Modeling Process

Data Sets

Ames

```
ames <- AmesHousing::make_ames()
ames.h2o <- as.h2o(ames)
```

Attrition

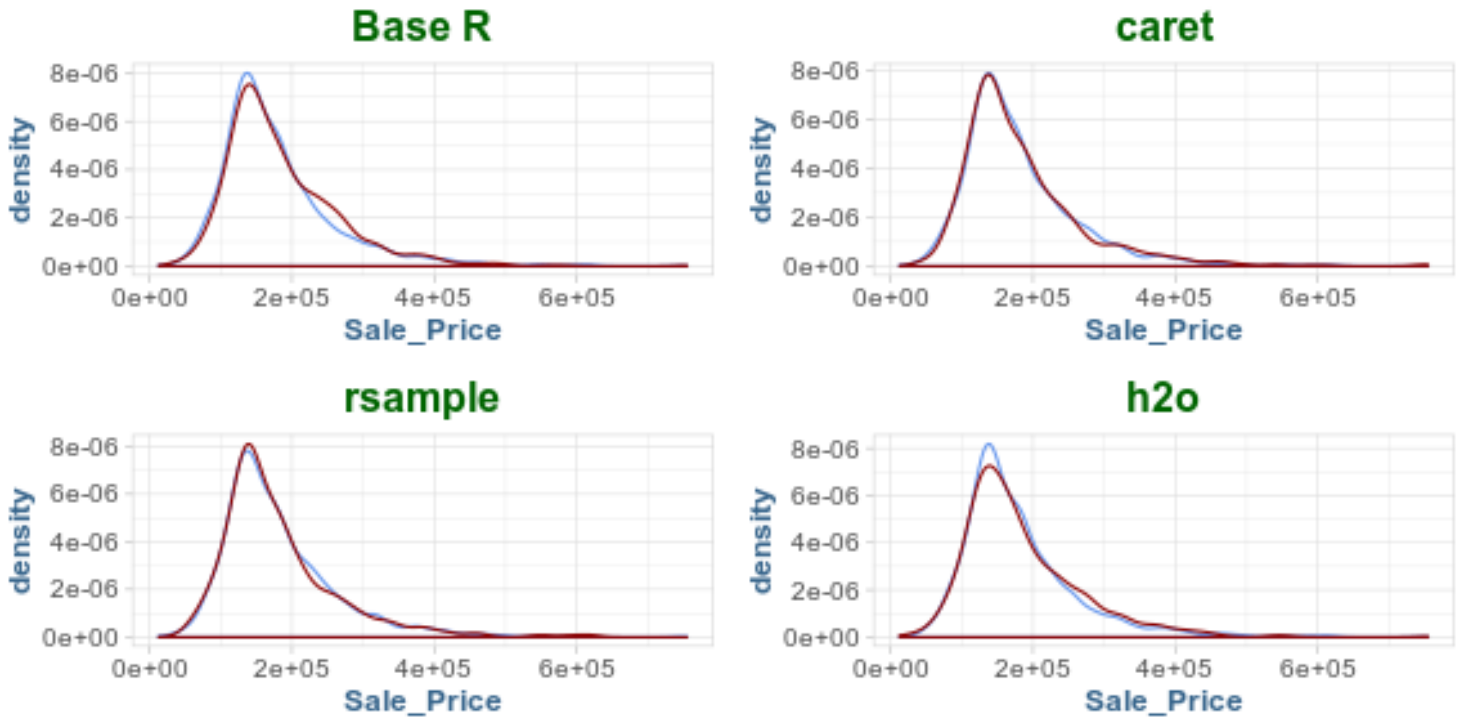
```
attrition <- rsample::attrition

churn <- attrition %>%
   mutate_if(is.ordered, .funs = factor, ordered = F)

churn <- as.h2o(churn)
```

Splitting

```
set.seed(123)

# Base R

index_1 <- sample(1:nrow(ames), round(nrow(ames) * .7), replace = F)
train_1 <- ames[index_1, ]
test_1 <- ames[-index_1, ]

# Using caret package

set.seed(123)

index_2 <- createDataPartition(ames$Sale_Price, p = 0.7,
                               list = F)
train_2 <- ames[index_2, ]
test_2 <- ames[-index_2, ]

# Using rsample package

set.seed(123)
```

```
split_1 <- initial_split(ames, prop = 0.7)
train_3 <- training(split_1)
test_3 <- testing(split_1)

# Using h20 package

split_2 <- h2o.splitFrame(ames.h2o, ratios = 0.7,
                          seed = 123)

train_4 <- split_2[[1]]
test_4 <- split_2[[2]]
```

```
p1 <- ggplot() +
   geom_density(data = train_1, aes(Sale_Price), col = "cornflowerblue") +
   geom_density(data = test_1, aes(Sale_Price), col = "darkred") +
   labs(title = "Base R")

p2 <- ggplot() +
   geom_density(data = train_2, aes(Sale_Price), col = "cornflowerblue") +
   geom_density(data = test_2, aes(Sale_Price), col = "darkred") +
   labs(title = "caret")

p3 <- ggplot() +
   geom_density(data = train_3, aes(Sale_Price), col = "cornflowerblue") +
   geom_density(data = test_3, aes(Sale_Price), col = "darkred") +
   labs(title = "rsample")

p4 <- ggplot() +
   geom_density(data = as.data.table(train_4), aes(Sale_Price), col = "cornflowerblue") +
   geom_density(data = as.data.table(test_4), aes(Sale_Price), col = "darkred") +
   labs(title = "h2o")

grid.arrange(p1, p2, p3, p4, nrow = 2)
```



Stratified sampling

Original Response Distribution

```
table(as.data.table(churn)$Attrition) %>% prop.table()
```

```
       No       Yes 
0.8387755 0.1612245 
```

Stratified Sampling with the rsample package

```
set.seed(123)

split_strat <- initial_split(attrition, prop = 0.7,
                             strata = "Attrition")

train_strat <- training(split_strat)
test_strat <- testing(split_strat)

table(train_strat$Attrition) %>% prop.table()
```

```
      No      Yes 
0.838835 0.161165 
```

```
table(test_strat$Attrition) %>% prop.table()
```

```
       No       Yes 
0.8386364 0.1613636 
```

Class Imbalances

Imbalanced data can have significant impact on model perfomance and predictions. A class imbalance is when there is a disproportionate distribution of a specific value.

Example: Defaults (5%) vs Non-defaults (95%)

Generally, the way to resolve is with either up-sampling or down-sampling.

Down-sampling balances the dataset by reducing the size of the abundant class(es) to match the frequencies in the least prevalant class. This method is used when the quantity of the data is sufficent.

Up-sampling is is used with the quantity of the data is insufficent for down-sampling. This technique balances the dataset by increasing the size of the rarer samples.

Cross-Validation

Cross-validation is a resampling method that randomly divides the training data into *k* groups (aka folds) of approximately equal size. The model is fit on k - 1 folds and then the remaining folds are used for performance evaluation. This procedure is repeated *k* times; each time a different fold is treated as the validtion set. After the model runs on all folds, the results are averaged across all runs for the final indicator.

- Rules of thumb for size of *k*: 5, 10
- Most extreme k = n (or leave-one-out cross validation)

Examples:

- h2o CV

```
h2o.cv <- h2o.glm(
   x = c("Sale_Condition"),
   y = c("Sale_Price"),
   training_frame = ames.h2o,
   nfolds = 10 # perform 10-fold CV
)
```

- rsample CV

```
rsample::vfold_cv(ames, v = 10) # nested df
```

```
#  10-fold cross-validation
# A tibble: 10 x 2
```

```
   splits             id
   <named list>       <chr>
 1 <split [2.6K/293]> Fold01
 2 <split [2.6K/293]> Fold02
 3 <split [2.6K/293]> Fold03
 4 <split [2.6K/293]> Fold04
 5 <split [2.6K/293]> Fold05
 6 <split [2.6K/293]> Fold06
 7 <split [2.6K/293]> Fold07
 8 <split [2.6K/293]> Fold08
 9 <split [2.6K/293]> Fold09
10 <split [2.6K/293]> Fold10
```

Bootstrapping

Bootstrapping is another resampling technique that uses random sampling *with replacement*. A bootstrap sample is constructed using the same size as the origional sample.

Observations not contained in a particular bootstrap sample are called out-of-bag (**OOB**).

r+ sample bootstrap

```
bootstraps(ames, times = 10)
```

```
# Bootstrap sampling
# A tibble: 10 x 2
   splits               id
   <list>               <chr>
 1 <split [2.9K/1.1K]> Bootstrap01
 2 <split [2.9K/1.1K]> Bootstrap02
 3 <split [2.9K/1.1K]> Bootstrap03
 4 <split [2.9K/1.1K]> Bootstrap04
 5 <split [2.9K/1.1K]> Bootstrap05
 6 <split [2.9K/1.1K]> Bootstrap06
 7 <split [2.9K/1.1K]> Bootstrap07
 8 <split [2.9K/1.1K]> Bootstrap08
 9 <split [2.9K/1.1K]> Bootstrap09
10 <split [2.9K/1.1K]> Bootstrap10
```

Bias / Variance trade-off

Bias is the difference between the expected (or average) prediction of our model and the correct value which we are trying to predict. It measures how far off in general a model's predictions are from the correct value.

Models with high bias are rarely affected by the noise introduced by resampling.

Variance is the error due to the variabiliy of the data vs a models prediction at that point.

High variance models are more prone to overfitting, using resampling procedures are critical to reduce this risk.

Hyperparameter Tuning

Hyperparameters (aka *tuning parameters*) are the "knobs to twiddle" to control the complexity of a machine learning algorithm, and therefor the bias/variance trade-off.

- One way to perform hyperparameter tuning is to fiddle with hyperparamters manually until you find a great combination.
- There are several automated mechanism that can help here: cartesian grid search, random grid search, etc.

Model Evaluation

The traditional approach to assessing model fit is to assess the residuals of the model and goodness-of-fit (R^2). However, this can lead to misleading conclusions.

A more robust approach is to use a loss fuction. Loss fuctions are metrics that compare the predicted value to the actual value, the output of this function is often referred to as the *error* or the *pseudo residual*.

Regression Models

- **MSE**: Mean squared error is the average of the squared error (MSE = $\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)$). The sqquared component results in larger errors having larger penalties. This (along with RMSE) is the most common error metric to use. **Objective: minimize**
- **RMSE**: Root mean squared error. This simply takes the square root of the MSE metric (RMSE = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \widehat{y}_i\right)^2}$) so that your error is in the same units as your response variable. If your response variable units are dollars, the units of MSE are dollars-squared, but the RMSE will be in dollars. **Objective: minimize**
- **Deviance**: Short for mean residual deviance. In essence, it provides a degree to which a model explains the variation in the set of data when using maximum likelihood estimation. Essentially, this computes a saturated model (i.e., fully featured model) to an unsaturaded model (i.e., the intercept model or the average model). If the response variable distribution is Gaussian, then it will be approximately equal to MSE. When not, it usually gives a more usefull estimate of error. Deviance is often used with classification models. **Objective: minimize**
- **MAE**: Mean absolute error. Similar to MSE but rather than squaring, it just takes the mean absolute difference between the actual and predicted values (MAE = $\frac{1}{n}\sum_{i=1}^{n}\left(|y_i - \widehat{y}_i|\right)$). This results in less emphasis on larger errors than MSE. **Objective: minimize**
- **RMSLE**: Root mean squared logarithmic error. Similar to RMSE but it performs a *log()* transform on the actual and predicted values prior to computing the difference (RMSLE = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(log(y_i+1) - log(\widehat{y}_i+1)\right)^2}$). When your response variable has a wide range of values, large response values with large errors can dominate the MSE/RMSE metric. RMSLE minimizes this impact so that small response values with large errors can have just as meaningful of an impact as large response values with large errors. **Objective: minimize**

- R^2: This is a popular metric that represents the proportion of the variance in the dependent variable that is predictable from the independent variable(s). Unfortunately, it has several limitations. For example, two models built from two different data sets could have the exact same RMSE, but if one has less variability in the response variable, then it would have a lower R^2 than the other. Should not place too much emphasis on this metric. **Objective: maximize**

Classification Models

- **Misclassification:** This is the overall error. For example, say you are predicting 3 classes (*high, medium, low*) and each class has 25, 30, 35 observations, respectfully (90 total). If you misclassify 3 observations of class *high*, 6 of class *medium*, and 4 of class *low*, then you misclassifiedd *13* out of 90 observations resulting in a 14% misclassification rate. **Objective: minimize**
- Mean per class error: This is the average error rate for each class. For the above example, this would be the mean of $\frac{3}{25}, \frac{6}{30}, \frac{34}{35}$, which is 14.5%. If your classes are balanced this will be identical to misclassification. **Objective: minimize**
- MSE: Mean squared error. Computes the distance from 1.0 to the probability suggested. So, say we have three classes, A, B and C and your model predicts a probability of 0.91 for A, 0.07 for B, and 0.02 for C. For example, if the correct answer was A the MSE = $0.09^2 = 0.0081$. **Objective: minimize**
- **Cross-enthropy (aka Log Loss or Deviance)**: Similar to MSE but it incorporates a log of the predicted probability multiplied by the true class. Consequently, this metric disproportionately punished predictions where we predict a small probability for the true class, which is another way of saying having high confidence in the wrong answer is really bad. **Objective: minimize**
- **Gini index**: Mainly used with tree-based methods and commonly referred to as a measure of *purity* where a small value indicates that a node contains predominately observations from a single class. **Objective: minimize**
- **Accuracy**: Overall, how often is the classifier correct? Opposite of misclassification above. Example: $\frac{TP+TN}{total}$. **Objective: maximize**
- **Precision**: How accurately does the classifier predict events? This metric is concerned with maximizing the true positives to false positive ratio. In other words, for the number of predictions that we made, how many were correct? Example: $\frac{TP}{TP+FP}$. **Objective: maximize**
- **Sensitivity (aka recall)**: How accurately does the classifier classify actual events? This metric is concerned with maximizing the true positives to false negatives ratio. In other words, for the events that occurred, how many did we predict? Example: $\frac{TP}{TP+FN}$. **Objective: maximize**
- **Specificity:** How accurately does the classifier classify actual non-events? Example: $\frac{TN}{TN+FP}$. **Objective: maximize**
- **AUC**: Area under the curve. A good classifier will have high precision and sensitivity. This means the classifier does well when it predicts and event will and will not occur, which minimizes false positives and false negatives. **Objective: maximize**

Putting it together

Stratified Sample:

```
set.seed(123)

split <- initial_split(ames, prop = 0.7,
                       strata = "Sale_Price")

ames_train <- training(split)
ames_test <- testing(split)
```

Use a k-nearest neighbor regressor (via caret)

- Resample method: 10-fold CV
- Grid search: hyperparameter k
- Model training & validation: train a k-nn model using our pre-specified resampling procedure (trControl = cv)

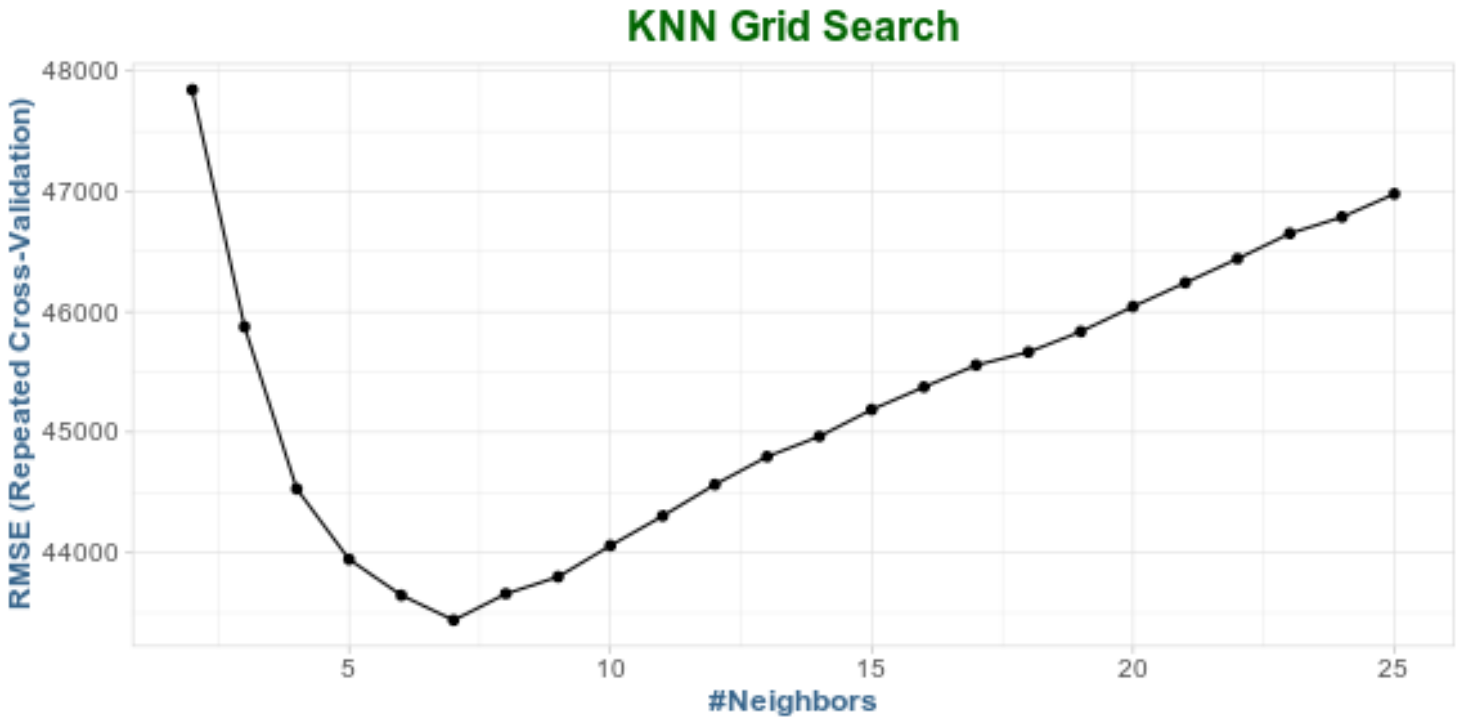
```
# Resampling strategy
cv <- trainControl(
   method = "repeatedcv",
   number = 10,
   repeats = 5
)

# Create grid of hyperparameter values
hyper_grid <- expand.grid(k = seq(2, 25, by = 1))

# Tune a knn model using grid search
knn_fit <- train(
   Sale_Price ~.,
   data = ames_train,
   method = "knn",
   trControl = cv,
   tuneGrid = hyper_grid,
   metric = "RMSE"
)
```

```
knn_fit
```

```
k-Nearest Neighbors

2053 samples
  80 predictor

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 1848, 1848, 1848, 1847, 1849, 1847, ...
Resampling results across tuning parameters:
```

```
  k   RMSE      Rsquared   MAE
  2   47844.53  0.6538046  31002.72
  3   45875.79  0.6769848  29784.69
  4   44529.50  0.6949240  28992.48
  5   43944.65  0.7026947  28738.66
  6   43645.76  0.7079683  28553.50
  7   43439.07  0.7129916  28617.80
  8   43658.35  0.7123254  28769.16
  9   43799.74  0.7128924  28905.50
 10   44058.76  0.7108900  29061.68
 11   44304.91  0.7091949  29197.78
 12   44565.82  0.7073437  29320.81
 13   44798.10  0.7056491  29475.33
 14   44966.27  0.7051474  29561.70
 15   45188.86  0.7036000  29731.56
 16   45376.09  0.7027152  29860.67
 17   45557.94  0.7016254  29974.44
 18   45666.30  0.7021351  30018.59
 19   45836.33  0.7013026  30105.50
 20   46044.44  0.6997198  30235.80
 21   46242.59  0.6983978  30367.95
 22   46441.87  0.6969620  30481.48
 23   46651.66  0.6953968  30611.48
 24   46788.22  0.6948738  30681.97
 25   46980.13  0.6928159  30777.25

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 7.
```

```
ggplot(knn_fit) + labs(title = "KNN Grid Search")
```


KNN Grid Search
RMSE (Repeated Cross-Validation)
48000
47000
46000
45000
44000
5
10
15
20
25
#Neighbors