# BACS HW17

109090046

2023-06-06

---

This week, we will look at a dataset of US health insurance premium charges. We will build models that can predict what someone's insurance charges might be, given several factors about them. You download the dataset, and find more information about it, at the Kaggle platform where machine learning people like to host challenges and share datasets: US Health Insurance Dataset

---

**Setup**: Download the data, load it in your script, and omit any rows with missing values (NAs)

**Note**: The values of charges are large, so MSE values will be very large. This week let's use RMSE, or the Root-Mean-Square Error (the square-root of MSE), so we have smaller numbers.

---

```
insurance <- read_csv("D:/下載/insurance.csv")
insurance <- na.omit(insurance)
flextable(head(insurance))
```

| age | sex | bmi | children | smoker | region | charges |
|---|---|---|---|---|---|---|
| 19 | female | 27.900 | 0 | yes | southwest | 16,884.924 |
| 18 | male | 33.770 | 1 | no | southeast | 1,725.552 |
| 28 | male | 33.000 | 3 | no | southeast | 4,449.462 |
| 33 | male | 22.705 | 0 | no | northwest | 21,984.471 |
| 32 | male | 28.880 | 0 | no | northwest | 3,866.855 |
| 31 | female | 25.740 | 0 | no | southeast | 3,756.622 |

---

## Question 1) Create some explanatory models to learn more about charges:

---

### a. Create an OLS regression model and report which factors are significantly related to charges.

```
lm_model <- lm(charges ~ ., data = insurance)
summary(lm_model)

##
## Call:
## lm(formula = charges ~ ., data = insurance)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11304.9  -2848.1   -982.1   1393.9  29992.8
```

```
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -11938.5      987.8 -12.086  < 2e-16 ***
## age                 256.9       11.9  21.587  < 2e-16 ***
## sexmale            -131.3      332.9  -0.394 0.693348
## bmi                 339.2       28.6  11.860  < 2e-16 ***
## children            475.5      137.8   3.451 0.000577 ***
## smokeryes         23848.5      413.1  57.723  < 2e-16 ***
## regionnorthwest    -353.0      476.3  -0.741 0.458769
## regionsoutheast   -1035.0      478.7  -2.162 0.030782 *
## regionsouthwest    -960.0      477.9  -2.009 0.044765 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6062 on 1329 degrees of freedom
## Multiple R-squared:  0.7509, Adjusted R-squared:  0.7494
## F-statistic: 500.8 on 8 and 1329 DF,  p-value: < 2.2e-16
```

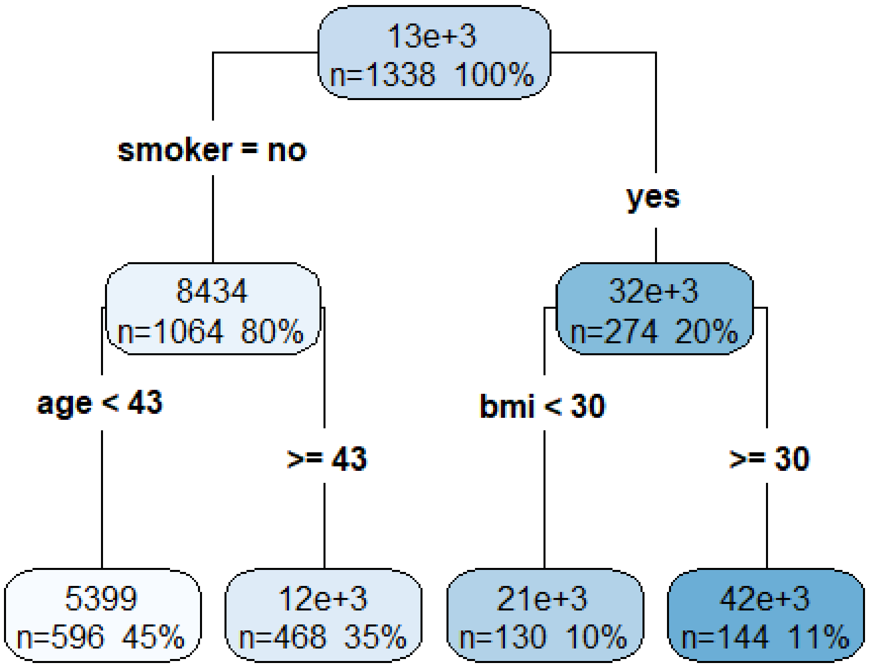
**ANS:** age, `bmi`, `children` and `smokeryes` are are significantly related to charges because of the small p-value.

**b. Create a decision tree (specifically, a regression tree) with default parameters to `rpart()`.**

```
tree <- rpart(charges ~ ., data = insurance)
```

*i.Plot a visual representation of the tree structure*

```
rpart.plot(tree, type=4, extra=101)
```



*ii. How deep is the tree (see nodes with "decisions" – ignore the leaves at the bottom)*

```
tree$control$maxdepth

## [1] 30
```

*iii. How many leaf groups does it suggest to bin the data into?*

```
length(which(tree$frame$var == "<leaf>"))

## [1] 4
```

*iv. What conditions (combination of decisions) describe each leaf group?*

```
as.party(tree)

##
## Model formula:
## charges ~ age + sex + bmi + children + smoker + region
##
## Fitted party:
## [1] root
## |   [2] smoker in no
## |   |   [3] age < 42.5: 5398.850 (n = 596, err = 13198544329.6)
## |   |   [4] age >= 42.5: 12299.887 (n = 468, err = 12505445362.3)
## |   [5] smoker in yes
## |   |   [6] bmi < 30.01: 21369.223 (n = 130, err = 3286654885.0)
## |   |   [7] bmi >= 30.01: 41692.809 (n = 144, err = 4859010033.0)
##
## Number of inner nodes:    3
## Number of terminal nodes: 4
```

**ANS**:

- no smoke under 42.5
- no smoke over 42.5
- smoke bmi<30.01
- smoke bmi >=30.01

---

## Question 2) Let's use LOOCV to see how how our models perform predictively overall

---

```
k_fold_rmse <- function(model_formula, dataset, k = nrow(dataset)) {
    folds <- createFolds(dataset$charges, k = k)
    rmse_values <- vector("numeric", k)
    for(i in 1:k) {
        test_indices <- folds[[i]]
        test_set <- dataset[test_indices, ]
        train_set <- dataset[-test_indices, ]
        model <- update(model_formula, train_set)
        predictions <- predict(model, test_set)
        rmse_values[i] <- sqrt(mean((test_set$charges - predictions)^2))
    }
    return (mean(rmse_values))
}
```

### a. What is the RMSEout for the OLS regression model?

```
## RMSEout for the OLS:  13231.22
```

**b. What is the RMSEout for the decision tree model?**

```
## RMSEout for the decision tree:  13231.22
```

---

For bagging and boosting, we will only use split-sample testing to save time: partition the data to create training and test sets using an 80:20 split. Use the regression model and decision tree you created in Question 1 for bagging and boosting.

---

## Question 3) Let's see if bagging helps our models

---

**a. Implement the bagged_learn(…) and bagged_predict(…) functions using the hints in the class notes and help from your classmates on Teams. Feel free to share your code on Teams to get feedback, or ask others for help.**

```
set.seed(123)

rmse_oos <- function(actuals, preds) {
  sqrt(mean( (actuals - preds)^2 ))
}

bagged_retrain <- function(model, dataset, b){
  resample <- unique(sample(1:nrow(dataset), replace = TRUE))
  train_data <- dataset[resample,]
  train_model <- update(model, data = train_data)
  train_model
}

bagged_learn <- function(model, dataset, b=100){
  lapply(1:b, bagged_retrain, model = model, dataset = dataset)
}

pred <- function(model, dataset, b){
  model = model[[b]]
  predict(model, dataset)
}

bagged_predict <- function(bagged_model, dataset, b){
  prediction <- lapply(1:b, pred, model = bagged_model, dataset = dataset)
  rmse_oos(unlist(prediction), rep(unlist(dataset[7]), times = b))
}

train_indices<- sample(1:nrow(insurance), size = 0.8*nrow(insurance))
train_set <- insurance[train_indices,]
test_set <- insurance[-train_indices,]
```

**b. What is the RMSEout for the bagged OLS regression?**

```
lm_bagged_models <- bagged_learn(lm_model, train_set, 100)
lm_bagged_rmse <- bagged_predict(lm_bagged_models, test_set, 100)

## RMSEout for the bagged OLS regression:  5780.911
```

**c. What is the RMSEout for the bagged decision tree?**

```
tree_bagged_models <- bagged_learn(tree, train_set, 100)
tree_bagged_rmse <- bagged_predict(tree_bagged_models, test_set, 100)

## RMSEout for the bagged decision tree:  5041.874
```

---

## Question 4) Let's see if boosting helps our models. You can use a learning rate of 0.1 and adjust it if you find a better rate.

---

**a. Write boosted_learn(…) and boosted_predict(…) functions using the hints in the class notes and help from your classmates on Teams. Feel free to share your code generously on Teams to get feedback, or ask others for help.**

```
rmse_oos <- function(actuals, preds) {
  sqrt(mean( (actuals - preds)^2 ))
}

boosted_learn <- function(model, dataset, outcome, n=100, rate=0.1, type) {
  # get data frame of only predictor variables
  predictors <- dataset[, -which(names(dataset) %in% outcome)]
  # Initialize residuals and models
  res <- dataset[,outcome] # get vector of actuals to start
  models <- list()
  for (i in 1:n) {
    this_model <- update(model, data = cbind(charges = res, predictors))
    # update residuals with learning rate
    if (type == "l")
    {
      # res <- res - rate * this_model$fitted.values
      res <- res - rate * predict(this_model, newdata = cbind(charges = res, predictors))
    }
    else
    {
      # res <- res - rate * this_model$y
      res <- res - rate * predict(this_model, newdata = cbind(charges = res, predictors))
    }
    models[[i]] <- this_model # Store model
  }
  list(models=models, rate=rate)
}

boosted_predict <- function(boosted_learning, new_data) {
  boosted_models <- boosted_learning$models
  rate <- boosted_learning$rate
  n <- length(boosted_models)



  # get predictions of new_data from each model
  predictions <- lapply(boosted_models, function(model) {
    predict(model, newdata = new_data)
  })
  # convert the list of predictions to a data frame
  pred_frame <- do.call(cbind, predictions)

  # apply a sum over the columns of predictions, weighted by learning rate
  predicted_values <- rate * rowSums(pred_frame)

  a <- sum((predicted_values - new_data[,7])^2) / nrow(new_data)

  return (sqrt(a))
}
```

**b. What is the RMSEout for the boosted OLS regression?**

```
lm_boosted_model <- boosted_learn(lm_model, train_set, outcome = "charges", type = "l")
lm_boosted_rmse <- boosted_predict(lm_boosted_model, test_set)

## RMSEout for the boosted OLS regression:  5763.366
```

**c. What is the RMSEout for the boosted decision tree?**

```
tree_stump <- rpart(charges ~ ., data = train_set, cp=0, maxdepth=1)
tree_boosted_model <- boosted_learn(tree_stump, train_set, outcome="charges", type='t')
tree_boosted_rmse <- boosted_predict(tree_boosted_model, test_set)

## RMSEout for the boosted decision tree:  5716.181
```

---

## Question 5) Let's engineer the best predictive decision trees. Let's repeat the bagging and boosting decision tree several times to see what kind of base tree helps us learn the fastest. But this time, split the data 70:20:10 — use 70% for training, 20% for fine-tuning, and use the last 10% to report the final RMSEout.

---

```
set.seed(123)
n <- nrow(insurance)
train_indices <- sample(1:n, size = round(n*0.7), replace = FALSE)
train_set <- insurance[train_indices, ]
remain_set <- insurance[-train_indices, ]

fine_tune_indices <- sample(1:nrow(remain_set), size = round(nrow(remain_set)*0.67), repl
ace = FALSE)
fine_tune_set <- remain_set[fine_tune_indices, ]
test_set <- remain_set[-fine_tune_indices, ]
```

**a. Repeat the bagging of the decision tree, using a base tree of maximum depth 1, 2, … n, keep training on the 70% training set while the RMSEout of your 20% set keeps dropping; stop when the RMSEout has started increasing again (show prediction error at each depth). Report the final RMSEout using the final 10% of the data as your test set.**

```
signal <- 0 # break the loop or not
maxdepth_ins <- 1 # maxdepth of the tree
maxdepth_vector <- c()
bagged_rmse <- c()

while(signal == 0){
```

```r
  control <- rpart.control(maxdepth = maxdepth_ins, cp = 0)

  # Using code in Question 3
  tree <- rpart(charges ~ ., data = train_set, control = control)
  tree_models <- bagged_learn(tree, train_set, 100)
  tree_bagged_rmse <- bagged_predict(tree_models, fine_tune_set, 100)
  cat("RMSE at", maxdepth_ins, ":", tree_bagged_rmse, "\n")

  # append the result to the vector
  maxdepth_vector <- c(maxdepth_ins)
  bagged_rmse <- c(bagged_rmse, tree_bagged_rmse)

  # To determine to brake the loop or not
  if(length(bagged_rmse) >= 2){
    if(bagged_rmse[maxdepth_ins-1] < bagged_rmse[maxdepth_ins]){
      signal <- 1 # break
    }
  }
  maxdepth_ins <- maxdepth_ins + 1
}
```

```
## RMSE at 1 : 7409.562
## RMSE at 2 : 4794.739
## RMSE at 3 : 4374.247
## RMSE at 4 : 4293.112
## RMSE at 5 : 4325.395
```

```r
best_depth <- maxdepth_ins -2
cat("\nBest depth:", best_depth)
```

```
##
## Best depth: 4
```

```r
control <- rpart.control(maxdepth = best_depth, cp = 0)
tree <- rpart(charges ~ ., data = train_set, control = control)
tree_models <- bagged_learn(tree, train_set, 100)
tree_bagged_rmse <- bagged_predict(tree_models, test_set, 100)
```

```
## RMSEout for the bagged decision tree:  5244.83
```

**b. Repeat the boosting of the decision tree, using a base tree of maximum depth 1, 2, … n, keep training on the 70% training set while the RMSEout of your 20% set keeps dropping; stop when the RMSEout has started increasing again (show prediction error at each depth). Report the final RMSEout using the final 10% of the data as your test set.**

```r
signal <- 0 # break the loop or not
maxdepth_ins <- 1 # maxdepth of the tree
maxdepth_vector <- c()
boost_rmse <- c()

while(signal == 0){
  control <- rpart.control(maxdepth = maxdepth_ins, cp = 0)

  # Using code in Question 4 (c)
  tree_stump <- rpart(charges ~ ., data=train_set, control = control)
  tree_boosted_model <- boosted_learn(tree_stump, train_set, outcome="charges", type='t')
  tree_boosted_rmse <- boosted_predict(tree_boosted_model, fine_tune_set)
  cat("RMSE at", maxdepth_ins, ":", tree_boosted_rmse, "\n")

  # append the result to the vector
  maxdepth_vector <- c(maxdepth_ins)
  boost_rmse <- c(boost_rmse, tree_boosted_rmse)

  # To determine to brake the loop or not
  if(length(boost_rmse) >= 2){
    if(boost_rmse[maxdepth_ins-1] < boost_rmse[maxdepth_ins]){
      signal <- 1 # break
    }
  }
  maxdepth_ins <- maxdepth_ins + 1
}
```

```
## RMSE at 1 : 5697.109
## RMSE at 2 : 4052.626
## RMSE at 3 : 4109.837
```

```r
best_depth <- maxdepth_ins -2
cat("\nBest depth:", best_depth)
```

```
##
## Best depth: 2
```

```r
control <- rpart.control(maxdepth = best_depth, cp = 0)
tree_stump <- rpart(charges ~ ., data=train_set, control = control)
tree_boosted_model <- boosted_learn(tree_stump, train_set, outcome="charges", type='t')
tree_boosted_rmse <- boosted_predict(tree_boosted_model, test_set)
```

```
## RMSEout for the bagged decision tree:  5074.212
```