KNN Regression

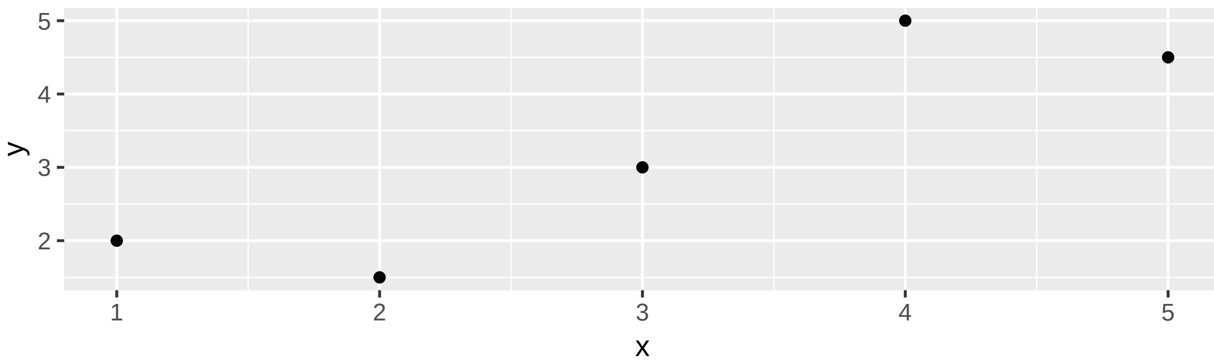
Most examples/code here are adapted from examples discussed at https://daviddalpiaz.github.io/r4sl/knn-reg.html (this is a useful companion to our text).

Suppose we have the following data:

```
example_data
```

```
##   x   y
## 1 1 2.0
## 2 2 1.5
## 3 3 3.0
## 4 4 5.0
## 5 5 4.5
```

```
ggplot(data = example_data, mapping = aes(x = x, y = y)) +
  geom_point()
```



2 Basic Approaches to Estimating $f(x)$:

1. Specify a model like $f(x) = \beta_0 + \beta_1 x$. Estimate the *parameters* β_0 and β_1.
2. Local Approach: $f(x_0)$ should look like the data in a neighborhood of x_0

Models that don't specify a specific parametric form for f are often called *nonparametric*.

(One possible) formal definition of a nonparametric model: The number of model parameters is an unbounded function of the sample size.

K Nearest Neighbors

$$\hat{f}(x_0) = \frac{1}{K} \sum_{i \in N_0^{(k)}} y_i$$

Here $N_0^{(k)}$ is a set of indices for the k observations that have values x_i nearest to the test point x_0.

Example:

- Suppose $x_0 = 3.75$
- Set $k = 3$
- In our training data set, the k nearest neighbors are $i = 3$, $i = 4$, and $i = 5$. $N_0^{(3)} = \{3, 4, 5\}$
- Our predicted value at x_0 is the average of the responses for those three observations:

```
# we will never actually compute f_hat like this in practice!
f_hat <- mean(example_data$y[c(3, 4, 5)])
f_hat
```

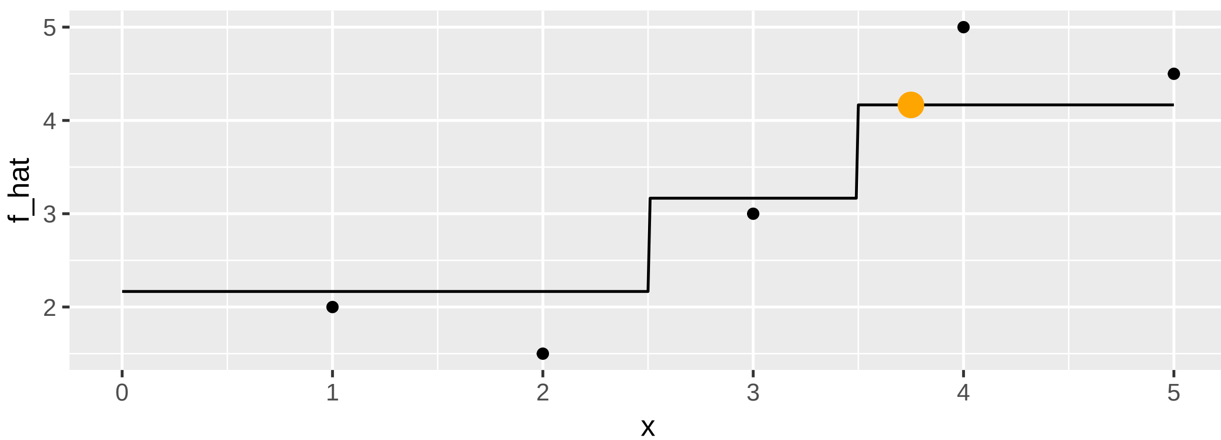
```
## [1] 4.166667
```

The `knn.reg` function in the `FNN` package will do these calculations for us if we provide training data and a vector of test values for x:

```r
library(FNN)
f_hat_data <- data.frame(
  x = seq(from = 0, to = 5, by = 0.01)
)
head(f_hat_data)
```

```
##      x
## 1 0.00
## 2 0.01
## 3 0.02
## 4 0.03
## 5 0.04
## 6 0.05
```

```r
knn_predictions <- knn.reg(
  train = example_data %>% select(x), # a data frame with train set explanatory variables
  test = f_hat_data, # a data frame with test set explanatory variables
  y = example_data %>% select(y), # a data frame with train set response variable
  k = 3) # k for k nearest neighbors
f_hat_data <- f_hat_data %>%
  mutate(
    f_hat = knn_predictions$pred
  )

ggplot() +
  geom_line(data = f_hat_data, mapping = aes(x = x, y = f_hat)) +
  geom_point(data = example_data, mapping = aes(x = x, y = y)) +
  geom_point(mapping = aes(x = 3.75, y = 4.166667), color = "orange", size = 4)
```

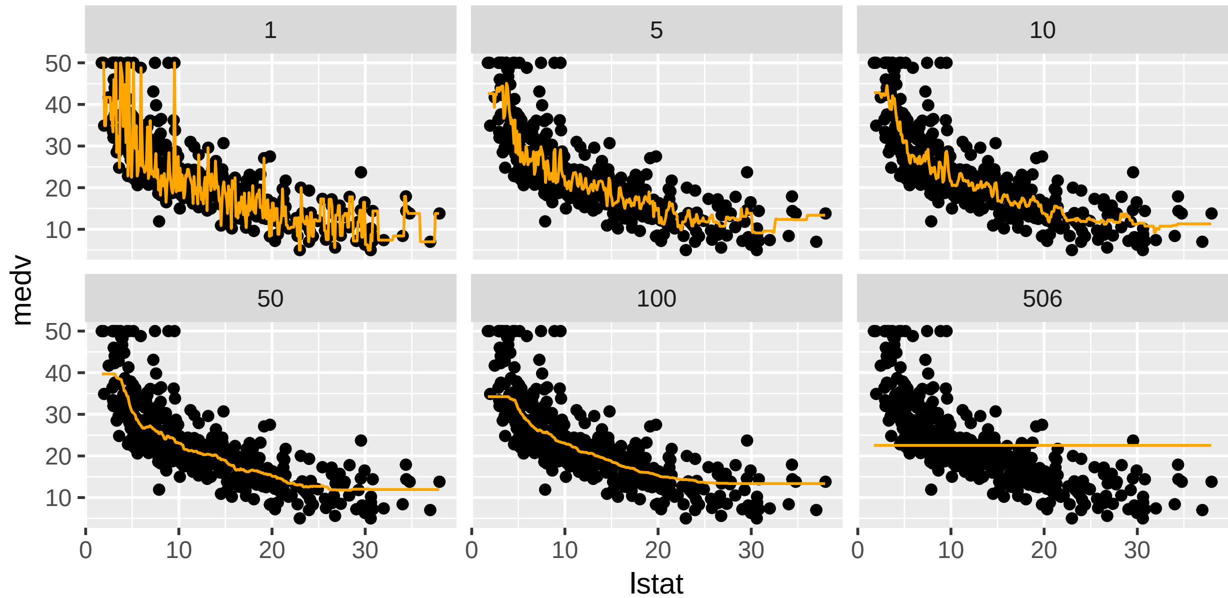


Flexibility is determined by k

```r
library(MASS) # for Boston data - note it masks the select function from dplyr package!
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

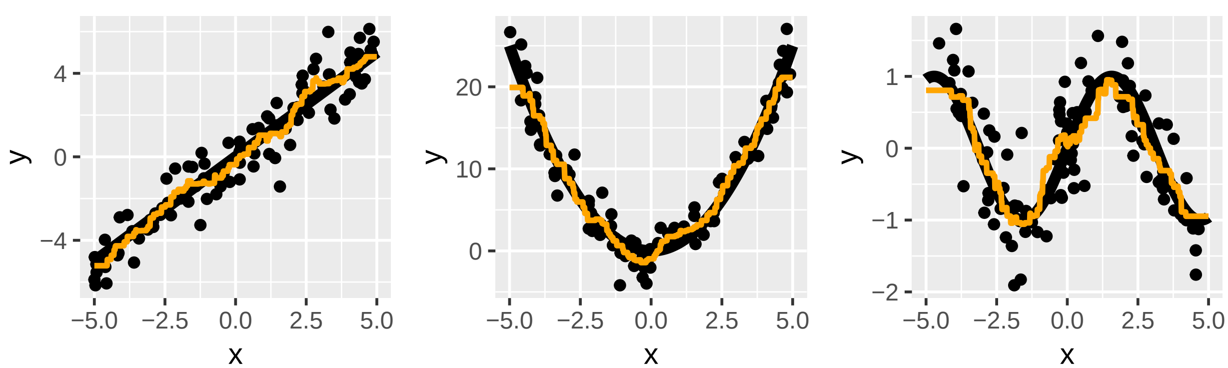
```r
set.seed(42)

lstat_grid <- seq(from = min(Boston$lstat), to = max(Boston$lstat), by = 0.1)
test_data <- data.frame(
  lstat = lstat_grid
)

get_test_preds_df_k <- function(k) {
  data.frame(
    lstat = lstat_grid,
    medv_hat = knn.reg(
      train = Boston %>% dplyr::select(lstat),
      test = test_data,
      y = Boston %>% dplyr::select(medv),
      k = k)$pred,
    k = k
  )
}

test_boston <- map_dfr(
  c(1, 5, 10, 50, 100, nrow(Boston)),
  get_test_preds_df_k
)

ggplot() +
  geom_point(data = Boston, mapping = aes(x = lstat, y = medv)) +
  geom_line(data = test_boston, mapping = aes(x = lstat, y = medv_hat), color = "orange") +
  facet_wrap( ~ k)
```



For now we will choose k by looking at plots; see Chapter 5 in about 2-3 weeks for more formal approaches.

KNN Automatically Adjusts to Different Functional Forms

```
line_reg_fun <- function(x) {
  x
}

quad_reg_fun <- function(x) {
  x ^ 2
}

sine_reg_fun <- function(x) {
  sin(x)
}

get_plot_knn_vs_true <- function(reg_fun, sample_size = 100, noise_sd = 1, k = 10) {
  x <- runif(n = sample_size, min = -5, max = 5)
  y <- rnorm(n = sample_size, mean = reg_fun(x), sd = noise_sd)
  train_data <- data.frame(x = x, y = y)

  test_x_grid <- data.frame(x = seq(-5, 5, by = 0.01))

  test_results <- test_x_grid %>%
    mutate(
      y_true_mean = reg_fun(x),
      y_knn = FNN::knn.reg(
        train = train_data %>% dplyr::select(x),
        test = test_x_grid,
        y = train_data %>% dplyr::select(y),
        k = 10)$pred
    )

  plot_object <- ggplot(mapping = aes(x = x)) +
    geom_point(data = train_data, mapping = aes(y = y)) +
    geom_line(data = test_results, mapping = aes(y = y_true_mean), size = 2) +
    geom_line(data = test_results, mapping = aes(y = y_knn), color = "orange", size = 1)

  return(plot_object)
}

set.seed(42)
line_plot <- get_plot_knn_vs_true(line_reg_fun, sample_size = 100, noise_sd = 1, k = 10)
quad_plot <- get_plot_knn_vs_true(quad_reg_fun, sample_size = 100, noise_sd = 2, k = 10)
sine_plot <- get_plot_knn_vs_true(sine_reg_fun, sample_size = 100, noise_sd = 0.5, k = 10)

grid.arrange(line_plot, quad_plot, sine_plot, ncol = 3)
```



If $p > 1$, KNN Performance Improves if we Scale the Explanatory Variables

- Let's measure performance by the square root of test set MSE (RMSE in test set).

```r
set.seed(42)
train_inds <- sample(1:nrow(Boston), size = 250)
train_boston <- Boston[train_inds, ]
test_boston <- Boston[-train_inds, ]

X_train_boston <- train_boston %>% dplyr::select(-medv)
X_test_boston = test_boston %>% dplyr::select(-medv)
y_train_boston = train_boston %>% dplyr::select(medv)
y_test_boston = test_boston %>% dplyr::select(medv)

scale_values <- X_train_boston %>%
  summarize_all(sd)

scaled_pred = knn.reg(
  train = scale(X_train_boston, scale = scale_values),
  test = scale(X_test_boston, scale = scale_values),
  y = y_train_boston,
  k = 10)$pred

unscaled_pred = knn.reg(
  train = X_train_boston,
  test = X_test_boston,
  y = y_train_boston,
  k = 10)$pred

# test rmse
rmse <- function(actual, predicted) {
  sqrt(mean((actual - predicted)^2))
}
rmse(actual = y_test_boston$medv, predicted = scaled_pred) # with scaling
```

```
## [1] 5.690695
```

```r
rmse(actual = y_test_boston$medv, predicted = unscaled_pred) # without scaling
```

```
## [1] 7.540342
```

Curse of Dimensionality: KNN Performance Degrades Quickly as p Increases

- Degradation in performance affects all methods, but affects non-parametric methods more
  - Parametric models assume a restricted parametric form for f and are trying to learn only a few parameters
  - Non-parametric methods are trying to learn the functional form. This is more difficult in higher dimensions

```
sim_knn_data <- function(n_obs = 50) {
  x1 <- seq(0, 10, length.out = n_obs)
  x2 <- runif(n = n_obs, min = 0, max = 10)
  x3 <- runif(n = n_obs, min = 0, max = 10)
  x4 <- runif(n = n_obs, min = 0, max = 10)
  x5 <- runif(n = n_obs, min = 0, max = 10)
  y <- x1 ^ 2 + rnorm(n = n_obs)
  data.frame(y, x1, x2, x3, x4, x5)
}

set.seed(42)
knn_data_train <- sim_knn_data()
knn_data_test <- sim_knn_data()

# define helper function for getting knn.reg predictions
# note: this function is highly specific to this situation and data set
get_test_rmse <- function(p = 1, k = 5) {
  x_vars <- paste0("x", seq_len(p))

  x_train <- knn_data_train %>%
    dplyr::select(x_vars)
  x_test <- knn_data_test %>%
    dplyr::select(x_vars)
  scale_values <- x_train %>%
    summarize_all(sd)
  x_train <- scale(x_train, scale = scale_values) %>%
    as.data.frame()
  x_test <- scale(x_test, scale = scale_values) %>%
    as.data.frame()

  y_train <- knn_data_train %>%
    dplyr::select(y)
  y_test <- knn_data_test %>%
    dplyr::select(y)

  pred_knn <- FNN::knn.reg(
    train = x_train,
    test = x_test,
    y = y_train,
    k = k)$pred

  lm_fit <- lm(y ~ ., data = cbind(y_train, x_train))
  pred_lm <- predict(lm_fit, newdata = x_test)

  data.frame(
    p = p,
    knn = rmse(actual = y_test$y, predicted = pred_knn),
    lm = rmse(actual = y_test$y, predicted = pred_lm)
  )
}

cod_results <- map_dfr(
  seq_len(5),
```

```
  get_test_rmse
)

ggplot(data = cod_results, mapping = aes(x = p)) +
  geom_line(mapping = aes(y = knn), color = "orange") +
  geom_line(mapping = aes(y = lm), color = "cornflowerblue") +
  ylab("RMSE")
```