Prediction of Solar Power System Coverage

MADHUSUDHAN ASHWATH - 19203116
04/23/2020

Abstract:

In this project we are going to analyze how to we predict the solar power system coverage given all the predictor variables in which the target variable will be **solar_system_count**. Using different types of supervised classification algorithms we predict the best model to be used for the prediction. Based on the best method selected we give the final conclusion.

Introduction:

The data we are using is a subset of the Deep Solar Database, a solar installation data base from the US. DeepSolar is a deep learning framework that analyzes satellite imagery to identify the GPS locations and sizes of solar photovoltaic panels. Each image tile records the amount of solar panel system and is complemented with the features describing social, economic, environmental, geographical and meteorological aspects. As such, the database can be employed to relate key environmental, weather and social economic factors with the adoption of solar photovoltaic panels.

Each row of the data set is a "tile" of interest that is an area corresponding to a detected solar power system, constituted by a set of solar panel 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 in which system has been detected.

We need to predict the solar power system coverage of a tile corresponding to the variable **solar_system_count** in our data set. The target variable is a binary variable and takes an outcome "low" if the tile has a low number of solar power systems and it takes a "high" if the tile has larger number of solar power system. Here we are fitting the model for the given data and finally arrive with an accurate model.

Methods:

Data Import:

We import the data set **"data_deepsolar.csv".** We can study the structure of the data using str() by the observation we can say that the data set consists of 20736 observation with 81 variables. We can see that our target variable is of type factor with two levels "high" and "low". We can also observe that the data set is good with each row being observation and each column being feature.

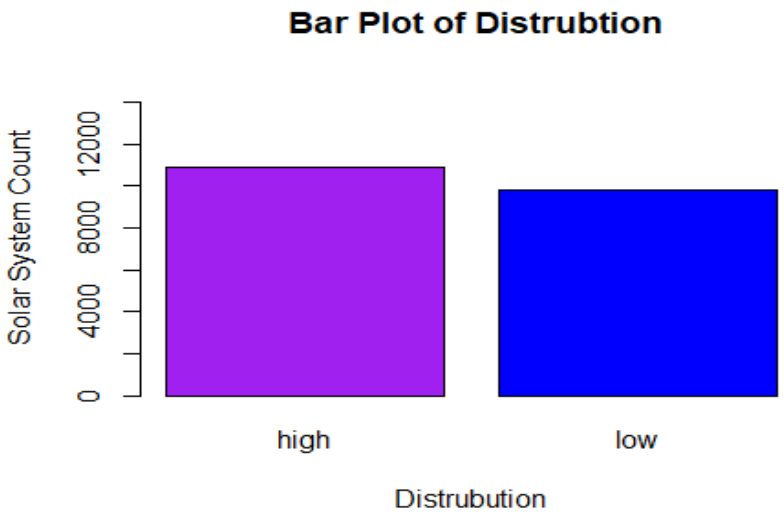
```
rm(list = ls())
# Setting seed
set.seed(19203116)

# Import the data
solar = read.csv("data_project_deepsolar.csv", header = TRUE)

# Data information
str(solar)
```

```
## '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 ...
```

More about the Target Variable:

- As our target variable is a type factor we have two factors **high and low,** we use summary to find out the number of the high and low in the dataset.
- To check is the data is divided evenly, we find the probability of both the factors, based on the results we can see that the data is divided evenly and there is no class imbalance.
- For a visualization purpose we have plotted the distribution.

```
# Summary of target Variable
summary(solar$solar_system_count)

##  high   low
## 10900  9836

# Probability of target variable
prop.table(table(solar$solar_system_count))

##
##      high       low
## 0.5256559 0.4743441

# Plot of target variable
plot(solar$solar_system_count, main = " Bar Plot of Distrubtion", xlab =
"Distrubution", ylab = "Solar System Count", ylim = c(0,15000), col =
c("purple", "blue"))
```



Data Check:

- We us **is.na** to check if there is any missing values.
- We use **duplicated** to check whether there is a duplicate value in the dataset.
- From the below observation we can say that our data is clean.

```
# Checking is data is clean
colSums(is.na(solar))

##                      solar_system_count
state
##                                       0
0

# Check the dupliacte data
solar[duplicated(solar), ]

##  [1] solar_system_count

## <0 rows> (or 0-length row.names)
```

Fitting the Data:

- In order to use the data for further analysis we need to remove the categorical data, which includes the target variable, so that we can check of multi-collinearity.
- In order to reduce the variables in our data which doesn't play a significant role, we find how many bivariate relations have high correlation and if it's significant.
- The correlation matrix will be of 77*77, as the categorical variables are removed from the data.
- We find out the values for different cut-off values of correlation.
- We decide to remove the variables having 0.9 correlations; with each other to remove the effect of multicollinearity using a function of abs we can remove both positive and negative correlation values.

- We can observe using the dim() function there are 61 variables without the target variable.
- We add the target variable to the new data set in other to perform the analysis.

```
# Removing the categotical Variable
solar_num =  solar[-c(1,2,76,79)]

# Check the multicolinearity

## Cut-off at 0.5
sum((cor(solar_num) > 0.5 | cor(solar_num) < -0.5) & cor(solar_num) < 1) /
(77*77)

## [1] 0.05835723

## Cut-off at 0.7
sum((cor(solar_num) > 0.7 | cor(solar_num) < -0.7) & cor(solar_num) < 1) /
(77*77)

## [1] 0.02597403

## Cut-off at 0.9
sum((cor(solar_num) > 0.9 | cor(solar_num) < -0.9) & cor(solar_num) < 1) /
(77*77)

## [1] 0.0094451

# Removing the unwanted variables
var = cor(solar_num)
var[!lower.tri(var)] = 0

# Filtered data and function to scale the data
solar_new = solar_num[,!apply(var,2,function(k) any(abs(k) > 0.9))]
dim(solar_new)

## [1] 20736    61

# Data set with all reduced variable

# Adding the target variable from the original data set
data  = cbind(solar$solar_system_count,solar_new)

# Setting up the column name
colnames(data)[1] = c("solar_system_count")
```

Data Modeling:

The first step in data modeling is to see if the data is clean, we have seen that the data is clean with minimal number of variables.

We are insisted to use the supervised classification algorithms, so we are using the following methods:

- Multinomial/ Logistic Regression
- Random Forest
- Support Vector Machine(SVM)
- Boosting

Logistic and SVM modeling is used as we consider these two methods to be an ideal method as our data is Binary data. And we use Random forest and Boosting as it can give a good accuracy rate when compared to other supervised classification methods.

About the Algorithms:

1. Multinomial/ Logistic Regression :

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

2. Random Forest:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees Random decision forests correct for decision trees' habit of over fitting to their training set.

3. Support Vector Machine(SVM):

Support Vector Machine (SVM) is one of the most popular Machine Learning Classifier. It falls under the category of supervised learning algorithms and uses the concept of Margin to classify between classes.

4. Boosting:

The term 'Boosting' refers to a family of algorithms which converts weak learner to strong learners. Boosting is an ensemble method for improving the model predictions of any given learning algorithm. The idea of boosting is to train weak learners sequentially, each trying to correct its predecessor.

Model Information:

- We need to divide the dataset into 3 parts, namely testing, training and validation sets.
- In our code we are performing 20 trails on the data set.
- The matrix variable “dat” is used as training and validation dataset.
- The matrix variable “dat_test” is used for testing data set to predict once the best model is selected.
- The data is split in a ratio of 75:25 for training and testing data.
- Once the model function is performed the output will be the accuracy, so we use the “acc” matrix to store the accuracy.
- Later based on the best classifier for each repetition, we use the model for prediction using the test data and we get our Best model for each repetition with its accuracy in the column 5 and 6 from the matrix variable “mat”
- The output matrix “mat“ whose size will be of 20 rows and 6 columns which will be storing the validation accuracy of each model in the first 4 columns for each repetition.
- The 5^{th} and 6^{th} column of the output matrix “mat” stores the best model selected for the particular repetition and accuracy of the best model respectively.

Dividing the Dataset for Testing, Training and Validation:

```
# Dividing the data for testing, validation and training
keep = sample(1:nrow(data), size = 0.75 * nrow(data))
test = setdiff(1:nrow(data),keep)

# Validation data
dat = data[keep,]

# Test data
dat_test = data[test,]

# Row Count
v = nrow(dat)

# No of trails
R = 20

# Matrix to store the output
mat = matrix(NA , R , 6)
mat = as.data.frame(mat)
acc = matrix(NA,R,4)

# Change the column name to store the result
colnames(mat) = c("Multinomial" , "RandomForest", "SVM", "Boosting", "Best" ,
"Test")
```

Model Building and Classifier Compression:

```
# Library Files
library(mlbench)
library(nnet)
library(randomForest)
library(kernlab)

# Looping for all the trails


for (n in 1 : R)
{
  # training and validation sets
  print(paste0("Trials :", R))

  ## 75% of data as training set
  dat_train = sample(1:v, size = 0.75*v)

  ## 25% of data as Validation set
  dat_val = sample(setdiff(1:v, dat_train), size = 0.25*v)




  # Modelling

  ## 1. Multinomial Regression
  fitmr = multinom(solar_system_count ~ ., data = dat, subset = dat_train,
trace = FALSE)

  ## 2. Random Forest
  fitrf = randomForest(solar_system_count ~ ., data = dat, subset =
dat_train, importance = TRUE)


  ## 3. SVM
  fitsvm = ksvm(solar_system_count~., data=dat[dat_train,])

  ## 4. Boosting
  fitboost = boosting(solar_system_count~., data =dat[dat_train,], coeflearn
="Breiman", boos =FALSE)
```

```
  # Predicting the model and tabulating

  ## 1. Multinomial Regression
  predmr = predict(fitmr, newdata =  dat[dat_val,])
  tabmr = table(predmr, dat$solar_system_count[dat_val])

  ## 2. Random Forest
  predrf = predict(fitrf, newdata =  dat[dat_val,])
  tabrf = table(predrf, dat$solar_system_count[dat_val])

  ## 3. SVM
  predsvm = predict(fitsvm, newdata =  dat[dat_val,])
  tabsvm = table(predsvm, dat$solar_system_count[dat_val])

  ## 4. Boosting
    predboost = predict(fitboost, newdata =  dat[dat_val,])
  tabboost = predboost$confusion




  # Calculating the accuracy

  ## 1. Multinomial Regression
  accmr = sum(diag(tabmr))/sum(tabmr)

  ## 2. Random Forest
  accrf = sum(diag(tabrf))/sum(tabrf)

  ## 3. SVM
  accsvm = sum(diag(tabsvm))/sum(tabsvm)

  ## 4. Boosting
  accboost = sum(diag(tabboost))/sum(tabboost)



  # Storing accuracy onto matrix


  acc = c(Multinomial = accmr, RandomForest = accrf, SVM = accsvm, Boosting =
accboost)
  mat[n,1] = accmr
  mat[n,2] = accrf
  mat[n,3] = accsvm
  mat[n,4] = accboost
```

```
 # Finding the accurate model


  Best = names(which.max(acc))
  switch(Best,

         ## 1. Multinomial Regression
         Multinomial =
           {
           predtestmr = predict(fitmr, type = "class", newdata = dat_test)
           tabtestmr = table(predtestmr, dat_test$solar_system_count)
           accbest = sum(diag(tabtestmr))/sum(tabtestmr)
           },

         ## 2. Random Forest
         RandomForest =
           {
           predtestrf = predict(fitrf, type = "class", newdata = dat_test)
           tabtestrf = table(predtestrf, dat_test$solar_system_count)
           accbest = sum(diag(tabtestrf))/sum(tabtestrf)
           },

         ## 3. SVM
         SVM =
           {
           predtestsvm = predict(fitsvm, type = "class", newdata = dat_test)
           tabtestsvm = table(predtestsvm, dat_test$solar_system_count)
           accbest = sum(diag(tabtestsvm))/sum(tabtestsvm)
           },

           ## 4. Boosting
         Boosting =
           {
           predtestboost = predict(fitboost, type = "class", newdata =
dat_test)
           tabtestboost = table(predtestboost, dat_test$solar_system_count)
           accbest = sum(diag(tabtestboost))/sum(tabtestboost)
           }
  )



  mat[n,5] = Best
  mat[n,6] = accbest
}
```

```
# Tabulating the result


table(mat[,5])

##
## RandomForest
##           20

tapply(mat[,6], mat[,5], summary)

## $RandomForest
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.9008  0.9016  0.9020  0.9021  0.9028  0.9037
```

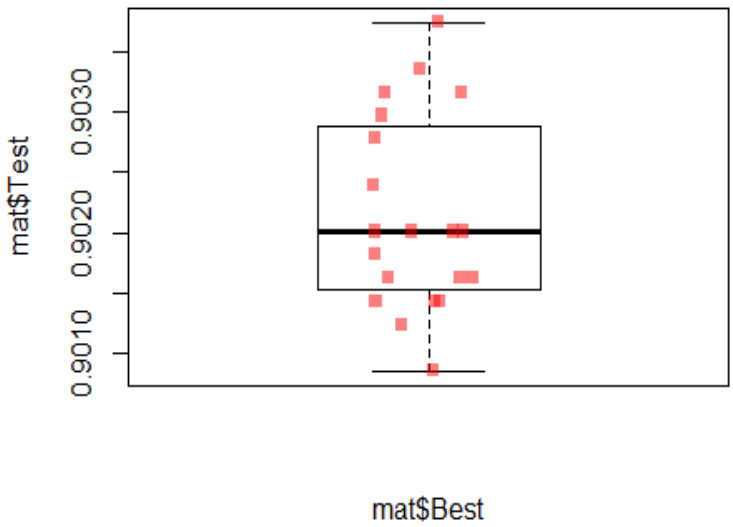
Average, Mean and Standard Deviation:

```
# Average


avg = t(colMeans(as.matrix(mat[,1:4])))
print("Average")

## [1] "Average"

avg

##      Multinomial RandomForest      SVM  Boosting
## [1,]   0.8856739     0.901376 0.897608 0.8919624




# Mean


mean_acc = colMeans(avg)
print("Mean of Average Accuracy")

## [1] "Mean of Average Accuracy"

mean_acc

##  Multinomial RandomForest          SVM     Boosting
##    0.8856739    0.9013760    0.8976080    0.8919624
```

```
# Standard Deviation


std = apply(mat[,1:4], 2, sd)/sqrt(R)
std = as.matrix(std)
print("Standard Deviation")

## [1] "Standard Deviation"

std

##                       [,1]
## Multinomial  0.0010566048
## RandomForest 0.0007895526
## SVM          0.0010583169
## Boosting     0.0011211897
```

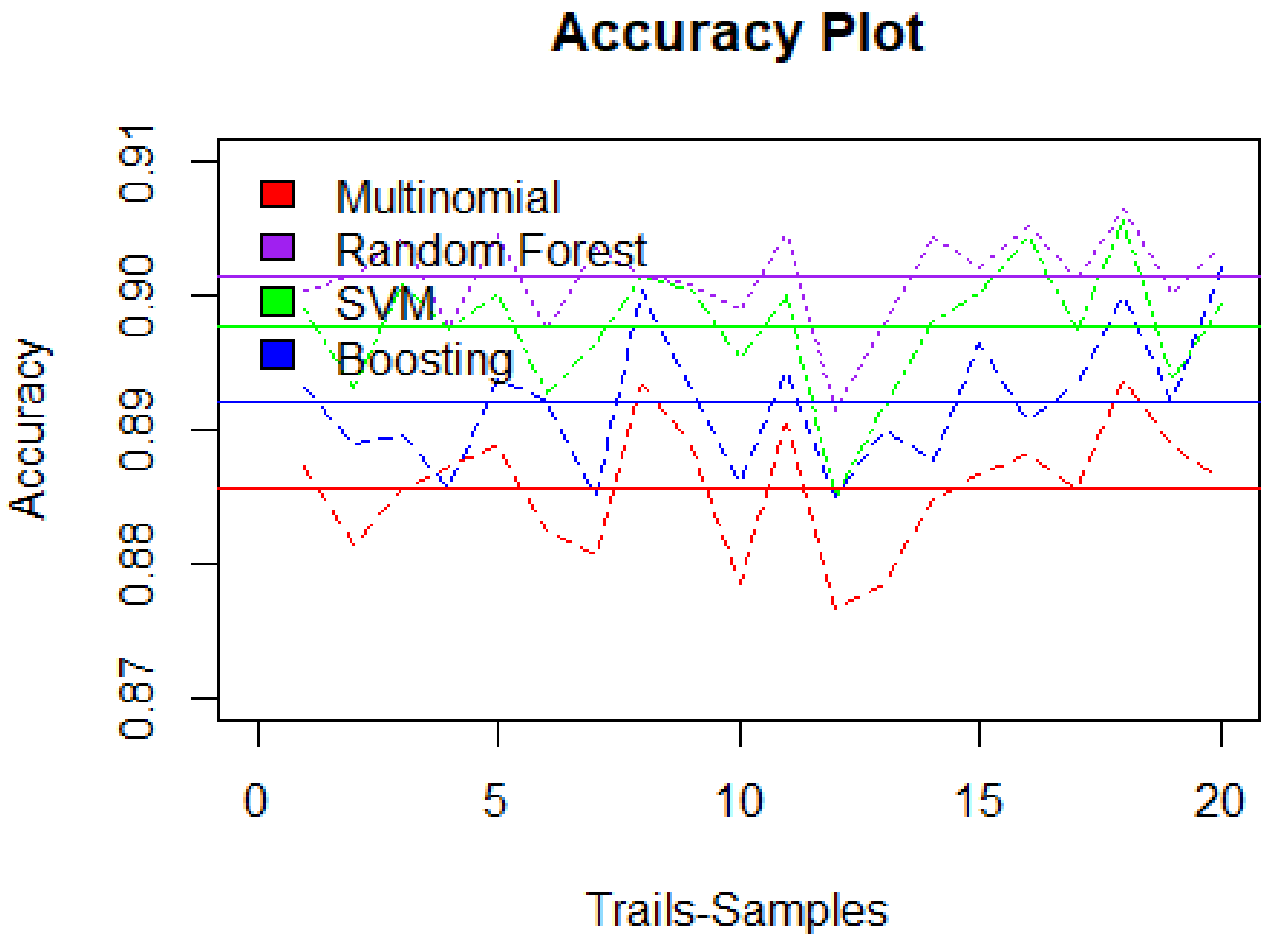
Boxplot of the Best Model:

```
# plotting Box plot to observe the distrubution
# Box plot


boxplot(mat$Test ~ mat$Best)
stripchart(mat$Test ~ mat$Best , add = TRUE, vertical = TRUE, method =
"jitter", pch = 15, col = adjustcolor("red", 0.5))
```



Accuracy Plot of all the models:

```
# plot the values


matplot(mat[,1:4], type = "l", lty = c(2,3,4), col = c("red",
"purple","green","blue"), xlab = "Trails-Samples", ylab = "Accuracy", main =
"Accuracy Plot",ylim = c(0.87, 0.91), xlim = c(0,20))

# Mean Line
abline(h = mean_acc, col = c("red", "purple","green","blue"))

# Legend
legend("topleft", fill = c("red", "purple","green","blue"), legend =
c("Multinomial", "Random Forest","SVM","Boosting"), bty = "n")
```



Result and Discussion:

- When we speak about result of a model we need to calculate the accuracy of the model, which means how well did the model perform for the given dataset.
- The accuracy of the model is given by the ration of number of correct prediction to the total number of input samples, and the accuracy is obtained using the confusion matrix.
- From the above output we can see that our best model is selected as random forest classifier.
- We find the performance of all the 4 models by taking the average of the validation accuracies.
- We find the mean validation accuracy for each model for the graphical representation.
- As we have done 20 repetition for each model, which shows the model performance over 20 repetition.
- In all the 20 repetition we can see that the Random Forest is selected as the best model.
- The above graph is plotted over accuracy of all the models with 20 repetition, we can see that Random forest has the highest accuracy of 90.13% and it is followed by SVM, Boosting and Logistic with accuracy rate of 89.76%, 89.19% and 88.57% respectively.
- Thus we can see that Logistic Regression has the least and Random forest is the best accuracy amount he models used for the given set of data,
- The accuracy of the 4 models used ranges between 0.88 to 0.91.

Random Forest > SVM > Bagging > Multinomial

Conclusion:

We use EDA to know about dataset and based on it we reduced the variables which variables having 0.9 correlations (Highly correlated); with each other to remove the effect of multicollinearity.

We discussed about the 4 supervised classification algorithms namely logistic regression, random forest, SVM and boosting.

For further analysis me divided data into training, validation and test sets for the modeling, proceeding we predicted the model and calculated the accuracy which is used to judge the performance of the classifier.

Based on the results obtained on the repetitive run of the model , we can conclude that out of four models Random Forest Classification is the Best Model for our dataset with Maximum Accuracy of 90.13%.

The accuracy tells us about what we need to find out if an unknown tile has to be classified as high or low with the set of predictor variables used in the model and the same response variable. Where random forest classification will give you an accurate result.

Reference:

1. STAT – 30270 Statistical Machine Learning Lecture notes.
2. "http://web.stanford.edu/group/deepsolar/home" (About the dataset)
3. "https://towardsdatascience.com/a-comprehensive-machine-learning-workflow-with-multiple-modelling-using-caret-and-caretensemble-in-fcbf6d80b5f2 "(EDA)
4. "https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590"(Supervised Learning)
5. "https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/"(Machine learning algorithms)