School of Mathematical and Statistical Sciences
Arizona State University

Student: Juyoung Choi

Other students: Ziwei Chen

Course: STP 598 (Machine learning)

Program: Statistics MS

Instructor: Robert McCulloch

Date: Fall 2019

Machine Learning Final Project

Juyoung Choi, Ziwei Chen

November 25, 2019

Introduction

This data was extracted from the 1994 Census bureau database by Ronny Kohavi and Barry Becker. It is a binary response dataset. This document is a research paper to analyze the dataset with four important methods (Lasso regression, random forest, boosting and Support Vector Machines). We did grid search on all methods to find the best final model to predict the response. The prediction task is to determine whether a person makes over 50K a year. If he does, that will consider as "high" class or will consider as "low" class. Comparing four methods based on plots and results for AUC. Also, we look at the accuracy of the final model. Since the dataset including many columns, it is a difficult dataset which is also unbalanced. Also, we tried some resampling methods like SMOTE, ROSE, simple oversampling to resample dataset to balanced data.

Data

Dataset can be found from "https://www.kaggle.com/uciml/adult-census-income". It is a classification social dataset with one binary response. Total 16383 observations with 14 attributes and one response.

Variable	Description
`age`	continuous.
`workclass`	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
`fnlwgt`	continuous.
`education`	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters,1st-4th, 10th, Doctorate, 5th-6th, Preschool.
`education-num`	continuous.
`marital-status`	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
`occupation`	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
`relationship`	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
`race`	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
`sex`	Female, Male.
`capital-gain`	continuous.
`capital-loss`	continuous.
`hours-per-week`	continuous.
`native-country`	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.
`income`	>50K, <=50K

Clean and Processing Data

We spent a lot of time here. But this part is very interesting. The dataset is not clean with missing values and it is complicated including both factor variables and continuous variables. Each factor variable contains several levels which make our data clean difficult. Thus, recoding to the factor variables is very important to our research. Also, information to some columns are overlapped. Besides, the dataset is unbalanced and the values for numerical variables have a large range. These items will make huge impacts to the results need to be normalized. In machine learning, if values are not numeric, that will become difficult to do future research. Here is a very vital part that use Dummy variable to deal with such data. For example, we can make factors into numerical value.

1. Recode Nominal Variables

- `income`: <=50K = 0, >50K = 1
- `workclass`: Private =1, State-gov =2, Federal-gov =3, Self-emp-not-inc =4, Self-emp-inc =5, Local-gov =6, Without-pay =7, Never-worked =8
- `marital status`: Widowed =1, Divorced =2, Separated =3, Never-married =4, Married-civ-spouse =5, Married-AF-spouse =6, Married-spouse-absent =7
- `relationships`: Not-in-family =1, Unmarried =2, Own-child =3, Other-relative =4, Husband =5, Wife =6.
- `occupation`: delete, no recoding. Since it has fourteen levels in this factor. And we cannot give numerical value like education to it. Because we can don't have any reasons to rank it. Every occupation is even. People are even when they are born. And if we give 0-13 value to divide it into fourteen categories, it is too complicated which have no much meaning to our results.
- `fnlwgt`: delete it. Not relevant to the data.
- `education`: delete, no recoding. Another column named "Education-num" is the numerical value for this column. They are overlapped. Person has higher education background will get higher values. From 1 to 16.
- `race`: White =1, Black =2, Asian-Pac-Islander =3, Amer-Indian-Eskimo =4, Other =5 It is not a rank, we still consider it as nominal variable, just divide it into five categories to see if race has effects to income.
- `sex`: Female =1, male =2
- `native country`: United-States=1, all other will consider as 2 There are many levels in this factor. We just want to see residents and foreigners. That's why we only divided it into two groups.

2. Delete Missing Values

Almost every column contains missing value. The best way to do is dismiss such observation rather than just delete the column or the row. Also, observation who doesn't work will be delete from the dataset.

3. Divide Data into Training and Testing Datasets

- 75% : training data, 25% : test data

4. Create dummy variables

It is converting a categorical variable to as many binary variables as here are categories. Our data is complicated, and many columns come with factors. How do deal with them? The good way is to do dummy variable. After this process, all attributes become numerical value.

5. Scale Dataset

We normalized the continuous variables in both training and test. Because their values are in a wide range, which will affect our results. After normalizing, values of AUC were improved.

6. Randomize Dataset

For avoiding 0 and 1 are clustering together, we randomize dataset, in both training and test.

7. Resampling Training Dataset

This step should do after normalizing and randomizing. We tried SMOTE, ROSE, Oversampling, Undersampling and Bothsampling. No one will improve AUC. Our training dataset consists of 30% "1" and 70% "0", not an extremely unbalanced dataset, just a little. So, we listened to Dr. McCulloch, don't do resampling.

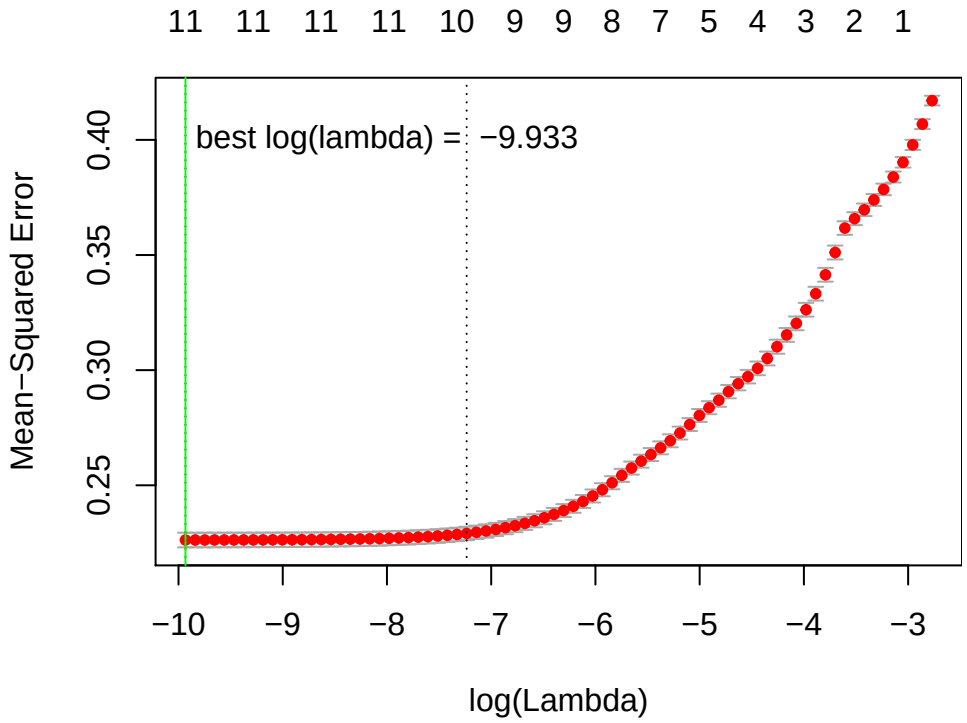
Machine Learning Methods

A. LASSO Regression

We use lasso regression instead of logistic regression. Because it computes faster and we don't need to do stepwise, this step needs a long time and our dataset is not a small dataset. Also stepwise has disadvantages. For instance, if we use backward stepwise, once the variable was given a zero coefficient, it will be deleted forever in this step, cannot enter into the next step. We use package named `glmnet` to obtain lasso regression. It is a shrinkage method. The package `pROC` wes used to obtain the ROC (Receiver Operating Characteristic) curve and find AUC.(area under ROC curve). We did grid search to find the best lambda for our final model.

$$Sensitivity = \frac{number\ of\ true\ positives}{number\ of\ true\ positives + number\ of\ false\ negatives}$$

We can calculate AUC (Area Under the ROC Curve) using pROC and AUC for LASSO is 0.901 and Accuracy is 0.839 from `confusionMatrix`.



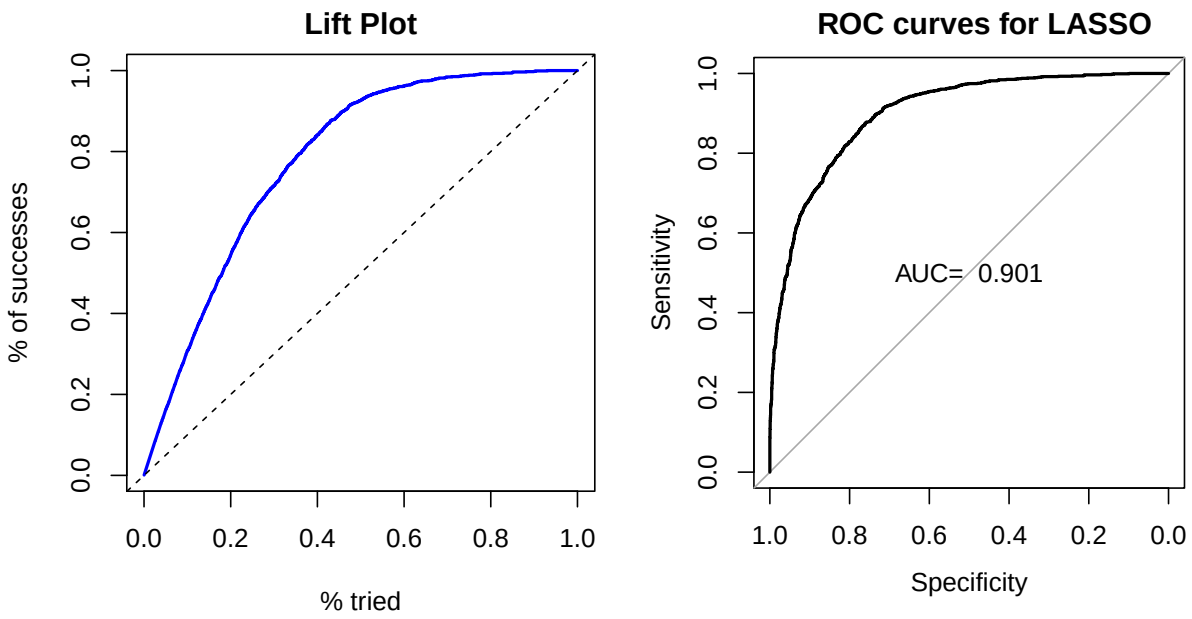


Figure 1: Parameter selection of λ (top), lift plot (bottom left), ROC plot (bottom right)

B. Random Forests

We use a package named `randomForest` and `caret` in R to do it. We do grid search on mtry, maxnodes and ntrees to choose the best parameter. Parameters are too much, out data is huge. We have to wait for a long time to obtain each answer, so we only pick up three parameters to do grid search. If you wanted to do more, the methods are exactly the same.

- `ntree`: number of trees in the forest
- `mtry`: Number of candidates draw to feed the algorithm. By default, it is the square of the number of columns.
- `maxnodes`: Set the maximum amounts of terminal nodes in the forest Grid search is simple, the model will be evaluated over all the combination you pass in the function, using cross-validation. Here we use 10-fold validation.

Step 1: we choose `mtry` and the best answer is 6.
Step 2: we choose `maxnodes` and the best answer is 7.
Step 3: we choose `ntree` and the best answer is 800.
Therefore, our final model appeared. (`mtry`=6, `maxnodes`=7, `ntree`=800)
Still use ROC and lift plot to measure performance. AUC is 0.904 which suggests that sensitivity can get to 90%, it is great. And accuracy is 0.8361 from `confusionMatrix`.

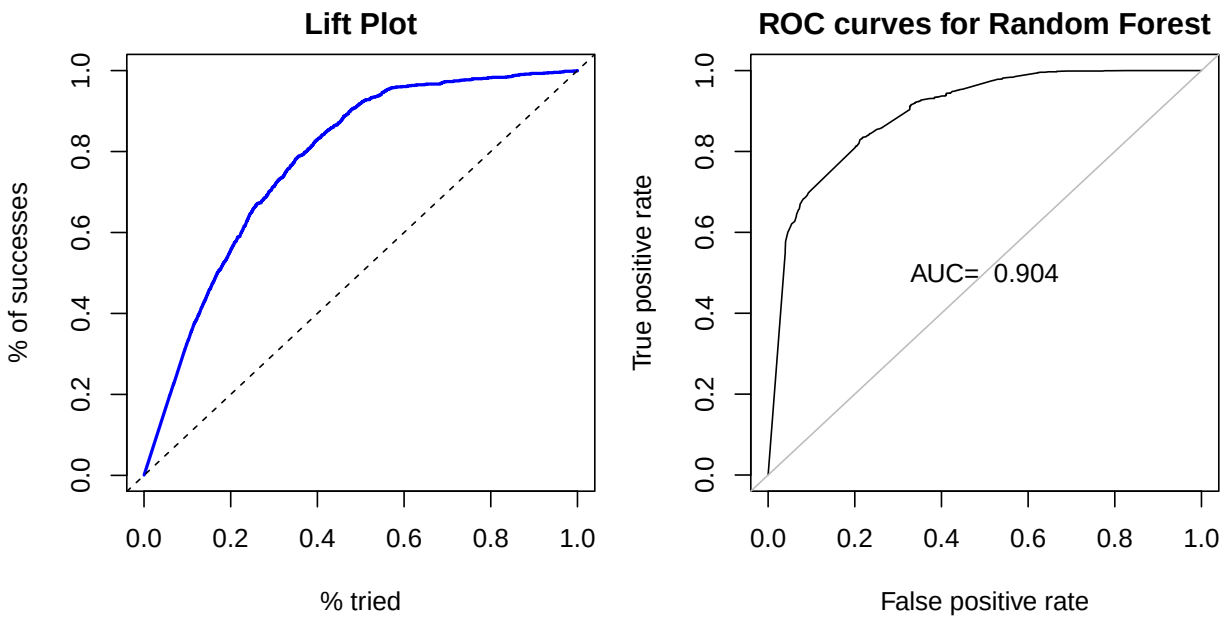


Figure 2: Lift plot (left), ROC curve (right) for Random Forests

C. Boosting

In this method, we use the package `glm` in r. Still do cross-validation to choose the best model based on ROC. (metric='ROC'). We use library `caret` to do grid search based on `traincontrol` function. This time, we didn't pick parameters one by one. We did hyperparameter optimization. We waited for about 2 hours to get the results, it is very slow. Fortunately, we get the final model with shrinkage factor =0.01, number of tress= 1500 and the size of each new tree=9. AUC for boosting is 0.939, pretty good. And the accuracy is 0.8364, also good.

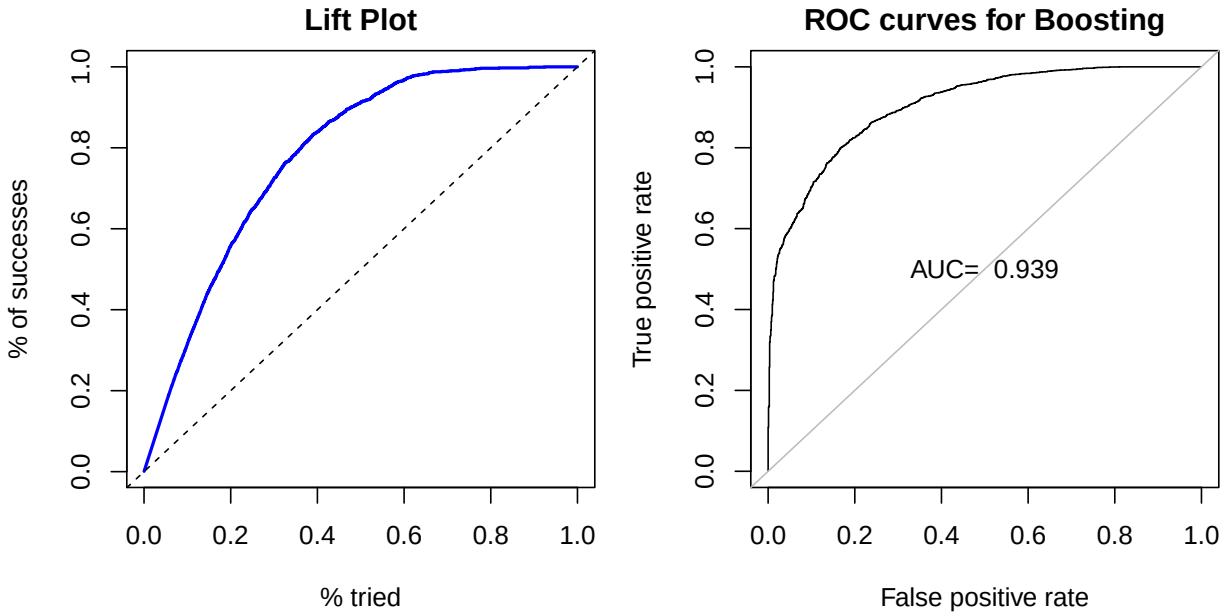


Figure 3: Lift plot (left), ROC curve (right) for Boosting

D. Support Vector Machines

Our data is binary data, so we want to try SVM, a very popular method for classifier. This wasn't methods taught in class, but we read the book. (Chapter 12 of Element) Support Vectors Classifier tries to find the best hyperplane to separate the different classes by maximizing the distance between sample points and the hyperplane. There are four itemes which need to do grid search.

1. `Kernel` represents which model we use. It selects the type of hyperplane used to separate the data. 'linear' will use a linear hyperplane (a line in the case of 2D data). 'rbf' and 'poly' uses a nonlinear hyper-plane

2. `gamma` is a parameter for nonlinear hyperplanes. The higher the gamma value, the more the model tries to fit the training data set, which means more accurate. But large gamma leads to overfitting.

3. `C` is the penalty parameter of the error term. There is a trade-off between smooth decision boundary and classifying the training points correctly. C controls trade off. Also, large C represents overfitting.

4. `degree` is a parameter used when kernel is set to 'poly'. It is the degree of the polynomial used to find the hyperplane to split the data. `degree`=1 means 'linear' kernel. Our data is large, we only tune on two main parameters. After tunning, the best final model came with parameter `cost`=1 and `gamma`=0.03125. And the AUC of SVM is 0.906 and accuracy is 0.8403. Very competitive to compare with other methods. We use e1071 to tune parameters which is way faster than cross-validation. When we use 10-fold cross-validation, repeated 5 times for nonlinear kernel. We waited for a whole night, the result didn't appear on the next morning.

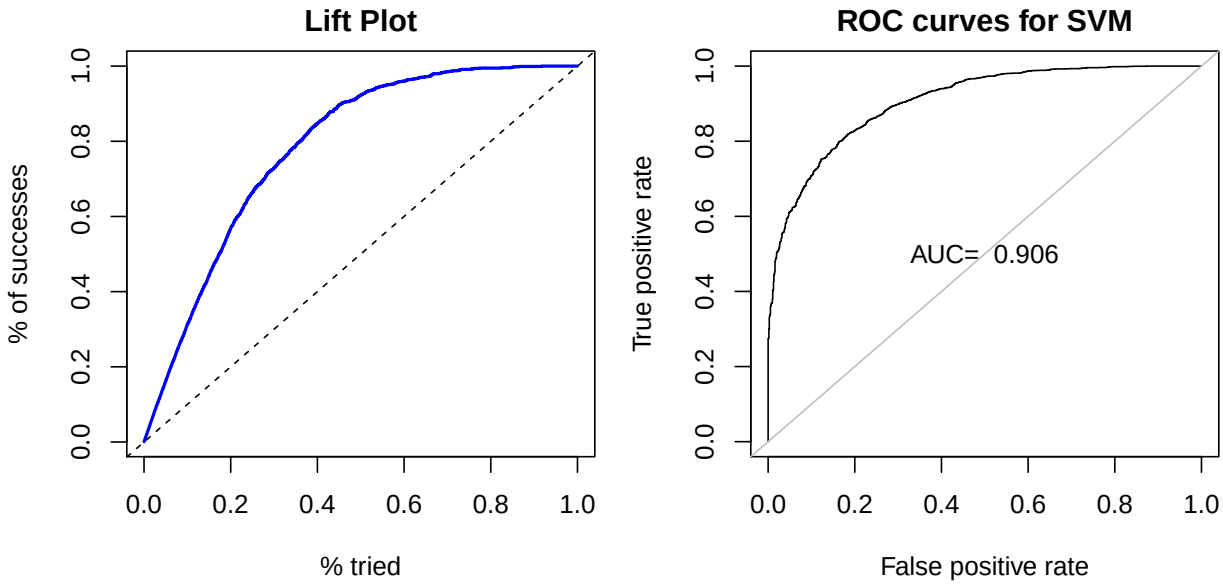


Figure 4: Lift plot (left), ROC curve (right) for SVM

Confusion Matrix

A confusion matrix shows how many observations were correctly or incorrectly classified. The diagonal elements of Confusion matrix indicate correct predictions, while the off-diagonals represent incorrect predictions. From this table, it appears that SVM is working a little bit better than random forests and boosting.

Method	Accuracy		low	high
LASSO	0.8390	low	2462	414
		high	198	727
Random Forests	0.8361	low	2511	474
		high	149	667
Boosting	0.8364	low	2495	457
		high	165	684
SVM	0.8403	low	2463	401
		high	206	731

Variable Importance

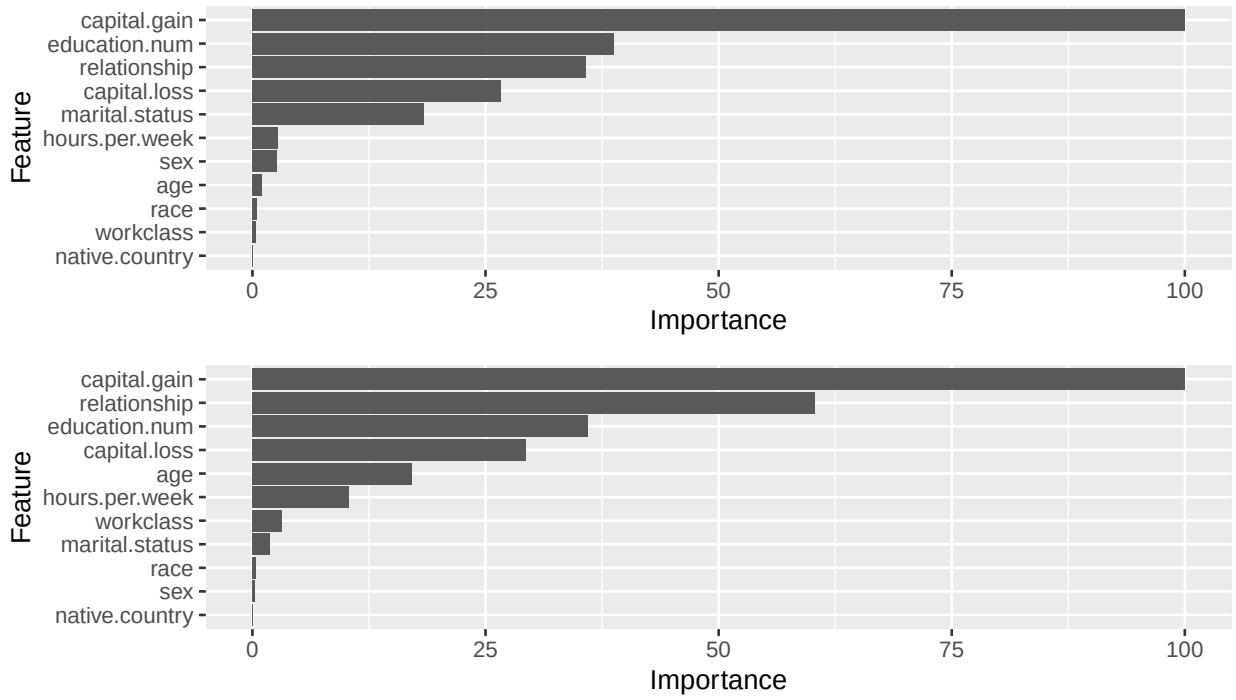


Figure 5: Variable importance plot for Random Forest (top) and Boosting (bottom)

Here are the variable importance foreach of the 11 predictor variables obtained from Random Forests and Boosting for. Not surprisingly, we can see that `capital.gain` is the most important variable for both random forests and boosting. For random forests, top five variables are: `capital.gain`, `education.num`, `relationship`, `capital.loss`, `martial.status`. For Boosting, top five variables are: `capital.gain`, `relationship`, `education.num`, `capital.loss`, `age`. The othes are somewhat less influential.

Comparison of AUC Performance

AUC is the area under the ROC curve and commonly used to ciassify binary data by displaying a curve that is able to better represent the algorithm's performance. For much easier to compare AUC for four different methods: lasso regression, random forest, boosting and SVM. We make a table for them. From table, all the results is pretty good and suggests a valuable model. Boosting gives highest AUC compared to other three methods.

Method	AUC
LASSO	0.901
Random Forests	0.904
Boosting	0.939
SVM	0.906

Further Research

During working, we spent a lot of time on cleaning data and pretreatment on data. Our dataset is complicated and a little unbalanced. As mentioned before, we tried five methods to resample it. After applying any of them to the resampling step, the AUC decreases 1 to 2 percentage. we are very confused, so we asked Dr. McCulloch. He told me not to do resampling. But we are thinking why such resampling methods decreases AUC, it doesn't make sense. So, we pick an extremely unbalanced classification dataset to see if resampling improves AUC, the percentage of the training dataset is almost 2% on "1" and 98% on "0". We used SMOTE to resample it. The AUC increased as expected and accuracy increased as well. Why resampling isn't useful to our dataset? After looking up much information, we make an assumption that our dataset is not a serious unbalanced dataset, if we do resampling, the noise for the model will also increase. The advantages brought by resampling cannot remedy for the disadvantages brought by noise. That's why it works on extremely unbalanced dataset. But it is only our assumption, we need to do more work on it to find the reason. And also, we find that resampling should to be done after normalizing. It is very important. First normalizing, then resampling will increase AUC about 3 to 4 percentage.

Conclusion

We employed four different machine learning methods, which are lasso regression, random forests, boosting, and SVM in the census dataset. Variable importance and AUC are both available measurements for judging a machine learning methods for binary data. Sometimes, accuracy can't give a reasonable explanation for those methods for some difficult data. Using variable importance and AUC, we found that all of the methods yield reasonable AUC values, but boosting has higher AUC than Lasso, Random Forests, and SVM. So we could conclude that boosting is performing better on average in this dataset. And grid search is very important to machine learning, with grid search we can get better predictions than with default parameters for each method.

Please see the following pages for the R script used in this project.

```
rm(list=ls())
library(readxl)
ad = read_excel("Desktop/stp598/final project/adult.xls")
head(ad)
################################################################################
####################      Pretreatment to Dataset    ######################
################################################################################
#-------------------------------- delete missing values-----------------------------------------#
ad[ad=="?"] = NA
#which(is.na(ad))
## delete observations which contain missing values
ad <- na.omit(ad)
## check missing values again
which(is.na(ad))
#---------------------- define reponse variable as binary variable (income)----------------
ad[,15]=ifelse(ad[,15]=="<=50K",0,1)
#------------------------- recode some factors into numeric-----------------------------
library(dplyr)
ad$workclass=recode(ad$workclass, 'Private'=1, 'State-gov'=2, 'Federal-gov'=3,
                    'Self-emp-not-inc'=4, 'Self-emp-inc'=5,
                   'Local-gov'=6, 'Without-pay'=7, 'Never-worked'=8)
ad$marital.status=recode(ad$marital.status,'Widowed'=1, 'Divorced'=2, 'Separated'=3,
                         'Never-married'=4,'Married-civ-spouse'=5,'Married-AF-spouse'=6,
                         'Married-spouse-absent'=7)

ad$relationship=recode(ad$relationship, 'Not-in-family'=1, 'Unmarried'=2,'Own-child'=3,
                       'Other-relative'=4,'Husband'=5, 'Wife'=6)

ad$race=recode(ad$race, 'White'=1,'Black'=2,'Asian-Pac-Islander'=3,'Amer-Indian-Eskimo'=4,
               'Other'=5)
ad$sex=ifelse(ad$sex=="Female",1,2)
ad$native.country=ifelse(ad$native.country=="United-States",1,2)
#-------------------------- define continuous and nominal variables---------------------------------#
#drop useless column
ad$education=NULL
ad$education=NULL
ad$occupation=NULL

ad$occupation=NULL
ad$fnlwgt=NULL

ad$income=as.factor(ad$income)
ad$age=as.numeric(ad$age)
ad$workclass=as.integer(ad$workclass)
ad$education.num=as.numeric(ad$education.num)
ad$marital.status=as.integer(ad$marital.status)
ad$relationship=as.integer(ad$relationship)
```

```
ad$race=as.integer(ad$race)
ad$sex=as.integer(ad$sex)
ad$capital.gain=as.numeric(ad$capital.gain)
ad$capital.loss=as.numeric(ad$capital.loss)
ad$hours.per.week=as.numeric(ad$hours.per.week)
ad$native.country=as.integer(ad$native.country)
#------------------------ divide data into training and testing datasets----------------------------#
## 75% of the sample size
smp_size = floor(0.75 * nrow(ad))

#set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(ad)), size = smp_size)
# Build train and test dataset
ad.train = ad[train_ind, ]
ad.test = ad[-train_ind, ]
y=ad.train$income
income=ad.test$income
ad.train$income=as.factor(ad.train$income)
# One-Hot Encoding
# Creating dummy variables is converting a categorical variable to as many binary variables as here are categories.
dummies_model1 = dummyVars(income ~ ., data=ad.train)
dummies_model2 = dummyVars(income ~ ., data=ad.test)
# Create the dummy variables using predict. The Y variable (Purchase) will not be present in trainData_mat.
trainData_mat = predict(dummies_model1, newdata = ad.train)
testData_mat = predict(dummies_model2, newdata = ad.test)
# # Convert to dataframe
ad.train = data.frame(trainData_mat)
ad.train=data.frame(cbind(ad.train,y))# Append the resonse variable

ad.test = data.frame(testData_mat)
ad.test=data.frame(cbind(ad.test,income))# Append the response variable
# # See the structure of the new dataset
str(ad.train)
str(ad.test)

#--------------------------------------scale the dataset--------------------------------------------#
ad.train[] = lapply(ad.train, function(x)
  if(is.numeric(x))
  {
  (x-min(x))/(max(x)-min(x))
  }
  else x)
ad.test[] = lapply(ad.test, function(x)
  if(is.numeric(x))
  {
    (x-min(x))/(max(x)-min(x))
```

```
  }
 else x)
#---------------------------------------randomize data-------------------------------------------#
ad.test = ad.test[sample(nrow(ad.test)),]
ad.train = ad.train[sample(nrow(ad.train)),]
ad.test = ad.test[sample(nrow(ad.test)),]
ad.train = ad.train[sample(nrow(ad.train)),]
row.names(ad.train)=NULL
row.names(ad.test)=NULL
#---------------------------------------Over Sampling-------------------------------------------#
#                  *********** IMPORTANT ***********                        #
#                  ****** Do AFTER Normalized ******                       #
#                  *********************************                       #
#library(DMwR)
#set.seed(9560)
#ad.train=SMOTE(income~., data=as.data.frame(ad.train),k=10,perc.over =200, perc.under = 150 )
#table(ad.train$income)
#prop.table(table(ad.train$income))
#-------////////----ROSE----////////-------#
#set.seed(123)
#library(ROSE)
#ad.train=ovun.sample(income~., data=as.data.frame(ad.train), method = "over", N=16268)$data
#--------------####------Grid Search on lamda for lasso regression--------------###
 ## Do cross validation to generate best lamda
load("adult.RData")
x_train.lasso = model.matrix(y~., ad.train)[,-1]
x_test.lasso = model.matrix(income~., ad.test)[,-1]
y_train.lasso=ad.train$y
y_test.lasso=ad.test$income
set.seed(1234)
cv.out = cv.glmnet(x_train.lasso, y_train.lasso,family="binomial", alpha = 1,
                standardize = FALSE ,lambda = NULL, type.measure = "mse")
# Fit lasso model on training data
plot(lasso_mod)    #   Draw plot of coefficients
#min value of lambda
lambda_min = cv.out$lambda.min
#best value of lambda
lambda_1se = cv.out$lambda.1se
#regression coefficients
coef(cv.out,s=lambda_1se)
#get final model
fit_lasso=glmnet(x_train.lasso, y_train.lasso,family="binomial",
              standardize = FALSE,lambda = lambda_min)
#predict class, type=â??classâ??
lasso_prob = predict(cv.out,newx = x_test.lasso,s=lambda_1se,type="response")
#translate probabilities to predictions
lasso_predict <- rep("0",nrow(ad.test))
```

```
lasso_predict[lasso_prob>.5] = "1"
#confusion matrix
table(pred=lasso_predict,true=ad.test$income)
#accuracy
mean(lasso_predict==ad.test$income)
## ROC curve
roccurve = roc(response = as.double(ad.test$income),predictor=lasso_prob,auc=TRUE,plot=TRUE)
cat("the auc is:",roccurve$auc,"\n")
source("http://www.rob-mcculloch.org/2019_ml/webpage/notes/lift-loss.R")
lift.plot(lasso_prob,ad.test$income)
#-------------------------------------------------------------------------------------------
library(randomForest) # basic implementation
library(ranger)      # a faster implementation of randomForest
library(caret)       # an aggregator package for performing many machine learning models
library(h2o)
library(e1071)
library(doParallel)
library(parallel)
library(doSNOW)
library(snow)
#----------**********  grid search for random forest **********----------#
levels(ad.train$y) = c("low", "high")

# Define the control
trControl <- trainControl(method = "cv",
                          number = 10,
                          savePredictions = 'final',
                          classProbs = T,
                          summaryFunction=twoClassSummary,
                          search = "grid")
set.seed(123)
# Run the model
rf_default <- train(y~.,
                    data = ad.train,
                    method = "rf",
                    metric = "ROC",
                    trControl = trControl)
# Print the results
print(rf_default)
#The algorithm uses 500 trees and tested three different values of mtry: 2, 6, 11.
#The final value used for the model was mtry = 6 with an ROC of 0.9139765. Let's try to get a higher score.

# Step (1) Search best mtry
## test the model with values of mtry from 1 to 10
set.seed(123)
tuneGrid <- expand.grid(.mtry = c(1: 10))
rf_mtry <- train(y~.,
```

```
            data = ad.train,
            method = "rf",
            metric = "ROC",
            tuneGrid = tuneGrid,
            trControl = trControl,
            importance = TRUE,
            nodesize = 14,
            ntree = 300)
print(rf_mtry)
#The final value used for the model was mtry = 6.
best_mtry = rf_mtry$bestTune$mtry # store best mtry
print(best_mtry) # =6

#Step (2) Search the best maxnodes

store_maxnode = list()
tuneGrid = expand.grid(.mtry = best_mtry)
#store_maxnode <- list(): The results of the model will be stored in this list
#expand.grid(.mtry=best_mtry): Use the best value of mtry
#for (maxnodes in c(15:25)) { ... }: Compute the model with values of maxnodes starting from 15 to 25.

for (maxnodes in c(5: 15)) {
 set.seed(123)
 rf_maxnode = train(y~.,
                    data = ad.train,
                    method = "rf",
                    metric = "ROC",
                    tuneGrid = tuneGrid,
                    trControl = trControl,
                    importance = TRUE,
                    nodesize = 14,
                    maxnodes = maxnodes,
                    ntree = 300)
 current_iteration = toString(maxnodes)
 store_maxnode[[current_iteration]] = rf_maxnode
}
results_mtry = resamples(store_maxnode)
summary(results_mtry)
## from output out highest ROC when maxnodes=7, so we don't need to expand the range of maxnodes

#Step 4) Search the best ntrees
store_maxtrees = list()
for (ntree in c(250, 300, 350, 400, 450, 500, 550, 600, 800, 1000, 2000)) {
 set.seed(1234)
 rf_maxtrees = train(y~.,
                     data = ad.train,
                     method = "rf",
```

```
                metric = "ROC",
                tuneGrid = tuneGrid,
                trControl = trControl,
                importance = TRUE,
                nodesize = 14,
                maxnodes = 7,
                ntree = ntree)
 key <- toString(ntree)
 store_maxtrees[[key]] <- rf_maxtrees
}
results_tree = resamples(store_maxtrees)
summary(results_tree)
## ntree=800 comes with largest ROC
#Finally, we got the best model with maxnodes=7, ntree=800, mtry=4

## Final model
fit_rf = train(y~.,
          ad.train,
          method = "rf",
          metric = "ROC",
          tuneGrid = tuneGrid,
          trControl = trControl,
          importance = TRUE,
          nodesize = 14,
          ntree = 800,
          maxnodes = 7)
print(fit_rf)
cat("the auc is:",fit_rf$results[,2],"\n")
#Step 5) Evaluate the final model

# (a) Plot ROC
levels(ad.test$income) = c("low", "high")
pred_rf_class= predict(fit_rf, ad.test)
pred_rf = predict(fit_rf, newdata = ad.test, type = "prob")
RF = prediction(pred_rf[,1], ad.test$income)
RF = performance(RF, "tpr", "fpr")
plot(RF, main = "ROC curves for randomForest")
library(randomForest)

#(b) plot variable importance
varimp_rf = varImp(fit_rf)
plot(varimp_rf, main="Variable Importance with rf")
print(varimp_rf)

# (c) Compute the confusion matrix,
# and get Accuracy for out best model, we have overall accuarcy of 83.08%
library(ROCR)
```

```
prediction =predict(fit_rf, ad.test)
confusionMatrix(prediction, ad.test$income)
#----------------------  Stochastic Gradient Boosting  ---------------------------------------------#
# for reproducibility
library(gbm)         # basic implementation
library(xgboost)     # a faster implementation of gbm
library(caret)       # an aggregator package for performing many machine learning models
library(h2o)         # a java-based platform
library(pdp)         # model visualization
library(ggplot2)     # model visualization
library(lime)        # model visualization
# train GBM model
set.seed(1234)
library(gbm)
fitControl <- trainControl(## 10-fold CV
  method = "repeatedcv",savePredictions = 'final',
  classProbs = T,
  summaryFunction=twoClassSummary,
  search = "grid",
  number = 10,
  ## repeated ten times
  repeats = 10)
## this time, I don't want to pick one by one, Let's do a hyperparameter optimization(grid search)
gbmGrid =  expand.grid(interaction.depth = c(1, 5, 9),
                       n.trees = (1:30)*50,
                       shrinkage = c(0.1,0.01),
                       n.minobsinnode = 20)

set.seed(1234)
gbmFit2 = train(y ~ ., data = ad.train,
                method = "gbm", metric='ROC',
                trControl = fitControl,
                verbose = FALSE,
                tuneGrid = gbmGrid)
## After waiting for over one hour, we got the final model with n.tree=1500,
## depth=9, shrinkage=0.01.

## Final model
set.seed(1234)
boost_fit = train(y ~ ., data = ad.train,
                method = "gbm", metric='ROC',
                trControl = fitControl,
                verbose = FALSE,
                tuneGrid = data.frame(interaction.depth = 9,
                                      n.trees = 1500,
                                      shrinkage = .01,
                                      n.minobsinnode = 20))
```

```
print(boost_fit)
cat("the auc is:",boost_fit$results[,5],"\n")
#Step 5) Evaluate the final model

# (a) Plot ROC
pred_boost_class= predict(boost_fit, ad.test)
pred_boost = predict(boost_fit, newdata = ad.test, type = "prob")
GB = prediction(pred_boost[,1], ad.test$income)
GB = performance(GB, "tpr", "fpr")
plot(GB, main = "ROC curves for Boosting")
library(randomForest)

#(b) plot variable importance
varimp_boost = varImp(boost_fit)
plot(varimp_boost, main="Variable Importance with boosting")
print(varimp_boost)

# (c) Compute the confusion matrix,
# and get Accuracy for our best model, we have overall accuarcy of 83.08%
prediction =predict(boost_fit, ad.test)
confusionMatrix(prediction, ad.test$income)
#----------------------Support Vector Machines(with grid search) ----------------------------
library(e1071)
# svm requires tuning
svm.tune <- tune(svm, y~., data = ad.train,
                 ranges = list(gamma = 2^(-8:1), cost = 2^(0:4)),
                 tunecontrol = tune.control(sampling = "fix"))

svm.tune
## best papramater is cost=1,gamma=0.03125
## Here comes our final model
svm_fit = svm(y~., data = ad.train, cost=1, gamma=0.03125, probability = TRUE)
## Prediction
svm.pred = predict(svm_fit, type="prob", newdata=ad.test, probability = TRUE)
confusionMatrix(svm.pred, ad.test$income)
require(ROCR)
svm.prob.rocr = prediction(attr(svm.pred, "probabilities")[,1], ad.test$income)
svm.perf = performance(svm.prob.rocr, "tpr","fpr")
plot(svm.perf,main = "SVM ROC curve", colorize = T)
abline(a = 0, b = 1)
auc_ROCR = performance(svm.prob.rocr, measure = "auc")
auc_ROCR = auc_ROCR@y.values[[1]]
cat("the auc is:",auc_ROCR,"\n")
```