Data Science for Public Policy

Aaron R. Williams - Georgetown University

Supervised Machine Learning Part 1

Reading

- Bit By Bit section 3.6.2
- Hands on Machine Learning with R sections 1.1-1.2 and chapter 2
- R2D3 introduction to the bias-variance tradeoff

Machine Learning Concepts #1

Machine Learning: The use of computer algorithms to parse data and estimate models that can be used for prediction, categorization, or dimension reduction.

Predictive Modeling: "The process of developing a mathematical tool or model that generates an accurate prediction." ~ Max Kuhn and Kjell Johnson in Applied Predictive Modeling

Supervised Learning

Supervised learning: Predictive modeling with a "target", "response", or "outcome" variable.

Regression: Supervised learning with a numeric outcome.

Classification: Supervised learning with a categorical outcome. The output of these models can be predicted classes of a categorical variable or predicted probabilities (e.g. 0.75 for "A" and 0.25 for "B").

Unsupervised Learning

Unsupervised learning: A process of summarizing data without a "target", "response", or "outcome" variable.

Clustering: Grouping observations into homogeneous groups.

Dimension reduction: Reducing the number of variables in a data set while maintaining the statistical properties of the data.

Three Reasons to Build a Statistical Model

1. Statistical summary
2. Inference
3. Prediction

1. Statistical summary

Much like a mean can be used to summarize a collection of numbers, a statistical model can be used to summarize relationships between variables. Sample mean is the expected value of a variable $E(Y)$. Simple linear regression represents a conditional mean $E(Y_i|X_i) = \beta_0 + \beta_1 X_i$.

The simplest example is a linear regression with Y continuous as the dependent variable and X categorical as $n-1$ dummy variables X_j representing the levels of the categorical variable as the independent variable. The estimated relationship is equivalent to Analysis of Variance (ANOVA) and can be used to examine the group-wise means in the data.

We will focus on methods for statistical summary and dimension reduction in a few weeks.

2. Inference

The second reason to build a statistical model is for inference. Here, the goal is to test a set of formal hypotheses with H_0 as the null hypothesis and H_a as the alternative hypothesis. The hypotheses usually focus on the coefficients (e.g. $\beta_1 \neq 0$). Care should be taken with inference to develop hypotheses based on theory and to limit the number of tests conducted on a given set of data. Assumptions are also tremendously important. For example, simple linear regression assumes:

1. Population model: $Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$
2. The estimation data come from a random sample or experiment
3. $\epsilon_i \sim N(0, \sigma^2)$ independently and identically distributed (i.i.d.)

If these assumptions are approximately met, then test statistics can be developed from known sampling distributions. For coefficients, this is the t-distribution with $n-p$ degrees of freedom.

3. Prediction

The final motivation for building a statistical model is prediction. Here, the goal is make informed and accurate guesses about the value or level of a variable given a set of predictor variables. Unlike inference, which usually focuses on coefficients of predictor variables, the focus here is on the dependent variable.

Prediction will be the focus of supervised machine learning.

...

The same statistical model can summarize, be used for inference, and make valid and accurate predictions. However, the optimal model for one motivation is rarely best for all three motivations. Thus, it is important to clearly articulate the motivation for a statistical model before picking which tools and diagnostics to use.

Contrasting Approaches to OLS Regression

Approach 1: Inferencial Models for Prediction (How regression if typically taught)

GOAL: Find the minimum variance unbiased estimator of a linear relationship between a predictor variable and an outcome variable–possibly holding some other set of predictors constant.

Process:

1. Develop a null hypothesis H_0 and alternative hypothesis H_a from theory and potentially cautious exploratory data analysis (EDA). Avoid mining the data for spurious relationships.
2. Develop an identification strategy and a regression specification that considers functional form and covariates.
3. Make assumptions.
    1. Population model: $Y_i = \beta_0 + \beta_j X_{ij} + \epsilon_i$
    2. The estimation data come from a random sample or experiment
    3. $\epsilon_i \sim N(0, \sigma^2)$ i.i.d.
4. Estimate the regression model.
5. Validate the model for goodness-of-fit and model assumptions.
    1. Use residual analysis to evaluate the normality and constancy of the error term.
    2. R^2
    3. Check for multicollinearity.
6. Remediate based on meeting the assumptions (e.g. use WLS if the errors are non-constant).
7. Robustness checks.

Ideally, only one model is estimated until remediation or robustness checks to avoid data mining or Hypothesizing After the Results Are Known (HARKing). If not, confidence intervals and standard errors should be adjusted for multiple testing (e.g. Bonferroni correction).

The focus is usually on the coefficients and the hypothesis test but the estimated model can be used to make predictions. Prediction intervals for an individual $\hat{Y}$ can be calculated:

$$\hat{Y}_h \pm t_{n-p,\frac{\alpha}{2}} \cdot \sqrt{MSE[\frac{1}{m} + \frac{1}{n} + \frac{(X_h - \bar{X})^2}{\sum(X_i - \bar{X})^2}]}$$

A lot of effort goes into ensuring that the model doesn't over fit the data. The process begins with theory. Adjustments are made for multiple testing.

Furthermore, by meeting certain assumptions and using sampling distributions of test statistics, inferential models can make valid statements about the uncertainty of the model, its coefficients, and predictions made from the model. Note, model validation, assessing the goodness-of-fit, is done on the same data used for estimating the model. The process is "in sample".

Approach 2: Prediction (an ML approach)

GOAL: Minimize the prediction error of the estimated regression model on new data (out-of-sample).

There are several metrics for estimating error. The most popular for linear regression is Root Mean Square Error (RMSE):

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

With this prediction approach, much less emphasis is placed on the model assumptions. It's probably important that the underlying relationship is linear for linear regression, but the distribution of the error terms isn't that important if the model has more predictive accuracy.

Ensuring that our model does not over fit the data and is generalizable to other data is a motivating challenge in machine learning. Accordingly, we estimate the prediction error of the model on new data (an out-of-sample error rate) using some hold-out data and resampling methods.

Two Implications

Switching from an inferential framework to a predictive framework results in two important implications.

1. New Algorithms

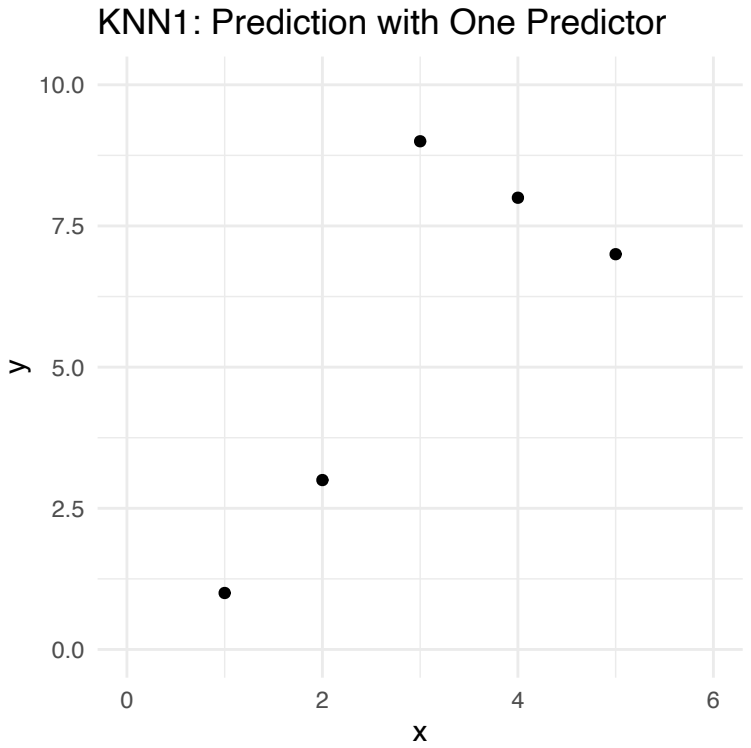
Models that are easily interpretable are less useful if the sole objective is to make accurate predictions. Accordingly, predictive modeling considers a much wider range of parametric and nonparametric models. Some of the models are difficult or nearly impossible to understand.

It is easy to get caught up in all of algorithms, but it is far more important to understand the process of predictive modeling. For now, one new algorithm will be enough.

K-Nearest Neighbors (KNN)

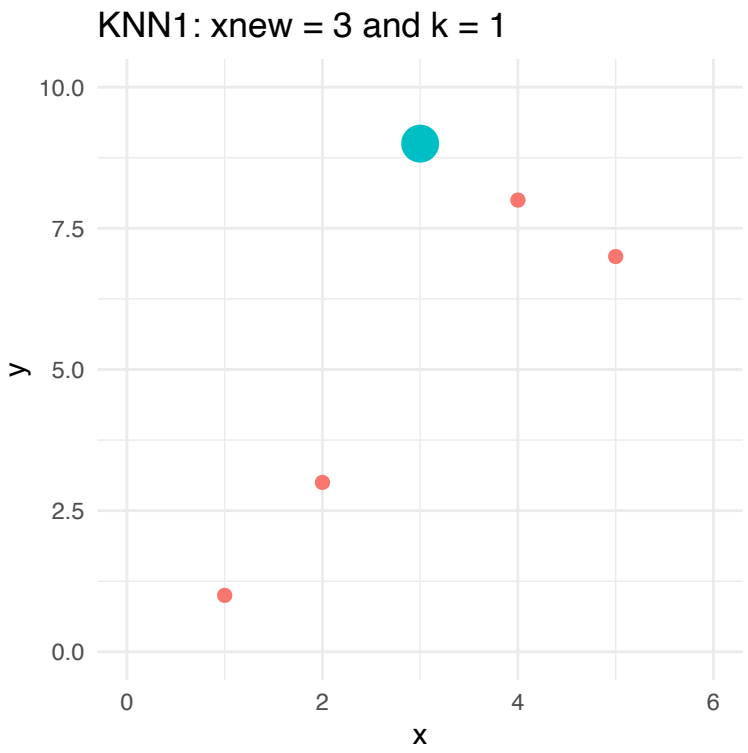
K-Nearest Neighbors (KNN) is an algorithm that makes predictions based on the average (regression) or majority vote (classification) of the k most similar observations. Similar is measured by the distances between predictors in the training data and the observation for which a prediction is being made.

Consider the following five observations:

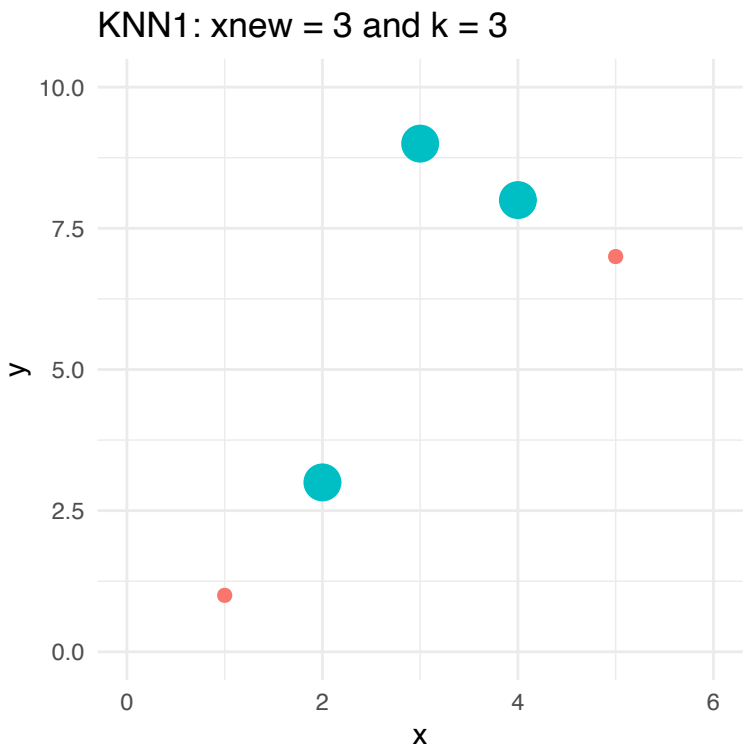


Suppose for a given *new* observation with a known value of x_{new} and an unknown value of y, the goal is to predict $\hat{y}$. KNN finds the closest k values of x to x_{new} and predicts $\hat{y}$ as the mean of the y values for the k observations.

If $x_{new} = 3$ and $k = 1$, then $\hat{y} = 9$.



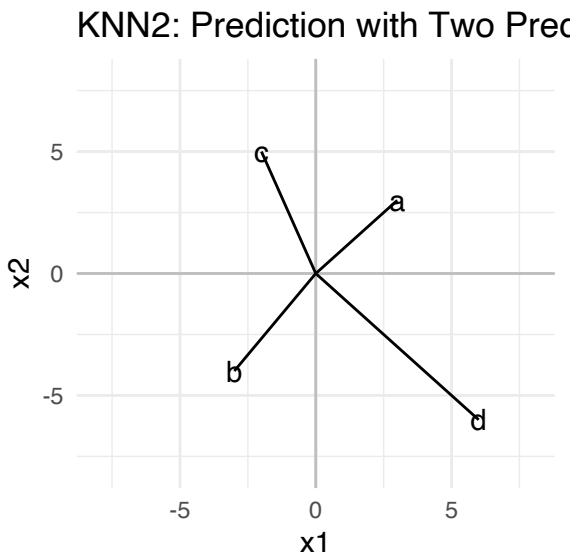
If $x_{new} = 3$ and $k = 3$, then $\hat{y} = \frac{3+9+8}{3} \approx 6.67$.



Finding the closest observation is trivial for a 1-dimensional predictor. Of course, most applications have more than one predictor. In these applications, Euclidean distance is a common way of measuring the distances between $\mathbf{X}$ and $\vec{x}_{new}$:

$$\text{Euclidean Distance} = \sqrt{\sum_{k=1}^{P}(x_{ik} - x_{jk})^2}$$

Consider the following four observations with two predictors, x_1 and x_2.



$$dist_a = \sqrt{(3-0)^2 + (3-0)^2} = \sqrt{9+9} = \sqrt{18}$$

$$dist_b = \sqrt{(-3-0)^2 + (-4-0)^2} = \sqrt{9+16} = \sqrt{25}$$

$$dist_c = \sqrt{(-2-0)^2 + (5-0)^2} = \sqrt{4+25} = \sqrt{29}$$

$$dist_d = \sqrt{(6-0)^2 + (-6-0)^2} = \sqrt{36+36} = \sqrt{72}$$

When $k = 3$, the closest three observations are $\{a, b, c\}$

Note: All predictors need to be numeric and the scale of predictors is an important consideration when using KNN.

2. How do we determine if our model is useful?

Unlike algorithms used for inference, many of the algorithms used for predictive modeling do not have distributional assumptions. This means that many diagnostics based on distributional assumptions are no longer available for model evaluation: F-test, standard error of a coefficients, t-test, prediction interval, regression line interval.

Instead, an error metric is chosen and then a key objective is estimating the out-of-sample value of that error rate using available data. The out-of-sample error rate is rarely known. Instead it is estimated.

Out-of-sample Error Rate

Model uncertainty for inferential models is estimated with sample variance and sampling distributions. In a purely predictive approach, uncertainty is estimated by the error rate on data not used to estimate the predictive model. This means observations from a data set need to be set aside before estimating the model.

Generalizability: How well a model makes predictions on unseen data relative to how well it makes predictions on the data used to estimate the model.

In-sample error: The predictive error of a model measured on the data used to estimate the predictive model.

Out-of-sample error: The predictive error of a model measured on the data **not** used to estimate the predictive model. Out-of-sample error is generally greater than the in-sample error.

Training set: A subset of data used to develop a predictive model. The share of data committed to a training set depends on the number of observations, the number of predictors in a model, and heterogeneity in the data. 0.8 is a common share.

Testing set: A subset of data used to estimate model performance. The testing set usually includes all observations not included in the testing set. *Do not look at these data until the very end and only estimate the out-of-sample error rate on the testing set once.* If the error rate is estimated more than once on the testing data, it will underestimate the error rate.

Data leakage: When information that won't be available when the model makes out-of-sample predictions is used when estimating a model. Looking at data from the testing set creates data leakage. Data leakage leads to an underestimate of out-of-sample error.

Bias-Variance Trade-off

Using specialized algorithms (implication 1) and optimizing for an error metric (implication 2) creates a problem. Many algorithms can memorize the training data. This results in overfit models that don't generalize well.

Source: Chris Albon

Not all error is created the same. Prediction error can be decomposed into three types of error: irreducible error, bias error, and variance error. Understanding the types of error will inform modeling decisions.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

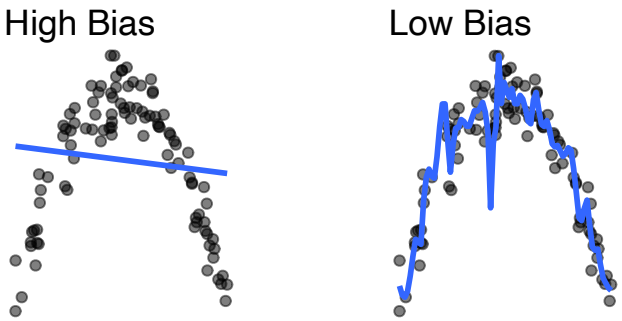
Assuming y_i independent and $y_i - \hat{y}_i \sim N(0, \sigma^2)$, then

$$E[MSE] = \sigma^2 + (\text{model bias})^2 + \text{model variance}$$

Irreducible error (σ^2): Error that can't be reduced regardless of model quality. This is often caused by factors that affect the outcome of interest that aren't measured or included in the data set.

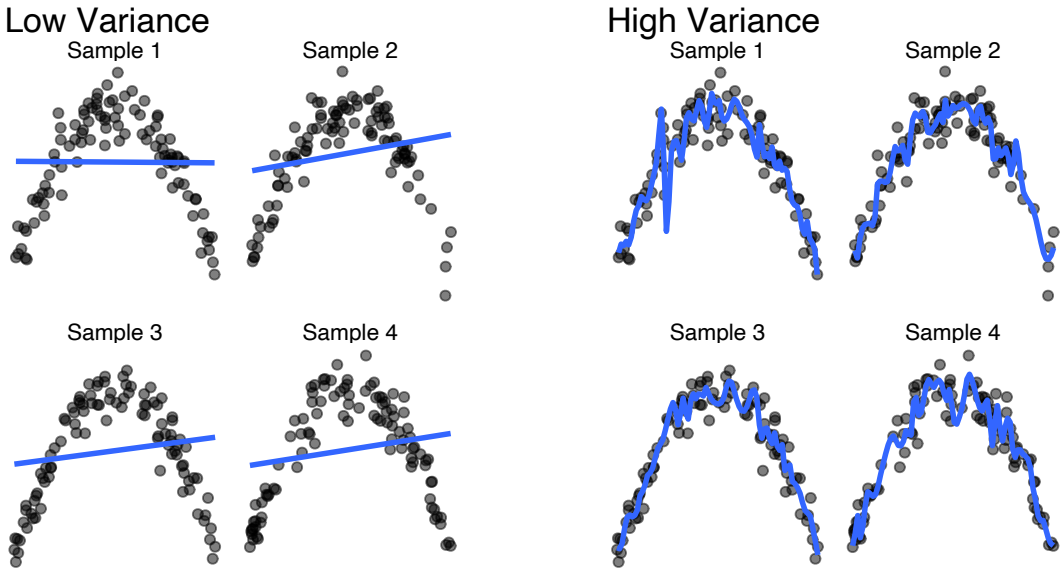
Bias error: Difference between the expected prediction of a predictive model and the correct value. This is how well a model fits the underlying structure of the data.

Here, the model on the left does a poor job fitting the data (high bias) and the model on the right does a good job fitting the data (low bias).



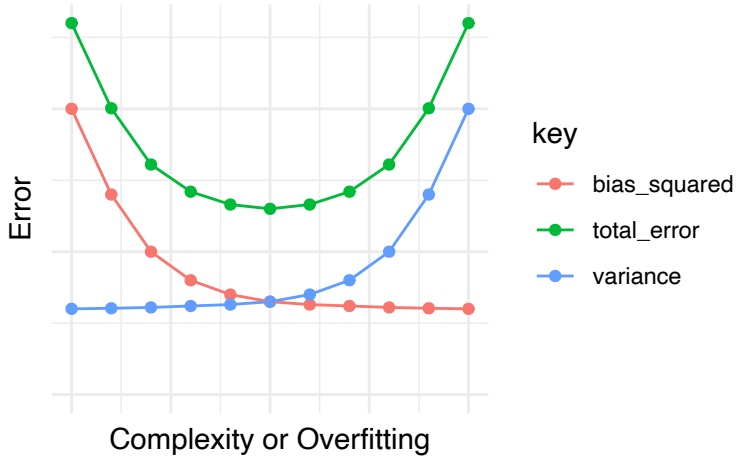
Variance error: Variability of a model prediction given for a given data point. This is how much a model prediction will change if different training data are used.

Here, the model on the left does not change much as the training data change (low variance) and the model on the right changes a lot when the training data change (high variance).



As can be seen in the error decomposition and plots above, for any given amount of total prediction error, there is a trade-off between bias error and variance error. When one type of error is reduced, the other type of error increases.

The bias-variance trade-off can generally be reconceived as a simplicity-complexity trade-off. Simple models often have high bias and low variance. A linear regression with a few predictor variables, a pretty simple model, may do a mediocre job capturing the underlying relationship in the data but it will be stable across resamples and between the training and testing data. Meanwhile, polynomial regression or regression splines, complex models, may do a great job capturing the underlying relationship in any given sample but the estimated model could change a lot between resamples and the training and testing data.



Similarly, the bias-variance trade-off can be reconceived as an underfitting-overfitting

trade-off. KNN with large k tends to be underfit with KNN with small k tends to be overfit.

Overall, the objective is to minimize the total error. In the beginning, this often means building up a model to reduce the bias/simplicity/underfitting of the model. Later, it means reducing a model to reduce variance/complexity/overfitting.

Creating Useful Models

1. Create many candidate models
2. Pick the "best" model from the candidate models

"Best" can be defined by the lowest error estimate. Sometimes the "best" model doesn't have the lowest error estimate but is parsimonious, computationally feasible, avoids issues with bias, or accounts for the costs of different types of errors.

Creating Many Models

Supervised machine learning is mostly concerned with the process of creating many candidate models and then picking the "best" model from among the candidate models. There are at least three ways to generate candidate models.

1. Feature Engineering

Feature engineering: The addition, deletion, and transformation of variables before applying algorithms to the data.

Feature engineering is often the most labor intensive part of the predictive modeling process. Many of the potential steps are similar in concept to preprocessing for estimating a model for statistical inference. However, feature engineering needs be repeated for each resampling iteration.

When preprocessing, Boehmke and Greenwell recommended the following order of preprocessing steps:

1. Filter out predictors with low or no variance
2. Impute missing values
3. Normalize data to resolve skewness
4. Standardize numeric features
5. Dimension reduction
6. Lumping, one-hot encoding, and dummy encoding

2. Picking algorithms

There are many different algorithms than can be used to create predictive models for any given application. Linear regression and logistic regression are two common methods in

policy analysis. There are many other algorithms, like tree-based algorithms, that will be discussed next week.

3. Hyperparameter Tuning

Hyperparameter: Model parameters that are specified before estimating the model.

Linear regression and logistic regression don't have any hyperparameters. K-Nearest Neighbors has one parameter, K, the number of nearest observations to be considered for prediction. The best way to determine the optimal value of K is to try many values of K in a process called hyperparameter tuning. The simplest method for hyperparameter tuning is grid search.

Grid search: A process of trying unique combinations of hyperparameters by specifying a Cartesian grid of the hyperparameters and estimated a model for each row of the grid (with resampling).

For example, build a grid with values of K ranging from 1 to 15 by increments of 2. Estimate a KNN model with resampling for each K. Compare the resulting error metrics and determine the optimal hyperparameter.

Picking from Many Models

Now we'll cover the process of picking the best model from candidate models.

With one model, the training data is used to estimate the model and the testing data is used to estimate the out-of-sample error rate. *The testing data should only be used at the end of the modeling process to estimate the true out-of-sample error rate.* Using the testing data more than once will result in overfitting and an underestimate of the training error because the decisions used to create a model on the training data are at least partially optimized to the unique features of the training data.

So how do we estimate and compare out-of-sample error rates if we can only use the testing data once? Given unlimited data, we could just keep splitting the data into more and more testing sets to estimate the out-of-sample error. That is often impractical. Instead, model performance is generally compared with resampling methods.

Resampling methods iteratively split the training data into subgroups, estimate the model on one portion of the data, and measure the error rate on the other portion of the data. This process happens many times and the error rate metric is then averaged. For example, if resampling happens j times, then then $\hat{RMSE}$ is the mean of the ten $RMSE_j$

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

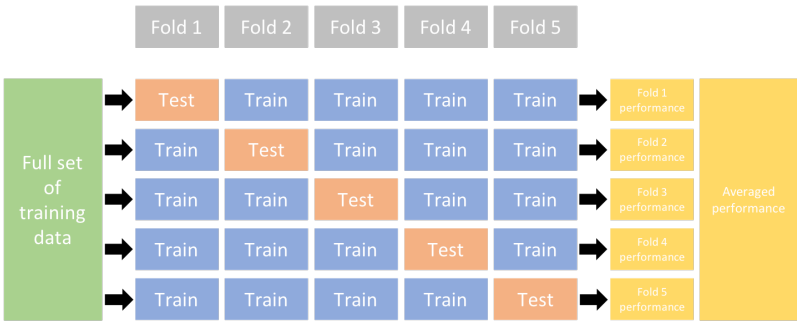
Three popular resampling methods are v-fold cross-validation (also called k-fold cross-validation), leave-one-out cross-validation (a special case of v-fold cross-validation), and bootstrapping. We will focus on v-fold cross-validation.

Analysis data: Training data within a resample (Max Kuhn)

Assessment data: Testing data within a resample (Max Kuhn)

v-fold cross-validation: Break the data into v equal-sized, exclusive groups. Train the model on data from $v-1$ folds (analysis data) and measure the error metric (i.e. RMSE) on the left-out fold (assessment data). Repeat this process v times, each time leaving out a different fold. Average the v error metrics.

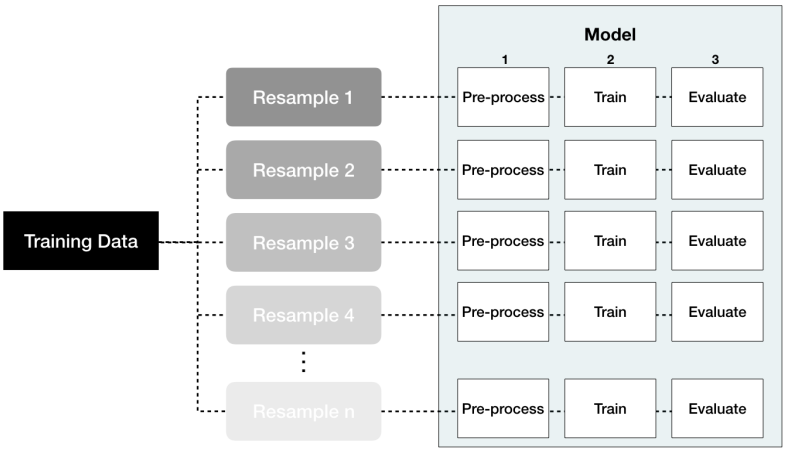
v-fold cross-validation is usually applied to the training data.



Source: Hands-on Machine Learning with R

v is contextual but 10 is standard. Max Kuhn's blog has some convincing simulations that demonstrate the performance of different v.

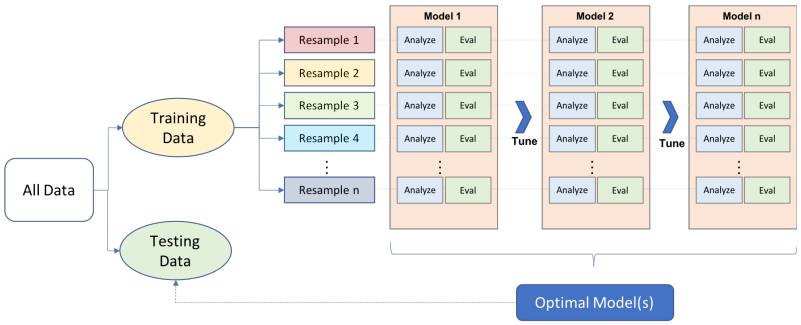
Note: Sample-specific preprocessing needs to be re-estimated within each resample. For example, when mean centering, means need to be recalculated for each resample instead of for the overall training data.



Source: Hands-on Machine Learning with R

Putting it All Together

1. Split the data into a training set and a testing set.
2. Pick a resampling method.
3. Consider many potential models and estimate the out-of-sample error rate based on a resampling method like v-fold cross-validation.
    1. Preprocess and feature engineer the analysis data (training data within a resample). Repeat the process from scratch for each resample.
    2. Use different algorithms
    3. For each algorithm, tune hyperparameters with a method like grid search
4. Pick a final model. Train the final model on all training data without resampling.
5. Estimate the out-of-sample error rate by predicting values in the testing data set.
6. Make predictions.



Source: Hands-on Machine Learning with R

R Code

Source: RStudio

Much predictive modeling in R can be handled with `library(tidymodels)`.

- `library(rsample)` handles resampling.
- `library(parsnip)` is a common interface to packages with predictive algorithms.
- `library(recipes)` handles feature engineering.
- `library(workflows)` manages putting everything together in resampling.
- `library(tune)` helps with hyperparameter tuning.
- `library(yardstick)` is used for evaluating models.

library(rsample)

- `initial_split()` Create an index for creating training and testing data
- `training()` Use the `initial_split()` object to create a training set
- `testing()` Use the `initial_split()` object to create a testing set
- `vfold_cv()` Create indices for v-fold cross-validation
- `analysis()` Create training sets within resamples. Max Kuhn calls these analysis sets.
- `assessment()` Create testing sets within resamples. Max Kuhn calls these assessment sets.

library(recipes)

- `recipe()` Begin creating a recipe for preprocessing
- `step_*()` A collection of functions with preprocessing and feature engineering steps
- `prep()` Estimate parameters for a recipe
- `bake()` Apply the computations from a recipe to a new data set

library(parsnip)

- `nearest_neighbor()` Generate a KNN specification before fitting a model
- `linear_reg()` Generate a linear regression specification before fitting a model
- `random_forest()` Generate a random forests specification before fitting a model
- `set_engine()` Pick the package used to fit a model
- `fit()` Estimate model parameters

library(yardstick)

- `metrics()` Estimate common performance metrics
- `rmse()` Estimate Root Mean Squared Error

library(tune)

- `fit_resamples()` Estimate model parameters with resampling.
- `collect_metrics()` Obtain and format results produced during resampling.
- `collect_predictions()` Obtain and format predictions produced during resampling.

library(workflows)

- `workflow()` Create a container object to manage the model making process.
- `add_model()` Add a parnsip model to a workflow object.
- `add_recipe()` Add a recipe to a workflow object.

Examples

Example 1

Example 1 uses simulated data with one predictor x and one outcome variable y.

Step 0. Load the data

```
library(tidymodels)

set.seed(20201004)

x <- runif(n = 1000, min = 0, max = 10)

data1 <- bind_cols(
  x = x,
  y = 10 * sin(x) + x + 20 + rnorm(n = length(x), mean = 0, sd = 2)
)
```

Here, we know `y = f(x)`. In practice, this is unknown and our goal is to estimate or approximate it.

Step 1. Split the data into training and testing data

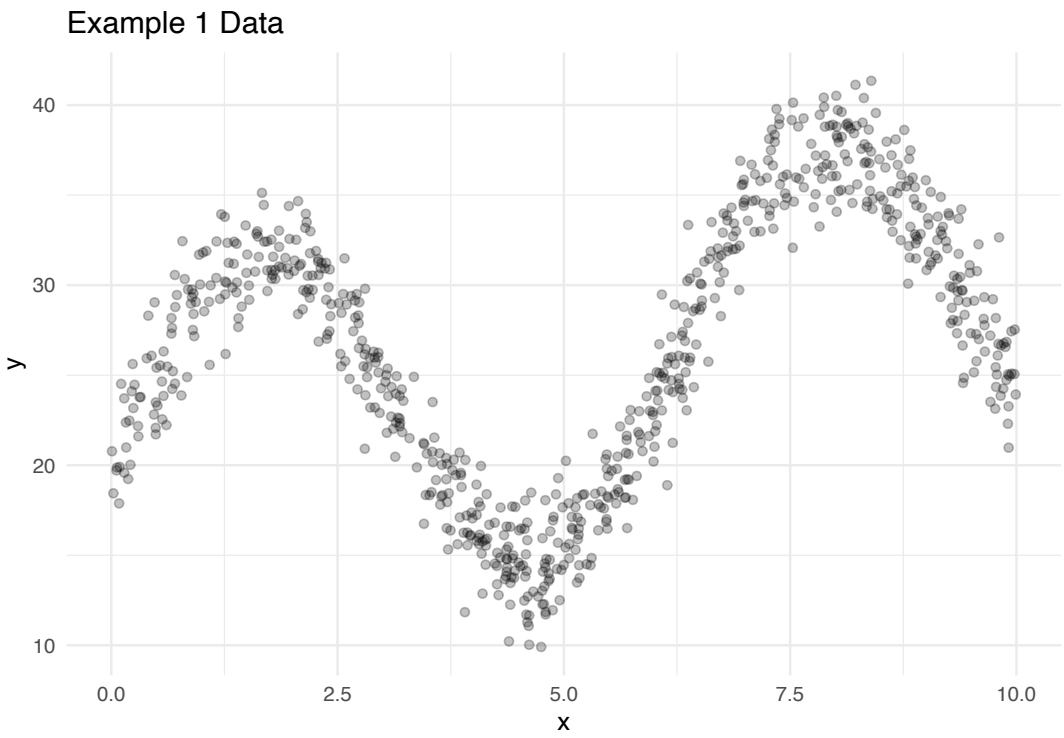
```
set.seed(20201007)

# create a split object
data1_split <- initial_split(data = data1, prop = 0.75)

# create the training and testing data
data1_train <- training(x = data1_split)
data1_test  <- testing(x = data1_split)
```

Step 2. EDA

```
# visualize the data
data1_train %>%
  ggplot(aes(x = x, y = y)) +
  geom_point(alpha = 0.25) +
  labs(title = "Example 1 Data") +
  theme_minimal()
```



3. Estimate a Model

```
# create a knn model specification
knn_mod <-
  nearest_neighbor(neighbors = 5) %>%
  set_engine(engine = "kknn") %>%
  set_mode(mode = "regression")

# fit the knn model specification on the training data
knn_fit <- knn_mod %>%
  fit(formula = y ~ x, data = data1_train)
```

4. Evaluate a Model

```
# use the estimated model to predict values in the testing data
predictions <-
  bind_cols(
    data1_test,
    predict(object = knn_fit, new_data = data1_test)
  )

# calculate the rmse on the testing data
rmse(data = predictions, truth = y, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        2.20
```

How good is this RMSE? It is relatively small when compared to the range of y, which suggests that this model is accurate.

```
summary(data1_test$y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   9.979  20.741  26.985  26.376  31.479  42.581
```

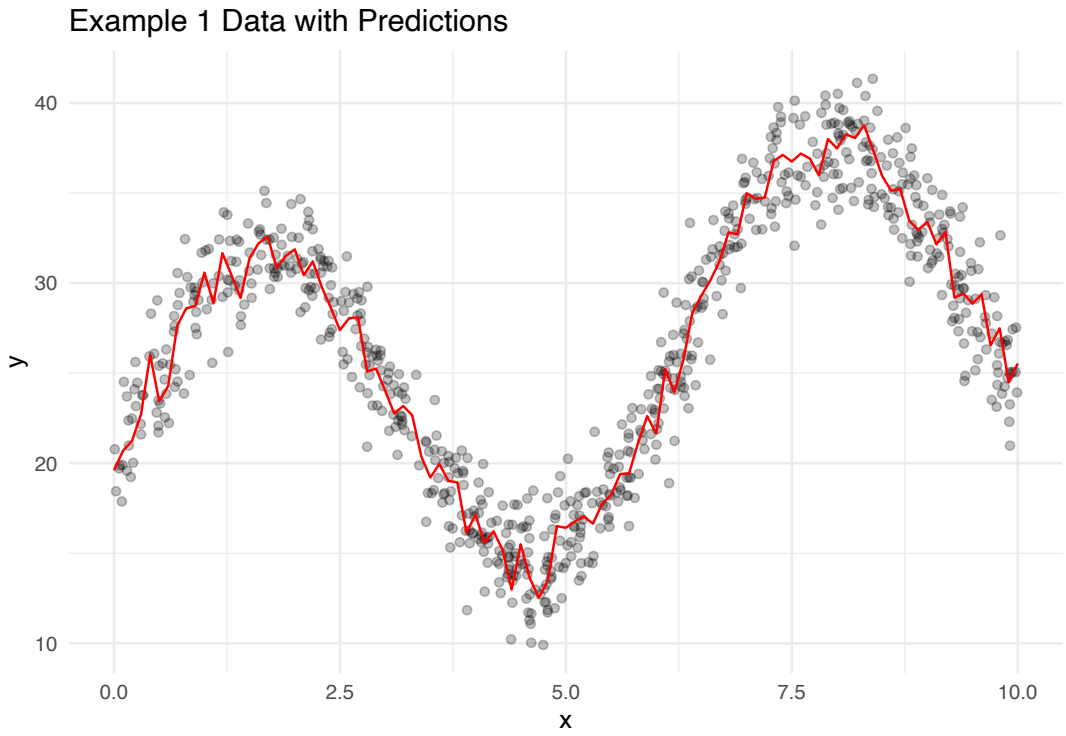
5. Make a New Prediction

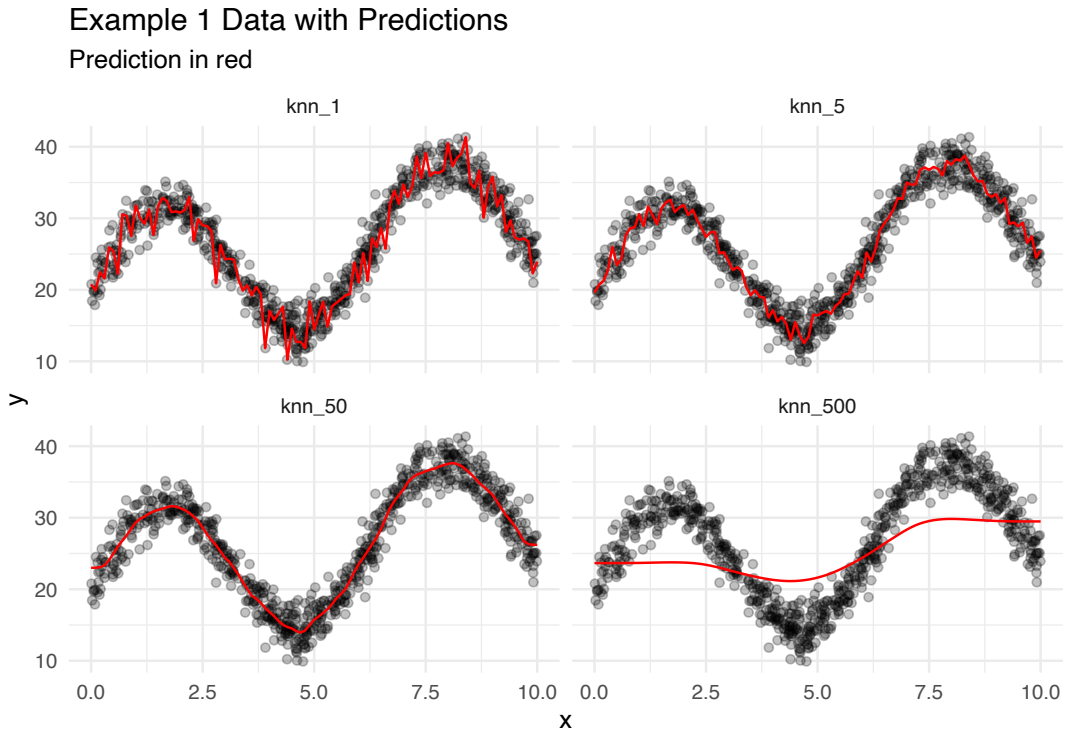
```
# make a novel prediction
predict(object = knn_fit, new_data = tibble(x = 1:5))
```

```
## # A tibble: 5 x 1
##   .pred
##   <dbl>
## 1  30.6
## 2  31.9
## 3  24.0
## 4  17.1
## 5  16.4
```

```
# create a grid of predictions
predictions_grid <- tibble(
  x = seq(0, 10, 0.1),
  predict(object = knn_fit, new_data = tibble(x = seq(0, 10, 0.1)))
)

# visualize the data
ggplot() +
  geom_point(data = data1_train, aes(x = x, y = y), alpha = 0.25) +
  geom_path(data = predictions_grid, aes(x = x, y = .pred), color = "red") +
  labs(title = "Example 1 Data with Predictions") +
  theme_minimal()
```


Example 1 Data with Predictions
40
30
20
10
y
0.0
2.5
5.0
7.5
10.0
x


Example 1 Data with Predictions
Prediction in red
knn_1
knn_5
knn_50
knn_500
40
30
20
10
y
0.0
2.5
5.0
7.5
10.0
x

Example 2

Example 2 uses simulated data with two predictors, x_1 and x_2, and one outcome variable y.

Step 0. Load the data

```
set.seed(20201005)

x1 <- runif(n = 1000, min = 0, max = 10)
x2 <- runif(n = 1000, min = 10, max = 20)

data2 <- bind_cols(
  x1 = x1,
  x2 = x2,
  y = 10 * sin(x1) + x2 + 20 + rnorm(n = length(x), mean = 0, sd = 2)
)
```

Step 1. Split the data into training and testing data

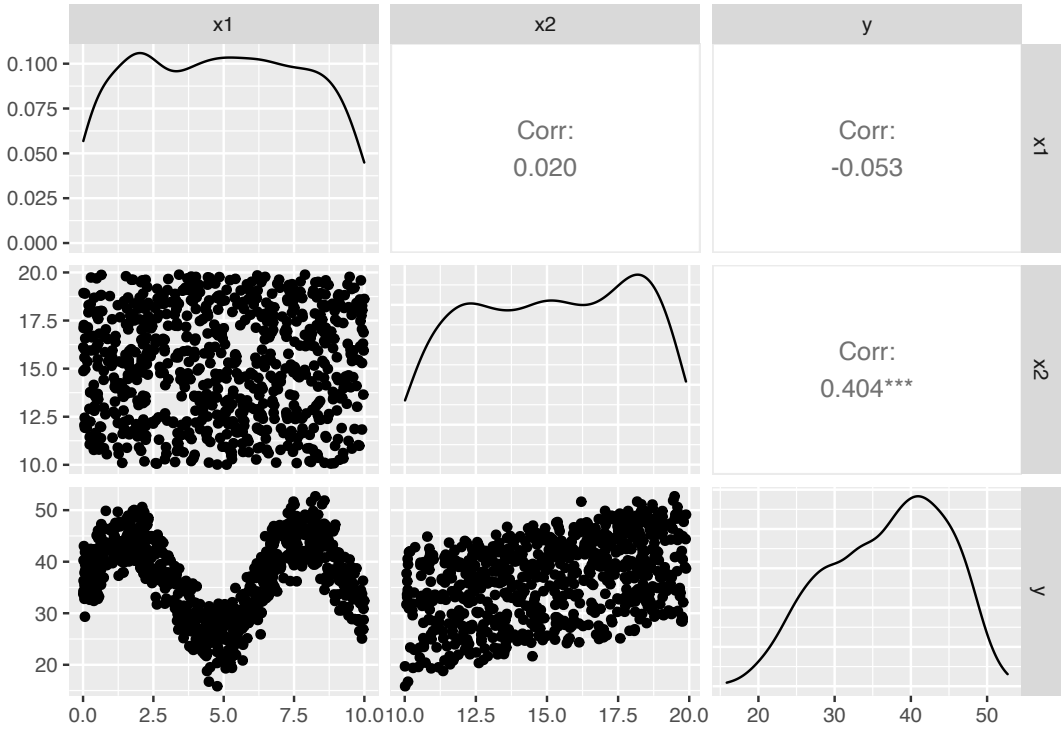
```
set.seed(20201007)

# create a split object
data2_split <- initial_split(data = data2, prop = 0.75)

# create the training and testing data
data2_train <- training(x = data2_split)
data2_test  <- testing(x = data2_split)
```

Step 2. EDA

```
GGally::ggpairs(data = data2_train)
```



Step 3. Create a Recipe

x_1 and x_2 are not on the same scale. It is important to *normalize* both variables before model estimation.

```
knn_recipe <-
  recipe(formula = y ~ ., data = data2_train) %>%
  step_normalize(x1, x2)
```

Step 4. Create resamples

This creates a data structure with 10 v-folds.

```
folds <- vfold_cv(data = data2_train, v = 10)

folds
```

```
## #  10-fold cross-validation
## # A tibble: 10 x 2
##    splits           id
##    <list>           <chr>
##  1 <split [675/75]> Fold01
##  2 <split [675/75]> Fold02
##  3 <split [675/75]> Fold03
```

```
##  4 <split [675/75]> Fold04
##  5 <split [675/75]> Fold05
##  6 <split [675/75]> Fold06
##  7 <split [675/75]> Fold07
##  8 <split [675/75]> Fold08
##  9 <split [675/75]> Fold09
## 10 <split [675/75]> Fold10
```

Step 5. Create a model

```
# create a knn model specification
knn_mod <-
  nearest_neighbor(neighbors = 5) %>%
  set_engine(engine = "kknn") %>%
  set_mode(mode = "regression")
```

Step 6. Create a workflow

```
knn_workflow <-
  workflow() %>%
  add_model(spec = knn_mod) %>%
  add_recipe(recipe = knn_recipe)
```

Step 7. Fit resamples

```
#set.seed(456)
knn_fit_rs <-
  knn_workflow %>%
  fit_resamples(resamples = folds)
```

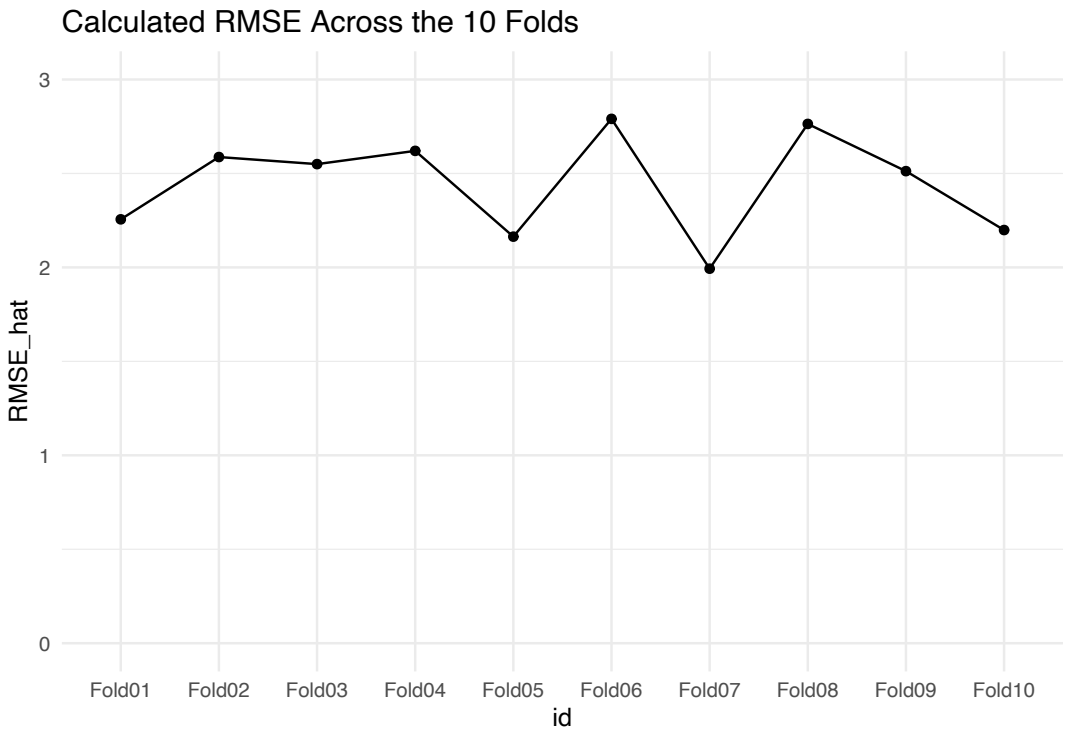
Step 8. Evaluate the model

```
collect_metrics(knn_fit_rs)
```

```
## # A tibble: 2 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   2.44     10 0.0861  Preprocessor1_Model1
## 2 rsq     standard   0.897    10 0.00656 Preprocessor1_Model1
```

```
collect_metrics(knn_fit_rs, summarize = FALSE) %>%
  filter(.metric == "rmse") %>%
  ggplot(aes(id, .estimate, group = .estimator)) +
  geom_line() +
  geom_point() +
  scale_y_continuous(limits = c(0, 3)) +
```

```
  labs(title = "Calculated RMSE Across the 10 Folds",
       y = "RMSE_hat") +
  theme_minimal()
```



From here, it is possible to estimate the model on all of the training data, get one estimate of $\hat{RMSE}$ using the testing data, and then make predictions on new data.

Example 3

Example 3 is the same as example 2 but adds hyperparameter tuning to determine the optimal k.

Step 1. Split the data into training and testing data (repeat)

Step 2. EDA (repeat)

Step 3. Create a Recipe (repeat)

Step 4. Create resamples

Step 5. Create a model

Instead of specifying `neighbors = 5`, use `tune()` as a placeholder so the different values of k can be passed to the model during hyperparameter tuning.

```
# create a knn model specification
knn_mod <-
  nearest_neighbor(neighbors = tune()) %>%
  set_engine(engine = "kknn") %>%
  set_mode(mode = "regression")
```

Step 6. Create a workflow

```
knn_workflow <-
  workflow() %>%
  add_model(spec = knn_mod) %>%
  add_recipe(recipe = knn_recipe)
```

Step 7. Create a tuning grid

Create a tuning grid for values of hyperparameters ranging from 1 to 15 by twos. It is common to use odd values of k to limit ties.

```
knn_grid <- tibble(neighbors = seq(from = 1, to = 15, by = 2))

knn_grid
```

```
## # A tibble: 8 x 1
##   neighbors
##       <dbl>
## 1         1
## 2         3
## 3         5
## 4         7
## 5         9
## 6        11
## 7        13
## 8        15
```

Step 8. Estimate models with resampling for each row in the tuning grid

```
knn_res <-
  knn_workflow %>%
  tune_grid(resamples = folds,
            grid = knn_grid,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(rmse))
```

Evaluate the model

```
# look at summaries
knn_res %>%
  collect_metrics()
```

```
## # A tibble: 8 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1         1 rmse    standard    3.04    10  0.0924 Preprocessor1_Model1
## 2         3 rmse    standard    2.60    10  0.0946 Preprocessor1_Model2
## 3         5 rmse    standard    2.44    10  0.0861 Preprocessor1_Model3
## 4         7 rmse    standard    2.39    10  0.0821 Preprocessor1_Model4
## 5         9 rmse    standard    2.37    10  0.0806 Preprocessor1_Model5
## 6        11 rmse    standard    2.36    10  0.0808 Preprocessor1_Model6
## 7        13 rmse    standard    2.36    10  0.0831 Preprocessor1_Model7
## 8        15 rmse    standard    2.37    10  0.0846 Preprocessor1_Model8
```

```
# show the best models
knn_res %>%
  show_best()
```

```
## # A tibble: 5 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1        13 rmse    standard    2.36    10  0.0831 Preprocessor1_Model7
## 2        11 rmse    standard    2.36    10  0.0808 Preprocessor1_Model6
## 3         9 rmse    standard    2.37    10  0.0806 Preprocessor1_Model5
## 4        15 rmse    standard    2.37    10  0.0846 Preprocessor1_Model8
## 5         7 rmse    standard    2.39    10  0.0821 Preprocessor1_Model4
```

```
# pick the best model
knn_res %>%
  select_best()
```

```
## # A tibble: 1 x 2
##   neighbors .config
##       <dbl> <chr>
## 1        13 Preprocessor1_Model7
```

```
# look at predicted values from the resamples
knn_res %>%
  collect_predictions()
```

```
## # A tibble: 6,000 x 6
##    id     .pred  .row neighbors     y .config
##    <chr>  <dbl> <int>     <dbl> <dbl> <chr>
##  1 Fold01  42.4     1         1  37.3 Preprocessor1_Model1
##  2 Fold01  15.8     2         1  16.7 Preprocessor1_Model1
##  3 Fold01  47.0    17         1  45.4 Preprocessor1_Model1
##  4 Fold01  35.1    18         1  33.1 Preprocessor1_Model1
##  5 Fold01  46.7    63         1  45.3 Preprocessor1_Model1
##  6 Fold01  38.7    74         1  41.2 Preprocessor1_Model1
##  7 Fold01  38.4    81         1  36.6 Preprocessor1_Model1
##  8 Fold01  47.3    99         1  46.1 Preprocessor1_Model1
##  9 Fold01  31.7   100         1  28.1 Preprocessor1_Model1
## 10 Fold01  41.4   121         1  36.7 Preprocessor1_Model1
## # ... with 5,990 more rows
```

From here, it is possible to estimate the model with the optimal k on all of the training data, get one estimate of $\hat{RMSE}$ using the testing data, and then make predictions on new data.

Resources

- An Introduction to Statistical Learning
- Elements of Statistical Learning
- Get Started with Tidymodels
- Gentle Introduction to the Bias-Variance Trade-Off in Machine Learning
- A tutorial on tidy cross-validation with R
- A great collection of other resources

Appendix A: Common predictive models

- Linear Regression
- Ridge Regression
- Lasso Regression
- Regression Splines
- Smoothing Splines
- Support Vector Machines
- KNN
- Decision Trees/Classification and Regression Trees (CART)
- Random Forests
- Boosted Regression Trees/Gradient Boosting
- Neural Networks

Appendix B: Examples from Public Policy

Targeting Interventions

- Predictive Modeling for Public Health: Preventing Childhood Lead Poisoning

Imputation and "Amplified Asking"

- Predicting Poverty and Wealth from Mobile Phone Metadata link 1 link 2

Forecasting and nowcasting

- Google Flu Trends (warning: this ended poorly)