Identification of Solar Coverage in a Given Region

By Shubhang Periwal 19201104

Abstract

I will be comparing multiple supervised machine learning algorithms such as Random forest, linear regression, multinomial regression, classification trees, bagging, boosting, SVM (state vector machine), one of them was also using polling among three algorithms. This is preceded by dimension reduction technique such as PCA (reducing 57 dimensions to 31), this helps in a faster training, testing and better results in a few cases. Furthermore, I have compared all algorithms mentioned approximately 50 times using random data sampling. As well as done a polling between top 3 algorithms to show whether they are able to perform better when they're used together.

Introduction:

Data

The task is to predict solar power coverage. The data is a subset of the DeepSolar database, a solar installation database 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 target variable is solar_system_count. This variable is a binary variable indicating the coverage of solar power systems in a given tile. The variable 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).

The data is divided into 3 parts, testing, training and validation set. The test set contains 30% of the total number of rows, train set contains 80% of the remaining i.e. 56% of the data and validation set contains 14%.

Pre-processing

Every machine learning algorithm involves pre-processing of data, which involves reduction of dimensions, replication, reduction of noise in data, preparation of data to give as an input in the model. For this step, I am using correlation between two variables followed by PCA(Principle Component Analysis) for dimension reduction.

Model Learning

I am using this pre-processed data as an input in multiple machine models, such as logistic regression, classification tree, random forest, multinomial analysis, SVM, bagging and boosting. I

also tried using polling in between bagging, random forest and SVM in attempt to get a better result.

Model Validation

For model validation I took 50 random samples of the entire data and used all above mentioned algorithms to find out that SVM performed best on the given pre-processed data. This is followed by a graph for each model in order to show the variation in accuracy of the model on validation set.

Methods

Correlation Matrix

A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables.

In perspective of storing data in databases, storing correlated features is somehow similar to storing redundant information which it may cause wasting of storage and also it may cause inconsistent data after updating or editing tuples.If we add so much correlated features to the model we may cause the model to consider unnecessary features and we may have curse of high dimensionality problem.

So, in our case I remove all extra variables which have more than 80% correlation. As both of them would lead to the same prediction. This also helps in lightening of data. I have also removed 2 columns of derived data. From 81, the number of dimensions get reduced to 60, including the predictor variable.

```
corr <- cor(data[train,3:79])
corr[upper.tri(corr)] <- 0
diag(corr)<- 0
temp <- data[1:2]
data <- data[,!apply(corr,2,function(x) any(x > .80))]
dim(data)
```

PCA

Principal component analysis (PCA) is one of the most popular dimension reduction technique. The idea behind PCA is that the data can be expressed in a lower dimensional subspace, characterized by independent coordinate vectors which can explain most of the variability present in the original data. The number of such vectors, and ultimately the dimension of the subspace, is usually set by keeping the first Q vectors which can explain a pre-specified proportion of the total variability in the data (usually in the range 0.70 - 0.99). Sometimes, the size Q of the subspace can also be specified arbitrarily in advance.

```
#computing pca to reduce dimensions, to improve algorithm accuracy
pca <- prcomp(data[train,])
prop <- cumsum(pca$sdev^2)/sum(pca$sdev^2)# compute cumulative proportion of variance
Q <-length( prop[prop<0.95] ) #maintaining atleast 95% of information
Q# only a handful is retained
```

I tried to maintain more than 95% information, which in turn decreased the number of dimensions to 29+2 (one of them being a predictor variable).

```
xz_train <-pca$x[,1:Q]# extract first Q principal components
dat_train <- data.frame(cbind(temp[train,],xz_train))#creating new data frame after
xz_test <- predict(pca,data[test,])[,1:Q]
dat_test <- data.frame(cbind(temp[test,],xz_test))
xz_val <- predict(pca,data[val,])[,1:Q]
dat_val <- data.frame(cbind(temp[val,],xz_val))
```

```
data <- data.frame(cbind(temp),predict(pca,data)[,1:Q])
```

Logistic Regression

The main function for fitting a logistic regression model with binary response variable is glm. Logistic regression is a statistical analysis method used to predict a data value based on prior observations of a data set. Logistic regression is one of the most commonly used machine learning algorithms for binary classification problems, which are problems with two class values, including predictions such as “this or that,” “yes or no” and “A or B.” The purpose of logistic regression is to estimate the probabilities of events, including determining a relationship between features and the probabilities of particular outcomes.

We then calculate the value of optimum tau, which has the highest accuracy for the validation set, and then use it on the testing set to improve the results.

```
fit_glm <- glm(solar_system_count ~., data =  dat_train, family = "binomial")
summary(fit_glm)
```

```
predf <- predict.glm(fit_glm,newdata = dat_test,type = "response" )
```

```
y_test_hat <- ifelse(predf > 0.5,"low","high")
table(temp[test,1],y_test_hat)
```

Random Forest

Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

```
fitRf <- randomForest( solar_system_count ~ ., data = dat_train,maxit=300, trace=FALSE)
```

```
#  Random Forest
predValRf <- predict(fitRf, type = "class", newdata = dat_test)
tabValRf <- table(dat_test$solar_system_count, predValRf)
tabValRf
```

Multinomial Regression

Multinomial logistic regression is a classification method that generalizes logistic regression to multiclass problems, i.e. with more than two possible discrete outcomes.[1] That is, it is a model that is used to predict the probabilities of the different possible outcomes of a categorically distributed dependent variable, given a set of independent variables (which may be real-valued, binary-valued, categorical-valued, etc.).

```
fitLog <- multinom(solar_system_count ~ ., data = dat_train,maxit=300, trace=FALSE)
```

```
# Multinomial Regression
predValLog <- predict(fitLog, type = "class", newdata = dat_test)
tabValLog <- table(dat_test$solar_system_count, predValLog)
tabValLog
```

Classification Tree

Decision tree learning is one of the predictive modeling approaches used in statistics, data mining and machine learning. It uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). 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.

```
# classification tree

fitRp <- rpart(solar_system_count ~ ., data = dat_train)
```

```
# classification tree
predValRp <- predict(fitRp, type = "class", newdata = dat_test)
tabValRp <- table(dat_test$solar_system_count, predValRp)
tabValRp
```

Bagging

Bagging (from bootstrap aggregating), is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

```
#bagging
fitBg <- bagging(solar_system_count ~ ., data = dat_train)
```

```
# bagging
predValBg <- predict(fitBg, type = "class", newdata = dat_test)
tabValBg <- table(dat_test$solar_system_count, predValBg$class)
tabValBg
```

Boosting

Boosting is a powerful procedure that combines the outputs of many "weak" classifiers to produce a "strong" classifier. A classifier can be considered as "weak" when it only performs slightly better than random guessing. The purpose of boosting is to apply a classification algorithm to repeatedly modified versions of the data. The predictions from all the classifiers are combined through a weighted majority vote to produce a final prediction. Like bagging, boosting is a general approach that can be applied to different supervised learning methods. Here we restrict our attention to the context of decision trees.

```
# boosting
predValBst <- predict(fitBst, type = "class", newdata = dat_test)
tabValBst <- table(dat_test$solar_system_count, predValBst$class)
tabValBst
```

SVM

Support vector machines (SVM) are a very popular method for building classifiers. The essential idea is to find the best plane to separate two classes. Extend this to patterns that are not linearly separable by transforming the original data. This will yield a classifier with high performance.

```
#svm
fitSvm <-ksvm(solar_system_count ~ ., data = dat_train)
```

```
#svm
predValSvm <-predict(fitSvm,newdata =dat_test)
tabValSvm <-table(dat_test$solar_system_count, predValSvm)
tabValSvm
```

Polling between Random Forest, SVM and Bagging

```
fin_res <- cbind(predValRf,predValSvm, predValBg$class)
for(1 in 1:6221)
{
  if(fin_res[1,1]=="2")
```

```
    fin_res[1,1] = "low"
  if(fin_res[1,2]=="2")
    fin_res[1,2] = "low"
  if(fin_res[1,1]=="1")
    fin_res[1,1] = "high"
  if(fin_res[1,2]=="1")
    fin_res[1,2] = "high"
}

res <- apply(fin_res,1,function(x){names(which.max(table(x)))})

tabVal <- table(data[test,]$solar_system_count,res)
tabVal
```

I have merged the results of random forest, SVM and Bagging and have used polling system to generate the output for each input. So, solar count is assigned a value of high or low based on how the majority of the 3 algorithms classify it.

Model Comparison

- Division of data into test, train and validation sets.
- Fitting of multiple classifiers into newly sampled data.
- Checking the accuracy of each data set on the validation set, to obtain the best result.
- Iterating this over 50 times to obtain the best performing algorithm.
- Switching to the best performing algorithm for that particular sample and storing all the results to create plots.

Results and Discussion

Correlation among variables

After removing variables from correlation, we are left with 58 columns + 2 columns from 81 dimensions. This helps in reducing the dimension of the data, hence, preventing overfitting and reducing complexity of the machine learning algorithms.

PCA (Principal Component Analysis)

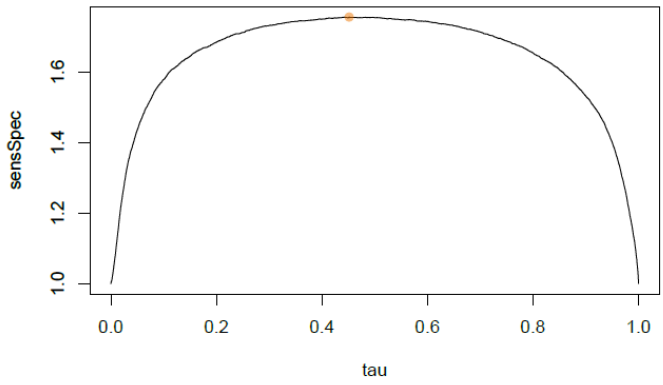
PCA is performed to further reduce the dimensions of the data while maintaining 95% of the original variance. After performing PCA we are left with just 29+2 columns. Which is a 50% reduction from the original set while maintaining 95% of variability.

Logistic Regression

```
     y_test_hat
      high  low
high 2871  388
low   403 2559
```

Accuracy of this model = 87.2%

The next goal is to compute the most efficient value of tau for improving this result further.



	y_test_hat	
	0	1
high	2829	430
low	352	2610

New Accuracy of this model = 87.43%

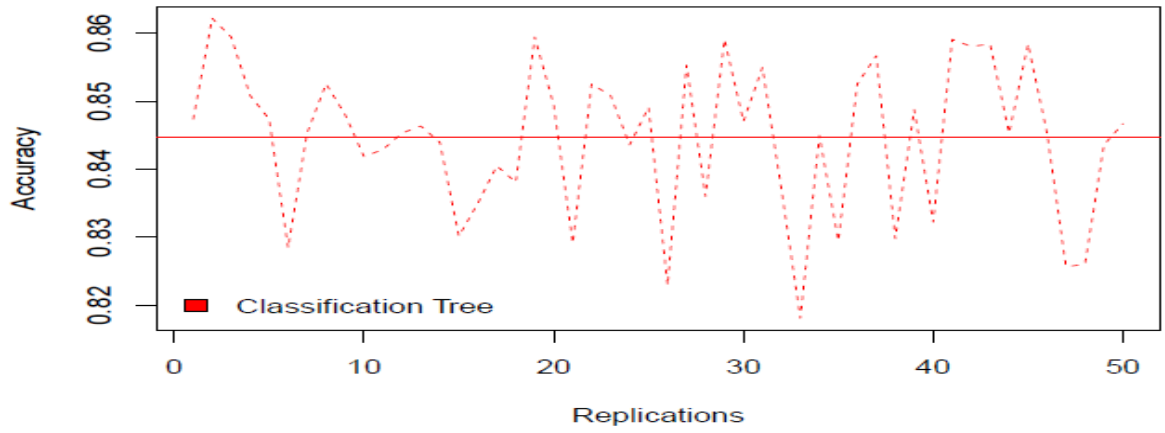
Even though there is not much improvement, as the value of tau (0.451) is really close to the previous assumed value of 0.5.

Classification Tree

	predValRp	
	high	low
high	2524	735
low	342	2620

Accuracy: 82.7%

The classification tree does not perform as well as other algorithms such as Random Forest, but those algorithms are an improvement of this algorithm.

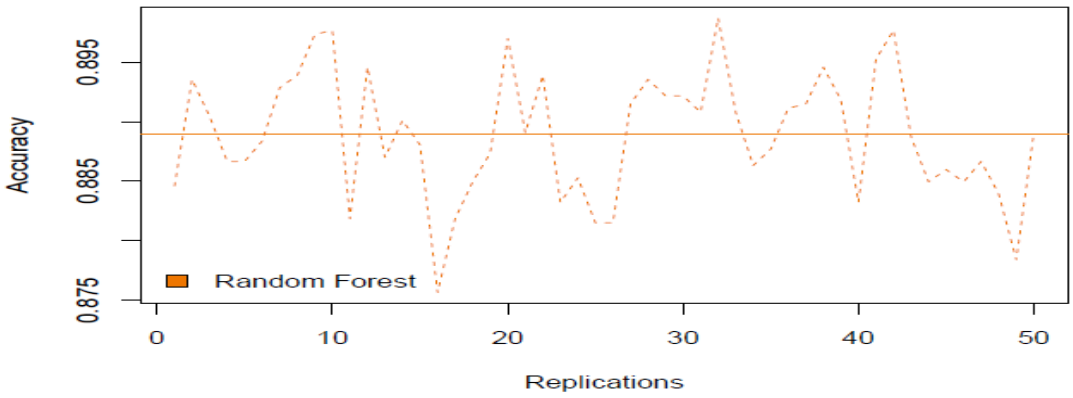


This is the result of algorithms performing on over 50 iterations on random sampled data.

Random Forest

```
      predValRf
       high  low
high 2945  314
low   354 2608
```

Accuracy = 89.26%



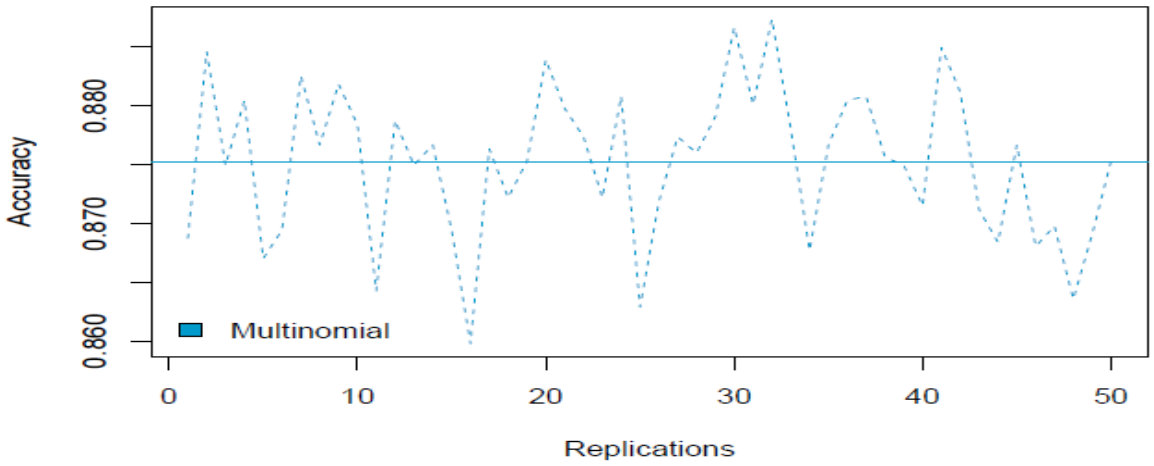
This is the result provided by random forest over 50 replications. We can see that the results consistently lie between 87.5 and 89.5%. This is a good result when compared to classification trees.

Multinomial Regression

```
      predValLog
       high  low
high 2871  388
low   403 2559
```

Accuracy = 87.285%

This is the result provided by random forest over 50 replications. We can see that the results consistently lie between 86 and 88.5%. This is a good result when compared to classification trees and other algorithms as well.

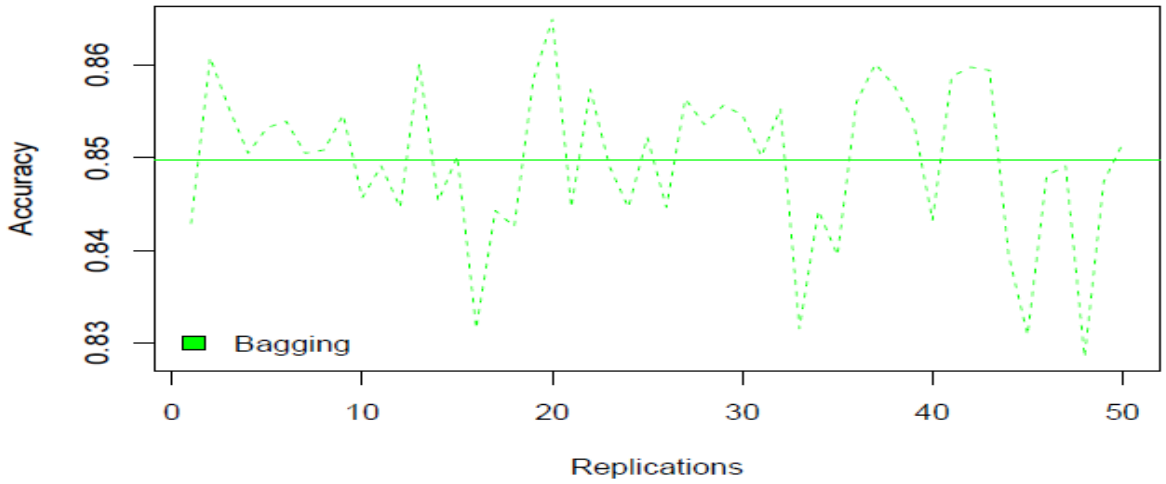


Bagging

Bagging in this case does not perform as good as other algorithms such as logistic regression, random forest, multinomial and SVM.

```
      high  low
high 2603  656
low   363 2599
```

It tends to misclassify “high” as “low”, leading to inaccurate results.



Boosting

Boosting takes multiple models into consideration and performs polling, therefore takes relatively more time but definitely gives good results in general.

```
      high  low
high 2938  321
low   399 2563
```

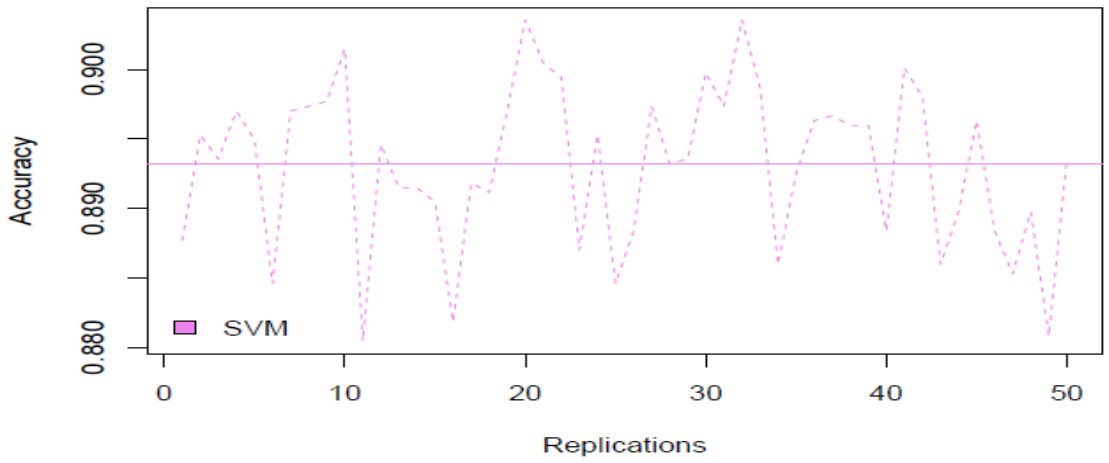
Accuracy: 88.43%

SVM(State Vector Machine)

```
     predValSvm
      high  low
high 2926  333
low   338 2624
```

Accuracy: 89.21%

This is one of the best performing algorithms on this dataset. It provides high amount of accuracy as well as outperforms all other algorithms in accuracy terms.



Polling

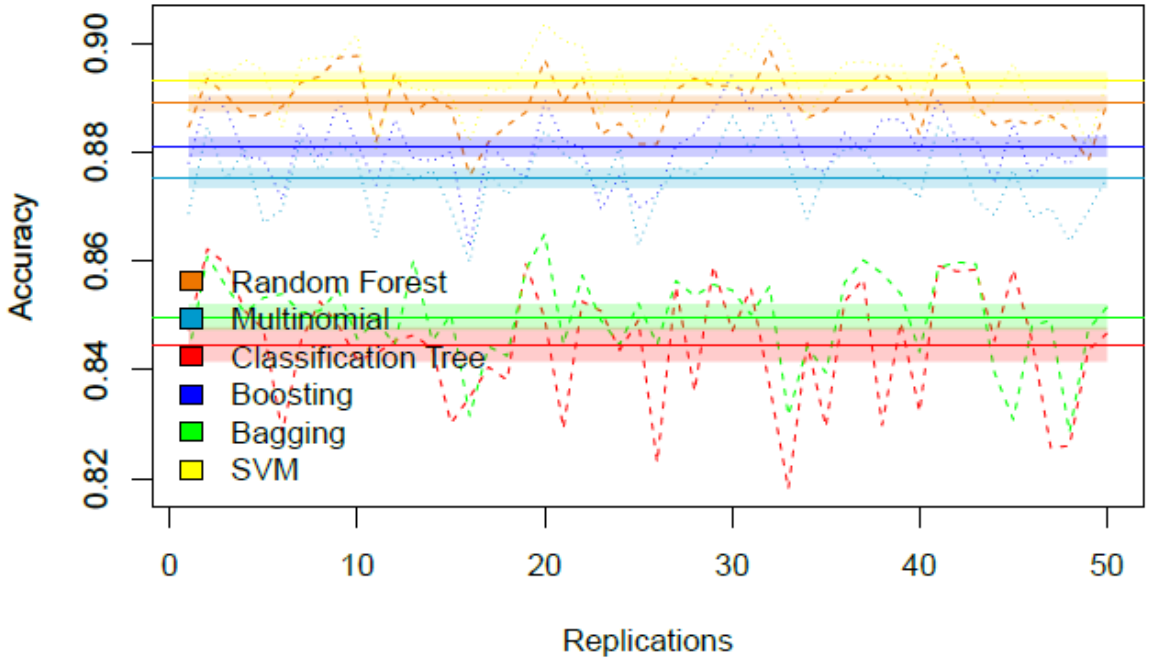
I have used polling among Random Forest, SVM and Bagging, to retain a high accuracy of 88.74%. We can try multiple combinations of polling to obtain the best possible algorithm. In general, polling leads to a higher accuracy, even though it is slower than all algorithms performing individually. We can also conclude that all these algorithms had equivalent accuracy of around 88-89%, so it was not much of an improvement, which means that these algorithms are misclassifying a similar set of inputs. Further research on those particular inputs could be made in future, to improve further accuracy.

```
    res
     high  low
high 2911  348
low   352 2610
```

Random Sampling

Random sampling is performed over 50 times to obtain the optimum result, to see which algorithm outperforms the rest. We can see that SVM performs best among all other algorithms. Out of 50 random iterations, SVM performs better 84% times when compared to the other 5 algorithms. Random Forest algorithm also performs nice here, compared to all other algorithms.

```
Boosting random_Forest           SVM
       1             7            42
```



Conclusion

SVM performs best on reduced data, it even outperforms polling. So, to further increase our accuracy, we could consider training on the entire 57 dimensions instead of reducing it to 31 using correlation and PCA. But the reduction in complexity was substantial and increased the effective pace in the final random sampling scenario. In future we could work on improving the remaining 10%, by looking at the misclassified data and identifying the features which became the differentiating factor, which lead them to misclassification when compared to other algorithms.

References

- ML and AI lab 1 for PCA
- https://colab.research.google.com/ (for execution of some part of the code)
- Kaggle for execution and analysis of the code
- https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

- https://en.wikipedia.org/wiki/Random_forest
- https://scikit-learn.org/stable/modules/svm.html
- http://www.stat.cmu.edu/~ryantibs/statml/

Appendix

---

title: "SML Project"

author: "Shubhang Periwal 19201104"

date: "4/22/2020"

output:

  pdf_document: default

  word_document: default

```{r}

data <- read.csv("data_project_deepsolar.csv",header=TRUE)

set.seed(19201104)

library(randomForest)

library(nnet)

library(rpart)

library(kernlab)

library(ROCR)

library(partykit)

library(adabag)

library(lattice)

library(caret)

library(doParallel)

```

```{r}
```

```
size = nrow(data)
keep <- sample(1:size,(size*0.7))
test <- setdiff(1:size,keep)#30% for testing
train <- sample(keep,(length(keep)*.80))
val <- setdiff(keep,train)
```

```{r}
data = data[,c(-79,-76)]#removing derived data
```

```{r}
corr <- cor(data[train,3:79])
corr[upper.tri(corr)] <- 0
diag(corr)<- 0
temp <- data[1:2]
data <- data[,!apply(corr,2,function(x) any(x > .80))]
dim(data)
#removing highly related data
```

```{r}
data <- scale(data)#scaling data
dim(data)
```

```{r}
#computing pca to reduce dimensions, to improve algorithm accuracy
pca <- prcomp(data[train,])
prop <- cumsum(pca$sdev^2)/sum(pca$sdev^2)# compute cumulative proportion of variance
Q <-length( prop[prop<0.95] ) #maintaining atleast 95% of information
Q# only a handful is retained
#more than 95 % data can be explained using 29 + 2 dimensions, so
#it is unnecessary to  have these many dimensions
```

```{r}
xz_train <-pca$x[,1:Q]# extract first Q principal components
dat_train <- data.frame(cbind(temp[train,],xz_train))#creating new data frame after reducing dimensions
xz_test <- predict(pca,data[test,])[,1:Q]
dat_test <- data.frame(cbind(temp[test,],xz_test))
xz_val <- predict(pca,data[val,])[,1:Q]
dat_val <- data.frame(cbind(temp[val,],xz_val))
```

```{r}
data <- data.frame(cbind(temp),predict(pca,data)[,1:Q])
```

```{r}
fit_glm <- glm(solar_system_count ~., data =  dat_train, family = "binomial")
summary(fit_glm)
predf <- predict.glm(fit_glm,newdata = dat_test,type = "response" )
```

```{r}
y_test_hat <- ifelse(predf > 0.5,"low","high")
table(temp[test,1],y_test_hat)

predObj <- prediction(fitted(fit_glm),temp[train,1])
perf <-performance(predObj,"tpr","fpr")
plot(perf)
abline(0,1,col ="darkorange2",lty =2)

#compute area under roc curve
auc <- performance(predObj,"auc")

```

```{r}
auc@y.values
sens <-performance(predObj,"sens")
spec <-performance(predObj,"spec")
tau <-sens@x.values[[1]]
sensSpec <-sens@y.values[[1]]+spec@y.values[[1]]
best <-which.max(sensSpec)
plot(tau, sensSpec,type ="l")
points(tau[best], sensSpec[best],pch =19,col=adjustcolor("darkorange2",0.5))
tau[best]
```

y_test_hat <- ifelse(predf>tau[best],1,0)

table(temp[test,1],y_test_hat)

```

```{r}

fitRf <- randomForest( solar_system_count ~ ., data = dat_train,maxit=300, trace=FALSE) # randomforest

```

```{r}

#  Random Forest

predValRf <- predict(fitRf, type = "class", newdata = dat_test)

tabValRf <- table(dat_test$solar_system_count, predValRf)

tabValRf

accRf <- sum(diag(tabValRf))/sum(tabValRf)

accRf

```

```{r}

fitLog <- multinom(solar_system_count ~ ., data = dat_train,maxit=300, trace=FALSE)

```

```{r}

# Multinomial Regression

predValLog <- predict(fitLog, type = "class", newdata = dat_test)

tabValLog <- table(dat_test$solar_system_count, predValLog)

tabValLog

accLog <- sum(diag(tabValLog))/sum(tabValLog)

accLog
```

```{r}
# classification tree

fitRp <- rpart(solar_system_count ~ ., data = dat_train)
```

```{r}
# classification tree
predValRp <- predict(fitRp, type = "class", newdata = dat_test)
tabValRp <- table(dat_test$solar_system_count, predValRp)
tabValRp
accRp <- sum(diag(tabValRp))/sum(tabValRp)
accRp
```

```{r}
#bagging
fitBg <- bagging(solar_system_count ~ ., data = dat_train)
```

```{r}
# bagging
predValBg <- predict(fitBg, type = "class", newdata = dat_test)
tabValBg <- table(dat_test$solar_system_count, predValBg$class)
tabValBg
```
```

```
accBg <- sum(diag(tabValBg))/sum(tabValBg)
accBg
```

```{r}
#boosting
fitBst <- boosting(solar_system_count ~ ., data = dat_train, coeflearn = "Breiman", boos = FALSE)
```

```{r}
# boosting
predValBst <- predict(fitBst, type = "class", newdata = dat_test)
tabValBst <- table(dat_test$solar_system_count, predValBst$class)
tabValBst
accBst <- sum(diag(tabValBst))/sum(tabValBst)
accBst
```

```{r}
#svm
fitSvm <-ksvm(solar_system_count ~ ., data = dat_train)
```

```{r}
#svm
predValSvm <-predict(fitSvm,newdata =dat_test)
tabValSvm <-table(dat_test$solar_system_count, predValSvm)
```

```
tabValSvm
accSvm <-sum(diag(tabValSvm))/sum(tabValSvm)
accSvm
```

```{r}
fin_res <- cbind(predValRf,predValSvm, predValBst$class,predf)
for(i in 1:6221)
{
  if(fin_res[i,1]=="2")
    fin_res[i,1] = "low"
  if(fin_res[i,2]=="2")
    fin_res[i,2] = "low"
  if(fin_res[i,1]=="1")
    fin_res[i,1] = "high"
  if(fin_res[i,2]=="1")
    fin_res[i,2] = "high"
}

res <- apply(fin_res,1,function(x){names(which.max(table(x)))})

tabVal <- table(data[test,]$solar_system_count,res)
tabVal
accMax <- sum(diag(tabVal))/sum(tabVal)
accMax
```

```{r}
# replicate the process a number of times
```

```
R <- 50
out <- matrix(NA, R, 4)
colnames(out) <- c("val_random_forest", "val_logistic", "best", "test")
out <- as.data.frame(out)

for ( r in 1:R ) {

  size = nrow(data)
  keep <- sample(1:size,(size*0.7))
  test <- setdiff(1:size,keep)#30% for testing
  train <- sample(keep,(length(keep)*.80))
  val <- setdiff(keep,train)

  # fit classifiers to only the training data

  fitRf <- randomForest( solar_system_count ~ ., data = data[train,], trace=FALSE) #random forest
  fitLog <- multinom(solar_system_count ~ ., data = data[train,], trace=FALSE) #multinomial
  fitBg <- bagging(solar_system_count ~ ., data = data[train,])#bagging
  fitSvm <-ksvm(solar_system_count ~ ., data = data[train,]) #SVM
  fitBst <- boosting(solar_system_count ~ ., data = data[train,], coeflearn = "Breiman", boos =
FALSE)#boosting
  fitRp <- rpart(solar_system_count ~ ., data = data[train,]) #classification tree

  # classify the validation data observations
  #  Random Forest
  predValRf <- predict(fitRf, type = "class", newdata = data[val,])
  tabValRf <- table(data[val,]$solar_system_count, predValRf)
  #tabValRf
  accRf <- sum(diag(tabValRf))/sum(tabValRf)
```

```
 # Multinomial Regression
predValLog <- predict(fitLog, type = "class", newdata = data[val,])
tabValLog <- table(data[val,]$solar_system_count, predValLog)
#tabValLog
accLog <- sum(diag(tabValLog))/sum(tabValLog)
accLog

# classification tree
predValRp <- predict(fitRp, type = "class", newdata = data[val,])
tabValRp <- table(data[val,]$solar_system_count, predValRp)
#tabValRp
accRp <- sum(diag(tabValRp))/sum(tabValRp)
accRp

# boosting
predValBst <- predict(fitBst, type = "class", newdata = data[val,])
tabValBst <- table(data[val,]$solar_system_count, predValBst$class)
#tabValBst
accBst <- sum(diag(tabValBst))/sum(tabValBst)
accBst

# bagging
predValBg <- predict(fitBg, type = "class", newdata = data[val,])
tabValBg <- table(data[val,]$solar_system_count, predValBg$class)
#tabValBg
accBg <- sum(diag(tabValBg))/sum(tabValBg)
accBg
```

```
  #svm
predValSvm <-predict(fitSvm,newdata =data[val,])
tabValSvm <-table(data[val,]$solar_system_count, predValSvm)
#tabValSvm
accSvm <-sum(diag(tabValSvm))/sum(tabValSvm)
accSvm



  # accuracy
  acc <- c(random_Forest = accRf, multinomial = accLog, classificationTree = accRp, Boosting =
accBst, Bagging = accBg, SVM = accSvm)
  out[r,1] <- accRf
  out[r,2] <- accLog
  out[r,3] <- accRp
  out[r,4] <- accBst
  out[r,5] <- accBg
  out[r,6] <- accSvm



  # use the method that did best on the validation data
  # to predict the test data
  best <- names( which.max(acc) )
  switch(best,
         multinomial = {
           predTestLog <- predict(fitLog, type = "class", newdata = data[test,])
           tabTestLog <- table(data[test,]$classes, predTestLog)
           accBest <- sum(diag(tabTestLog))/sum(tabTestLog)
```

```
        },
        #  Random Forest
  random_Forest = {
  predValRf <- predict(fitRf, type = "class", newdata = data[test,])
  tabValRf <- table(data[test,]$solar_system_count, predValRf)
  #tabValRf
  accBest <- sum(diag(tabValRf))/sum(tabValRf)
  },
  # Multinomial Regression
  multinomial = {
    predValLog <- predict(fitLog, type = "class", newdata = data[test,])
tabValLog <- table(data[test,]$solar_system_count, predValLog)
#tabValLog
accBest <- sum(diag(tabValLog))/sum(tabValLog)
  },

# classification tree
classificationTree = {
predValRp <- predict(fitRp, type = "class", newdata = data[test,])
tabValRp <- table(data[test,]$solar_system_count, predValRp)
#tabValRp
accBest <- sum(diag(tabValRp))/sum(tabValRp)
},

# boosting
Boosting = {
predValBst <- predict(fitBst, type = "class", newdata = data[test,])
tabValBst <- table(dat_test$solar_system_count, predValBst$class)
```

```
#tabValBst
accBest <- sum(diag(tabValBst))/sum(tabValBst)
},



# bagging
Bagging = {
predValBg <- predict(fitBg, type = "class", newdata = data[test,])
tabValBg <- table(data[test,]$solar_system_count, predValBg$class)
#tabValBg
accBest <- sum(diag(tabValBg))/sum(tabValBg)
},

SVM = {
  #svm
predValSvm <-predict(fitSvm,newdata =data[test,])
tabValSvm <-table(data[test,]$solar_system_count, predValSvm)
#tabValSvm
accBest <-sum(diag(tabValSvm))/sum(tabValSvm)

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

}
```

```{r}
# check out the error rate summary statistics
table(out[,7])
tapply(out[,8], out[,7], summary)
boxplot(out$test ~ out$best)
stripchart(out$test ~ out$best, add = TRUE, vertical = TRUE,
        method = "jitter", pch = 19, col = adjustcolor("magenta3", 0.2))

#avg <- t( sapply(out[1:6], colMeans))
avg <- out[,1:6]
meanAcc <- colMeans(out[,1:6]) # estimated mean accuracy
meanAcc
sdAcc <- apply(avg, 2, sd)/sqrt(R) # estimated mean accuracy standard deviation
sdAcc


matplot(avg, type = "l", lty = c(2,3), col = c("darkorange2", "deepskyblue3","red","blue","green","yellow"),
xlab = "Replications", ylab = "Accuracy")
#
# add confidence intervals
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("red", 0.2), border = FALSE)
polygon(c(1:R, R:1), bounds4, col = adjustcolor("blue", 0.2), border = FALSE)
polygon(c(1:R, R:1), bounds5, col = adjustcolor("green", 0.2), border = FALSE)
polygon(c(1:R, R:1), bounds6, col = adjustcolor("yellow", 0.2), border = FALSE)
#
# add estimated mean line
abline(h = meanAcc, col = c("darkorange2", "deepskyblue3","red","blue","green","yellow"))
#
# add legend
legend("bottomleft", fill = c("darkorange2", "deepskyblue3","red","blue","green","yellow"),
legend = c("Random Forest", "Multinomial","Classification Tree","Boosting","Bagging","SVM"), bty = "n")
```

```{r}
#Random Forest
matplot(avg[1], type = "l", lty = c(2,3), col = c("darkorange2"),
xlab = "Replications", ylab = "Accuracy")
#
# add estimated mean line
abline(h = meanAcc[1], col = c("darkorange2"))
#
# add legend
legend("bottomleft", fill = c("darkorange2"),
legend = c("Random Forest"), bty = "n")
```

```
#Multinomial
matplot(avg[2], type = "l", lty = c(2,3), col = c("deepskyblue3"),
xlab = "Replications", ylab = "Accuracy")

# add estimated mean line
abline(h = meanAcc[2], col = c("deepskyblue3"))
#
# add legend
legend("bottomleft", fill = c("deepskyblue3"),
legend = c("Multinomial"), bty = "n")



#Classification tree
matplot(avg[3], type = "l", lty = c(2,3), col = c("red"),
xlab = "Replications", ylab = "Accuracy")

# add estimated mean line
abline(h = meanAcc[3], col = c("red"))
#
# add legend
legend("bottomleft", fill = c("red"),
legend = c("Classification Tree"), bty = "n")


#Boosting
```

```
matplot(avg[4], type = "l", lty = c(2,3), col = c("blue"),
xlab = "Replications", ylab = "Accuracy")
#
# add estimated mean line
abline(h = meanAcc[4], col = c("blue"))
#
# add legend
legend("bottomleft", fill = c("blue"),
legend = c("Boosting"), bty = "n")



#Bagging
matplot(avg[5], type = "l", lty = c(2,3), col = c("green"),
xlab = "Replications", ylab = "Accuracy")

# add estimated mean line
abline(h = meanAcc[5], col = c("green"))
#
# add legend
legend("bottomleft", fill = c("green"),
legend = c("Bagging"), bty = "n")



#SVM
matplot(avg[6], type = "l", lty = c(2,3), col = c("violet"),
xlab = "Replications", ylab = "Accuracy")
```

```
# add estimated mean line
abline(h = meanAcc[6], col = c("violet"))
#
# add legend
legend("bottomleft", fill = c("violet"),
legend = c("SVM"), bty = "n")

```