CYOincome

Jin Xin

Oct, 2022

Executive Summary

This report is going to try different methods to achieve a good F_1 score on the target "income" class among all the surveyed candidates. The used Adult Income Census dataset has two income classes: "<= 50K" and ">50K". The goal is trying to predict all the ">50K" income samples in the testset, while the challenge is coming from the class dominance of "<=50K", what makes our target ">50K" class to be severely short-sampled, so it is hard to achieve a good prediction accuracy for the target ">50K" income class. The methods used in this report include CART (classification and regression tree), random forest, SVM (support vector machine), and logistic regression. The best performing individual method is random forest with a F_1 score of 0.6908837. In addition, the stacking method is tested to combine the tuned random forest, SVM, and GLM, and the achieved F_1 score is 0.6879334. The results suggest the random forest is the best performing method among all the individual and ensemble methods tested on the dateset.

```
#libraries are used in this report
if(!require(tidyverse))
  install.packages("tidyverse")
if(!require(data.table))
  install.packages("data.table")
if(!require(caret))
  install.packages("caret")
if(!require(doParallel))
  install.packages("doParallel")
if(!require(e1071))
  install.packages("e1071")
if(!require(caretEnsemble))
  install.packages("caretEnsemble")
if(!require(GGally))
  install.packages("GGally")

library(tidyverse)
library(data.table)
library(caret)
library(doParallel)
library(e1071)
library(caretEnsemble)
library(GGally)

#dataset can be downloaded from the following links
#the downloaded file is a zip
#www.kaggle.com/datasets/uciml/adult-census-income
#or
```

```
#github.com/dsjinx/harvardx-ds-capstone/blob/main/CYO/AdultCensusIncome.zip

#copy the downloaded zip file into your current working directory
getwd()
#read the dataset in the environment, name it as "data"
data <- fread(unzip("AdultCensusIncome.zip"))
```

Introduction

We are going to apply different but not exhaustive classification methods in this report. The F_1 score is used as evaluation metric, the reason is coming from the unbalanced sample size in the dataset. The Adult Income Census datast we use in this report classifies the income levels into two groups, one is "<=50K", and the other is ">50K", but the ">50K" class has much smaller sample size comparing to "<=50" class. According to Nielsen (2015) and many practical machine learning texts, increased sample size can effectively result a good learning performance, but on the contrary a small sample size can hinder the model performance. So we choose to evaluate the candidate methods by the balanced F_1 score, which can give us a good evaluation over the sensitivity and specificity of different models on the target ">50K" income class. This feature of F_1 score makes it a rather effective metric comparing to the overall accuracy, which can falsely report us a high value, due to the high accuracy contribution from the big sample size in "<=50K" class.

The following content is broken into three parts. In the data exploration, we will explore the data in a top-down strategy, by having a broad overview of the dataset structure first, then break the attribute predictors into groups according to the type of the element values, and look for any trend or relations across those attributes. After that, we will get into the methods in details, we justify the logic behind choosing those candidate methods to be tested in this report. Lastly, we will evaluate the overall performances across different methods applied, and discuss about what other possible approaches may be taken to improve the performance in further.

Data Exploration

The Adult Census Income dataset used in this report is from the 1994 Census bureau database. This dataset has the size of 32561 by 15. It has samples in rows, and measured variables in columns, so every single sample has 14 different features, plus one income class information, and there are 32561 samples in total.

```
## Classes 'data.table' and 'data.frame':   32561 obs. of  15 variables:
##  $ age           : int  90 82 66 54 41 34 38 74 68 41 ...
##  $ workclass     : chr  "?" "Private" "?" "Private" ...
##  $ fnlwgt        : int  77053 132870 186061 140359 264663 216864 150601 88638 422013 70037 ...
##  $ education     : chr  "HS-grad" "HS-grad" "Some-college" "7th-8th" ...
##  $ education.num : int  9 9 10 4 10 9 6 16 9 10 ...
##  $ marital.status: chr  "Widowed" "Widowed" "Widowed" "Divorced" ...
##  $ occupation    : chr  "?" "Exec-managerial" "?" "Machine-op-inspct" ...
##  $ relationship  : chr  "Not-in-family" "Not-in-family" "Unmarried" "Unmarried" ...
##  $ race          : chr  "White" "White" "Black" "White" ...
##  $ sex           : chr  "Female" "Female" "Female" "Female" ...
##  $ capital.gain  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ capital.loss  : int  4356 4356 4356 3900 3900 3770 3770 3683 3683 3004 ...
##  $ hours.per.week: int  40 18 40 40 40 45 40 20 40 60 ...
##  $ native.country: chr  "United-States" "United-States" "United-States" "United-States" ...
##  $ income        : chr  "<=50K" "<=50K" "<=50K" "<=50K" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

The attributes are measured both quantitatively and qualitatively with respect to numeric/integer value columns and character value columns. The dependent variable "income" is what we are trying to predict, and it is in a qualitative form with only two different categories: "<=50K" and ">50K". This two classes element structure can help us narrow down the possible methods used to be either producing "YES"/"NO" results, or giving the probability toward each category.

```
##            age      workclass         fnlwgt      education  education.num
##      "integer"    "character"      "integer"    "character"      "integer"
## marital.status     occupation   relationship           race            sex
##    "character"    "character"    "character"    "character"    "character"
##   capital.gain   capital.loss hours.per.week native.country         income
##      "integer"      "integer"      "integer"    "character"    "character"
```

Data Visualisation

Before we explore the data in further details, we want to check on the dataset for the strange and missing values. There is no "NA" value in the dataset, but we can see some "?" showing in "workclass", "occupation", and "native.country" as in below table. They are all in the qualitative attributes columns, and recorded in character string. However we will decide the way to handle them, when we explore these attributes in details.

```
sum(is.na(data))
```

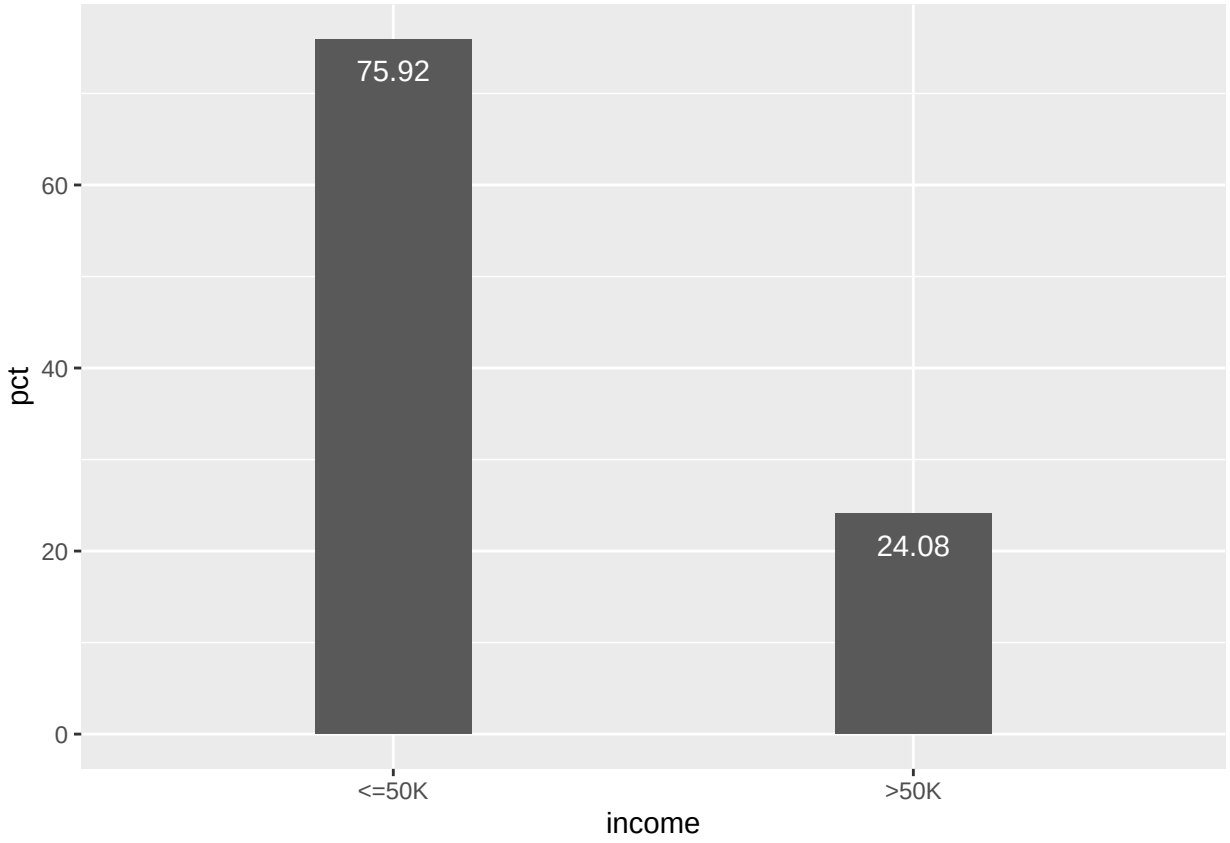
```
## [1] 0
```

vars	num of ?
age	0
workclass	1836
fnlwgt	0
education	0
education.num	0
marital.status	0
occupation	1843
relationship	0
race	0
sex	0
capital.gain	0
capital.loss	0
hours.per.week	0
native.country	583
income	0

After an initial glimpse of the dateset, we now get into the details about the dependent variable "income" column, and the other 14 attribute columns. For the "income" column, all the samples are classified into two categories: "<=50K" and ">50K". By checking the distribution across those two income levels, we can see that the income level of "<=50K" is the dominant group across all the samples, which has the size of being 3 times of the ">50K" group, and only about 24% of the samples are in ">50K" group.

```
sum(data$income == "<=50K") / sum(data$income == ">50K")
```

```
## [1] 3.152659
```



This unbalanced sample size can potentially make it more challenging to achieve a high accuracy on predicting the ">50K" income group, comparing to the "<=50K" ones. The reason is that, we have more samples to train for the "<=50K" group, but the ">50K" group is short-sampled, so if the two different group samples are not separately clustered, or lack of strong manifestations with the 14 attributes, then we can only reply on minimising the error on "<=50K" group to improve the ">50K" group accuracy. On the other hand, this could be viewed as a benefit of having only two groups of dependents to work with.

As mentioned in above, there are two different classes of predictors: "integer" and "character", and we are going to look those two classes of attributes separately. We will look the "integer" class first.

```
#collect the numeric cols names
cols <- names(which(sapply(data, class) == "integer"))
```

```
summary(data[, ..cols]) #check any strange value
```

```
##       age            fnlwgt        education.num    capital.gain
##  Min.   :17.00   Min.   :  12285   Min.   : 1.00   Min.   :    0
##  1st Qu.:28.00   1st Qu.: 117827   1st Qu.: 9.00   1st Qu.:    0
##  Median :37.00   Median : 178356   Median :10.00   Median :    0
##  Mean   :38.58   Mean   : 189778   Mean   :10.08   Mean   : 1078
##  3rd Qu.:48.00   3rd Qu.: 237051   3rd Qu.:12.00   3rd Qu.:    0
##  Max.   :90.00   Max.   :1484705   Max.   :16.00   Max.   :99999
##   capital.loss    hours.per.week
##  Min.   :   0.0   Min.   : 1.00
```

```
##  1st Qu.:   0.0   1st Qu.:40.00
##  Median :   0.0   Median :40.00
##  Mean   :  87.3   Mean   :40.44
##  3rd Qu.:   0.0   3rd Qu.:45.00
##  Max.   :4356.0   Max.   :99.00
```

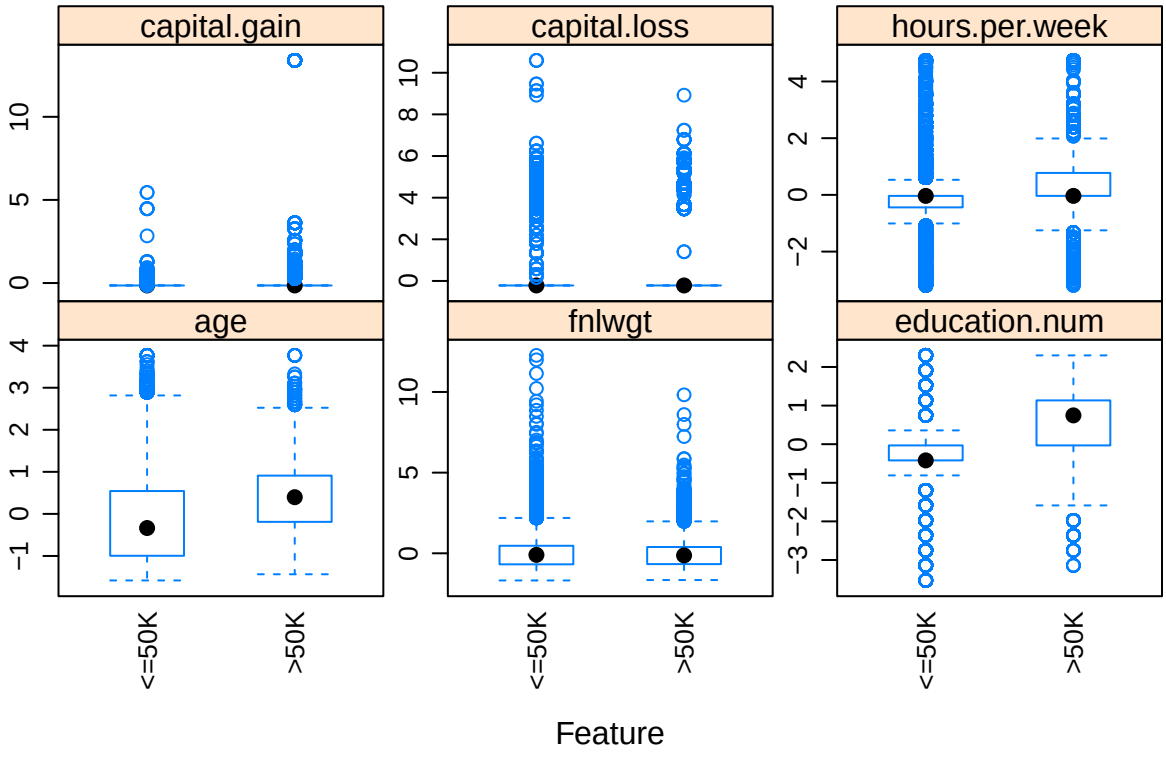
The above lot of "integer" columns summary is not showing any strange numbers, except the "capital.gain" and "capital.loss" columns have 0 values for most of their statistics. It makes sense that if most people are earning less than 50K/year, then most of them won't have any investment asset in place. Apart from this, the column "fnlwgt" we need to make some clarification about its meaning in here. It is stand for "final weight", simply speaking it represents the amount of people is sharing the same attributes of the row, where that "fnlwgt" value is. Of course the census bureau has more rigorous explanation on it, but we just want to highlight its basic meaning to avoid the ambiguity caused by the "weight" in its name.

```
num_fp <- data[, ..cols][, lapply(.SD,
              function(j) (j - mean(j)) / sd(j)), .SDcols = cols]

summary(num_fp) #check potential outliers by normalising the columns
```

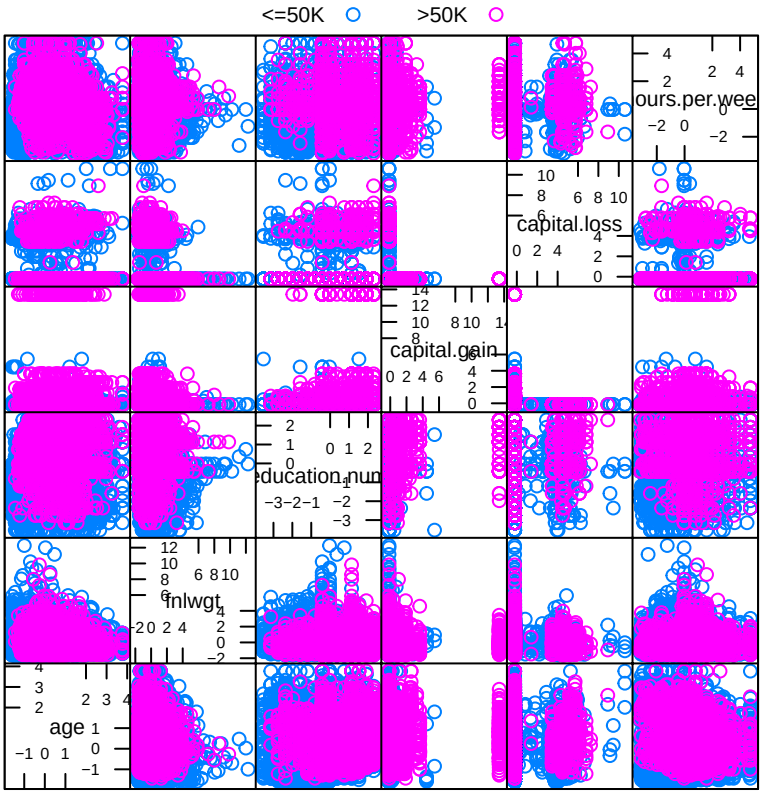
```
##       age              fnlwgt         education.num       capital.gain
##  Min.   :-1.5822   Min.   :-1.6816   Min.   :-3.52960   Min.   :-0.1459
##  1st Qu.:-0.7758   1st Qu.:-0.6817   1st Qu.:-0.42005   1st Qu.:-0.1459
##  Median :-0.1160   Median :-0.1082   Median :-0.03136   Median :-0.1459
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000
##  3rd Qu.: 0.6905   3rd Qu.: 0.4479   3rd Qu.: 0.74603   3rd Qu.:-0.1459
##  Max.   : 3.7696   Max.   :12.2684   Max.   : 2.30080   Max.   :13.3944
##   capital.loss     hours.per.week
##  Min.   :-0.2167   Min.   :-3.19398
##  1st Qu.:-0.2167   1st Qu.:-0.03543
##  Median :-0.2167   Median :-0.03543
##  Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.:-0.2167   3rd Qu.: 0.36951
##  Max.   :10.5933   Max.   : 4.74289
```

After we normalise all the "integer" columns, we get a much clearer picture about how far away those outliers are. The boxplots in below illustrate our findings in further details. All of these six numeric attribute columns have many outliers indeed, and majority of the outliers are locating in the upper extreme, except for the "weekly working hour" and "number of years education taken", the outliers in those two attributes are locating in both upper and lower extremes. By comparing the distributions between "<=50K" on the left and ">50K" on the right, the age attribute is showing a reasonable distribution to illustrate that, the strong earning group's age range are a bit older than "<=50K" group, but the upper bound age is lower than "<=50K" group, the logic behind might be the ">50K" group may retire at a comparable younger age than the "<=50K" group. Meanwhile, the distribution of years of education is telling us that, the extra number of years we invest in study have very high possibility to reward us a good income in later days. At the same time, the weekly working hour shows the hard work pays off as in everybody's belief. Next we will look into the relations in between those numeric attributes under different income groups.



By inspecting the scatter plots matrix in below, none of the pairs out of those six numeric attributes are showing any clear trend or correlations, instead all the pairs are really just random clusters. And this kind of non-correlated randomness are shared in both "<=50k" and ">50K" income groups. The only meaningful find out in those pairs is related to the "capital gain". For all the five pairs boxes between the "capital gain" and the other 5 attributes, we can see all the top valued in "capital gain" points are pink, which represent the ">50K" high income group.

(to be continued on next page)



Scatter Plot Matrix

We now turn to the character class attributes. But before we move on, we need to decide how we are going to treat those "?" from the exploration in above. Firstly, we want to have a general idea about how much portion of our dataset is "?", because from the size of them, we can evaluate the potential harm of having them in place, then we decide how they could be treated to fit in our models later.

```
#collect all the char columns for future process
char_cols <- names(data)[-which(names(data) %in% cols)][-9]
```

```
###check the missing ?
sum(data == "?") / (dim(data)[1] * dim(data)[2])
```

```
## [1] 0.008726186
```

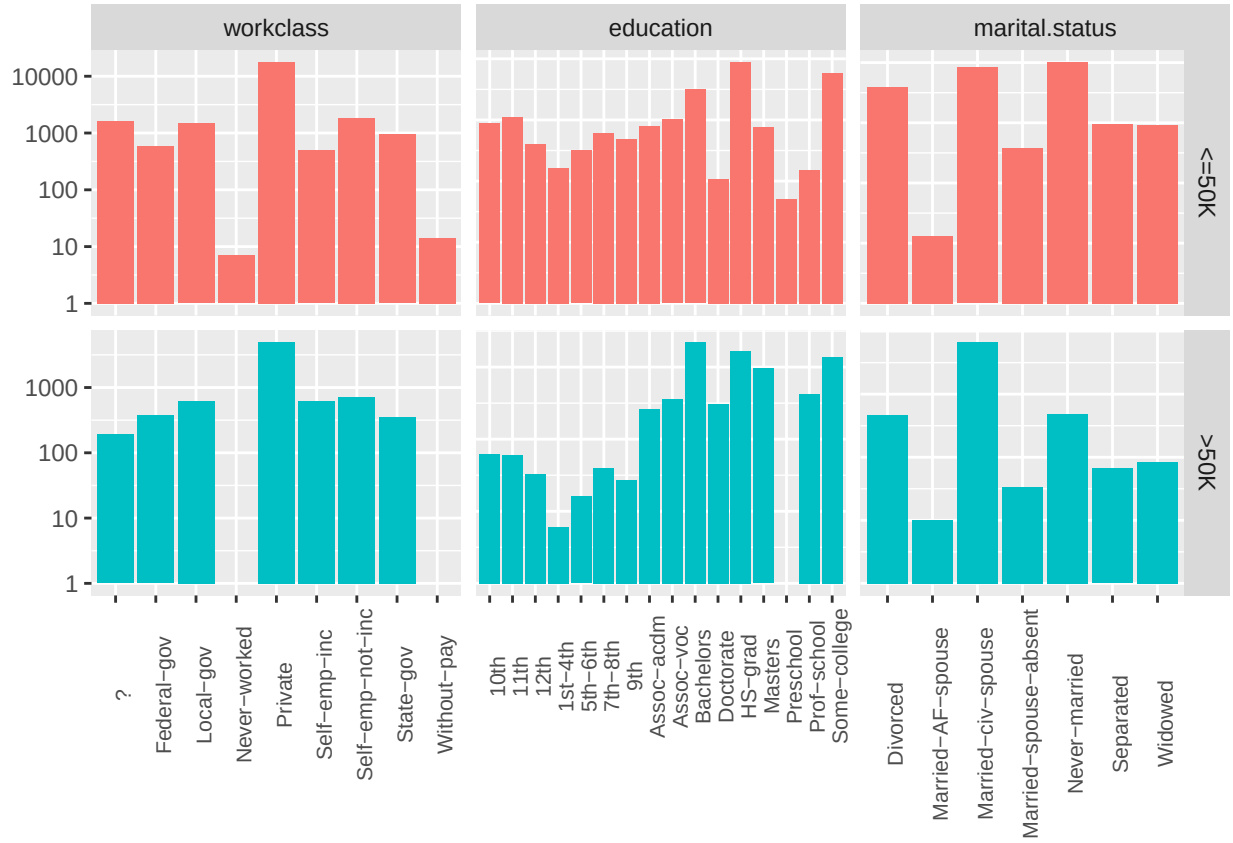
```
mis_locate <- data[, lapply(.SD, function(i) sum(i == "?")),
                   .SDcols = char_cols]
data.frame(var.name = names(mis_locate),
           pct.in.data = as.numeric(100 * mis_locate / dim(data)[1])) %>%
           knitr::kable() #total % of each col elements are "?"
```

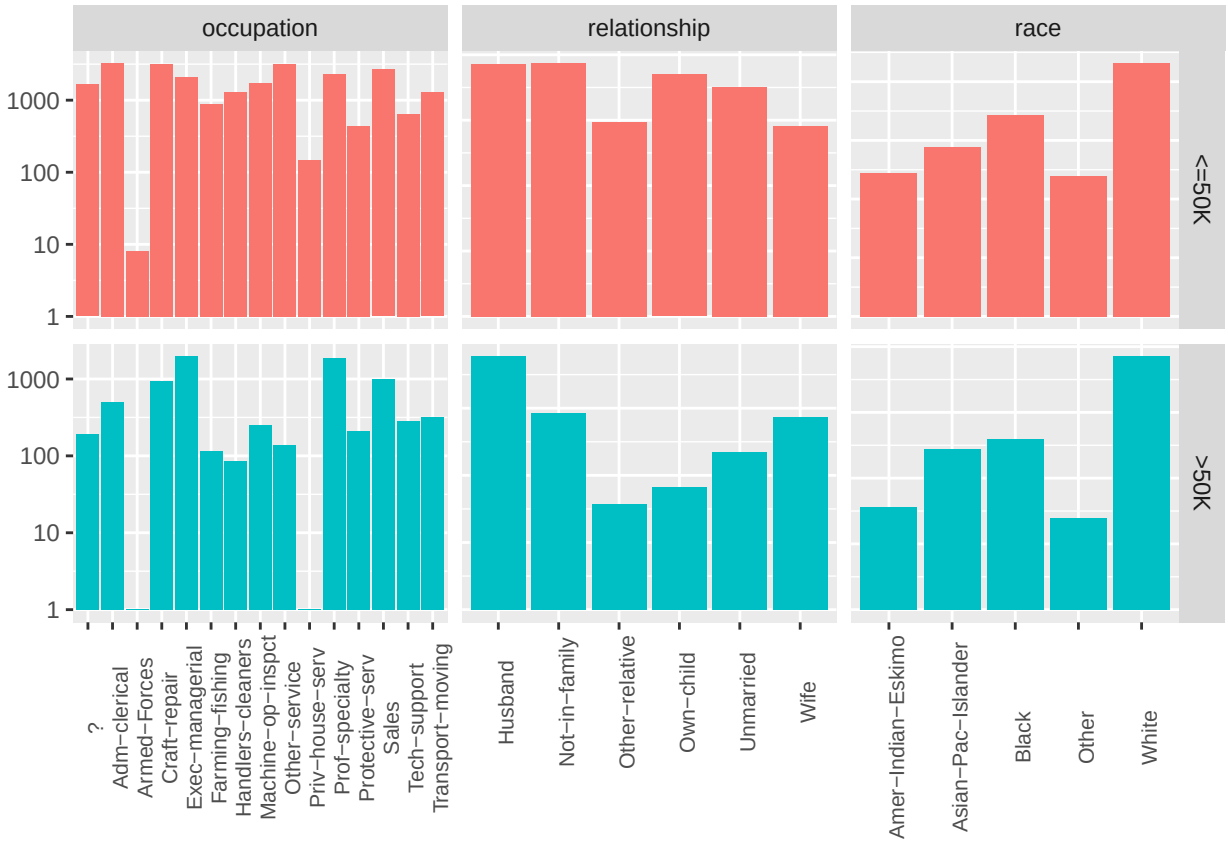
var.name	pct.in.data
workclass	5.638648
education	0.000000
marital.status	0.000000
occupation	5.660146

var.name	pct.in.data
relationship	0.000000
race	0.000000
sex	0.000000
native.country	1.790486

The above figures show that the "?" issue is not major, which is only 0.87% out of the whole dataset. In the column wise, we have seen the "?" only exists in three columns, and only "work class" and "occupation" have 5% of the data are recorded in "?", the "native.country" has less than 2% out of the total. So those "?" values are too small to cause our dataset being sparse in the whole. Also the good thing about the locations of them is that, none of the "?" is in the dependent "income" column, that means all the rows are valid samples can be classified. The other good thing is that, they are all categorical data, if we use a tree method to train them, they will not cause any problem in those kind of "YES"/"NO" logical rules. So we will leave them in place for now, and only treat them on an ad hoc basis.

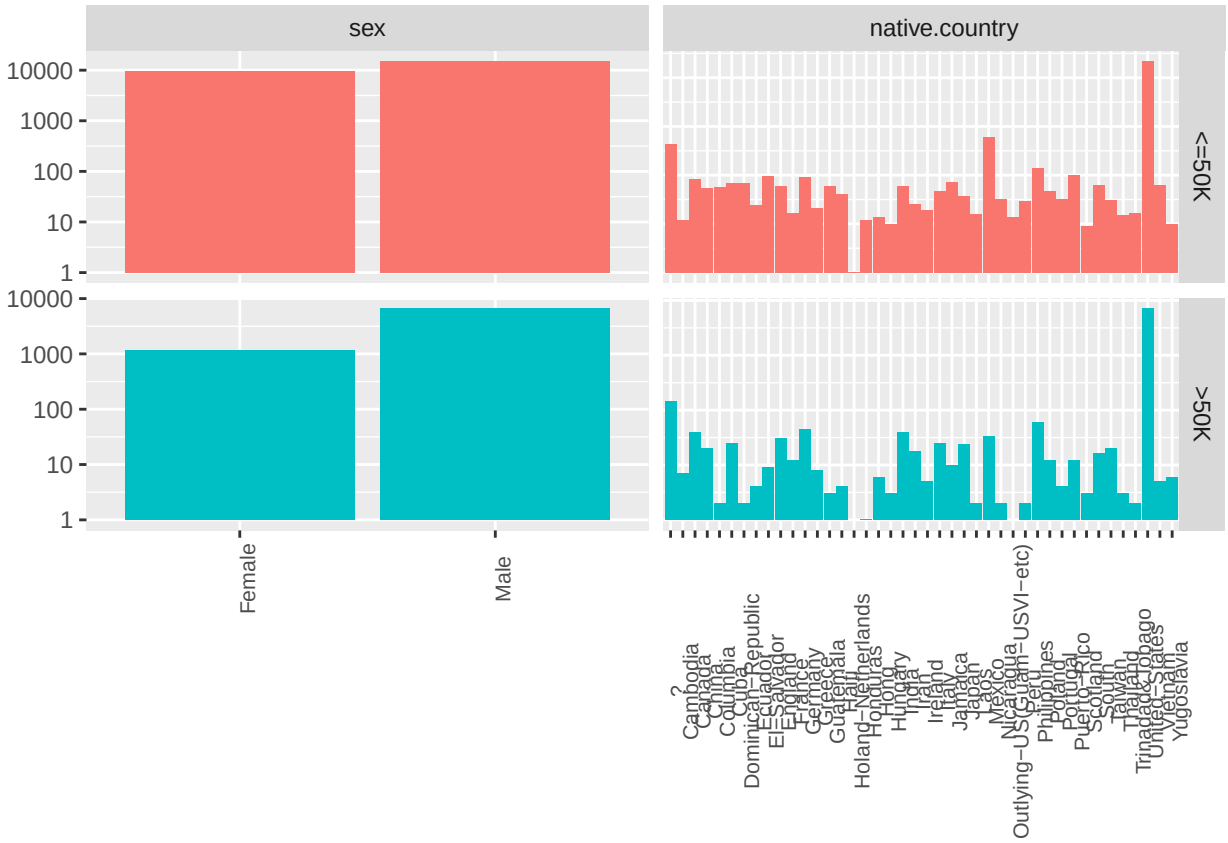
From the following block of bar plots, we can have a visual understanding of what the major class is in each attribute, that makes a surveyed sample to be a ">50K" income earner. The way to visually decide a major driver of one income group is that, if there is big gap for a same attribute value showing across the two income groups. For example, the "workclass" attribute is clearly showing that the "never worked" and "without pay" are the two major attribute elements can help distinguish the "<=50K" from the ">50K" income group. The "education" attribute has a very interesting find out, it shows that those highly educated candidates with degrees in bachelor, master, and doctor are not primarily clustered in high income group, instead there is quite a lot of them are in "<=50K" income group. That is very counter intuitive with the findings from the education years in above numeric attributes.





We can also see some absolute differences between the two groups. For "workclass", the "never worked" and "without pay" are totally missing in the high income group for the obvious reason. As well as for the missing "pre-school" class in "education". The missing "armed-force" and "private-house-server" of "occupation" in high income group is also telling us those jobs are not likely to pay us great. Apart from that, the attributes allocation between the two income groups are quite similar in general.

(to be continued on next page)



So after going through those numeric and categorical attributes visually, at least by human eyes, we can not set any model up to classify the two income groups effectively. But at least, these visual information can help us choose the possible methods, which could be effective to handle those not strongly featured data.

Methods Details

After the above visual understanding of the dataset, straightaway we can narrow down the methods options down to trees, because the tree family can effectively handle different classes of variables, no matter they are numbers or character strings. So we will use a single tree method first, then use a forest method to compare how much better the result can be. But before everything, we will utilise the dominant class phenomenon in the dependent "income" column, to make predictions purely on guessing, we can use this dummy algorithm result as our benchmark to measure the other attending methods performances.

0. Guessing

We are breaking the original dataset into train/test set by a 8/2 ratio first, and use the prevalence of the dominant income group of "<=50" in the training set to toss a highly biased coin with two faces of "<=50K" and ">50K", we will toss the coin for the number of times equal to the number of samples we are guessing in the test set.

```
#break data into 8/2 train/test set
set.seed(1, sample.kind = "Rounding")
ind <- createDataPartition(1:dim(data)[1], times = 1, list = FALSE, p = 0.2)
train <- data[-ind, ]
test <- data[ind, ]
```

```r
#0. guessing
#chance of seeing <=50k in training set
prev <- sum(train$income == "<=50K") / dim(train)[1]

#use the "<=50k" chance to make a biased coin
#toss the coin for dim(test)[1] number of times
set.seed(10, sample.kind = "Rounding")
pred_guess <- sample(c("<=50K", ">50K"), dim(test)[1], replace = TRUE,
                     prob = c(prev, 1 - prev))
gauge_guess <- confusionMatrix(as.factor(pred_guess), test$income,
                               mode = "prec_recall",
                               positive = ">50K")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction <=50K >50K
##      <=50K  3702 1187
##      >50K   1229  395
##
##                Accuracy : 0.629
##                  95% CI : (0.6172, 0.6408)
##     No Information Rate : 0.7571
##     P-Value [Acc > NIR] : 1.0000
##
##                   Kappa : 4e-04
##
##  Mcnemar's Test P-Value : 0.4042
##
##               Precision : 0.24323
##                  Recall : 0.24968
##                      F1 : 0.24641
##              Prevalence : 0.24290
##          Detection Rate : 0.06065
##    Detection Prevalence : 0.24935
##       Balanced Accuracy : 0.50022
##
##        'Positive' Class : >50K
##

##       F1
## 0.246413
```

We get a F_1 score of 0.246413 by tossing the biased coin, and this will be as other methods performance benchmark, so we are expecting all the selected algorithms in below to have a substantial improvement over this baseline. By looking at the above performance stats of guessing, it proves that the accuracy is a highly biased performance metric with the value of 0.629, that is the consequence of dominant income group in both train and test dataset. The kappa value of 4e-04 is telling us this accuracy is nothing different from coincidence. If we guess everyone is "<=50K", we get an even higher accuracy of 0.7571012.

```r
sum(test$income == "<=50K") / dim(test)[1]
```

```
## [1] 0.7571012
```

1. One tree

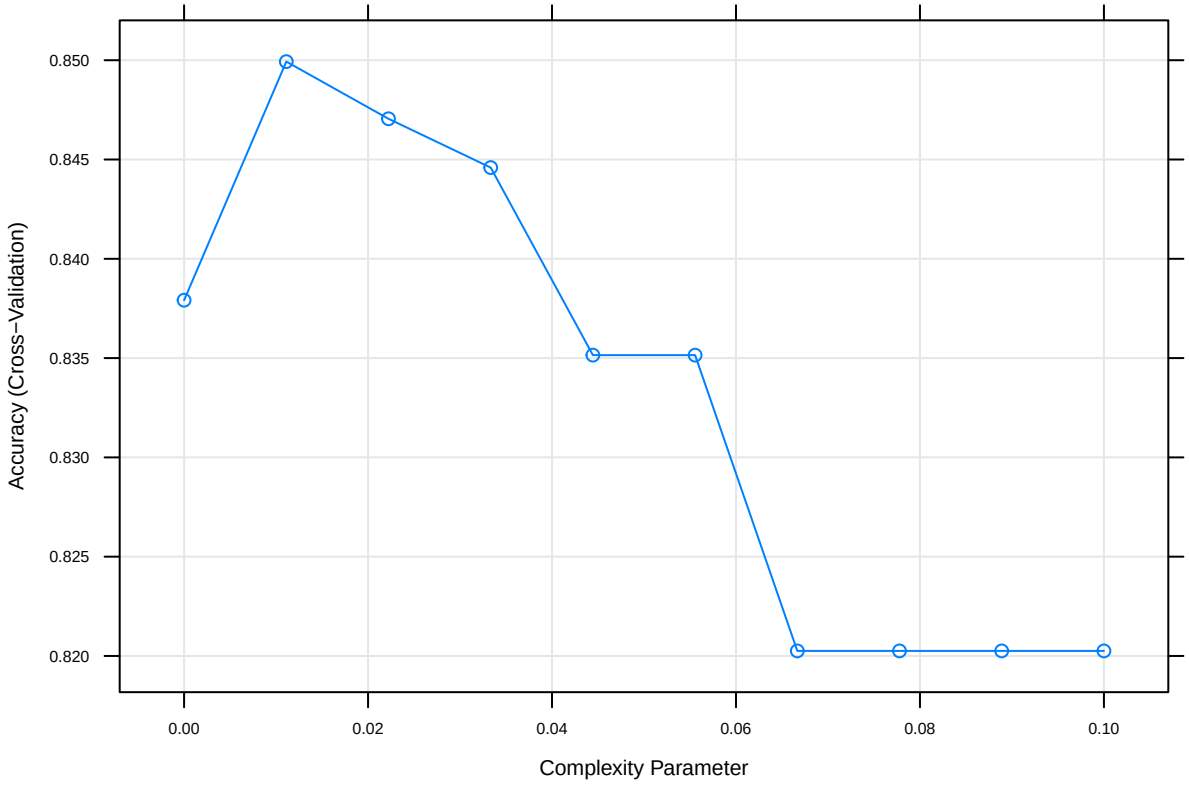
Now we are turning to the property algorithms. We will start with a single CART tree by using the "rpart" package. According to the package vignettes, there are a list of parameters can be tuned to affect the model performance, but the most effective one is complexity parameter "cp" (Therneau and Atkinson 2022). It is the parameter to decide how the tree is going to be pruned from the default tree, which is grown by the set of system default parameters. The "cp" value is meant to be in the range of [0, 1], when "cp" = 1 then the tree could be a stick with no branches at all, what can be defined as the case of under training. When the "cp" = 0 could cause the tree to have the number of branches to be the total number of elements of the whole dataset - 1, alternatively it is to be said the model is very over trained. We then need to find the best value to fit our purpose. For the minimum number of samples to create more branches split, and the minimum samples to be in a terminal node, we will just use the default values for them. The "raprt" package is made to be able to handle missing values with the surrogate parameters, meanwhile we have a small fraction of predictors are missing and recorded in "?", but we will arbitrarily treat them as a valid class value, since they are all recorded as character strings inside the character attributes columns, and they can only be possibly used for creating some "YES/NO" based classification rules, without any numerical calculation on them.

```
registerDoParallel(cores = 4)
#caret can parallel the process to speed up the training
#set the number of cores according to your own hardware situation
#otherwise it can crash your system

set.seed(2, sample.kind = "Rounding")
ind_cv <- createFolds(1:dim(train)[1], k = 5, returnTrain = TRUE)

#define the cross validation index to make the result reproducible
treecontrol <- trainControl(method = "cv", index = ind_cv)
fit_tree <- train(income ~ ., data = train, method = "rpart",
                  trControl = treecontrol,
                  tuneGrid = data.frame(cp = seq(0, 0.1, length.out = 10)))
```

(to be continued on next page)




```
## n= 26048
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 26048 6259 <=50K (0.75971284 0.24028716)
##    2) relationship< 4.5 14263  962 <=50K (0.93255276 0.06744724)
##      4) capital.gain< 7073.5 14001  708 <=50K (0.94943218 0.05056782) *
##      5) capital.gain>=7073.5 262    8 >50K (0.03053435 0.96946565) *
##    3) relationship>=4.5 11785 5297 <=50K (0.55053034 0.44946966)
##      6) education.num< 12.5 8244 2738 <=50K (0.66787967 0.33212033)
##       12) capital.gain< 5095.5 7840 2342 <=50K (0.70127551 0.29872449)
##         24) capital.loss< 1782.5 7551 2128 <=50K (0.71818302 0.28181698) *
##         25) capital.loss>=1782.5 289   75 >50K (0.25951557 0.74048443) *
##       13) capital.gain>=5095.5 404    8 >50K (0.01980198 0.98019802) *
##      7) education.num>=12.5 3541  982 >50K (0.27732279 0.72267721) *
```

The cross validation results suggest when "cp" = 0.01111111, the model can provide us a favourable result. From the printing of the final model, it shows our tuned tree has six leaf nodes or terminal nodes marked by the $*$. In those leafs we can see that the majority of the classifiers are the capital gain and capital loss, which just coincide with our visual findings in above.

```
#use the tuned tree to predict the test set
pred_tree <- predict(fit_tree, test)
gauge_tree <- confusionMatrix(pred_tree, test$income,
```

```
                                mode = "prec_recall",
                                positive = ">50K")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction <=50K >50K
##      <=50K  4657  714
##      >50K    274  868
##
##                Accuracy : 0.8483
##                  95% CI : (0.8394, 0.8569)
##     No Information Rate : 0.7571
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5445
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.7601
##                  Recall : 0.5487
##                      F1 : 0.6373
##              Prevalence : 0.2429
##          Detection Rate : 0.1333
##    Detection Prevalence : 0.1753
##       Balanced Accuracy : 0.7466
##
##        'Positive' Class : >50K
##
```

```
##        F1
## 0.6372981
```

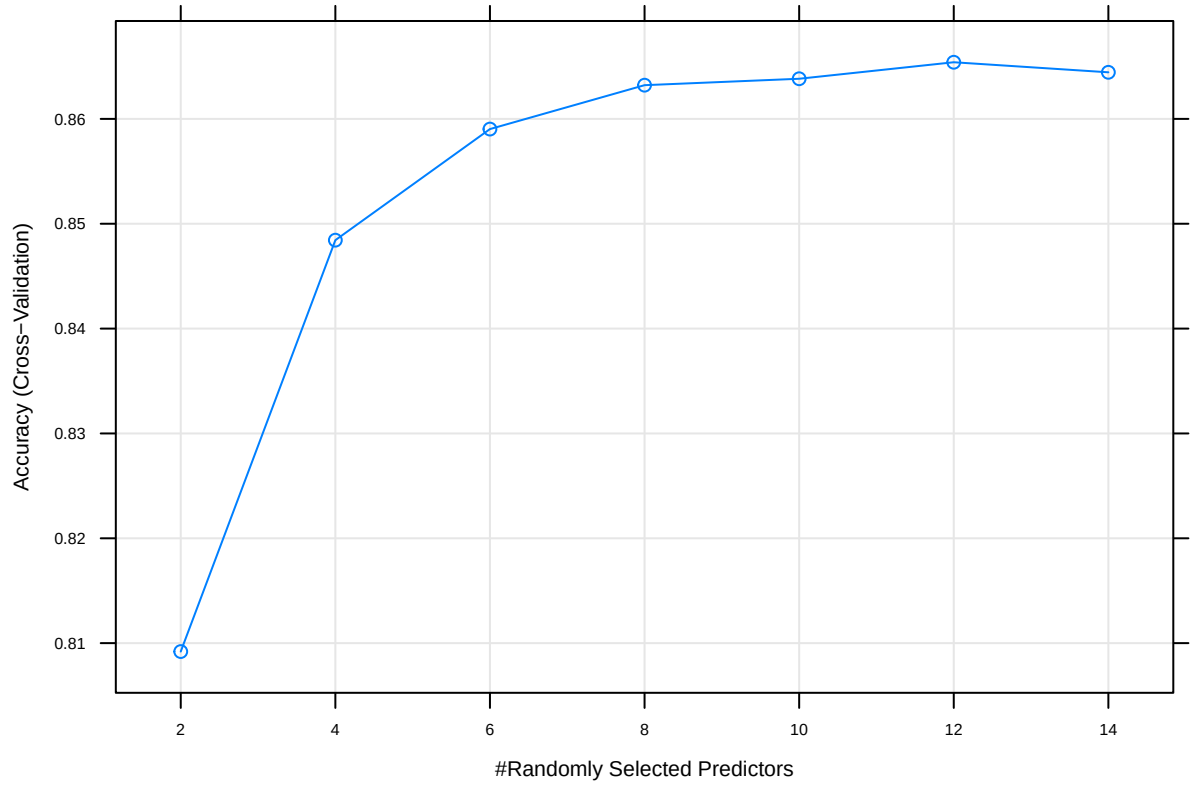
The final F_1 score of our single tree is 0.6372981, which is very encouraging comparing to 0.246413 of the guessing benchmark, and that is 158% improvement. At the same time the kappa ratio 0.5445 is telling us our higher overall accuracy is not by luck.

2. Forest

Since we have tried the tree method, then the forest method can not be ignored in the tree family. As stated in Prof. Irizarry's text, random forests can improve the single decision tree performance by bagging on multiple smaller trees (Irizarry 2022). The logic behind the bagging is that, instead of growing one tree out of all the available 14 predictors, we randomly pick "mtry" number of predictors to grow "ntree" number of small trees, the size of the trees is defined by the "nodesize", with smaller trees having larger "nodesize". So with the central idea of bagging, we would avoid to have the situation we faced in above single tree, where very few predictors such as "capital gain" and "capital loss", that comprise the majority of the decision rules. This could be a sign of under training. And there also could be the case of over training in a tree, if the rule of branch splitting is too granular, then the bagging method can also accommodate this situation.

As we described in above, we will tune our "randomForest" with three major parameters: "mtry", "nodesize", "ntree" in a sequential form.

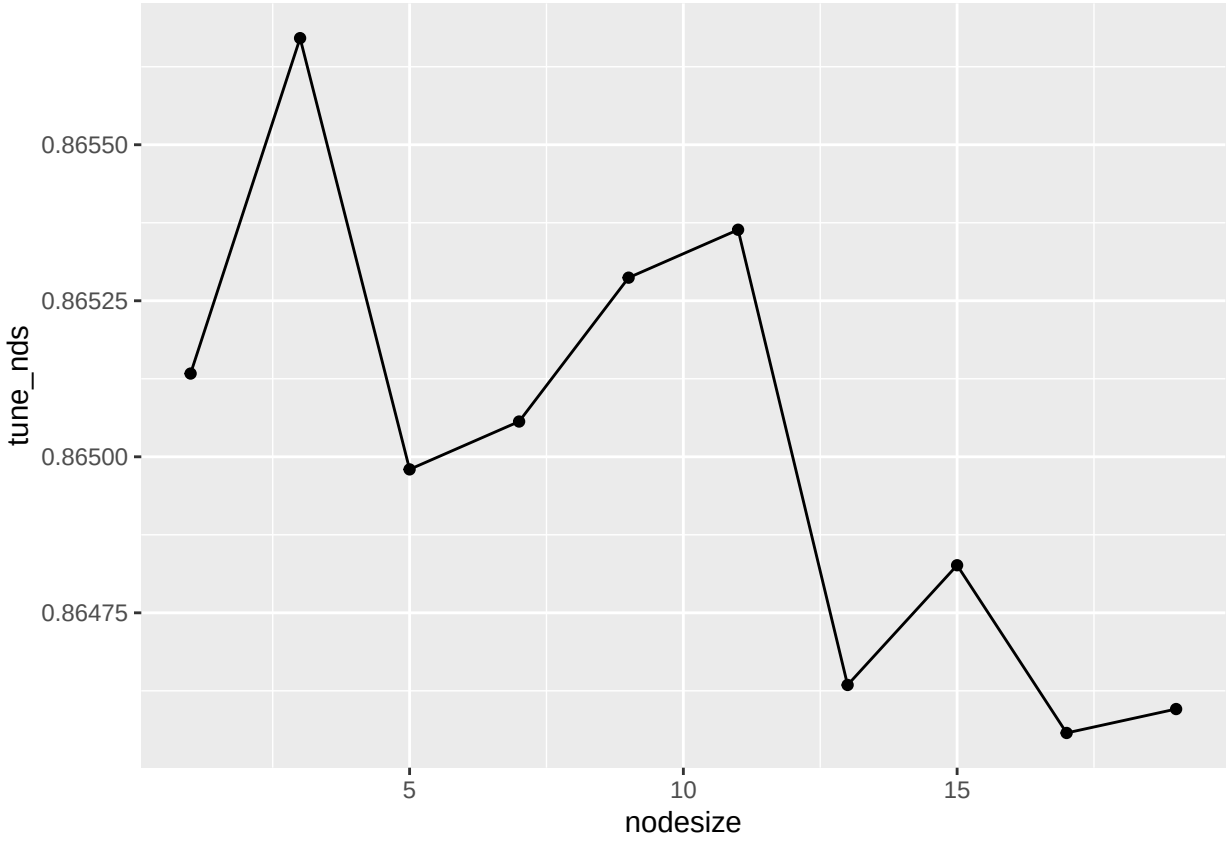
```r
#tune the mtry 1st to decide how many predictors for each trees
#same cv index from the tree
rfcontrol <- trainControl(method = "cv", index = ind_cv)
tune_forest <- train(income ~., data = train, method = "rf",
                     trControl = rfcontrol,
                     tuneGrid = data.frame(mtry = seq(2, 14, 2)))
```



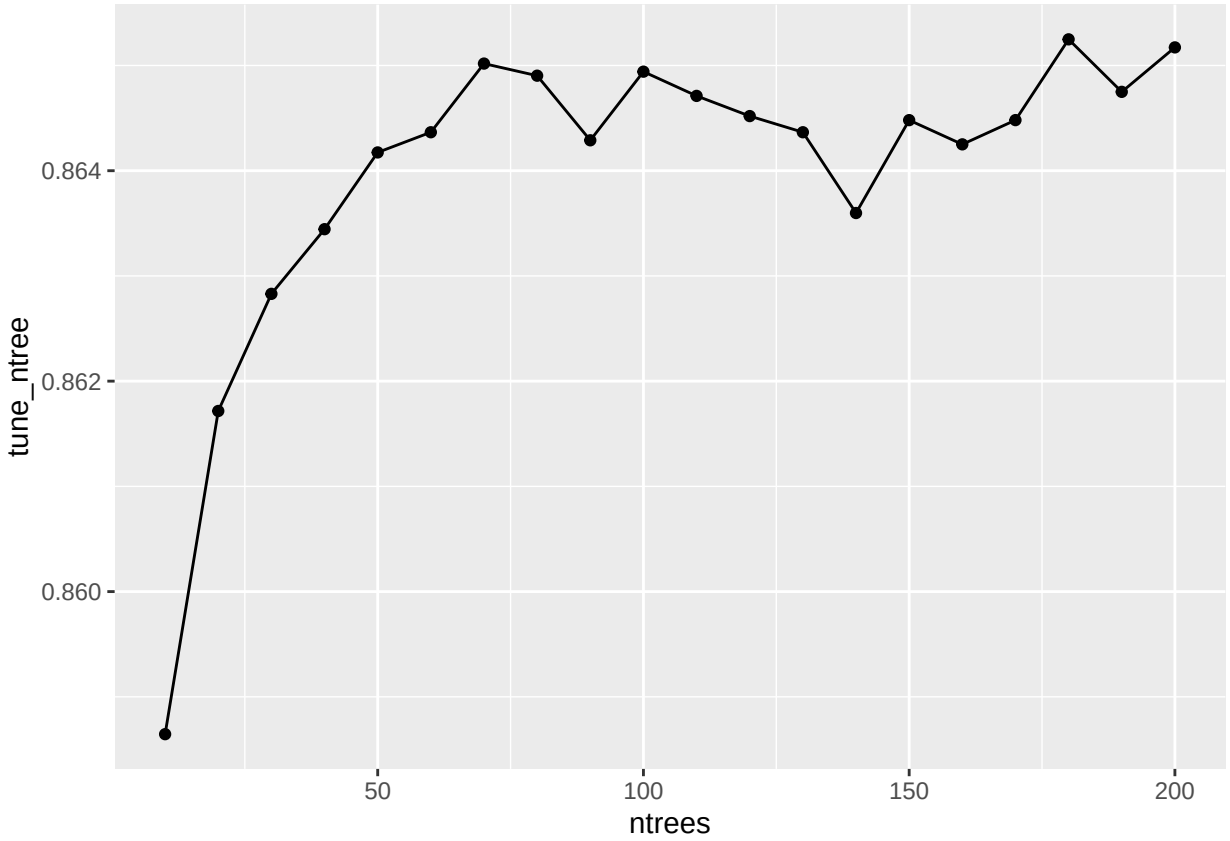

```
## Random Forest
##
## 26048 samples
##    14 predictor
##     2 classes: '<=50K', '>50K'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 20838, 20840, 20838, 20838, 20838
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8091972  0.3050591
##    4    0.8484331  0.5274223
##    6    0.8590292  0.5765757
##    8    0.8632138  0.5952431
##   10    0.8638280  0.6002937
##   12    0.8654022  0.6056670
```

```
##   14    0.8644424  0.6037342
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 12.
```

```r
#tune the nodesize to decide how big the leaf node is
nodesize <- seq(1, 20, 2)
tune_nds <- sapply(nodesize, function(nd){
  train(income ~., data = train, method = "rf",
        trControl = rfcontrol,
        tuneGrid = data.frame(mtry = best_mtry),
        nodesize = nd)$results$Accuracy
})
```




```r
#tune the number of trees in the forest
ntrees <- seq(10, 200, 10)
tune_ntree <- sapply(ntrees, function(nt){
  train(income ~., data = train, method = "rf",
        trControl = rfcontrol,
        tuneGrid = data.frame(mtry = best_mtry),
        nodesize = best_node,
        ntree = nt)$results$Accuracy})
```



The above turning process suggest that the optimised forest is composed by "ntrees" = 180 trees, the minimum leaf size "nodesize" = 3, which means the trees in the forest are quite large, and the number of predictors "mtry" we use to grow each tree is 12, almost include all the 14 available predictors. Then we check on the performance on test set in below.

```
#test the tuned forest
fit_forest <- train(income ~., data = train, method = "rf",
                    tuneGrid = data.frame(mtry = best_mtry),
                    nodesize = best_node,
                    ntree = best_ntree)

pred_forest <- predict(fit_forest, test)
gauge_forest <- confusionMatrix(pred_forest, test$income,
                                mode = "prec_recall",
                                positive = ">50K")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction <=50K >50K
##      <=50K  4639  593
##      >50K    292  989
##
##                Accuracy : 0.8641
##                  95% CI : (0.8556, 0.8724)
##     No Information Rate : 0.7571
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.605
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.7721
##                  Recall : 0.6252
##                      F1 : 0.6909
##              Prevalence : 0.2429
##          Detection Rate : 0.1519
##    Detection Prevalence : 0.1967
##       Balanced Accuracy : 0.7830
##
##        'Positive' Class : >50K
##
```

```
##        F1
## 0.6908837
```

The final F_1 score 0.6908837 does meet our initial expectation of achieving a better performance than the single tree in before, although the improvement is very limited for just over 8%. In next we will jump out of trees family to try another approach in a more linear algebra way, which is the Support Vector Machine (SVM).

3. Support Vector Machine

Broadly speaking the SVM is to search a hyperplane that can effectively separate two different class samples in the attributes space. In our case each row of the dataset represents a surveyed sample, which has 14 attributes dimensions, so we are searching a hyperplane inside a 14 dimensional space. Intuitively this hyperplane should have the dimension of 13, because in a x-y 2 dimension, it is a single line, and in a x-y-z 3 dimensional space, it is a 2 dimensional plane. According to the original paper, this hyperplane is spanned by the selective vectors out of the original dataset (Cortes and Vapnik 1995). What it means is that, we believe the two class samples are clustering toward two opposite directions, and those samples at the joint border of the two clusters are considered as supporting vectors.

There are two popular SVM packages in R, and we use the LIBSVM based "e1071" package, the LIBSVM is created and maintained by the team from National Taiwan University. We need to tune the model based on the chosen kernel. According to the package document, if the samples are in high dimensions, then the most basic linear kernel would be the best performing choice (Chang and Lin 2011). So we will turn the model with linear kernel first, then try different kernels and approaches to help us decide, whether the basic linear model is the best choice to serve our purpose.

Before train the model, we need to make our dataset to fit the algorithm requirement, because the algorithm only works on quantitative data, but not qualitative data. For all the qualitative attributes columns, there are two options to digitise them, one option is to use the 'dummyVars' function from "caret" package can turn those columns' categorical values into binary data, and creating a new column for every categorical data value in addition to the original dataset. For example, in the "education" column, we have 15 different categories of data in place.

```
##  [1] "HS-grad"      "Some-college" "7th-8th"      "10th"         "Doctorate"
##  [6] "Prof-school"  "Bachelors"    "Masters"      "11th"         "Assoc-acdm"
## [11] "Assoc-voc"    "1st-4th"      "5th-6th"      "12th"         "9th"
## [16] "Preschool"
```

The function 'dummyVars' will create another 15 new columns with the names being those qualitative values. If we use this function on all the 8 qualitative attributes columns, we will have lots of 0's in our new dataset, which means we get a huge sparse dataset. High sparsity could become another risk for our model. The central idea of pre-processing and data cleaning is to fit in the algorithm, and improve the performance, so we will take another approach for those qualitative values. Our approach is to digitise them locally, use the same "education" column example, we will simply mark them with digits accordingly with their position indexes in the result of 'uniqueN' function. For the "?", we will force them to be 0, regardless of their position index, so for the three columns have the "?" values, we will treat them a little differently.

```
##digitise all categorical columns
data_digit <- data[, ..char_cols]
sapply(data_digit, uniqueN)
sapply(data_digit, unique)

#find all the "?" and turn them into 0
data_digit[, lapply(.SD, function(j) sum(str_detect(j, "\\?")))]

data_digit <- data_digit[,
              lapply(.SD, function(j) str_replace(j, "\\?", "0"))]

data_digit[, lapply(.SD, function(j) sum(str_detect(j, "\\?")))]
data_digit[, lapply(.SD, function(j) sum(str_detect(j, "0")))]

sapply(data_digit, unique)

##cols contain "?", force them to be "0"
#the other values follow the rule to be transformed into the index number
wc <- 1:uniqueN(data_digit$workclass)
for(w in wc){data_digit$workclass[which(data_digit$workclass ==
                    unique(data_digit$workclass)[w])] <- wc[w] - 1}
rm(w, wc)

oc <- 1:uniqueN(data_digit$occupation)
for(o in oc){data_digit$occupation[which(data_digit$occupation ==
                    unique(data_digit$occupation)[o])] <- oc[o] - 1}
rm(o, oc)

#"0" is in 2nd place of uniqueN(), special treatment convert "2" -> "0"
nc <- 1:uniqueN(data_digit$native.country)
for(n in nc){data_digit$native.country[which(data_digit$native.country ==
                    unique(data_digit$native.country)[n])] <- nc[n]}
data_digit$native.country[which(data_digit$native.country == "2")] <- 0
rm(n, nc)

ed <- 1:uniqueN(data_digit$education)
for(e in ed){data_digit$education[which(data_digit$education ==
                    unique(data_digit$education)[e])] = ed[e]}
rm(e, ed)

ms <- 1:uniqueN(data_digit$marital.status)
for(m in ms){data_digit$marital.status[which(data_digit$marital.status ==
                    unique(data_digit$marital.status)[m])] = ms[m]}
rm(m, ms)
```

```
rl <- 1:uniqueN(data_digit$relationship)
for(r in rl){data_digit$relationship[which(data_digit$relationship ==
                    unique(data_digit$relationship)[r])] = rl[r]}
rm(r, rl)

rc <- 1:uniqueN(data_digit$race)
for(r in rc){data_digit$race[which(data_digit$race ==
                    unique(data_digit$race)[r])] = rc[r]}
rm(r, rc)

sx <- 1:uniqueN(data_digit$sex)
for(s in sx){data_digit$sex[which(data_digit$sex ==
                    unique(data_digit$sex)[s])] = sx[s]}
rm(s, sx)

data_digit <- data_digit[, lapply(.SD, as.numeric)]
data_digit <- data[, (char_cols) := data_digit]
str(data_digit)

#same partition index from trees to make the digitised train/test set
train_svm <- data_digit[-ind, ]
test_svm <- data_digit[ind, ]
```

After we have the data ready, we can now turn the model. Because we are using the linear kernel first, the only parameter we need to tune is the "cost". This parameter is acting as the Lagrange multiplier in the dual problem for solving the SVM (Cortes and Vapnik 1995). Geometrically it is the trade-off margin of our optimised hyperplane, because most of the time, we will not get an ideal dataset, such that the two sample classes are perfectly separable, the error of classification is certainly to happen. We also call this kind of hyperplane as the soft margin hyperplane, which allows training errors within a boundary of [0, Cost] Platt (1998).

```
#use the same cv index from trees to make the training process reproducible
svmcontrol <- trainControl(method = "cv", index = ind_cv)
fit_svm <- train(income ~., data = train_svm, method = "svmLinear2",
                 trControl = svmcontrol,
                 tuneGrid = data.frame(cost = c(
                      2^-13, 2^-10, 2^-8 ,2^-5, 2^-1, 1, 2^3)))

pred_svm <- predict(fit_svm, test_svm)
gauge_svm <- confusionMatrix(pred_svm, test_svm$income,
                             mode = "prec_recall",
                             positive = ">50K")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction <=50K >50K
##      <=50K  4630  718
##      >50K    301  864
##
##                Accuracy : 0.8435
##                  95% CI : (0.8345, 0.8523)
##     No Information Rate : 0.7571
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5328
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.7416
##                  Recall : 0.5461
##                      F1 : 0.6290
##              Prevalence : 0.2429
##          Detection Rate : 0.1327
##    Detection Prevalence : 0.1789
##       Balanced Accuracy : 0.7426
##
##        'Positive' Class : >50K
##


##        F1
## 0.6290499
```

The final F_1 score 0.6290499 is certainly favourable comparing to the guessing benchmark, but is not as good as the two tree methods in above. This result makes us doubt our choice of linear kernel, so we will try to map our data with a 2nd degree polynomial kernel for comparison. The general idea about kernel is mapping the data into a higher dimension, which can effectively solve the low dimension data, but our data are in a 14 dimensional space already, so we used the linear kernel at first.

In addition to the "cost" parameter, there is one more parameter "gamma" to be turned for the polynomial kernel. The "gamma" is a scalar coefficient that scales the output value of the the kernel. The function in below is a 2nd degree polynomial kernel function, and the $\vec{x_i}$ and $\vec{x_j}$ are any two different sample points in the dataset, the "r" is a constant set as 0 by default, so we just use the "r = 0" at default in our case. As stated in above, the purpose of 2nd or higher dimension kernel is to solve the low dimensional data, so we will just prudently apply the 2nd degree instead of higher ones for our 14 dimensional sample data.

$$K_{poly} = (\gamma(\vec{x_i} \cdot \vec{x_j}) + r)^2$$

Instead of tune these two parameters sequentially, we will tune them in a grid structure. With the 'expand.grid' function we can make a number of potential "cost" times number of potential "gamma" sized data frame, with each row is a cost-gamma pair to be trained with the cross validation. We also use a mini set to train the linear and polynomial kernels for comparison, because the polynomial kernel can increase the computation cost exponentially. To create the mini set, we use 20% of the original data, and keep the same proportions of the two income groups. To mimic the dependent classes distribution, we can make sure the polynomial kernel is also tested in a unbalanced sample environment. At the end, we test these two mini set trained SVM on the full size test set to compare their performances. To stress that the performance of this test method is just for comparison within the two kernels, if we use the full size train set, it can take few days even up to a week for the polynomial kernel.

```
#2nd degree polynomial kernel, compare with linear kernel
c <- c(2^-5, 1, 2^5)
g <- c(2^-5, 1, 2^5)
para_grid <- expand.grid(cost = c, gamma = g)

#mini tuning cv set, mimic the income class distribution from original dataset
under_ind <- which(data$income == "<=50K")
above_ind <- which(data$income == ">50K")
```

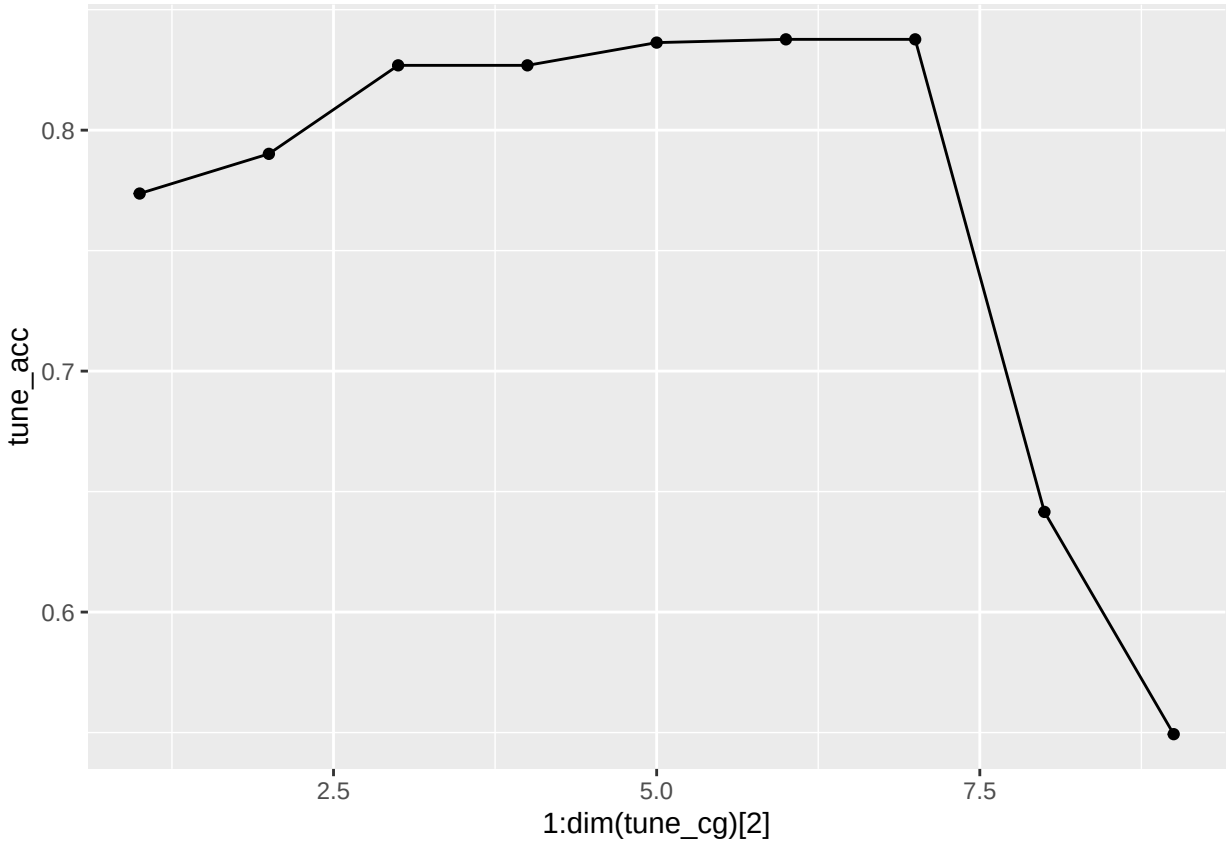
```
set.seed(3, sample.kind = "Rounding")
mini_under <- sample(under_ind, length(under_ind) * 0.2,
                     replace = FALSE)
set.seed(4, sample.kind = "Rounding")
mini_above <- sample(above_ind, length(above_ind) * 0.2,
                     replace = FALSE)
mini_ind <- c(mini_under, mini_above)

#create the 5-fold cv sets
set.seed(5, sample.kind = "Rounding")
mini_cv <- createFolds(mini_ind, k = 5, returnTrain = TRUE)
mini_cv_ind <- lapply(1:5, function(k) mini_ind[mini_cv[[k]]])

tune_cg <- foreach(j = 1:dim(para_grid)[1], .combine = cbind.data.frame,
                   .packages = "e1071") %:%
  foreach(k = 1:5, .combine = c) %dopar% {
    cv_train <- svm(income ~., data = data_digit[mini_cv_ind[[k]],],
                    cost = para_grid[j, 1], gamma = para_grid[j, 2],
                    kernel = "polynomial", degree = 2)
    val_acc <- sum(predict(cv_train, data_digit[-mini_cv_ind[[k]],]) ==
      data_digit[-mini_cv_ind[[k]],]$income) /
      dim(data_digit[-mini_cv_ind[[k]],])[1]
  }

tune_acc <- apply(tune_cg, 2, mean)
best_cg <- para_grid[which.min(tune_acc), ]
```

(to be continued on next page)




```
#test the mini set trained polynomial kernel
polyfit_svm <- svm(income ~., data = data_digit[mini_ind,],
                   cost = best_cg$cost, gamma = best_cg$gamma,
                   kernel = "polynomial", degree = 2)

polypred_svm <- predict(polyfit_svm, test_svm)
polygauge_svm <- confusionMatrix(polypred_svm, test_svm$income,
                                 mode = "prec_recall",
                                 positive = ">50K")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction <=50K >50K
##      <=50K  2638  739
##      >50K   2293  843
##
##                Accuracy : 0.5345
##                  95% CI : (0.5223, 0.5466)
##     No Information Rate : 0.7571
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0509
##
##  Mcnemar's Test P-Value : <2e-16
##
```

```
##               Precision : 0.2688
##                  Recall : 0.5329
##                      F1 : 0.3574
##              Prevalence : 0.2429
##          Detection Rate : 0.1294
##    Detection Prevalence : 0.4815
##       Balanced Accuracy : 0.5339
##
##        'Positive' Class : >50K
##
```

```
##        F1
## 0.3573548
```

```
#test the linear kernel on the mini set too
#but use the tuned "cost" from the previous full train set
linfit_svm <- train(income ~., data = data_digit[mini_ind,],
                    method = "svmLinear2",
                    tuneGrid = data.frame(cost = c(2^-5)))

linpred_svm <- predict(linfit_svm, test_svm)
lingauge_svm <- confusionMatrix(linpred_svm, test_svm$income,
                                mode = "prec_recall",
                                positive = ">50K")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction <=50K >50K
##      <=50K  4691  818
##      >50K    240  764
##
##                Accuracy : 0.8376
##                  95% CI : (0.8284, 0.8464)
##     No Information Rate : 0.7571
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4958
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.7610
##                  Recall : 0.4829
##                      F1 : 0.5909
##              Prevalence : 0.2429
##          Detection Rate : 0.1173
##    Detection Prevalence : 0.1542
##       Balanced Accuracy : 0.7171
##
##        'Positive' Class : >50K
##
```

```
##        F1
## 0.5908739
```

Kernel	test_F_1
polynomial	0.3573548
linear	0.5908739

The two F_1 scores of the results are 0.3573548 for 2nd degree polynomial, and 0.5908739 for linear. They are proving our original belief that, linear kernel is better than higher dimensional mapping kernels in an already high dimensional sample space. In next we question ourselves in further, because the SVM was designed to solve the purely numeric attributes samples in a vector space, but we have many qualitative attributes, even we have pre-processed them into numbers, those numbers are rather out of nowhere dummy indexes without any real meaning. Then we want to test out, if we just train the linear kernel on those quantitative attributes, whether the performance can be improved.

```
#only use numeric predictors
#the 'cols' is picking all the numeric columns
#we defined the 'cols' in the beginning of visualisation
numcols <- c(cols, "income")
numtrain_svm <- data[, ..numcols][-ind,]
numtest_svm <- data[, ..numcols][ind,]

numfit_svm <- train(income ~., data = numtrain_svm, method = "svmLinear2",
                    trControl = svmcontrol,
                    tuneGrid = data.frame(cost = c(
                      2^-13, 2^-10, 2^-8 ,2^-5, 2^-1, 1, 2^3)))
plot(numfit_svm)
numfit_svm
numfit_svm$finalModel

numpred_svm <- predict(numfit_svm, numtest_svm)
numgauge_svm <- confusionMatrix(numpred_svm, numtest_svm$income,
                                mode = "prec_recall",
                                positive = ">50K")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction <=50K >50K
##      <=50K  4811 1140
##      >50K    120  442
##
##                Accuracy : 0.8065
##                  95% CI : (0.7967, 0.8161)
##     No Information Rate : 0.7571
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3266
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.78648
##                  Recall : 0.27939
##                      F1 : 0.41231
##              Prevalence : 0.24290
```

```
##          Detection Rate : 0.06786
##    Detection Prevalence : 0.08629
##       Balanced Accuracy : 0.62753
##
##        'Positive' Class : >50K
##
```

```
##        F1
## 0.4123134
```

Attributes	test_F_1
numeric	0.4123134
full	0.6290499

The resulting F_1 score is only 0.4123134, much worse than the full attribute linear model, so it means the way we treat the qualitative attributes are working. In next, we will try optimise our attributes selections to train the linear kernel again, to see whether the approach can bring us any improvement on the result. The motivation is from the tree family, for example the random forest trained result suggests that "race" and "native.country" are not top import attributes to make the classifications, so we will try to exclude these two attribute columns in our linear SVM, to see whether a better result can be obtained.

```
## rf variable importance
##
##   only 20 most important variables shown (out of 100)
##
##                                 Overall
## capital.gain                     916.08
## marital.statusMarried-civ-spouse  765.14
## age                              615.42
## education.num                    570.68
## hours.per.week                   421.03
## fnlwgt                           419.93
## marital.statusNever-married      269.30
## capital.loss                     245.96
## occupationExec-managerial        168.13
## relationshipNot-in-family        153.95
## occupationProf-specialty         144.32
## sexMale                          134.81
## educationBachelors               125.68
## relationshipOwn-child            120.90
## relationshipUnmarried             77.24
## educationMasters                  73.16
## relationshipWife                  68.86
## occupationOther-service           58.95
## workclassPrivate                  55.32
## workclassSelf-emp-not-inc         53.88
```

```
#exclude the "race" and "native country" columns in predictor
imptrain_svm <- data[, -c("race", "native.country")][-ind,]
imptest_svm <- data[, -c("race", "native.country")][ind,]
impfit_svm <- train(income ~., data = imptrain_svm, method = "svmLinear2",
```

```
                    trControl = svmcontrol,
                    tuneGrid = data.frame(cost = c(
                      2^-13, 2^-10, 2^-8 ,2^-5, 2^-1, 1, 2^3)))

imppred_svm <- predict(impfit_svm, imptest_svm)
impgauge_svm <- confusionMatrix(imppred_svm, imptest_svm$income,
                                mode = "prec_recall",
                                positive = ">50K")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction <=50K >50K
##      <=50K  4628  715
##      >50K    303  867
##
##                Accuracy : 0.8437
##                  95% CI : (0.8346, 0.8524)
##     No Information Rate : 0.7571
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5338
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.7410
##                  Recall : 0.5480
##                      F1 : 0.6301
##              Prevalence : 0.2429
##          Detection Rate : 0.1331
##    Detection Prevalence : 0.1796
##       Balanced Accuracy : 0.7433
##
##        'Positive' Class : >50K
##

##        F1
## 0.6300872
```

The result F_1 score 0.6300872 is about the same as the full attributes model, this result can somewhat validate our assumption that, use the selective important predictors is at least as good as the full predictor model, and in some particular context, it could be a slightly better choice, if the computation cost is considered for the process. By talking about computation cost, each of the SVM models we have tried in above can take few hours to train, especially the polynomial kernel can take days to train, by considering my hardware environment is a 5 years old i7 cpu windows machine.

Attributes	test_F_1
numeric	0.4123134
full	0.6290499
imp	0.6300872

Overall the best SVM is important variable linear kernel, as well as the full predictors linear kernel, because their F_1 scores have very slight difference. Since we have transformed our dataset into full numeric set already

for the SVM algorithm, intuitively we can use the same numeric dataset to train some other methods, which is based on pure numeric data. In next, we will try to fit the logistic regression with our transformed numeric dataset, to see if it can compete with trees and SVM methods.

4. Logistic Regression

We will use both the full predictors and selective important predictors from the above SVM dataset to test the performance of logistic regression, and we are using the binomial "glm" in "caret" package to implement the algorithm.

```
#use the full predictor to train and test
fit_glm <- train(income ~., data = train_svm, method = "glm",
                 family = binomial)

pred_glm <- predict(fit_glm, test_svm)
gauge_glm <- confusionMatrix(pred_glm, test_svm$income,
                             mode = "prec_recall",
                             positive = ">50K")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction <=50K >50K
##      <=50K  4617  721
##      >50K    314  861
##
##                Accuracy : 0.8411
##                  95% CI : (0.832, 0.8499)
##     No Information Rate : 0.7571
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5266
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.7328
##                  Recall : 0.5442
##                      F1 : 0.6246
##              Prevalence : 0.2429
##          Detection Rate : 0.1322
##    Detection Prevalence : 0.1804
##       Balanced Accuracy : 0.7403
##
##        'Positive' Class : >50K
##


##        F1
## 0.6245919
```

```
#use the important predictor to train and test
impfit_glm <- train(income ~., data = imptrain_svm, method = "glm",
                 family = binomial)
```

```r
imppred_glm <- predict(impfit_glm, imptest_svm)
impgauge_glm <- confusionMatrix(imppred_glm, imptest_svm$income,
                                mode = "prec_recall",
                                positive = ">50K")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction <=50K >50K
##      <=50K  4611  719
##      >50K    320  863
##
##                Accuracy : 0.8405
##                  95% CI : (0.8314, 0.8493)
##     No Information Rate : 0.7571
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5256
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.7295
##                  Recall : 0.5455
##                      F1 : 0.6242
##              Prevalence : 0.2429
##          Detection Rate : 0.1325
##    Detection Prevalence : 0.1816
##       Balanced Accuracy : 0.7403
##
##        'Positive' Class : >50K
##

##        F1
## 0.6242315
```

The result F_1 scores of full predictors and important predictors are 0.6245919 and 0.6242315 respectively, so we can conclude that the logistic regression is performing the same over these two datasets. After all those methods we have tried by now, it is instantaneously for us to think about combining them in together with ensemble method.

Attributes	F_1
full	0.6245919
important	0.6242315

5. Ensemble

In general there are three kinds of ensemble methods: bagging, boosting, and stacking, and we will try to combine our previous models by stacking. The reason is that stacking method treats different models results like predictors, and fit them with a defined meta-model (Hastie, Tibshirani, and Friedman 2009), and the meta-model options are very versatile, so it can mimic both bagging and boosting with the respective model

selection. And because our case is a classification kind problem, we will try logistic regression and adaptive boosting methods as our meta-model to combine the random forest, linear SVM, and logistic regression.

Because the SVM and logistic regression can only take numeric predictors, we need to decide whether the random forest need to be re-turned for the numeric dataset. We also need to do some pre-processing work to convert the "<=" into "loe", and ">" into "gt" for the two income groups, this is due to the data type requirement by the "caretEnsemble" package, what we use for implementing the ensemble process.

```
#"caretEnsemble" pkg does not recognise symbols,
#need to be replaced by charc
cnvt_income <- train_svm[, lapply(.SD,
                            function (j) str_replace(j, "\\>", "gt")),
                            .SDcols = c("income")]
cnvt_income <- cnvt_income[, lapply(.SD,
                            function(j) str_replace(j, "\\<=", "loe"))]

cnvtst_income <- test_svm[, lapply(.SD,
                           function (j) str_replace(j, "\\>", "gt")),
                           .SDcols = c("income")]
cnvtst_income <- cnvtst_income[, lapply(.SD,
                            function(j) str_replace(j, "\\<=", "loe"))]

train_svm[, income := cnvt_income]
test_svm[, income := cnvtst_income]

#use the tuned parameters from above
#to test the "rf" performance on the transformed dataset
enfit_rf <- train(income ~., data = train_svm, trControl = rfcontrol,
                  method = "rf", tuneGrid = data.frame(mtry = best_mtry),
                  nodesize = best_node, ntree = best_ntree)
enpred_rf <- predict(enfit_rf, test_svm)
engauge_rf <- confusionMatrix(enpred_rf, as.factor(test_svm$income),
                              mode = "prec_recall",
                              positive = "gt50K")
```

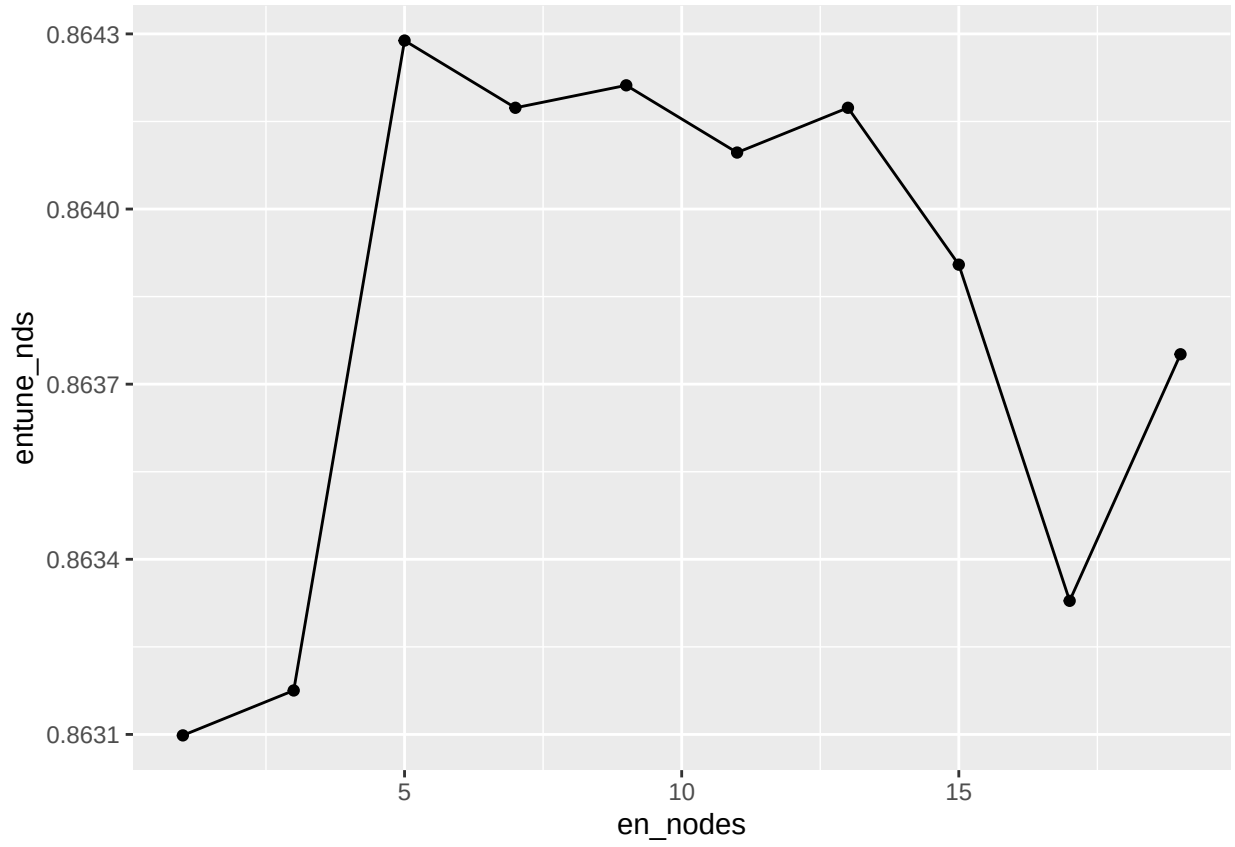
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction gt50K loe50K
##     gt50K    983    340
##     loe50K   599   4591
##
##                Accuracy : 0.8558
##                  95% CI : (0.8471, 0.8643)
##     No Information Rate : 0.7571
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5849
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.7430
##                  Recall : 0.6214
##                      F1 : 0.6768
```

```
##              Prevalence : 0.2429
##          Detection Rate : 0.1509
##    Detection Prevalence : 0.2031
##       Balanced Accuracy : 0.7762
##
##        'Positive' Class : gt50K
##
```

```
##        F1
## 0.6767642
```

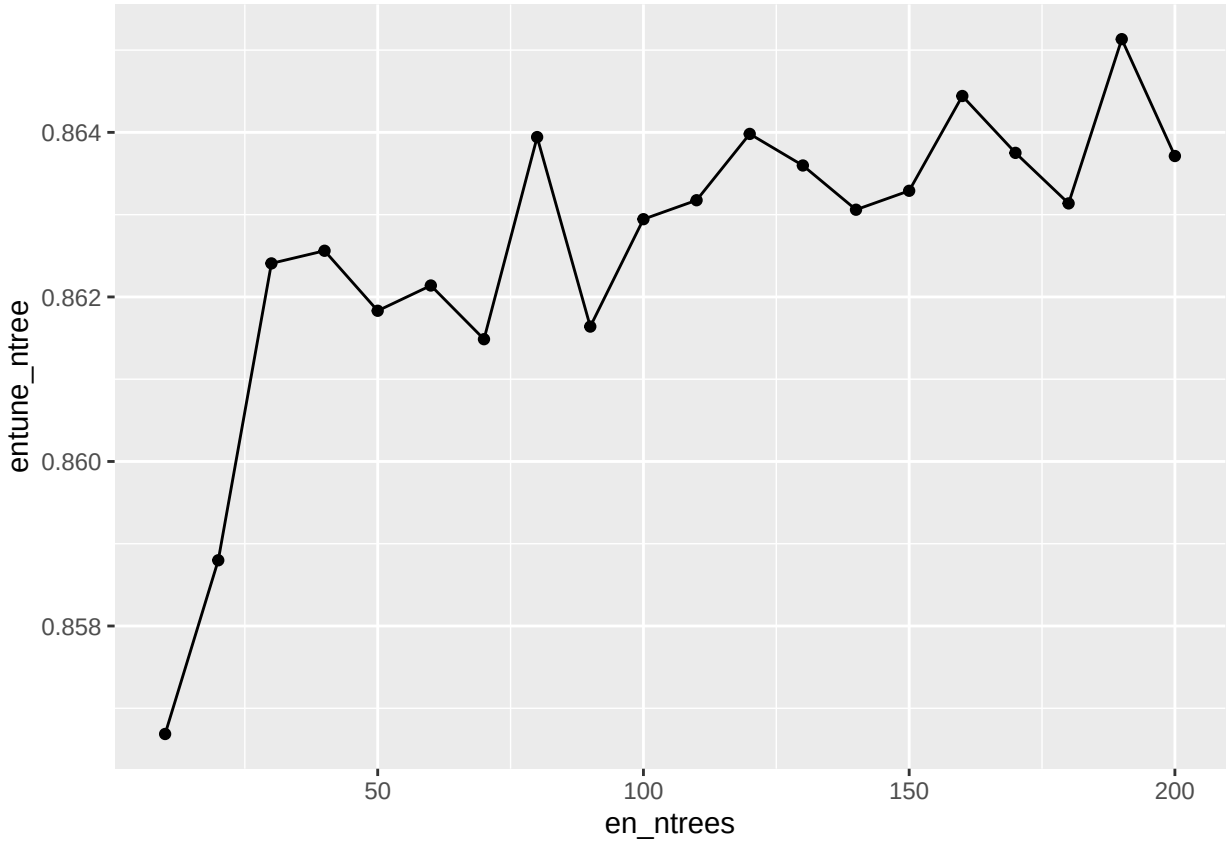
```
#re-tune the rf with the digitised dateset to compare the results
entune_rf <- train(income ~., data = train_svm, method = "rf",
                   trControl = rfcontrol,
                   tuneGrid = data.frame(mtry = seq(1, 5, 1)))
enbest_mtry <- entune_rf$bestTune$mtry #re-turned mtry

en_nodes <- seq(1, 20, 2) #re-tune the nodesize
entune_nds <- sapply(en_nodes, function(nd){
  train(income ~., data = train_svm, method = "rf",
        trControl = rfcontrol,
        tuneGrid = data.frame(mtry = enbest_mtry),
        nodesize = nd)$results$Accuracy
})
enbest_node <- en_nodes[which.max(entune_nds)]
```



```r
en_ntrees <- seq(10, 200, 10) #re-turn the ntree
entune_ntree <- sapply(en_ntrees, function(nt){
  train(income ~., data = train_svm, method = "rf",
        trControl = rfcontrol,
        tuneGrid = data.frame(mtry = enbest_mtry),
        nodesize = enbest_node,
        ntree = nt)$results$Accuracy})
enbest_ntree <- ntrees[which.max(entune_ntree)]
```

(to be continued on next page)




```r
#use the re-tuned parameters to test the result
enfit_forest <- train(income ~., data = train_svm, method = "rf",
                      tuneGrid = data.frame(mtry = enbest_mtry),
                      nodesize = enbest_node,

enpred_forest <- predict(enfit_forest, test_svm)
engauge_forest <- confusionMatrix(enpred_forest, as.factor(test_svm$income),
                                  mode = "prec_recall",
                                  positive = "gt50K")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction gt50K loe50K
```

```
##     gt50K    983    293
##     loe50K   599   4638
##
##                Accuracy : 0.863
##                  95% CI : (0.8545, 0.8713)
##     No Information Rate : 0.7571
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6015
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.7704
##                  Recall : 0.6214
##                      F1 : 0.6879
##              Prevalence : 0.2429
##          Detection Rate : 0.1509
##    Detection Prevalence : 0.1959
##       Balanced Accuracy : 0.7810
##
##        'Positive' Class : gt50K
##

##        F1
## 0.6878936
```

By comparing the two result F_1 scores, the re-turned result is 0.6878936, which is better but in a very limited scope than the pre-tuned 0.6767642. We choose to use the re-tuned parameters in our final ensemble. Because we are using the digitised dataset from SVM in above, we can use the same parameters as before for the binomial "glm" and "svmLinear2".

```
#5-fold cross validation used to create dependent probs from learning
#for later use in the combining process
crEnsem <- trainControl(method = "cv", index = ind_cv,
                        savePredictions = "final", classProbs = TRUE)
ensemble_fit <- caretList(income ~., data = train_svm, trControl = crEnsem,
                          tuneList = list(
                          svm = caretModelSpec(method = "svmLinear2",
                                  tuneGrid = data.frame(cost = 0.03125)),
                          glm = caretModelSpec(method = "glm",
                                  family = binomial),
                          rf = caretModelSpec(method = "rf",
                                  tuneGrid = data.frame(mtry = enbest_mtry),
                                  nodesize = enbest_node,
                                  ntree = enbest_ntree)))

#stacking by "logistic regression"
stack <- caretStack(ensemble_fit, method = "glm",
                    trControl = trainControl(method = "boot", number = 5,
                                             savePredictions = "final"))
pred_stack <- predict(stack, test_svm)
gauge_stack <- confusionMatrix(pred_stack, as.factor(test_svm$income),
                               mode = "prec_recall",
                               positive = "gt50K")
```

```
##        F1
## 0.6879334
```

```
#stacking by "adaptive boosting"
stack_boost <- caretStack(ensemble_fit, method = "AdaBoost.M1",
                          trControl = trainControl(method = "boot", number = 5,
                                                   savePredictions = "final"))
pred_stackbst <- predict(stack, test_svm)
gauge_stackbst <- confusionMatrix(pred_stack, as.factor(test_svm$income),
                                  mode = "prec_recall",
                                  positive = "gt50K")
```

```
##        F1
## 0.6879334
```

The results are interesting, the two different meta-models are giving us the exact same F_1 scores 0.6879334. We suspect the same results are caused by two possible reasons. Firstly, the two stacking meta-models we use are all voting decision based, which use the consensus from the results of those three base models. Secondly, these three base models are generating about the same prediction results as shown in below. According to the 'vignette' of the "caretEnsemble" package, we should consider to select the uncorrelated or at least weak correlated base models for ensemble (Deane-Mayer and Knowles 2019). The original numeric and categorical attributes data structure actually creates a paradox of choosing uncorrelated models and without modifying the dataset. We believe the process of digitising those qualitative values can loose the intrinsic information of those attributes at some level. But if we do not digitise those qualitative values, then we have very limited choices of models to choose from, basically only the tree family we can possibly use.

```
##           svm       glm        rf
## svm 1.0000000 0.9384295 0.9628687
## glm 0.9384295 1.0000000 0.9348694
## rf  0.9628687 0.9348694 1.0000000
```

Conclusion

By comparing all the methods we have tested in above, it concludes that the best performing result is the random forest having the F_1 score of 0.6908837, and it even outperformed the stacking method, but the difference is not major. By comparing across all individual methods F_1 scores, The trees family methods outperform other numeric methods. This report in overall proves that, for a quantitative and qualitative mixed attributes dataset, the most effective algorithm would be the tree based.

Methods	F_1_score
Guessing	0.2464130
Tree.rpart	0.6372981
Forest.rf	0.6908837
SVM.linear	0.6290499
glm	0.6245919
Stacking.glm	0.6879334
Stacking.boost	0.6879334

The last thing we want to justify is that, even the best result F_1 score is still under 0.7, a very mediocre number by considering the range of F_1 is [0, 1]. That is because the two income groups in our dataset is

highly unbalanced with a 75/25 distribution ratio, and our target ">50K" group is the under sampled one. Since the F_1 score only cares about the target class of ">50K", it tells us how good our prediction in both the only ">50K" showing environment, and both income groups showing environment. In order to improve the F_1 score, we could try to increase the result in either environment. The possible approaches would be using a more sophisticate way to digitise those qualitative attributes, then we might be able to achieve a better result from the SVM, and also can fit with the popular neural net. In terms of the stacking, with a better digitised dataset, we may be able to find some uncorrelated methods to achieve a better ensemble result.

References

Chang, Chih-Chung, and Chih-Jen Lin. 2011. "LIBSVM: A Library for Support Vector Machines." *ACM Transactions on Intelligent Systems and Technology* 2: 27:1–27.

Cortes, Corinna, and Vladimir Vapnik. 1995. *Machine Learning* 20 (3): 273–97. https://doi.org/10.1023/a:1022627411411.

Deane-Mayer, Zachary A., and Jared E. Knowles. 2019. *caretEnsemble: Ensembles of Caret Models*. https://CRAN.R-project.org/package=caretEnsemble.

Hastie, Trevor, Robert Tibshirani, and J H Friedman. 2009. *The Elements of Statistical Learning*. 2nd ed. Springer Series in Statistics. New York, NY: Springer.

Irizarry, Rafael A. 2022. "Introduction to Data Science." *Rafalab*. https://rafalab.github.io/dsbook/.

Nielsen, Michael A. 2015. *Neural Networks and Deep Learning*. Determination Press. http://neuralnetworksanddeeplearning.com/index.html.

Platt, John C. 1998. "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines." *Microsoft Research Technical Report MSR-TR-98-14*.

Therneau, Terry, and Beth Atkinson. 2022. *Rpart: Recursive Partitioning and Regression Trees*. https://CRAN.R-project.org/package=rpart.