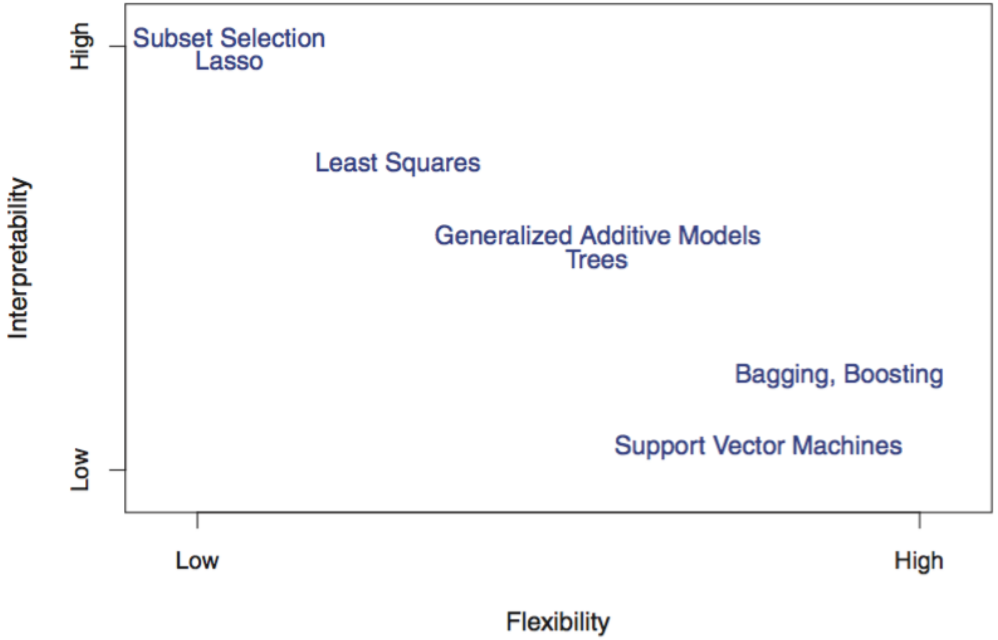
Supervised Learning

Nonparametric regression

July 8th, 2021

Model flexibility vs interpretability

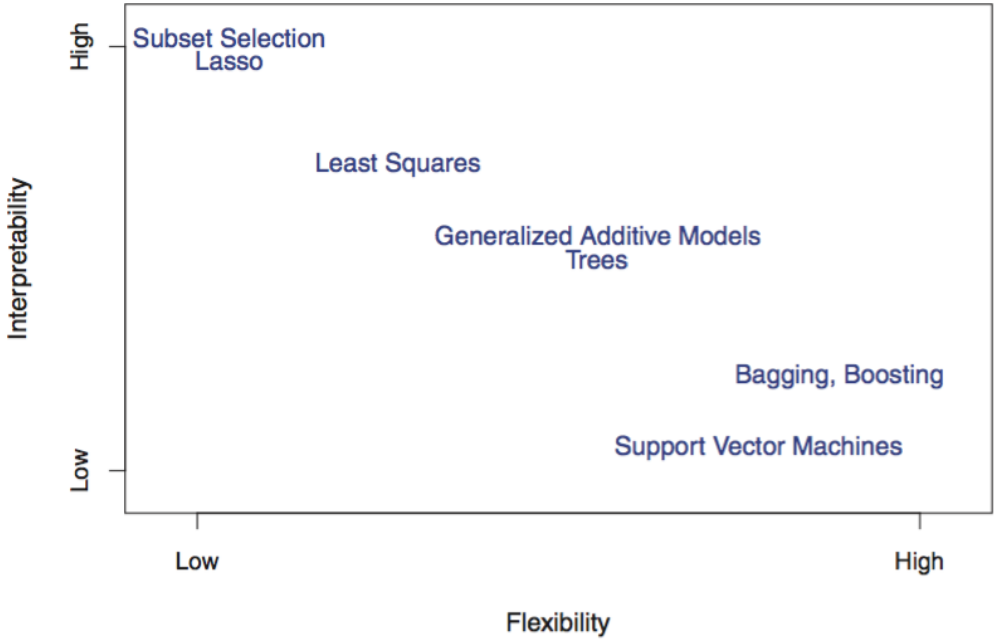
Figure 2.7, Introduction to Statistical Learning with Applications in R (ISLR)



Tradeoff between model's *flexibility* (i.e. how "curvy" it is) and how **interpretable** it is

- Simpler, parametric form of the model $\Rightarrow$ the easier it is to interpret

Model flexibility vs interpretability



- **Parametric** models, for which we can write down a mathematical expression for $f(X)$ **before observing the data**, *a priori* (e.g. linear regression), **are inherently less flexible**
- **Nonparametric** models, in which $f(X)$ is **estimated from the data** (e.g. kernel regression)

K Nearest Neighbors (KNN)

- Find the k data points **closest** to an observation x, use these to predit
  - Need to use some measure of distance, e.g., Euclidean distance
- KNN is data-driven, but we can actually write down the model *a priori*
- Regression:

$$\hat{Y}|X = \frac{1}{k}\sum_{i=1}^{k} Y_i \ ,$$

- Classification:

$$\hat{P}[Y = j|X] = \frac{1}{k}\sum_{i=1}^{k} 1(Y_i = j) \ ,$$

  - $1(\cdot)$ is the indicator function: returns 1 if TRUE, and 0 otherwise.
  - Summation yields the proportion of neighbors that are of class j

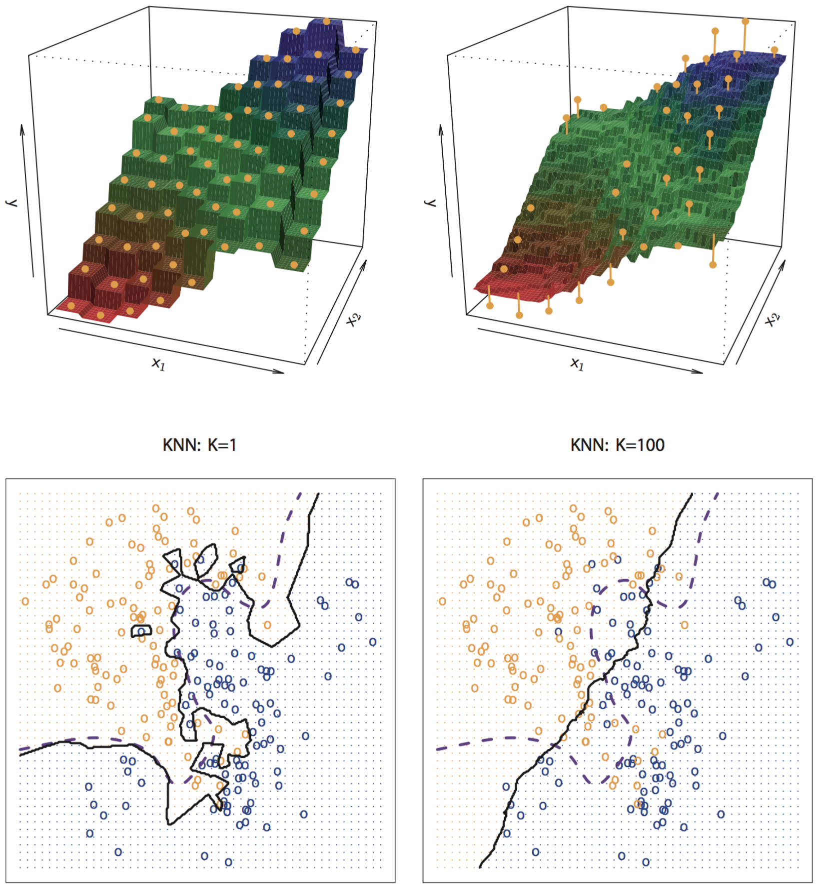
Finding the optimal number of neighbors k

The number of neighbors k is a tuning parameter (like λ is for ridge / lasso)

Determining the optimal value of k requires balancing bias and variance:

- If k is too small, the resulting model is *too flexible*,
    - low bias (it is right on average…if we apply KNN to an infinite number of datasets sampled from the same parent population)
    - high variance (the predictions have a large spread in values when we apply KNN to our infinite data). See the panels to the left on the next slide.
- If k is too large, the resulting model is *not flexible enough*,
    - high bias (wrong on average) and
    - low variance (nearly same predictions, every time). See the panels to the right on the next slide.

Finding the optimal number of neighbors k



(Figures 3.16 [top] and 2.16 [bottom], *Introduction to Statistical Learning* by James et al.)

KNN in context

Here are two quotes from ISLR to keep in mind when thinking about KNN:

- "As a general rule, parametric methods [like linear regression] will tend to outperform non-parametric approaches [like KNN] when there is a small number of observations per predictor." This is the *curse of dimensionality*: for data-driven models, the amount of data you need to get similar model performance goes up exponentially with p.

$\Rightarrow$ KNN might not be a good model to learn when the number of predictor variables is very large.

- "Even in problems in which the dimension is small, we might prefer linear regression to KNN from an interpretability standpoint. If the test MSE of KNN is only slightly lower than that of linear regression, we might be willing to forego a little bit of prediction accuracy for the sake of a simple model…"

$\Rightarrow$ KNN is not the best model to learn if inference is the goal of an analysis.

KNN: two critical points to remember

1. To determine which neighbors are the nearest neighbors, pairwise Euclidean distances are computed…so we may need to scale (or standardize) the individual predictor variables so that the distances are not skewed by that one predictor that has the largest variance.

2. Don't blindly compute a pairwise distance matrix! For instance, if n = 100,000, then your pairwise distance matrix will have 10^{10} elements, each of which uses 8 bytes in memory…resulting in a memory usage of 80 GB! Your laptop cannot handle this. It can barely handle 1-2 GB at this point. If n is large, you have three options: a. subsample your data, limiting n to be $\lesssim$ 15,000-20,000; b. use a variant of KNN that works with sparse matrices (matrices that can be compressed since most values are zero); or c. make use of a "kd tree" to more effectively (but only approximately) identify nearest neighbors.

The `FNN` package in `R` has an option to search for neighbors via the use of a kd tree.

But instead we will use the `caret` package…

Example data: MLB 2021 batting statistics

Downloaded MLB 2021 batting statistics leaderboard from Fangraphs

```
library(tidyverse)
mlb_data <- read_csv("http://www.stat.cmu.edu/cmsac/sure/2021/materials/data/fg_batting_2021.csv'
head(mlb_data)
```

```
## # A tibble: 6 × 23
##   Name    Team      G    PA    HR     R   RBI    SB `BB%` `K%`    ISO BABIP   AVG
##   <chr>   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr> <chr> <dbl> <dbl> <dbl>
## 1 Vladi… TOR      82   354    27    66    69     2 14.4% 17.2% 0.336 0.346 0.336
## 2 Ferna… SDP      68   288    27    66    58    18 12.5% 28.1% 0.395 0.333 0.302
## 3 Carlo… HOU      79   347    16    61    52     0 13.5% 17.0% 0.231 0.324 0.298
## 4 Marcu… TOR      82   372    21    63    54    10 8.9%  23.9% 0.256 0.329 0.286
## 5 Ronal… ATL      78   342    23    67    51    16 13.2% 24.3% 0.313 0.306 0.278
## 6 Shohe… LAA      82   322    31    60    67    12 11.2% 28.0% 0.418 0.29  0.277
## # … with 10 more variables: OBP <dbl>, SLG <dbl>, wOBA <dbl>, xwOBA <dbl>,
## #   `wRC+` <dbl>, BsR <dbl>, Off <dbl>, Def <dbl>, WAR <dbl>, playerid <dbl>
```

Data cleaning

- `janitor` package has convenient functions for data cleaning like `clean_names()`
- `parse_number()` function provides easy way to convert character to numeric columns

```
library(janitor)
mlb_data_clean <- clean_names(mlb_data)
mlb_data_clean <- mlb_data_clean %>%
  mutate_at(vars(bb_percent:k_percent), parse_number)
head(mlb_data_clean)
```

```
## # A tibble: 6 × 23
##   name      team      g    pa    hr     r   rbi    sb bb_pe…¹ k_per…²   iso babip
##   <chr>     <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>   <dbl> <dbl> <dbl>
## 1 Vladimi… TOR      82   354    27    66    69     2    14.4    17.2 0.336 0.346
## 2 Fernand… SDP      68   288    27    66    58    18    12.5    28.1 0.395 0.333
## 3 Carlos … HOU      79   347    16    61    52     0    13.5    17   0.231 0.324
## 4 Marcus … TOR      82   372    21    63    54    10     8.9    23.9 0.256 0.329
## 5 Ronald … ATL      78   342    23    67    51    16    13.2    24.3 0.313 0.306
## 6 Shohei … LAA      82   322    31    60    67    12    11.2    28   0.418 0.29
## # … with 11 more variables: avg <dbl>, obp <dbl>, slg <dbl>, w_oba <dbl>,
## #   xw_oba <dbl>, w_rc <dbl>, bs_r <dbl>, off <dbl>, def <dbl>, war <dbl>,
## #   playerid <dbl>, and abbreviated variable names ¹bb_percent, ²k_percent
```

KNN example

`caret` is a package of functions designed to simplify training, tuning, and testing statistical learning methods

- first create partitions for training and test data using `createDataPartition()`

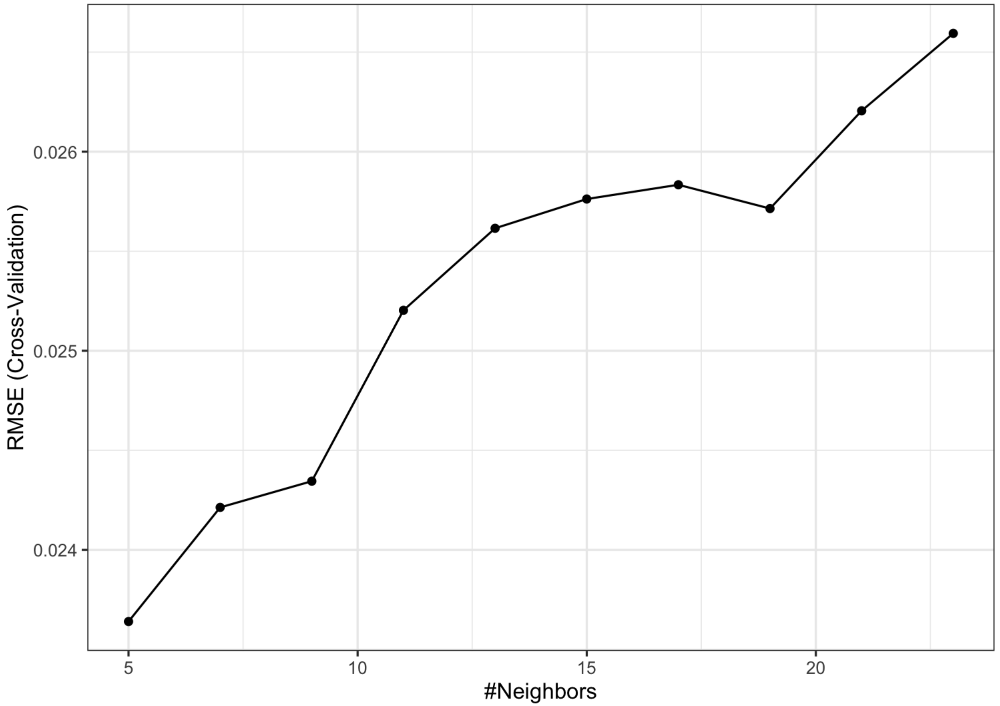
```
library(caret)
set.seed(1960)
train_i <- createDataPartition(y = mlb_data_clean$w_oba, p = 0.7, list = FALSE) %>%
  as.numeric()
train_mlb_data <- mlb_data_clean[train_i,]
test_mlb_data <- mlb_data_clean[-train_i,]
```

- next `train()` to find the optimal `k` on the training data with cross-validation

```
set.seed(1971)
init_knn_mlb_train <- train(w_oba ~ bb_percent + k_percent + iso,
                            data = train_mlb_data, method = "knn",
                            trControl = trainControl("cv", number = 10),
                            preProcess = c("center", "scale"),
                            tuneLength = 10)
```

KNN example

```
ggplot(init_knn_mlb_train) + theme_bw()
```



KNN example

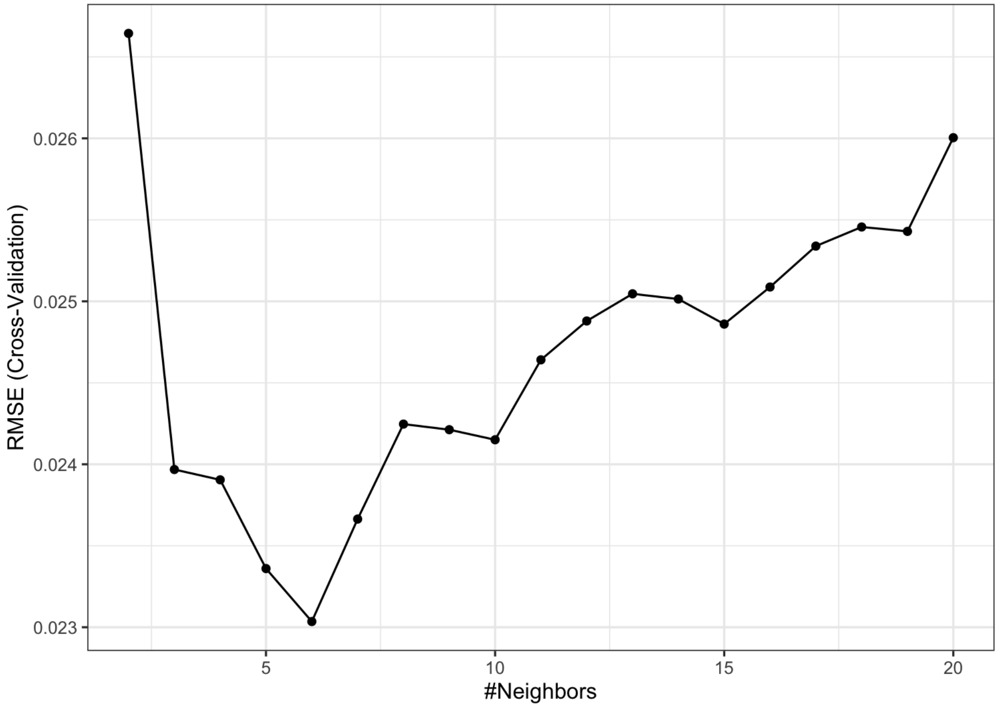
Can manually create a **tuning grid** to search over for the tuning parameter k

```
set.seed(1979)
tune_knn_mlb_train <- train(w_oba ~ bb_percent + k_percent + iso,
                            data = train_mlb_data, method = "knn",
                            trControl = trainControl("cv", number = 10),
                            preProcess = c("center", "scale"),
                            tuneGrid = expand.grid(k = 2:20))
tune_knn_mlb_train$results
```

```
##     k       RMSE  Rsquared        MAE      RMSESD RsquaredSD       MAESD
## 1   2 0.02664407 0.4918627 0.02190051 0.005738257  0.2369480 0.005007983
## 2   3 0.02396789 0.6196434 0.01953764 0.004799971  0.1591891 0.004490957
## 3   4 0.02390454 0.6207057 0.01964823 0.004899709  0.1827772 0.004354430
## 4   5 0.02336004 0.6244300 0.01938406 0.005161427  0.1949702 0.004402360
## 5   6 0.02303508 0.6335212 0.01917729 0.004972606  0.1561728 0.004147938
## 6   7 0.02366376 0.6262434 0.01946073 0.005400162  0.1666846 0.004507068
## 7   8 0.02424653 0.6101839 0.01990244 0.005227669  0.1568779 0.004293239
## 8   9 0.02421224 0.6337331 0.01979337 0.005645555  0.1611892 0.004766752
## 9  10 0.02415043 0.6448377 0.01986623 0.005677580  0.1629994 0.005095074
## 10 11 0.02464093 0.6455271 0.02023478 0.005640372  0.1534025 0.004948348
## 11 12 0.02487926 0.6445562 0.02042024 0.005492106  0.1663580 0.004889936
## 12 13 0.02504601 0.6374670 0.02045053 0.005894155  0.1922161 0.005138379
```

KNN example

```
ggplot(tune_knn_mlb_train) + theme_bw()
```



KNN example

```
tune_knn_mlb_train$bestTune
```

```
##   k
## 5 6
```

```
test_preds <- predict(tune_knn_mlb_train, test_mlb_data)
head(test_preds)
```

```
## [1] 0.3861667 0.3736667 0.3738333 0.3771667 0.3381667 0.3716667
```

```
RMSE(test_preds, test_mlb_data$w_oba)
```

```
## [1] 0.02631488
```

What does KNN remind you of?...

Kernels

A kernel $K(x)$ is a weighting function used in estimators, and technically has only one required property:

- $K(x) \geq 0$ for all x

However, in the manner that kernels are used in statistics, there are two other properties that are usually satisfied:

- $\int_{-\infty}^{\infty} K(x)dx = 1$; and
- $K(-x) = K(x)$ for all x.

In short: **a kernel is a symmetric PDF!**

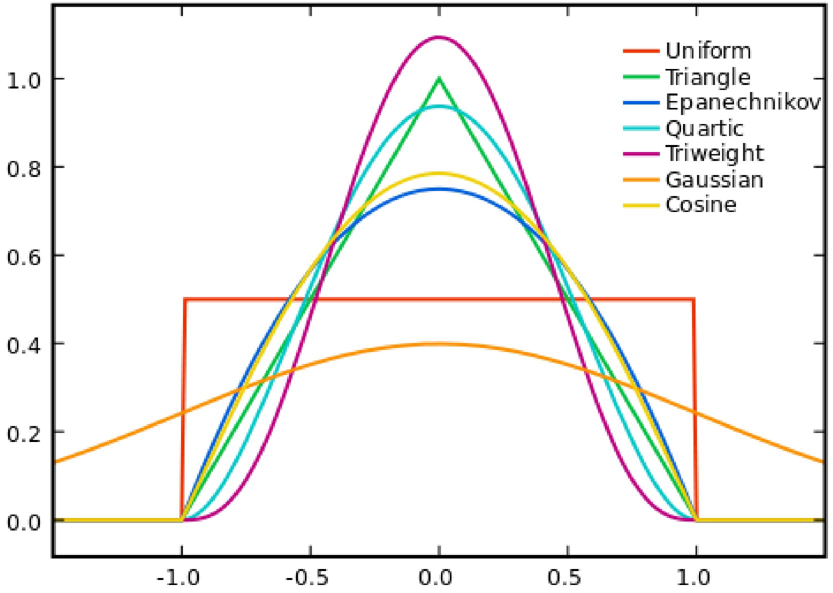
Kernel density estimation

Goal: estimate the PDF $f(x)$ for all possible values (assuming it is continuous / smooth)

$$\text{Kernel density estimate: } \hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h} K_h(x - x_i)$$

- $n =$ sample size, $x =$ new point to estimate $f(x)$ (does NOT have to be in dataset!)
- $h =$ **bandwidth**, analogous to histogram bin width, ensures $\hat{f}(x)$ integrates to 1
- $x_i = i$th observation in dataset
- $K_h(x - x_i)$ is the **Kernel** function, creates **weight** given distance of ith observation from new point
    - as $|x - x_i| \to \infty$ then $K_h(x - x_i) \to 0$, i.e. further apart ith row is from x, smaller the weight
    - as **bandwidth** $h \uparrow$ weights are more evenly spread out (as $h \downarrow$ more concentrated around x)
    - typically use **Gaussian** / Normal kernel: $\propto e^{-(x-x_i)^2/2h^2}$
    - $K_h(x - x_i)$ is large when x_i is close to x

Commonly Used Kernels



A general rule of thumb: the choice of kernel will have little effect on estimation, particularly if the sample size is large! The Gaussian kernel (i.e., a normal PDF) is by far the most common choice, and is the default for `R` functions that utilize kernels.

Kernel regression

We can apply kernels in the regression setting as well as in the density estimation setting!

The classic kernel regression estimator is the **Nadaraya-Watson** estimator:

$$\hat{y}_h(x) = \sum_{i=1}^{n} w_i(x) Y_i \ ,$$

where

$$w_i(x) = \frac{K\left(\frac{x - X_i}{h}\right)}{\sum_{j=1}^{n} K\left(\frac{x - X_j}{h}\right)} \ .$$

Regression estimate is the average of all the *weighted* observed response values;

- Farther x is from observation $\Rightarrow$ less weight that observation has in determining the regression estimate at x

Kernel regression with `np`

Use the `npregbw` function to tune bandwidth using **generalized cross-validation**

```
library(np)
mlb_bw0 <- npregbw(w_oba ~ bb_percent + k_percent + iso,
                   data = train_mlb_data)
```

Generate predictions with `npreg` with provided bandwidth object

```
mlb_test_npreg <- npreg(mlb_bw0, newdata = test_mlb_data)
RMSE(mlb_test_npreg$mean, test_mlb_data$w_oba)
```

```
## [1] 0.02107194
```