Ensemble Methods

Sai Gonuguntla

source: https://www.kaggle.com/datasets/uciml/adult-census-income (https://www.kaggle.com/datasets/uciml/adult-census-income) date: 10/15/22

```
data1 <- read.csv("income_evaluation.csv", header=TRUE)
str(data1)
```

```
## 'data.frame':    32561 obs. of  15 variables:
##  $ age           : int  39 50 38 53 28 37 49 52 31 42 ...
##  $ workclass     : chr  " State-gov" " Self-emp-not-inc" " Private" " Private" ...
##  $ fnlwgt        : int  77516 83311 215646 234721 338409 284582 160187 209642 4578
1 159449 ...
##  $ education     : chr  " Bachelors" " Bachelors" " HS-grad" " 11th" ...
##  $ education.num : int  13 13 9 7 13 14 5 9 14 13 ...
##  $ marital.status: chr  " Never-married" " Married-civ-spouse" " Divorced" " Marri
ed-civ-spouse" ...
##  $ occupation    : chr  " Adm-clerical" " Exec-managerial" " Handlers-cleaners" "
Handlers-cleaners" ...
##  $ relationship  : chr  " Not-in-family" " Husband" " Not-in-family" " Husband" ..
.
##  $ race          : chr  " White" " White" " White" " Black" ...
##  $ sex           : chr  " Male" " Male" " Male" " Male" ...
##  $ capital.gain  : int  2174 0 0 0 0 0 0 0 14084 5178 ...
##  $ capital.loss  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ hours.per.week: int  40 13 40 40 40 40 16 45 50 40 ...
##  $ native.country: chr  " United-States" " United-States" " United-States" " Unite
d-States" ...
##  $ income        : chr  " <=50K" " <=50K" " <=50K" " <=50K" ...
```

Data cleaning

will use the education.num, hours.per.week, age, and income columns, andincome will be the target.

```
df <- data1[,c(1, 5, 13, 15)]
attach(df)
income <- as.factor(income)
str(df)
```

```
## 'data.frame':    32561 obs. of  4 variables:
##  $ age           : int  39 50 38 53 28 37 49 52 31 42 ...
##  $ education.num : int  13 13 9 7 13 14 5 9 14 13 ...
##  $ hours.per.week: int  40 13 40 40 40 40 16 45 50 40 ...
##  $ income        : chr  " <=50K" " <=50K" " <=50K" " <=50K" ...
```

```
head(df)
```

```
##   age education.num hours.per.week income
## 1  39            13             40  <=50K
## 2  50            13             13  <=50K
## 3  38             9             40  <=50K
## 4  53             7             40  <=50K
## 5  28            13             40  <=50K
## 6  37            14             40  <=50K
```

Handle missing values

no missing values

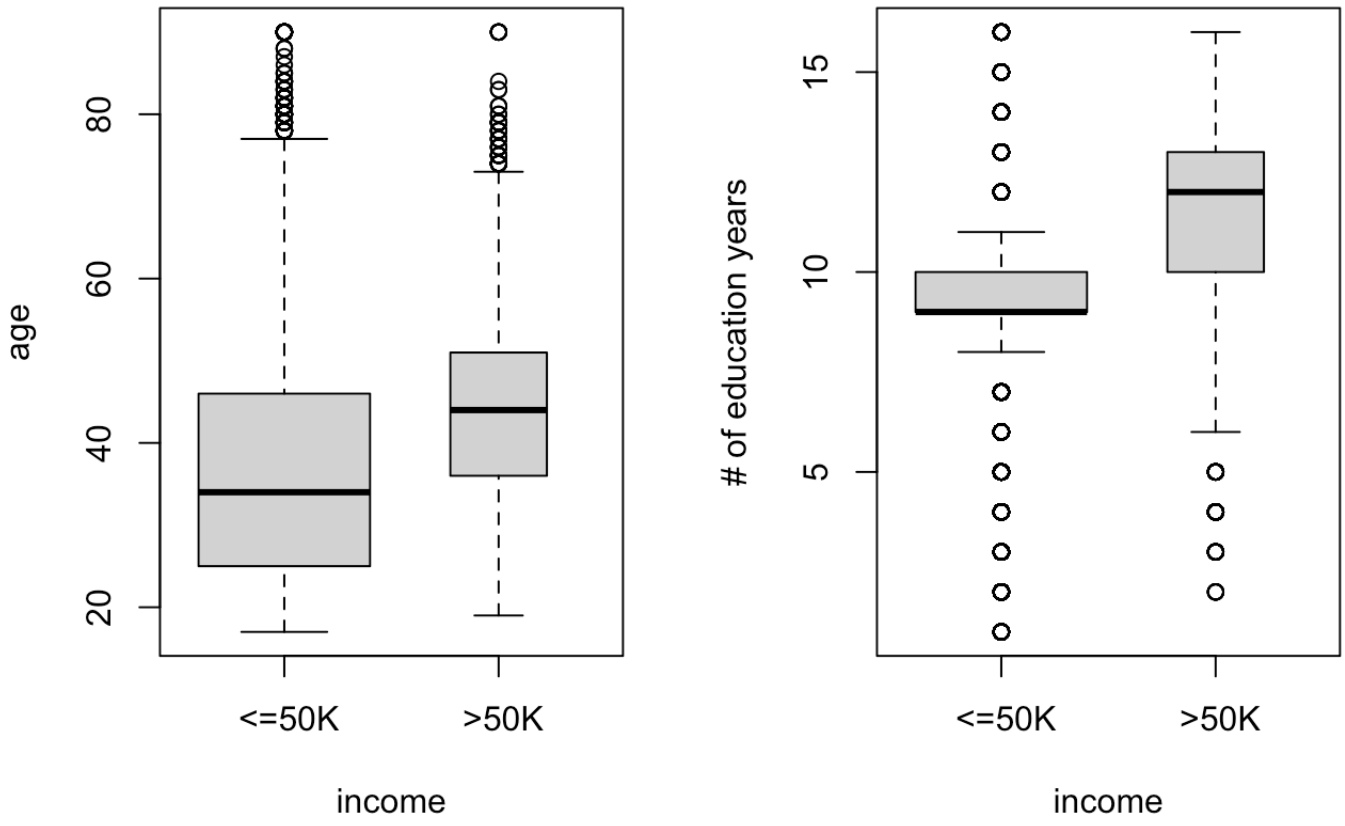
```
sapply(df, function(x) sum(is.na(x)==TRUE))
```

```
##            age  education.num hours.per.week         income
##              0              0              0              0
```

Plotting data

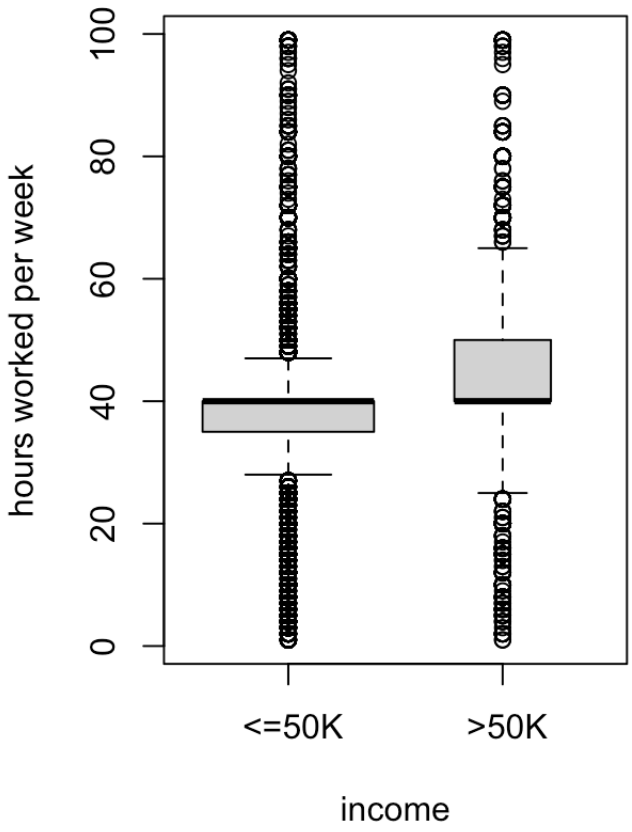
with the predictor age, the median age for a person who makes <=50k is about 35 and for a person who makes >50k is about 43 . With the predictor being # of years educated, the median years for a person who makes <=50k is about 9 and for a person who makes >50k its about 13 . With # of hours worked per week, the median # of hours is about 40 for both someone who makes <50k and for someone for makes >50k.

```
par(mfrow=c(1,2))
plot(income, age, xlab="income", ylab="age", varwidth=TRUE)
plot(income, education.num, xlab="income", ylab="# of education years", varwidth=TRUE
)
```




```
plot(income, hours.per.week, xlab="income", ylab="hours worked per week", varwidth=TR
UE)
summary(df)
```

```
##       age        education.num   hours.per.week     income
##  Min.   :17.00   Min.   : 1.00   Min.   : 1.00   Length:32561
##  1st Qu.:28.00   1st Qu.: 9.00   1st Qu.:40.00   Class :character
##  Median :37.00   Median :10.00   Median :40.00   Mode  :character
##  Mean   :38.58   Mean   :10.08   Mean   :40.44
##  3rd Qu.:48.00   3rd Qu.:12.00   3rd Qu.:45.00
##  Max.   :90.00   Max.   :16.00   Max.   :99.00
```



Train and test

80/20 train and test

```
set.seed(1234)
i <- sample(1:nrow(df), 0.8*nrow(df), replace=FALSE)
train <- df[i,]
test <- df[-i,]
```

Decision Trees

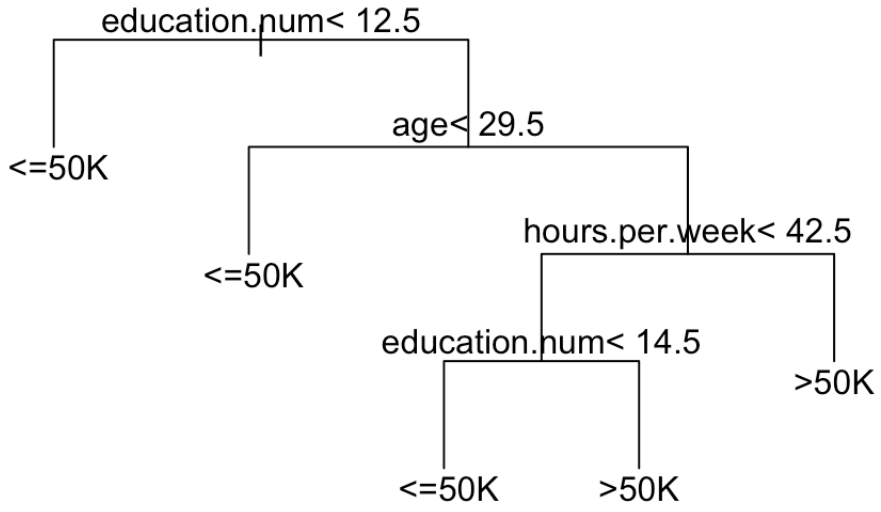
Using rpart

```
library(rpart)
tree1 <- rpart(income~., data=df, method="class")
tree1
```

```
## n= 32561
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 32561 7841  <=50K (0.7591904 0.2408096)
##    2) education.num< 12.5 24494 3932  <=50K (0.8394709 0.1605291) *
##    3) education.num>=12.5 8067 3909  <=50K (0.5154332 0.4845668)
##      6) age< 29.5 1617  232  <=50K (0.8565244 0.1434756) *
##      7) age>=29.5 6450 2773  >50K (0.4299225 0.5700775)
##       14) hours.per.week< 42.5 3501 1667  <=50K (0.5238503 0.4761497)
##         28) education.num< 14.5 3103 1399  <=50K (0.5491460 0.4508540) *
##         29) education.num>=14.5 398  130  >50K (0.3266332 0.6733668) *
##       15) hours.per.week>=42.5 2949  939  >50K (0.3184130 0.6815870) *
```

Plotting the rpart tree

```
plot(tree1, uniform=TRUE, margin=0.2)
text(tree1)
```



Using tree() package with training data

mean accuracy of 0.785 runtime of .217

```
library(tree)

start_time <- Sys.time()
set.seed(1958)
tree2 <- tree(as.factor(income)~., data=train)
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 0.08222103 secs
```

```
tree_pred <- predict(tree2, newdata=test, type="class")
table(tree_pred, test$income)
```

```
##
## tree_pred  <=50K  >50K
##     <=50K   4358   835
##     >50K     564   756
```

```
mean(tree_pred==test$income)
```

```
## [1] 0.7851988
```

Logistic regression on all predictors.

accuracy is .788 and mcc is .340

```
library(mltools)

glm1 <- glm(as.factor(income)~., data=train, family=binomial)
probs <- predict(glm1, newdata=test, type="response")
pred <- ifelse(probs>0.5, 2, 1)
acc_logreg <- mean(pred==as.integer(as.factor(test$income)))
mcc_logreg <- mcc(pred, as.integer(as.factor(test$income)))
print(paste("accuracy=", acc_logreg))
```

```
## [1] "accuracy= 0.788423153692615"
```

```
print(paste("mcc=", mcc_logreg))
```

```
## [1] "mcc= 0.340113083502466"
```

Random Forest

runtime is 22.01 seconds

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
start_time <- Sys.time()

set.seed(1234)
rf <- randomForest(factor(income)~., data=train, importance=TRUE)
rf
```

```
##
## Call:
##  randomForest(formula = factor(income) ~ ., data = train, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 20.06%
## Confusion matrix:
##         <=50K  >50K class.error
##  <=50K  18491  1307  0.06601677
##  >50K    3917  2333  0.62672000
```

```
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 16.23311 secs
```

accuracy= 0.797, mcc= 0.378

```
pred <- predict(rf, newdata=test, type="response")
acc_rf <- mean(pred==test$income)
mcc_rf <- mcc(factor(pred), factor(test$income))
print(paste("accuracy=", acc_rf))
```

```
## [1] "accuracy= 0.797328420082911"
```

```
print(paste("mcc=", mcc_rf))
```

```
## [1] "mcc= 0.377842845418163"
```

fastAdaboost

took 8.211 secs

```
#install.packages("https://cran.r-project.org/src/contrib/Archive/fastAdaboost/fastAdaboost_1.0.0.tar.gz", dependencies=TRUE, repos = NULL, type="source")
library(fastAdaboost)
start_time <- Sys.time()
set.seed(1234)
fadab <- adaboost(income~., train, 10)
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 4.7386 secs
```

```
summary(fadab)
```

```
##                   Length Class   Mode
## formula            3     formula call
## trees             10     -none-  list
## weights           10     -none-  numeric
## classnames         2     -none-  character
## dependent_variable 1     -none-  character
## call               4     -none-  call
```

accuracy= 0.756, mcc= 0.362

```
pred <- predict(fadab, newdata=test, type="response")
# pred$class holds the classification
acc_fadab <- mean(pred$class==test$income)
mcc_fadab <- mcc(pred$class, as.factor(test$income))
print(paste("accuracy=", acc_fadab))
```

```
## [1] "accuracy= 0.756487025948104"
```

```
print(paste("mcc=", mcc_fadab))
```

```
## [1] "mcc= 0.36207177781385"
```

XGBoost

took 1.714 sec

```
library(xgboost)
start_time <- Sys.time()

train_label <- ifelse(train$income==" >50K", 1, 0)
train_matrix <- data.matrix(train[, -4])
model <- xgboost(data=train_matrix, label=train_label,
                 nrounds=100, objective='binary:logistic')
```

```
## [1]  train-logloss:0.578103
## [2]  train-logloss:0.515435
## [3]  train-logloss:0.478265
## [4]  train-logloss:0.454822
## [5]  train-logloss:0.439649
## [6]  train-logloss:0.429694
## [7]  train-logloss:0.422992
## [8]  train-logloss:0.418335
## [9]  train-logloss:0.415160
## [10] train-logloss:0.412917
## [11] train-logloss:0.411266
## [12] train-logloss:0.410092
## [13] train-logloss:0.408939
## [14] train-logloss:0.408099
## [15] train-logloss:0.407684
## [16] train-logloss:0.407187
## [17] train-logloss:0.406711
## [18] train-logloss:0.406264
## [19] train-logloss:0.405917
## [20] train-logloss:0.405749
## [21] train-logloss:0.405379
## [22] train-logloss:0.404935
## [23] train-logloss:0.404792
## [24] train-logloss:0.404504
## [25] train-logloss:0.404131
## [26] train-logloss:0.403637
## [27] train-logloss:0.403452
## [28] train-logloss:0.402853
## [29] train-logloss:0.402652
## [30] train-logloss:0.402327
## [31] train-logloss:0.401967
## [32] train-logloss:0.401274
## [33] train-logloss:0.400999
## [34] train-logloss:0.400884
## [35] train-logloss:0.400596
## [36] train-logloss:0.400432
## [37] train-logloss:0.400371
## [38] train-logloss:0.400203
```

```
## [39] train-logloss:0.399973
## [40] train-logloss:0.399887
## [41] train-logloss:0.399188
## [42] train-logloss:0.398734
## [43] train-logloss:0.398328
## [44] train-logloss:0.397825
## [45] train-logloss:0.397612
## [46] train-logloss:0.397350
## [47] train-logloss:0.397242
## [48] train-logloss:0.396710
## [49] train-logloss:0.396306
## [50] train-logloss:0.395855
## [51] train-logloss:0.395612
## [52] train-logloss:0.395468
## [53] train-logloss:0.395214
## [54] train-logloss:0.394646
## [55] train-logloss:0.394412
## [56] train-logloss:0.393903
## [57] train-logloss:0.393635
## [58] train-logloss:0.393521
## [59] train-logloss:0.393257
## [60] train-logloss:0.393171
## [61] train-logloss:0.393160
## [62] train-logloss:0.392882
## [63] train-logloss:0.392322
## [64] train-logloss:0.392257
## [65] train-logloss:0.392100
## [66] train-logloss:0.391920
## [67] train-logloss:0.391602
## [68] train-logloss:0.391369
## [69] train-logloss:0.391272
## [70] train-logloss:0.391235
## [71] train-logloss:0.390931
## [72] train-logloss:0.390596
## [73] train-logloss:0.390545
## [74] train-logloss:0.390525
## [75] train-logloss:0.390362
## [76] train-logloss:0.390068
## [77] train-logloss:0.390028
## [78] train-logloss:0.389985
## [79] train-logloss:0.389779
## [80] train-logloss:0.389707
## [81] train-logloss:0.389679
## [82] train-logloss:0.389591
## [83] train-logloss:0.389500
## [84] train-logloss:0.389467
## [85] train-logloss:0.389306
```

```
## [86] train-logloss:0.389202
## [87] train-logloss:0.389050
## [88] train-logloss:0.389006
## [89] train-logloss:0.388985
## [90] train-logloss:0.388895
## [91] train-logloss:0.388863
## [92] train-logloss:0.388816
## [93] train-logloss:0.388770
## [94] train-logloss:0.388437
## [95] train-logloss:0.388110
## [96] train-logloss:0.387873
## [97] train-logloss:0.387825
## [98] train-logloss:0.387523
## [99] train-logloss:0.387190
## [100]    train-logloss:0.387119
```

```
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 1.194837 secs
```

accuracy = 0.793, mcc = 0.372

```
test_label <- ifelse(test$income==" >50K", 1, 0)
test_matrix <- data.matrix(test[, -4])

probs <- predict(model, test_matrix)
pred <- ifelse(probs>0.5, 1, 0)

acc_xg <- mean(pred==test_label)
mcc_xg <- mcc(pred, test_label)
print(paste("accuracy=", acc_xg))
```

```
## [1] "accuracy= 0.793489943190542"
```

```
print(paste("mcc=", mcc_xg))
```

```
## [1] "mcc= 0.37184293753649"
```

Based on accuracy,random forest performed the best with an accuracy of 0.797, mcc of 0.378 and run time of 22.01 seconds which is the highest of all runtimes. Then XGBoost with an accuracy of 0.793, mcc of 0.372 and a run time of 1.714 seconds. Followed by adaboost with an accuracy of 0.756, mcc of 0.362 and a

runtime of 8.211 seconds. As the baseline, decision tree also did well with an accuracy of 0.785 and a very quick runtime of 0.217.

Based on mcc random forest performed the best, followed by adaboost followed by XGBoost. Based on runtime, decision tree performed the best, followed by XGBoost, followed by adaboost, and then random forest which had the highest runtime.