DeepSolar Data Analysis

Nisarga G-19203753

01/05/2020

ABSTRACT

The main task of the project is to predict the target variable in the given data set using supervised classification methods given the set of predictive variables. The complete data analysis is carried out with those supervised methods explained in the course,finally the best model is chosen with highest accuracy.

INTRODUCTION

The dataset used here is a subset of the DeepSolar database, a solar installation database for the US, built by extracting information from satellite images, consisting of 20736 obs and 81 variables out of which 18 are non-numeric variables and 62 are numeric.The main aim is to predict the solar power system coverage using solar_system_count(binary variable) as the target variable. Supervised classification method is used because it is the process of predicting the target variables using a function of input variables.The six methods used here are svm, logistic regression, boosting, bagging, random forest and classification tree.The complete analysis is performed on the dataset.Finally,Hold out sample method of cross-validation is used to evaluate the relative merits of range of classification methods used.

METHODS

LOADING THE DATA:

The data is loaded and stored in variable called data.

```r
data = read.csv("data_project_deepsolar.csv", header = TRUE)
```

Summary of the target variable: This step helps us to know if the data is evenly distributed. The target variable "solar_system_count" contains 10900 observation under "high" and 9836 records under "low" , hence almost evenly distributed which is good.

```r
table(data[,1])

##
##  high   low
## 10900  9836
```

PREPROCESING THE DATA: The data must be prepared before analysis, this step includes missing values analysis, removal of redundant/duplicate records and variables, it is also referred to as data cleaning.

NA Analysis : there are no missing values in the entire data set.

```
sum(sapply(data,function(x)sum(is.na(x))))

## [1] 0
```

Duplicate record removal: There are no duplicates in the dataset.

```
require(dplyr)

## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

distinct <- nrow(data %>%
  distinct())
nrow(data) - distinct

## [1] 0
```

EXPLORATORY DATA ANALYSIS:

This step is a must to have prior knowledge about the data ,its variables. Some of the EDA functions require numeric type variables , therefore we subset a dataframe which contains only numeric columns.

```
num <- data[, sapply(data, class) == "numeric"]
fac <- data[, sapply(data, class) != "numeric"]
```

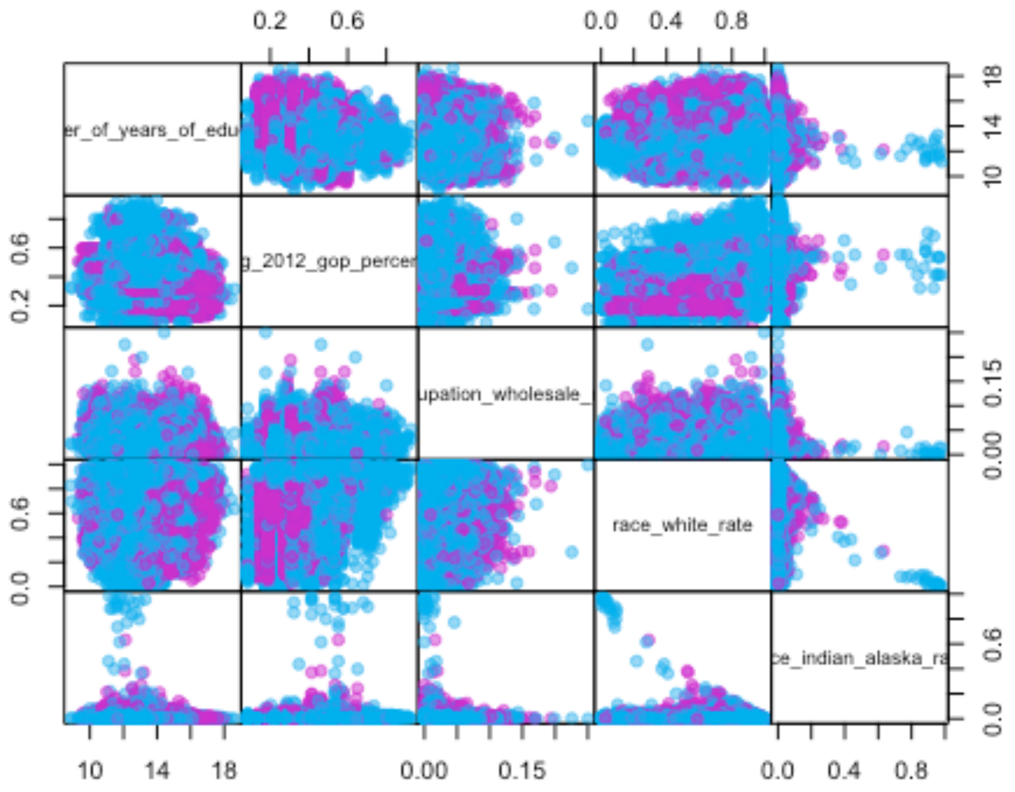
• Pair plots: Only few variables are visualized as there are too many, from the pair plots below we can infer the association between numeric variables.

• The second kind of pair plot is flexible which includes regression lines and box plot, from observing the plot we can infer that there is linear relationship between variables(negative or positive).There are too many outliers for variables race_other_rate, race_two_more rate and so on. The r values gives the correlation, we can see all the variables plotted below are weakly correlated (less than 0.5).

• Parallel coordinates plot : This type of representation is useful to explore differences between classes and observations over different dimensions. The second type highlight the parallel coordinates lines of each class by using the colours.

• Heatmap is a graphical representation of data where the entries contained in a data matrix are directly represented as colors. The intensity of the colours is related to the magnitude of the data points.

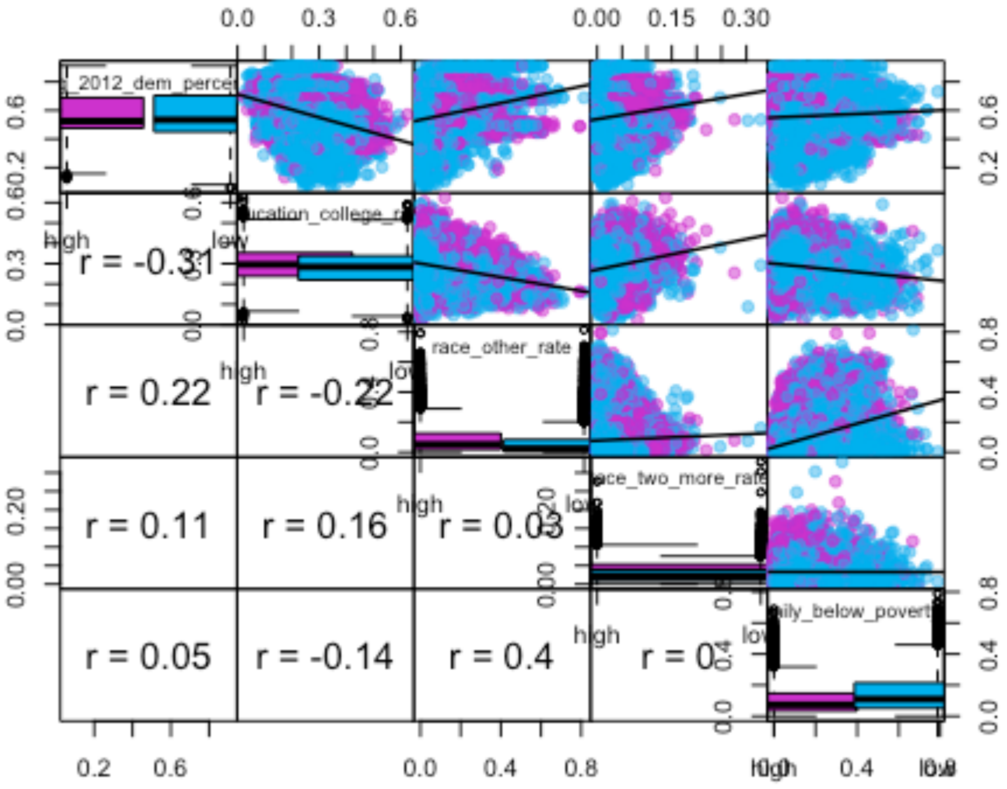
```r
#pairs plot
class <- data$solar_system_count
set <- c(80,78,68,20,22) # select a set of variables for display
colvec <- c("magenta3", "deepskyblue2")
cols <- adjustcolor(colvec[class], 0.5) # set class colors
pairs(data[,set], gap = 0, pch = 19, col = cols)
```



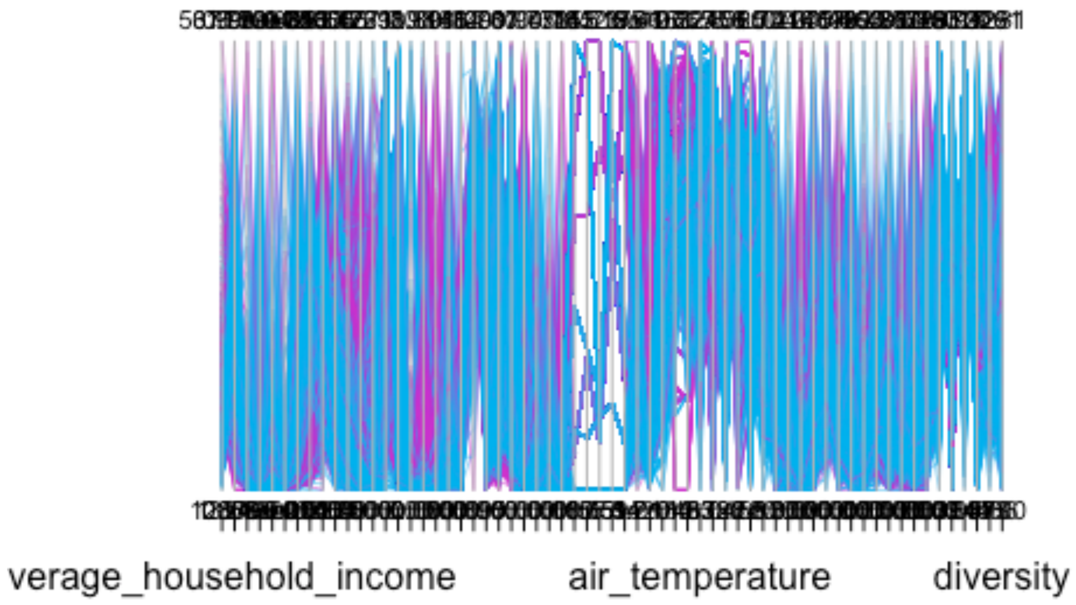
Type2 :

```r
# set a function for calculating regression line
panel.line <- function(x, y){
  fit <- lm(y ~ x)
  points(x, y, pch = 19, col = cols)
  abline(fit, col = 1, lwd = 1.5)
}
# set function for correlation
panel.cor <- function(x, y) {
  r <- round(cor(x, y), 2)
  txt <- paste0("r = ", r)
  loc <- c( mean(range(x)), mean(range(y)) )
  text(loc[1], loc[2], txt, cex = 1.5)
}
# set function for boxplot
```

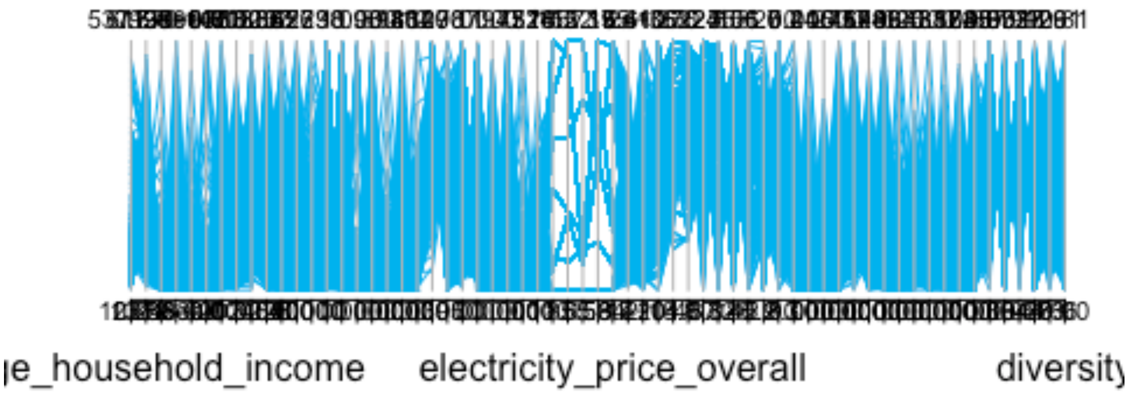
```
panel.box <- function(x, y) {
  r <- range(x) # to center boxplots r <- r + rev(r*0.5)
  boxplot(x ~ class, add = TRUE,at = seq(r[1], r[2], length = 2), col =
colvec)
# to place text in the center
}
# plot
set <- c(60,9,19,20,22)
pairs(num[,set], gap = 0, lower.panel = panel.cor,
upper.panel = panel.line, diag.panel = panel.box)
```



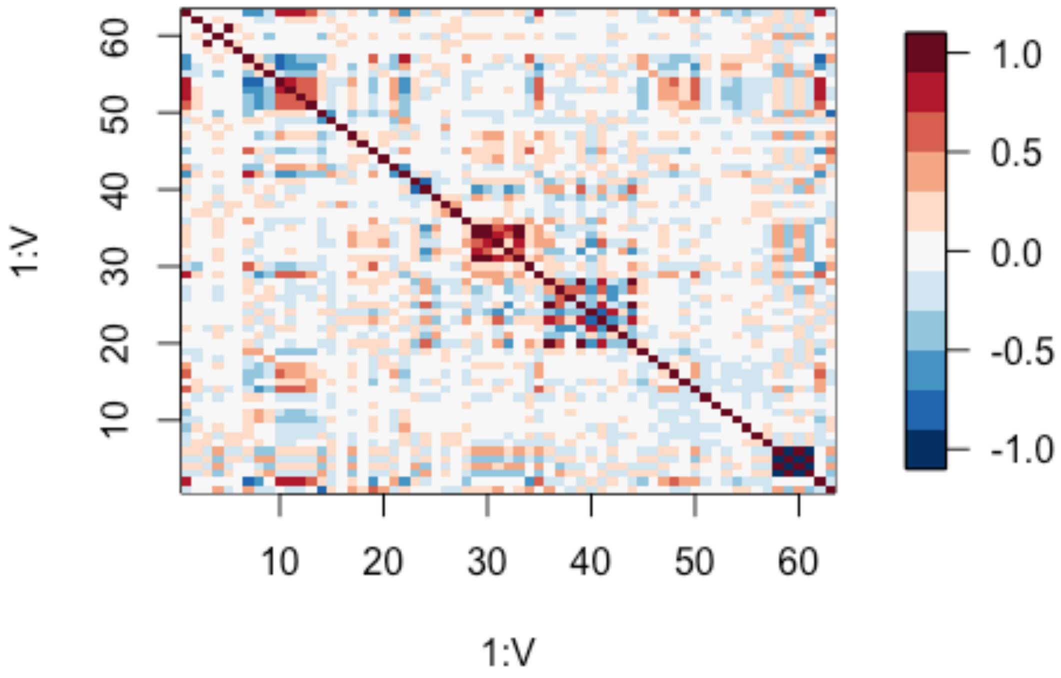
```
#parallel coordniates plot
library(MASS)

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

cols <- adjustcolor(colvec[class], 0.5)
parcoord(num, col = cols, var.label = TRUE)
```



```
cols0 <- colvec[class]
K <- length(colvec) # number of classes
# plot multiple parallel coordinates
par(mfrow = c(K, 1), mar = c(3,2,1,0.5))
for ( k in 1:K ) {
cols <- cols0
cols[cols != colvec[k]] <- adjustcolor("gray", 0)
parcoord(num, col = cols, var.label = TRUE)
}
```



```
#library(superheat)
library(RColorBrewer)
# set color palette
pal <- brewer.pal(11, "BrBG")
#install.packages("fields")
library(fields)

## Loading required package: spam

## Loading required package: dotCall64

## Loading required package: grid

## Spam version 2.5-1 (2019-12-12) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.

##
## Attaching package: 'spam'
## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve

## Loading required package: maps

## See https://github.com/NCAR/Fields for
##  an extensive vignette, other supplements and source code

R <- cor(num) # compute correlation matrix
V <- ncol(R)
# 'blue' denotes negative correlation, 'red' positive correlation
pal <- rev( brewer.pal(11, "RdBu") )
# we need to make sure correct ordering
image.plot(1:V, 1:V, R[,V:1], col = pal, nlevel = 11, zlim = c(-1, 1))
```



CORRELATION AND FEATURE SELECTION:

- Multicollinearity occurs when independent variables in a model are correlated.
- This correlation is a problem because independent variables should be independent.
- If the degree of correlation between variables is high enough, it can cause problems when you fit the model and interpret the results.

- Multicollinearity can also result from the repetition of the same kind of variable.
- Therefore the highly correlated variables are found using “findCorrelation” function in caret package and removed. The final data contains 62 variables.
- Later data is scaled as scaling the data can get rid of many outliers and also Scaling the data often improves the "numerical characteristics" of the data and facilitate convergence of the various algorithms.

```
#install.packages("caret")
library('caret')

## Loading required package: lattice

## Loading required package: ggplot2

df = cor(num)
hc = findCorrelation(df, cutoff=0.7) #generally coefficient for cut-off is
used as 0.7
red_data = num[,-c(hc)]
scaled_data <- scale(red_data)
final_data <- cbind(fac,scaled_data)
```

APPLYING CLASSIFICATIONS AND EVALUATING THE GENERALIZED PERFORMANCE:

- The central goal is to obtain a machine learning model which will perform well at predicting the target variable on new unseen inputs.
- To assess the general performance of a model, we aim to estimate its general prediction error, which is denoted as “generalization error”.
- Install and load packages of multinomial logistic regression, random forest, svm ,classification tree, bagging and boosting.
- Multinomial regression is same as logistic which uses tau=0.5.Support vector machines, used in conjunction with kernels, provide an effective supervised classifier for data with complex structure.
- It is good practice to replicate the procedure of training-validation-testing a number of times to account for sampling variation and assess uncertainty,hence here we replicate the process 100 times.
- Since there are large number of observations , hold out procedure is suitable.
- Each time in the loop, the data is splitted into train-val-test , fit classifiers to training data, and is predicted on val data, only the best classifier is chosen to perform on new test data.

- The accuracy is recorded in the variable out and the highest accuracy every time is recorded in the variable accBest.

```
#install.packages("adabag")
library(kernlab)#svm

##
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
##
##     alpha

library(rpart) #classification tree
library(nnet) #multinomial regression
library(adabag) #bagging

## Loading required package: foreach

## Loading required package: doParallel

## Loading required package: iterators

## Loading required package: parallel

library(randomForest) #random forest

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine

# replicate the process a number of times
R <- 100
out <- matrix(NA, R, 8)
colnames(out) <- c( "val_svm", "val_multinom_log", "val_class_tree",
"random_forest", "bagging", "boosting","best", "test")
out <- as.data.frame(out)

for ( r in 1:R ) {
  # split the data into training, validation and test sets
  N <- nrow(final_data)
  train <- sample(1:N, size = 0.50*N)
```

```r
  val <- sample( setdiff(1:N, train), size = 0.25*N )
  test <- setdiff(1:N, union(train, val))

  # fit classifiers to only the training data
  fitsvm <- ksvm(solar_system_count ~ ., data = final_data[train,])
  fitml <- multinom(solar_system_count ~ ., data = final_data, subset =
train,maxit=300,trace=FALSE)
  fitct<- rpart(solar_system_count ~ ., data = final_data, subset = train)
  fitrf <- randomForest(solar_system_count ~.,data = final_data[train,])
  fitbg <-bagging(solar_system_count~.,data = final_data[train,])
  fitbs <- boosting(solar_system_count~.,data = final_data[train,],coeflearn
="Breiman",boos =FALSE)
# fit on validation data
#
  # classification trees
  predValct <- predict(fitct, type = "class", newdata = final_data[val,])
  tabValct <- table(final_data$solar_system_count[val], predValct)
  accct <- sum(diag(tabValct))/sum(tabValct)
#
#
  # support vector machines
  predValSvm <- predict(fitsvm, newdata = final_data[val,])
  tabValSvm <- table(final_data$solar_system_count[val], predValSvm)
  accsvm <- sum(diag(tabValSvm))/sum(tabValSvm)

  #multinomial regression
  predValml <- predict(fitml, type = "class", newdata = final_data[val,])
  tabValml <- table(final_data$solar_system_count[val], predValml)
  accml <- sum(diag(tabValml))/sum(tabValml)

  # random forest
  predValrf <- predict(fitrf, type = "class", newdata = final_data[val,])
  tabValrf <- table(final_data$solar_system_count[val], predValrf)
  accrf <- sum(diag(tabValrf))/sum(tabValrf)

  # bagging
  predValbg <- predict(fitbg, newdata = final_data[val,])
  tabValbg<- predValbg$confusion
  accbg <- sum(diag(tabValbg))/sum(tabValbg)

  # boosting
  predValbs <- predict(fitbs,newdata = final_data[val,])
  tabValbs <- predValbs$confusion
  accbs <- sum(diag(tabValbs))/sum(tabValbs)

  # compute accuracy
  acc <- c(val_svm=accsvm,
val_multinom_log=accml,val_class_tree=accct,random_forest=accrf,
bagging=accbg,boosting=accbs)
```

```
  out[r,1] <- accsvm
  out[r,2] <- accml
  out[r,3] <- accct
  out[r,4] <- accrf
  out[r,5] <- accbg
  out[r,6] <- accbs
  # use the method that did best on the validation data # to predict the test
data
  best <- names( which.max(acc) )
  switch(best,
    val_svm = {
    predTestsvm <- predict(fitsvm,newdata = final_data[test,])
    tabTestsvm <- table(final_data$solar_system_count[test], predTestsvm)
    accBest <- sum(diag(tabTestsvm))/sum(tabTestsvm)
    },
    val_multinom_log= {
     predTestml <- predict(fitml, type = "class", newdata =
final_data[test,])
     tabTestml<- table(final_data$solar_system_count[test], predTestml)
     accBest <- sum(diag(tabTestml))/sum(tabTestml)
    },
    val_class_tree={
    predTestrf <- predict(fitrf, type = "class", newdata = final_data[test,])
    tabTestrf <- table(final_data$solar_system_count[test], predTestrf)
    accBest <- sum(diag(tabTestrf))/sum(tabTestrf)
    },
    random_forest ={
      predtestrf <- predict(fitrf, type = "class", newdata =
final_data[test,])
  tabtestrf <- table(final_data$solar_system_count[test], predtestrf)
  accBest <- sum(diag(tabtestrf))/sum(tabtestrf)},

    bagging ={
      predtestbg <- predict(fitbg,newdata = final_data[test,])
  tabtestbg<- predtestbg$confusion
  accBest <- sum(diag(tabtestbg))/sum(tabtestbg)},

    boosting={
      predtestbs <- predict(fitbs, newdata = final_data[test,])
  tabtestbs <- predtestbs$confusion
  accBest<- sum(diag(tabtestbs))/sum(tabtestbs)})

out[r,7] <- best
out[r,8] <- accBest
}
```

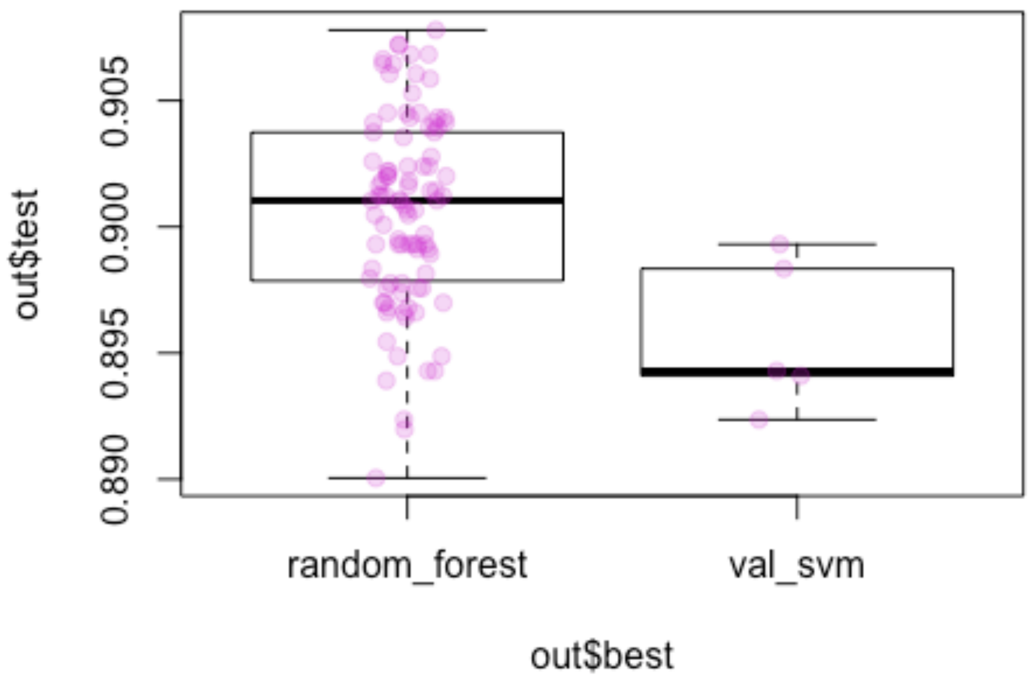
RESULTS AND DISCUSSION:

when table function is applied to 7th column of out variable, it gives the total number of times a classifer was chosen as best out of 100. From the output we can see that random forest classification method was chosen 95 times whereas svm was chosen 5 times. Random forest classification method was chosen majority of the times , hence Random forest is chosen as the best classifier for the Deepsolar database.

```
table(out[,7])

##
## random_forest       val_svm
##            95             5
```

- The summary of best chosen classifier is produced below. It is the summary of the accuracy recorded of the best classifier after each of the 100 repetitions.
- In the random forest's summary we can observe that the minimum accuracy recorded was 89% while the maximum recorded was 90.78%. In most of the repetitions the typical values of the accuracy of random forest classifier lies between the range 89%-90.37%. The average accuracy is 90%.
- In the svm's summary we can observe that the minimum accuracy recorded was 89.24% while the maximum recorded was 89.99%. In most of the repetitions the typical values of the accuracy of random forest classifier lies between the range 89.41%-89.83%. The average accuracy is 89.57%.
- Comparing the two summaries we can conclude that random forest classifier can be chosen the best.

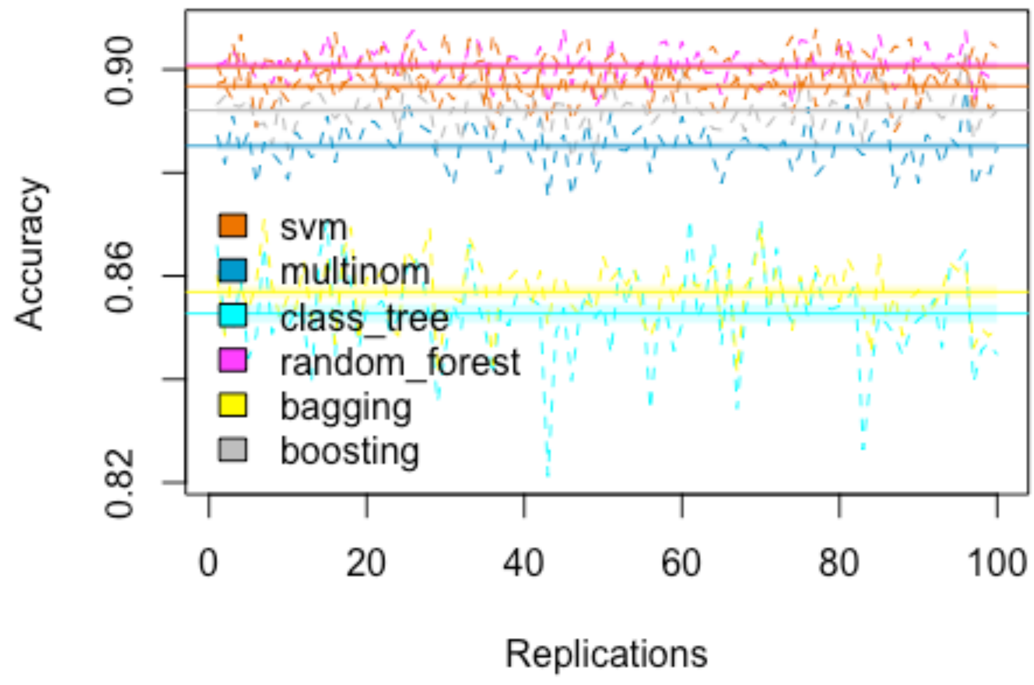
```
#summary
tapply(out[,8],out[,7], summary)

## $random_forest
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8900  0.8979  0.9010  0.9007  0.9037  0.9078
##
## $val_svm
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8924  0.8941  0.8943  0.8957  0.8983  0.8993

# plotting
boxplot(out$test ~ out$best)
stripchart(out$test ~ out$best, add = TRUE, vertical = TRUE,
method = "jitter", pch = 19, col = adjustcolor("magenta3", 0.2))
```



INTERPRETING THE BOXPLOT:

The boxplot gives the distribution of the accuracy points of the best chosen classifier. From the plot we can infer that median of the random_forest boxplot is higher than that of svm. Also the whiskers indicating the maximum accuracy is higher for random forest.

```r
out <- out[,-7]
out<- as.matrix.data.frame(out)
meanAcc <- colMeans(out)
sdAcc <-apply(out,2, sd)/sqrt(R)

# plot
matplot(out,type ="l",lty=2,col =c("darkorange2","deepskyblue3",5,6,7,8),xlab
="Replications",ylab ="Accuracy")
## add confidence interval
bounds1 <-rep(c(meanAcc[1]-2*sdAcc[1], meanAcc[1]+2*sdAcc[1]),each =R )
bounds2 <-rep(c(meanAcc[2]-2*sdAcc[2], meanAcc[2]+2*sdAcc[2]),each =R )
bounds3 <-rep(c(meanAcc[3]-2*sdAcc[3], meanAcc[3]+2*sdAcc[3]),each =R )
bounds4 <-rep(c(meanAcc[4]-2*sdAcc[4], meanAcc[4]+2*sdAcc[4]),each =R )
bounds5 <-rep(c(meanAcc[5]-2*sdAcc[5], meanAcc[5]+2*sdAcc[5]),each =R )
bounds6 <-rep(c(meanAcc[6]-2*sdAcc[6], meanAcc[6]+2*sdAcc[6]),each =R )
polygon(c(1:R, R:1), bounds1,col =adjustcolor("darkorange2",0.2),border
=FALSE)
polygon(c(1:R, R:1), bounds2,col =adjustcolor("deepskyblue3",0.2),border
=FALSE)
polygon(c(1:R, R:1), bounds3,col =adjustcolor(5,0.2),border =FALSE)
polygon(c(1:R, R:1), bounds4,col =adjustcolor(6,0.2),border =FALSE)
polygon(c(1:R, R:1), bounds5,col =adjustcolor(7,0.2),border =FALSE)
polygon(c(1:R, R:1), bounds6,col =adjustcolor(8,0.2),border =FALSE)
## add estimated mean line
abline(h =meanAcc,col =c("darkorange2","deepskyblue3",5,6,7,8))

## add legend
legend("bottomleft",fill =c("darkorange2","deepskyblue3",5,6,7,8),legend
=c("svm","multinom","class_tree","random_forest","bagging","boosting"),bty
="n")
```



PLOT TO COMPARE ACCURACY OF VARIOUS CLASSIFIERS:

We can calculate the mean classification accuracy on all the replications and produce a plot to visually compare the estimated accuracy of the two classifiers. We can interpret the above plot as – Classification tree method has the least accuracy, the next least is bagging.

- The topmost classifier is random forest, svm comes second followed by boosting and multinomial regression.
- Bagging and Random forest are ensemble methods usually based on classification trees. One disadvantage of classification tree is it suffers from high variance, hence bagging can be replaced.
- Random forest uses classification trees as well as bootstrapping extensively. Boosting is a powerful procedure that combines the outputs of many weak classifiers to produce a "strong" classifier.

CONCLUSION

The complete data analysis is performed on DeepSolar dataset , starting with preprocessing the data which includes NA analysis, duplicates removal which shows that there is no NAs and duplicates in the dataset. From the EDA plots we can observe that there exist linear relationship between many variables , variables are weakly correlated which tells its importance and for few there are too many outliers due to which data is scaled. Some highly correlated variables are removed to avoid multicollinearity. The data is fitted to several classification models and this process is repeated for 100 times , and hold out approach is used as cross validation method. In the end we can see that random forest was chosen as the best classifiers majority of the times.

Therefore we can conclude that random forest is the best classifier for DeepSolar data which predicts the variable solar_system_count with the highest accuracy of 90.78% among many other classifiers.

REFERENCE

1. Package "dplyr" : https://www.rdocumentation.org/packages/dplyr/versions/0.7.8
2. Package "caret" : http://topepo.github.io/caret/index.html
3. Variable selection : https://www.statisticssolutions.com/multicollinearity/