Supervised Machine Learning and Learning Theory

Lecture 3: The bias-variance trade-off, and K-nearest neighbors

September 13, 2024

In-class quiz questions

- Recall the definition of R^2: $1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$, what is the meaning of R^2 as a measure of linear regression? Is R^2 always non-negative?

- Can you explain when R^2 is non-negative?

In-class quiz questions

- Recall the definition of correlation coefficient: $\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\cdot\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})}}$,
  where $\bar{x}=\frac{1}{n}\sum_{i=1}^{n}x_i$, $\bar{y}=\frac{1}{n}\sum_{i=1}^{n}y_i$
- Let x be a uniformly random draw from $\{x_1, x_2, \ldots, x_n\}$. Similarly, let y be a uniformly random draw from $\{y_1, y_2, \ldots, y_n\}$
- Suppose that x and y are independent, meaning that for any realization of x, the value of y is unaffected, i.e., $\Pr[x=x_i, y=y_j]=\Pr[x=x_i]\cdot\Pr[y=y_j]$. What is the correlation coefficient between x and y?

- Generalize this to the case when x and y are arbitrary, independent random variables?

In-class quiz questions

- Recall the ordinary least squares estimator as follows:

$$\hat{\beta} = (X^\top X)^{-1} X^\top y$$

- When is the OLS estimator well-defined?

In-class quiz questions

- What is the rank of the following matrix?

$$A = diag([n, n-1, \dots, 1]) = \begin{bmatrix} n, 0, & \dots & , 0 \\ 0, n-1, 0, & & \dots, 0 \\ 0, 0, n-2, & & \dots, 0 \\ & \dots & \\ 0, 0, & \dots & , 0, 1 \end{bmatrix}$$

- What about the following matrix?

$$A = diag([0, \dots, 0, r, r-1, \dots, 1]) = \begin{bmatrix} 0, 0, & \dots & , 0 \\ & \dots & \\ 0, 0, \dots, r, 0, & \dots & , 0 \\ & 0, 0, \dots, 0, r-1 \dots, 0 & \\ & \dots & \\ 0, 0, & \dots & , 0, 1 \end{bmatrix}$$

Lecture plan

- **The bias-variance tradeoff**

A fundamental trade-off in machine learning

- The bias-variance trade-off is a fundamental aspect of a machine learning model
- Recall the mathematical setup of supervised machine learning: we have a set of samples $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, in which every sample is drawn from an unknown distribution D
- The training loss of a model f_W is defined as

$$\hat{L}(f_W) = \frac{1}{n}\sum_{i=1}^{n} \ell(f_W(x_i), y_i)$$

- The test loss is defined as

$$L(f_W) = \mathbb{E}_{(x,y)\sim D}[\ell(f_W(x), y)]$$

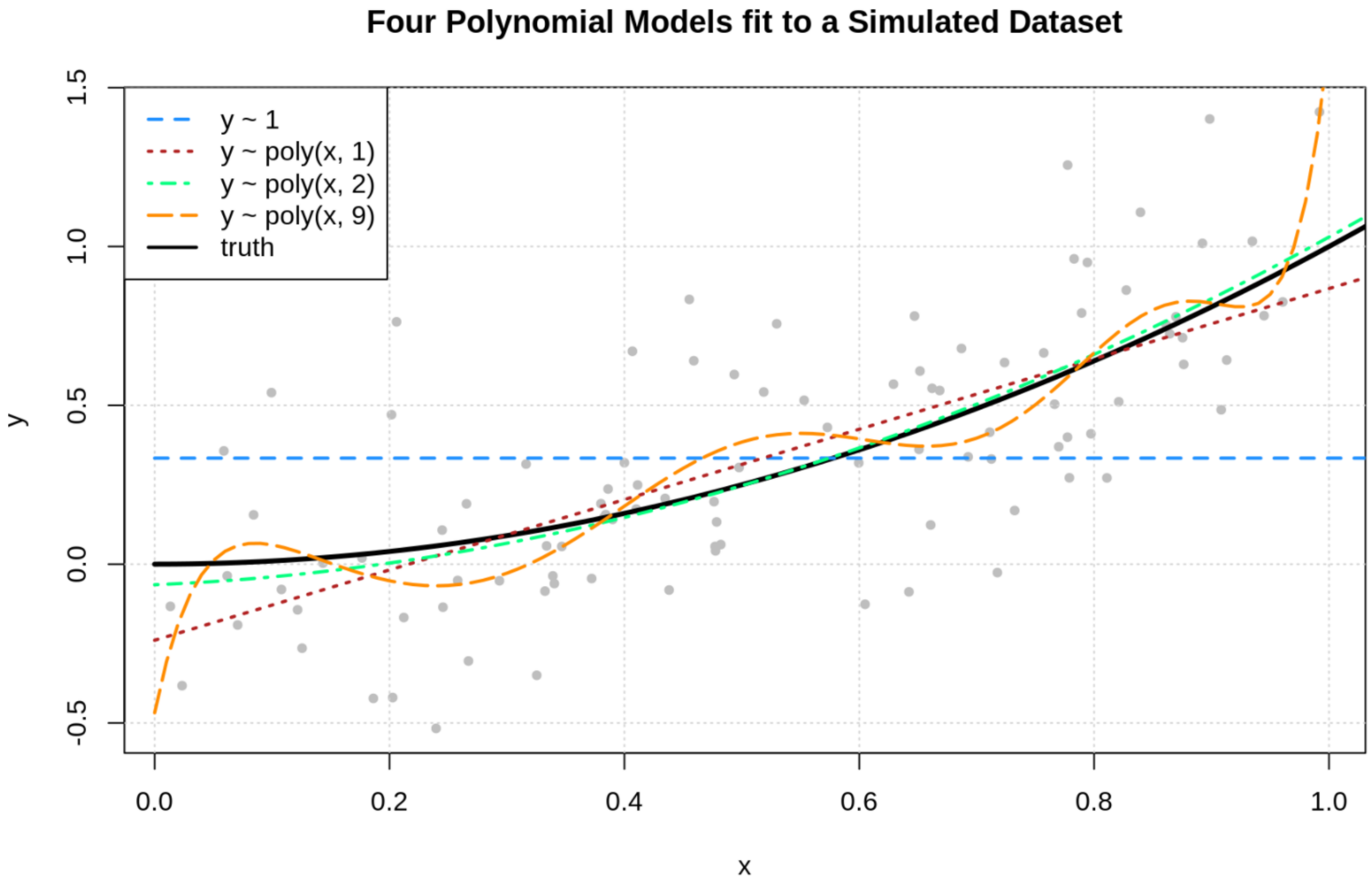
Ensuring that the gap between these two are small is a fundamental challenge

Let us look at a case study

- Suppose we would like to train a model to learn the true regression function $f(x) = x^2$ (x is a scalar)

- We use polynomial features in this case study:
  - A constant function: $\hat{f}_0(x) = \hat{\beta}_0$
  - A linear function: $\hat{f}_1(x) = \hat{\beta}_0 + x \cdot \hat{\beta}_1$
  - A quadratic function: $\hat{f}_2(x) = \hat{\beta}_0 + x \cdot \hat{\beta}_1 + x^2 \cdot \hat{\beta}_2$
  - A ninth-degree polynomial function: $\hat{f}_9(x) = \hat{\beta}_0 + x \cdot \hat{\beta}_1 + \cdots + x^9 \cdot \hat{\beta}_9$

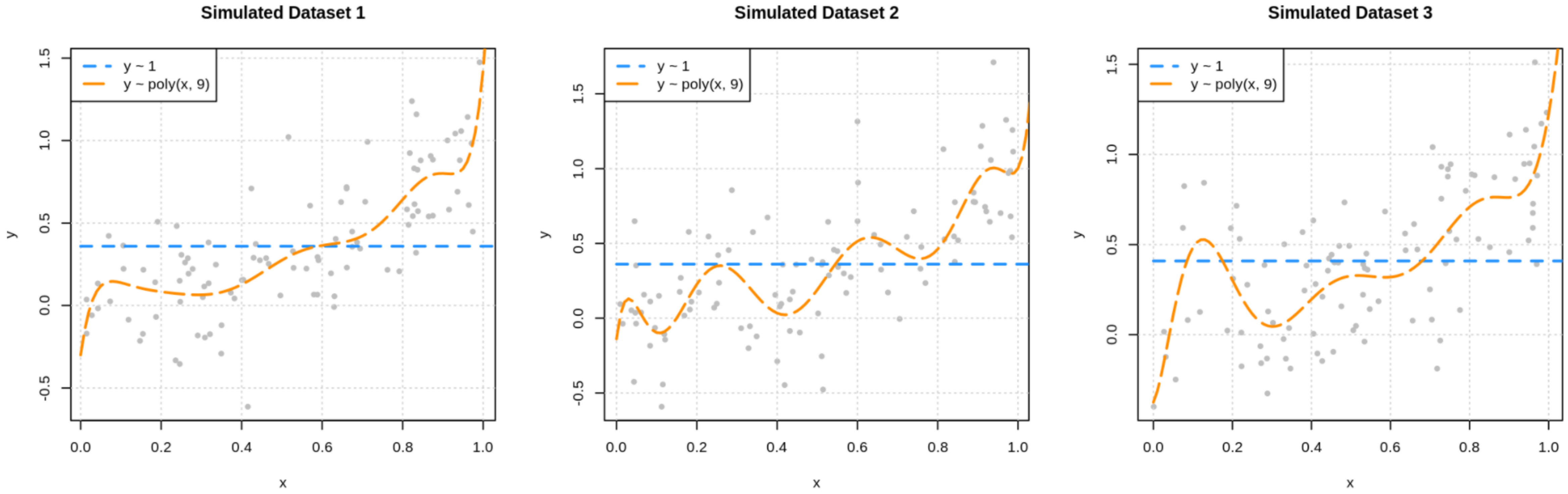
Four fitted models



- Zero predictor model fits poorly
- Linear model is reasonable
- Quadratic model fits much better
- Ninth degree model seems rather wild

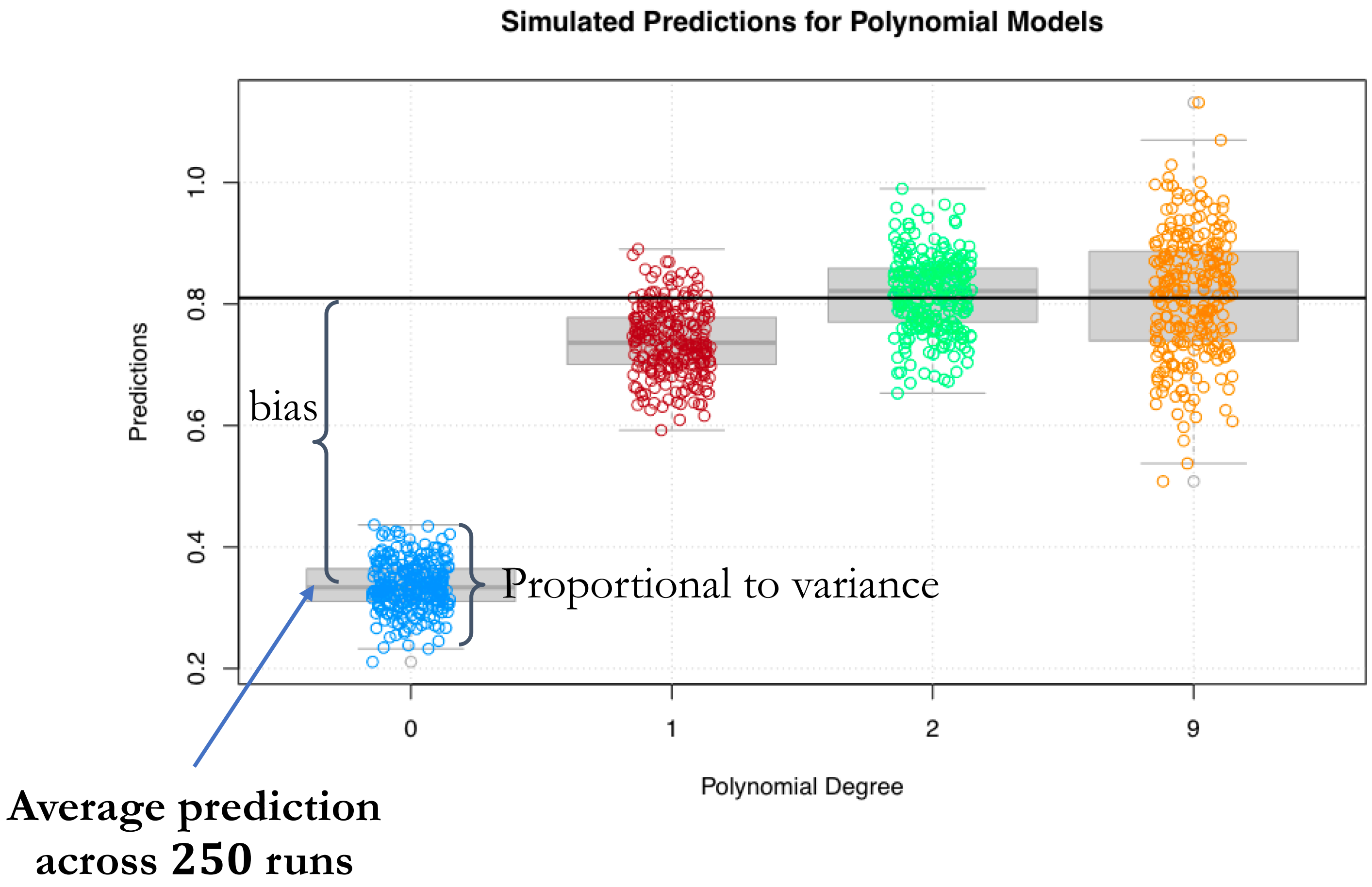
Repeat the experiment for three times

- The zero predictor $\hat{f}_0(x)$ slightly varies, but the ninth-degree polynomial varies $\hat{f}_9(x)$ quite a bit
- Variance of $\hat{f}_0(x)$ is smaller than the variance of $\hat{f}_9(x)$



Predicting $f(x_0)$

- $x_0 = 0.9$
- $y = f(0.9) = x_0^2 = 0.81$
- 250 independent runs: For each resample, we fit polynomials with degree $0, 1, 2, 9$, and plot $\hat{f}(0.9)$



Predicting $f(x_0)$

- Squared bias:

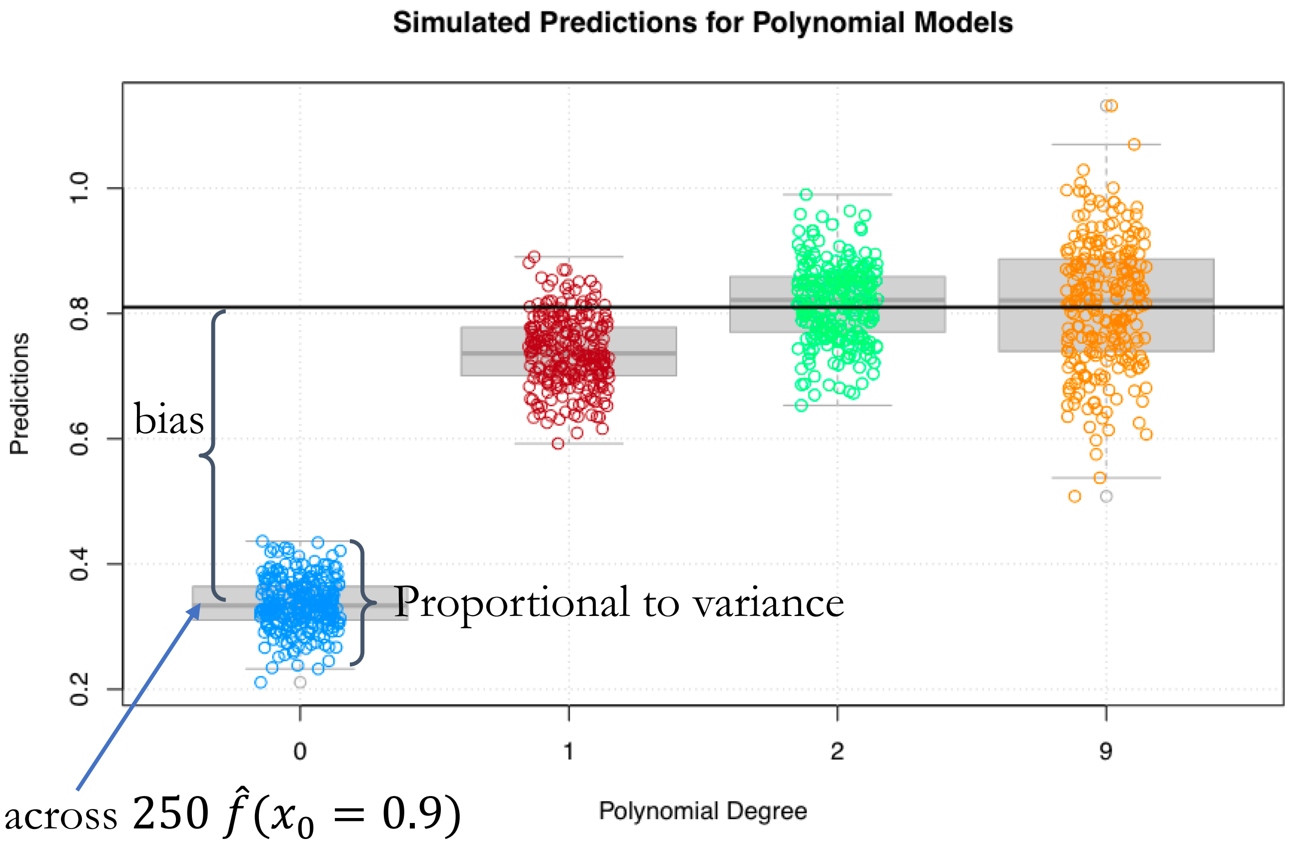
  $$\hat{f}_2(x) \approx \hat{f}_9(x) < \hat{f}_1(x) < \hat{f}_0(x)$$

- Increasing degree from 2 to 9 does not further reduce bias

- Variance:

  $$\hat{f}_0(x) < \hat{f}_1(x) < \hat{f}_2(x) < \hat{f}_9(x)$$

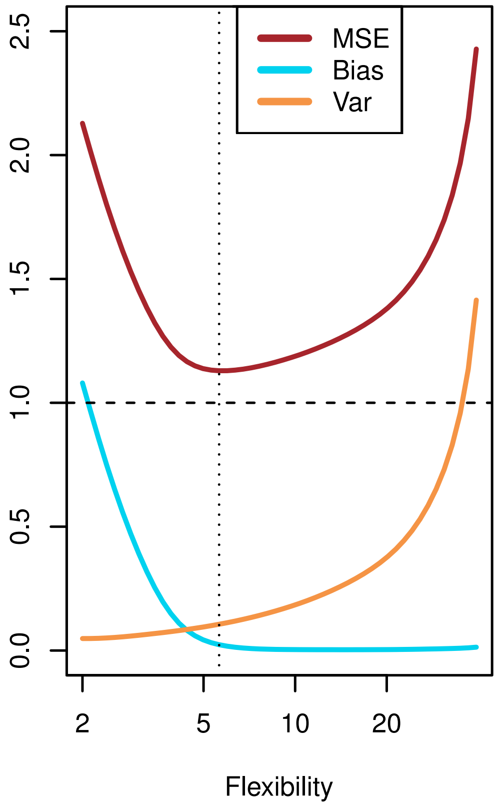
- Increasing degree increases variance



Average prediction across 250 $\hat{f}(x_0 = 0.9)$

Illustration

- Bias-variance curve as a function of the degree of the polynomial:



Let us study the test loss more deeply

- Suppose the true function is f
- Let the labels be defined as $y = f(x) + \varepsilon$, where $\mathbb{E}[\varepsilon] = 0$
- Let $S = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ be the training dataset
- Let $\hat{f}$ be a function estimated from the training dataset
- Let x be a random sample drawn from D. The test MSE is defined as

$$L(x) = E_{(x,y)\sim D}\left[\left(y - \hat{f}(x)\right)^2\right]$$

Let us expand the test loss

- The test MSE is equal to

$$
\begin{aligned}
L(x) &= \mathbb{E}_{(x,y)\sim D}[(y-\hat{f}(x))^2] \\
&= \mathbb{E}_{(x,y)\sim D}\left[\left(y - f(x) + f(x) - \mathbb{E}_S[\hat{f}(x)] + \mathbb{E}_S[\hat{f}(x)] - \hat{f}(x)\right)^2\right] \\
&= \mathbb{E}_{(x,y)\sim D}\left[\left(\varepsilon + f(x) - \mathbb{E}_S[\hat{f}(x)] + \mathbb{E}_S[\hat{f}(x)] - \hat{f}(x)\right)^2\right]
\end{aligned}
$$

- Recall that $\mathbb{E}[\varepsilon] = 0$, thus, the above must be equal to

$$
L(x) = \mathbb{E}_{(x,y)\sim D}[\varepsilon^2] + \mathbb{E}_{(x,y)\sim D}\left[\left(f(x) - \mathbb{E}_S[\hat{f}(x)] + \mathbb{E}_S[\hat{f}(x)] - \hat{f}(x)\right)^2\right]
$$

- $\mathrm{Var}(\varepsilon) = \mathbb{E}_{(x,y)\sim D}[\varepsilon^2]$ is the **irreducible error** from observing label y

Let us look at the reducible error

- The reducible error term:

$$\mathbb{E}_{(x,y)\sim D,S}\left[\left(f(x) - \mathbb{E}_S[\hat{f}(x)] + \mathbb{E}_S[\hat{f}(x)] - \hat{f}(x)\right)^2\right]$$

$$= \mathbb{E}_{(x,y)\sim D,S}\left[\left(f(x) - \mathbb{E}_S[\hat{f}(x)]\right)^2\right] + \mathbb{E}_{(x,y)\sim D,S}\left[\left(\mathbb{E}_S[\hat{f}(x)] - \hat{f}(x)\right)^2\right]$$
$$+ 2\mathbb{E}_{(x,y)\sim D,S}\left[\left(f(x) - \mathbb{E}_S[\hat{f}(x)]\right) \cdot \left(\mathbb{E}_S[\hat{f}(x)] - \hat{f}(x)\right)\right]$$

This is zero: $\mathbb{E}_{\boldsymbol{x}}[\mathbb{E}_{\boldsymbol{x}}[\boldsymbol{x}] - \boldsymbol{x}] = \mathbf{0}$

$$= \mathbb{E}_{(x,y)\sim D,S}\left[\left(f(x) - \mathbb{E}_S[\hat{f}(x)]\right)^2\right] + \mathbb{E}_{(x,y)\sim D,S}\left[\left(\mathbb{E}_S[\hat{f}(x)] - \hat{f}(x)\right)^2\right]$$

This is $\boldsymbol{y}$

$$= \mathrm{Bias}\left(\hat{f}(x)\right)^2 + \mathrm{Var}\left(\hat{f}(x)\right)$$

To summarize the derivations

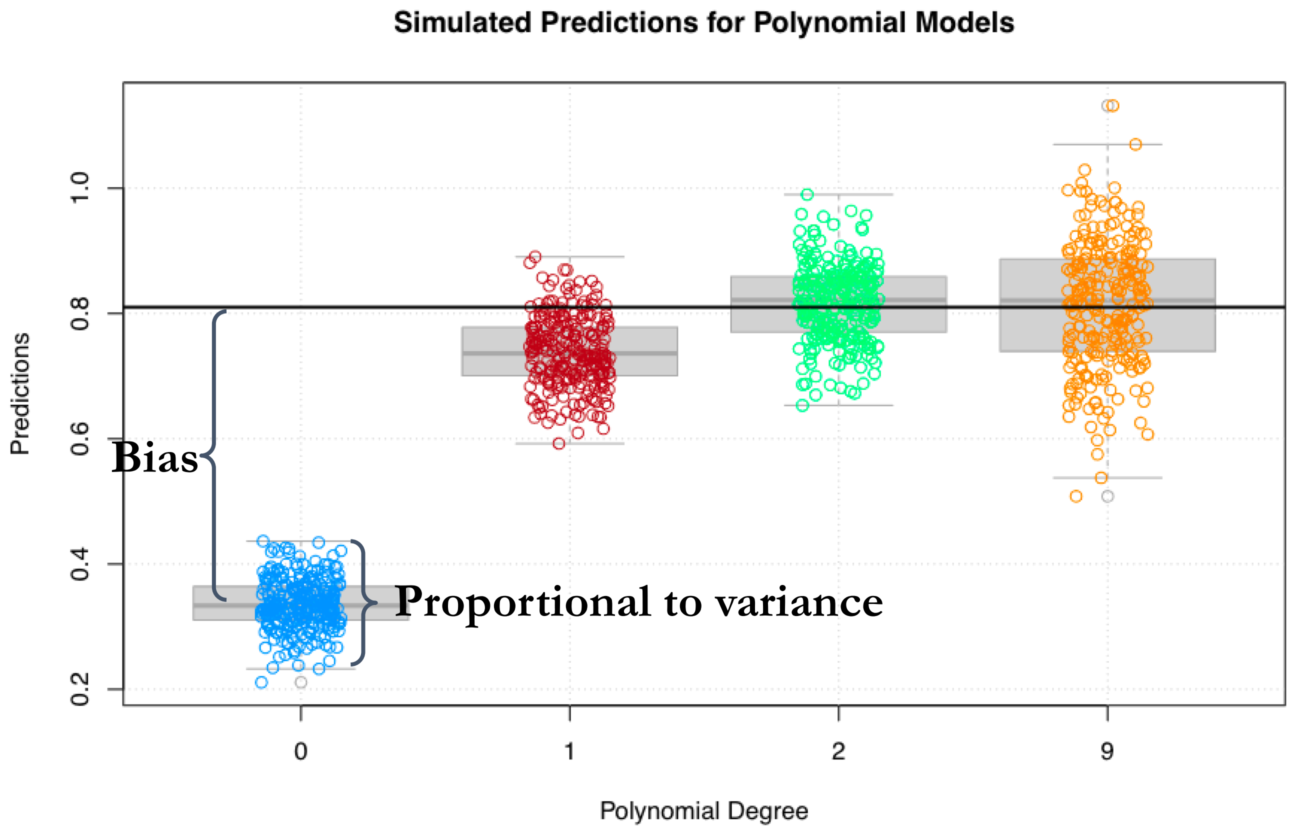
- Let x be a test sample from D and let $y = f(x) + \varepsilon$
- Let $\hat{f}$ be the estimator learned from the training dataset
- The expected test error over the training dataset is equal to

$$\mathbb{E}_S[L(x)] = \mathbb{E}_S\left[\left(\varepsilon + f(x) - \mathbb{E}_S[\hat{f}(x)] + \mathbb{E}_S[\hat{f}(x)] - \hat{f}(x)\right)^2\right]$$

$$= \underbrace{\mathrm{Var}(\varepsilon)}_{\textbf{Irreducible error}} + \mathrm{Bias}\left(\hat{f}(x)\right)^2 + \mathrm{Var}\left(\hat{f}(x)\right)$$

This variance is from the randomness of the training dataset upon the estimator $\hat{f}$

Back to the case study

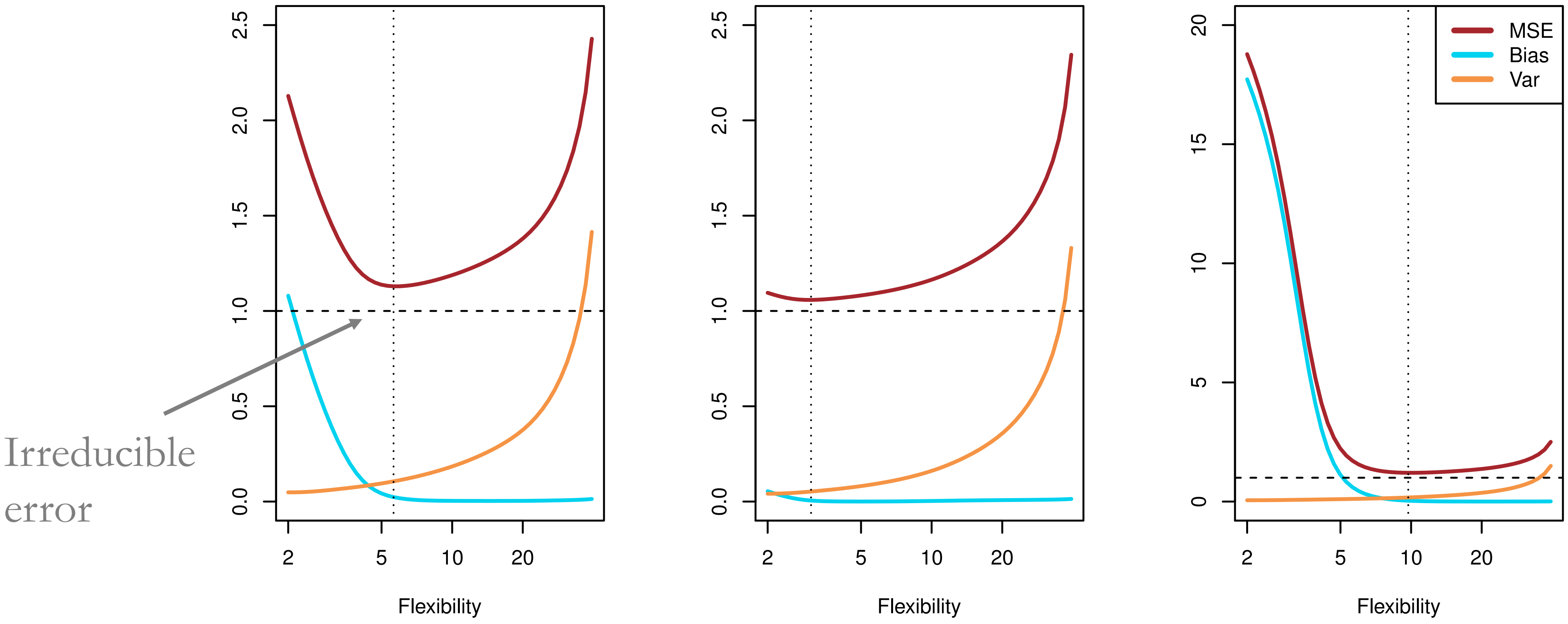


Visualization of bias variance

f is degree nine, high noise

f is degree two, high noise

f is degree one, low noise



Lecture plan

- K-nearest neighbors (KNN)

K-nearest neighbors regression

- Unlike linear regression, here there are no parameters (aka. non-parametric)
- K is a user-defined constant: K is an integer, e.g., $1,2,3,\cdots$
- Given a value for K and a prediction point x_0, $\hat{f}(x_0)$ is the average of the responses of K nearest neighbors:

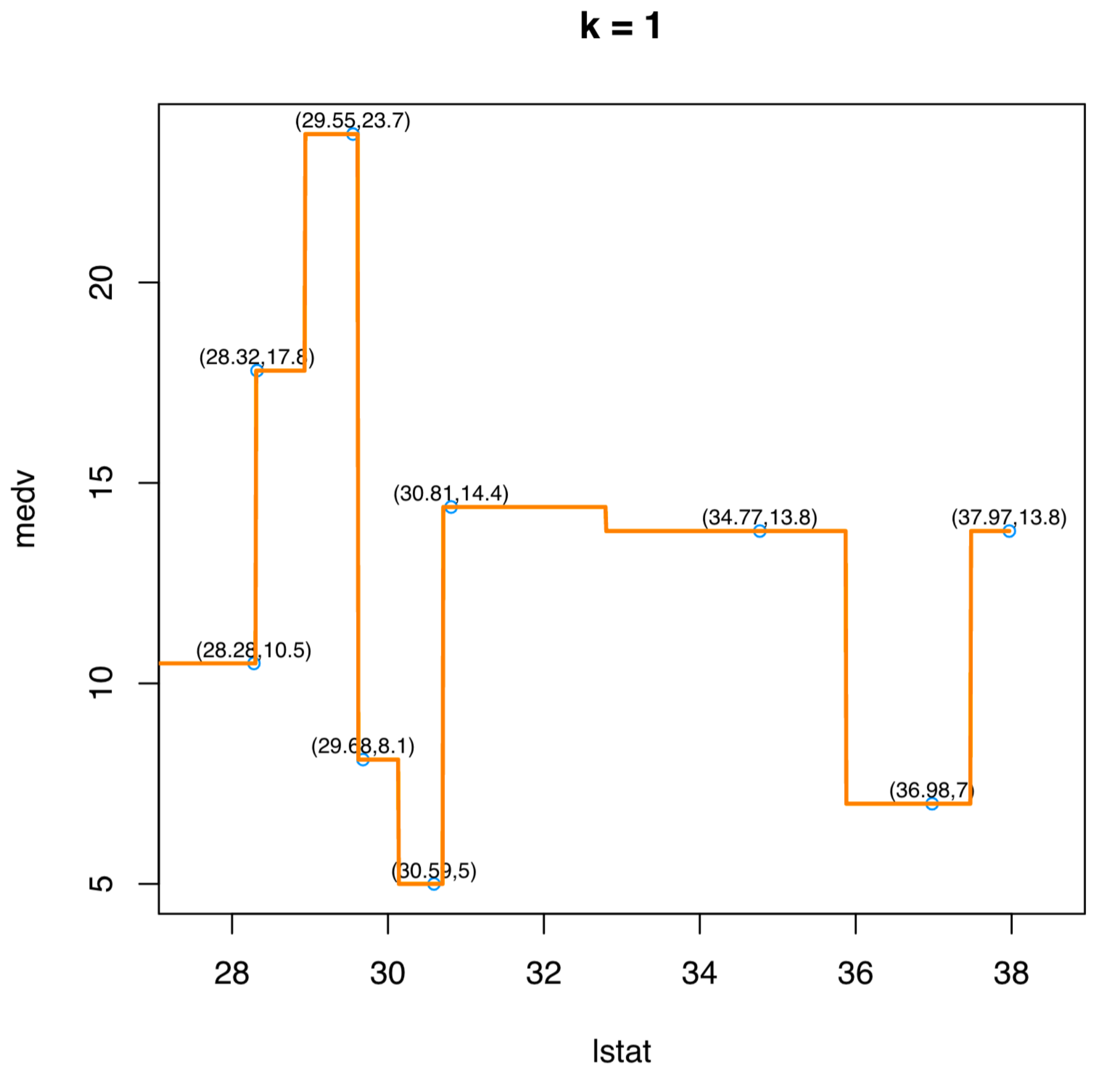
$$\hat{f}(x_0) = \frac{1}{K} \sum_{x_i \in N_K(x_0)} y_i$$

- $N_K(x_0)$ is the set of K training observations that are closest to x_0

Example: 1-nearest neighbor regression

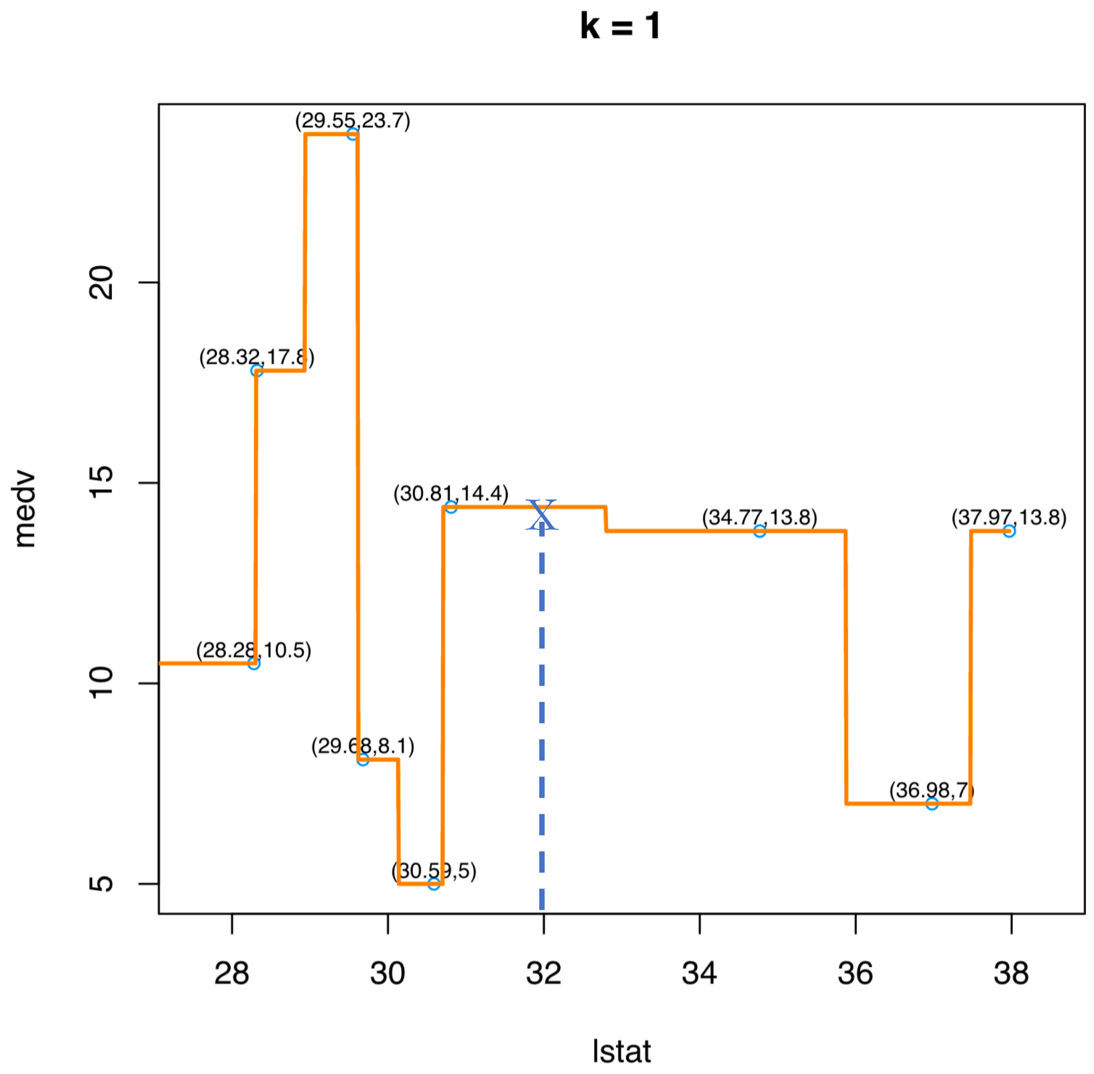
- Prediction of the median house value of a neighbor given the percentage of households with low socioeconomic status (LSTAT)

- Orange curve: $\hat{f}(x_0)$
  - $\hat{f}(x_0)$ equals to the response of x_0's nearest neighbor
  - $\hat{f}(x_0)$ is a step function



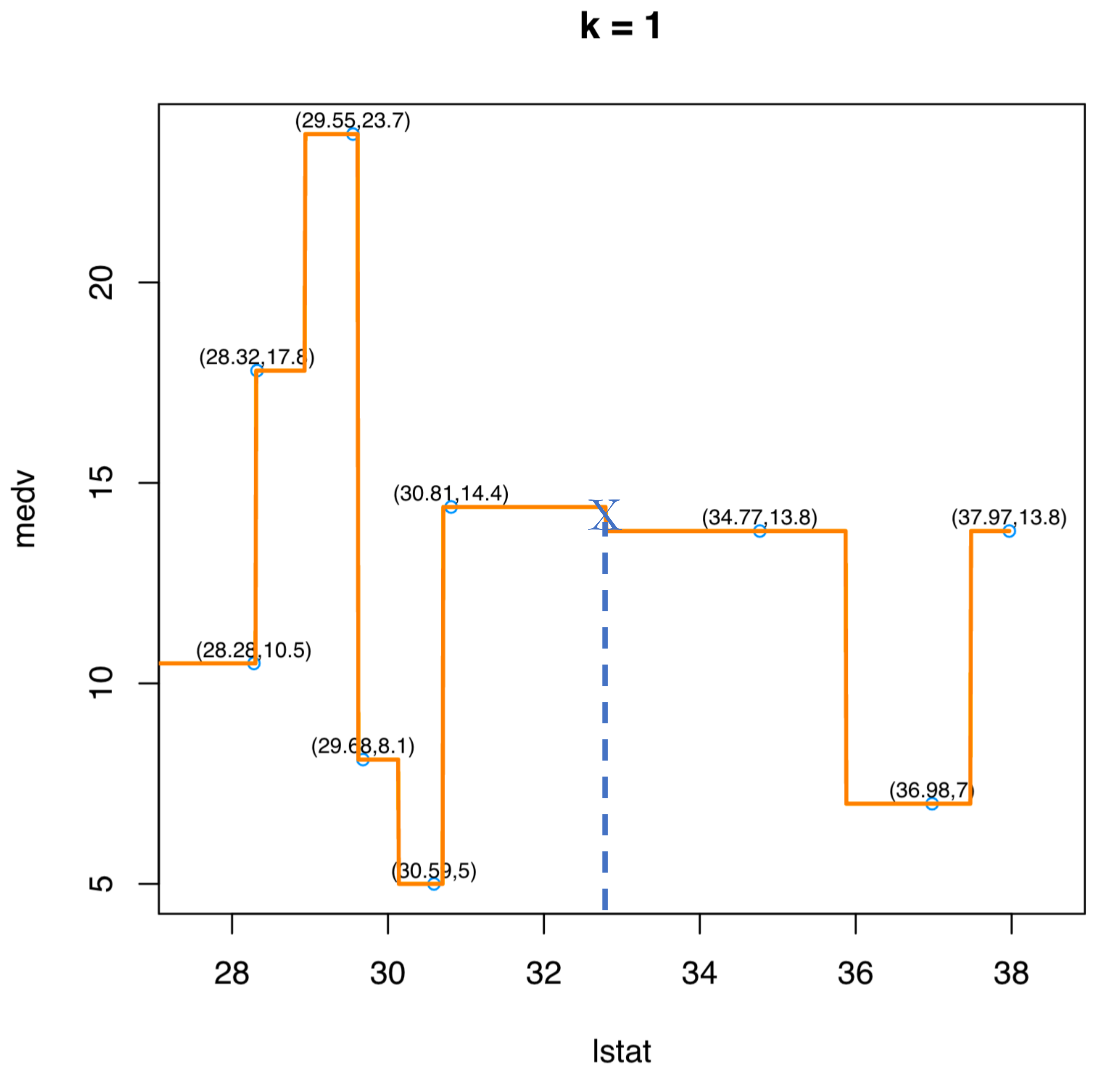
Example: 1-nearest neighbor regression

- $x_0 = 32$
- $N_K(x_0) = \{30.81\}$
- $\hat{f}(x_0 = 32) = 14.4$



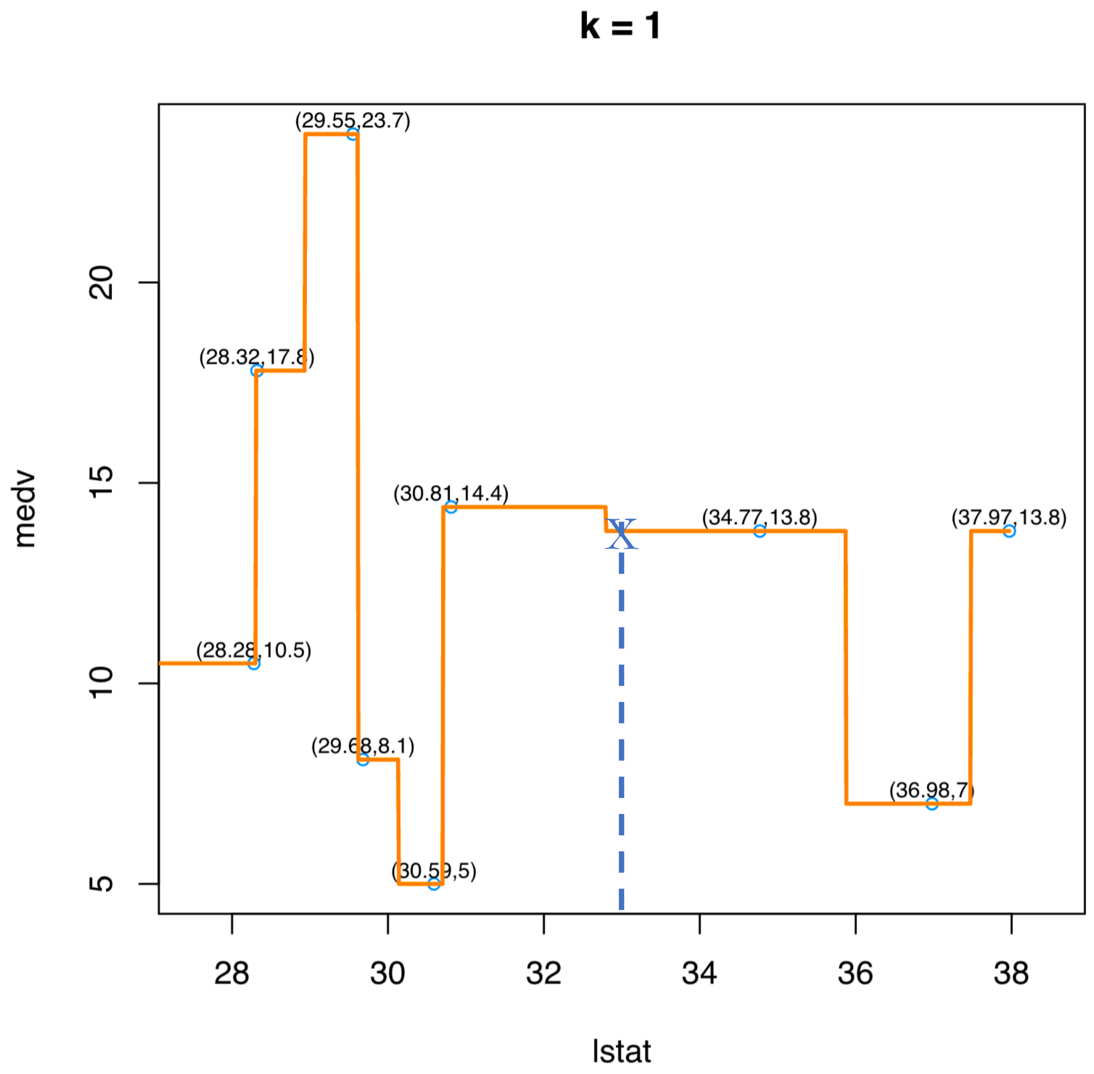
$\hat{f}(x_0)$ is a step function

- $x_0 = 32.79$: it is a switching point
- $N_K(x_0) = \{30.81\}$ or $N_K(x_0) = \{34.77\}$
  - Note that $32.79 - 30.81 = 1.98 = 34.77 - 32.79$
- $\hat{f}(x_0 = 32.79) = 14.4$ or $\hat{f}(x_0 = 32.79) = 13.8$



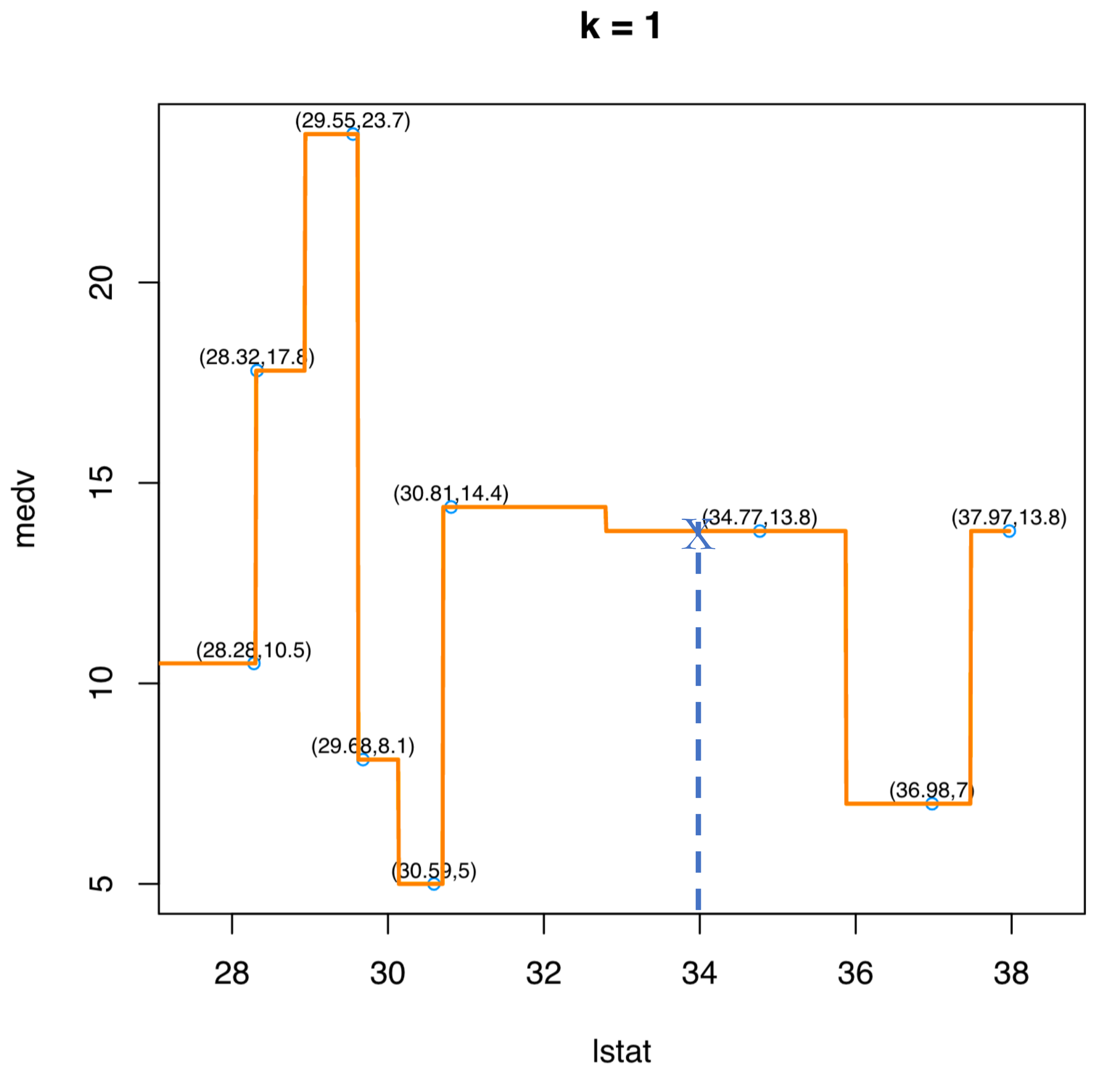
$\hat{f}(x_0)$ is a step function

- $x_0 = 33$
- $N_K(x_0) = \{34.77\}$
  - Note that $34.77 - 33 = 1.77 < 33 - 30.81 = 2.19$
- $\hat{f}(x_0 = 33) = 13.8$



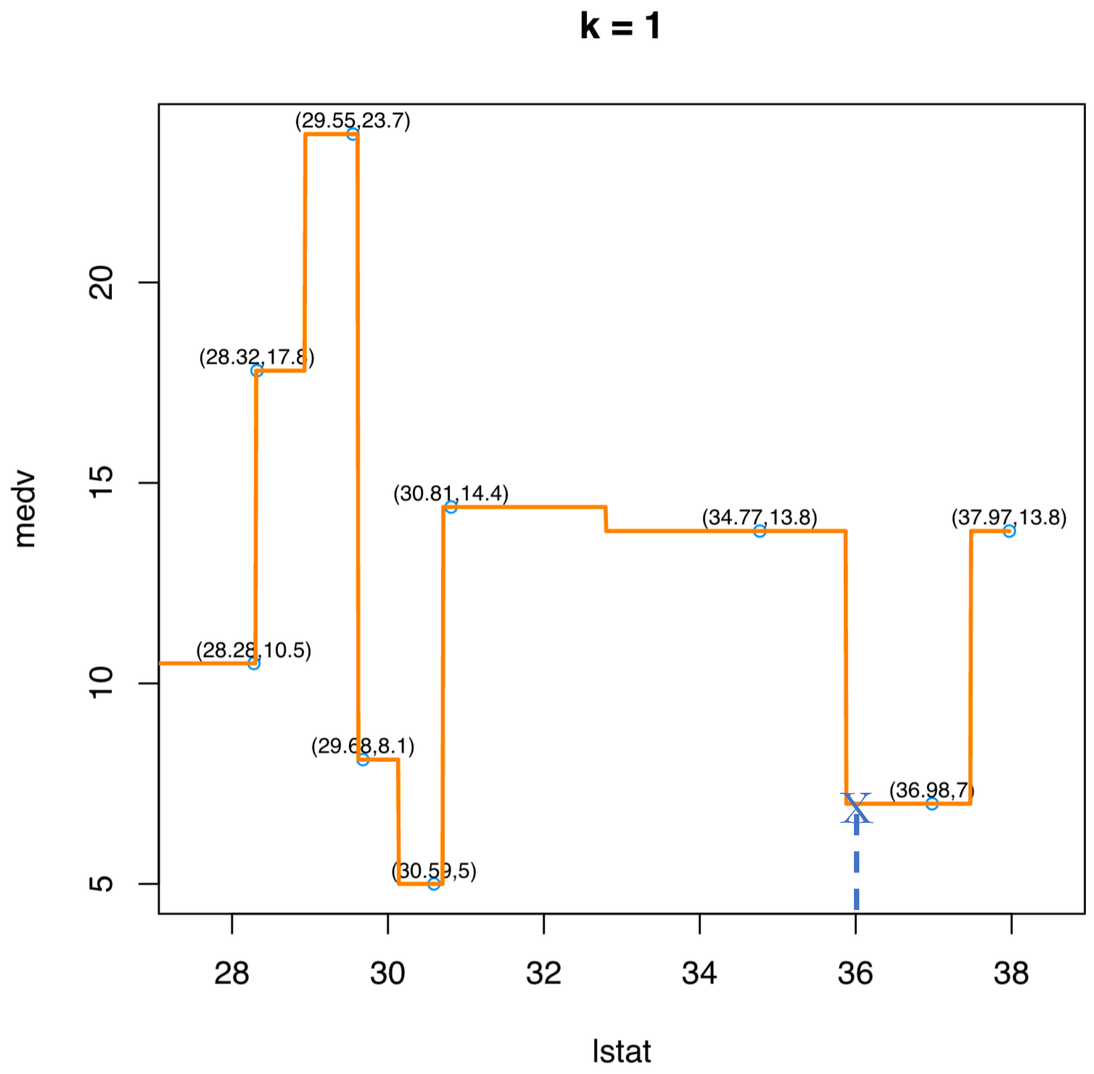
$\hat{f}(x_0)$ is a step function

- $x_0 = 34$
- $N_K(x_0) = \{34.77\}$
- $\hat{f}(x_0 = 34) = 13.8$



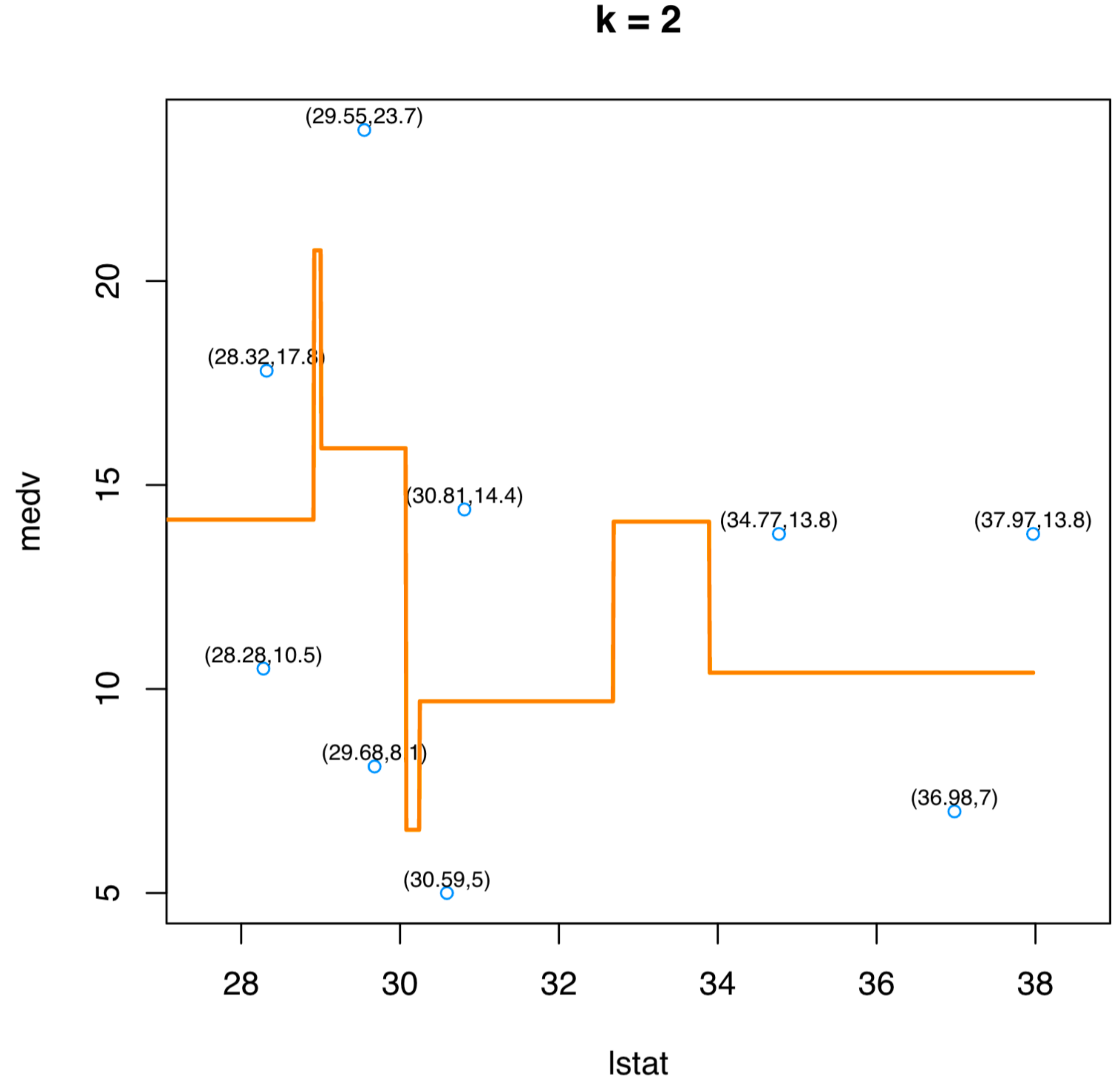
$\hat{f}(x_0)$ is a step function

- $x_0 = 36$
- $N_K(x_0) = \{36.98\}$
- $\hat{f}(x_0 = 36) = 7$



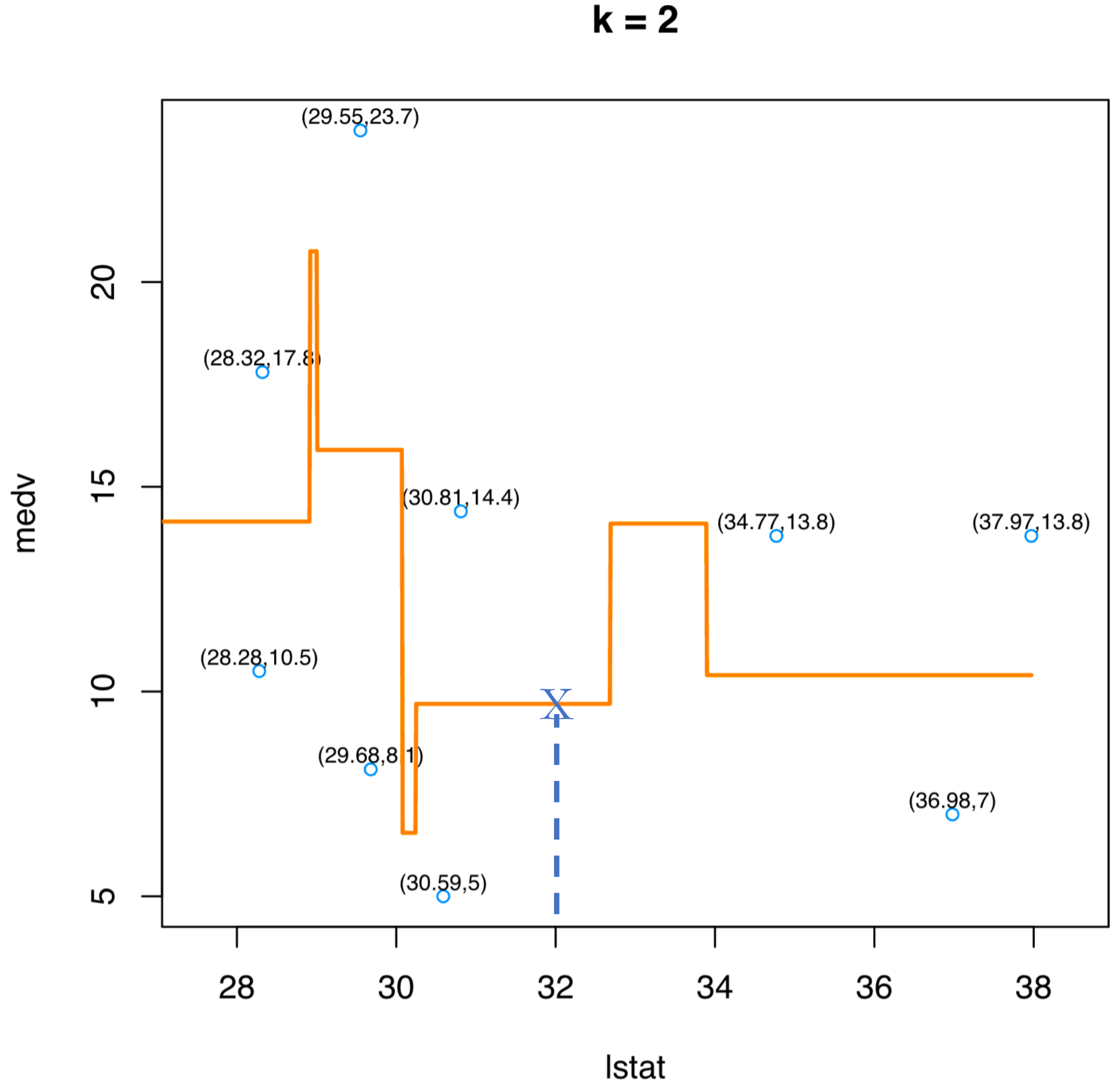
Example: 2-nearest neighbor regression

- $\hat{f}(x_0)$ equals to the average of responses of x_0's 2 nearest neighbors



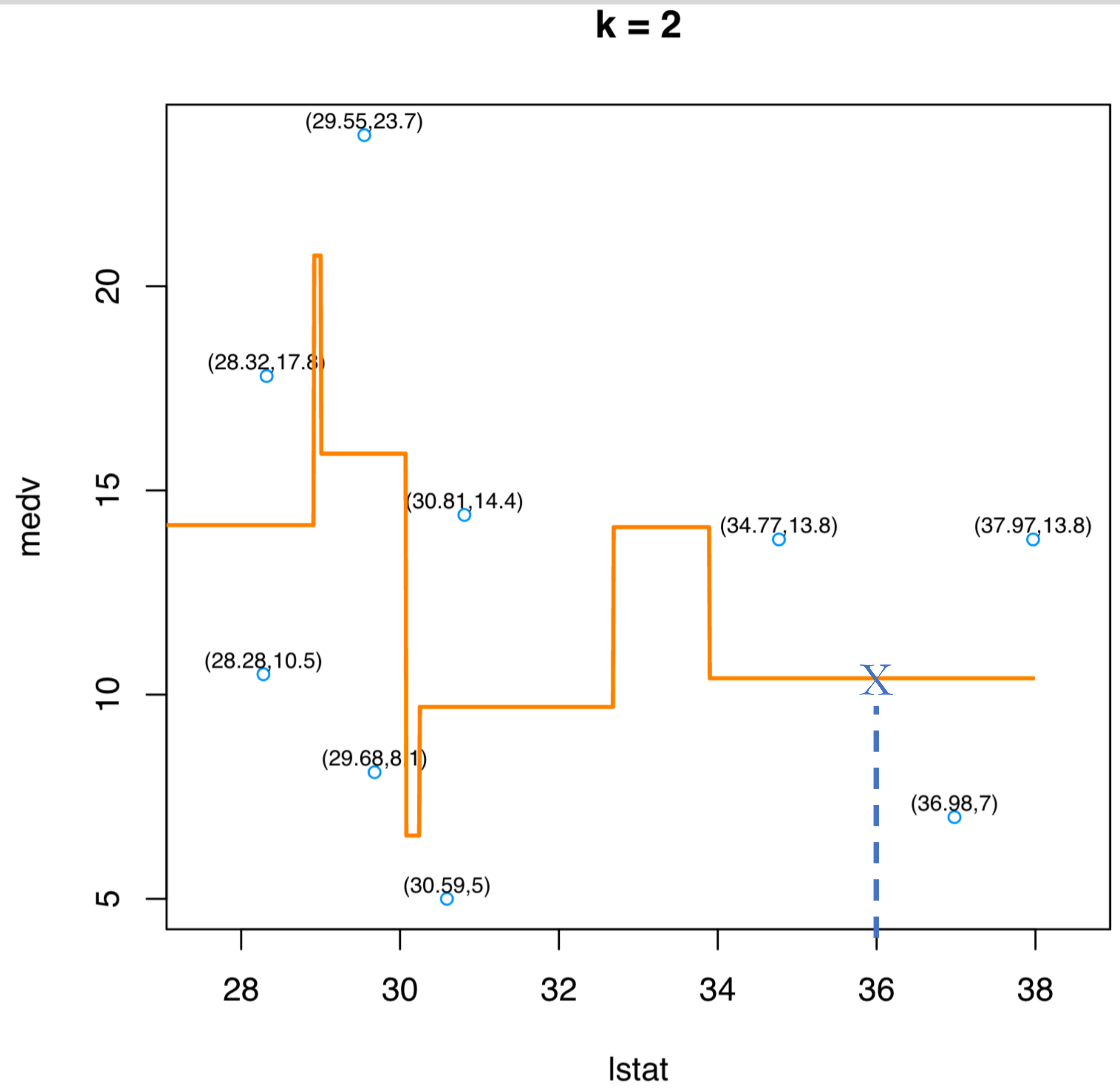
Example: 2-nearest neighbor regression

- $x_0 = 32$
- $N_K(x_0) = \{30.59, 30.81\}$
- $\hat{f}(x_0 = 32) = \frac{5 + 14.4}{2}$



Example: 2-nearest neighbor regression

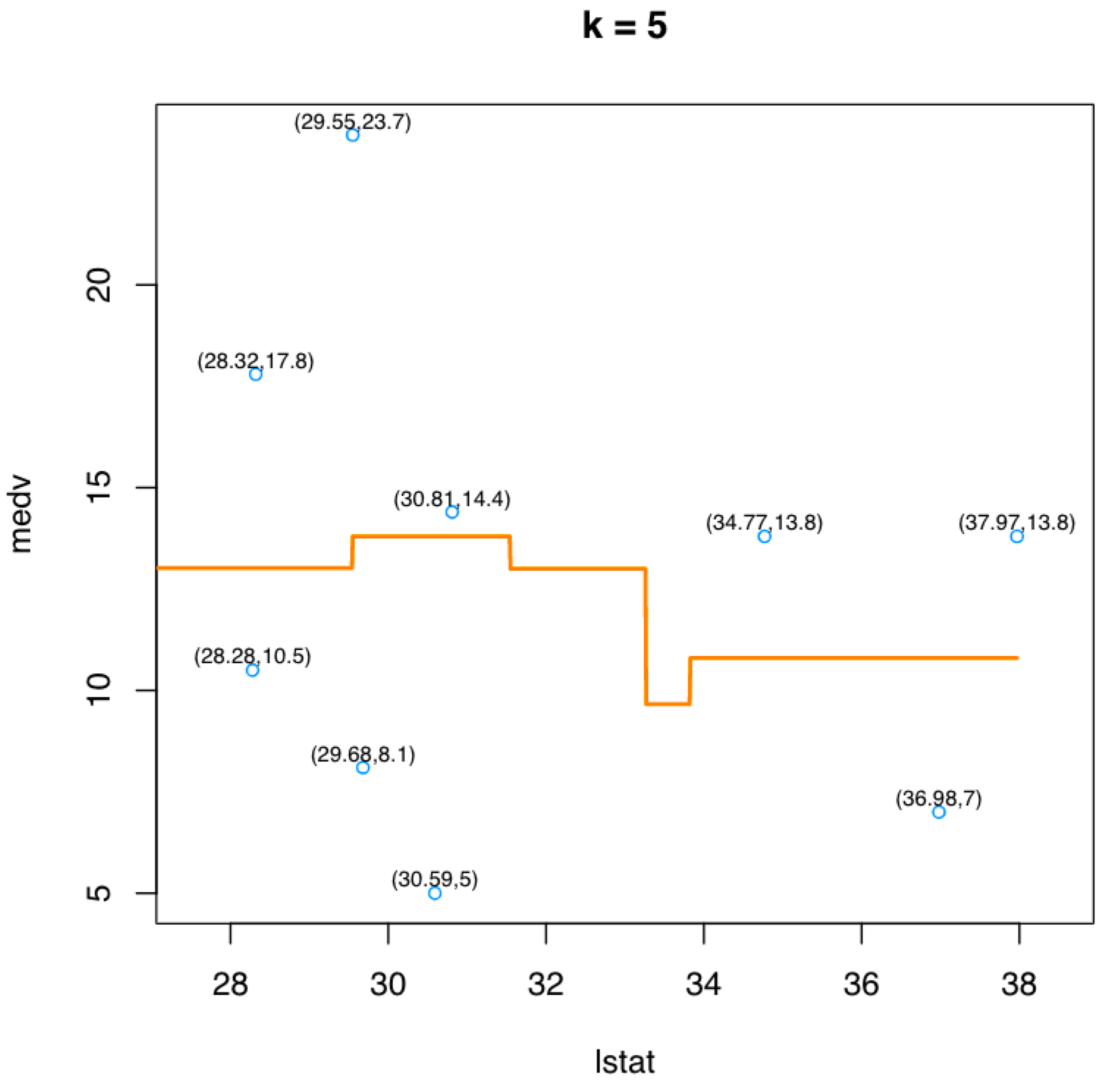
- $x_0 = 36$
- $N_K(x_0) = \{34.77, 36.98\}$
- $\hat{f}(x_0 = 36) = \frac{13.8+7}{2}$



Example: 5-nearest neighbor regression

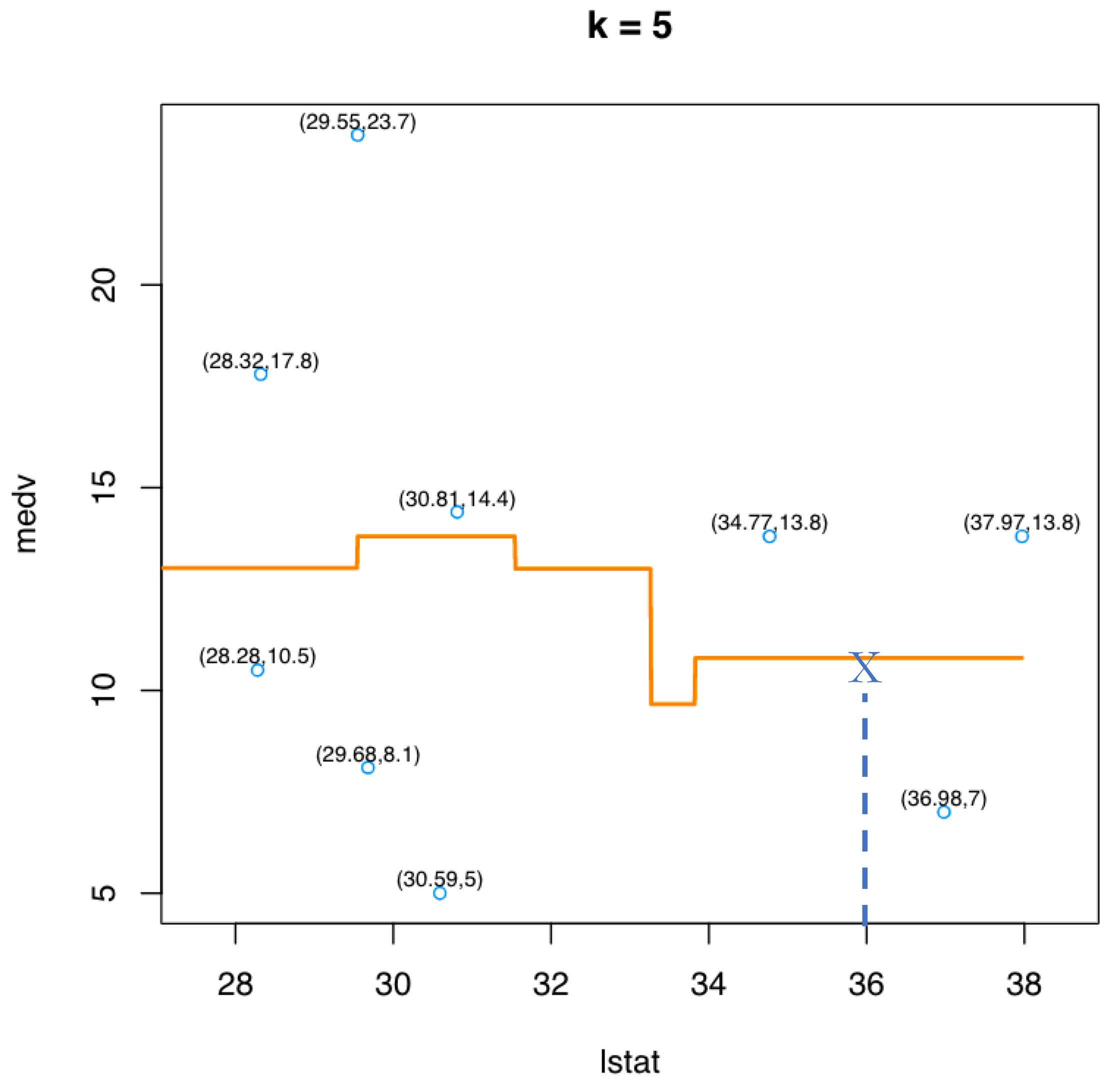
- $\hat{f}(x_0)$ equals to the average of responses of x_0's 5 nearest neighbors

- $\hat{f}(x_0)$ is smoother as K increases

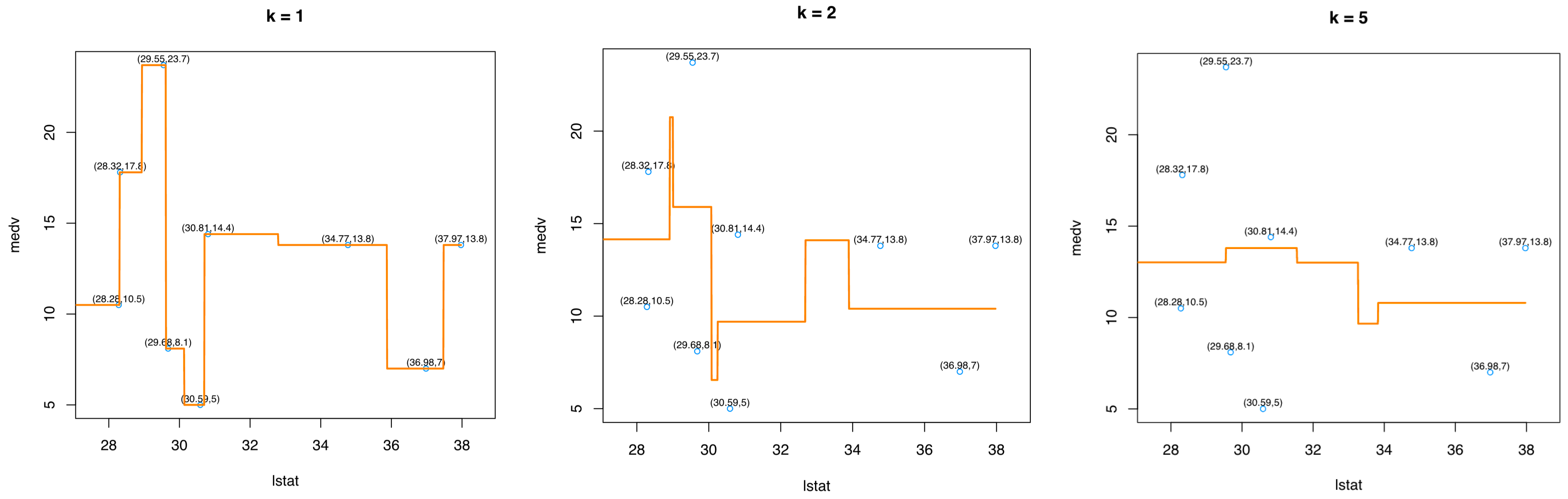


Example: 5-nearest neighbor regression

- $x_0 = 36$
- $N_K(x_0) = \{30.59, 30.81, 34.77, 36.98, 37.97\}$
- $\hat{f}(x_0 = 36) = \frac{5+14.4+13.8+7+13.8}{5}$



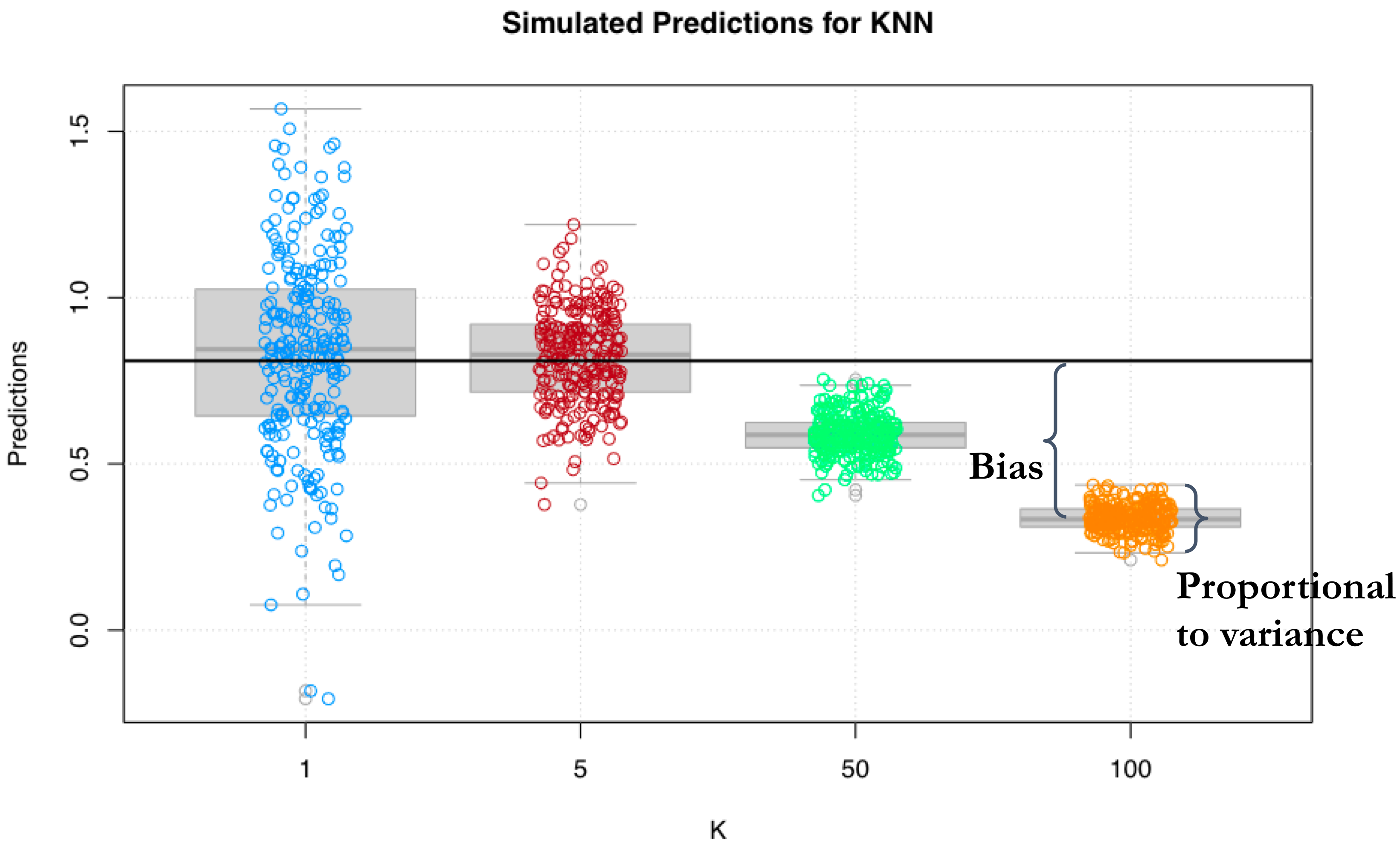
$\hat{f}(x_0)$ is smoother for a larger K



- **Question:** Is the model more flexible or less flexible for a larger K?

The bias-variance tradeoff

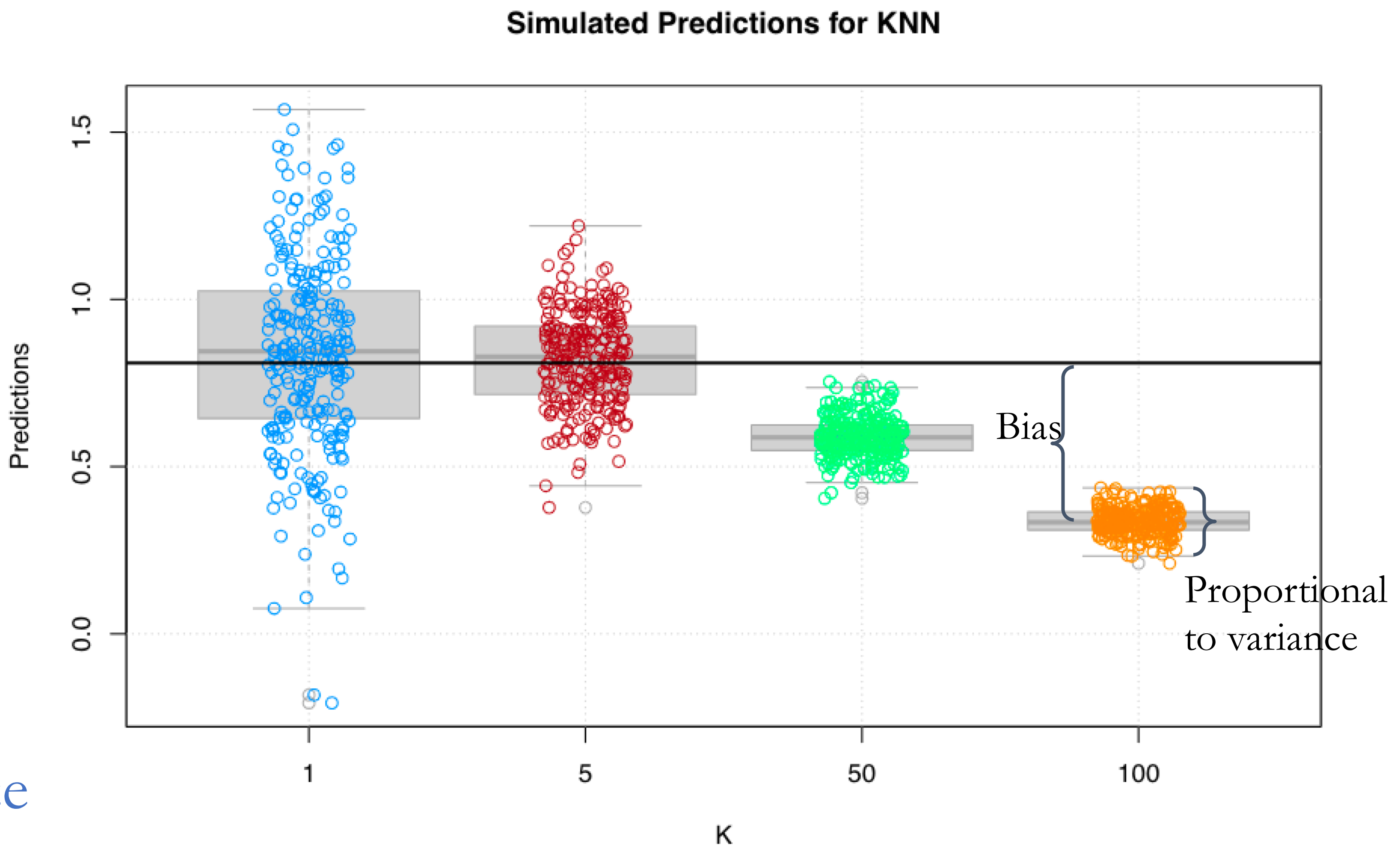
- Train a KNN model to learn the true function $f(x) = x^2$ (x is a scalar)
- $x_0 = 0.9$
- $y = f(0.9) = 0.81$
- 250 runs: for each dataset, we fit KNN with $K = 1, 5, 50, 100$, and plot $\hat{f}(0.9)$



The bias-variance tradeoff

- The square of bias
  $$\hat{f}_5(x) \approx \hat{f}_1(x) < \hat{f}_{50}(x) < \hat{f}_{100}(x)$$
- Increasing K increases bias

- Variance
  $$\hat{f}_{100}(x) < \hat{f}_{50}(x) < \hat{f}_5(x) < \hat{f}_1(x)$$
- Increasing K reduces variance



Reference

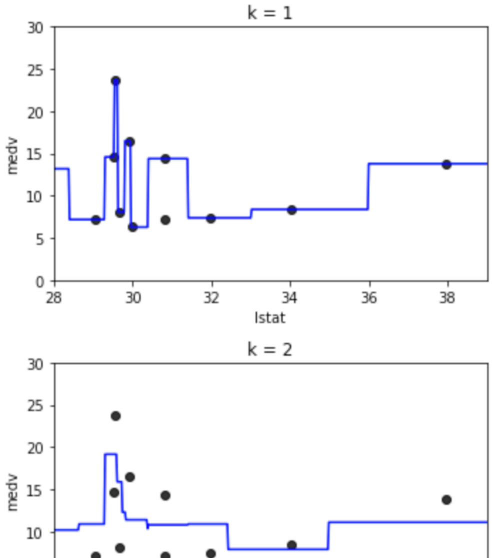
- Linear regression
  - In sklearn: **linear_model.LinearRegression**
    - See coding examples at https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
  - In statsmodels: OLS estimator
    - See coding examples at https://www.statsmodels.org/stable/regression.html, from which you can read off the standard errors to construct the confidence intervals

K-nearest neighbors regression in Python

Alternatively, weights = ‘distance’, where weight points by the inverse of their distance

```
In [3]: from sklearn.neighbors import KNeighborsRegressor
        import pandas as pd
        import seaborn as sns
```

```
In [4]: df_train = pd.DataFrame({'lstat': X_train.reshape(-1,), 'medv': y_train.reshape(-1,)})

        fig, axes = plt.subplots(3, 1, figsize = (5,10))

        n_neighbors = [1, 2, 5]

        T = np.linspace(28, 39, 500)[:, np.newaxis] # For graphing

        for i, n in enumerate(n_neighbors):
            knn = KNeighborsRegressor(n, weights = 'uniform')
            y_pred = knn.fit(X_train, y_train).predict(T)
            fit_df = pd.DataFrame({"T": T.reshape((-1,)), "y_pred": y_pred.reshape((-1,))})

            sns.lineplot(data = fit_df, x = 'T', y = 'y_pred', color = 'blue', ax = axes[i])
            sns.regplot(data = df_train, x = 'lstat', y = 'medv', ax = axes[i], fit_reg = False, scatter_kws={"color": "black"}

            axes[i].set_xlim([28, 39])
            axes[i].set_ylim([0, 30])


        fig.tight_layout()
```



Reference

Estimating the coefficients in Python

- sklearn.linear_model.LogisticRegression
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Announcements

- Homework one will be released in the afternoon—stay tuned on piazza!