BACS - HW17

109006241, helped by 109006217

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: https://www.kaggle.com/datasets/teertha/ushealthinsurancedataset

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.

```
set.seed(555)
data = read.csv("insurance.csv")
data = na.omit(data)
formula_full = charges ~ age + factor(sex) + bmi + children + factor(smoker) + factor(region)
head(data, 5)
```

```
##   age    sex    bmi children smoker    region   charges
## 1  19 female 27.900        0    yes southwest 16884.924
## 2  18   male 33.770        1     no southeast  1725.552
## 3  28   male 33.000        3     no southeast  4449.462
## 4  33   male 22.705        0     no northwest 21984.471
## 5  32   male 28.880        0     no northwest  3866.855
```

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(formula_full, data=data)
summary(lm_model)
```

```
##
## Call:
## lm(formula = formula_full, data = data)
##
## 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 ***
## factor(sex)male            -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 ***
## factor(smoker)yes         23848.5      413.1  57.723  < 2e-16 ***
## factor(region)northwest    -353.0      476.3  -0.741 0.458769
## factor(region)southeast   -1035.0      478.7  -2.162 0.030782 *
## factor(region)southwest    -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
```

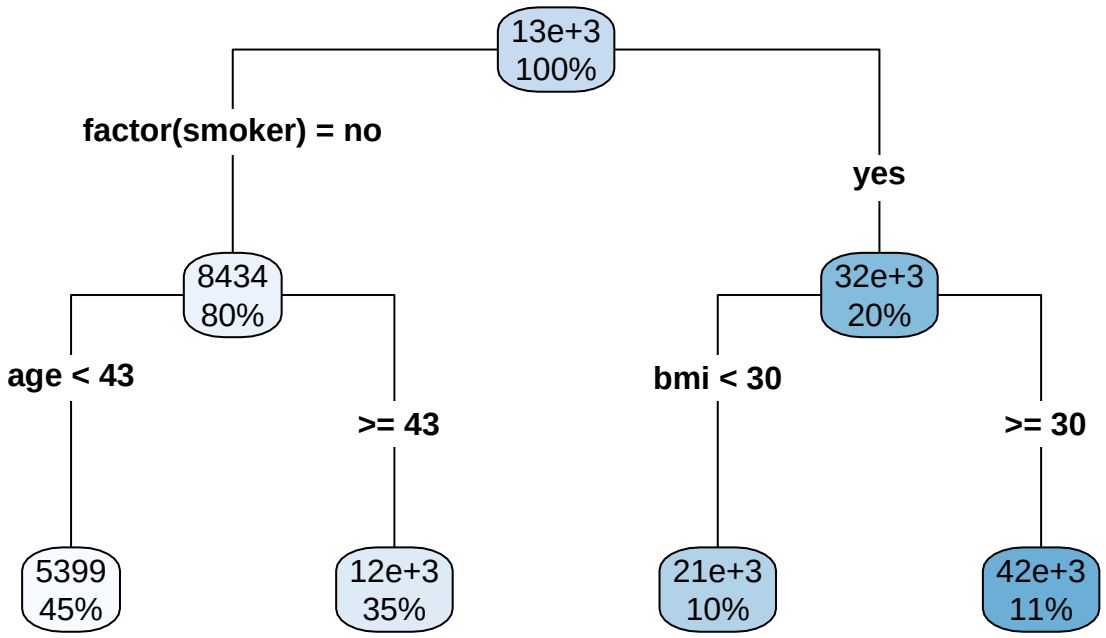
Using a significance level of 0.05, the **age, bmi, children, factor(smoker)yes, factor(region)northwest, factor(region)southeast, and factor(region)southwest** variables are all significantly related to charges.

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

```
tree_model = rpart(formula_full, data=data)
```

i. Plot a visual representation of the tree structure

```
rpart.plot(tree_model, type=4, main="Visual Representation of the Tree Structure")
```



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

The tree has a depth of **2**.

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

4 leaf groups.

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

From left to right:
Leaf 1: Smoker = no, Age < 43
Leaf 2: Smoker = no, Age >= 43
Leaf 3: Smoker = yes, BMI < 30
Leaf 4: Smoker = yes, BMI >= 30

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

```r
fold_i_pe = function(i, k, model, dataset, outcome) {
  folds = cut(1:nrow(dataset), breaks=k, labels=FALSE)

  test_indices = which(folds == i)
  test_set = dataset[test_indices,]
  train_set = dataset[-test_indices,]
  trained_model = update(model, data=train_set)

  predictions = predict(trained_model, test_set)
  dataset[test_indices, outcome] - predictions
}

k_fold_mse = function(model, dataset, outcome, k=nrow(dataset)) {
  shuffled_indicies = sample(1:nrow(dataset))
  dataset = dataset[shuffled_indicies,]

  fold_pred_errors = sapply(1:k, function(kth) {
    fold_i_pe(kth, k, model, dataset, outcome)
  })

  pred_errors = unlist(fold_pred_errors)

  mean(pred_errors^2)
}
```

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

```r
lm_rmseout = sqrt(k_fold_mse(lm_model, data, "charges"))
cat("RMSEout for the OLS regression model =", lm_rmseout, "\n")
```

```
## RMSEout for the OLS regression model = 6087.388
```

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

```r
tree_rmseout = sqrt(k_fold_mse(tree_model, data, "charges"))
cat("RMSEout for the decision tree model =", tree_rmseout, "\n")
```

```
## RMSEout for the decision tree model = 5135.175
```

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.

```r
train_indices = sample(1:nrow(data), size=0.80*nrow(data))
train_set = data[train_indices,]
test_set = data[-train_indices,]
```

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.

```
bagged_learn = function(model, dataset, b=100) {
  lapply(1:b, function(i) {
    # 1. Get a bootstrapped (resampled w/ replacement) dataset
    boot_data = dataset[sample(nrow(dataset), replace=TRUE),]
    # 2. Return a retrained (updated) model
    update(model, data=boot_data)
  })
}

bagged_predict = function(bagged_models, new_data) {
  # get b predictions of new_data
  predictions_list = lapply(bagged_models, function(model) {
    predict(model, new_data)
  })
  # apply a mean over the columns of predictions
  as.data.frame(predictions_list) |> apply(1, mean)
}

mse = function(actual, predictions) {
  mean((actual-predictions)^2)
}
```

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

```
actual = test_set$charges
predictions = unlist(bagged_predict(bagged_learn(lm_model, train_set), test_set))
bagged_lm_rmseout = sqrt(mse(actual, predictions))
cat("RMSEout for the bagged OLS regression =", bagged_lm_rmseout, "\n")
```

```
## RMSEout for the bagged OLS regression = 5423.69
```

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

```
actual = test_set$charges
predictions = unlist(bagged_predict(bagged_learn(tree_model, train_set), test_set))
bagged_tree_rmseout = sqrt(mse(actual, predictions))
cat("RMSEout for the bagged decision trees =", bagged_tree_rmseout, "\n")
```

```
## RMSEout for the bagged decision trees = 4259.306
```

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.

```
boost_learn = function(model, dataset, outcome, n=100, rate=0.1) {
  predictors = dataset[, -which(names(dataset) == outcome)] # get data frame of only predictor variable

  # Initialize residuals and models
  res = dataset[, outcome] # set res to vector of actuals (y) to start
  models = list()

  for (i in 1:n) {
    this_model = update(model, data=cbind(predictors, charges=res))

    y_hat = predict(this_model, cbind(predictors, charges=res))
    res = res - rate*y_hat # update residuals with learning rate

    models[[i]] = this_model # Store model
  }

  list(models=models, rate=rate)
}

boost_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, new_data)
  })
  pred_frame = as.data.frame(predictions) |> unname()

  # apply a sum over the columns of predictions, weighted by learning rate
  pred_frame = lapply(pred_frame, function(column) {
    column * rate
  })
  pred_frame = as.data.frame(pred_frame) |> unname()
  apply(pred_frame, 1, sum)
}
```

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

```
actual = test_set$charges
predictions = unlist(boost_predict(boost_learn(lm_model, train_set, "charges"), test_set))
boosted_lm_rmseout = sqrt(mse(actual, predictions))
cat("RMSEout for the boosted OLS regression =", boosted_lm_rmseout, "\n")
```

```
## RMSEout for the boosted OLS regression = 5421.646
```

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

```
actual = test_set$charges
predictions = unlist(boost_predict(boost_learn(tree_model, train_set, "charges"), test_set))
boosted_tree_rmseout = sqrt(mse(actual, predictions))
cat("RMSEout for the boosted decision trees =", boosted_tree_rmseout, "\n")
```

```
## RMSEout for the boosted decision trees = 3874.532
```

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.

```
# Split data into 70:20:10 for train, val, test
n = nrow(data)
train_indices = sample(n, 0.7*n)
rest_indices = setdiff(1:n, train_indices)
val_indices = sample(rest_indices, 0.2*n)
test_indices = setdiff(rest_indices, val_indices)

train_set = data[train_indices,]
val_set = data[val_indices,]
test_set = data[test_indices,]

cat(" Length of Train Set =", nrow(train_set), "\n",
     "Length of Validation Set =", nrow(val_set), "\n",
     "Length of Test Set =", nrow(test_set), "\n")
```

```
##  Length of Train Set = 936
##  Length of Validation Set = 267
##  Length of Test Set = 135
```

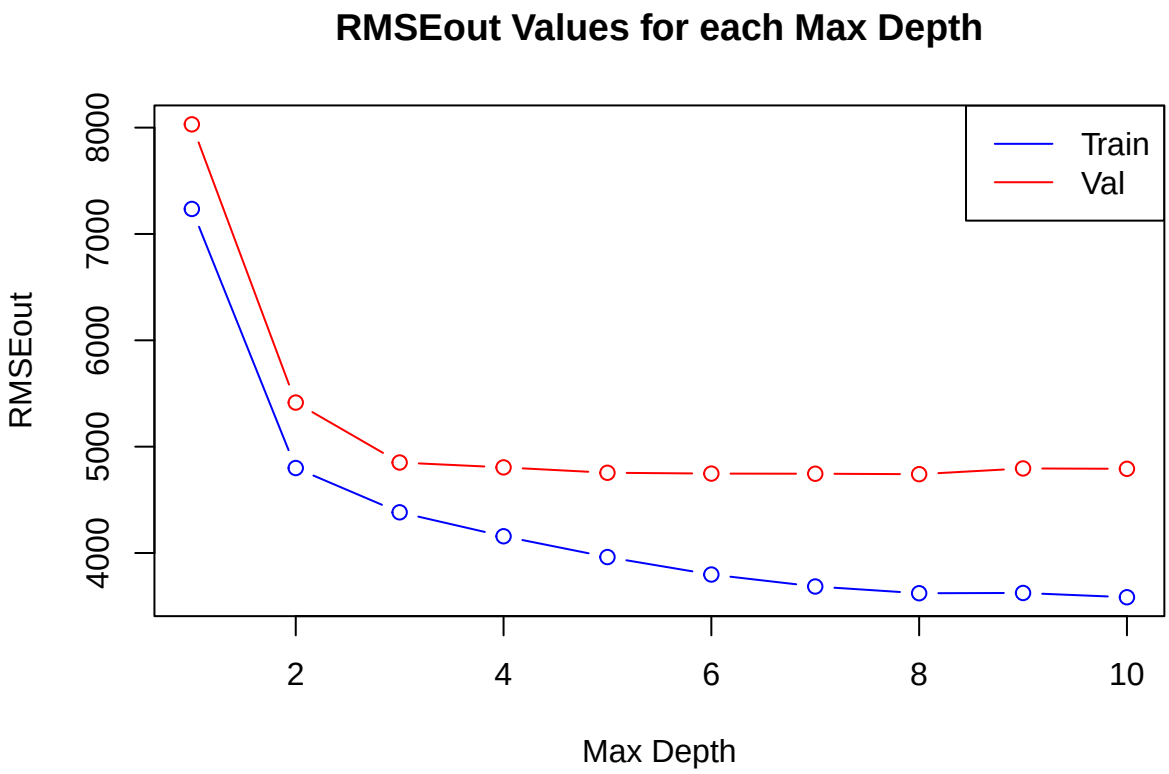
a. Repeat the bagging of the decision tree, using a base tree of maximum depth 1, 2, . . . n, keep training on the 70% training set while the RMSEout of your 20% set keeps dropping; stop when the RMSEout has started increasing again (show prediction error at each depth). Report the final RMSEout using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

```
train_rmseout_list = c()
val_rmseout_list = c()
for(depth in 1:10) {
  base_model = rpart(formula_full, data=data, cp=0, maxdepth=depth)

  train_actual = train_set$charges
```

```
  train_predictions = unlist(bagged_predict(bagged_learn(base_model, train_set), train_set))
  train_bagged_tree_rmseout = sqrt(mse(train_actual, train_predictions))

  train_rmseout_list = append(train_rmseout_list, train_bagged_tree_rmseout)

  val_actual = val_set$charges
  val_predictions = unlist(bagged_predict(bagged_learn(base_model, train_set), val_set))
  val_bagged_tree_rmseout = sqrt(mse(val_actual, val_predictions))

  val_rmseout_list = append(val_rmseout_list, val_bagged_tree_rmseout)

  cat("Val. RMSEout for the bagged trees with max depth", depth,"=", val_bagged_tree_rmseout, "\n")
}
```

```
## Val. RMSEout for the bagged trees with max depth 1 = 8030.711
## Val. RMSEout for the bagged trees with max depth 2 = 5414.809
## Val. RMSEout for the bagged trees with max depth 3 = 4850.964
## Val. RMSEout for the bagged trees with max depth 4 = 4805.201
## Val. RMSEout for the bagged trees with max depth 5 = 4754.581
## Val. RMSEout for the bagged trees with max depth 6 = 4746.776
## Val. RMSEout for the bagged trees with max depth 7 = 4745.377
## Val. RMSEout for the bagged trees with max depth 8 = 4741.112
## Val. RMSEout for the bagged trees with max depth 9 = 4795.133
## Val. RMSEout for the bagged trees with max depth 10 = 4791.564
```

```
minlim = min(c(train_rmseout_list, val_rmseout_list))
maxlim = max(c(train_rmseout_list, val_rmseout_list))
plot(x=1:length(train_rmseout_list), y=train_rmseout_list, type="b", col="blue",
     main="RMSEout Values for each Max Depth", xlab="Max Depth", ylab="RMSEout",
     xlim=c(1,10), ylim=c(minlim, maxlim))
lines(x=1:length(val_rmseout_list), y=val_rmseout_list, type="b", col="red")
legend("topright", legend=c("Train", "Val"), col=c("blue", "red"), lty=1)
```




```
base_model = rpart(formula_full, data=data, cp=0, maxdepth=4)
actual = test_set$charges
predictions = unlist(bagged_predict(bagged_learn(base_model, train_set), test_set))
final_rmseout = sqrt(mse(actual, predictions))
cat("Final RMSEout using the final 10% of the data as test set =", final_rmseout)
```

```
## Final RMSEout using the final 10% of the data as test set = 4697.346
```

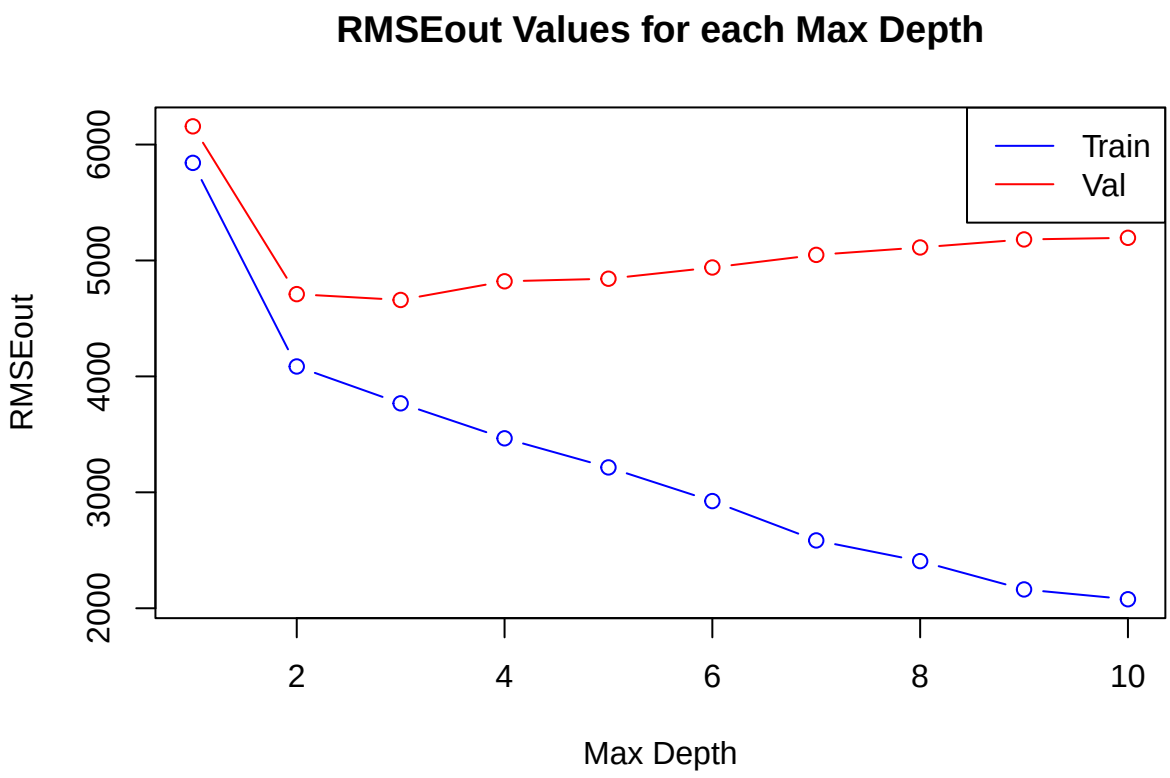
b. Repeat the boosting of the decision tree, using a base tree of maximum depth 1, 2, . . . n, keep training on the 70% training set while the RMSEout of your 20% set keeps dropping; stop when the RMSEout has started increasing again (show prediction error at each depth). Report the final RMSEout using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

```
train_rmseout_list = c()
val_rmseout_list = c()
for(depth in 1:10) {
  base_model = rpart(formula_full, data=data, cp=0, maxdepth=depth)

  train_actual = train_set$charges
  train_predictions = unlist(boost_predict(boost_learn(base_model, train_set, "charges"), train_set))
  train_bagged_tree_rmseout = sqrt(mse(train_actual, train_predictions))

  train_rmseout_list = append(train_rmseout_list, train_bagged_tree_rmseout)

  val_actual = val_set$charges
```

```
  val_predictions = unlist(boost_predict(boost_learn(base_model, train_set, "charges"), val_set))
  val_boosted_tree_rmseout = sqrt(mse(val_actual, val_predictions))

  val_rmseout_list = append(val_rmseout_list, val_boosted_tree_rmseout)

  cat("Val. RMSEout for the boosted trees with max depth", depth,"=", val_boosted_tree_rmseout, "\n")
}
```

```
## Val. RMSEout for the boosted trees with max depth 1 = 6157.068
## Val. RMSEout for the boosted trees with max depth 2 = 4709.669
## Val. RMSEout for the boosted trees with max depth 3 = 4659.435
## Val. RMSEout for the boosted trees with max depth 4 = 4820.391
## Val. RMSEout for the boosted trees with max depth 5 = 4842.767
## Val. RMSEout for the boosted trees with max depth 6 = 4939.454
## Val. RMSEout for the boosted trees with max depth 7 = 5048.355
## Val. RMSEout for the boosted trees with max depth 8 = 5112.377
## Val. RMSEout for the boosted trees with max depth 9 = 5181.539
## Val. RMSEout for the boosted trees with max depth 10 = 5195.498
```

```
minlim = min(c(train_rmseout_list, val_rmseout_list))
maxlim = max(c(train_rmseout_list, val_rmseout_list))
plot(x=1:length(train_rmseout_list), y=train_rmseout_list, type="b", col="blue",
     main="RMSEout Values for each Max Depth", xlab="Max Depth", ylab="RMSEout",
     xlim=c(1,10), ylim=c(minlim, maxlim))
lines(x=1:length(val_rmseout_list), y=val_rmseout_list, type="b", col="red")
legend("topright", legend=c("Train", "Val"), col=c("blue", "red"), lty=1)
```




```
base_model = rpart(formula_full, data=data, cp=0, maxdepth=3)
actual = test_set$charges
predictions = unlist(boost_predict(boost_learn(base_model, train_set, "charges"), test_set))
final_rmseout = sqrt(mse(actual, predictions))
cat("Final RMSEout using the final 10% of the data as test set =", final_rmseout)
```

```
## Final RMSEout using the final 10% of the data as test set = 4798.85
```