Prediction of a Solar Power System Coverage of a tile

Tanay Sawant 19203264

21/04/2020

Abstract:

In the following report we are going to talk about how we predicted the solar power system coverage of a tile given the collection of predictor features where the target variable is **solar_system_count.** We use different types of supervised classification algorithms, compare them and select the best model to be used for prediction. Based on the model which is selected, we do our predictions and arrive at our conclusion.

Introduction:

The data is a subset of the Deep Solar database, a solar installation data base for the US., built by extracting information from satellite images. Photovoltaic panel installations are identified from over one billion image tiles covering all urban areas as well as locations in the US by means of an advanced machine learning framework. Each image tile records the amount of solar panel systems (in terms of panel surface and number of solar panels) and is complemented with features describing social, economic, environmental, geographical, and meteorological aspects. As such, the database can be employed to relate key environmental, weather and socioeconomic factors with the adoption of solar photovoltaics energy production.

The dataset data_project_deepsolar.csv contains a subset of the DeepSolar database. Each row of the dataset is a "tile" of interest, that is an area corresponding to a detected solar power system, constituted by a set of solar panels on top of a building or at a single location such as a solar farm. For each system, a collection of features record social, economic, environmental, geographical, and meteorological aspects of the tile (area) in which the system has been detected.

What are we trying to predict? (Research Question.)

We need to predict the solar power system coverage of a tile. It takes outcome low if the tile has a low number of solar power systems (less than or equal to 10), while it takes outcome high if the tile has a large number of solar power systems (more than 10).

With this prediction, we will be able to answer if and when we want to install the solar panel in future (which is highly likely) as the demand for the utilization of natural resources is increasing and cost effectiveness and helping the environment are it's additional benefits, we can help the purchaser with the adequate tile requirement based on the predictor variables.

What type of problem are we dealing with?

Our target variable “Solar_system_count” is a binary variable and we are dealing with a classification problem with multiple features.

What type of data do we have?

We have a csv file “data_project_deepsolar.csv” which contains the dataset to be used for the project.

Methods:

We now dive deep into the process we went through for the selection of the best model.

Import the dataset:

We import the dataset “data_deepsolar.csv” and observe the data set for some EDA. With str() and dim() we have our first look at the dataset.

```
data=read.csv("data_project_deepsolar.csv",header = T)
str(data)   # to study how is our data

## 'data.frame':    20736 obs. of  81 variables:
##  $ solar_system_count                   : Factor w/ 2 levels "high","low
": 2 2 2 1 2 1 1 1 1 2 ...
##  $ state                                : Factor w/ 7 levels "az","ca","
il",..: 6 6 6 6 6 6 6 6 6 6 ...
##  $ average_household_income             : num  108955 122821 39522 48647
129045 ...
##  $ employed                             : int  2168 1727 1279 880 2491 2
196 3241 1890 3077 935 ...
##  $ gini_index                           : num  0.407 0.443 0.534 0.393 0
.509 ...
##  $ land_area                            : num  10.584 47.047 0.4 0.897 2
4.671 ...
##  $ per_capita_income                    : int  39801 53892 7160 8461 500
92 18426 36784 35642 38457 8324 ...
##  $ population                           : int  4315 3596 6727 4423 5073
5226 7078 3977 6546 4644 ...
##  $ population_density                   : num  407.7 76.4 16819.4 4930.2
205.6 ...
##  $ total_area                           : num  11.206 49.346 0.406 0.902
24.826 ...
##  $ unemployed                           : int  206 83 68 49 181 276 193
123 339 35 ...
##  $ water_area                           : num  0.62238 2.29924 0.00562 0
.00522 0.15451 ...
##  $ education_less_than_high_school_rate   : num  0.0648 0.0559 0.3566 0.32
52 0.0593 ...
##  $ education_high_school_graduate_rate  : num  0.266 0.179 0.445 0.451 0
.216 ...
##  $ education_college_rate               : num  0.307 0.235 0.158 0.163 0
.278 ...
##  $ education_bachelor_rate              : num  0.2317 0.2858 0.0264 0.02
```

```
3 0.2213 ...
##  $ education_master_rate                    : num  0.1041 0.1772 0.014 0.019
1 0.1641 ...
##  $ education_professional_school_rate     : num  0.0149 0.03 0 0.0184 0.05
17 ...
##  $ education_doctoral_rate                : num  0.0116 0.037 0 0 0.009 ..
.
##  $ race_white_rate                        : num  0.847 0.88 0.995 1 0.908
...
##  $ race_black_africa_rate                 : num  0.0633 0.0192 0 0 0.0755
...
##  $ race_indian_alaska_rate                : num  0 0 0.00535 0 0 ...
##  $ race_asian_rate                        : num  0.0153 0.04922 0 0 0.0019
7 ...
##  $ race_islander_rate                     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ race_other_rate                        : num  0.0311 0.0339 0 0 0 ...
##  $ race_two_more_rate                     : num  0.0433 0.0172 0 0 0.015 .
..
##  $ employ_rate                            : num  0.913 0.954 0.95 0.947 0.
932 ...
##  $ poverty_family_below_poverty_level_rate: num  0.02194 0.00552 0.57651 0
.34301 0.01185 ...
##  $ heating_fuel_gas_rate                  : num  0.632 0.351 0.88 0.924 0.
414 ...
##  $ heating_fuel_electricity_rate          : num  0.0324 0.141 0.1072 0.028
6 0.0466 ...
##  $ heating_fuel_fuel_oil_kerosene_rate    : num  0.3217 0.4932 0 0.0469 0.
4941 ...
##  $ heating_fuel_coal_coke_rate            : num  0.01165 0.00451 0.00824 0
0.0238 ...
##  $ heating_fuel_other_rate                : num  0 0.0103 0.00495 0 0.0217
3 ...
##  $ heating_fuel_none_rate                 : num  0.00259 0 0 0 0 ...
##  $ electricity_price_residential          : num  18.5 18.5 18.5 18.5 18.5
...
##  $ electricity_price_commercial           : num  15.3 15.3 15.3 15.3 15.3
...
##  $ electricity_price_industrial           : num  6.31 6.31 6.31 6.31 6.31
6.31 6.31 6.31 6.31 6.31 ...
##  $ electricity_price_transportation       : num  12.9 12.9 12.9 12.9 12.9
...
##  $ electricity_price_overall              : num  15.3 15.3 15.3 15.3 15.3
...
##  $ electricity_consume_residential        : int  601 601 601 601 601 601 6
01 601 601 601 ...
##  $ electricity_consume_commercial         : int  6082 6082 6082 6082 6082
6082 6082 6082 6082 6082 ...
##  $ electricity_consume_industrial         : int  197457 197457 197457 1974
57 197457 197457 197457 197457 197457 197457 ...
##  $ electricity_consume_total              : int  204140 204140 204140 2041
```

```
40 204140 204140 204140 204140 204140 204140 ...
##  $ household_count                         : int  1545 1553 1213 768 1933 2
067 2824 1217 2660 814 ...
##  $ average_household_size                  : num  2.79 2.29 5.55 5.76 2.61
2.39 2.46 3.06 2.46 5.68 ...
##  $ housing_unit_count                      : int  1585 1713 1261 827 2179 2
436 2996 1286 3232 886 ...
##  $ housing_unit_median_value               : num  282400 418100 307400 4000
00 366700 ...
##  $ elevation                               : int  260 260 260 260 260 260 2
60 260 260 260 ...
##  $ heating_design_temperature              : num  -8.91 -8.91 -8.91 -8.91 -
8.91 -8.91 -8.91 -8.91 -8.91 -8.91 ...
##  $ cooling_design_temperature              : num  27.6 27.6 27.6 27.6 27.6
...
##  $ earth_temperature_amplitude             : num  20.3 20.3 20.3 20.3 20.3
...
##  $ frost_days                              : int  117 117 117 117 117 117 1
17 117 117 117 ...
##  $ air_temperature                         : num  9.5 9.5 9.5 9.5 9.5 9.5 9
.5 9.5 9.5 9.5 ...
##  $ relative_humidity                       : num  0.663 0.663 0.663 0.663 0
.663 0.663 0.663 0.663 0.663 0.663 ...
##  $ daily_solar_radiation                   : num  3.69 3.69 3.69 3.69 3.69
3.69 3.69 3.69 3.69 3.69 ...
##  $ atmospheric_pressure                    : num  98.6 98.6 98.6 98.6 98.6
98.6 98.6 98.6 98.6 98.6 ...
##  $ wind_speed                              : num  4.3 4.3 4.3 4.3 4.3 4.3 4
.3 4.3 4.3 4.3 ...
##  $ earth_temperature                       : num  9.3 9.3 9.3 9.3 9.3 9.3 9
.3 9.3 9.3 9.3 ...
##  $ heating_degree_days                     : int  3458 3458 3458 3458 3458
3458 3458 3458 3458 3458 ...
##  $ cooling_degree_days                     : int  1594 1594 1594 1594 1594
1594 1594 1594 1594 1594 ...
##  $ occupation_construction_rate            : num  0.065 0.0643 0.0117 0.025
0.104 ...
##  $ occupation_public_rate                  : num  0.0637 0.0596 0.0149 0 0.
0301 ...
##  $ occupation_information_rate             : num  0.05812 0.0139 0.01251 0
0.00923 ...
##  $ occupation_finance_rate                 : num  0.0452 0.1274 0.0547 0.07
61 0.0454 ...
##  $ occupation_education_rate               : num  0.226 0.195 0.407 0.355 0
.286 ...
##  $ occupation_administrative_rate          : num  0.0969 0.1662 0.1486 0.06
48 0.1441 ...
##  $ occupation_manufacturing_rate           : num  0.083 0.0979 0.0758 0.077
3 0.0285 ...
##  $ occupation_wholesale_rate               : num  0.0323 0.0266 0.0586 0.03
```

```
75 0.0594 ...
##  $ occupation_retail_rate                 : num  0.14 0.129 0.154 0.198 0.
145 ...
##  $ occupation_transportation_rate         : num  0.04889 0.03474 0.00391 0
.07159 0.0289 ...
##  $ occupation_arts_rate                   : num  0.053 0.0782 0.0141 0.089
8 0.0879 ...
##  $ occupation_agriculture_rate            : num  0.0304 0 0 0 0 ...
##  $ occupancy_vacant_rate                  : num  0.0252 0.0934 0.0381 0.07
13 0.1129 ...
##  $ voting_2016_dem_percentage             : num  0.448 0.448 0.448 0.448 0
.448 ...
##  $ voting_2016_gop_percentage             : num  0.512 0.512 0.512 0.512 0
.512 ...
##  $ voting_2016_dem_win                    : Factor w/ 2 levels "False","Tr
ue": 1 1 1 1 1 1 1 1 1 1 ...
##  $ voting_2012_dem_percentage             : num  0.519 0.519 0.519 0.519 0
.519 0.519 0.519 0.519 0.519 0.519 ...
##  $ voting_2012_gop_percentage             : num  0.467 0.467 0.467 0.467 0
.467 0.467 0.467 0.467 0.467 0.467 ...
##  $ voting_2012_dem_win                    : Factor w/ 2 levels "False","Tr
ue": 2 2 2 2 2 2 2 2 2 2 ...
##  $ number_of_years_of_education           : num  14.1 15.1 11.1 11.4 14.7
...
##  $ diversity                              : num  0.2754 0.2206 0.0106 0 0.
1704 ...
```

```r
dim(data)
```

```
## [1] 20736    81
```

We observe that the dimension of the dataset is 20736 * 81. Our dataset is tidy, with each row being an observation and each column being a feature. We check the data types of the variables and find them to be okay. We also observe that our response variable is of the type factor, with expected two levels (high) and (low).

To check for class imbalance in our target variable:

Our target variable is class, with two levels: high and low. High indicates if the tile has a large number of solar power systems (more than 10) and low indicates if the tile has a low number of solar power systems (less than or equal to 10).

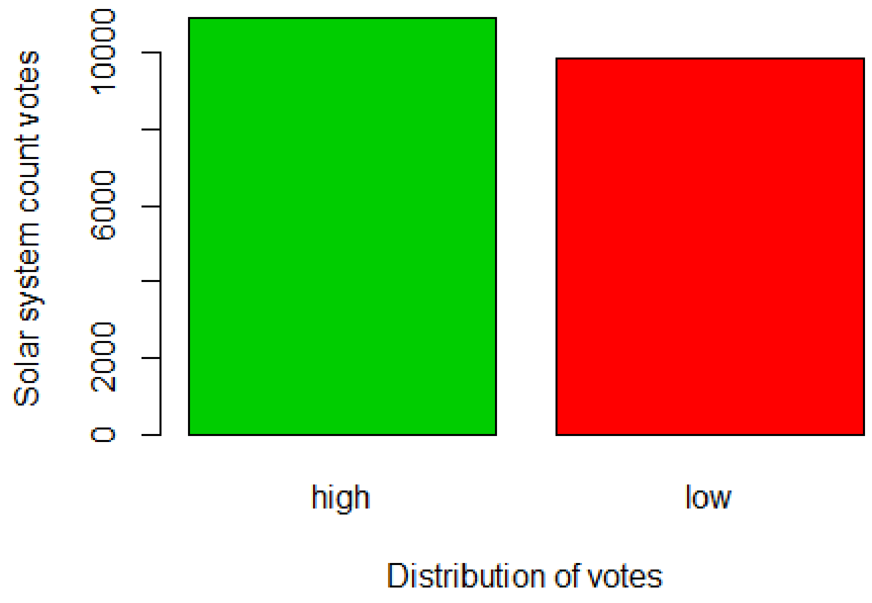
```r
# reading more about our target variable:

summary(data$solar_system_count)
```

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

How many high and low in the target variable?

We check the summary of the target variable to find out the number of high and low in our dataset and we get that there are high and low.
We even plot the distribution of the votes for a graphical visualization.

```
plot(data$solar_system_count,ylab="Solar system count
votes",xlab="Distribution of votes",col=c(3:2))
```



Their proportions:

In order to check if there is class imbalance, we check the proportions of high and low using the function prop.table.

```
prop.table(table(data$solar_system_count))

##
##      high       low
## 0.5256559 0.4743441
```

We find that our data is neatly divided and there is no class imbalance. This is an indication of our target variable evenly distributed.

Missing values?

We use the function is.na and complete. cases to be check if we have any missing observation in our data.

```
#to check for missing data:

sum(is.na(data))

## [1] 0
```

```r
sum(complete.cases(data))

## [1] 20736
```

We get the value of is.na to be 0 and complete. cases to be 20736 (all the observations covered) and with this, we are sure of no missing observations in our dataset.

Removing categorical data:

We remove the categorical columns (4 including the target variable which can be observed in the structure of the data.) from our data as we are going to check for multi-collinearity in our data. For that, we need all the numeric columns in the dataset. We store this under a new dataframe.

```r
#removing the categorical variables:

data1<-data[-c(1,2,76,79)]
```

Checking for multicollinearity:

Last step of our EDA phase is to check for multicollinearity. We will do some analysis of how many bivariate relations have high correlation and if it's a significant high proportion:
As our new data frame has 77 features which are numeric, our correlation matrix is of 77*77.

```r
#to check for multicolinearity:

sum((cor(data1) > 0.5 | cor(data1) < -0.5) & cor(data1) < 1) / (77*77)

## [1] 0.05835723

sum((cor(data1) > 0.7 | cor(data1) < -0.7) & cor(data1) < 1) / (77*77)

## [1] 0.02597403

sum((cor(data1) > 0.9 | cor(data1) < -0.9) & cor(data1) < 1) / (77*77)

## [1] 0.0094451

#removing the highly correlated variables:

tmp <- cor(data1)
tmp[!lower.tri(tmp)] <- 0

data.new <- data1[,!apply(tmp,2,function(x) any(abs(x) > 0.9))]

data.new<-scale(data.new)
```

We find out that out of 5929 possible variable correlations:

- 5.8% present scores higher than 0.5
- 2.5% present scores higher than 0.7
- 0.94% present scores higher than 0.9

With the help of above possible correlations, we decide to remove the variables having correlation 0.9 (positive or negative) with each other to remove the effect of multicollinearity. After removing the variable that are highly correlated to each other, our dataframe gets reduced to 61 variables. We store this reduced dataframe under a new variable. We scale the reduced dataset at this point.

Adding the target variable:

We add the target variable “solar_system_count” to this reduced dataframe and make it ready for our model selection process.

```r
# final data frame:

mydata<-cbind(data$solar_system_count,data.new)
colnames(mydata)[1]<-("solar_system_count")
```

Modelling:

We now have the dataframe **“mydata”** ready for modelling. We have tested for missing values, removed collinear features and done the EDA for the data. The supervised classification algorithms covered by us during the course of time are:

- Classification Trees.
- Logistic/ Multinomial Regression
- Bagging
- Random Forest
- Boosting
- Support Vector Machine

Our problem being of Binary classification, the two algorithms (logistic and svm) get a direct heads up for model selection process as those are considered to be the ideal ones. We go ahead with RandomForest as our third choice.

With thorough considerations, testing and running the models for fewer repetitions, we find out that for our given dataset, Classification Trees and Bagging (for which, RF is the extension of bagging and we have decided to select RF being another reason) do not give a good accuracy rate for validation dataset and hence we do not consider these two methods for model selection process and move forward with the remaining 4 models.

In order to reduce the time complexity, we use parallel programming for the following chunk of code.

Parallelization is done with the two libraries “doParallel” and “foreach” where we load the required packages in the foreach statements and rbind all the results in the variable res. We detect our number of cores of our processor, and keep one aside for the machine to do different tasks and engage the remaining cores for R, which in result improve the speed of calculations.

About the model:

1. The model we have created is divided into 3 parts: training, validation and testing.

2. With the commented code, we can observe that the number of replications we are doing is set to 100 and the data matrix **“dat”** is used as a training and validation dataset while the data matrix **“dat_test”** is going to be used for prediction once the best model is selected.
3. We go ahead with a **75-25 splitting** of the data and set a seed to replicate the results. We store the validation accuracy of the models in the “acc” matrix.
4. We fit the 4 models using their functions from respective libraries and based on the best classifier for each repetition, we use that model for prediction using the test data and we get our “BEST MODEL” for each repetition with it’s “BEST ACCURACY” in columns 5 and 6 out the matrix out.
5. Here the matrix out is the one to constantly monitor, which will be a matrix of 100 rows and 6 columns where the columns are named as "val_log", "val_rf", "val_svm", "val_boost", "best","test", where the first 4 columns represent the validation accuracy for each repetition and the column 5 represents the best model selected for that particular repetition and we store the accuracy of the best model in column 6.

```
library(doParallel)     #To enable parallel programming
library(foreach)        #Used for for loop in parallel programming.


#Detecting the number of processors available.
numCores <- detectCores()-1

numCores

## [1] 7

registerDoParallel(numCores)

#Assigning 1 less core which is to be used for other processes in the system.

# model:

set.seed(1234)

keep=sample(1:nrow(mydata),size=0.75*nrow(mydata))  #dividing the data for tr
aining,validation, testing.
test=setdiff(1:nrow(mydata),keep)
dat=mydata[keep,]       #training and validation data set.
dat_test=mydata[test,]  #testing data set.

R=100  #No.of replicates.
out <-matrix(NA, R, 6)
colnames(out) <- c("val_log","val_rf","val_svm", "val_boost","best","test")
out <- as.data.frame(out)

N=nrow(dat)
```

```r
acc=matrix(NA,R,4)

res<-foreach(r=1:R,.packages = c("randomForest","adabag","kernlab","nnet")) %
dopar%
  {

  train=sample(1:N,size=0.75*N)
  val=setdiff(1:N,train)

  #Multinomial Regression: (fitting, predicting and storing the accuracy.)

  fitmultinom=multinom(solar_system_count~.,data=dat,subset=train,trace=F)
  predmultinom= predict(fitmultinom,newdata=dat[val,])
  tab1=table(predmultinom,dat$solar_system_count[val])
  acclog =sum(diag(tab1))/sum(tab1)

  #Random Forest Classification: (fitting, predicting and storing the accurac
y.)

  fitrandom=randomForest(solar_system_count~.,data=dat,subset=train,importanc
e=T)
  predrandom=predict(fitrandom,newdata=dat[val,])
  tab2=table(predrandom,dat$solar_system_count[val])
  accrf=sum(diag(tab2))/sum(tab2)

  #Support vector machine: (fitting, predicting and storing the accuracy.)

  fitsvm=ksvm(solar_system_count~.,data=dat[train,])
  predsvm=predict(fitsvm,newdata=dat[val,])
  tab3=table(predsvm,dat$solar_system_count[val])
  accsvm=sum(diag(tab3))/sum(tab3)

  #Boosting: (fitting, predicting and storing the accuracy.)

  fitboost<-boosting(solar_system_count~.,data =dat[train,],coeflearn ="Breim
an",boos =FALSE)
  predboost=predict(fitboost,newdata=dat[val,])
  tab4=predboost$confusion
  accboost=sum(diag(tab4))/sum(tab4)

  acc <-c(logistic = acclog, rf = accrf , svm =accsvm, boost = accboost)
  out[r,1] <-acclog
  out[r,2] <-accrf
  out[r,3] <-accsvm
  out[r,4] <-accboost

  #selecting the best model:
  i = names(which.max(acc))
  if(i=="logistic"){
    predTestLog <-predict(fitmultinom,type ="class",newdata =dat_test)
    tabTestLog <-table(predTestLog,dat_test$solar_system_count)
    accbest <-sum(diag(tabTestLog))/sum(tabTestLog)
    }

  if(i=="rf"){
    predTestrf <-predict(fitrandom,type ="class",newdata =dat_test)
    tabTestrf <-table(predTestrf,dat_test$solar_system_count)
    accbest<-sum(diag(tabTestrf))/sum(tabTestrf)
  }
  if(i=="svm"){
    predTestSvm <-predict(fitsvm,newdata =dat_test)
    tabTestSvm <-table(predTestSvm,dat_test$solar_system_count)
    accbest<-sum(diag(tabTestSvm))/sum(tabTestSvm)
  }
  if(i=="boost"){
    predTestboost<-predict(fitboost,newdata =dat_test)
    tabTestboost <-table(predTestboost,dat_test$solar_system_count)
    accbest<-sum(diag(tabTestboost))/sum(tabTestboost)
  }

  out[r,5] <-i
  out[r,6] <-accbest

  return(out)

  }
```

Results and Discussions:

The main tasks for the given project were:

1.) Finding out the best supervised classification model which we have learned till now in the course.

2.) Using that best model for prediction of the solar power system coverage of a tile given the collection of predictor features where the target variable is solar_system_count.

Let us first understand what **classification accuracy** is:

It is the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

This accuracy is obtained using the confusion matrix (we use the table function to create that matrix in our code) and stored in the column 6 of our out matrix.

Extracting the results from matrix out:

Now that we have done all the work of identifying the best model, using it for prediction for each model, we extract the results stored in our matrix out for it's interpretation.

```
# extract results
for (i in 1:R) {
  out[i, ] <- res[[i]][i, ]
}

head(out,3)

##     val_log    val_rf   val_svm val_boost best      test
## 1 0.8886317 0.9071502 0.9027778 0.8989198   rf 0.9012346
## 2 0.8814300 0.9014918 0.8991770 0.8891461   rf 0.9021991
## 3 0.8886317 0.9009774 0.9014918 0.8984053  svm 0.8939043
```

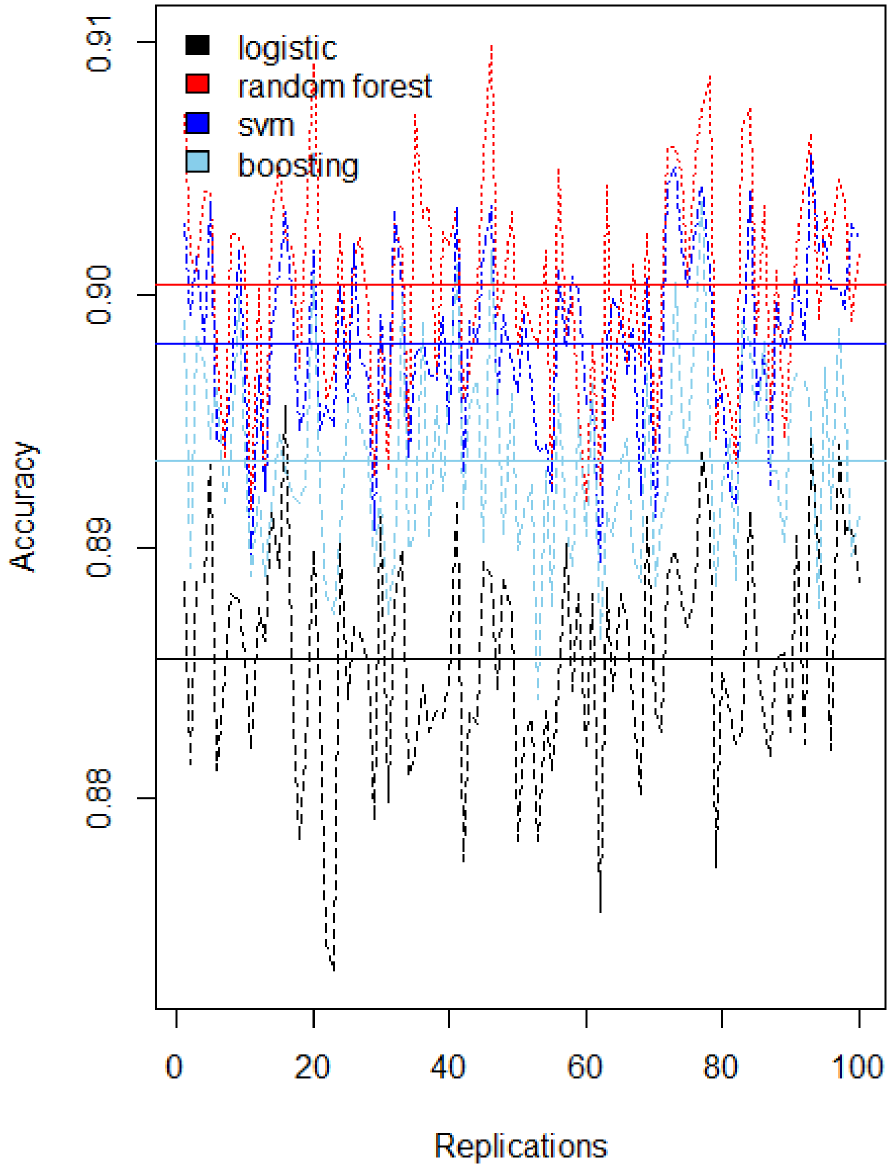
From the above output, we just print the 3 rows i.e. the first 3 repetitions and observe that for the first 3 reps, we have our best model selected as   random forest classification twice and support vector machine once with their prediction accuracies represented in the last column.

How did all the 4 models perform?

With this chunk of code, we try and find out the performance of all the 4 models used for selection of the best model. Here we compare their validation accuracies on which the selection for the best model is done. The flatline in the graph represents the estimated mean validation accuracy for each model, we get the accuracy for 100 repetitions for each model which gives us clear indication how the model has performed over 100 reps.

```
avg <- t(colMeans(as.matrix(out[,c(1,2,3,4)])))
avg

##        val_log    val_rf   val_svm val_boost
## [1,] 0.8855453 0.9003832 0.8981147 0.8934053

meanAcc<-colMeans(avg)
meanAcc

##   val_log    val_rf   val_svm val_boost
## 0.8855453 0.9003832 0.8981147 0.8934053

sdAcc <- apply(out[,c(1,2,3,4)],2,sd)/sqrt(R)
sdAcc

##      val_log       val_rf      val_svm    val_boost
## 0.0004514611 0.0004376124 0.0003711874 0.0003911830

matplot(out[,c(1,2,3,4)], type = "l", lty = c(2,3,4), col = c("black", "red",
"blue","skyblue"), xlab = "Replications", ylab = "Accuracy")

abline(h = meanAcc, col = c("black", "red","blue","skyblue"))
```

```r
legend("topleft", fill = c("black", "red","blue","skyblue"), legend = c("logi
stic", "random forest","svm","boosting"), bty = "n")
```



We get a clear picture with the graph that logistic regression and boosting both have poor performance on the dataset compared to that of Random Forest Classification and Support Vector Machine. This accuracy rate is suggesting that our model is training properly. The mean accuracy rates for the 4 models are as follows (higher to lower):

1. **Random Forest Classification: 90.03%**
2. **Support Vector Machine: 89.81%**

3. **Boosting: 89.34%**
4. **logistic regression is 88.55%**

All 4 models perform at a similar accuracy rate with the range for validation accuracy overall being just under below 0.88 to just below 0.91.

Who has the upper hand? Which is the best classifier? How many times?

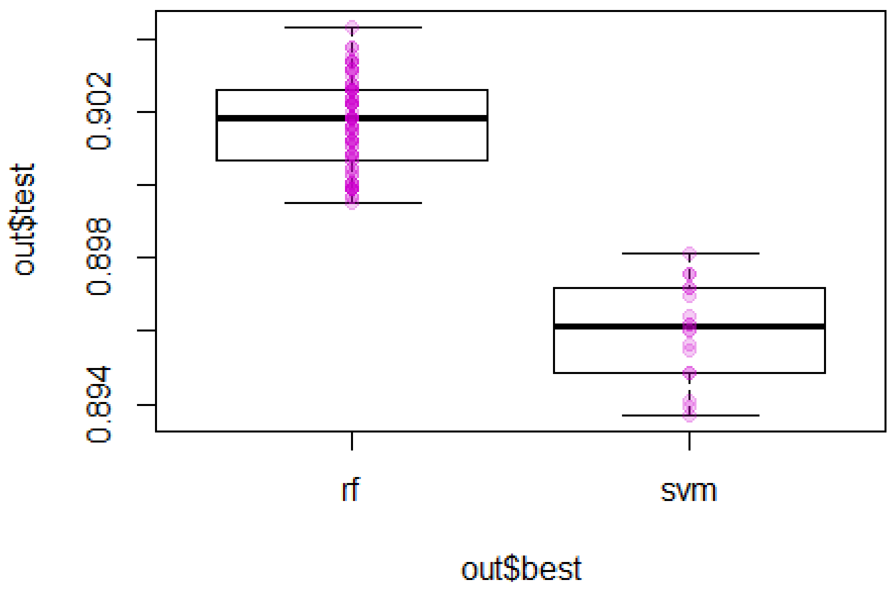
```
# how many times each classifier was selected
table(out[,5])/R

##
##   rf  svm
## 0.82 0.18
```

Out of the 100 reps, we observe that the best models selected each time are Random Forest Classification and Support Vector Machine with **RF dominating most of the times (82 out of 100)** indicating that the best suited model for our dataset is Random Forest Classification.

Comparing the two best classifiers:

With the chunck of code below, we compare the two best classifiers obtained.

```
# summary test accuracy of the selected classifiers
tapply(out[,6], out[,5], summary)

## $rf
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8995  0.9007  0.9018  0.9016  0.9026  0.9043
##
## $svm
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8937  0.8950  0.8961  0.8960  0.8971  0.8981

boxplot(out$test~out$best)
stripchart(out$test~out$best,add =TRUE,vertical =TRUE,pch =19,col =adjustcolo
r("magenta3",0.2))
```



We observe that using the summary statistics and the boxplot, there is no much difference in these two methods for the best minimum, mean, maximum accuracies with random forest just tipping over support vector machine most of the times.

The maximum accuracy obtained for random forest classification is 90.43% while that for Support Vector Machine is 89.81%.

About Random Forest Classifier:

Random forests extend bagging by considering random splits, and this to further make the trees more diverse. Here we will use random forests to perform supervised classification on the training data.

```
library(randomForest)

fitrandom=randomForest(solar_system_count~.,data=dat,importance=T)
```

Extracting the classification table:

```
predrandom=predict(fitrandom,newdata=dat_test)
tabTestrf <-table(predrandom,dat_test$solar_system_count)
tabTestrf

##
## predrandom high  low
##       high 2449  257
##       low   249 2229
```

With the above table we can see how well the classification of high to high and low to low is occurring. We correctly classify 2449 and 2229 high-high and low-low with an accuracy rate 0f 90.43%

Conclusion:

In order to answer to the main research question, we summariese our findings:

Throughout the project, we did the EDA to find out how our dataset is and reduced its dimensionality by removing the highly correlated variables.

We discussed about the various methods learned throughout the course and with proper reasoning, we decided to conduct our model selection process on the 4 methods viz. Logistic Regression, Random Forest Classification, Support Vector Machine and Boosting. With the results obtained after training, validating and then testing the best model for each repetition we get that:

1.) Out of the 4 models, Random Forest Classification is the BEST MODEL for our dataset.
2.) MAXIMUM TESTING ACCURACY for the BEST MODEL is 90.43%

This accuracy gives us the information that when we want to find out that if an unknow tile has to be classified as high or low with the set of predictor variables used in the model and the same target variable, random forest classification will give you the correct classification with an accuracy of approximately 91%.

References:

1.) Lecture Notes for STAT-30270 Statistical Machine Learning.
2.) https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234 (This link is useful for more information related to classification accuracy)
3.) https://towardsdatascience.com/a-comprehensive-machine-learning-workflow-with-multiple-modelling-using-caret-and-caretensemble-in-fcbf6d80b5f2 (For EDA)
4.) https://www.r-bloggers.com/parallel-r-model-prediction-building-and-analytics/ (parallel programming.)