# Tuning Parameters

(based on tmwr.org)

Matthew McDonald

# Hyperparamters

- Some parameters required for prediction can be estimated directly from the training data,
- Other parameters, called *tuning parameters* or *hyperparameters*, must be specified ahead of time and can’t be directly found from training data.
- These are unknown structural or other kind of values that have significant impact on the model.

# OLS Model Parameters

In ordinary linear regression, there are two parameters $\beta_0$ and $\beta_1$ of the model:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

When we have the outcome ($y$) and predictor ($x$) data, we can estimate the two parameters $\beta_0$ and $\beta_1$:

$$\hat{\beta_1} = \frac{\sum_i (y_i - \bar{y})(x_i - \bar{x})}{\sum_i (x_i - \bar{x})^2}$$

and

$$\hat{\beta_0} = \bar{y} - \hat{\beta_1} \bar{x}.$$

# K-Nearest Neighbor Model

K-nearest neighbors stores the training set (including the outcome).

When a new sample is predicted, K training set points are found that are most similar to the new sample being predicted.

The predicted value for the new sample is some summary statistic of the neighbors, usually:

- the mean for regression, or
- the mode for classification.

# KNN Model Parameters

For the KNN model, the prediction equation for a new value $x_0$ is

$$\hat{y} = \frac{1}{K} \sum_{\ell=1}^{K} x_\ell^*$$

- $K$ is the number of neighbors and the $x_\ell^*$ are the $K$ closest values to $x_0$ in the training set.
- The model itself is not defined by a model equation
- The number of neighbors has a profound impact on the model; it governs the flexibility of the class boundary.
- For small values of $K$, the boundary is very elaborate while for large values, it might be quite smooth.

# Note on KNN Model

Since the model is measuring distance, we typically should add a pre-processing step to center and scale all numeric parameters to ensure they're on the same scale.

# Fitting a KNN Model to Ames

```r
knn_mod <-
  nearest_neighbor(neighbors = 5) %>%
  set_engine("kknn") %>%
  set_mode("regression")

# since Longitude and Latitude are already on the same scale,
# we can get away without centering and scaling
knn_wflow <-
  workflow() %>%
  add_formula(Sale_Price ~ Longitude + Latitude) %>%
  add_model(knn_mod)

set.seed(1001)
ames_folds <- vfold_cv(ames_train, v = 10)

knn_fit <- knn_wflow %>% fit_resamples(resamples = ames_folds)
collect_metrics(knn_fit)
```

```
# A tibble: 2 × 6
  .metric .estimator   mean     n std_err .config
  <chr>   <chr>       <dbl> <int>   <dbl> <chr>
1 rmse    standard   0.0987    10 0.00352 Preprocessor1_Model1
2 rsq     standard   0.686     10 0.0179  Preprocessor1_Model1
```

# Setting K = 100

```r
knn_mod <-
  nearest_neighbor(neighbors = 100) %>%
  set_engine("kknn") %>%
  set_mode("regression")

knn_wflow <-
  knn_wflow %>%
  remove_model() %>%
  add_model(knn_mod)

knn_fit <- knn_wflow %>% fit_resamples(resamples = ames_folds)
collect_metrics(knn_fit)
```

```
# A tibble: 2 × 6
  .metric .estimator  mean     n std_err .config
  <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1 rmse    standard   0.115    10 0.00335 Preprocessor1_Model1
2 rsq     standard   0.574    10 0.0151  Preprocessor1_Model1
```

# What is the Best Choice for K?

# The tune() Function

How can we signal to tidymodels functions which arguments should be optimized? Parameters are marked for tuning by assigning them a value of `tune()`.

The `tune()` function doesn't execute any particular parameter value; it only returns an expression:

```r
tune()
```

```
tune()
```

Embedding this `tune()` value in an argument will tag the parameter for optimization.

# Tuning our KNN Model

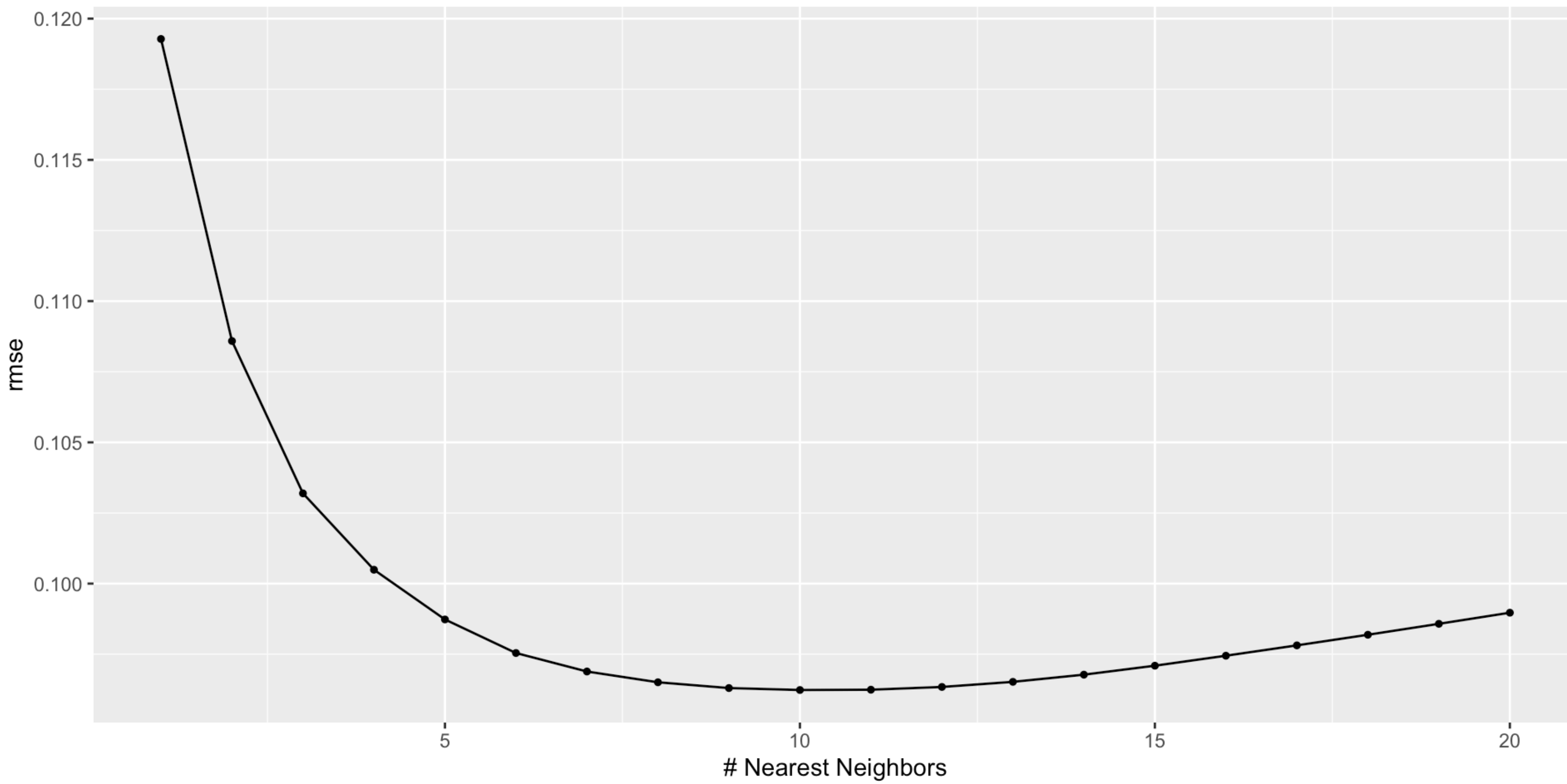
```r
knn_mod <-
  nearest_neighbor(neighbors = tune('K')) %>%
  set_engine("kknn") %>%
  set_mode("regression")

knn_wflow <-
  knn_wflow %>%
  remove_model() %>%
  add_model(knn_mod)

knn_tune <- knn_wflow %>% tune_grid(resamples = ames_folds,
                                    grid = tibble(K=1:20),
                                    metrics=metric_set(rmse))
collect_metrics(knn_tune)
```

# Tuning our KNN Model

```
# A tibble: 20 × 7
       K .metric .estimator   mean     n std_err .config
   <int> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
 1     1 rmse    standard   0.119     10 0.00417 Preprocessor1_Model01
 2     2 rmse    standard   0.109     10 0.00393 Preprocessor1_Model02
 3     3 rmse    standard   0.103     10 0.00371 Preprocessor1_Model03
 4     4 rmse    standard   0.100     10 0.00360 Preprocessor1_Model04
 5     5 rmse    standard   0.0987    10 0.00352 Preprocessor1_Model05
 6     6 rmse    standard   0.0975    10 0.00347 Preprocessor1_Model06
 7     7 rmse    standard   0.0969    10 0.00344 Preprocessor1_Model07
 8     8 rmse    standard   0.0965    10 0.00342 Preprocessor1_Model08
 9     9 rmse    standard   0.0963    10 0.00338 Preprocessor1_Model09
10    10 rmse    standard   0.0962    10 0.00333 Preprocessor1_Model10
11    11 rmse    standard   0.0962    10 0.00329 Preprocessor1_Model11
12    12 rmse    standard   0.0963    10 0.00328 Preprocessor1_Model12
13    13 rmse    standard   0.0965    10 0.00327 Preprocessor1_Model13
```

# Plotting the Results

```
autoplot(knn_tune)
```



# Selecting the Best Parameters

```r
knn_best <- select_best(knn_tune)
knn_best
```

```
# A tibble: 1 × 2
      K .config
  <int> <chr>
1    10 Preprocessor1_Model10
```

# Getting the Best Model for Prediction

```r
knn_final <-
  knn_wflow %>%
  finalize_workflow(knn_best)

knn_final
```

```
══ Workflow ════════════════════════════════════════════════════════════════════
Preprocessor: Formula
Model: nearest_neighbor()

── Preprocessor ────────────────────────────────────────────────────────────────
Sale_Price ~ Longitude + Latitude

── Model ───────────────────────────────────────────────────────────────────────
K-Nearest Neighbor Model Specification (regression)

Main Arguments:
  neighbors = 10

Computational engine: kknn
```

Note: Once we have this “final” model, we would still need to fit it with the **entire** training data set in order to then use it for prediction.

# Other Tuning Parameters

- Boosting is an ensemble method that combines a series of base models, each of which is created sequentially and depends on the previous models
  - The number of boosting iterations is a tuning parameter
- In single-layer artificial neural network, the predictors are combined using two or more hidden units. The hidden units are linear combinations of the predictors that are captured in an *activation function* (typically a nonlinear function, such as a sigmoid).
  - The number of hidden units and the type of activation are tuning parameters.
- Modern gradient descent methods are improved by finding the right optimization parameters.
  - Examples of such hyperparameters are learning rates, momentum, and the number of optimization iterations/epochs.
  - Neural networks and some ensemble models use gradient descent to estimate the model parameters.

# Tuning Preprocessing Steps

- In principal component analysis, the predictors are replaced with new, artificial features that have better properties related to collinearity.
  - The number of extracted components can be tuned.
- Imputation methods estimate missing predictor values using the complete values of one or more predictors. One effective imputation tool uses $K$-nearest neighbors of the complete columns to predict the missing value.
  - The number of neighbors can be tuned.

# Tuning Structural Parameters

- In binary regression, the logit link is commonly used (i.e., logistic regression). Other link functions, such as the probit and complementary log-log, are also available and can be tuned.
- Non-Bayesian longitudinal and repeated measures models require a specification for the covariance or correlation structure of the data. Options include compound symmetric (a.k.a. exchangeable), autoregressive, Toeplitz, and others, which can be tuned.

# Two general strategies for optimization

Tuning parameter optimization usually falls into one of two categories: *grid search* and *iterative search*.

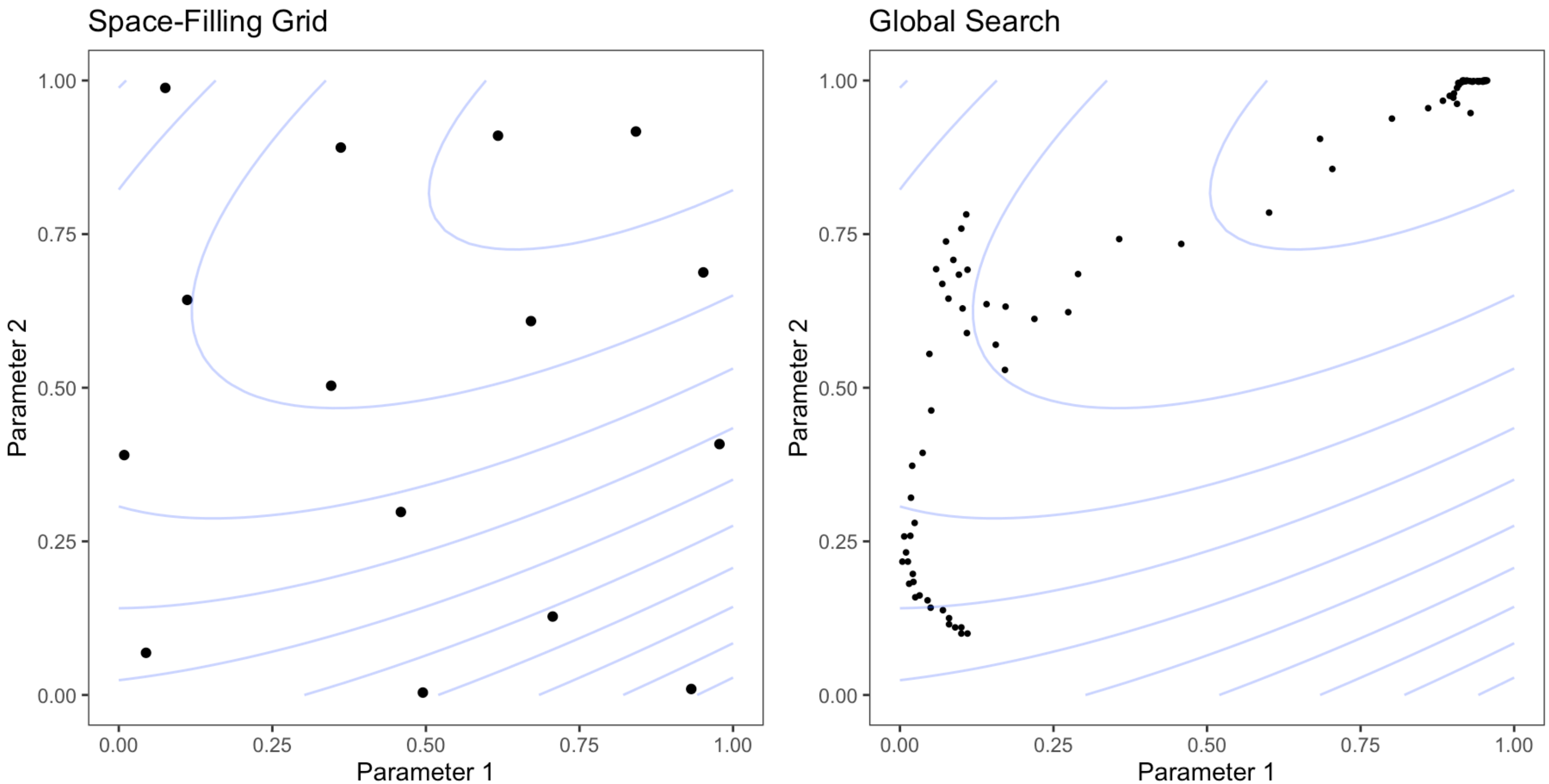
# Grid Search

- *Grid search* is when we predefine a set of parameter values to evaluate
- The main choices involved in grid search are how to make the grid and how many parameter combinations to evaluate
- Grid search is often judged as inefficient since the number of grid points required to cover the parameter space can become unmanageable with the curse of dimensionality
- Lots of times it gets the job done

# Iterative Search

- *Iterative search* or sequential search is when we sequentially discover new parameter combinations based on previous results
- Almost any nonlinear optimization method is appropriate, although some are more efficient than others.
- In some cases, an initial set of results for one or more parameter combinations is required to start the optimization process.

# Visualizing the Two Approaches




Examples of pre-defined grid tuning and an iterative search method. The lines represent contours of a performance metric; it is best in the upper-right-hand side of the plot.