House price prediction algorithm

Alix Benoit

6/16/2020

Introduction

This project uses the Ames Housing Dataset. This dataset includes information about 2930 house sales from 2006 to 2010 in Ames, Iowa, and contains 79 different explanatory variables. It was put together by Dean De Cock from Truman State University; more information about it can be found *here*.

I acquired this dataset from an ongoing "getting started" kaggle machine learning competion: *House Prices: Advanced Regression Techniques*

For the purposes of writing this report and calculating my final RMSE, I also downloaded the actual house prices for all entries from the original dataset (Note that this would not be possible if this were not a "getting started competition).

The following libraries were used:
- Tidyverse
- Caret
- Kknn
- Rborist

Of the 79 explanatory variables, 43 were identified as categorical and 36 numeric.

The goal of this project is to accurately be able to predict house prices in Ames, Iowa based on a variety of different factors. It is also to learn more about feature engineering/machine learning and to apply skills gained in Harvardx's proffessional certificate program in data science.

The root mean squared error of the log predictions was chosen as the final measure of accuracy of the final model; the log is taken so that errors on expensive houses will be weighted the same as errors on cheap houses.

$$RMSE_{log} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\ln \hat{Y}_i - \ln Y_i)^2}$$

I also use the regular RMSE to estimate the accuracy of the models when training them, since the Caret package only gives the regular RMSE, and I did not want to overtrain my models.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2}$$

```
# Function to calculate log RMSE:
RMSE_log <- function(actual_price, predicted_price){
  sqrt(mean((log(actual_price) - log(predicted_price))^2))
}
# Function to calculate regular RMSE:
RMSE <- function(actual_price, predicted_price){
  sqrt(mean(actual_price - predicted_price)^2)
}
```

In short, the final model was obtained by:
- Combining explanatory variables from the given train and test set in order to impute NAs
- Splitting full set of explanatory variables back into a train and test set
- Obtaining the best measure of center for ratings in the train set.
- Fitting a weighted knn model to the numerical variables of the train set, predicting centered rating
- Fitting a random forest model to the categorical variables of the train set, predicting centered rating, minus predictions from the knn model.

Method/Analysis

Data wrangling and cleaning

I acquired the dataset from a competition, therefore it was already split 50/50 into a train set and test set. The train set contains a "Sale Price" column, while the test set does not. Since I need to show my final RMSE in this report I also downloaded the actual test set prices from the public dataset, but I only used it to calculate the RMSE of my final model.

In order to impute NAs in the same way for both sets (keep factor levels the same, make sure the same columns are imputed, etc...), I temporarily removed the Sale Price column from the train set (leaving only explanatory variables), and bound* rows from the test set to the train set, to obtain the "full set".

```
full_set <- train %>% select(-SalePrice) %>% rbind(test)
y <- train$SalePrice
```

*(Since I didn't touch the Sale Price variable, this should not lead to overtraining, I am only removing NAs, and will use only info from the train set if I need any specific info from variables).

I then split up explanatory variables based on whether they were categorical or numeric:

```
numeric_cols <- full_set %>% select_if(is.numeric) %>% select(-Id)
categorical_cols <- full_set %>% select_if(function(x) ! is.numeric(x))

cat_col_names <- colnames(categorical_cols)
num_col_names <- colnames(numeric_cols)
```

Imputing Categorical variables:

In order to impute categorical variables, I chose to turn all NAs into the character "NA", and then turn the variable into a factor vector again. I did not want to replace NAs with an existing level, since the fact that an NA is present could itself be informative.

```
categorical_cols <- apply(categorical_cols, 2, function(x) {
  ifelse(is.na(x), "NA", x)}) %>% as.data.frame(.) %>%
  mutate_all(factor)

# Selecting numerical cols, and binding new categorical_cols back on
full_set <- full_set %>% select(-all_of(cat_col_names)) %>% cbind(categorical_cols)
```

Imputing Numeric variables:

First I found which variables contained NAs:

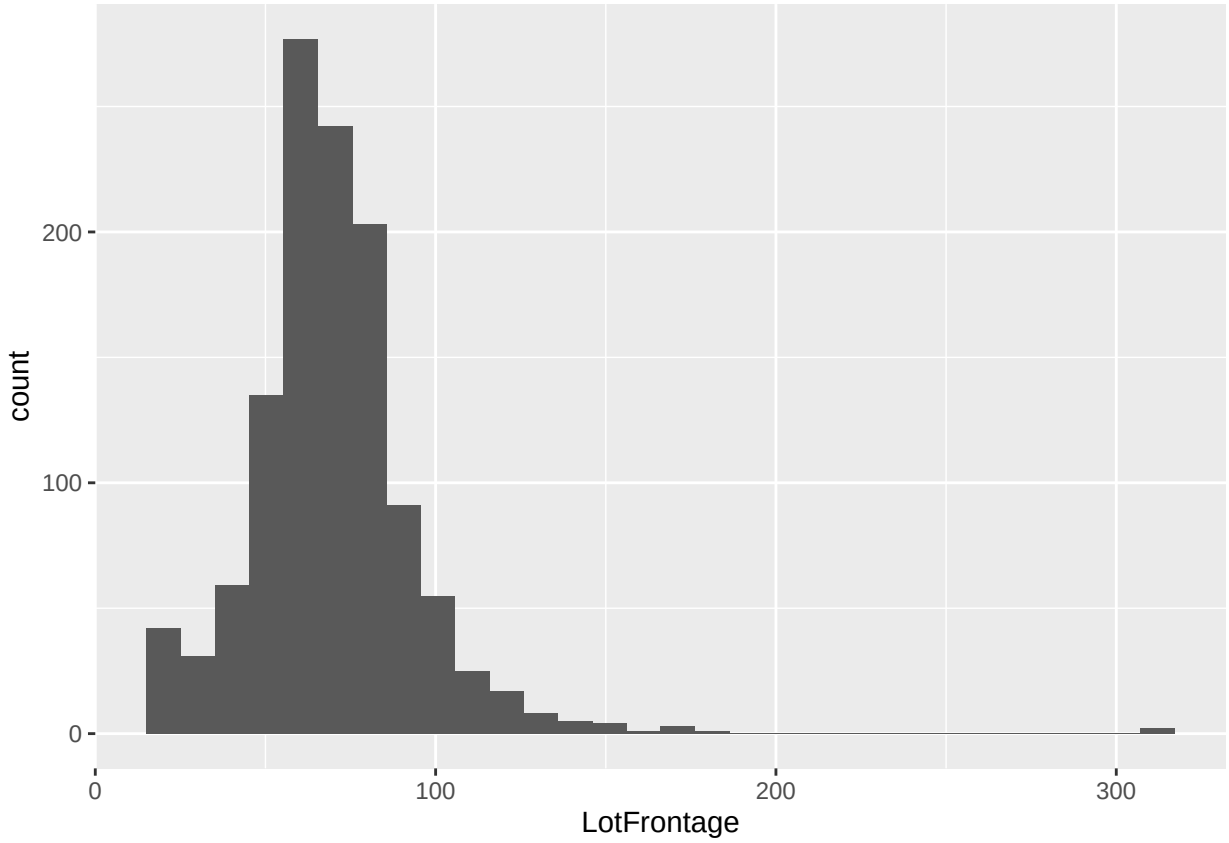
```
NA_cols_numeric <-apply(numeric_cols, 2, function(x) any(is.na(x)))
which(NA_cols_numeric == T)
```

```
##  LotFrontage   MasVnrArea   BsmtFinSF1   BsmtFinSF2    BsmtUnfSF  TotalBsmtSF
##            2            8            9           10           11           12
## BsmtFullBath BsmtHalfBath  GarageYrBlt   GarageCars   GarageArea
##           17           18           25           26           27
```

I then looked at the distributions of some of these variables:

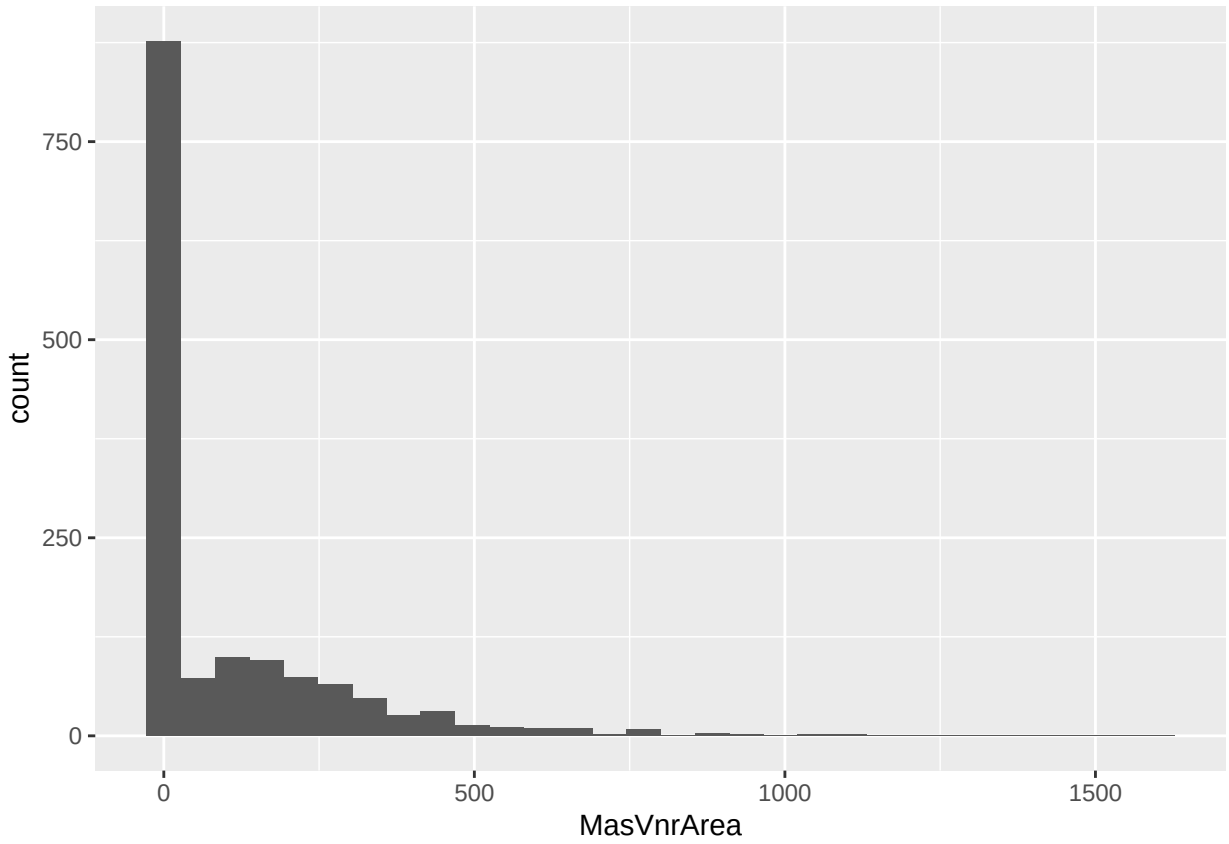
```
# Distribution of the lot frontage
train %>% ggplot(aes(LotFrontage)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



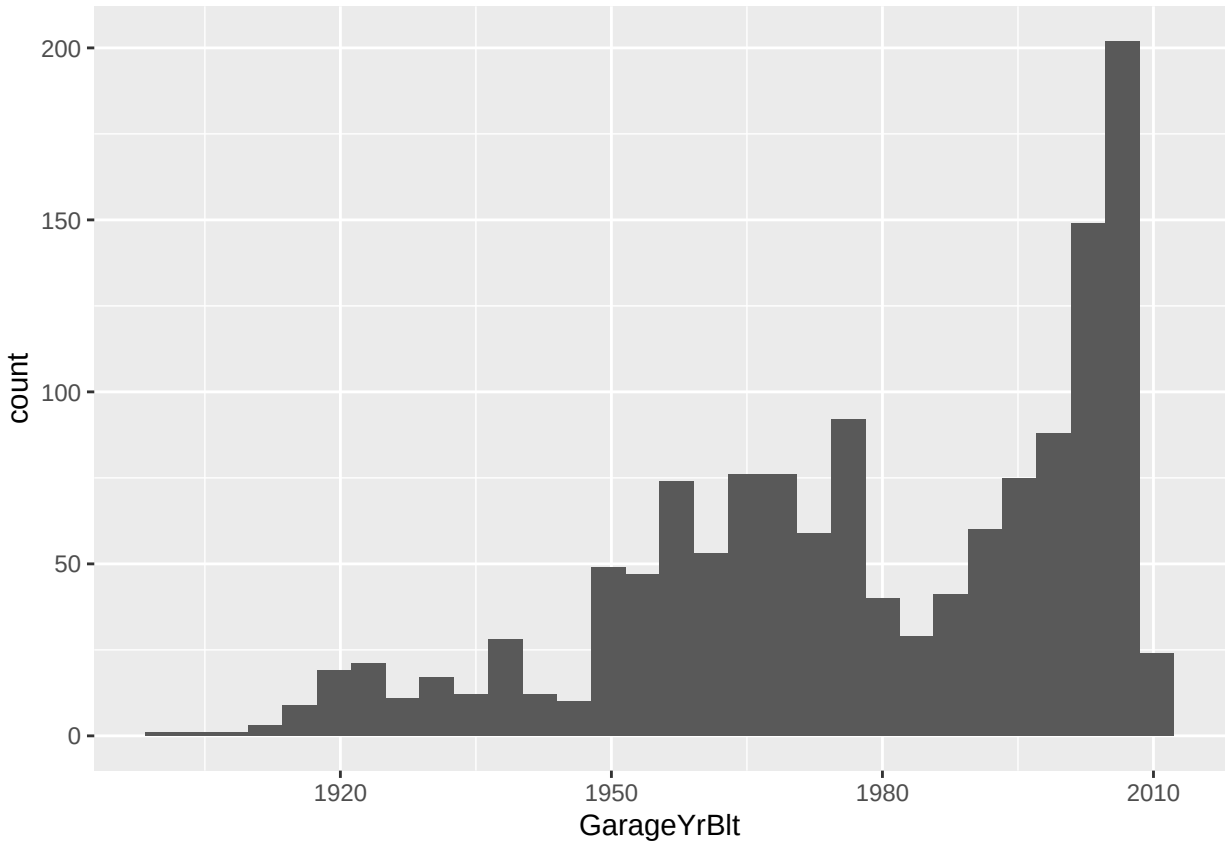

```
# Distribution of the Mas Vnr Area
train %>% ggplot(aes(MasVnrArea)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Distribution of the Year the Garage was built
train %>% ggplot(aes(GarageYrBlt)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Since all of these distributions are either skewed or affected by outliers, I reasoned that it would be unwise to assume any variable's distribution to be symmetrical. I therefore replaced all numerical NAs with the median value (on the train set) of their respective variable. Example for GarageYrBlt:

```
numeric_cols <- numeric_cols %>%
  mutate(GarageYrBlt = ifelse(is.na(GarageYrBlt),
                              median(train$GarageYrBlt,na.rm = T), GarageYrBlt))
```

We can now bind the updated numeric variables back onto the full set:

```
full_set <- full_set %>% select(-all_of(num_col_names)) %>% cbind(numeric_cols)
# No more Nas!
NA_full_set <-apply(full_set, 2, function(x) any(is.na(x)))
which(NA_full_set == T)
```

```
## named integer(0)
```

And now that the data cleaning is done, we can split the full set back into the train and test sets:

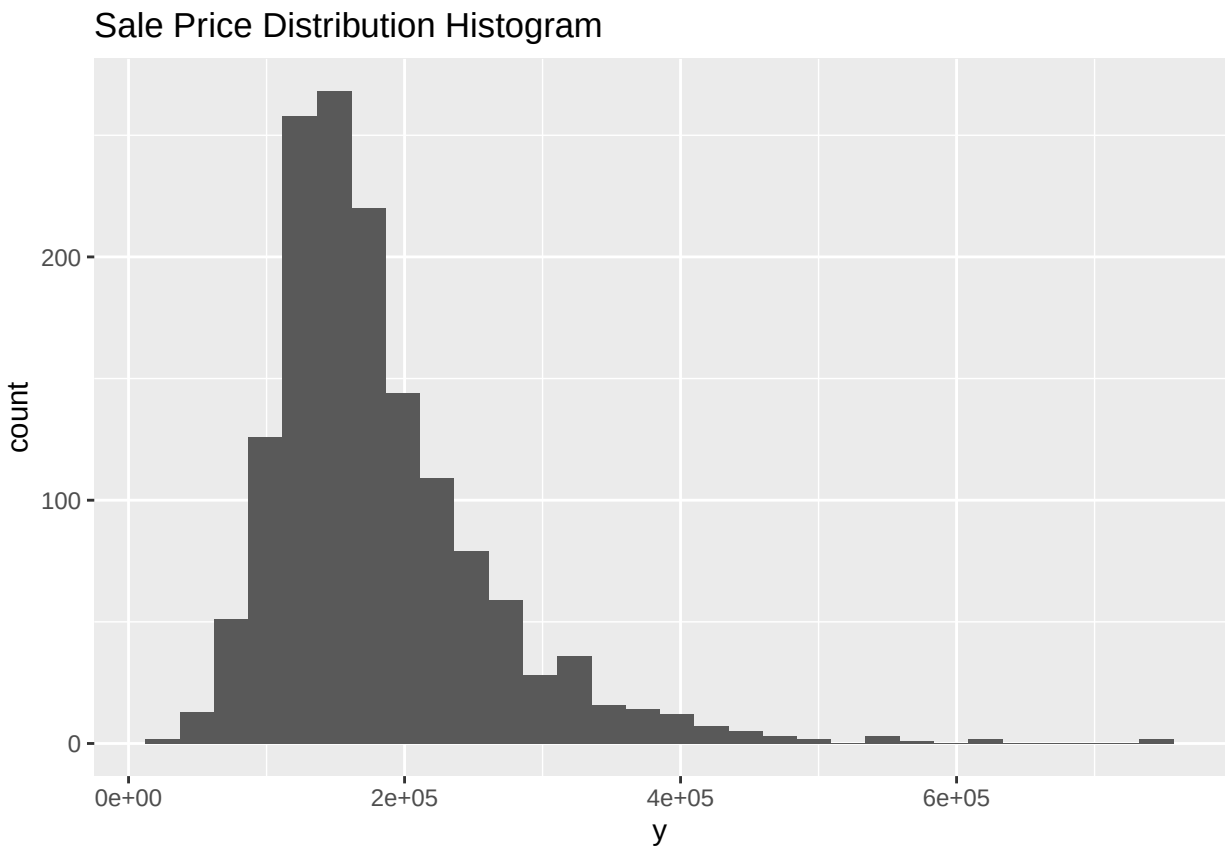
```
i <- 1:nrow(train)
train <- full_set[i,] %>% cbind(y)
test <- full_set[-i,]
```

Baseline model

In order to find the best measure of center for the Sale Price, we can plot its distribution:

```
# Histogram of the SalePrice distribution
train %>% ggplot(aes(y)) +
  geom_histogram() +
  ggtitle("Sale Price Distribution Histogram")
```

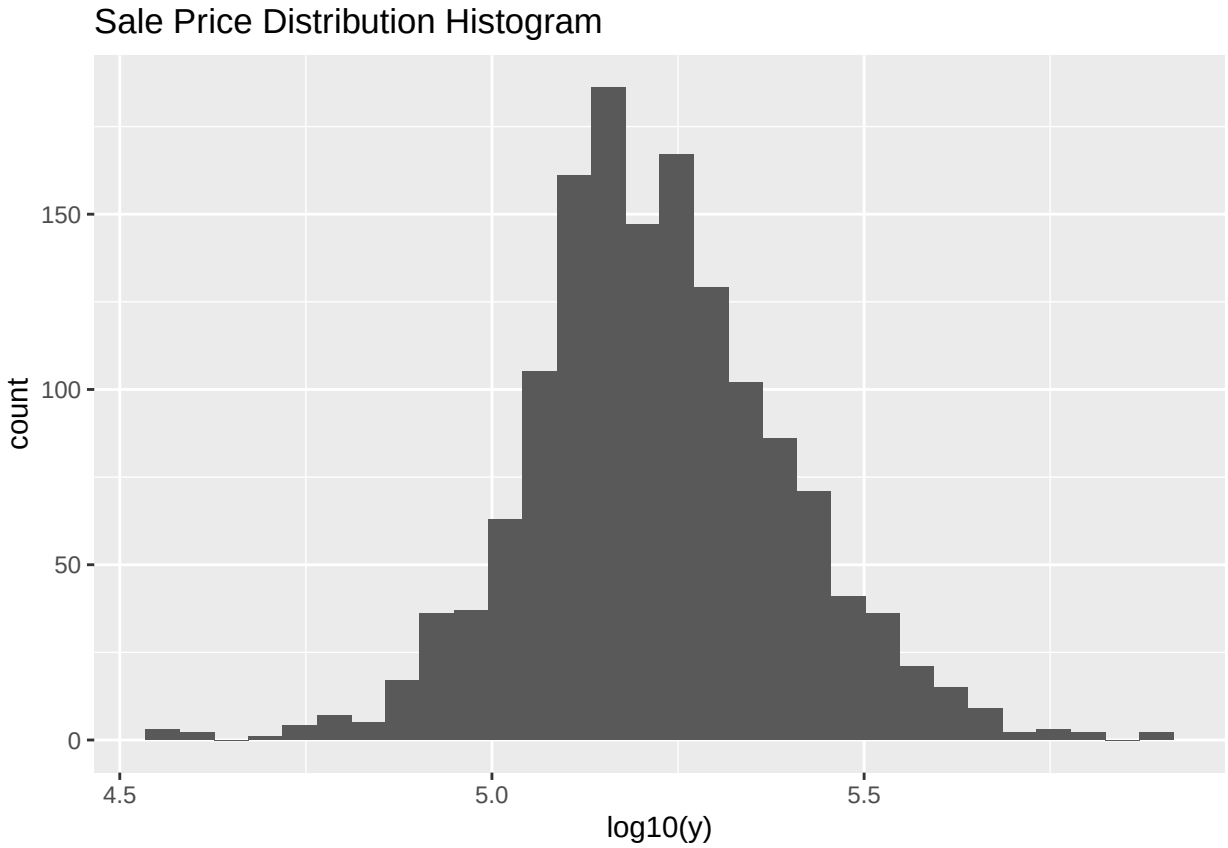
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We can see that it is skewed right, let's try taking the $\log_{10}$ of the sale price instead:

```
train %>% ggplot(aes(log10(y))) +
  geom_histogram() +
  ggtitle("Sale Price Distribution Histogram")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The distribution is now approximatively normal! We can therefore use this equation as the measure of center:

$$\hat{\mu} = 10^{\frac{1}{n}\sum_{i=1}^{n}(\log_{10}(Y_i))}$$

This would give us the following baseline model:

$$\mathbf{Y_i} = \mu + \epsilon_i$$

To use this model in further algorithms, we can add a coulumn to the train set for centered price:

```
y_centered <- train$y - mu
train <- train %>% mutate(y_centered = y_centered)
```

Weighted Knn

I then decided to fit a weighted k nearest neighboors model on the numeric variables (I had previously tried regular knn, but weighted proved itself much more effective).

First I selected the training rows from the numeric columns object, then bound the y-centered column to it:

```
numeric_cols_train <- numeric_cols[i,] %>% cbind(y_centered)
```

Then I trained the model using the caret "train" function, and the default tuning parameters (kmax = c(5,7,9), distance = 2, kernel = "optimal):

```
fit_num_kknn <- train(y_centered~., method = "kknn",
                      data = numeric_cols_train)
```

The estimated RMSE of this model is:

```
min(fit_num_kknn$results$RMSE)
```

```
## [1] 47080.35
```

Predicting for the test set:

```
predictions_kknn <- predict(fit_num_kknn, test, type = "raw") + mu
predictions_kknn <- data.frame(Id = test$Id, SalePrice = predictions_kknn)
```

In order to implement this model into the next algorithm, we save the effects:

```
num_effect_train <- predict(fit_num_kknn, train, type = "raw")
num_effect_test <- predictions_kknn$SalePrice - mu
y_minus_effects <- y_centered - num_effect_train
```

Random Forest

Since Random Forests are particularly effective on categorical variables, we will use the Rborist package to fit a random forest on the residuals from the previous two models based on the categorical explanatory variables.

First we bind the Sale Price minus effects to the "train" portion of the categorical variables:

```
categorical_cols_train <- categorical_cols[i,] %>% cbind(y_minus_effects)
```

Then we can train the forest using the default tuning parameters (predFixed = c(2,117,232), minNode = 3)

```
fit_kknn_rf <- train(y_minus_effects~., method = "Rborist", data = categorical_cols_train)
```

The following parameters are chosen for the final RF model:

```
fit_kknn_rf$bestTune
```

```
##   predFixed minNode
## 1         2       3
```

And our estimated RMSE is:

```
min(fit_kknn_rf$results$RMSE)
```

```
## [1] 24663.33
```

This is 47.6143856% smaller than only using weighted knn, we will therefore use this as our final model.

Results

In order to get the final predictions for the final model we can use the caret predict function with the random forest model fit on the test set, and add $\hat{\mu}$ and the kknn model effects:

```
predictions_kknn_rf <- (predict(fit_kknn_rf, test, type = "raw") +
                                mu + num_effect_test)
predictions_kknn_rf <- data.frame(Id = test$Id,
                                  SalePrice = predictions_kknn_rf)
```

We can now use the actual predictions to calculate our final RMSE and log RMSE:

```
RMSE(actual_prices$SalePrice, predictions_kknn_rf$SalePrice)
```

```
## [1] 1962.143
```

```
RMSE_log(actual_prices$SalePrice, predictions_kknn_rf$SalePrice)
```

```
## [1] 0.1652389
```

The model actually performed better than estimated on the test set, and gives an overall fairly decent RMSE given the simplicity of the models. When submitted as an entry into the kaggle "getting started" competition, the final predictions rank in the 69th percentile.

Conclusion

While we were able to achieve a decent RMSE with this final model, it is still far from perfect. More types of models could be used, or better tuning.

Future improvements on this model could include splitting explanatory variables even further based on whether they are, continuous, ordinal, discrete, etc... Different algorithms could be tested on each variable, and some unpredictive variables could be left out. Once models are chosen, more thorough tuning could be used, with a wider array of options.

This model could potentially be used in realty to find houses that are worth more or less than their market price, or it could be used on sites such as Zillow.com which rent appartments.

Some limitations of the model are that the dataset it was built on only uses houses from Ames, Iowa. Houses in different areas may depend more on a particular variable, and less on others; therefore this model is mainly useful for predictions in Ames, Iowa, or places similar to it.

Citations:

- Hechenbichler K. and Schliep K.P. (2004) Weighted k-Nearest-Neighbor Techniques and Ordinal Classification, Discussion Paper 399, SFB 386, Ludwig-Maximilians University Munich