FinalExam

Rhodes Kirkpatrick

3/18/2022

```
setwd("/Users/Owner/OneDrive/Documents/UOregon/Q2/EconometricsII")
options(stringsAsFactors = F)
library(pacman)
p_load(readr, dplyr, kernlab,tidyverse, tidymodels,patchwork, data.table, parallel, magrittr, parsnip,xgboost,rpa
rt,rpart.plot, baguette,modeldata,janitor,utils,rsample)
finaldata = read.csv("final-data.csv")
finaldata=finaldata%>%mutate(`undervalued` = as.numeric(`undervalued`))
finaldata_df = data.frame(ID = (finaldata$id),
                          undervalued = as.factor(finaldata$undervalued),
                          frontyard= as.numeric(finaldata$lot_frontage), lottotal=as.numeric(finaldata$lot_area),
quality=as.numeric(finaldata$overall_qual), condition=as.numeric(finaldata$overall_cond),
built=finaldata$year_built, remod=finaldata$year_remod_add, basementsf=as.numeric(finaldata$total_bsmt_sf), fullb
ath = as.numeric(finaldata$full_bath),
halfbath=as.numeric(finaldata$half_bath), garagecars=as.numeric(finaldata$garage_cars), totalrooms=as.numeric(fin
aldata$tot_rms_abv_grd), bedrooms=as.numeric(finaldata$bedroom_abv_gr))
finaldata_normalized = data.frame(ID = finaldata_df$ID, undervalued = finaldata_df$undervalued,scale(finaldata_df
[,c(3:14)]))
```

```
#Fix Data + Data split
final2 = na.omit(finaldata_normalized)
finaldatasplit = final2%>%initial_split(prop = 0.8)
finaldatasplit2 = finaldatasplit[["data"]]
```

```
#Two data sets
#Testing Set
finaltest = finaldatasplit2%>% sample_frac(size=.2)
#Training Set
finaltrain = setdiff(finaldatasplit2,finaltest)
```

```
#Split Data and cross fold
#Set seed for CV k-fold
set.seed(7586)
#Cross Validation
finaldata_cv = finaltrain %>%
  vfold_cv(v=5)
```

```
#Full list
alldata = rbindlist(
  list(finaltrain, finaltest),
  use.names = T, fill = T
)
# Need this to split data
i_train = 1:nrow(finaltrain)
i_test = (nrow(finaltrain)+1):nrow(alldata)
#Need split to collect predictions in last chunk
Listsplit = make_splits(list(analysis = i_train, assessment = i_test),
  data= alldata
)
```

```
#Universal Recipe, ID role
LuckyCharmz = recipe(`undervalued`~.,data=finaltrain)%>% update_role(ID, new_role = "ID")
```

1. Being able to predict when your model under predicts a value of a home implicitly provides a more truthful model. The issue is that in practice it doesn't point to a number, however, it would allow my REO to create arbitrage opportunities off yours if you were selling or bidding as I now hold a higher willingness to pay off an information asymmetry.
2. ◦ 3. Two Models: Random Forest and SVM models with performance summary

```
# Grid parameter
rf_grid = expand_grid(
  min_n = c(10, 100),
  mtry = c(15, 30, 45),
  trees = c(50,100,150,200)
)
```

```r
#Define Function
rf_i = function(i) {
  # The RF model for iteration i
  finalrf_i =
    rand_forest(
      trees = rf_grid$trees[i],
      min_n = rf_grid$min_n[i],
      mtry = rf_grid$mtry[i],
    ) %>%
    set_mode("classification") %>%
    set_engine(
      "ranger",
      splitrule = "gini"
    )
  # Set up workflow for each i to use iterations
  wf_i = workflow() %>% add_model(finalrf_i) %>% add_recipe(LuckyCharmz)
  # Fit it!
  fit_i = wf_i %>% fit(finaltrain)
  # Use Ed's way of getting error back
  tibble(
    mtry = rf_grid$mtry[i],
    min_n = rf_grid$min_n[i],
    trees = rf_grid$trees[i],
    # Note: OOB error is buried
    error_oob = fit_i$fit$fit$fit$prediction.error
  )
}
```

```r
# Apply grid to function, use mc.cores=1
rf_tuning = mclapply(
  X = 1:nrow(rf_grid),
  FUN = rf_i,
  mc.cores = min(1, nrow(rf_grid))
)
```

```
## Warning: 15 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 15 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 15 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 15 columns were requested but there were 12 predictors in the data. 12
## will be used.
```

```
## Warning: 30 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 30 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 30 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 30 columns were requested but there were 12 predictors in the data. 12
## will be used.
```

```
## Warning: 45 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 45 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 45 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 45 columns were requested but there were 12 predictors in the data. 12
## will be used.
## Warning: 15 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 15 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 15 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 15 columns were requested but there were 12 predictors in the data. 12
## will be used.
```

```
## Warning: 30 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 30 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 30 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 30 columns were requested but there were 12 predictors in the data. 12
## will be used.
```

```
## Warning: 45 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 45 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 45 columns were requested but there were 12 predictors in the data. 12
## will be used.

## Warning: 45 columns were requested but there were 12 predictors in the data. 12
## will be used.
```

```
rf_tuning
```

```
## [[1]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    15    10    50     0.232
##
## [[2]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    15    10   100     0.227
##
## [[3]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    15    10   150     0.230
##
## [[4]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    15    10   200     0.229
##
## [[5]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    30    10    50     0.232
##
## [[6]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    30    10   100     0.230
##
## [[7]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    30    10   150     0.228
##
## [[8]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    30    10   200     0.230
##
## [[9]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    45    10    50     0.233
##
## [[10]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    45    10   100     0.232
##
## [[11]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    45    10   150     0.228
##
## [[12]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    45    10   200     0.229
##
## [[13]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    15   100    50     0.229
##
## [[14]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    15   100   100     0.224
##
## [[15]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    15   100   150     0.226
##
## [[16]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    15   100   200     0.226
##
## [[17]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    30   100    50     0.227
##
## [[18]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    30   100   100     0.226
##
## [[19]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    30   100   150     0.226
##
## [[20]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    30   100   200     0.226
##
## [[21]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    45   100    50     0.226
##
## [[22]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    45   100   100     0.225
##
## [[23]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    45   100   150     0.225
##
## [[24]]
## # A tibble: 1 x 4
##    mtry min_n trees error_oob
##   <dbl> <dbl> <dbl>     <dbl>
## 1    45   100   200     0.224
```

The best fit has a mtry=45 and min_n=100 and trees=150 (rf_tuning[[23]])

```
# Define our best model
rf_best = rand_forest(
  trees = tune(),
  min_n = tune(),
  mtry = tune()
) %>% set_mode("classification") %>%
set_engine("ranger", splitrule = "gini")
# Finalize workflow with the results from our OOB table (use [[23]])
wf_rf_best =
  workflow() %>%
  add_recipe(LuckyCharmz) %>%
  add_model(rf_best) %>%
  finalize_workflow(parameters = rf_tuning[[23]])
```

```
#Collect Accuracy of Predictions
veryfinalrf_cv = wf_rf_best %>% last_fit(Listsplit)
```

```
## Warning: package 'ranger' was built under R version 4.1.2
```

```
## ! train/test split: preprocessor 1/1, model 1/1: 45 columns were requested but there were 12...
```

```
veryfinalrf_cv%>% collect_metrics(sens(assessment),accuracy(assessment),specificity(assessment()))
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.629 Preprocessor1_Model1
## 2 roc_auc  binary         0.701 Preprocessor1_Model1
```

3. My prediction accuracy was decently high. I got an accuracy in the low to mid 60% range and an roc/auc area of .7. This is acceptable as I beat the null classifier of the random .5 line. 70% represents the amount of random positives it could separate from a random negative. Therefore, more often than not, we are able to gain on arbitrage from buying an undervalued home.

Part 2: SVM

```
#Create linear SVM with Radial Kernels
final_radial = svm_rbf(mode = "classification",
  cost = tune(),
  rbf_sigma = tune()
) %>% set_engine("kernlab")
# The linear-SVM workflow
radialwf_svm = workflow() %>%
  add_model(final_radial) %>%
  add_recipe(LuckyCharmz)
```

```
#Create Workflow
svmflow = workflow() %>%
  add_model(final_radial) %>% add_recipe(LuckyCharmz)
```

```
# Tune the linear SVM
tune_svm = svmflow%>%tune_grid(
  finaldata_cv,
  grid = expand_grid(
    cost = 10^seq(-4, 2, length = 10),
    rbf_sigma = 10^c(-3:0)
  ),
  metrics = metric_set(accuracy, sens,spec,precision,roc_auc)

)
```

The "best" model was chosen based off taking the best 4 models in accuracy, then the best 2 who had similar precision because I really care about not having false positives and not loosing my boss money, then I chose the one with the highest AUC.

```
#Model with optimal values
best_radial = svm_rbf(mode = "classification",
  cost = 4.64158883,
  rbf_sigma = 0.10
) %>% set_engine("kernlab")
```

```
#Workflow with final model
lastsvmflow = workflow() %>%
  add_model(best_radial) %>% add_recipe(LuckyCharmz)
```

```
#Create Final Fit
veryfinalsvm = lastsvmflow %>% last_fit(Listsplit)
veryfinalsvm%>% collect_metrics(assessment)
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.667 Preprocessor1_Model1
## 2 roc_auc  binary         0.717 Preprocessor1_Model1
```

3. This model fares decently well compared to the null. The accuracy of 67.5% and AUC of .735 put it significantly past the null. The precision was around 66%, so for every 2 houses my firm can make money on, another 1 will loose money, still most likely profitable if the variation in prices stays tight enough (an expensive wrong guess could make it unprofitable).

4: False Positives are more detrimental to the company. False negatives are houses that could have been bought undervalued to make more profit but weren't. This leads to undue opportunity cost and loss of economic profit, however false positives lead to negative profit as our willingness to pay increases off an information asymmetry that ends up being wrong. False negatives are opportunity cost, false positives are negative accounting profits.

5: Linear regression is not usable for binary classification because of the nature of continuous vs discrete variables. Here, the variance is a function of the mean and not constant over new mean values, thus violating an OLS assumption. Additionally, there is not any useful interpretability as that kind of model cannot describe an absolute state that the outcome variable takes.