Random Forests Prediction

What Variables Make Sense?

Bruce Liu

- UW ID: 20843361
- Kaggle public score: 0.20230
- Kaggle submission count/times 30

Summary

Preprocessing

Transformation (if any)

- price: Applied a power transform of 1/15 on the response variate price.
- saledate: Converted it to a numeric value of days since January 1st, 1970 (the beginning of Unix time)
- roof, extwall, intwall, nbhd: Applied factor() for better use in the model.
- cndtn, ac, style, grade, ward, heat, quadrant: Applied factor() and ordered by increase in price

New Variables

- proximity: This variable is defined as the distance from the most expensive property in the dataset. The measurement is in degrees.
- RpS: This variable is defined as the number of rooms per story of a building.
- yrDiff: This variable is defined as the difference in the years from the remodelling year and the year in which the oldest part of the building is built. (i.e. `ayb - yr_rmdl`)
- styleStories: This variable is defined as the number of stories multiplied by a numeric conversion of the style of building.

Missing data handling

- heat: replace the missing values in heat in dtrain with "No Data".
- stories, kitchens: replace the missing values in stories and kitchens in dtrain by imputing the k nearest neighbours to each missing value.
- quadrant: replace the missing values in quadrant by imputing the k nearest neighbours to each missing value.

Model Building/Tuning

A trial-and-error approach was used with `ranger`. The main model metric used was the out-of-bag performance of `ranger` through the `prediction.error` value. Adding, removing, and applying different transformations to the model and using `prediction.error` where lower is better to determine the best model to run.

Parameters tuned and their optimal values:

- `mtry`: 14
- `max.depth`: 44
- `min.bucket`: 1
- `min.node.size`: 3

- `respect.unordered.factors`: TRUE

1.Preprocessing

1.1 Loading data

```r
load("RF.Rdata")
```

For random forests, categorical variables do not need to be converted to numeric variables like they needed to be for smoothing models. So, for much of the preprocessing, we simply factor the categorical variables and use them as is. For convenience, we use the preprocessing from the smoothing project as our basis for this problem.

```r
  # Preprocessing the dtrain data

  maxpricelat <- dtrain$latitude[which.max(dtrain$price)]
  maxpricelon <- dtrain$longitude[which.max(dtrain$price)]
  proximity <- sqrt((dtrain$latitude - maxpricelat)^2 + (dtrain$longitude - maxpricelon)^2)
  dtrain$proximity <- proximity
  dtrain$grade <- factor(dtrain$grade, levels = c("Low Quality",
                                                  "Fair Quality", "Average",
                                                  "Above Average", "Good Quality",
                                                  "Very Good", "Excellent",
                                                  "Superior","Exceptional-A",
                                                  "Exceptional-B", "Exceptional-C",
                                                  "Exceptional-D"))
  dtrain$saledate <- as.numeric(as.Date(dtrain$saledate))
  dtrain$cndtn <- factor(dtrain$cndtn, levels = c("Poor","Fair","Average",
                                                  "Good", "Very Good", "Excellent"))
  dtrain$ac <- factor(dtrain$ac, levels = c("N", "Y"))
  dtrain$ward <- factor(dtrain$ward, levels = c("Ward 7", "Ward 8", "Ward 5",
                                                "Ward 6", "Ward 4", "Ward 1",
                                                "Ward 3", "Ward 2"))
  dtrain$heat <- factor(dtrain$heat, levels = c("Wall Furnace", "Air-Oil", "Elec Base Brd",
                                                "Gravity Furnace", "No Data","Evp Cool",
                                                "Hot Water Rad", "Forced Air", "Water Base Brd",
                                                "Ht Pump","Warm Cool","Air Exchng"))
  dtrain$nbhd <- factor(dtrain$nbhd)
  dtrain$roof <- factor(dtrain$roof)
  dtrain$extwall <- factor(dtrain$extwall)
  dtrain$intwall <- factor(dtrain$intwall)
  dtrain$style <- factor(dtrain$style, levels = c("Split Foyer","1 Story",
                                                  "1.5 Story Fin", "1.5 Story Unfin",
                                                  "2.5 Story Unfin", "2 Story",
                                                  "Bi-Level","Split Level", "Default",
                                                  "2.5 Story Fin", "3 Story", "4 Story"))

  # Preprocessing the dtest data

  proximity <- sqrt((dtest$latitude - maxpricelat)^2 + (dtest$longitude - maxpricelon)^2)
  dtest$proximity <- proximity
  dtest$grade <- factor(dtest$grade, levels = c("Low Quality",
                                                "Fair Quality", "Average",
                                                "Above Average", "Good Quality",
                                                 "Very Good", "Excellent",
                                                 "Superior","Exceptional-A",
                                                 "Exceptional-B", "Exceptional-C",
                                                 "Exceptional-D"))
dtest$saledate <- as.numeric(as.Date(dtest$saledate))
dtest$ward <- factor(dtest$ward, levels = c("Ward 7", "Ward 8", "Ward 5",
                                           "Ward 6", "Ward 4", "Ward 1",
                                           "Ward 3", "Ward 2"))
dtest$cndtn <- factor(dtest$cndtn, levels = c("Poor","Fair","Average",
                                             "Good", "Very Good", "Excellent"))
dtest$nbhd <- factor(dtest$nbhd)
dtest$ac <- factor(dtest$ac, levels = c("N", "Y"))
dtest$roof <- factor(dtest$roof)
dtest$extwall <- factor(dtest$extwall)
dtest$intwall <- factor(dtest$intwall)
dtest$heat <- factor(dtest$heat, levels = c("Wall Furnace", "Hot Water Rad",
                                           "Forced Air", "Water Base Brd",
                                           "Ht Pump","Warm Cool"))
dtest$style <- factor(dtest$style, levels = c("Split Foyer","1 Story",
                                             "1.5 Story Fin", "1.5 Story Unfin",
                                             "2.5 Story Unfin", "2 Story",
                                             "Bi-Level","Split Level",
                                             "2.5 Story Fin", "3 Story"))
```

1.2 Missing data handling

Unless there is an option within the variable to handling missing values, we employ the `impute.knn` function for the handling of missing data. For this dataset, we see that the four variables with missing values are: `heat`, `stories`, `kitchens`, and `quadrant`. Of the four variables, `heat` has a factor value of “No Data”. So, we assign all missing values of `heat` to “No Data”. For the other three variables, we impute based on the k-nearest neighbours algorithm employed by `impute.knn`. To handle `quadrant` being non-numeric, we convert it to a numeric equivalent (i.e. NE is assigned one due to alphabetical order of `factor` etc.) and reconvert the numeric value back to its equivalent quadrant.

```
library(impute)
# Handling of missing values in dtrain

for (i in 1:6000) {
  if (is.na(dtrain[i,"heat"])) {
    dtrain[i, "heat"] <- "No Data"
  }
}
dtrainYears <- dtrain[,c(7:9)]
knnYears <- impute::impute.knn(t(dtrainYears))
imputedYears <- t(knnYears$data)
dtrain$ayb[which(is.na(dtrain$ayb))] <- imputedYears[which(is.na(dtrain$ayb))]
dtrain$yr_rmdl[which(is.na(dtrain$yr_rmdl))] <- imputedYears[6000 + which(is.na(dtrain$yr_rmdl))]
dtrainRooms <- dtrain[, c("bathrm","hf_bathrm","rooms","bedrm",
                          "stories","kitchens","fireplaces")]
knnRooms <- impute::impute.knn(t(dtrainRooms))
imputedRooms <- t(knnRooms$data)
dtrain$stories[which(is.na(dtrain$stories))] <- imputedRooms[24000 + which(is.na(dtrain$stories))]
dtrain$kitchens[which(is.na(dtrain$kitchens))] <- imputedRooms[30000 + which(is.na(dtrain$kitchens))]
dtrain$RoomRatio <- as.numeric(unlist((dtrain["bathrm"]+dtrain["hf_bathrm"]+
```

```r
                                          dtrain["bedrm"]+dtrain["kitchens"]+
                                          dtrain["fireplaces"])/(dtrain["rooms"] + 1)))
dtrain$stories[which(is.na(dtrain$stories))] <- imputedRooms[24000 + which(is.na(dtrain$stories))]
dtrain$kitchens[which(is.na(dtrain$kitchens))] <- imputedRooms[30000 + which(is.na(dtrain$kitchens))]
dtrainLocation <- dtrain[, c(23:27)]
dtrainLocation$nbhd <- as.numeric(dtrainLocation$nbhd)
dtrainLocation$ward <- as.numeric(factor(dtrainLocation$ward))
dtrainLocation$quadrant <- as.numeric(factor(dtrainLocation$quadrant))
knnLocation <- impute::impute.knn(t(dtrainLocation))
imputedLocation <- t(knnLocation$data)


for (i in c(1:6000)) {
  if (is.na(dtrain$quadrant[i])) {
    if (abs(imputedLocation[24000 + i]) <= 1) {
      dtrain$quadrant[i] <- "NE"
    } else if (abs(imputedLocation[24000 + i]) <= 2) {
      dtrain$quadrant[i] <- "NW"
    } else if (abs(imputedLocation[24000 + i]) <= 3) {
      dtrain$quadrant[i] <- "SE"
    } else {
      dtrain$quadrant[i] <- "SW"
    }
  }
}
dtrain$quadrant <- factor(dtrain$quadrant, levels = c("SW","SE","NE","NW"))
dtrain$RpS <- dtrain$rooms/dtrain$stories
dtrain$yrDiff <- dtrain$yr_rmdl - dtrain$ayb
dtrain$styleStories <- as.numeric(dtrain$style)*dtrain$stories

# Handling missing values in dtest

dtestYears <- dtest[,c(8:10)]
knnTestYears <- impute::impute.knn(t(dtestYears))
imputedTestYears <- t(knnTestYears$data)
dtest$ayb[which(is.na(dtest$ayb))] <- imputedTestYears[which(is.na(dtest$ayb))]
dtest$yr_rmdl[which(is.na(dtest$yr_rmdl))] <- imputedTestYears[998 + which(is.na(dtest$yr_rmdl))]
dtestLocation <- dtest[, c(23:27)]
dtestLocation$nbhd <- as.numeric(dtestLocation$nbhd)
dtestLocation$ward <- as.numeric(factor(dtestLocation$ward))
dtestLocation$quadrant <- as.numeric(factor(dtestLocation$quadrant))
knnTestLocation <- impute::impute.knn(t(dtestLocation))
imputedTestLocation <- t(knnTestLocation$data)

for (i in c(1:998)) {
  if (is.na(dtest$quadrant[i])) {
    if (abs(imputedTestLocation[3992 + i]) <= 1) {
      dtest$quadrant[i] <- "NE"
    } else if (abs(imputedTestLocation[3992 + i]) <= 2) {
      dtest$quadrant[i] <- "NW"
    } else if (abs(imputedTestLocation[3992 + i]) <= 3) {
      dtest$quadrant[i] <- "SE"
    } else {
      dtest$quadrant[i] <- "SW"
    }
  }
}
dtest$quadrant <- factor(dtest$quadrant, levels = c("SW","SE","NE","NW"))
dtest$RpS <- dtest$rooms/dtest$stories
dtest$yrDiff <- dtest$yr_rmdl - dtest$ayb
dtest$styleStories <- as.numeric(dtest$style)*dtest$stories
```

2. Model building

For all of the model building, we will be using the `ranger` function in the `ranger` package. The package comes in handy because `ranger` contains built-in values that allow us to check for fit and cross-validation error. Much like when we analyzed the residual fits for the response variate (i.e. price) in the smoothing project, we do the same thing for random forests. However, this time we go with a more aggressive scaling factor of 1/15. This choice was arbitrary but it was the factor that provided the best public scores for this problem.

With the scaling of the response variate under control, we move on to an intial fit of the random forest with the imputed variables. However, we do not add the new variables yet.

```
library(ranger)
dTrainInit <- dtrain[,c(1:27)]
fit <- ranger(price^(1/15) ~ ., data = dTrainInit)
```

Now that we have a fitted model, let us see the prediction error for this model.

```
fit$prediction.error
```

```
## [1] 0.001174863
```

From the documentation, we have that the prediction error is the overall out-of-bag performance for this model. Since we are dealing with regression, this metric is the MSE of the out-of-bag performance for this model. Already, we see that the MSE is relatively low as around 0.0012. To improve this, let's try to tune the parameters for this model.

The main parameters for this model to tune are `mtry`, `max.depth`, `min.bucket` and `min.node.size`. `mtry` is the number of variables to split at in each node, `max.depth` is the maximum depth of the tree, `min.bucket` is the minimum size of the leaf, and `min.node.size` is the minimum size of a node that can be split. `respect.unordered.factors` is an argument that orders unordered factors by the mean response as highlighted in the documentation for `ranger`. Since it was recommended to use it, we decided to use it. We do not need to assign a value to the maximum number of trees for `ranger` because by default, `ranger` uses 500 trees, the amount of trees that were told to keep fixed. Our tuning procedure is as follows:

1. Tune in the order of `mtry`, `min.node.size`, `min.bucket`, `max.depth`.
2. Store all OOB error values in a vector.
3. Extract the value with the lowest error.

While running this procedure, we realized that `min.bucket` remained the same as the default parameter for `ranger`. The default parameters was `min.bucket` $ = 1$. This is the default parameter for regression, which is what we are doing.

Let's run an example.

```
fitVal <- c()
  for (k in c(1:10)) {
    for (j in c(1:10)) {
      for (i in c(1:10)) {
        fit <- ranger(price^(1/15) ~ ., data = dTrainInit, num.trees = 500,
                          mtry = k, min.node.size = j, min.bucket = 1, max.depth = i,
                          respect.unordered.factors = TRUE)
        fitVal <- c(fitVal, fit$prediction.error)
      }
    }
  }

which(fitVal == min(fitVal))
```

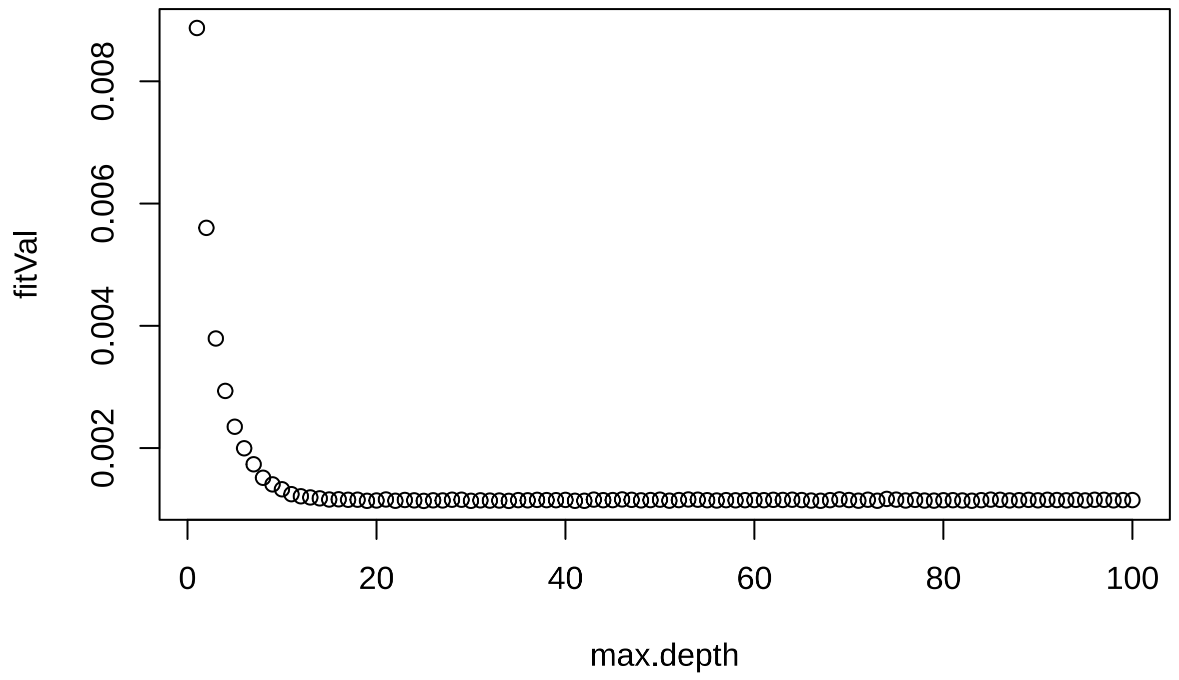
```
## [1] 970
```

```
min(fitVal)
```

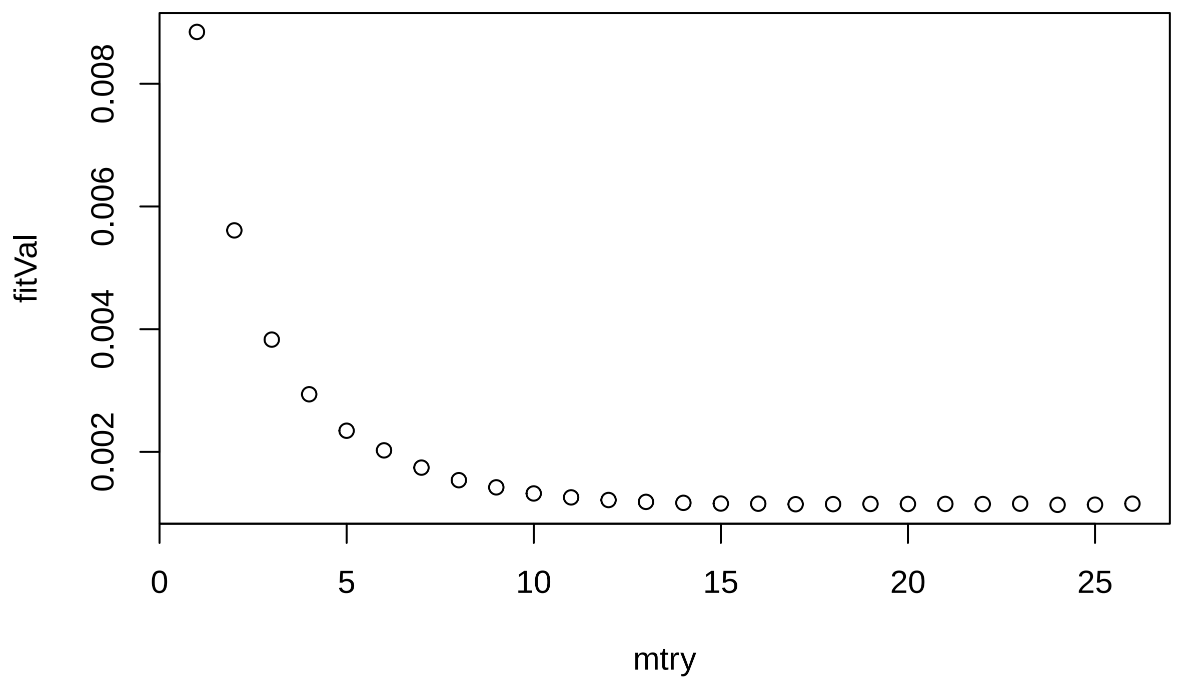
```
## [1] 0.001092461
```

So, for this example, when we unwrap the for loops, we see that the best parameters for this instance is `mtry` $= 10$, `min.node.size` $= 7$, and `max.depth` $= 10$. Now, at this point, you should notice that there is an elephant in the room. Random forests are random. How come you did not set the seed for your problem? We just realized it as of writing. We forgot. Hopefully, the results are good enough that not setting the seed is not too much of an issue for random forests. We see that the minimum OOB error is slightly smaller at around 0.00109.

We recall that the default values for `ranger` are `mtry` $= \sqrt{x}$ rounded down, `min.bucket` $= 1$ for regression, `min.node.size` $= 3$ for regression, and `max.depth` $=$ NULL which implies unlimited depth. Let us take a look at `max.depth` and see the shape of the error as we increase the maximum depth of a tree.

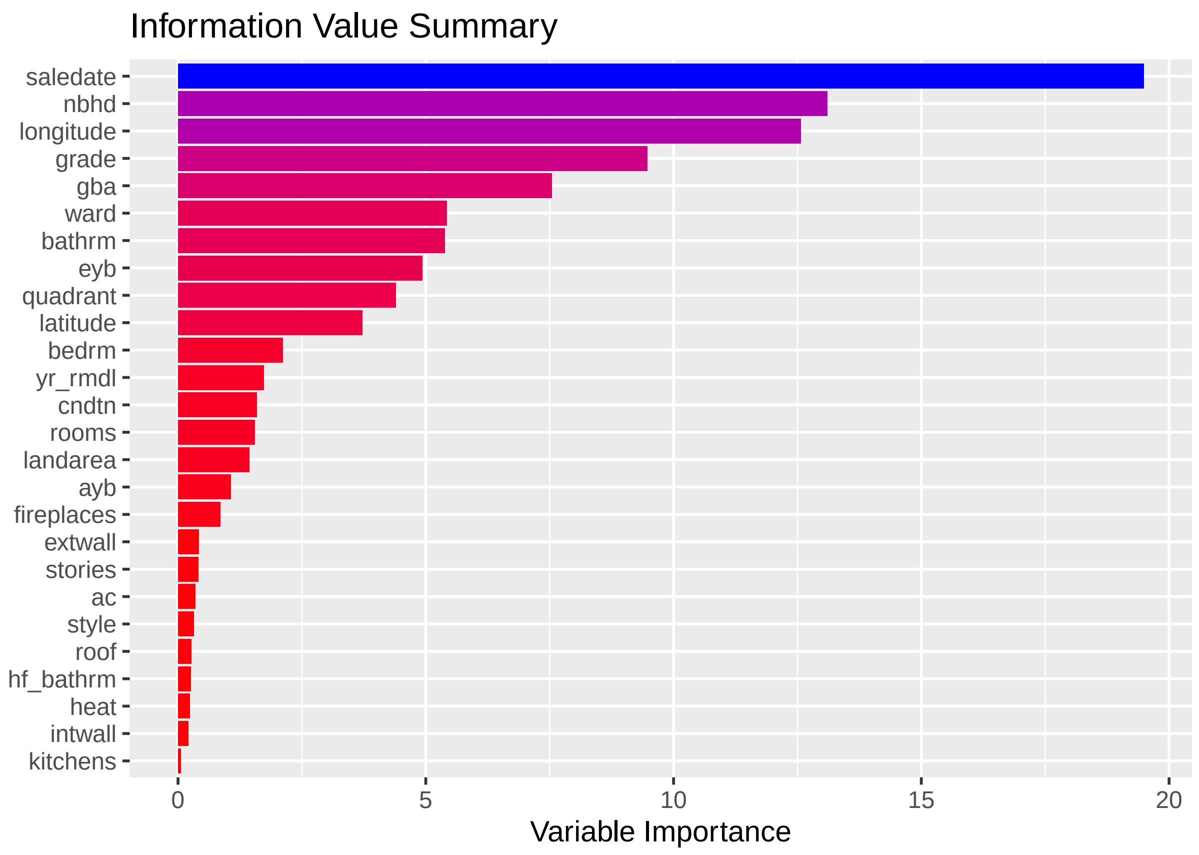


As we can see from the plot, at a certain point, there is no real benefit for having a deeper tree as the OOB error does not significantly change. Similarly, let us see the shape of the error as we increase the number of variables to split at each node. Since we have one response variate, we have 26 explanatory variates.



Much like `max.depth`, as a certain point, adding more variable does not significantly improve the OOB error of the random forest.

One last thing before we consider adding variables, `ranger` has an argument called `importance` that allows us to see which variables are significant for the random forest. For a regression based random forest, the metric is the variance of the responses. Let us see which variables are most important for this dataset.



As we can see from the plot, the sale date and the neighbourhood that the building is located in are the two most important variables for this data. As well, we see that the number of kitchens and the type of interior wall a building has are the least important variables for this data. When we decide to create new variables, we hope that they are variables that will be significant enough for the data.

In the end, we settled on following variables: `proximity`, `yrDiff`, `RpS`, and `styleStories`. We should mention that we originally had `RoomRatio` as a new variable. However, once again, we forgot to set that as a variable in the `dtest` dataset. From the summary, `proximity` and `yrDiff` are fairly self-explanatory. The other three new variables will require a brief explanation. `RpS` is a new variable under the assumption that the more rooms per story that a building has, the more likely it is that the building will be more expensive. `RoomRatio` on the other hand is the ratio of listed rooms from the other variables like `bathrm`, `kitchens`, and `bedrm` to the total number of rooms listed in `rooms` plus one. We had to add one due to one of the properties having zero rooms listed in `rooms`. Finally, `styleStories` is a new variable that combines the style of the building to the number of stories in that building. Why make a confusing variable? It is because style and stories individually are not important. We hoped that combined, this new variable could be more important than the variables individually.

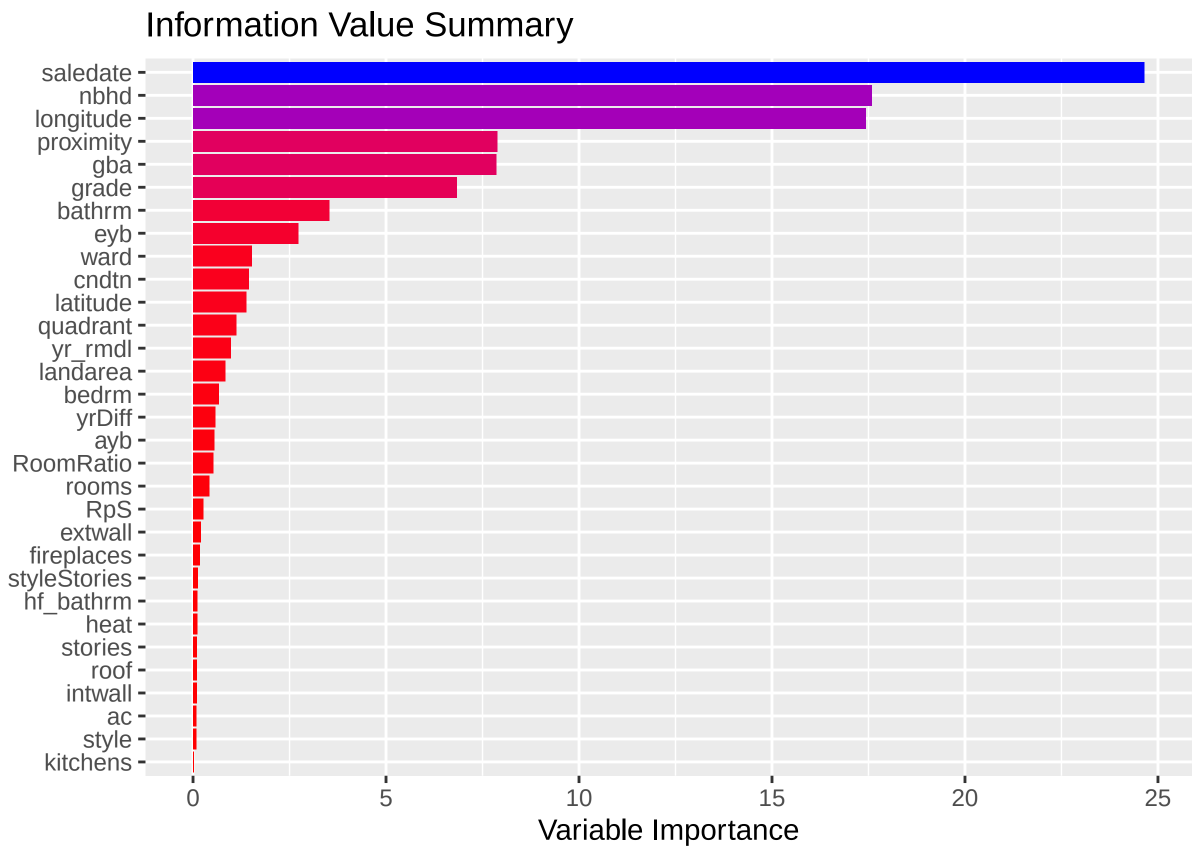
So, with these new variables in hand, we run our tuning procedure again to arrive at the following model.

```
fit <- ranger::ranger(price^(1/15) ~ ., data = dtrain, mtry = 14,
                      respect.unordered.factors = TRUE, min.bucket = 1, min.node.size = 3,
                      max.depth = 44)
```

```
fit$prediction.error
```

```
## [1] 0.001022319
```

Taking a look at the OOB error, we see that it has improved a lot from our original OOB error is 0.001174863. It is quite impressive seeing what amounts to a 10% decrease in the OOB error for this model. One final thing to look at is the variable importance of our new variables.



We see here that proximity is a very important variable for the data. we also see that the `styleStories` and `RpS` variables ended up being not important to the data. Such is the case when deriving variables. You win some, you lose some.

Concluding Thoughts

Well, how did it do? I think the model did alright. For a model like this, there is not much to be able to tune. As we saw from some cursory analysis, there is only so much you can tune before you chase the smallest of improvements. It does seem that to be able to get low public scores for this model, you would need to derive significant variables. One of the variables I derived made sense and performed well. The others? Not so much. Perhaps I could have found other variables that would have done better. Perhaps a one-hot encoding approach would have improved the model. Regardless, I think the model did alright and as always, I think there is room for improvement for this model.