Final Project Phase 1 Work

Christopher Dillard

2023-07-02

### Using Predictive Models to Predict Whether a Home is Sold Above the Median Sales Price

### Loading Libraries and Data Cleaning

library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.2 ✔ readr 2.1.4  
## ✔ forcats 1.0.0 ✔ stringr 1.5.0  
## ✔ ggplot2 3.4.2 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.2 ✔ tidyr 1.3.0  
## ✔ purrr 1.0.1   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(tidymodels)

## ── Attaching packages ────────────────────────────────────── tidymodels 1.1.0 ──  
## ✔ broom 1.0.4 ✔ rsample 1.1.1  
## ✔ dials 1.2.0 ✔ tune 1.1.1  
## ✔ infer 1.0.4 ✔ workflows 1.1.3  
## ✔ modeldata 1.1.0 ✔ workflowsets 1.0.1  
## ✔ parsnip 1.1.0 ✔ yardstick 1.2.0  
## ✔ recipes 1.0.6   
## ── Conflicts ───────────────────────────────────────── tidymodels\_conflicts() ──  
## ✖ scales::discard() masks purrr::discard()  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ recipes::fixed() masks stringr::fixed()  
## ✖ dplyr::lag() masks stats::lag()  
## ✖ yardstick::spec() masks readr::spec()  
## ✖ recipes::step() masks stats::step()  
## • Use suppressPackageStartupMessages() to eliminate package startup messages

library(glmnet)

## Loading required package: Matrix  
##   
## Attaching package: 'Matrix'  
##   
## The following objects are masked from 'package:tidyr':  
##   
## expand, pack, unpack  
##   
## Loaded glmnet 4.1-7

library(usemodels)

## Warning: package 'usemodels' was built under R version 4.3.1

library(e1071)

##   
## Attaching package: 'e1071'  
##   
## The following object is masked from 'package:tune':  
##   
## tune  
##   
## The following object is masked from 'package:rsample':  
##   
## permutations  
##   
## The following object is masked from 'package:parsnip':  
##   
## tune

library(ROCR) #for threshold selction

## Warning: package 'ROCR' was built under R version 4.3.1

ames = read\_csv("ames\_student-1.csv")

## Rows: 2053 Columns: 81  
## ── Column specification ────────────────────────────────────────────────────────  
## Delimiter: ","  
## chr (47): MS\_SubClass, MS\_Zoning, Street, Alley, Lot\_Shape, Land\_Contour, Ut...  
## dbl (34): Lot\_Frontage, Lot\_Area, Year\_Built, Year\_Remod\_Add, Mas\_Vnr\_Area, ...  
##   
## ℹ Use `spec()` to retrieve the full column specification for this data.  
## ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

#Variable for whether a property has been remodeled:  
ames <- ames %>%  
 mutate(  
 Remodeled = ifelse(Year\_Remod\_Add > Year\_Built, "Remodeled", "Not Remodeled"),  
 Remodeled = factor(Remodeled)  
 )  
#Variable for the construction decade:  
ames <- ames %>%  
 mutate(  
 Decade = floor(Year\_Built / 10) \* 10)

### Feature Selection for the Model

ames\_features = ames %>%  
 dplyr::select(Above\_Median, Remodeled, Year\_Built, Year\_Remod\_Add, Decade, Neighborhood, MS\_Zoning,Lot\_Area, Overall\_Cond, Year\_Sold,TotRms\_AbvGrd,Year\_Built,Fence,Bedroom\_AbvGr,First\_Flr\_SF,Gr\_Liv\_Area,Full\_Bath)

str(ames\_features)

## tibble [2,053 × 16] (S3: tbl\_df/tbl/data.frame)  
## $ Above\_Median : chr [1:2053] "Yes" "No" "Yes" "Yes" ...  
## $ Remodeled : Factor w/ 2 levels "Not Remodeled",..: 1 1 1 1 2 1 1 2 2 1 ...  
## $ Year\_Built : num [1:2053] 1960 1961 1958 1968 1997 ...  
## $ Year\_Remod\_Add: num [1:2053] 1960 1961 1958 1968 1998 ...  
## $ Decade : num [1:2053] 1960 1960 1950 1960 1990 1990 1990 1990 1990 1990 ...  
## $ Neighborhood : chr [1:2053] "North\_Ames" "North\_Ames" "North\_Ames" "North\_Ames" ...  
## $ MS\_Zoning : chr [1:2053] "Residential\_Low\_Density" "Residential\_High\_Density" "Residential\_Low\_Density" "Residential\_Low\_Density" ...  
## $ Lot\_Area : num [1:2053] 31770 11622 14267 11160 13830 ...  
## $ Overall\_Cond : chr [1:2053] "Average" "Above\_Average" "Above\_Average" "Average" ...  
## $ Year\_Sold : num [1:2053] 2010 2010 2010 2010 2010 2010 2010 2010 2010 2010 ...  
## $ TotRms\_AbvGrd : num [1:2053] 7 5 6 8 6 7 5 5 6 5 ...  
## $ Fence : chr [1:2053] "No\_Fence" "Minimum\_Privacy" "No\_Fence" "No\_Fence" ...  
## $ Bedroom\_AbvGr : num [1:2053] 3 2 3 3 3 3 2 2 3 2 ...  
## $ First\_Flr\_SF : num [1:2053] 1656 896 1329 2110 928 ...  
## $ Gr\_Liv\_Area : num [1:2053] 1656 896 1329 2110 1629 ...  
## $ Full\_Bath : num [1:2053] 1 1 1 2 2 2 2 2 2 1 ...

summary(ames\_features)

## Above\_Median Remodeled Year\_Built Year\_Remod\_Add  
## Length:2053 Not Remodeled:1085 Min. :1875 Min. :1950   
## Class :character Remodeled : 968 1st Qu.:1953 1st Qu.:1965   
## Mode :character Median :1972 Median :1993   
## Mean :1971 Mean :1984   
## 3rd Qu.:2000 3rd Qu.:2004   
## Max. :2010 Max. :2010   
## Decade Neighborhood MS\_Zoning Lot\_Area   
## Min. :1870 Length:2053 Length:2053 Min. : 1300   
## 1st Qu.:1950 Class :character Class :character 1st Qu.: 7500   
## Median :1970 Mode :character Mode :character Median : 9548   
## Mean :1966 Mean : 10258   
## 3rd Qu.:2000 3rd Qu.: 11600   
## Max. :2010 Max. :215245   
## Overall\_Cond Year\_Sold TotRms\_AbvGrd Fence   
## Length:2053 Min. :2006 Min. : 3.000 Length:2053   
## Class :character 1st Qu.:2007 1st Qu.: 5.000 Class :character   
## Mode :character Median :2008 Median : 6.000 Mode :character   
## Mean :2008 Mean : 6.442   
## 3rd Qu.:2009 3rd Qu.: 7.000   
## Max. :2010 Max. :15.000   
## Bedroom\_AbvGr First\_Flr\_SF Gr\_Liv\_Area Full\_Bath   
## Min. :0.000 Min. : 432 Min. : 480 Min. :0.000   
## 1st Qu.:2.000 1st Qu.: 882 1st Qu.:1137 1st Qu.:1.000   
## Median :3.000 Median :1088 Median :1447 Median :2.000   
## Mean :2.855 Mean :1168 Mean :1499 Mean :1.564   
## 3rd Qu.:3.000 3rd Qu.:1402 3rd Qu.:1737 3rd Qu.:2.000   
## Max. :6.000 Max. :5095 Max. :5095 Max. :4.000

### Converting Character Variables into Factors & Additional Cleaning:

ames\_features = ames\_features %>% mutate(Neighborhood=as\_factor(Neighborhood),MS\_Zoning=as\_factor(MS\_Zoning),Overall\_Cond=as\_factor(Overall\_Cond),Above\_Median=as\_factor(Above\_Median),Fence=as\_factor(Fence))  
  
#Reordering levels in Above\_Median Variable:  
#str(ames\_features)  
#summary(ames\_features)  
levels(ames\_features$Above\_Median)

## [1] "Yes" "No"

ames\_features = ames\_features %>% mutate(Above\_Median = fct\_relevel(Above\_Median,c("No","Yes")))  
levels(ames\_features$Above\_Median)

## [1] "No" "Yes"

### Looking at Correlations between numerical data first to compare later:

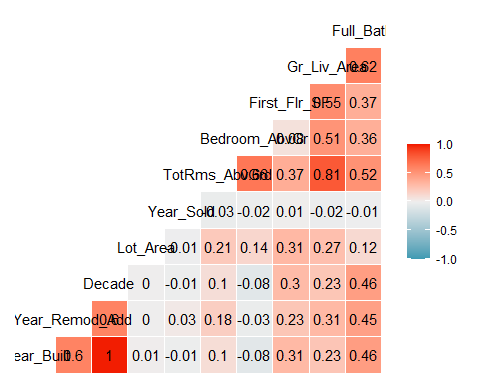
library(GGally)

## Warning: package 'GGally' was built under R version 4.3.1

## Registered S3 method overwritten by 'GGally':  
## method from   
## +.gg ggplot2

ggcorr(ames\_features, label = "TRUE", label\_round = 2)

## Warning in ggcorr(ames\_features, label = "TRUE", label\_round = 2): data in  
## column(s) 'Above\_Median', 'Remodeled', 'Neighborhood', 'MS\_Zoning',  
## 'Overall\_Cond', 'Fence' are not numeric and were ignored



This plot actually doesn’t tell us anything that we wouldn’t have probably assumed. Year\_Built and Decade are of course the 100% correlated. Gr\_Liv\_Area and Total Rooms Above Ground are also very correlated with each other at 0.81, which makes sense if we assume that more square footage means more potential for rooms.This plot does tell us that we will potentially need to eliminate some variables due to the risk of multicollinearity. This is clear with the Decade and Year\_Built variables. So, more than likely I will eliminate Decade of a predictor variable in my model. As for the Gr\_Liv\_Area and TotRms\_AbvGrd, I’ll run the model first to check for overfitting and other signs of multicollinearity in the summary to act accordingly.

ames\_features = ames\_features %>%  
 dplyr::select(-Decade, -Remodeled)

Since I began working in Phase 2, I realized that I didn’t choose the best variables for predicting whether a property sells above the Median price or not. For this phase I will first run the model on more optimal variables and then later compare the first model, which will have more variables and provide a more “holistic” view. The other half of this work will be just on these variables:

1. Neighborhood
2. Lot\_Area
3. Year\_Built and Year\_Remod\_Add, if applicable
4. Overall\_Cond (Condition)
5. MS\_Zoning (primarily to compare low-density and high-density later in Phase 2 of the Project)

Since there is a general trade-off between having more variables and less variables in Random Forests, I think this will serve to be an interesting point that could address the potential for overfitting with too many variables, while more variables also give us the ability to explore more relationships between variables that otherwise would be lost it we omit them. A smaller set of variables lowers the risk of overfitting, but could easily misrepresent data by being too general, leading to the issue of Omitted-Variable Bias, in addition to leaving out variables that absolutely play a significant role in predicting the response variable.

### Extra steps prior to building the models:

Here I’m just exploring the original data set again to check what type of data is in each column before continuing.

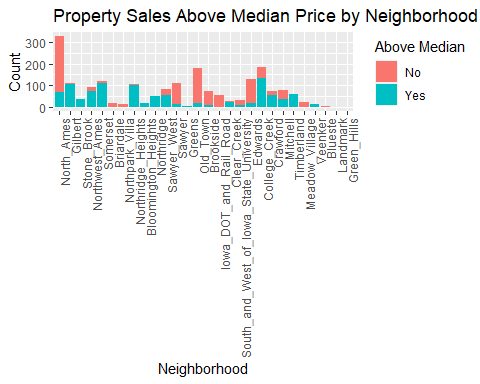
str(ames\_features)

## tibble [2,053 × 14] (S3: tbl\_df/tbl/data.frame)  
## $ Above\_Median : Factor w/ 2 levels "No","Yes": 2 1 2 2 2 2 2 2 2 2 ...  
## $ Year\_Built : num [1:2053] 1960 1961 1958 1968 1997 ...  
## $ Year\_Remod\_Add: num [1:2053] 1960 1961 1958 1968 1998 ...  
## $ Neighborhood : Factor w/ 28 levels "North\_Ames","Gilbert",..: 1 1 1 1 2 2 3 3 2 2 ...  
## $ MS\_Zoning : Factor w/ 7 levels "Residential\_Low\_Density",..: 1 2 1 1 1 1 1 1 1 1 ...  
## $ Lot\_Area : num [1:2053] 31770 11622 14267 11160 13830 ...  
## $ Overall\_Cond : Factor w/ 9 levels "Average","Above\_Average",..: 1 2 2 1 1 2 1 1 3 1 ...  
## $ Year\_Sold : num [1:2053] 2010 2010 2010 2010 2010 2010 2010 2010 2010 2010 ...  
## $ TotRms\_AbvGrd : num [1:2053] 7 5 6 8 6 7 5 5 6 5 ...  
## $ Fence : Factor w/ 5 levels "No\_Fence","Minimum\_Privacy",..: 1 2 1 1 2 1 1 1 3 1 ...  
## $ Bedroom\_AbvGr : num [1:2053] 3 2 3 3 3 3 2 2 3 2 ...  
## $ First\_Flr\_SF : num [1:2053] 1656 896 1329 2110 928 ...  
## $ Gr\_Liv\_Area : num [1:2053] 1656 896 1329 2110 1629 ...  
## $ Full\_Bath : num [1:2053] 1 1 1 2 2 2 2 2 2 1 ...

With a better picture of what I’m working with, I can create some nice descriptive graphs to include in the presentation.

ggplot(ames\_features, aes(x = Neighborhood, fill = Above\_Median)) +  
 geom\_bar(binwidth = 8) +  
 labs(x = "Neighborhood", y = "Count", fill = "Above Median") +  
 ggtitle("Property Sales Above Median Price by Neighborhood") +  
 theme(axis.text.x = element\_text(angle = 90, hjust = 1))

## Warning in geom\_bar(binwidth = 8): Ignoring unknown parameters: `binwidth`

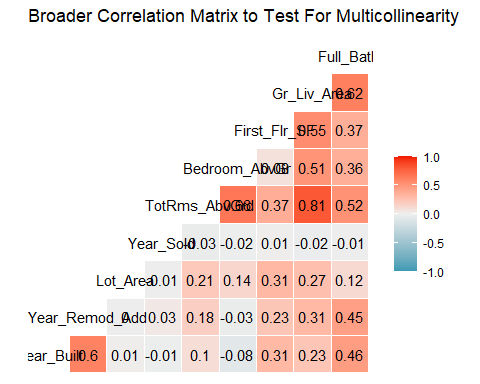


Seeing which neighborhoods have more above\_median sales for neighborhoods with higher counts is fairly easy, but this is not the case for neighborhoods with less sales.

One tool that is useful in measuring the relationship between variable prior to constructing model is the correlation matrix. It is limited to just the numerical variables, which is not a problem for this dataset since there are plenty of those that we can gain some valuable insight from.

One factor that could cause a model to overfit the dataset is Multicollinearity (ie. where variable have a higher correlation between eachother than with the response variable). Let’s graph the variables in a heat map to visualize this concept (using only variables with numerical data):

library(GGally)  
ames\_features\_ggcorr = ames\_features %>%  
 dplyr::select(Year\_Built,Year\_Remod\_Add,Lot\_Area,Year\_Sold,TotRms\_AbvGrd,Bedroom\_AbvGr,First\_Flr\_SF,Gr\_Liv\_Area,Full\_Bath)  
 ggcorr(ames\_features\_ggcorr, label = "TRUE", label\_round = 2) +  
 labs(title = "Broader Correlation Matrix to Test For Multicollinearity")

 Our highest correlated pair now is Gr\_Liv\_Area and TotRms\_AbvGr.

One method of overcoming the presence of multicollunearity and selecting the best variables for our models is by using Lasso Regression. By using this technique we can also tackle the potential issue of overfitting due to an excessively complex model that has high degrees of variability, which can cause the model to overfit the data by modeling random noise instead of the intended outputs.

### Preparing data for the 3 project models (Lasso,Classification Trees, and Random Forest): splitting into training and testing sets:

set.seed(1234)  
amesft\_split = initial\_split(ames\_features,prop=.7,strata=Above\_Median)  
train = training(amesft\_split)  
test=testing(amesft\_split)

ames\_features = ames\_features %>%  
 mutate(Above\_Median = as\_factor(Above\_Median))

### Lasso Regression

Lasso Regression will help with selecting the best variables to avoid multicolinearity.

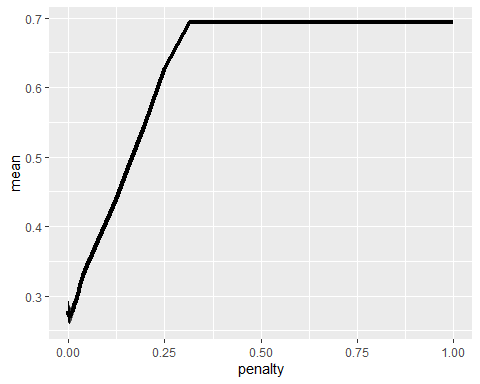
set.seed(1234)  
folds = vfold\_cv(train,v=10)

With this model, my hope is to see if Lasso weeds out any extraneous variables to them by importance.

lasso\_recipe <-   
 recipe(formula = Above\_Median ~ Year\_Built +Year\_Remod\_Add+Lot\_Area+Year\_Sold+TotRms\_AbvGrd+Bedroom\_AbvGr+First\_Flr\_SF+Gr\_Liv\_Area+Full\_Bath+Neighborhood, data = train) %>%   
 step\_dummy(all\_nominal(), -all\_outcomes()) %>%   
 step\_zv(all\_predictors(), -all\_nominal()) %>%  
 step\_normalize(all\_predictors(), -all\_nominal())   
  
lasso\_spec <-   
 logistic\_reg(penalty = tune(), mixture = 1) %>%   
 set\_mode("classification") %>%   
 set\_engine("glmnet")   
  
lasso\_workflow <-   
 workflow() %>%   
 add\_recipe(lasso\_recipe) %>%   
 add\_model(lasso\_spec)   
  
lasso\_grid = grid\_regular(penalty(), levels = 100)  
  
#note the use of alternative metric (min log loss)  
lasso\_tune <-   
 tune\_grid(lasso\_workflow, resamples = folds,   
 grid = lasso\_grid, metrics = metric\_set(mn\_log\_loss))

lasso\_tune %>%  
 collect\_metrics() %>%  
 ggplot(aes(penalty, mean)) +  
 geom\_errorbar(aes(  
 ymin = mean - std\_err,  
 ymax = mean + std\_err  
 ),  
 alpha = 0.5  
 ) +  
 geom\_line(size = 1.5) +  
 theme(legend.position = "none")

## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.  
## ℹ Please use `linewidth` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.



Clearly tbe best penalty is extremely small so blowing out is futil. Thankfully, I can use the following code:

best\_mnlog = lasso\_tune %>%  
 select\_best("mn\_log\_loss")  
best\_mnlog

## # A tibble: 1 × 2  
## penalty .config   
## <dbl> <chr>   
## 1 0.00298 Preprocessor1\_Model075

final\_lasso = lasso\_workflow %>% finalize\_workflow(best\_mnlog)

lasso\_fit = fit(final\_lasso, train)

options(scipen = 999)  
lasso\_fit %>%  
 extract\_fit\_parsnip() %>%  
 pluck("fit") %>%   
 coef(s = best\_mnlog$penalty) #show the coefficients for our selected lambda value

## 35 x 1 sparse Matrix of class "dgCMatrix"  
## s1  
## (Intercept) 0.269733896  
## Year\_Built 1.013147736  
## Year\_Remod\_Add 0.501973707  
## Lot\_Area 0.389081128  
## Year\_Sold .   
## TotRms\_AbvGrd -0.130387589  
## Bedroom\_AbvGr -0.316102160  
## First\_Flr\_SF 0.272077248  
## Gr\_Liv\_Area 1.977586893  
## Full\_Bath 0.332362770  
## Neighborhood\_Gilbert 0.379896883  
## Neighborhood\_Stone\_Brook 0.240308321  
## Neighborhood\_Northwest\_Ames 0.266240883  
## Neighborhood\_Somerset 0.115403014  
## Neighborhood\_Briardale -0.141220722  
## Neighborhood\_Northpark\_Villa -0.173575052  
## Neighborhood\_Northridge\_Heights 0.319274742  
## Neighborhood\_Bloomington\_Heights 0.052264004  
## Neighborhood\_Northridge .   
## Neighborhood\_Sawyer\_West .   
## Neighborhood\_Sawyer -0.056578312  
## Neighborhood\_Greens 0.201433519  
## Neighborhood\_Old\_Town -0.155296981  
## Neighborhood\_Brookside 0.105305213  
## Neighborhood\_Iowa\_DOT\_and\_Rail\_Road -0.002575749  
## Neighborhood\_Clear\_Creek 0.031404323  
## Neighborhood\_South\_and\_West\_of\_Iowa\_State\_University 0.009446956  
## Neighborhood\_Edwards -0.250644311  
## Neighborhood\_College\_Creek 0.086953215  
## Neighborhood\_Crawford 0.395437115  
## Neighborhood\_Mitchell .   
## Neighborhood\_Timberland 0.288566536  
## Neighborhood\_Meadow\_Village -0.116233686  
## Neighborhood\_Veenker 0.108141985  
## Neighborhood\_Blueste 0.039911271

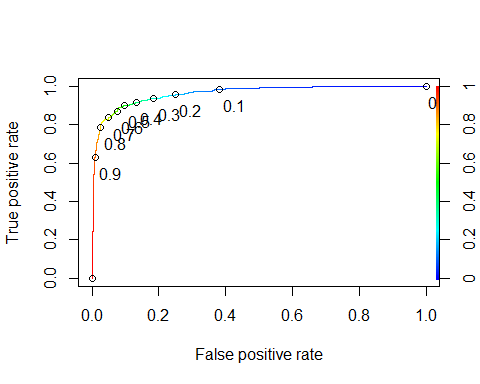
options(scipen = 0)

Year sold was dropped in the model. My initially hope was to check for many some influence from the housing bubble in the 2000s and see if prices were greatly affected in Ames. Other than that, a few neighborhoods were eliminated from the model’s equation.

tidy(lasso\_fit)

## # A tibble: 35 × 3  
## term estimate penalty  
## <chr> <dbl> <dbl>  
## 1 (Intercept) 0.270 0.00298  
## 2 Year\_Built 1.01 0.00298  
## 3 Year\_Remod\_Add 0.502 0.00298  
## 4 Lot\_Area 0.389 0.00298  
## 5 Year\_Sold 0 0.00298  
## 6 TotRms\_AbvGrd -0.130 0.00298  
## 7 Bedroom\_AbvGr -0.316 0.00298  
## 8 First\_Flr\_SF 0.272 0.00298  
## 9 Gr\_Liv\_Area 1.98 0.00298  
## 10 Full\_Bath 0.332 0.00298  
## # ℹ 25 more rows

predictions = predict(lasso\_fit, train, type="prob")[2]  
  
ROCRpred = prediction(predictions, train$Above\_Median)   
  
###You shouldn't need to ever change the next two lines:  
ROCRperf = performance(ROCRpred, "tpr", "fpr")  
plot(ROCRperf, colorize=TRUE, print.cutoffs.at=seq(0,1,by=0.1), text.adj=c(-0.2,1.7))



#Determine threshold to balance sensitivity and specificity  
#DO NOT modify this code  
opt.cut = function(perf, pred){  
 cut.ind = mapply(FUN=function(x, y, p){  
 d = (x - 0)^2 + (y-1)^2  
 ind = which(d == min(d))  
 c(sensitivity = y[[ind]], specificity = 1-x[[ind]],   
 cutoff = p[[ind]])  
 }, perf@x.values, perf@y.values, pred@cutoffs)  
}  
print(opt.cut(ROCRperf, ROCRpred))

## [,1]  
## sensitivity 0.8876712  
## specificity 0.9207921  
## cutoff 0.5746120

### Classification Tree

library(mice) #package for imputation

##   
## Attaching package: 'mice'

## The following object is masked from 'package:stats':  
##   
## filter

## The following objects are masked from 'package:base':  
##   
## cbind, rbind

library(VIM) #visualizing missingness

## Loading required package: colorspace

## Loading required package: grid

## The legacy packages maptools, rgdal, and rgeos, underpinning this package  
## will retire shortly. Please refer to R-spatial evolution reports on  
## https://r-spatial.org/r/2023/05/15/evolution4.html for details.  
## This package is now running under evolution status 0

## VIM is ready to use.

## Suggestions and bug-reports can be submitted at: https://github.com/statistikat/VIM/issues

##   
## Attaching package: 'VIM'

## The following object is masked from 'package:recipes':  
##   
## prepare

## The following object is masked from 'package:datasets':  
##   
## sleep

library(rpart) #for classification trees

##   
## Attaching package: 'rpart'

## The following object is masked from 'package:dials':  
##   
## prune

library(rpart.plot) #for plotting trees  
library(RColorBrewer) #better visualization of classification trees  
library(rattle) #better visualization of classification trees

## Loading required package: bitops

##   
## Attaching package: 'bitops'

## The following object is masked from 'package:Matrix':  
##   
## %&%

## Rattle: A free graphical interface for data science with R.  
## Versión 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.  
## Escriba 'rattle()' para agitar, sacudir y rotar sus datos.

##   
## Attaching package: 'rattle'

## The following object is masked from 'package:VIM':  
##   
## wine

library(caret)

## Loading required package: lattice

##   
## Attaching package: 'caret'

## The following objects are masked from 'package:yardstick':  
##   
## precision, recall, sensitivity, specificity

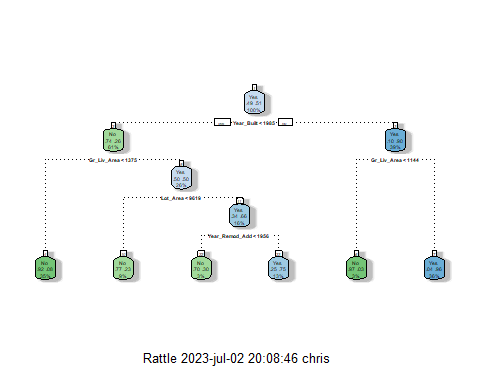
## The following object is masked from 'package:purrr':  
##   
## lift

#Classification Tree  
  
tree\_recipe = recipe(Above\_Median ~., train) %>%  
 step\_dummy(all\_nominal(),-all\_outcomes())%>%  
 step\_zv(all\_predictors(), -all\_nominal())  
  
tree\_model = decision\_tree() %>%   
 set\_engine("rpart", model = TRUE) %>% #don't forget the model = TRUE flag  
 set\_mode("classification")  
  
tree\_wflow =   
 workflow() %>%   
 add\_model(tree\_model) %>%   
 add\_recipe(tree\_recipe)  
  
tree\_fit = fit(tree\_wflow, train)

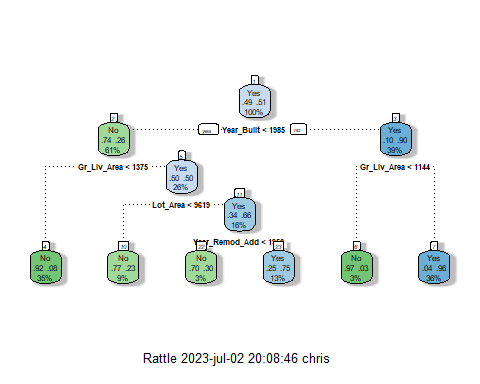
tree = tree\_fit %>%   
 pull\_workflow\_fit() %>%   
 pluck("fit")

## Warning: `pull\_workflow\_fit()` was deprecated in workflows 0.2.3.  
## ℹ Please use `extract\_fit\_parsnip()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.

fancyRpartPlot(tree)



fancyRpartPlot(tree, tweak=1.5) #tweak makes the tree a little easier to read



There are 5 splits here.

# Complexity Parameter and the xerror, cross validated error. best xerror is the smallest one yielded by the correspondig CP  
  
tree\_fit$fit$fit$fit$cptable

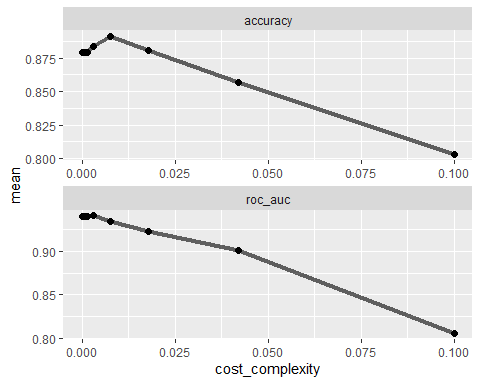
## CP nsplit rel error xerror xstd  
## 1 0.60678925 0 1.0000000 1.0381895 0.02680266  
## 2 0.05304102 1 0.3932107 0.4158416 0.02162962  
## 3 0.04809052 3 0.2871287 0.3889675 0.02109219  
## 4 0.02687412 4 0.2390382 0.2545969 0.01774827  
## 5 0.01000000 5 0.2121641 0.2277228 0.01691183

set.seed(1234)  
folds = vfold\_cv(train, v = 5)

tree\_recipe = recipe(Above\_Median ~., train) %>%  
 step\_dummy(all\_nominal(),-all\_outcomes())%>%  
 step\_zv(all\_predictors(), -all\_nominal())  
  
tree\_model = decision\_tree(cost\_complexity = tune()) %>%   
 set\_engine("rpart", model = TRUE) %>% #don't forget the model = TRUE flag  
 set\_mode("classification")  
  
tree\_grid = grid\_regular(cost\_complexity(),  
 levels = 25) #try 25 sensible values for cp  
  
tree\_wflow =   
 workflow() %>%   
 add\_model(tree\_model) %>%   
 add\_recipe(tree\_recipe)  
  
tree\_res =   
 tree\_wflow %>%   
 tune\_grid(  
 resamples = folds,  
 grid = tree\_grid  
 )  
  
tree\_res

## # Tuning results  
## # 5-fold cross-validation   
## # A tibble: 5 × 4  
## splits id .metrics .notes   
## <list> <chr> <list> <list>   
## 1 <split [1149/288]> Fold1 <tibble [50 × 5]> <tibble [0 × 3]>  
## 2 <split [1149/288]> Fold2 <tibble [50 × 5]> <tibble [0 × 3]>  
## 3 <split [1150/287]> Fold3 <tibble [50 × 5]> <tibble [0 × 3]>  
## 4 <split [1150/287]> Fold4 <tibble [50 × 5]> <tibble [0 × 3]>  
## 5 <split [1150/287]> Fold5 <tibble [50 × 5]> <tibble [0 × 3]>

tree\_res %>%  
 collect\_metrics() %>%  
 ggplot(aes(cost\_complexity, mean)) +  
 geom\_line(size = 1.5, alpha = 0.6) +  
 geom\_point(size = 2) +  
 facet\_wrap(~ .metric, scales = "free", nrow = 2)

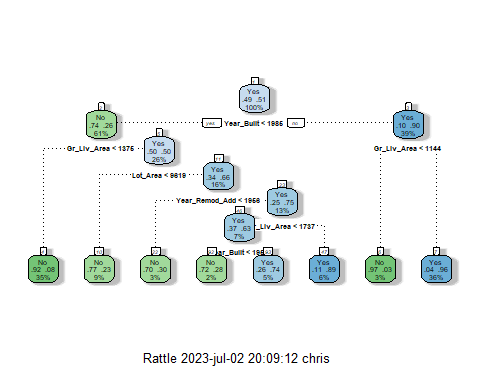


best\_tree = tree\_res %>%  
 select\_best("accuracy")  
  
best\_tree

## # A tibble: 1 × 2  
## cost\_complexity .config   
## <dbl> <chr>   
## 1 0.00750 Preprocessor1\_Model22

final\_wf =   
 tree\_wflow %>%   
 finalize\_workflow(best\_tree)

final\_fit = fit(final\_wf, train)  
  
tree = final\_fit %>%   
 pull\_workflow\_fit() %>%   
 pluck("fit")  
  
fancyRpartPlot(tree, tweak = 1.5)



The hidden label is Year\_Built < 1957. There are 7 splits.

As seen with our Lasso model, Year built and Ground living area the leading variables in our Random forests model in terms of significance, given that they are present at the first two splits.

One potential issue with classification trees, especially when there is potential for multicollinearity, is that there could be a higher chance of overfitting the model on our training set. Another method that needs to be explored is the Random Forests model, where we can build lots of trees that will in turn “vote” on the best prediction thanks to the idea of ensembling.

### Random Forests:

Our response variable is already treated as a factor, so we can skip any other preliminary work before jumping into the model. Again, this predictive model will be done on the training set.

rf\_recipe = recipe(Above\_Median ~., train) %>%  
 step\_dummy(all\_nominal(), -all\_outcomes())%>%  
 step\_zv(all\_predictors(), -all\_nominal())  
  
rf\_model = rand\_forest() %>%   
 set\_engine("ranger",importance = "permutation") %>%   
 set\_mode("classification")  
  
rf\_wflow =   
 workflow() %>%   
 add\_model(rf\_model) %>%   
 add\_recipe(rf\_recipe)  
  
### Ensuring same randomness when repeated model by setting the seed below:  
  
set.seed(1234)  
rf\_fit = fit(rf\_wflow, train)

Random forest details:

rf\_fit

## ══ Workflow [trained] ══════════════════════════════════════════════════════════  
## Preprocessor: Recipe  
## Model: rand\_forest()  
##   
## ── Preprocessor ────────────────────────────────────────────────────────────────  
## 2 Recipe Steps  
##   
## • step\_dummy()  
## • step\_zv()  
##   
## ── Model ───────────────────────────────────────────────────────────────────────  
## Ranger result  
##   
## Call:  
## ranger::ranger(x = maybe\_data\_frame(x), y = y, importance = ~"permutation", num.threads = 1, verbose = FALSE, seed = sample.int(10^5, 1), probability = TRUE)   
##   
## Type: Probability estimation   
## Number of trees: 500   
## Sample size: 1437   
## Number of independent variables: 52   
## Mtry: 7   
## Target node size: 10   
## Variable importance mode: permutation   
## Splitrule: gini   
## OOB prediction error (Brier s.): 0.06903366

The predictions:

predRF = predict(rf\_fit, train)  
head(predRF)

## # A tibble: 6 × 1  
## .pred\_class  
## <fct>   
## 1 No   
## 2 No   
## 3 No   
## 4 No   
## 5 No   
## 6 No

The confusion matrix:

confusionMatrix(predRF$.pred\_class, train$Above\_Median, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 693 25  
## Yes 14 705  
##   
## Accuracy : 0.9729   
## 95% CI : (0.9631, 0.9806)  
## No Information Rate : 0.508   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.9457   
##   
## Mcnemar's Test P-Value : 0.1093   
##   
## Sensitivity : 0.9658   
## Specificity : 0.9802   
## Pos Pred Value : 0.9805   
## Neg Pred Value : 0.9652   
## Prevalence : 0.5080   
## Detection Rate : 0.4906   
## Detection Prevalence : 0.5003   
## Balanced Accuracy : 0.9730   
##   
## 'Positive' Class : Yes   
##

### Test the RF Model on the Test set and compare:

Test Predictions

testpredrf = predict(rf\_fit, test)  
head(testpredrf)

## # A tibble: 6 × 1  
## .pred\_class  
## <fct>   
## 1 Yes   
## 2 No   
## 3 Yes   
## 4 Yes   
## 5 Yes   
## 6 Yes

confusionMatrix(testpredrf$.pred\_class, test$Above\_Median,   
 positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 277 30  
## Yes 26 283  
##   
## Accuracy : 0.9091   
## 95% CI : (0.8836, 0.9306)  
## No Information Rate : 0.5081   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.8182   
##   
## Mcnemar's Test P-Value : 0.6885   
##   
## Sensitivity : 0.9042   
## Specificity : 0.9142   
## Pos Pred Value : 0.9159   
## Neg Pred Value : 0.9023   
## Prevalence : 0.5081   
## Detection Rate : 0.4594   
## Detection Prevalence : 0.5016   
## Balanced Accuracy : 0.9092   
##   
## 'Positive' Class : Yes   
##

Save the model to a file to load later (if needed)

saveRDS(rf\_fit, "rf\_fit.rds")

Load the model

rf\_fit = readRDS("rf\_fit.rds")

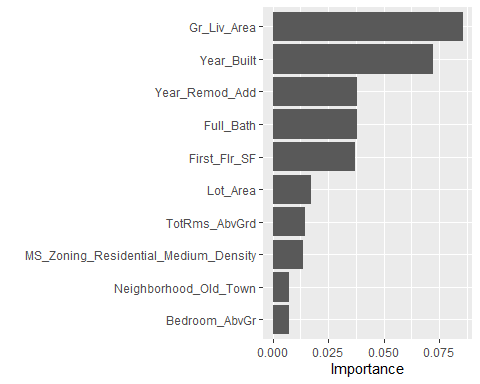
Check out variable importance

library(vip)

##   
## Attaching package: 'vip'

## The following object is masked from 'package:utils':  
##   
## vi

rf\_fit %>% pull\_workflow\_fit() %>% vip(geom = "col")



### Now Redo Everything with just the Originally selected variables from phase 2:

I selected my variables from the ames\_features dataset now since we already refacted the predictor variable and tidied it up.

ames\_features2 = ames\_features %>%  
 dplyr::select(Above\_Median, Year\_Built, Year\_Remod\_Add, Neighborhood, MS\_Zoning,Lot\_Area, Overall\_Cond, Year\_Sold,Year\_Built)

str(ames\_features2)

## tibble [2,053 × 8] (S3: tbl\_df/tbl/data.frame)  
## $ Above\_Median : Factor w/ 2 levels "No","Yes": 2 1 2 2 2 2 2 2 2 2 ...  
## $ Year\_Built : num [1:2053] 1960 1961 1958 1968 1997 ...  
## $ Year\_Remod\_Add: num [1:2053] 1960 1961 1958 1968 1998 ...  
## $ Neighborhood : Factor w/ 28 levels "North\_Ames","Gilbert",..: 1 1 1 1 2 2 3 3 2 2 ...  
## $ MS\_Zoning : Factor w/ 7 levels "Residential\_Low\_Density",..: 1 2 1 1 1 1 1 1 1 1 ...  
## $ Lot\_Area : num [1:2053] 31770 11622 14267 11160 13830 ...  
## $ Overall\_Cond : Factor w/ 9 levels "Average","Above\_Average",..: 1 2 2 1 1 2 1 1 3 1 ...  
## $ Year\_Sold : num [1:2053] 2010 2010 2010 2010 2010 2010 2010 2010 2010 2010 ...

summary(ames\_features2)

## Above\_Median Year\_Built Year\_Remod\_Add Neighborhood   
## No :1010 Min. :1875 Min. :1950 North\_Ames : 327   
## Yes:1043 1st Qu.:1953 1st Qu.:1965 College\_Creek: 183   
## Median :1972 Median :1993 Old\_Town : 181   
## Mean :1971 Mean :1984 Edwards : 129   
## 3rd Qu.:2000 3rd Qu.:2004 Somerset : 119   
## Max. :2010 Max. :2010 Gilbert : 109   
## (Other) :1005   
## MS\_Zoning Lot\_Area Overall\_Cond   
## Residential\_Low\_Density :1600 Min. : 1300 Average :1143   
## Residential\_High\_Density : 20 1st Qu.: 7500 Above\_Average: 376   
## Floating\_Village\_Residential: 87 Median : 9548 Good : 286   
## Residential\_Medium\_Density : 326 Mean : 10258 Very\_Good : 98   
## C\_all : 17 3rd Qu.: 11600 Below\_Average: 73   
## A\_agr : 2 Max. :215245 Fair : 35   
## I\_all : 1 (Other) : 42   
## Year\_Sold   
## Min. :2006   
## 1st Qu.:2007   
## Median :2008   
## Mean :2008   
## 3rd Qu.:2009   
## Max. :2010   
##

### Lasso Regression on

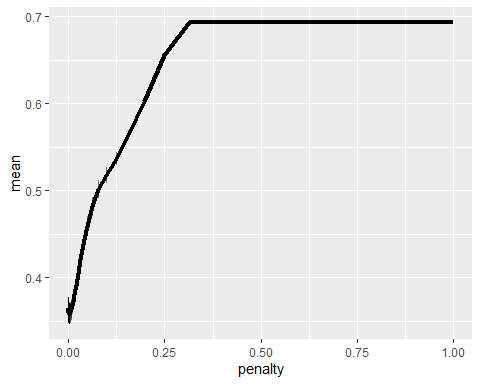
I need new training and testing sets since there are new (less) variables:

set.seed(1234)  
amesft\_split2 = initial\_split(ames\_features2,prop=.7,strata=Above\_Median)  
train2 = training(amesft\_split2)  
test2=testing(amesft\_split2)

set.seed(1234)  
folds2 = vfold\_cv(train2,v=10)

lasso\_recipe2 =   
 recipe(formula = Above\_Median ~ Year\_Built +Year\_Remod\_Add+Lot\_Area+Year\_Sold+Neighborhood, data = train2) %>%   
 step\_dummy(all\_nominal(), -all\_outcomes()) %>%   
 step\_zv(all\_predictors(), -all\_nominal()) %>%  
 step\_normalize(all\_predictors(), -all\_nominal())   
  
lasso\_spec2 =   
 logistic\_reg(penalty = tune(), mixture = 1) %>%   
 set\_mode("classification") %>%   
 set\_engine("glmnet")   
  
lasso\_workflow2 <-   
 workflow() %>%   
 add\_recipe(lasso\_recipe2) %>%   
 add\_model(lasso\_spec2)   
  
lasso\_grid2 = grid\_regular(penalty(), levels = 100)  
  
#note the use of alternative metric (min log loss)  
lasso\_tune2 <-   
 tune\_grid(lasso\_workflow2, resamples = folds2,   
 grid = lasso\_grid2, metrics = metric\_set(mn\_log\_loss))

lasso\_tune2 %>%  
 collect\_metrics() %>%  
 ggplot(aes(penalty, mean)) +  
 geom\_errorbar(aes(  
 ymin = mean - std\_err,  
 ymax = mean + std\_err  
 ),  
 alpha = 0.5  
 ) +  
 geom\_line(size = 1.5) +  
 theme(legend.position = "none")



best\_mnlog2 = lasso\_tune2 %>%  
 select\_best("mn\_log\_loss")  
best\_mnlog2

## # A tibble: 1 × 2  
## penalty .config   
## <dbl> <chr>   
## 1 0.00187 Preprocessor1\_Model073

final\_lasso2 = lasso\_workflow2 %>% finalize\_workflow(best\_mnlog2)

lasso\_fit2 = fit(final\_lasso2, train2)

options(scipen = 999)  
lasso\_fit2 %>%  
 extract\_fit\_parsnip() %>%  
 pluck("fit") %>%   
 coef(s = best\_mnlog2$penalty) #show the coefficients for our selected lambda value

## 30 x 1 sparse Matrix of class "dgCMatrix"  
## s1  
## (Intercept) 0.26885022641  
## Year\_Built 0.98758810140  
## Year\_Remod\_Add 0.65969507822  
## Lot\_Area 1.25729908475  
## Year\_Sold -0.02441888610  
## Neighborhood\_Gilbert 0.54993625055  
## Neighborhood\_Stone\_Brook 0.50348369035  
## Neighborhood\_Northwest\_Ames 0.38970971868  
## Neighborhood\_Somerset 0.41495332974  
## Neighborhood\_Briardale -0.23443818921  
## Neighborhood\_Northpark\_Villa -0.17369048668  
## Neighborhood\_Northridge\_Heights 0.80722306087  
## Neighborhood\_Bloomington\_Heights 0.16744287754  
## Neighborhood\_Northridge 0.56498705641  
## Neighborhood\_Sawyer\_West 0.10374114394  
## Neighborhood\_Sawyer -0.12837212034  
## Neighborhood\_Greens 0.27662444678  
## Neighborhood\_Old\_Town 0.07235068521  
## Neighborhood\_Brookside 0.18529420506  
## Neighborhood\_Iowa\_DOT\_and\_Rail\_Road .   
## Neighborhood\_Clear\_Creek 0.19672324324  
## Neighborhood\_South\_and\_West\_of\_Iowa\_State\_University 0.26842252994  
## Neighborhood\_Edwards -0.19266314726  
## Neighborhood\_College\_Creek 0.10708334150  
## Neighborhood\_Crawford 0.53408038976  
## Neighborhood\_Mitchell -0.00001784567  
## Neighborhood\_Timberland 0.44208072677  
## Neighborhood\_Meadow\_Village -0.25601348329  
## Neighborhood\_Veenker 0.17842950495  
## Neighborhood\_Blueste 0.02014179466

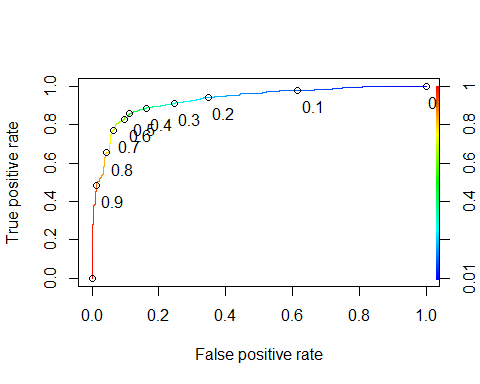
options(scipen = 0)

This time, only one neighborhood was dropped (compared to when the Lasso model was ran an the trainig set with more variables).

tidy(lasso\_fit2) [1:2]

## # A tibble: 30 × 2  
## term estimate  
## <chr> <dbl>  
## 1 (Intercept) 0.269   
## 2 Year\_Built 0.988   
## 3 Year\_Remod\_Add 0.660   
## 4 Lot\_Area 1.26   
## 5 Year\_Sold -0.0244  
## 6 Neighborhood\_Gilbert 0.550   
## 7 Neighborhood\_Stone\_Brook 0.503   
## 8 Neighborhood\_Northwest\_Ames 0.390   
## 9 Neighborhood\_Somerset 0.415   
## 10 Neighborhood\_Briardale -0.234   
## # ℹ 20 more rows

predictions2 = predict(lasso\_fit2, train2, type="prob")[2]  
  
ROCRpred2 = prediction(predictions2, train2$Above\_Median)   
  
ROCRperf2 = performance(ROCRpred2, "tpr", "fpr")  
plot(ROCRperf2, colorize=TRUE, print.cutoffs.at=seq(0,1,by=0.1), text.adj=c(-0.2,1.7))



#Determine threshold to balance sensitivity and specificity  
#DO NOT modify this code  
opt.cut = function(perf, pred){  
 cut.ind = mapply(FUN=function(x, y, p){  
 d = (x - 0)^2 + (y-1)^2  
 ind = which(d == min(d))  
 c(sensitivity = y[[ind]], specificity = 1-x[[ind]],   
 cutoff = p[[ind]])  
 }, perf@x.values, perf@y.values, pred@cutoffs)  
}  
print(opt.cut(ROCRperf2, ROCRpred2))

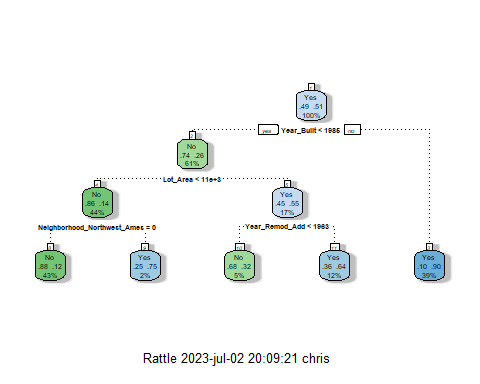
## [,1]  
## sensitivity 0.8589041  
## specificity 0.8882603  
## cutoff 0.4919238

### Classification Tree

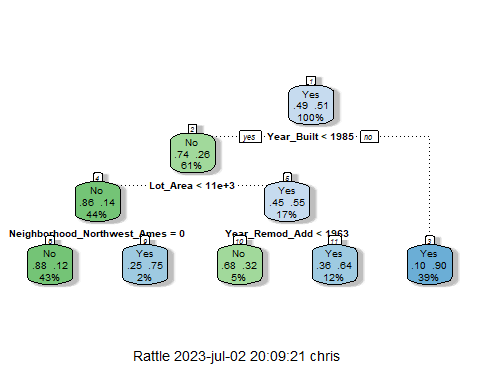
library(mice) #package for imputation  
library(VIM) #visualizing missingness  
library(rpart) #for classification trees  
library(rpart.plot) #for plotting trees  
library(RColorBrewer) #better visualization of classification trees  
library(rattle) #better visualization of classification trees  
library(caret)

#Classification Tree  
  
tree\_recipe2 = recipe(Above\_Median ~., train2) %>%  
 step\_dummy(all\_nominal(),-all\_outcomes())%>%  
 step\_zv(all\_predictors(), -all\_nominal())  
  
tree\_model2 = decision\_tree() %>%   
 set\_engine("rpart", model = TRUE) %>% #don't forget the model = TRUE flag  
 set\_mode("classification")  
  
tree\_wflow2 =   
 workflow() %>%   
 add\_model(tree\_model2) %>%   
 add\_recipe(tree\_recipe2)  
  
tree\_fit2 = fit(tree\_wflow2, train2)

tree2 = tree\_fit2 %>%   
 pull\_workflow\_fit() %>%   
 pluck("fit")  
  
fancyRpartPlot(tree2)



fancyRpartPlot(tree2, tweak=1.3) #tweak makes the tree a little easier to read



There are 5 splits here.

# Complexity Parameter and the xerror, cross validated error. best xerror is the smallest one yielded by the correspondig CP  
  
tree\_fit2$fit$fit$fit$cptable

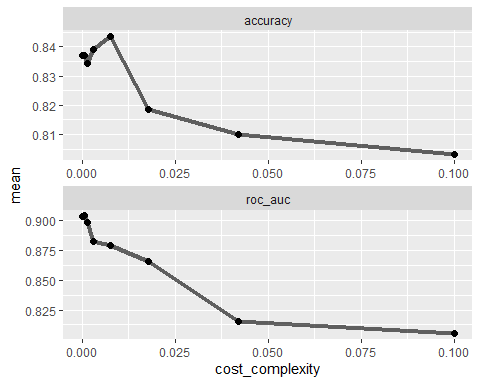
## CP nsplit rel error xerror xstd  
## 1 0.60678925 0 1.0000000 1.0381895 0.02680266  
## 2 0.03394625 1 0.3932107 0.3960396 0.02123723  
## 3 0.01697313 3 0.3253182 0.3946252 0.02120843  
## 4 0.01000000 4 0.3083451 0.3380481 0.01996547

set.seed(1234)  
folds2 = vfold\_cv(train2, v = 5)

tree\_recipe2 = recipe(Above\_Median ~., train2) %>%  
 step\_dummy(all\_nominal(),-all\_outcomes())%>%  
 step\_zv(all\_predictors(), -all\_nominal())  
  
tree\_model2 = decision\_tree(cost\_complexity = tune()) %>%   
 set\_engine("rpart", model = TRUE) %>% #don't forget the model = TRUE flag  
 set\_mode("classification")  
  
tree\_grid2 = grid\_regular(cost\_complexity(),  
 levels = 25) #try 25 sensible values for cp  
  
tree\_wflow2 =   
 workflow() %>%   
 add\_model(tree\_model2) %>%   
 add\_recipe(tree\_recipe2)  
  
tree\_res2 =   
 tree\_wflow2 %>%   
 tune\_grid(  
 resamples = folds2,  
 grid = tree\_grid2  
 )  
  
tree\_res2

## # Tuning results  
## # 5-fold cross-validation   
## # A tibble: 5 × 4  
## splits id .metrics .notes   
## <list> <chr> <list> <list>   
## 1 <split [1149/288]> Fold1 <tibble [50 × 5]> <tibble [0 × 3]>  
## 2 <split [1149/288]> Fold2 <tibble [50 × 5]> <tibble [0 × 3]>  
## 3 <split [1150/287]> Fold3 <tibble [50 × 5]> <tibble [0 × 3]>  
## 4 <split [1150/287]> Fold4 <tibble [50 × 5]> <tibble [0 × 3]>  
## 5 <split [1150/287]> Fold5 <tibble [50 × 5]> <tibble [0 × 3]>

tree\_res2 %>%  
 collect\_metrics() %>%  
 ggplot(aes(cost\_complexity, mean)) +  
 geom\_line(size = 1.5, alpha = 0.6) +  
 geom\_point(size = 2) +  
 facet\_wrap(~ .metric, scales = "free", nrow = 2)

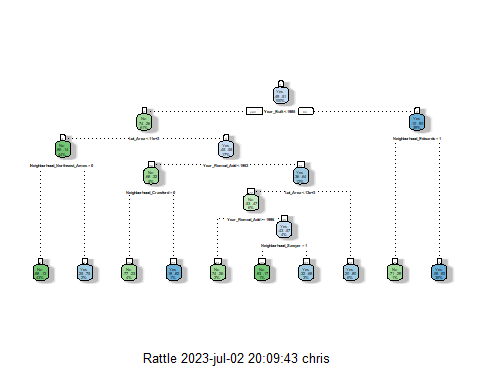


best\_tree2 = tree\_res2 %>%  
 select\_best("accuracy")  
  
best\_tree2

## # A tibble: 1 × 2  
## cost\_complexity .config   
## <dbl> <chr>   
## 1 0.00750 Preprocessor1\_Model22

final\_wf2 =   
 tree\_wflow2 %>%   
 finalize\_workflow(best\_tree2)

final\_fit2 = fit(final\_wf2, train2)  
  
tree2 = final\_fit2 %>%   
 pull\_workflow\_fit() %>%   
 pluck("fit")  
  
fancyRpartPlot(tree2, tweak = 1.5)



There are 9 splits in this dataset (with the variables chosen in phase 1). An added layer of complexity seen here is the mentioning of neighborhoods.

As seen with our Lasso model, Year built is the leading variable in this new Random forests model in terms of significance. After that comes Lot\_Area and Neighborhood\_Edwards. I’ll have to explore what’s so special about Neighborhood Edwards to detemrine why despite home being buit below 1986, the older homes in this neighborhoods are predicted to sell above the Median price.

A quick google search to realtor.com show a big disparity in prices, but there are multiple homes selling for over $1 Million dollars currently. It is located close to the University and has 3 public parks and is split by Lincoln Way (E41). This location allows for resident to easily commute to the US Route 30. It is also relatively close to the municipal airport, without being right next to it (perhaps lower noise pollution?), The hospital is also nearby. All of these additional variables are not in the dataset, but could play into why a property’s location within the Edwards neighborhood could allow for only properties to sell for so much.

As done before, I will now create a Random Forest on the new training set using the variables selected in phase 1.

### Random Forests:

Our response variable is already treated as a factor, so we can skip any other preliminary work before jumping into the model. Again, this predictive model will be done on the (second) training set.

rf\_recipe2 = recipe(Above\_Median ~., train2) %>%  
 step\_dummy(all\_nominal(), -all\_outcomes())%>%  
 step\_zv(all\_predictors(), -all\_nominal())  
  
rf\_model2 = rand\_forest() %>%   
 set\_engine("ranger",importance = "permutation") %>%   
 set\_mode("classification")  
  
rf\_wflow2 =   
 workflow() %>%   
 add\_model(rf\_model2) %>%   
 add\_recipe(rf\_recipe2)  
  
### Ensuring same randomness when repeated model by setting the seed below:  
  
set.seed(1234)  
rf\_fit2 = fit(rf\_wflow2, train2)

Random forest details:

rf\_fit2

## ══ Workflow [trained] ══════════════════════════════════════════════════════════  
## Preprocessor: Recipe  
## Model: rand\_forest()  
##   
## ── Preprocessor ────────────────────────────────────────────────────────────────  
## 2 Recipe Steps  
##   
## • step\_dummy()  
## • step\_zv()  
##   
## ── Model ───────────────────────────────────────────────────────────────────────  
## Ranger result  
##   
## Call:  
## ranger::ranger(x = maybe\_data\_frame(x), y = y, importance = ~"permutation", num.threads = 1, verbose = FALSE, seed = sample.int(10^5, 1), probability = TRUE)   
##   
## Type: Probability estimation   
## Number of trees: 500   
## Sample size: 1437   
## Number of independent variables: 43   
## Mtry: 6   
## Target node size: 10   
## Variable importance mode: permutation   
## Splitrule: gini   
## OOB prediction error (Brier s.): 0.09902135

The predictions:

predRF2 = predict(rf\_fit2, train2)  
head(predRF2)

## # A tibble: 6 × 1  
## .pred\_class  
## <fct>   
## 1 No   
## 2 No   
## 3 No   
## 4 No   
## 5 No   
## 6 No

The confusion matrix:

confusionMatrix(predRF2$.pred\_class, train2$Above\_Median, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 651 69  
## Yes 56 661  
##   
## Accuracy : 0.913   
## 95% CI : (0.8972, 0.9271)  
## No Information Rate : 0.508   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.826   
##   
## Mcnemar's Test P-Value : 0.2831   
##   
## Sensitivity : 0.9055   
## Specificity : 0.9208   
## Pos Pred Value : 0.9219   
## Neg Pred Value : 0.9042   
## Prevalence : 0.5080   
## Detection Rate : 0.4600   
## Detection Prevalence : 0.4990   
## Balanced Accuracy : 0.9131   
##   
## 'Positive' Class : Yes   
##

### Test the RF Model on the Test set and compare:

Test Predictions

testpredrf2 = predict(rf\_fit2, test2)  
head(testpredrf2)

## # A tibble: 6 × 1  
## .pred\_class  
## <fct>   
## 1 No   
## 2 No   
## 3 No   
## 4 Yes   
## 5 Yes   
## 6 Yes

confusionMatrix(testpredrf2$.pred\_class, test2$Above\_Median,   
 positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 269 41  
## Yes 34 272  
##   
## Accuracy : 0.8782   
## 95% CI : (0.8498, 0.903)  
## No Information Rate : 0.5081   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.7565   
##   
## Mcnemar's Test P-Value : 0.4884   
##   
## Sensitivity : 0.8690   
## Specificity : 0.8878   
## Pos Pred Value : 0.8889   
## Neg Pred Value : 0.8677   
## Prevalence : 0.5081   
## Detection Rate : 0.4416   
## Detection Prevalence : 0.4968   
## Balanced Accuracy : 0.8784   
##   
## 'Positive' Class : Yes   
##

Save the model to a file to load later (if needed)

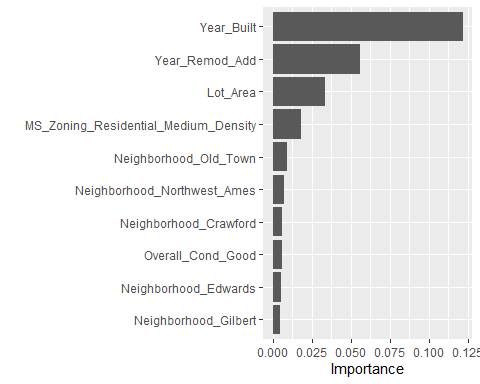
saveRDS(rf\_fit2, "rf\_fit2.rds")

Load the model

rf\_fit2 = readRDS("rf\_fit2.rds")

Check out variable importance

library(vip)  
rf\_fit2 %>% pull\_workflow\_fit() %>% vip(geom = "col")

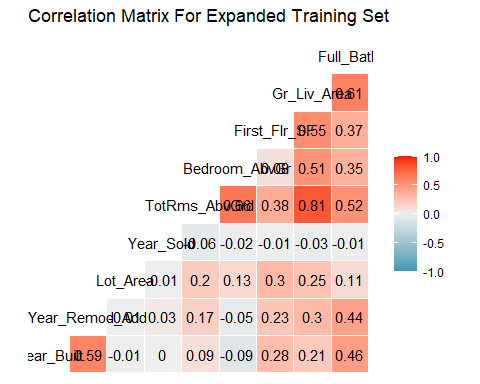


So, I initially suspected that Year built would have a major impact give the phase variable selection. Remodelation certainly play an important in conserving the value of a home,given the year it was built in.

Lot Area is pretty important in areas that aren’t densely populated (explored that metric in phase one with MS\_Zoning graph), and Ames Iowa is generally not densely populated and rarely do the properties in the dataset have a medium density score. That being said, when a property is determined to be within the meidum density zoning area, this appears to be an important factor in detemining whether the property will sell above or below the median sale price. Ames, Iowa is not a major metropolis like NYC, or San Francisco where a one bedroom apartment sometimes costs as much a multiple-bedroom house in the countryside. Neighborhood is of course split by the name of each neighborhood with Old Town being the most important according to the model.

Let’s compare multicollinearity on both training sets:

library(GGally)  
train\_ggcorr = train %>%  
 dplyr::select(Year\_Built,Year\_Remod\_Add,Lot\_Area,Year\_Sold,TotRms\_AbvGrd,Bedroom\_AbvGr,First\_Flr\_SF,Gr\_Liv\_Area,Full\_Bath)  
 ggcorr(train\_ggcorr, label = "TRUE", label\_round = 2) +  
 labs(title = "Correlation Matrix For Expanded Training Set")



library(GGally)  
train2\_ggcorr = train2 %>%  
 dplyr::select(Year\_Built,Year\_Remod\_Add,Lot\_Area,Year\_Sold)  
 ggcorr(train2\_ggcorr, label = "TRUE", label\_round = 2) +  
 labs(title = "Broader Correlation Matrix Between to Test For Multicollinearity")

