Boston Housing: KNN; Bias-Variance Trade-Off; Cross Validation

Chicago Booth ML Team

OVERVIEW

This R Markdown script uses the ***Boston Housing*** data set to illustrate the following:

- The k**-Nearest Neighbors** (**KNN**) algorithm;
- The **Bias-Variance Trade-Off**; and
- The use of **Cross Validation** to estimate Out-of-Sample (OOS) prediction error and determine optimal hyper-parameters, in this case the number of nearest neighbors k.

first, some boring logistics…

Let's first load some necessary R packages and helper functions and set the random number generator's seed:

```
# load CRAN libraries from CRAN packages
library(data.table)
library(ggplot2)
library(kknn)
# load modules from the common HelpR repo
helpr_repo_raw_url <- 'https://raw.githubusercontent.com/ChicagoBoothML/HelpR/master'
source(file.path(helpr_repo_raw_url, 'docv.R'))   # this has docvknn used below

# set randomizer's seed
set.seed(99)   # Gretzky was #99
```

Boston Housing Data Set

Let's then look at the **Boston Housing** data set:

```
# download data and read data into data.table format
boston_housing <- fread(
  'https://raw.githubusercontent.com/ChicagoBoothML/DATA___BostonHousing/master/BostonHousing.csv')
# count number of samples
nb_samples <- nrow(boston_housing)
# sort data set by increasing lstat
setkey(boston_housing, lstat)
boston_housing
```

```
##          crim zn indus chas   nox    rm   age    dis rad tax ptratio
##   1:  1.46336  0 19.58    0 0.605 7.489  90.8 1.9709   5 403    14.7
##   2:  1.83377  0 19.58    1 0.605 7.802  98.2 2.0407   5 403    14.7
##   3:  0.03359 75  2.95    0 0.428 7.024  15.8 5.4011   3 252    18.3
##   4:  0.57529  0  6.20    0 0.507 8.337  73.3 3.8384   8 307    17.4
##   5:  0.08664 45  3.44    0 0.437 7.178  26.3 6.4798   5 398    15.2
##  ---
## 502: 18.81100  0 18.10    0 0.597 4.628 100.0 1.5539  24 666    20.2
## 503:  1.62864  0 21.89    0 0.624 5.019 100.0 1.4394   4 437    21.2
## 504: 11.10810  0 18.10    0 0.668 4.906 100.0 1.1742  24 666    20.2
## 505: 45.74610  0 18.10    0 0.693 4.519 100.0 1.6582  24 666    20.2
## 506: 18.49820  0 18.10    0 0.668 4.138 100.0 1.1370  24 666    20.2
```

```
##       black lstat medv
##   1: 374.43  1.73 50.0
##   2: 389.61  1.92 50.0
##   3: 395.62  1.98 34.9
##   4: 385.91  2.47 41.7
##   5: 390.49  2.87 36.4
##  ---
## 502:  28.79 34.37 17.9
## 503: 396.90 34.41 14.4
## 504: 396.90 34.77 13.8
## 505:  88.27 36.98  7.0
## 506: 396.90 37.97 13.8
```

This data set has **506** samples.

We'll focus on **using the *lstat* variable to predict the *medv* variable**. Let's first plot them against each other:

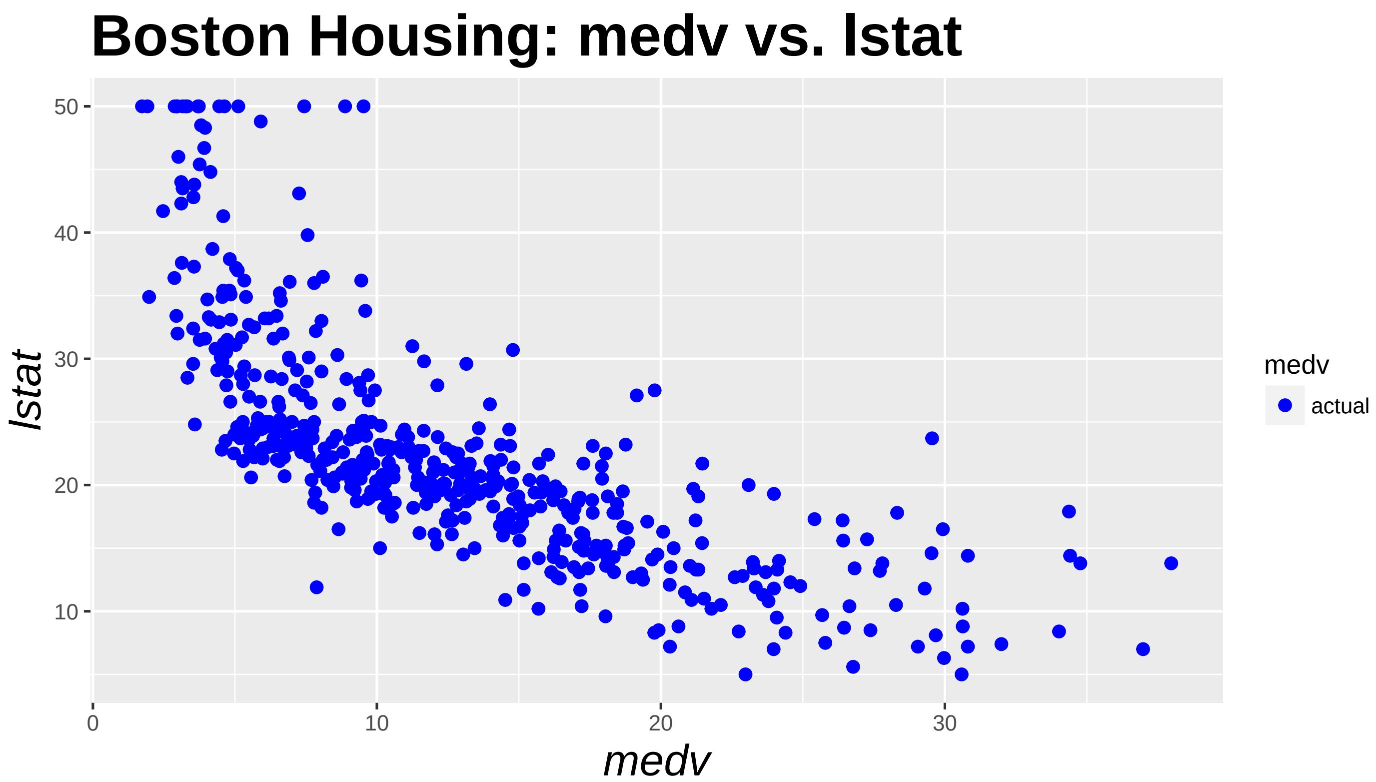
```r
plot_boston_housing_data <- function(boston_housing_data,
                                     title='Boston Housing: medv vs. lstat',
                                     plot_predicted=TRUE) {
  g <- ggplot(boston_housing_data) +
    geom_point(aes(x=lstat, y=medv, color='actual'), size=2) +
    ggtitle(title) +
    xlab('medv') + ylab('lstat')

  if (plot_predicted) {
    g <- g +
      geom_line(aes(x=lstat, y=predicted_medv, color='predicted'), size=0.6) +
      scale_colour_manual(name='medv',
                          values=c(actual='blue', predicted='darkorange'))
  } else {
    g <- g +
      scale_colour_manual(name='medv',
                          values=c(actual='blue'))
  }

  g <- g +
    theme(plot.title=element_text(face='bold', size=24),
        axis.title=element_text(face='italic', size=18))

  g
}

plot_boston_housing_data(boston_housing, plot_predicted=FALSE)
```



k-Nearest Neighbors algorithm and Bias-Variance Trade-Off

```
try_k <- 5
```

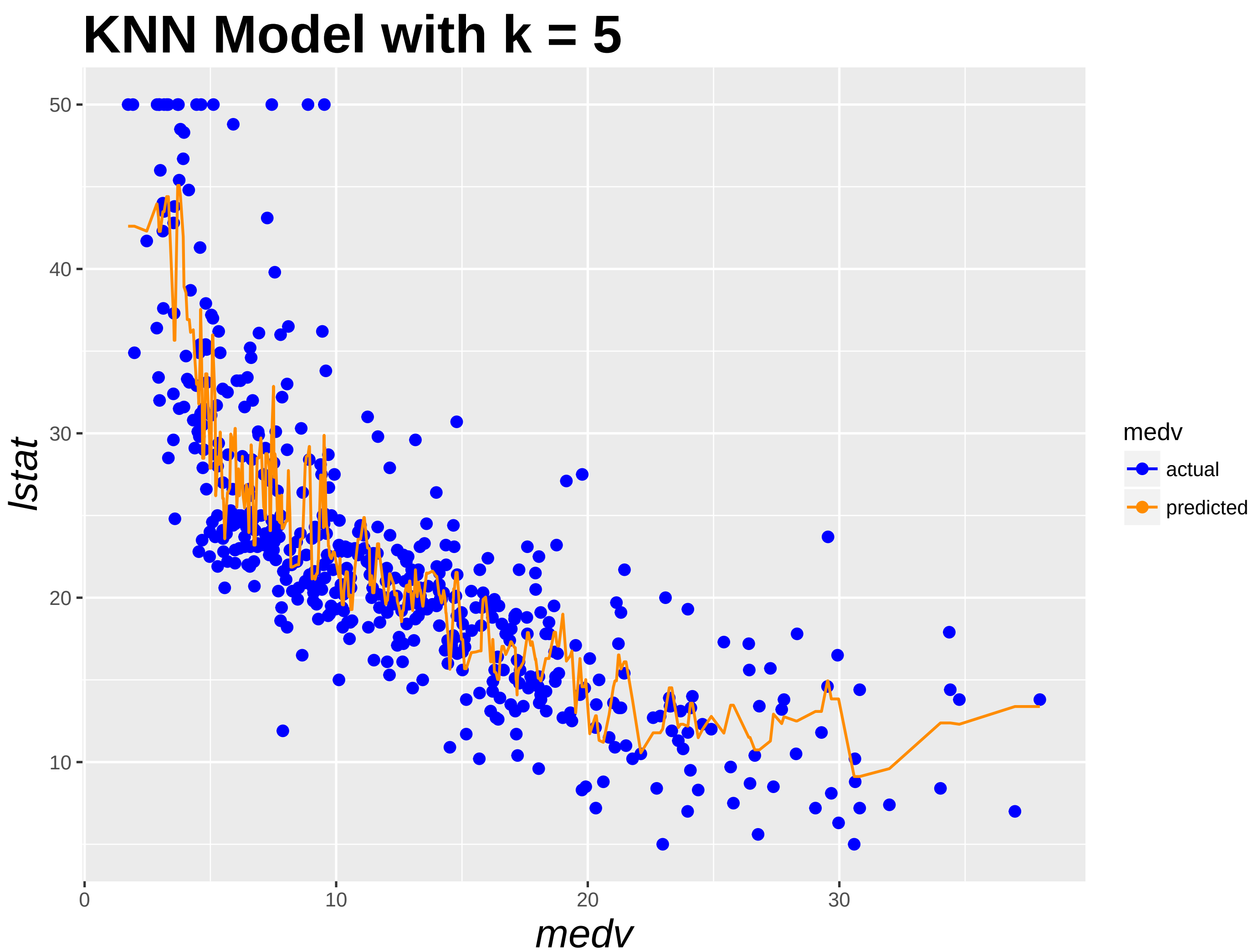
Let's now try fitting a KNN predictor, with $k = 5$, of *medv* from *lstat*, using the entire 506 samples:

```
knn_model <- kknn(medv ~ lstat,
                  train=boston_housing, test=boston_housing[, .(lstat)],
                  k=try_k, kernel='rectangular')
boston_housing[, predicted_medv := knn_model$fitted.values]
```

```
##          crim zn indus chas   nox    rm   age    dis rad tax ptratio
##   1:  1.46336  0 19.58    0 0.605 7.489  90.8 1.9709   5 403    14.7
##   2:  1.83377  0 19.58    1 0.605 7.802  98.2 2.0407   5 403    14.7
##   3:  0.03359 75  2.95    0 0.428 7.024  15.8 5.4011   3 252    18.3
##   4:  0.57529  0  6.20    0 0.507 8.337  73.3 3.8384   8 307    17.4
##   5:  0.08664 45  3.44    0 0.437 7.178  26.3 6.4798   5 398    15.2
##  ---
## 502: 18.81100  0 18.10    0 0.597 4.628 100.0 1.5539  24 666    20.2
## 503:  1.62864  0 21.89    0 0.624 5.019 100.0 1.4394   4 437    21.2
## 504: 11.10810  0 18.10    0 0.668 4.906 100.0 1.1742  24 666    20.2
## 505: 45.74610  0 18.10    0 0.693 4.519 100.0 1.6582  24 666    20.2
## 506: 18.49820  0 18.10    0 0.668 4.138 100.0 1.1370  24 666    20.2
##       black lstat medv predicted_medv
##   1: 374.43  1.73 50.0          42.60
##   2: 389.61  1.92 50.0          42.60
##   3: 395.62  1.98 34.9          42.60
##   4: 385.91  2.47 41.7          42.30
##   5: 390.49  2.87 36.4          43.96
##  ---
## 502:  28.79 34.37 17.9          12.38
## 503: 396.90 34.41 14.4          12.38
```

```
## 504: 396.90 34.77 13.8          12.30
## 505:  88.27 36.98  7.0          13.38
## 506: 396.90 37.97 13.8          13.38
```

```
plot_boston_housing_data(boston_housing, title=paste('KNN Model with k =', try_k))
```



With $k = 5$ – a small number of nearest neighbors – we have a very "squiggly" predictor, which **fits the training data well** but is **over-sensitive to small changes** in the *lstat* variable. We call this a **LOW-BIAS**, **HIGH-VARIANCE** predictor. We don't like it.

```
try_k <- 200
```

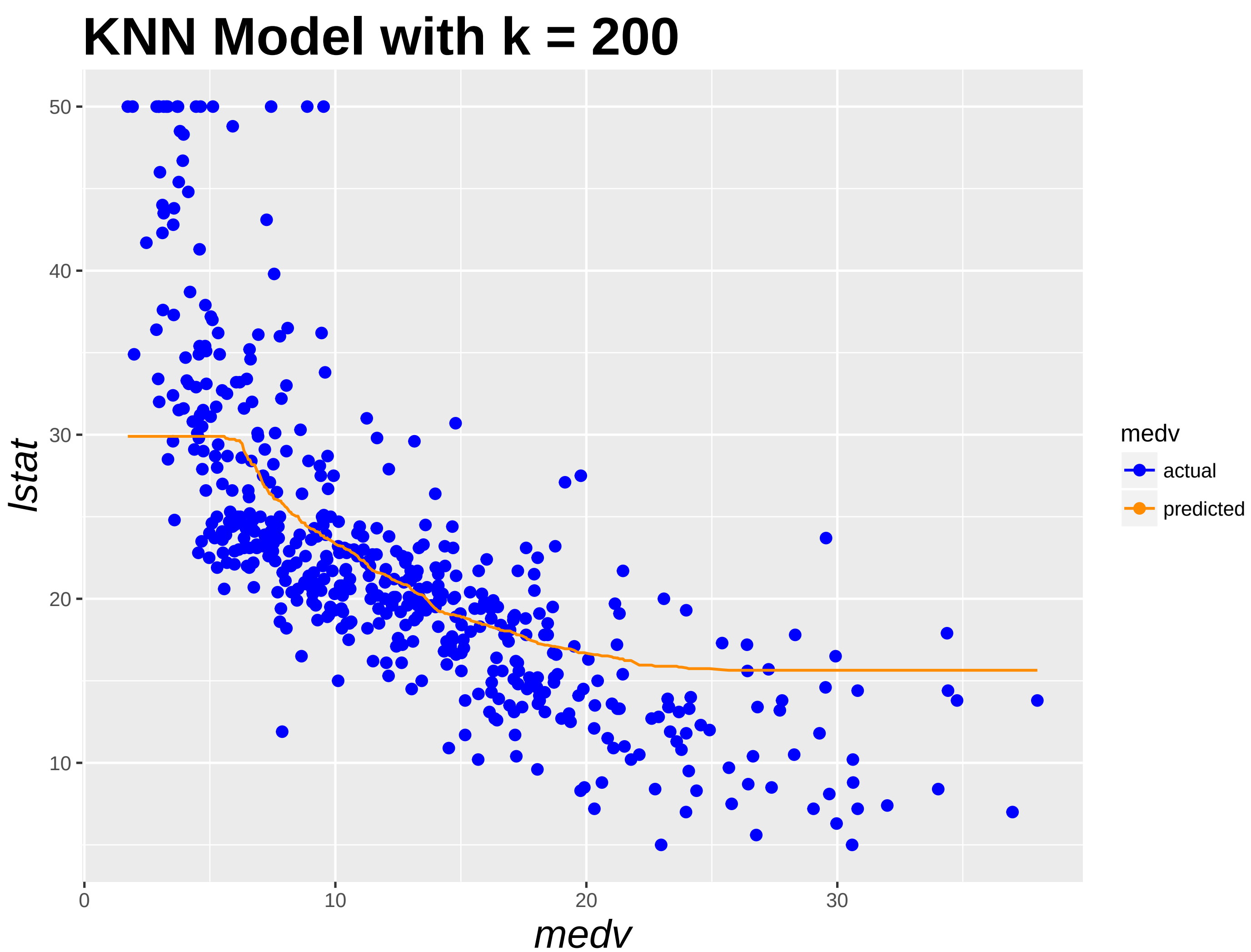
Now, with, say, $k = 200$, we have the following:

```
knn_model <- kknn(medv ~ lstat,
                  train=boston_housing, test=boston_housing[, .(lstat)],
                  k=try_k, kernel='rectangular')
boston_housing[, predicted_medv := knn_model$fitted.values]
```

```
##          crim zn indus chas   nox    rm   age    dis rad tax ptratio
##   1:  1.46336  0 19.58    0 0.605 7.489  90.8 1.9709   5 403    14.7
##   2:  1.83377  0 19.58    1 0.605 7.802  98.2 2.0407   5 403    14.7
##   3:  0.03359 75  2.95    0 0.428 7.024  15.8 5.4011   3 252    18.3
##   4:  0.57529  0  6.20    0 0.507 8.337  73.3 3.8384   8 307    17.4
##   5:  0.08664 45  3.44    0 0.437 7.178  26.3 6.4798   5 398    15.2
##  ---
## 502: 18.81100  0 18.10    0 0.597 4.628 100.0 1.5539  24 666    20.2
```

```
## 503:  1.62864  0 21.89    0 0.624 5.019 100.0 1.4394   4 437    21.2
## 504: 11.10810  0 18.10    0 0.668 4.906 100.0 1.1742  24 666    20.2
## 505: 45.74610  0 18.10    0 0.693 4.519 100.0 1.6582  24 666    20.2
## 506: 18.49820  0 18.10    0 0.668 4.138 100.0 1.1370  24 666    20.2
##       black lstat medv predicted_medv
##   1: 374.43  1.73 50.0        29.9000
##   2: 389.61  1.92 50.0        29.9000
##   3: 395.62  1.98 34.9        29.9000
##   4: 385.91  2.47 41.7        29.9000
##   5: 390.49  2.87 36.4        29.9000
##  ---
## 502:  28.79 34.37 17.9        15.6425
## 503: 396.90 34.41 14.4        15.6425
## 504: 396.90 34.77 13.8        15.6425
## 505:  88.27 36.98  7.0        15.6425
## 506: 396.90 37.97 13.8        15.6425
```

```
plot_boston_housing_data(boston_housing, title=paste('KNN Model with k =', try_k))
```



Meh..., we're not exactly jumping around with joy with this one, either. The predictor line is **not over-sensitive**, but **too "smooth" and too simple**, **not responding sufficiently to significant changes** in *lstat*. We call this a **HIGH-BIAS, LOW-VARIANCE** predictor.

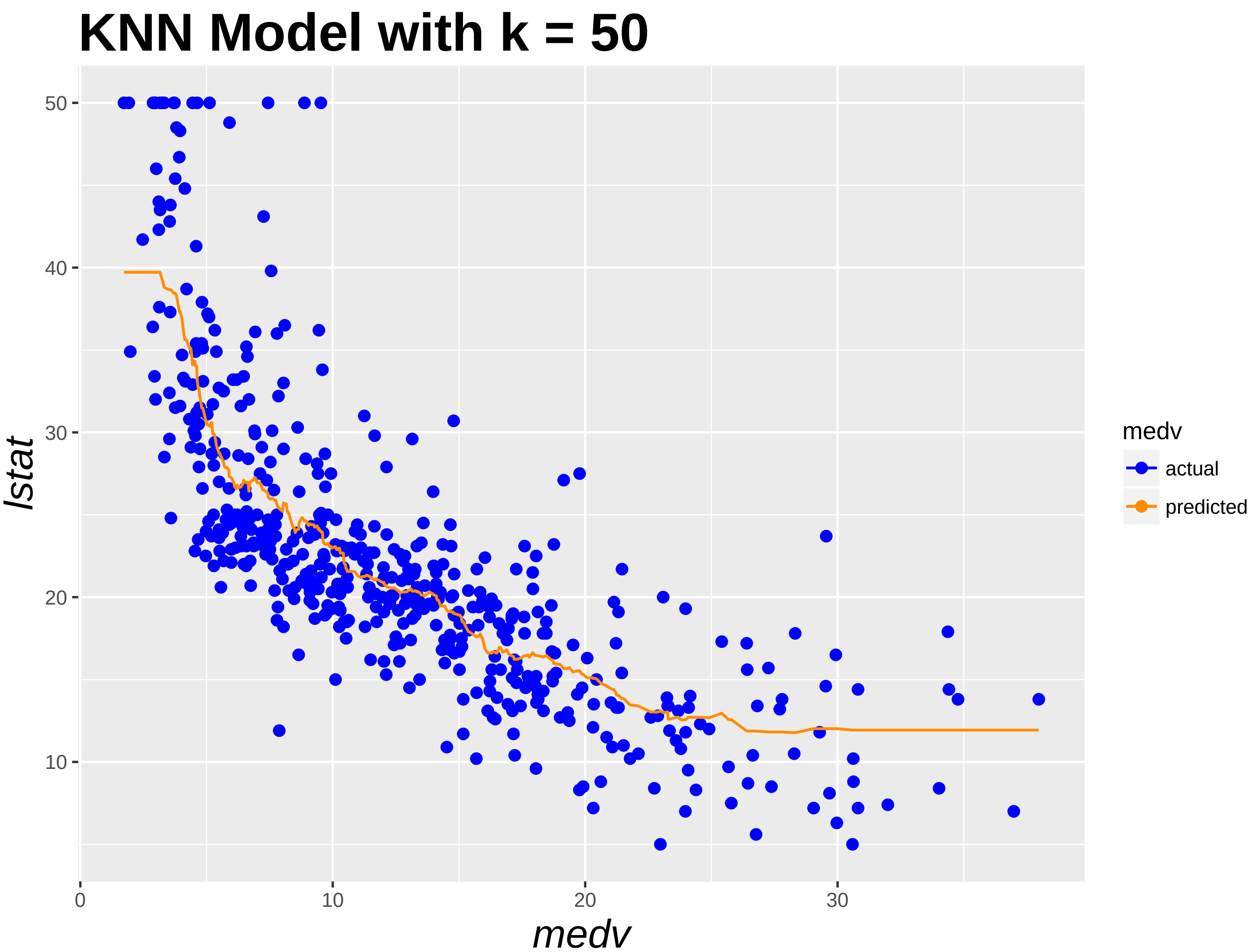
```
try_k <- 50
```

Let's try something in between, say, $k = 50$, to see if we have any better luck:

```r
knn_model <- kknn(medv ~ lstat,
                  train=boston_housing, test=boston_housing[, .(lstat)],
                  k=try_k, kernel='rectangular')
boston_housing[, predicted_medv := knn_model$fitted.values]
```

```
##          crim zn indus chas   nox    rm   age    dis rad tax ptratio
##   1:  1.46336  0 19.58    0 0.605 7.489  90.8 1.9709   5 403    14.7
##   2:  1.83377  0 19.58    1 0.605 7.802  98.2 2.0407   5 403    14.7
##   3:  0.03359 75  2.95    0 0.428 7.024  15.8 5.4011   3 252    18.3
##   4:  0.57529  0  6.20    0 0.507 8.337  73.3 3.8384   8 307    17.4
##   5:  0.08664 45  3.44    0 0.437 7.178  26.3 6.4798   5 398    15.2
##  ---
## 502: 18.81100  0 18.10    0 0.597 4.628 100.0 1.5539  24 666    20.2
## 503:  1.62864  0 21.89    0 0.624 5.019 100.0 1.4394   4 437    21.2
## 504: 11.10810  0 18.10    0 0.668 4.906 100.0 1.1742  24 666    20.2
## 505: 45.74610  0 18.10    0 0.693 4.519 100.0 1.6582  24 666    20.2
## 506: 18.49820  0 18.10    0 0.668 4.138 100.0 1.1370  24 666    20.2
##       black lstat medv predicted_medv
##   1: 374.43  1.73 50.0         39.718
##   2: 389.61  1.92 50.0         39.718
##   3: 395.62  1.98 34.9         39.718
##   4: 385.91  2.47 41.7         39.718
##   5: 390.49  2.87 36.4         39.718
##  ---
## 502:  28.79 34.37 17.9         11.934
## 503: 396.90 34.41 14.4         11.934
## 504: 396.90 34.77 13.8         11.934
## 505:  88.27 36.98  7.0         11.934
## 506: 396.90 37.97 13.8         11.934
```

```r
plot_boston_housing_data(boston_housing, title=paste('KNN Model with k =', try_k))
```



Now, this looks pretty reasonable, and we'd think this predictor would **generalize well** when facing new, not yet seen, data. This is a **low-bias**, **low-variance** predictor. We love ones like this.

Hence, the key take-away is that, throughout a range of **hyper-parameter** k from small to large, we have seen a spectrum of corresponding predictors from "low-bias high-variance" to "high-bias low-variance". This phenomenon is called the **BIAS-VARIANCE TRADE OFF**, a fundamental concept in Machine Learning that is applicable to not only KNN alone but to all modeling methods.

The bias-variance trade-off concerns the **generalizability of a trained predictor** in light of new data it's not seen before. If a predictor has high bias and/or high variance, it will not do well in new cases. **Good, generalizable predictors** need to have **both low bias and low variance**.

Out-of-Sample Error and Cross-Validation

To **quantify the generalizability of a predictor**, we need to estimate its **out-of-sample (OOS) error**, i.e. a certain measure of **how well the predictor performs on data not used in its training process**.

A popular way to produce such OOS error estimates is to perform **cross validation**. Refer to lecture slides or here for discussions on cross validation.

```
NB_CROSS_VALIDATION_FOLDS <- 5
NB_CROSS_VALIDATIONS <- 6
```

Now, let's consider **Root Mean Square Error** (**RMSE**) as our predictor-goodness evaluation criterion and use 5-fold cross validation 6 times to pick a KNN predictor that has satisfactory RMSE.

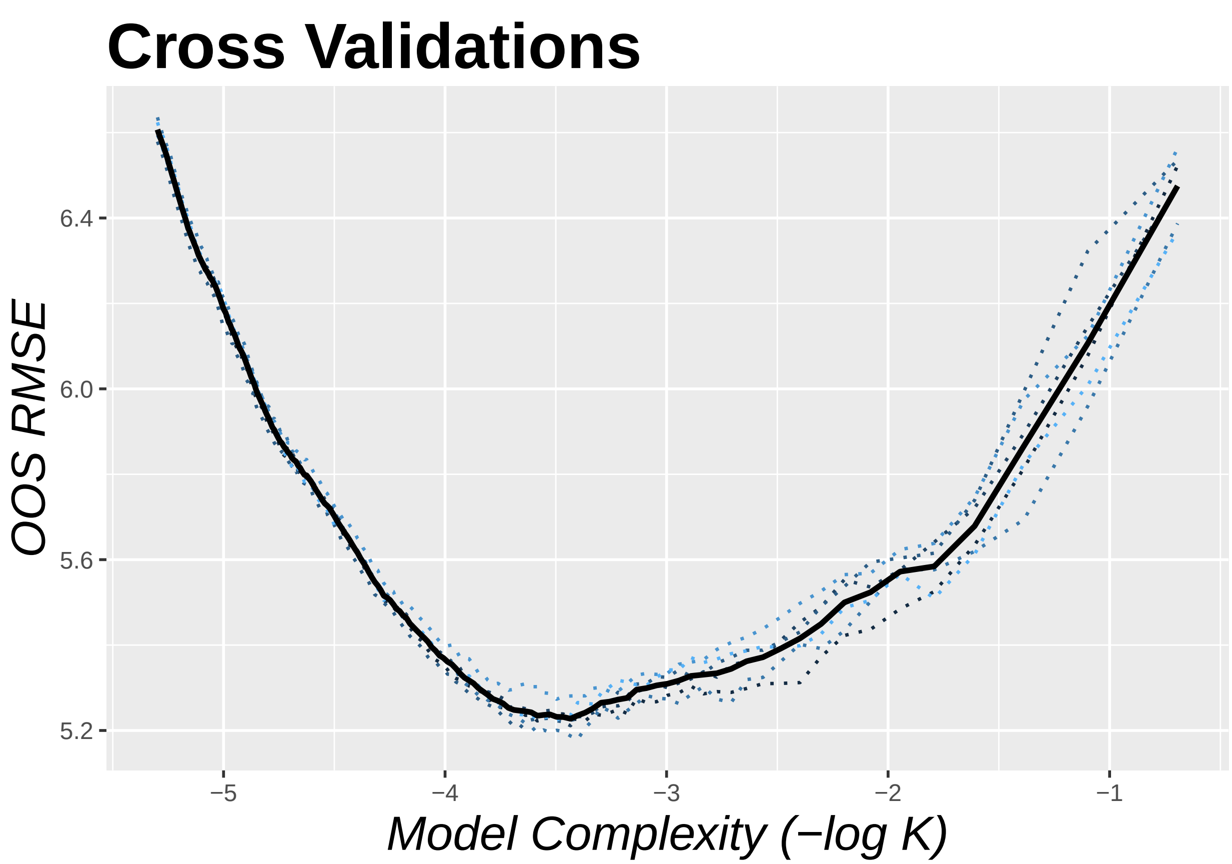
```r
k_range = 2 : 200
cross_validations_rmse = data.table(k=k_range, cv_avg_rmse=0.)
for (i in 1 : NB_CROSS_VALIDATIONS) {
  this_cross_validation_rmse =
    sqrt(docvknn(boston_housing[, .(lstat)], boston_housing$medv,
                 k=k_range, nfold=NB_CROSS_VALIDATION_FOLDS,
                 verbose=FALSE) / nb_samples)
  cross_validations_rmse[, (paste('cv_',  i, '_rmse', sep=''))] =
    this_cross_validation_rmse
  cross_validations_rmse[, cv_avg_rmse := cv_avg_rmse +
                           (this_cross_validation_rmse - cv_avg_rmse) / i]
}
```

```r
g <- ggplot(cross_validations_rmse)

for (i in 1 : NB_CROSS_VALIDATIONS) {
  g <- g + geom_line(aes_string(x='-log(k)', y=(paste('cv_',  i, '_rmse', sep='')),
                                color=i), linetype='dotted', size=0.6)
}

g <- g +
  geom_line(aes(x=-log(k), y=cv_avg_rmse),
            color='black', size=1) +
  ggtitle('Cross Validations') +
  xlab('Model Complexity (-log K)') + ylab('OOS RMSE') +
  guides(color=FALSE) +
  theme(plot.title=element_text(face='bold', size=24),
        axis.title=element_text(face='italic', size=18))

g
```



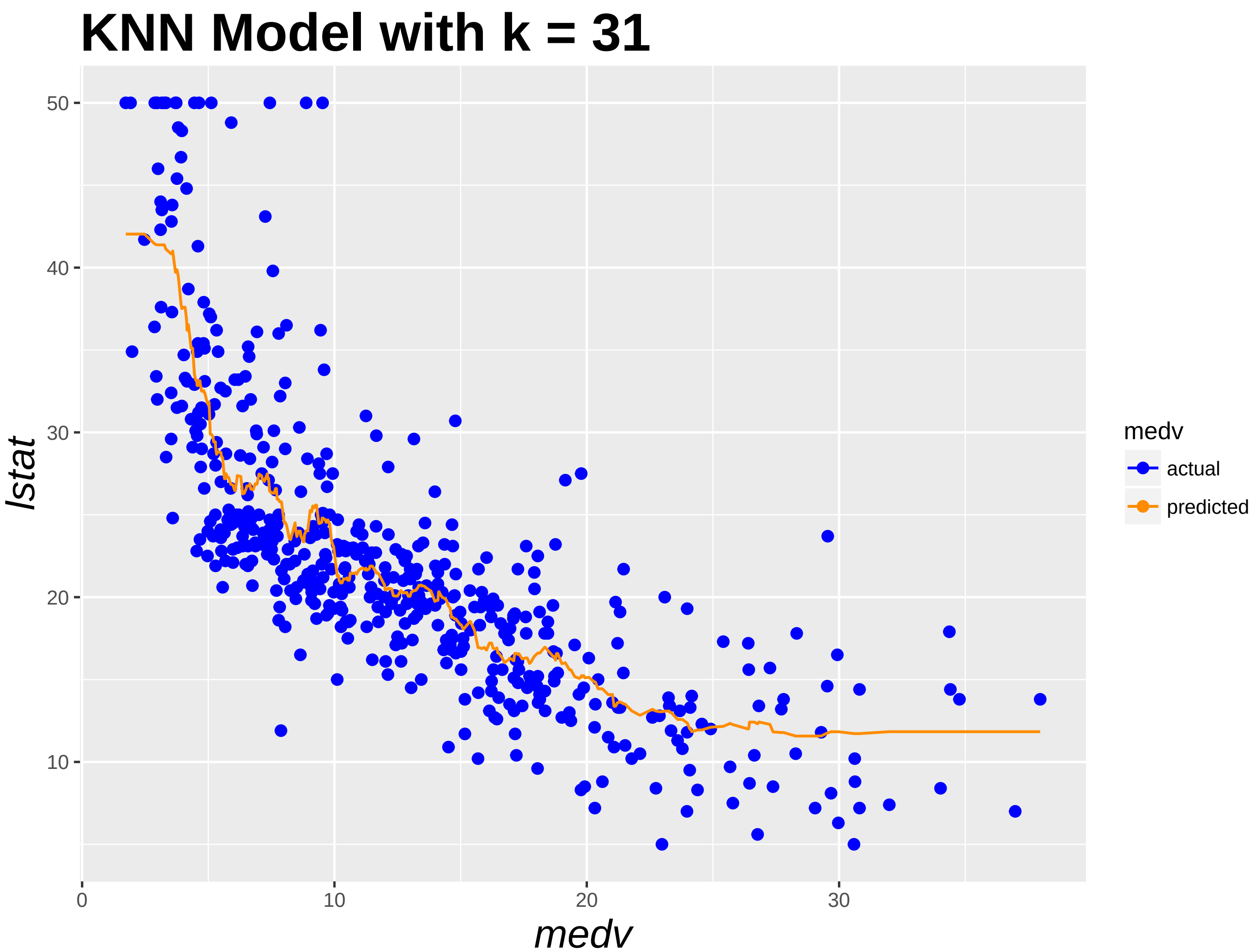
```r
best_k = k_range[which.min(cross_validations_rmse$cv_avg_rmse)]
```

From the above plot, the best k, one that minimizes the average cross-validation RMSE, is **31**, which produces the following predictor:

```r
knn_model <- kknn(medv ~ lstat,
                  train=boston_housing, test=boston_housing[, .(lstat)],
                  k=best_k, kernel='rectangular')
boston_housing[, predicted_medv := knn_model$fitted.values]
```

```
##          crim zn indus chas   nox    rm   age    dis rad tax ptratio
##   1:  1.46336  0 19.58    0 0.605 7.489  90.8 1.9709   5 403    14.7
##   2:  1.83377  0 19.58    1 0.605 7.802  98.2 2.0407   5 403    14.7
##   3:  0.03359 75  2.95    0 0.428 7.024  15.8 5.4011   3 252    18.3
##   4:  0.57529  0  6.20    0 0.507 8.337  73.3 3.8384   8 307    17.4
##   5:  0.08664 45  3.44    0 0.437 7.178  26.3 6.4798   5 398    15.2
##  ---
## 502: 18.81100  0 18.10    0 0.597 4.628 100.0 1.5539  24 666    20.2
## 503:  1.62864  0 21.89    0 0.624 5.019 100.0 1.4394   4 437    21.2
## 504: 11.10810  0 18.10    0 0.668 4.906 100.0 1.1742  24 666    20.2
## 505: 45.74610  0 18.10    0 0.693 4.519 100.0 1.6582  24 666    20.2
## 506: 18.49820  0 18.10    0 0.668 4.138 100.0 1.1370  24 666    20.2
##       black lstat medv predicted_medv
##   1: 374.43  1.73 50.0       42.03548
##   2: 389.61  1.92 50.0       42.03548
##   3: 395.62  1.98 34.9       42.03548
##   4: 385.91  2.47 41.7       42.03548
##   5: 390.49  2.87 36.4       41.44194
##  ---
## 502:  28.79 34.37 17.9       11.83548
## 503: 396.90 34.41 14.4       11.83548
## 504: 396.90 34.77 13.8       11.83548
## 505:  88.27 36.98  7.0       11.83548
## 506: 396.90 37.97 13.8       11.83548
```

```r
plot_boston_housing_data(boston_housing, title=paste('KNN Model with k =', best_k))
```



BONUS: implementation by the *caret* package

caret is a popular R package that provides standardized interfaces with 200+ Machine Learning algorithms.

Much of the above procedures can be re-done very succinctly with *caret* as follows:

```
library(caret)
```

```
cross_validated_knn_model =
  train(medv ~ lstat, data=boston_housing,
        method='kknn',
        tuneGrid=expand.grid(kmax=200,
                             kernel='rectangular',
                             distance=2),
        trControl=trainControl(method='repeatedcv',
                               number=NB_CROSS_VALIDATION_FOLDS,
                               repeats=NB_CROSS_VALIDATIONS,
                               allowParallel=TRUE))

cross_validated_knn_model
```

```
## k-Nearest Neighbors
##
## 506 samples
##   1 predictor
```

```
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 6 times)
## Summary of sample sizes: 405, 405, 405, 404, 405, 405, ...
## Resampling results:
##
##   RMSE      Rsquared
##   5.234369  0.6766338
##
## Tuning parameter 'kmax' was held constant at a value of 200
##
## Tuning parameter 'distance' was held constant at a value of 2
##
## Tuning parameter 'kernel' was held constant at a value of rectangular
##
```

```
best_k = cross_validated_knn_model$finalModel$best.parameters$k
```

The best k identified by *caret* is **39**. Note that there can be a range of acceptable "best" hyper-parameters because of randomization.