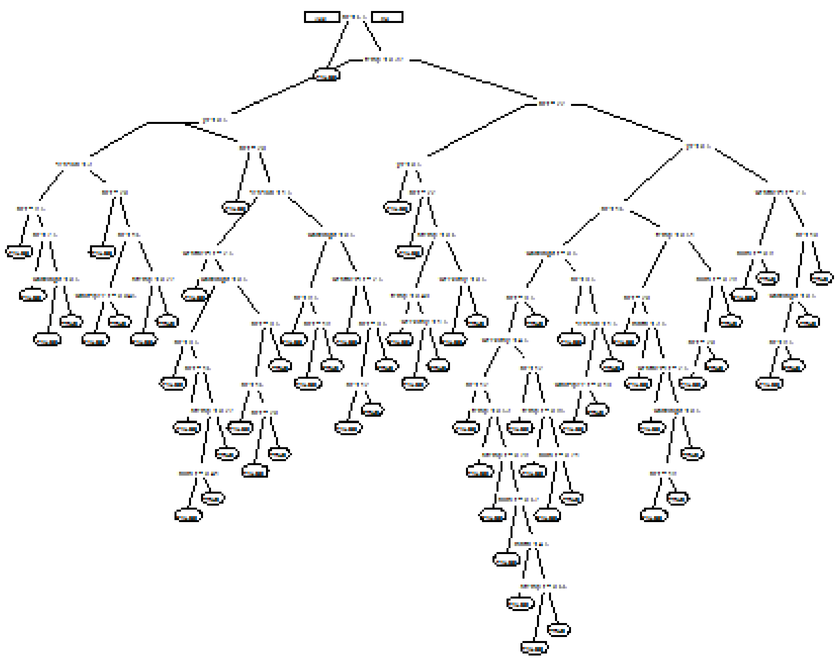
```
makeIntegerParam("minbucket", lower = 5, upper = 10),
makeNumericParam("cp", lower = 0.001, upper = 0.1)
)


control = makeTuneControlGrid()

treetune <- tuneParams(learner = cycletree, resampling = treecv, task = traintask, par.set = param,
control = control, measures = acc)

## [Tune] Started tuning learner classif.rpart for parameter set:

##              Type len Def      Constr Req Tunable Trafo
## minsplit  integer   -   -    10 to 20   -    TRUE     -
## minbucket integer   -   -     5 to 10   -    TRUE     -
## cp        numeric   -   - 0.001 to 0.1   -    TRUE     -

## With control class: TuneControlGrid

## [Tune] Result: minsplit=12; minbucket=5; cp=0.001 : acc.test.mean=0.919

treetune$x

## $minsplit
## [1] 12
##
## $minbucket
## [1] 5
##
## $cp
## [1] 0.001

tree = setHyperPars(cycletree, par.vals = treetune$x)
traintree = train(tree, traintask)
predicttree = predict(traintree, testtask)
table(Test$countclass,predicttree$data$response)

##
##        FALSE TRUE
##   FALSE  1765  140
##   TRUE    115 1150
```

Here the Acuuracy is 91%

```
plot(traintree$learner.model)
text(traintree$learner.model)
```

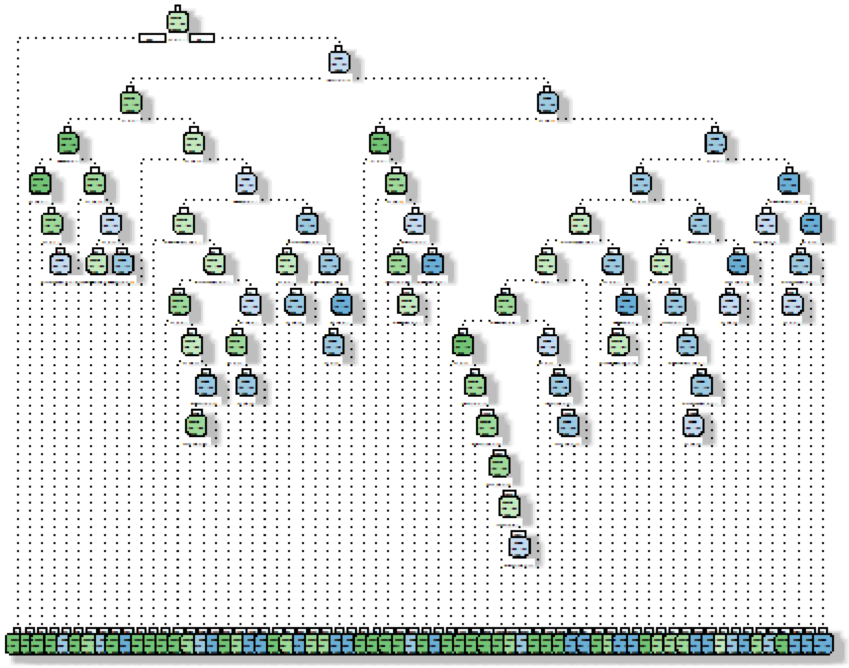



```
library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 3.4.2

## Loading required package: rpart

prp(traintree$learner.model)
```



```
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.4.2

## Rattle: A free graphical interface for data science with R.
## Version 5.1.0 Copyright (c) 2006-2017 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

fancyRpartPlot(traintree$learner.model)

## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Rattle 2017-Nov-26 03:29:20 bhash

random forest classfier

```
randomforest = makeLearner("classif.randomForest",predict.type = "response", par.vals = list(ntree = 200, mtry = 3))
randomforest$par.vals = list(importance = TRUE)

randomparam <- makeParamSet(
makeIntegerParam("ntree",lower = 50, upper = 450),
makeIntegerParam("mtry", lower = 3, upper = 10),
makeIntegerParam("nodesize", lower = 10, upper = 40)
)
randomcontrol = makeTuneControlRandom(maxit = 30L)
randomcross = makeResampleDesc("CV",iter =  10L)
randomtune <- tuneParams(learner = randomforest, resampling = randomcross, task = traintask, par.set = randomparam, control = randomcontrol, measures = acc)

## [Tune] Started tuning learner classif.randomForest for parameter set:

##          Type len Def    Constr Req Tunable Trafo
## ntree    integer   -   - 50 to 450   -    TRUE     -
```

```
## mtry     integer   -   -   3 to 10   -    TRUE     -
## nodesize integer   -   -  10 to 40   -    TRUE     -

## With control class: TuneControlRandom

## Imputation value: -0

## [Tune] Result: ntree=294; mtry=10; nodesize=11 : acc.test.mean=0.935

randomtune$y

## acc.test.mean
##     0.9354789

randomtune$x

## $ntree
## [1] 294
##
## $mtry
## [1] 10
##
## $nodesize
## [1] 11

randomtree = setHyperPars(randomforest, par.vals = randomtune$x)
randomtrain = train(randomtree, traintask)
getLearnerModel(randomtrain)

##
## Call:
##  randomForest(formula = f, data = data, classwt = classwt, cutoff = cutoff,      importance = TRU
E, ntree = 294L, mtry = 10L, nodesize = 11L)
##                Type of random forest: classification
##                      Number of trees: 294
## No. of variables tried at each split: 10
##
##         OOB estimate of  error rate: 6.45%
## Confusion matrix:
##       FALSE TRUE class.error
## FALSE  7258  423  0.05507095
## TRUE    403 4718  0.07869557

randompredict = predict(randomtrain,testtask)
table(Test$countclass, randompredict$data$response)

##
##         FALSE TRUE
##   FALSE  1801  104
##   TRUE     68 1197
```

Here the accuracy is 94.5 %

```
svmlearner = makeLearner("classif.ksvm", predict.type = "response")
```

```
randomcross = makeResampleDesc("CV",iters = 10L)

svmparameter<- makeParamSet(makeNumericParam("C", lower = -5, upper = 5, trafo = functio
n(x) 2^x),
                    makeDiscreteParam("sigma", values = 2^c(-8,-4,0,4))) #RBF Kernel Parameter

svmcontrol = makeTuneControlGrid()
svmtune= tuneParams("classif.ksvm", task = traintask, resampling = randomcross, par.set  = svmp
arameter, control = svmcontrol, measures = acc)

## [Tune] Started tuning learner classif.ksvm for parameter set:

##          Type len Def                Constr Req Tunable Trafo
## C      numeric   -   -              -5 to 5   -    TRUE     Y
## sigma discrete   -   - 0.00390625,0.0625,1,16   -    TRUE     -

## With control class: TuneControlGrid

## Imputation value: -0

## [Tune] Result: C=32; sigma=0.0625 : acc.test.mean=0.876

svmtune$y

## acc.test.mean
##     0.8757983

svmtune$x

## $C
## [1] 32
##
## $sigma
## [1] 0.0625

svmmodel = setHyperPars(svmlearner,par.vals = svmtune$x)
svmtrain = train(svmmodel, traintask)
getLearnerModel(svmtrain)

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 32
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0625
##
## Number of Support Vectors : 4098
##
## Objective Function Value : -109243.2

predictsvm = predict(svmtrain, testtask)
table (Test$countclass , predictsvm$data$response)
```

```
##
##        FALSE TRUE
##   FALSE  1711  194
##   TRUE    196 1069
```

0.87

bossting

```
boost = makeLearner("classif.gbm", predict.type = "response")

gbmcontrol =  makeTuneControlRandom(maxit = 40L)

gbmcv = makeResampleDesc("CV",iters = 10L)






gbmparam =  makeParamSet(makeDiscreteParam("distribution", values = "bernoulli"),
makeIntegerParam("n.trees", lower = 500, upper = 1000), #number of trees
makeIntegerParam("interaction.depth", lower = 2, upper = 6), #depth of tree
makeIntegerParam("n.minobsinnode", lower = 10, upper = 50),
makeNumericParam("shrinkage",lower = 0.01, upper = 0.7))



gbmtune = tuneParams(learner = boost, task = traintask, par.set = gbmparam, control = gbmcontro
l, measures = acc, resampling = gbmcv)

## [Tune] Started tuning learner classif.gbm for parameter set:

##                      Type len Def       Constr Req Tunable Trafo
## distribution      discrete   -   -    bernoulli   -    TRUE     -
## n.trees            integer   -   - 500 to 1e+03   -    TRUE     -
## interaction.depth  integer   -   -       2 to 6   -    TRUE     -
## n.minobsinnode     integer   -   -     10 to 50   -    TRUE     -
## shrinkage          numeric   -   -  0.01 to 0.7   -    TRUE     -

## With control class: TuneControlRandom

## Imputation value: -0

gbmtune$y

## acc.test.mean
##     0.9396958

gbmtune$x

## $distribution
## [1] "bernoulli"
##
```

```
## $n.trees
## [1] 634
##
## $interaction.depth
## [1] 4
##
## $n.minobsinnode
## [1] 30
##
## $shrinkage
## [1] 0.1184034
```

```
gbmboost = setHyperPars(boost,par.vals = gbmtune$x)
```

```
gbmtrain = train(gbmboost,traintask)
gbmpredict = predict(gbmtrain,testtask)
table(Test$countclass,gbmpredict$data$response)
```

```
##
##        FALSE TRUE
##   FALSE  1811   94
##   TRUE     74 1191
```

94 % Accuracy

lda

```
ldalearner = makeLearner("classif.lda", predict.type = "response")
ldatrain = train(ldalearner, traintask)
predictlda = predict(ldatrain, testtask)
table(Test$countclass, predictlda$data$response)
```

```
##
##        FALSE TRUE
##   FALSE  1586  319
##   TRUE    401  864
```

the accuracy is 77% #svm linear

```
svmlearnerl = makeLearner("classif.ksvm", predict.type = "response")

randomcross = makeResampleDesc("CV",iters = 3L)

svmparameter<- makeParamSet(makeNumericParam("C", lower = -5, upper = 5, trafo = functio
n(x) 2^x))


svmcontrol = makeTuneControlGrid()
svmtunel = tuneParams(svmlearnerl, task = traintask, resampling = randomcross, par.set  = svmpar
ameter, control = svmcontrol, measures = acc)
```

```
## [Tune] Started tuning learner classif.ksvm for parameter set:
```

```
##      Type len Def  Constr Req Tunable Trafo
## C numeric   -   - -5 to 5   -    TRUE     Y

## With control class: TuneControlGrid

## [Tune] Result: C=32 : acc.test.mean=0.872

svmtune$x

## $C
## [1] 32
##
## $sigma
## [1] 0.0625

svmtunel$y

## acc.test.mean
##     0.8715826

svmmodell = setHyperPars(svmlearnerl,par.vals = svmtunel$x)
svmtrainl = train(svmmodell, traintask)
getLearnerModel(svmtrainl)

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 32
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0621324884133046
##
## Number of Support Vectors : 4103
##
## Objective Function Value : -109401.7

predictsvml = predict(svmtrainl, testtask)
table (Test$countclass, predictsvml$data$response)

##
##         FALSE TRUE
##   FALSE  1709  196
##   TRUE    196 1069
```

the accuracy is 88%

```
dim(Train)

## [1] 12802    18

library(class)

Train1 = Train[3:14]
Test1 = Test[3:14]
```

```
model=knn(train=Train1,test = Test1, cl=Train$countclass,k=10)
summary(model)
```

```
## FALSE  TRUE
##  1841  1329
```

```
table(Test$countclass,model)
```

```
##        model
##         FALSE TRUE
##   FALSE  1765  140
##   TRUE     76 1189
```

For k = 10 accuracy is 93%

for Higher the K value, the results are not better as the cluster stays very close to each other.