Housing Prices Pt 2

Carla M Brookey, Fred Hintz, Kassidie Stokes, Thanh Tran

April 11, 2017

```
tenFoldCrossVal = function(formula, n = 10, data, type, ...){
  #n is the number of desired folds
  #data is the desired dataset
  #type is the type of analysis
  #formula is the formula for analysis

  folds = rep(c(1:n), length = nrow(data))
  folds = sample(folds)

  resultVector = rep(0, length(nrow(data)))

  for (i in 1:n){
    train = data[folds!=i,]
    test = data[folds==i,]
    fit = switch(type,
                 rpart = rpart(formula, data = train, ...),
                 randomForest = randomForest(formula, data = train, ...),
                 lda = lda(formula, data = train, ...),
                 qda = qda(formula, data = train, ...),
                 knn3 = knn3(formula, data = train, ...),
                 glm = glm(formula, data = train, ...),
                 ada = ada(formula, data = train, ...),
                 svm = svm(formula, data = train, ...),
                 gbm = gbm(formula, data = train, ...))

    resultVector[folds == i] = switch(type,
                                      rpart = predict(fit, newdata = test, ...),
                                      randomForest = predict(fit, newdata = test, ...),
                                      lda = predict(fit, data = test, ...)$class,
                                      qda = predict(fit, data = test, ...)$class,
                                      knn3 = predict(fit, data = test, ...),
                                      glm = predict(fit, data = test, ...),
                                      ada = predict(fit, data = test, ...),
                                      svm = predict(fit, data = test, ...),
                                      gbm = predict(fit, data = test, ...))
  }
  return(resultVector)
}

reverse_transformed_response <- function(response)
{
  return(exp(response) - 1)
}
```

R Markdown

Run with log(SalePrice).

```
#Load data
train <- read.csv('train.csv', sep = ',', header = TRUE)
test <- read.csv('test.csv', sep = ',', header = TRUE)

#Load Packages
library(rpart)
library(randomForest)
library(gbm)
library(caret)
library(e1071)
library(dplyr)
library(sparklyr)
library(neuralnet)
```

```
##
## Attaching package: 'neuralnet'

## The following object is masked from 'package:dplyr':
##
##     compute
```

```
library(glmnet)
```

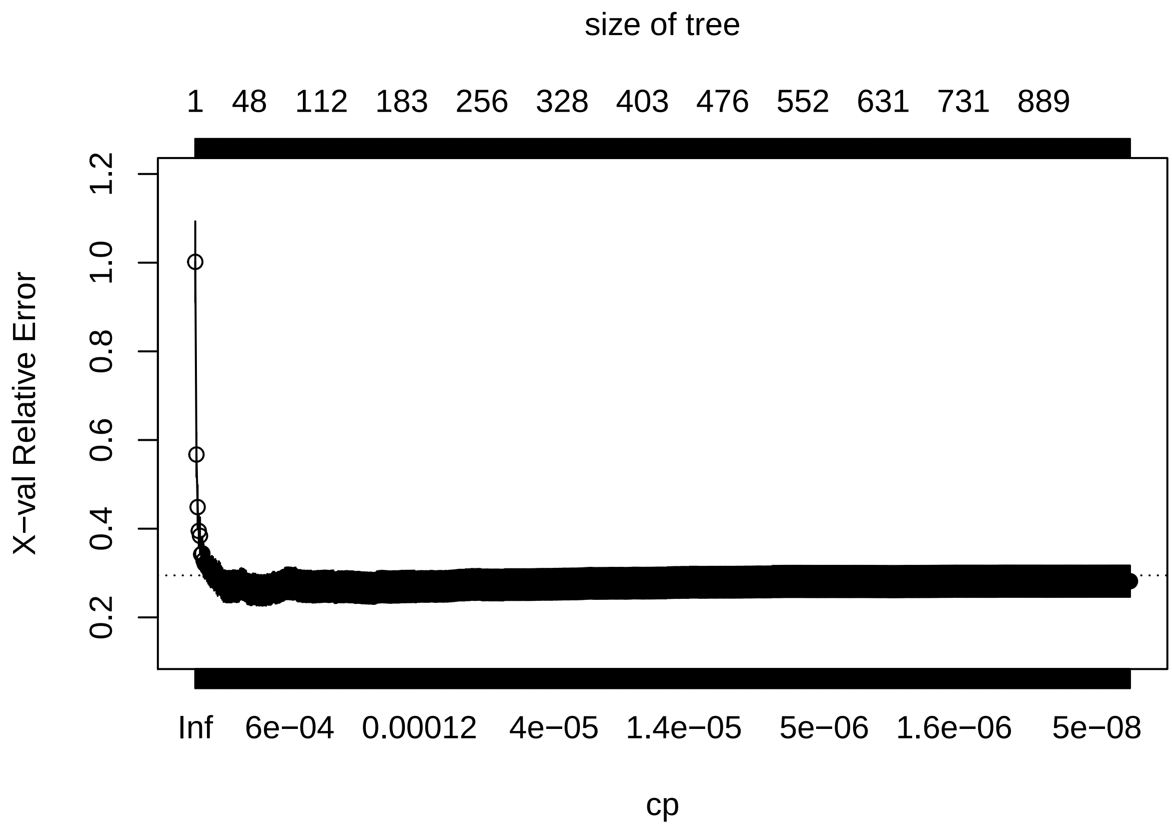
```
## Loading required package: Matrix

## Loading required package: foreach

## Loaded glmnet 2.0-5
```

```
library(ModelMetrics)
```

```
##
## Attaching package: 'ModelMetrics'

## The following object is masked from 'package:glmnet':
##
##     auc

## The following objects are masked from 'package:caret':
##
##     confusionMatrix, precision, recall, sensitivity, specificity
```

```
#Carla's doing random forest + gbm and uploading the 10-fold cv code

#than's doing xgboost
#kassidie's doing svm
#Fred doing neural nets


#Create a validation set
#' Splits data.frame into arbitrary number of groups
#'
#' @param dat The data.frame to split into groups
#' @param props Numeric vector. What proportion of the data should
#'              go in each group?
```

```r
#' @param which.adjust Numeric. Which group size should we 'fudge' to
#'              make sure that we sample enough (or not too much)
split_data <- function(dat, props = c(.8, .15, .05), which.adjust = 1){

    # Make sure proportions are positive
    # and the adjustment group isn't larger than the number
    # of groups specified
    stopifnot(all(props >= 0), which.adjust <= length(props))

    # could check to see if the sum is 1
    # but this is easier
    props <- props/sum(props)
    n <- nrow(dat)
    # How large should each group be?
    ns <- round(n * props)
    # The previous step might give something that
    # gives sum(ns) > n so let's force the group
    # specified in which.adjust to be a value that
    # makes it so that sum(ns) = n
    ns[which.adjust] <- n - sum(ns[-which.adjust])

    ids <- rep(1:length(props), ns)
    # Shuffle ids so that the groups are randomized
    which.group <- sample(ids)
    split(dat, which.group)
}

split = split_data(train, c(0.8, 0.2))
train1 = split$'1'
val1 = split$'2'

full.tree = rpart(SalePrice ~ . -Id,
                  data = train1,
                  method = "anova",
                  control = rpart.control(cp = 0.0, minsplit = 2))
plotcp(full.tree)
```




```
fit.tree = rpart(SalePrice ~ . -Id,
                 data = train1,
                 method = "anova",
                 control = rpart.control(cp = 0.0055, minsplit = 2))

sqrt(mean((val1$SalePrice - predict(fit.tree, val1, type = "vector"))^2))
```

```
## [1] 42707.64
```

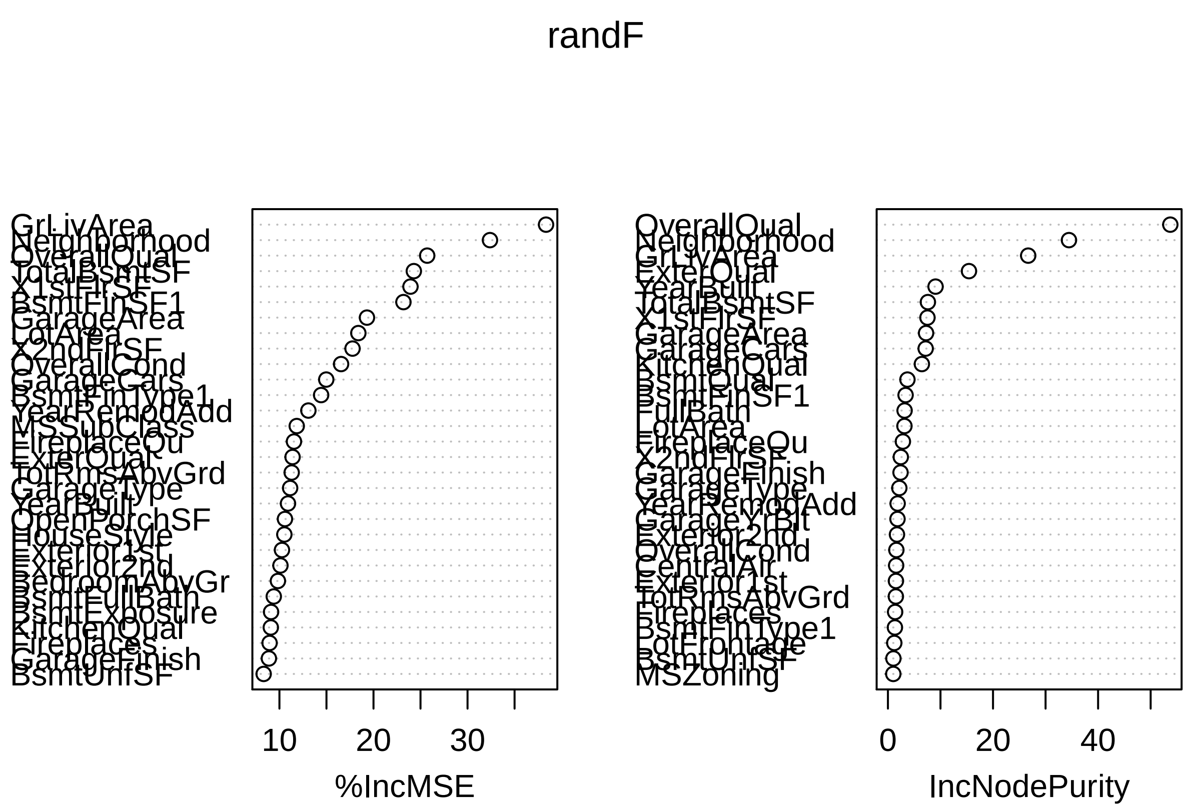
```
#Try again with the full dataset and the resubstitution error
fit.tree1 = rpart(SalePrice ~ . -Id,
                  data = train,
                  method = "anova",
                  control = rpart.control(cp = 0.0055, minsplit = 2))

sqrt(mean((train$SalePrice - predict(fit.tree1, train, type = "vector"))^2))
```

```
## [1] 31640.31
```

Minimum xerror at cp 46 1.299978e-03 47 8.908836e-02 0.2499940 0.02580239 Using the 1 SE rule, we get 18 5.573976e-03 17 1.643091e-01 0.2697831 0.02541736

```
#Must first deal with NA values
deal_missing_values <- function(dataSet, ntrain = 1460, type = 1)
{
  # type is the replace method for numeric values.
  #if type = 1 --> using mean to replace.
  #if type = 2 --> using median to replace.
```

```r
  dataSet[] <- lapply(dataSet, function(x){
    # check if variables have a factor:
    if(!is.factor(x)) {
      #replace NA by mean
      if (type == 1) x[is.na(x)] <- mean(x[1:1460], na.rm = TRUE)
      else if (type == 2) if (type == 1) x[is.na(x)] <- median(x[1:1460], na.rm = TRUE)
    }
    else {
      # otherwise include NAs into factor levels and change factor levels:
      x <- factor(x, exclude=NULL)
      levels(x)[is.na(levels(x))] <- "Missing"
    }
    return(x)
  })

  return(dataSet)
}

train2 = deal_missing_values(train)

randF = randomForest(log(SalePrice) ~ . -Id,
                     data = train2,
                     importance = TRUE)
varImpPlot(randF)
```



Based on the variable importance plot, use OverallQual, Neighborhood, GrLivArea, ExterQual, TotalBsmtSF, GarageCars, GarageArea, x1stFlrSF, KitchenQual, and YearBuilt.

```
rf = tenFoldCrossVal(log(SalePrice) ~ OverallQual + Neighborhood + GrLivArea +  ExterQual +
                      TotalBsmtSF + GarageCars + GarageArea + X1stFlrSF + KitchenQual +
                      YearBuilt,
                    n = 10,
                    data = train2,
                    type = "randomForest")
sol = reverse_transformed_response(rf)
rmse(actual = train$SalePrice, predicted = sol)
```

```
## [1] 29705.69
```

```
rf2 = tenFoldCrossVal(SalePrice ~ OverallQual + Neighborhood + GrLivArea +  ExterQual +
                       TotalBsmtSF + GarageCars + GarageArea + X1stFlrSF + KitchenQual +
                       YearBuilt,
                     n = 10,
                     data = train2,
                     type = "randomForest")
rmse(actual = train$SalePrice, predicted = rf2)
```

```
## [1] 29572.73
```

Unfortunately, neither of these worked very well just on their own, with the selected variables.

Now I'll try GBM on the selected variables.

```
#gbm = tenFoldCrossVal(log(SalePrice) ~ OverallQual + Neighborhood + GrLivArea +  ExterQual +
#                            TotalBsmtSF + GarageCars + GarageArea + X1stFlrSF + KitchenQual +
#                            YearBuilt,
#                       n = 10,
#                       data = train,
#                       type = "gbm",
#                       n.trees = 100)

gbm1 = gbm(log(SalePrice) ~ OverallQual + Neighborhood + GrLivArea +  ExterQual +
                           TotalBsmtSF + GarageCars + GarageArea + X1stFlrSF + KitchenQual +
                           YearBuilt,
           distribution = "gaussian",
           data = train2,
           n.trees = 100)
sol.gbm1 = reverse_transformed_response(predict(gbm1, newdata = train, n.trees = 100))
rmse(actual = train$SalePrice, predicted = sol.gbm1)
```

```
## [1] 77935.13
```

```
gbm2 = gbm(SalePrice ~ OverallQual + Neighborhood + GrLivArea +  ExterQual +
                           TotalBsmtSF + GarageCars + GarageArea + X1stFlrSF + KitchenQual +
                           YearBuilt,
           distribution = "gaussian",
           data = train2,
           n.trees = 100)
sol.gbm2 = predict(gbm2, newdata = train, n.trees = 100)
rmse(actual = train$SalePrice, predicted = sol.gbm2)
```

```
## [1] 76029.18
```

In the case of the tree based methods, the log transformed response is not going as well as the untransformed data. Further analysis will be completed without transforming SalePrice.

```
#fitControl = trainControl(method = "cv", number = 10)
#gbmGrid1 = expand.grid(n.trees = c(25, 50, 75, 100),
#                        interaction.depth = c(10, 12, 14, 16),
#                        shrinkage = c(0.01, 0.05, 0.1, 0.2),
#                        n.minobsinnode = 10)

#tune1 = train(SalePrice ~ OverallQual + Neighborhood + GrLivArea +  ExterQual +
#                 TotalBsmtSF + GarageCars + GarageArea + X1stFlrSF + KitchenQual +
#                 YearBuilt,
#               method = "gbm",
#               tuneGrid = gbmGrid1,
#               trControl = fitControl,
#               data = train2)
#tune1

#gbmGrid2 = expand.grid(n.trees = c(90, 95, 100, 105),
#                        interaction.depth = c(13, 14, 15),
#                        shrinkage = c(0.04, 0.05, 0.06),
#                        n.minobsinnode = 10)

#tune2 = train(SalePrice ~ OverallQual + Neighborhood + GrLivArea +  ExterQual +
#                 TotalBsmtSF + GarageCars + GarageArea + X1stFlrSF + KitchenQual +
#                 YearBuilt,
#               method = "gbm",
#               tuneGrid = gbmGrid2,
#               trControl = fitControl,
#               data = train2)
#tune2

#gbmGrid3 = expand.grid(n.trees = c(85, 90, 95),
#                        interaction.depth = c(12, 13, 14),
#                        shrinkage = c(0.05, 0.055, 0.06),
#                        n.minobsinnode = 10)

#tune3 = train(SalePrice ~ OverallQual + Neighborhood + GrLivArea +  ExterQual +
#                 TotalBsmtSF + GarageCars + GarageArea + X1stFlrSF + KitchenQual +
#                 YearBuilt,
#               method = "gbm",
#               tuneGrid = gbmGrid3,
#               trControl = fitControl,
#               data = train2)
#tune3

#gbmGrid4 = expand.grid(n.trees = c(90, 95, 100),
#                        interaction.depth = 13,
#                        shrinkage = c(0.055, 0.06, 0.065),
#                        n.minobsinnode = 10)

#tune4 = train(SalePrice ~ OverallQual + Neighborhood + GrLivArea +  ExterQual +
#                 TotalBsmtSF + GarageCars + GarageArea + X1stFlrSF + KitchenQual +
#                 YearBuilt,
#               method = "gbm",
```

```r
#                tuneGrid = gbmGrid4,
#                trControl = fitControl,
#                data = train2)
#tune4

gbm3 = gbm(SalePrice ~ OverallQual + Neighborhood + GrLivArea +  ExterQual +
                          TotalBsmtSF + GarageCars + GarageArea + X1stFlrSF + KitchenQual +
                          YearBuilt,
           distribution = "gaussian",
           data = train2,
           n.trees = 90,
           shrinkage = 0.055,
           interaction.depth = 13)
sol.gbm3 = predict(gbm3, newdata = train, n.trees = 90)
rmse(actual = train$SalePrice, predicted = sol.gbm3)
```

```
## [1] 21579.09
```

```r
train3 = encode_categorical_variables(train2, ntrain = 1460)
train4 = do_variable_selection(train3)
```

```
## [1] "Important variables:"
##  [1] "GarageType_bit1" "CentralAir_bit1" "MSZoning_bit3"
##  [4] "OverallQual"     "YearBuilt"       "YearRemodAdd"
##  [7] "BsmtFinSF1"      "TotalBsmtSF"     "X1stFlrSF"
## [10] "GrLivArea"       "BsmtFullBath"    "Fireplaces"
## [13] "GarageCars"      "GarageArea"
```

```r
rfd1 = tenFoldCrossVal(SalePrice ~ .,
                       n = 10,
                       data = train4,
                       type = "randomForest")
rmse(actual = train4$SalePrice, predicted = rfd1)
```

```
## [1] 29672.08
```

```r
gbmD1 = gbm(SalePrice ~ .,
            data = train4,
            distribution = "gaussian",
            n.trees = 100)

sol.gbmD1 = predict(gbmD1, newdata = train3, n.trees = 100)
rmse(actual = train$SalePrice, predicted = sol.gbmD1)
```

```
## [1] 76037.15
```

Not terrible, but let's see what tuning does.

```r
#fitControl = trainControl(method = "cv", number = 10)
#gbmGrid1 = expand.grid(n.trees = c(25, 50, 75, 100),
#                       interaction.depth = c(10, 12, 14, 16),
#                       shrinkage = c(0.01, 0.05, 0.1, 0.2),
#                       n.minobsinnode = 10)

#tuneD1 = train(SalePrice ~ .,
#               method = "gbm",
#               tuneGrid = gbmGrid1,
```

```r
#                trControl = fitControl,
#                data = train4)
#tuneD1

#gbmGrid2 = expand.grid(n.trees = c(90, 95, 100, 105, 110),
#                        interaction.depth = c(13, 14, 15),
#                        shrinkage = c(0.04, 0.05, 0.06),
#                        n.minobsinnode = 10)

#tuneD2 = train(SalePrice ~ .,
#                method = "gbm",
#                tuneGrid = gbmGrid2,
#                trControl = fitControl,
#                data = train4)
#tuneD2

#gbmGrid3 = expand.grid(n.trees = c(95, 100, 105),
#                        interaction.depth = c(1, 5, 10),
#                        shrinkage = c(0.04, 0.05, 0.06),
#                        n.minobsinnode = 10)

#tuneD3 = train(SalePrice ~ .,
#                method = "gbm",
#                tuneGrid = gbmGrid3,
#                trControl = fitControl,
#                data = train4)
#tuneD3

#gbmGrid4 = expand.grid(n.trees = 100,
#                        interaction.depth = 14,
#                        shrinkage = c(0.045, 0.05, 0.055),
#                        n.minobsinnode = c(1, 5, 10))

#tuneD4 = train(SalePrice ~ .,
#                method = "gbm",
#                tuneGrid = gbmGrid4,
#                trControl = fitControl,
#                data = train4)
#tuneD4

#gbmGrid5 = expand.grid(n.trees = c(90, 100, 110),
#                        interaction.depth = c(13, 14, 15),
#                        shrinkage = c(0.049, 0.05, 0.051),
#                        n.minobsinnode = 1)

#tuneD5 = train(SalePrice ~ .,
#                method = "gbm",
#                tuneGrid = gbmGrid5,
#                trControl = fitControl,
#                data = train4)
#tuneD5

gbmD2 = gbm(SalePrice ~ .,
```

```
            distribution = "gaussian",
            data = train4,
            n.trees = 100,
            shrinkage = 0.049,
            interaction.depth = 14,
            n.minobsinnode = 1)
sol.gbmD2 = predict(gbmD2, newdata = train4, n.trees = 100)
rmse(actual = train4$SalePrice, predicted = sol.gbmD2)
```

```
## [1] 16919.88
```

Clearly the tuned gbm on the dummy variables following variable selection did the best.

Now for curiousity's sake, I will run the test data through all of my models for submission to Kaggle.

```
test2 = deal_missing_values(dataSet = test, ntrain = 1459)
test3 = encode_categorical_variables(test2, ntrain = 1459)

test2$SalePrice <- predict(gbm3, test2, n.trees = 90)
submission <- data.frame(Id <- test2$Id, SalePrice <- test2$SalePrice)
names(submission) <- c('Id', 'SalePrice')
write.csv(file = 'submissionGBM.csv', x = submission, row.names = FALSE)

test3$SalePrice <- predict(gbmD2, test3, n.trees = 100)
submission <- data.frame(Id <- test3$Id, SalePrice <- test3$SalePrice)
names(submission) <- c('Id', 'SalePrice')
write.csv(file = 'submissionGBMDummy.csv', x = submission, row.names = FALSE)
```

The GBM with random forest variable selection and no dummy variables gave a kaggle result of 0.16.

The GBM with lasso variable selection and using dummy variables gave a Kaggle result of 0.14.

Neither is better than the linear regression result.

```
tenFoldCrossVal = function(formula, n = 10, data, type, nnn=1, ...){
  #n is the number of desired folds
  #data is the desired dataset
  #type is the type of analysis
  #formula is the formula for analysis
  require(dplyr)
  folds = rep(c(1:n), length = nrow(data))
  folds = sample(folds)

  resultVector = rep(0, length(nrow(data)))

  for (i in 1:n){
    train = data[folds!=i,]
    test = data[folds==i,]
    fit = switch(type,
                 rpart = rpart(formula, data = train, ...),
                 randomForest = randomForest(formula, data = train, ...),
                 lda = lda(formula, data = train, ...),
                 qda = qda(formula, data = train, ...),
                 knn3 = knn3(formula, data = train, ...),
                 glm = glm(formula, data = train, ...),
                 ada = ada(formula, data = train, ...),
                 svm = svm(formula, data = train, ...),
```

```r
                 gbm = gbm(formula, data = train, ...),
                 neunet = neuralnet(formula,data=train,hidden=c(nnn)))

    resultVector[folds == i] = switch(type,
                                      rpart = predict(fit, newdata = test, ...),
                                      randomForest = predict(fit, newdata = test, ...),
                                      lda = predict(fit, data = test, ...)$class,
                                      qda = predict(fit, data = test, ...)$class,
                                      knn3 = predict(fit, data = test, ...),
                                      glm = predict(fit, data = test, ...),
                                      ada = predict(fit, data = test, ...),
                                      svm = predict(fit, data = test, ...),
                                      gbm = predict(fit, data = test, ...),
                                    neunet = compute(fit,covariate=select(test,-SalePrice))$net.result)
  }
  return(resultVector)
}
```

```r
train_dum<-encode_categorical_variables(train2)
ntrain<-1460
train_dum_reduced<-do_variable_selection(train_dum)
```

```
## [1] "Important variables:"
##  [1] "GarageType_bit1" "CentralAir_bit1" "MSZoning_bit3"
##  [4] "LotArea"         "OverallQual"     "OverallCond"
##  [7] "YearBuilt"       "YearRemodAdd"    "BsmtFinSF1"
## [10] "TotalBsmtSF"     "X1stFlrSF"       "GrLivArea"
## [13] "BsmtFullBath"    "Fireplaces"      "GarageCars"
## [16] "GarageArea"      "WoodDeckSF"
```

```r
vrs<-names(train_dum_reduced)

vrs[!vrs %in% "SalePrice"]
```

```
##  [1] "GarageType_bit1" "CentralAir_bit1" "MSZoning_bit3"
##  [4] "LotArea"         "OverallQual"     "OverallCond"
##  [7] "YearBuilt"       "YearRemodAdd"    "BsmtFinSF1"
## [10] "TotalBsmtSF"     "X1stFlrSF"       "GrLivArea"
## [13] "BsmtFullBath"    "Fireplaces"      "GarageCars"
## [16] "GarageArea"      "WoodDeckSF"
```

```r
nn_form<-as.formula(paste("SalePrice ~", paste(vrs[!vrs %in% "SalePrice"],collapse= " + ")))

hpmeans<-apply(train_dum_reduced,2,mean)

train_dum_scaled<-as.data.frame(scale(train_dum_reduced, center=T,scale=T))

hnn1<-neuralnet(nn_form,data=train_dum_scaled,linear.output=T)

nn_cv_1node<-tenFoldCrossVal(formula=nn_form,data=train_dum_scaled,type="neunet",nnn=1)

nn_unscaled<-nn_cv_1node*sd(train_dum_reduced$SalePrice)+mean(train_dum_reduced$SalePrice)

RMSE<-sqrt(sum((train_dum_reduced$SalePrice - nn_unscaled)**2)/1460)
```

```r
#RMSE is extremely bad

test1<-deal_missing_values(test)
test_dum<-encode_categorical_variables(test1)
reduced_vars<-which(colnames(test_dum) %in% colnames(train_dum_reduced))

test_dum_reduced<-test_dum[,reduced_vars]

test_dum_scaled<-scale(test_dum_reduced)

prediction_nn<-compute(hnn1,covariate=test_dum_scaled)$net.result

prediction_nn<-prediction_nn*sd(train$SalePrice)+mean(train$SalePrice)
submission_nn<-data.frame(test$Id,prediction_nn)
colnames(submission_nn)<-c("ID","SalePrice")

write.csv(submission_nn,"submission_nn.csv",row.names = FALSE)
```

RMLSE for the 1 node, 1 layer neural net is .16521