

DATASCI 347 Machine Learning

Lecture 3: KNN

Ruoxuan Xiong

Suggested reading: ISL Chapters 2 and 4

Lecture plan

- K -nearest neighbors (KNN) regression
- K -nearest neighbors (KNN) classification



K -nearest neighbors regression

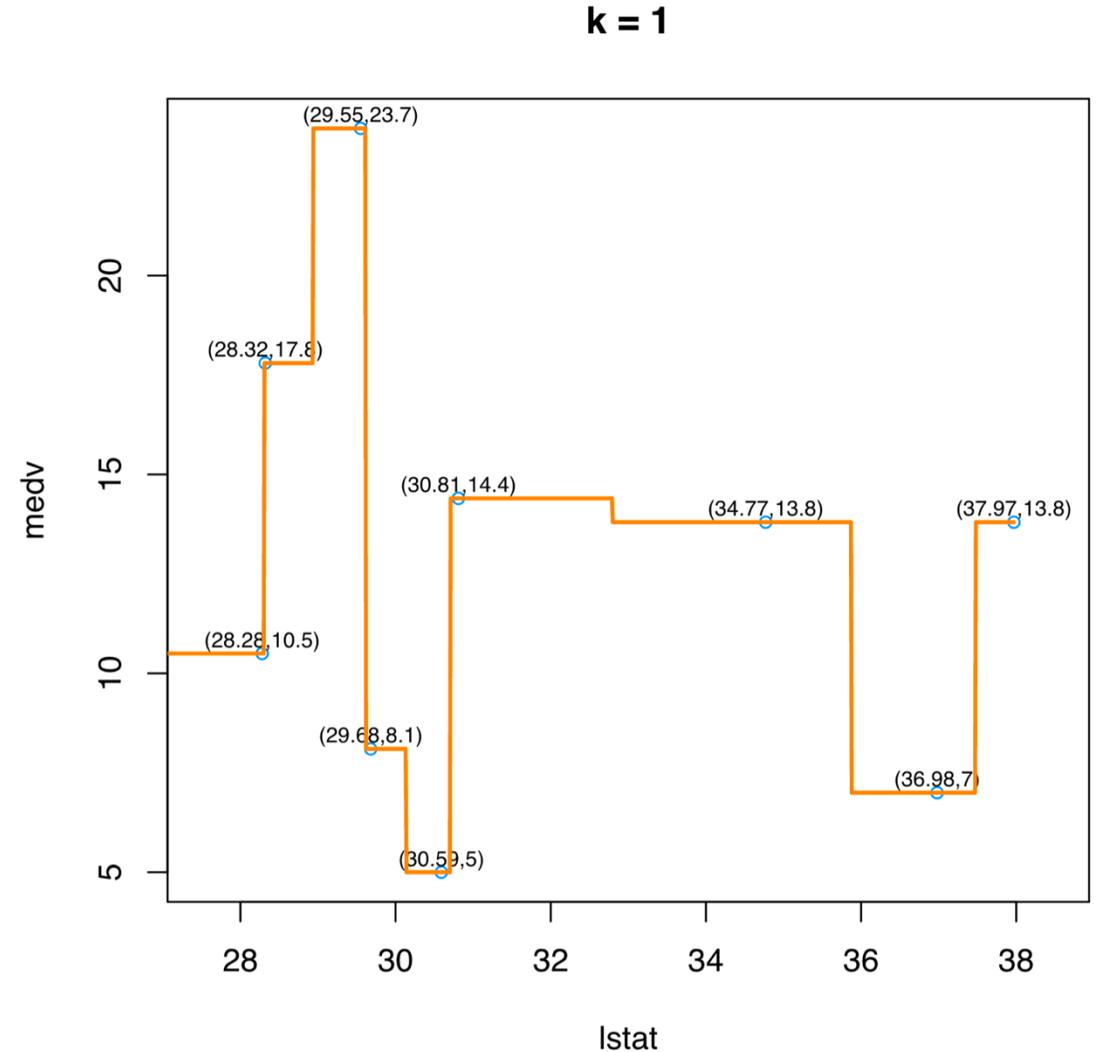
- A non-parametric approach
- K is a user-defined constant
 - K is an integer, e.g., 1,2,3, ...
- Given a value for K and a prediction point x_0 , $\hat{f}(x_0)$ is the average of the responses of K nearest neighbors

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

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

Example: 1-nearest neighbor regression

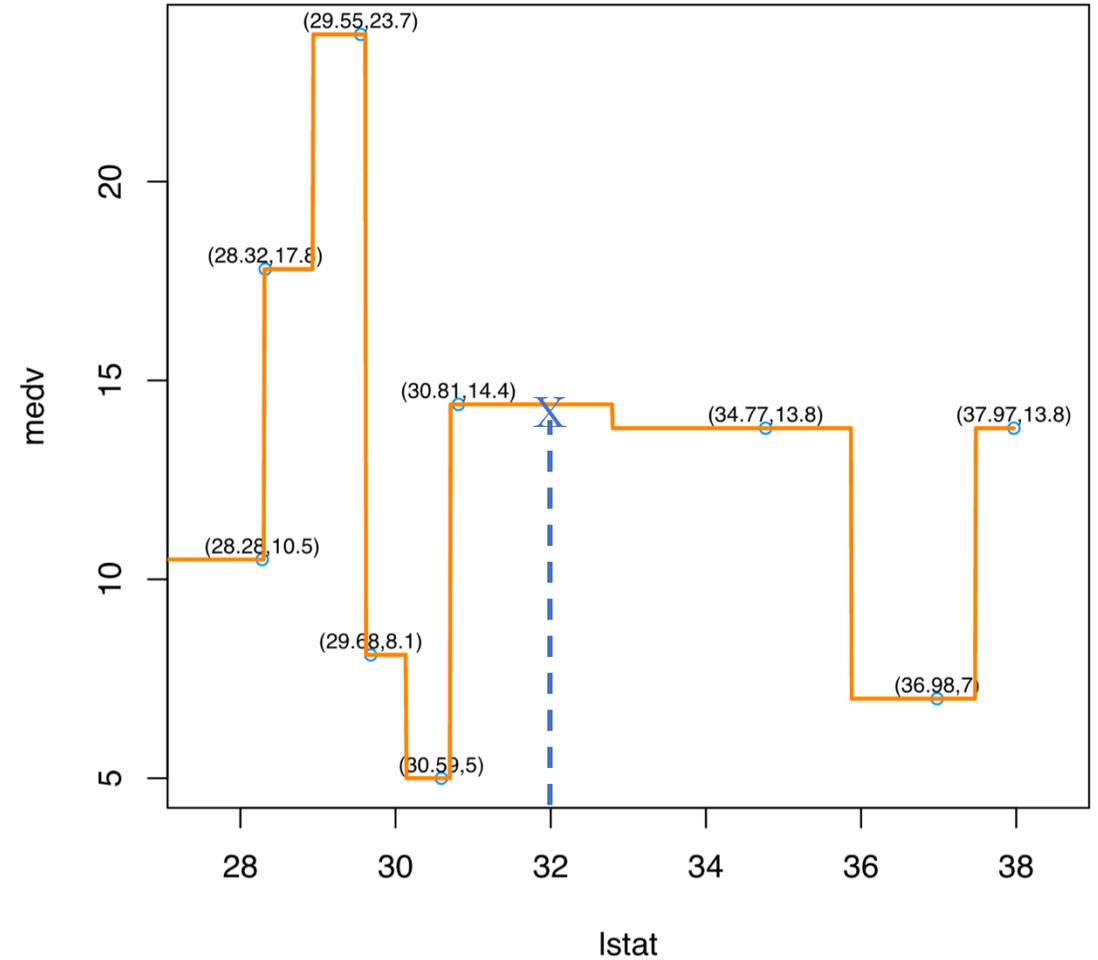
- Prediction of the median house value of a neighbor given the percentage of households with low socioeconomic status (*lstat*)
- Orange curve: $\hat{f}(x_0)$
 - $\hat{f}(x_0)$ equals to the response of x_0 's nearest neighbor
 - $\hat{f}(x_0)$ is a step function



Prediction at $x_0 = 32$ ($K = 1$)

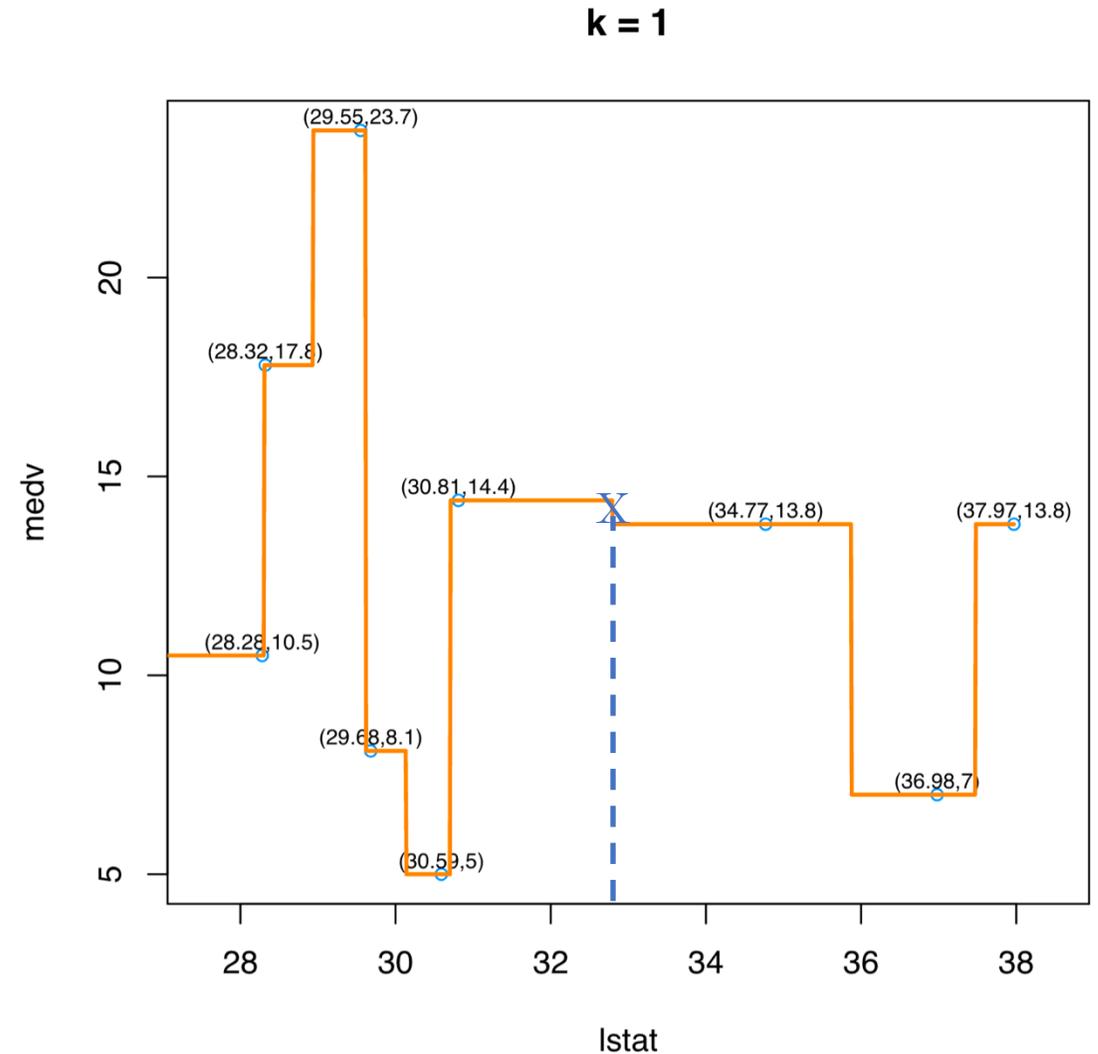
$k = 1$

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



Now let us understand why $\hat{f}(x_0)$ is a step function

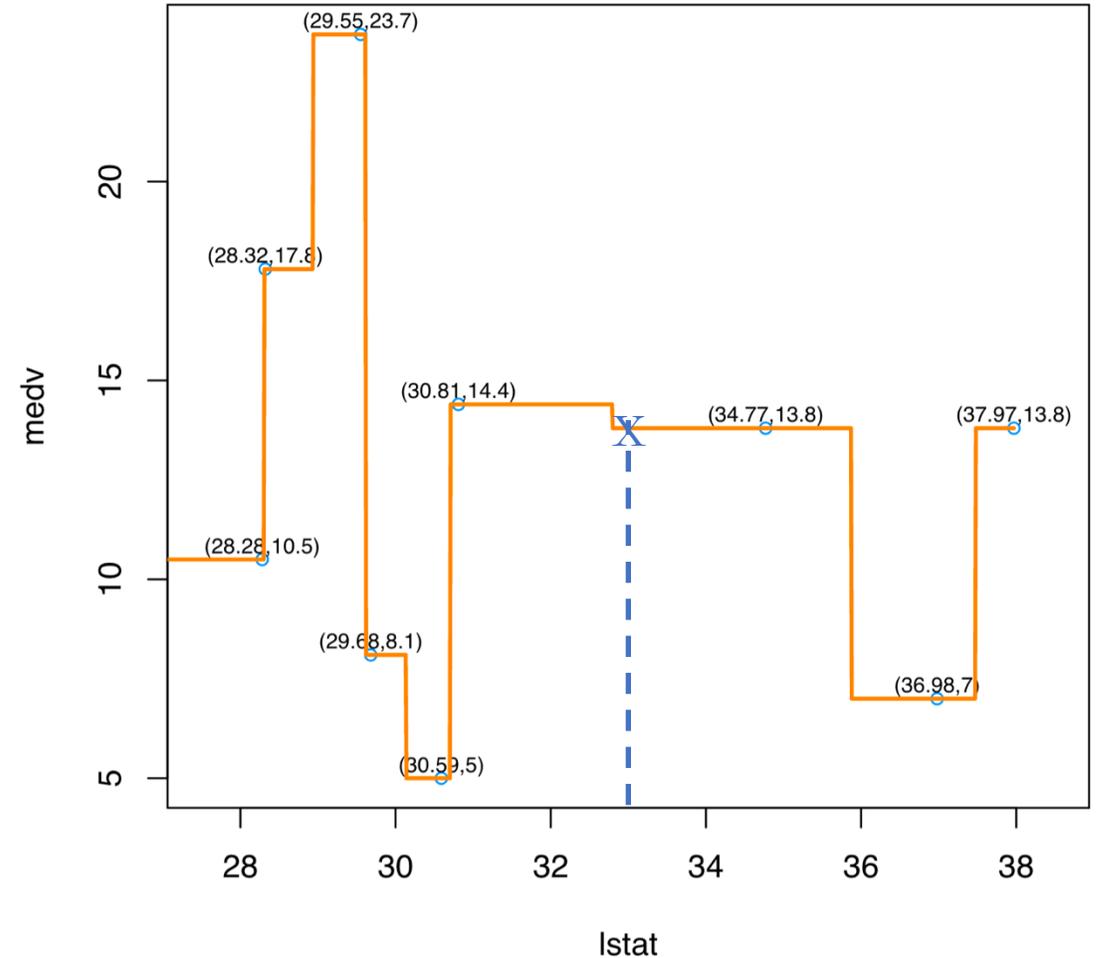
- $x_0 = 32.79$
 - It is a switching point!
- $N_K(x_0) = \{30.81\}$ or $N_K(x_0) = \{34.77\}$
 - Note that $32.79 - 30.81 = 1.98 = 34.77 - 32.79$
- $\hat{f}(x_0 = 32.79) = 14.4$ or $\hat{f}(x_0 = 32.79) = 13.8$



Prediction at $x_0 = 33$ ($K = 1$)

$k = 1$

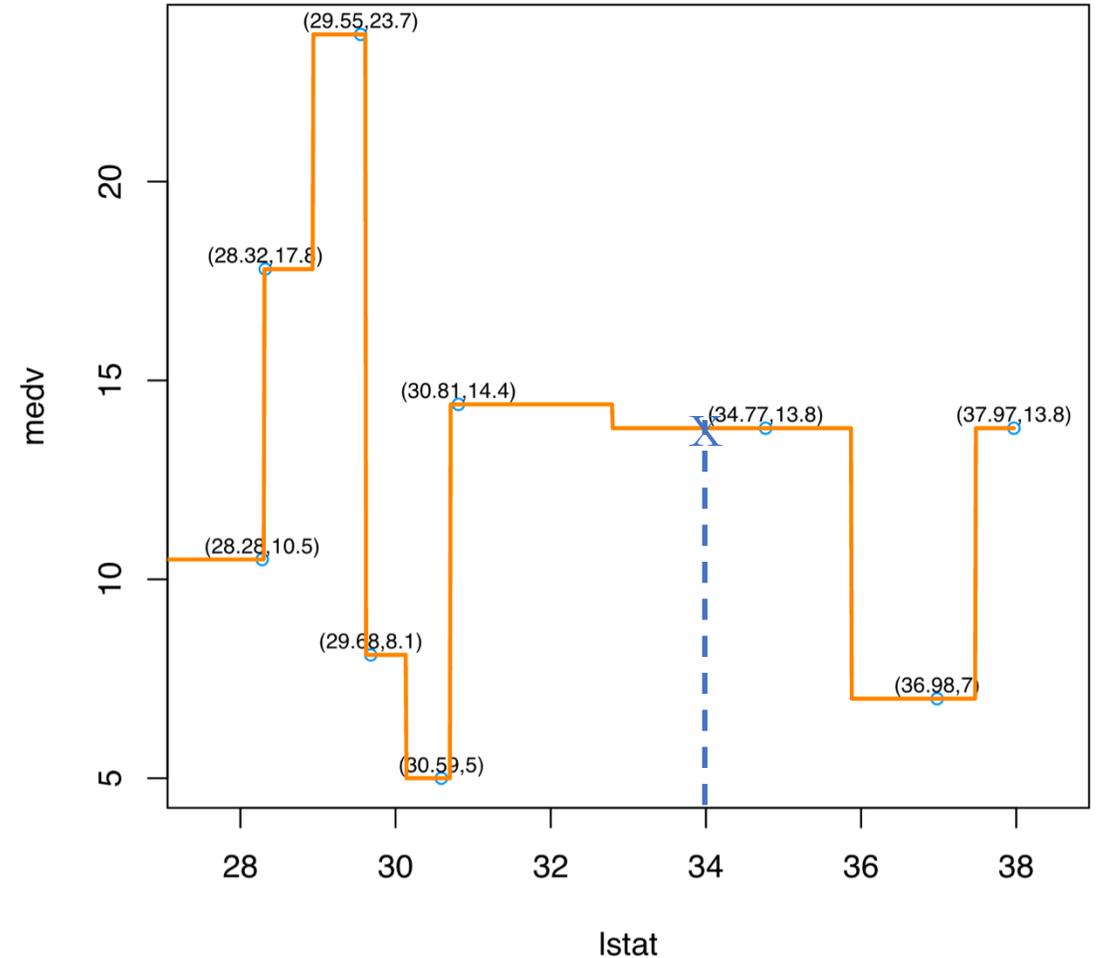
- $x_0 = 33$
- $N_K(x_0) = \{34.77\}$
 - Note that $34.77 - 33 = 1.77 < 33 - 30.81 = 2.19$
- $\hat{f}(x_0 = 33) = 13.8$



Prediction at $x_0 = 34$ ($K = 1$)

$k = 1$

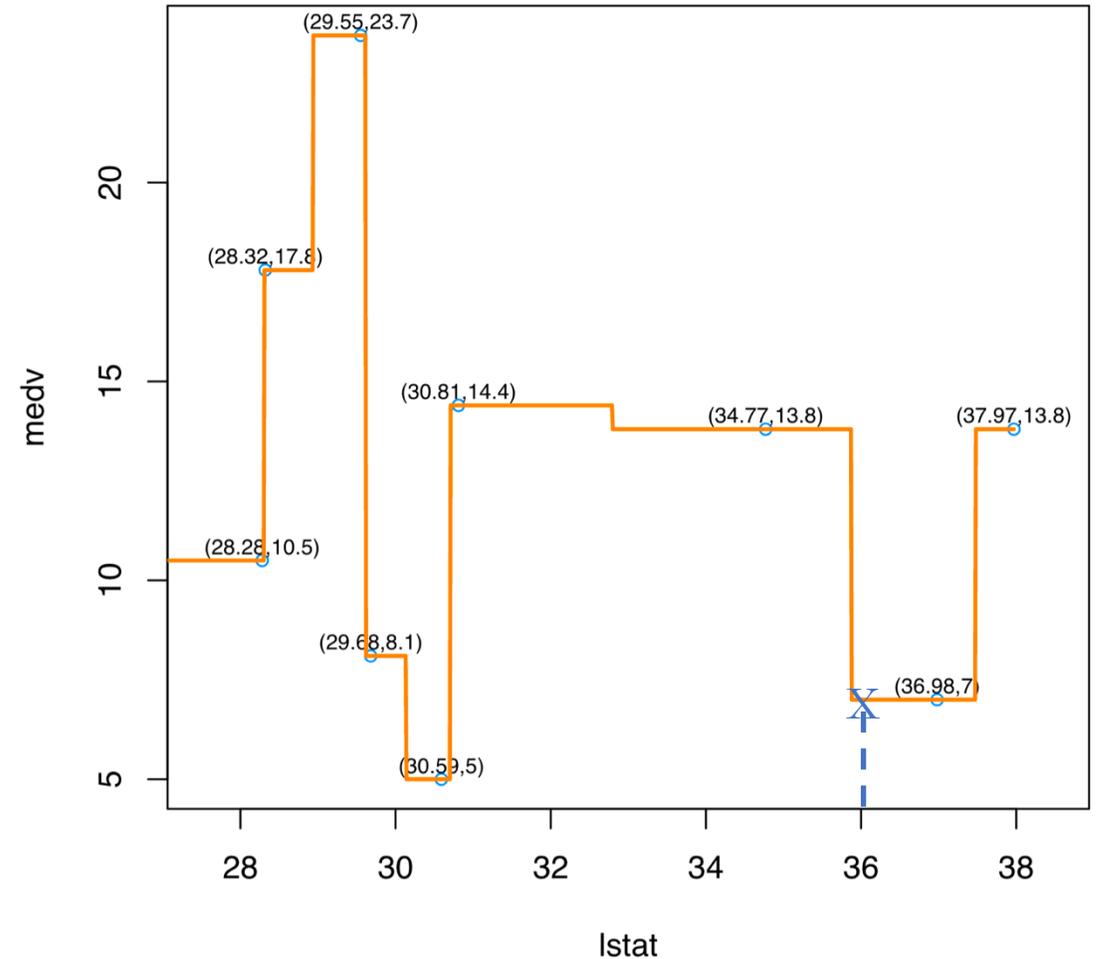
- $x_0 = 34$
- $N_K(x_0) = \{34.77\}$
- $\hat{f}(x_0 = 34) = 13.8$



Prediction at $x_0 = 36$ ($K = 1$)

$k = 1$

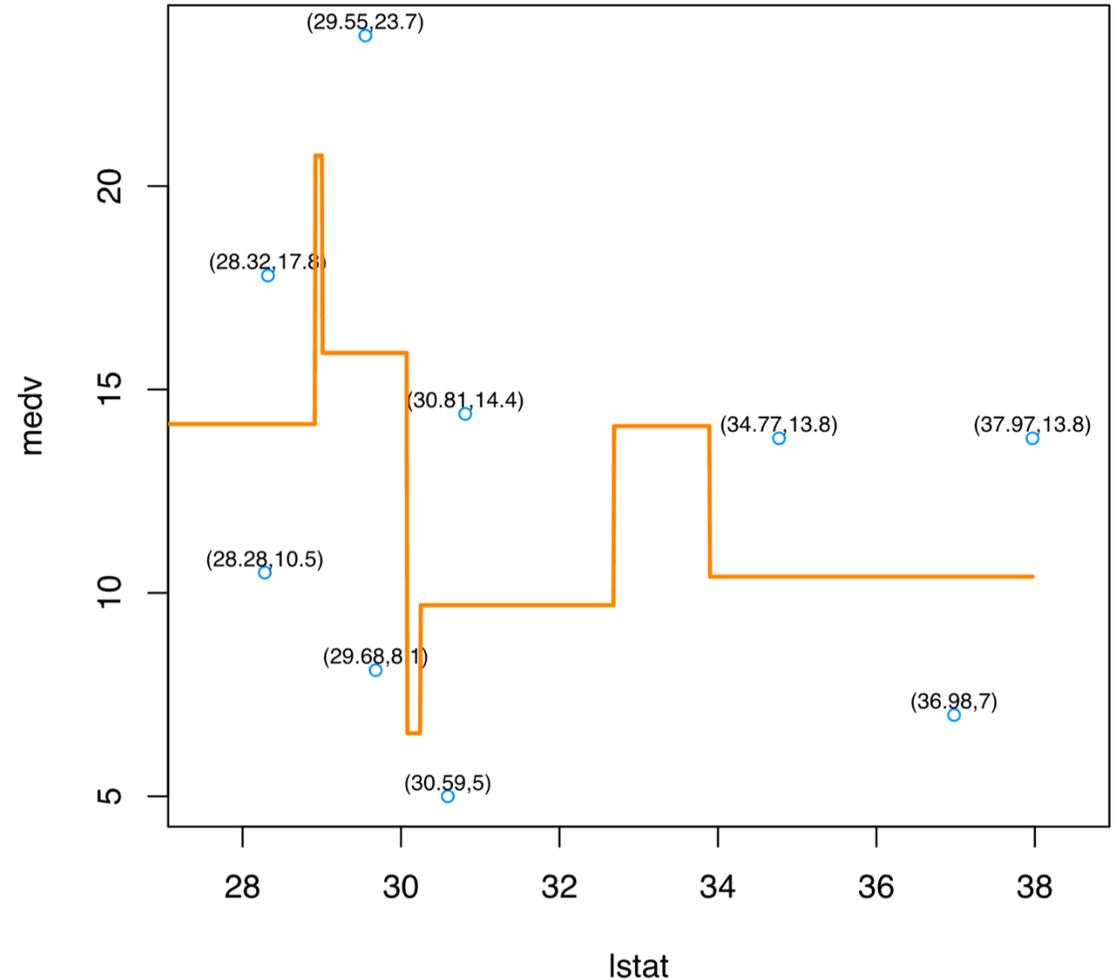
- $x_0 = 36$
- $N_K(x_0) = \{36.98\}$
- $\hat{f}(x_0 = 36) = 7$



Example: 2-nearest neighbor regression

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

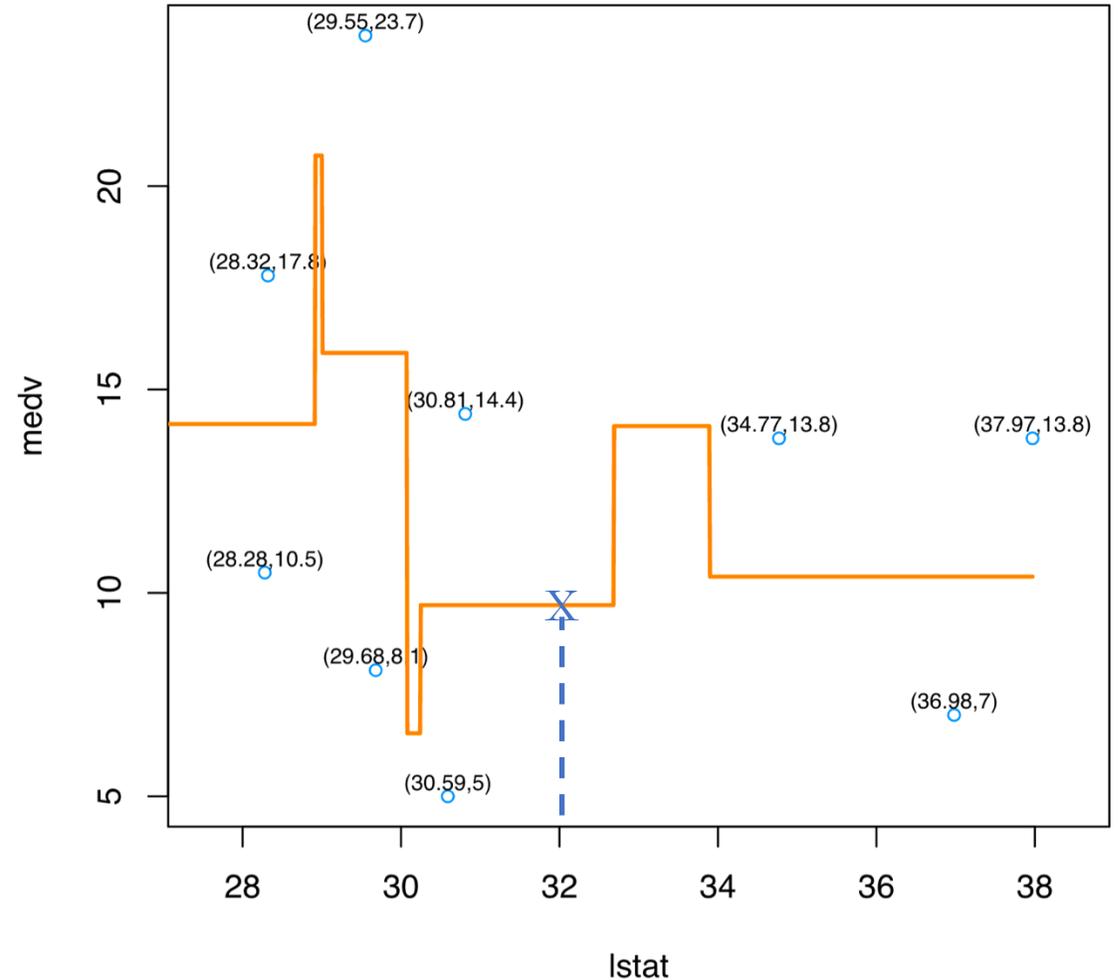
$k = 2$



Prediction at $x_0 = 32$ ($K = 2$)

$k = 2$

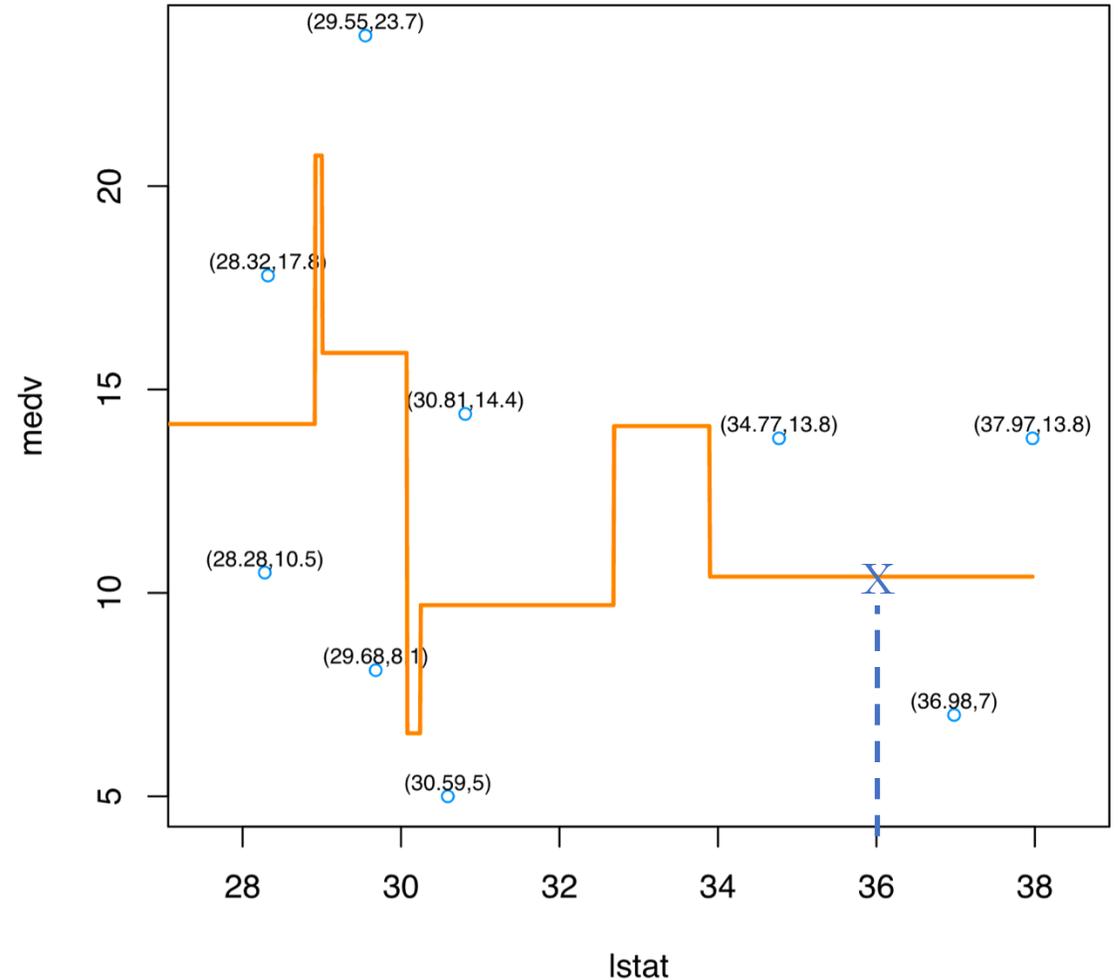
- $x_0 = 32$
- $N_K(x_0) = \{30.59, 30.81\}$
- $\hat{f}(x_0 = 32) = \frac{5 + 14.4}{2}$



Prediction at $x_0 = 36$ ($K = 2$)

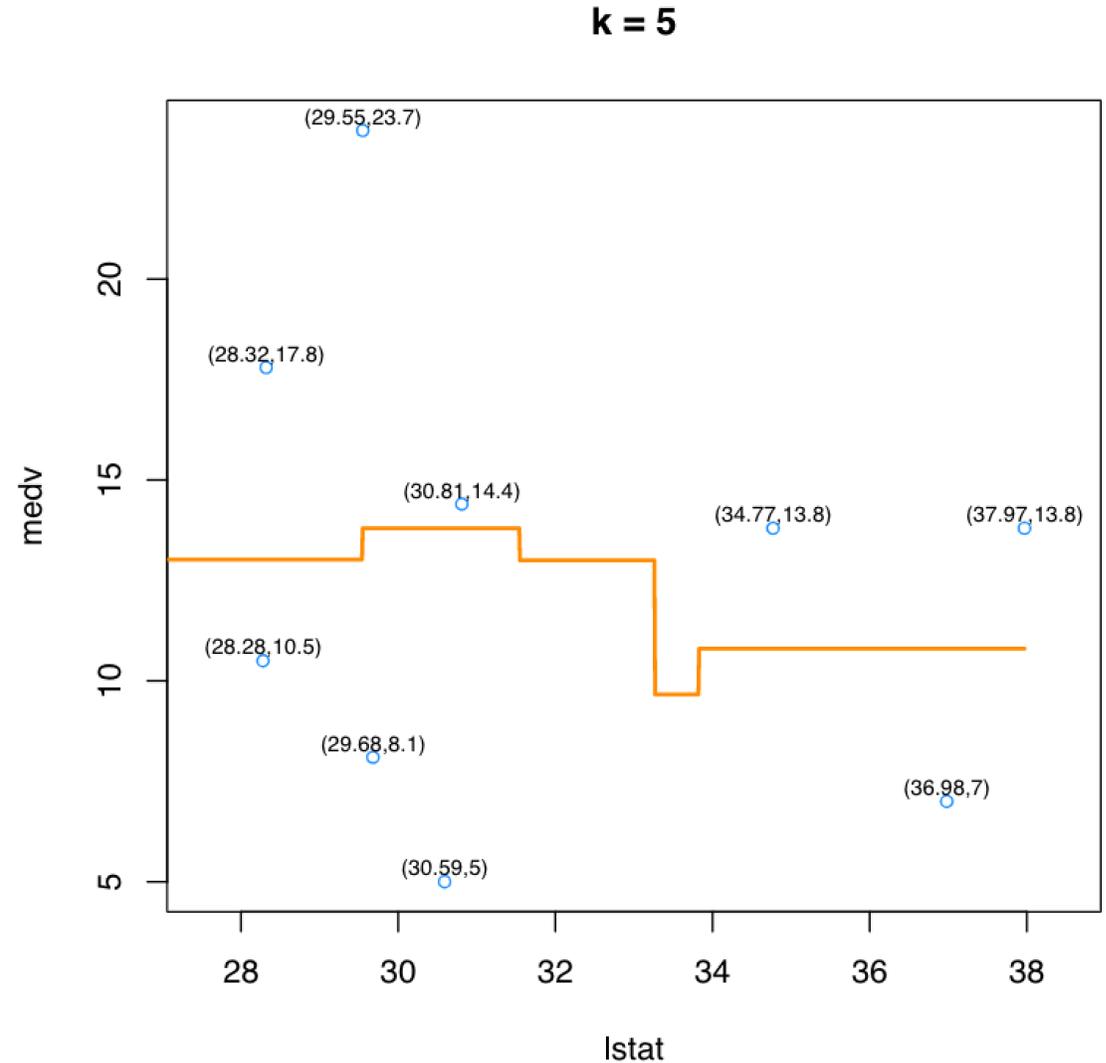
$k = 2$

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



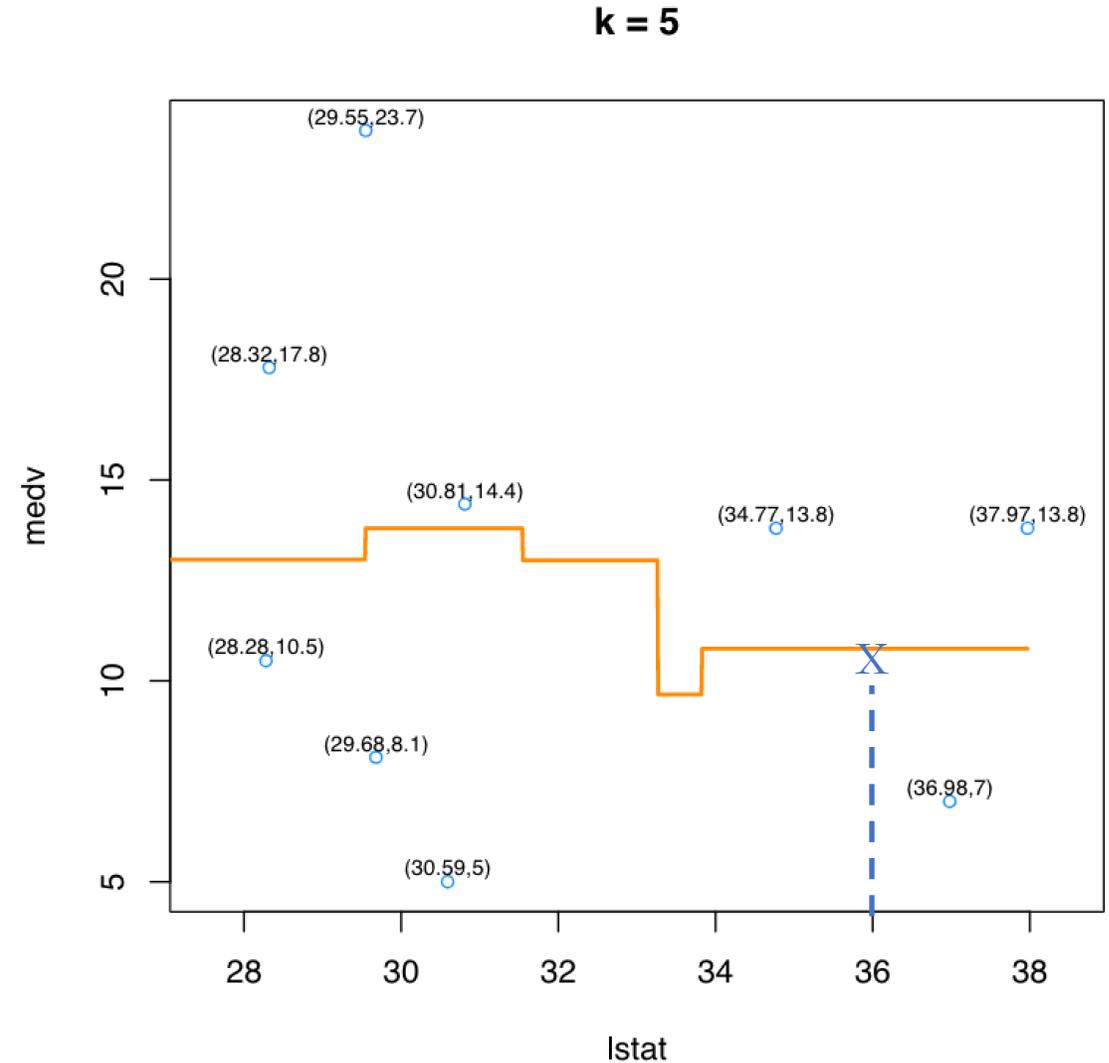
Example: 5-nearest neighbor regression

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

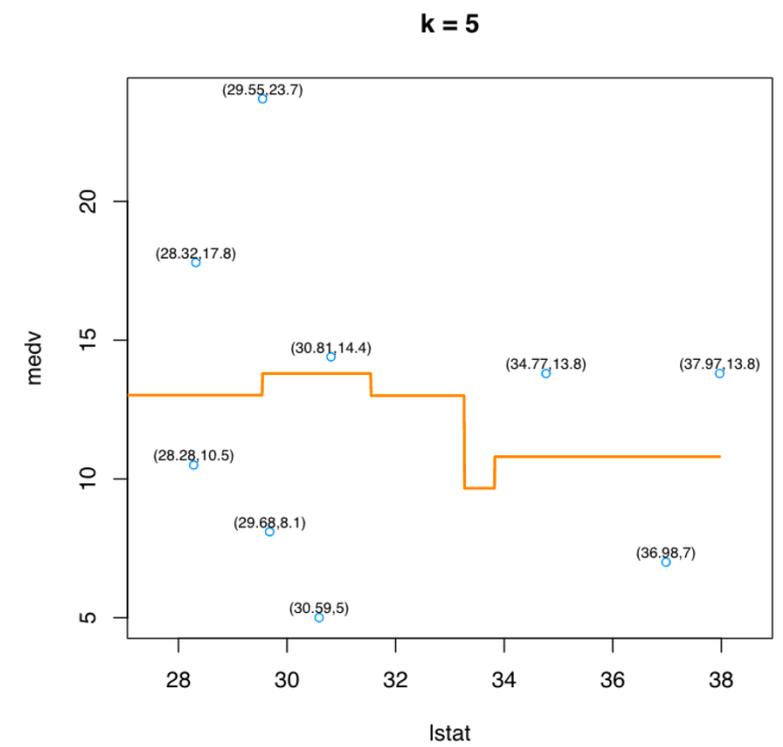
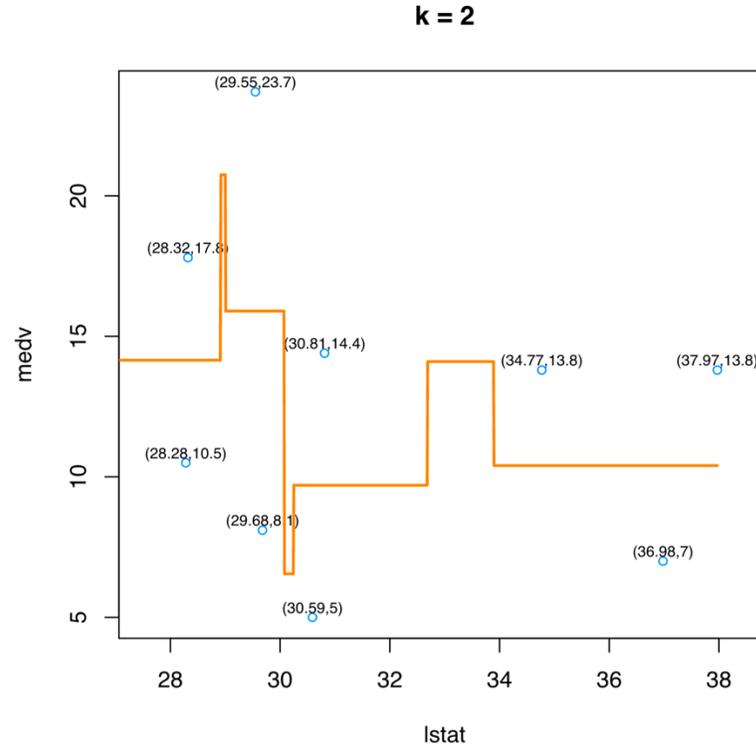
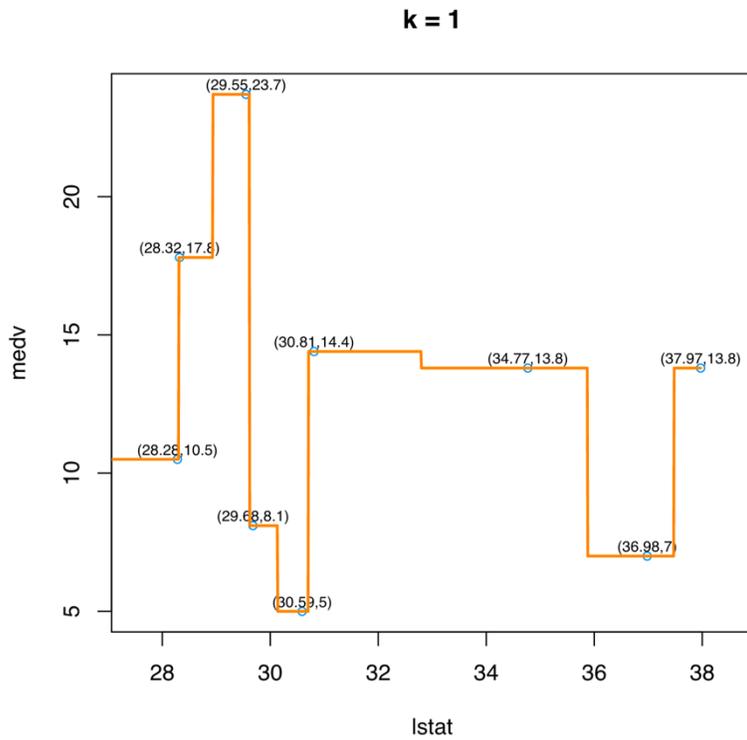


Prediction at $x_0 = 36$ ($K = 5$)

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



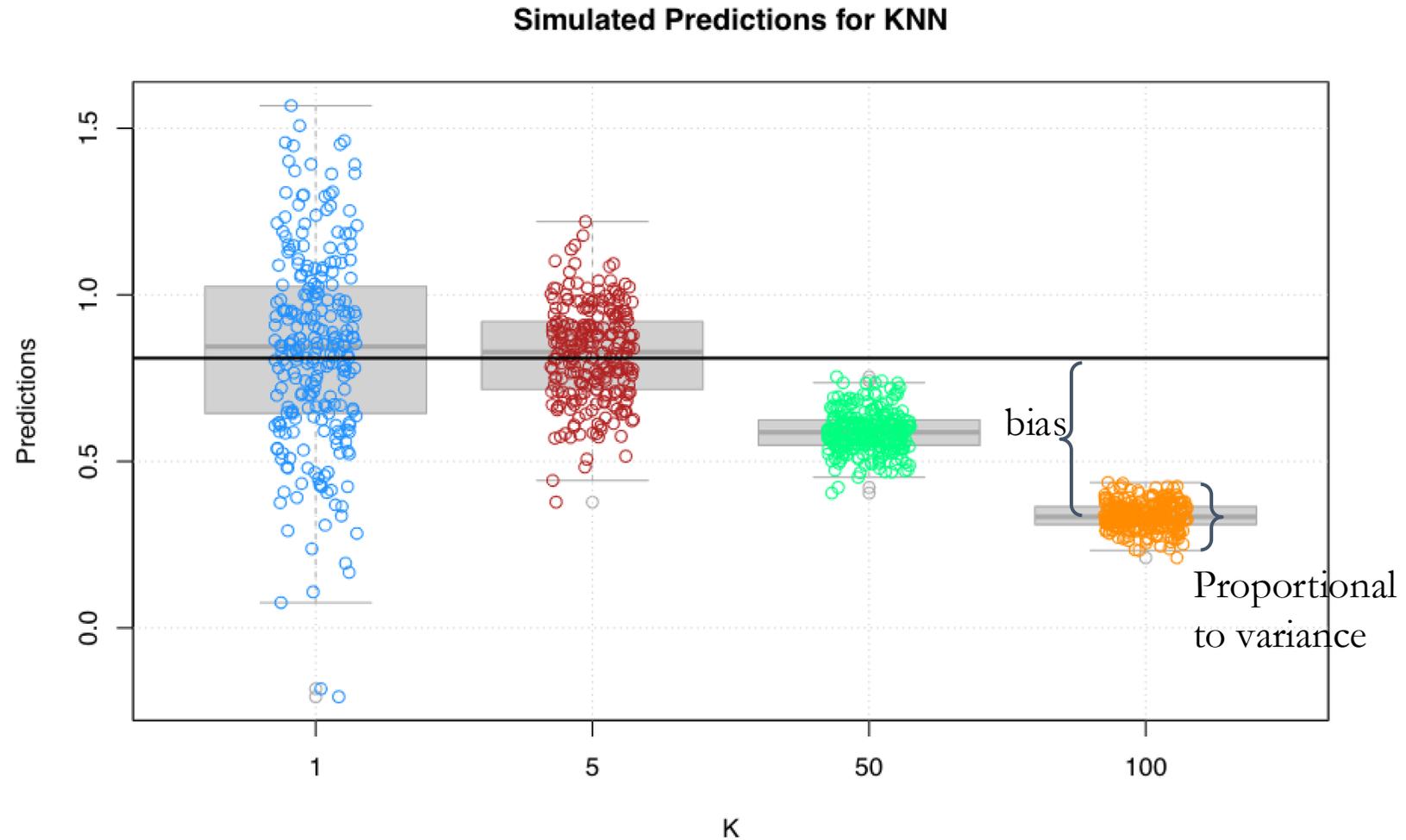
$\hat{f}(x_0)$ is smoother for a larger K



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

Bias-variance tradeoff for the optimal

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



Bias-variance tradeoff for the optimal

- The square of bias

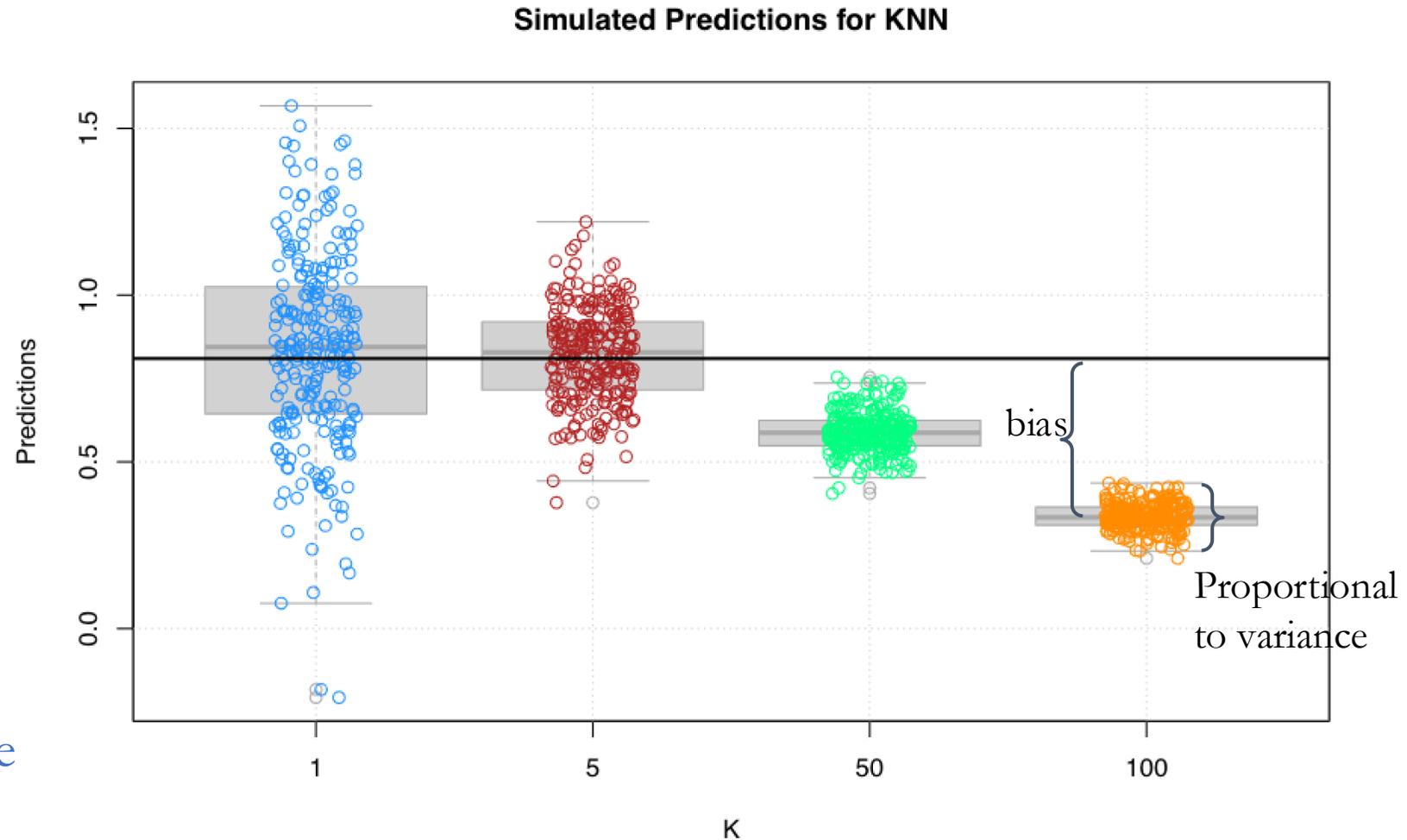
$$\hat{f}_5(x) \approx \hat{f}_1(x) < \hat{f}_{50}(x) < \hat{f}_{100}(x)$$

➤ 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



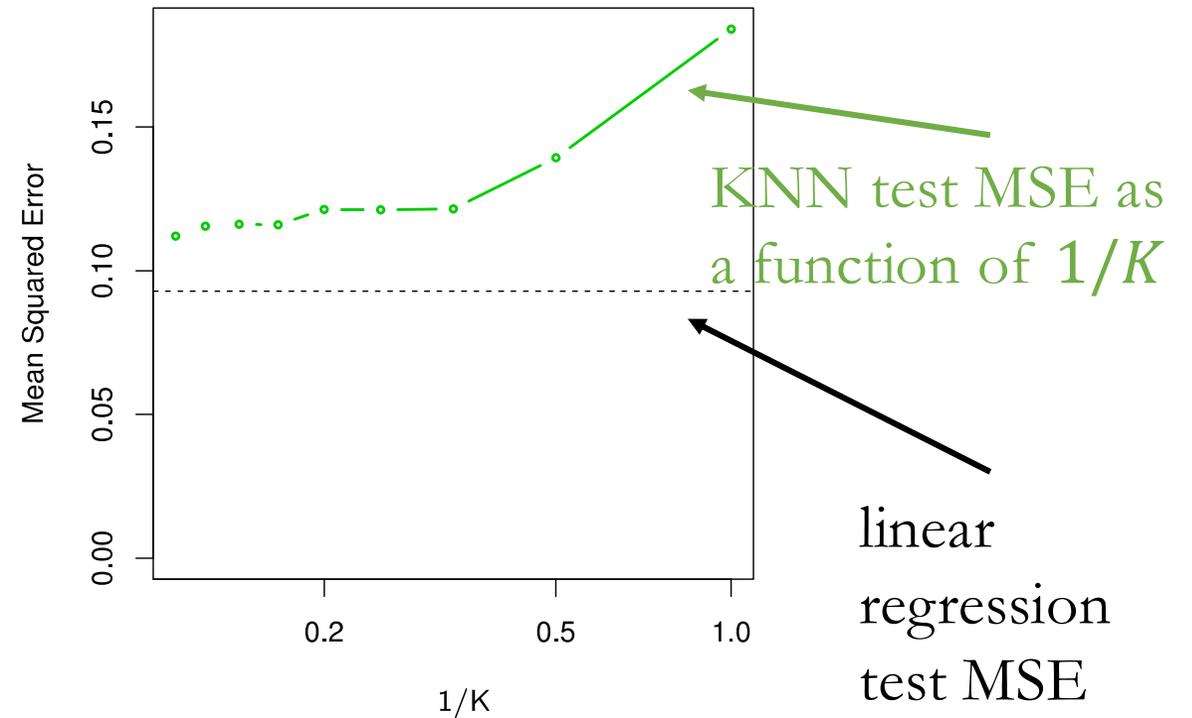
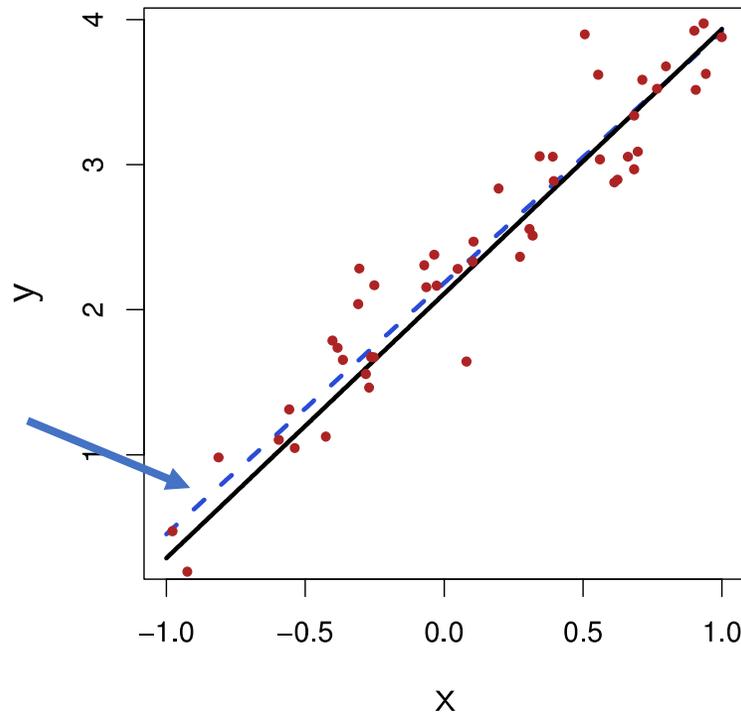
Linear regression vs K -nearest neighbors

- KNN is only **better** when the function f is **far from linear** (in which case linear model is *misspecified*)
- When n is **not much larger than p** , even if f is nonlinear, linear regression can outperform KNN
- KNN has **smaller bias**, but this comes at a price of **higher variance**

Linear models can dominate KNN

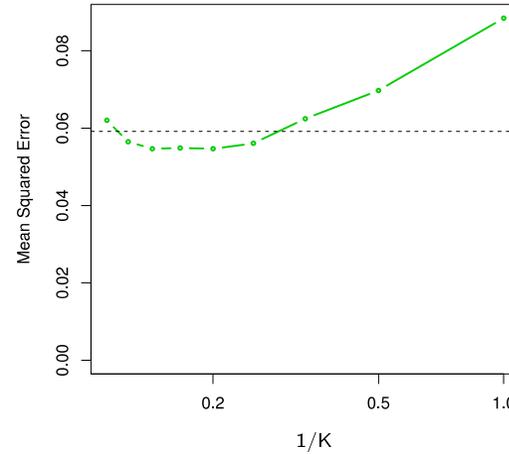
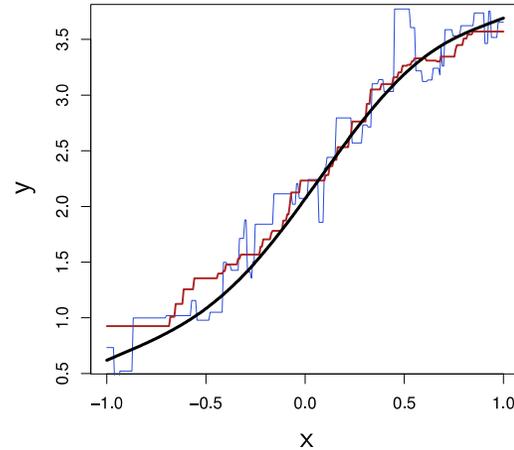
- Truth is linear, plot of test MSE vs. $1/K$ shows KNN worse than linear regression.

linear
regression



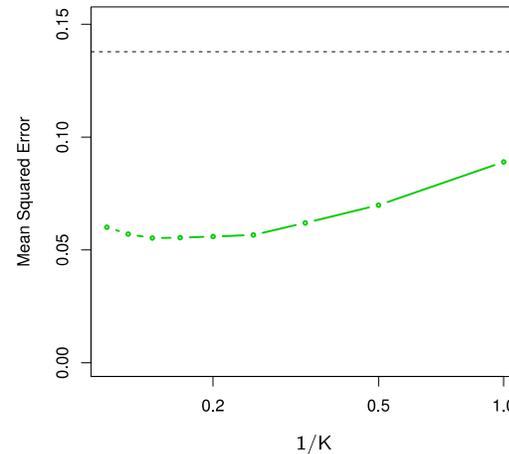
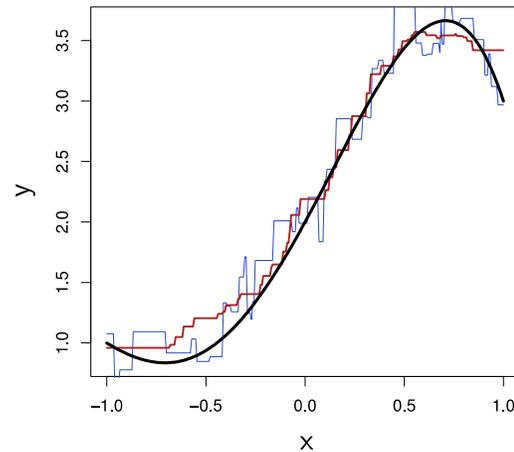
Increasing deviations from linearity

Slightly nonlinear relationship between X and Y



Blue: $K = 1$
Red: $K = 9$

Strongly nonlinear relationship between X and Y



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]

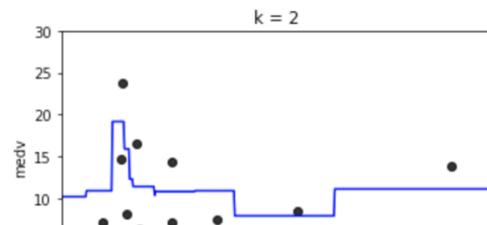
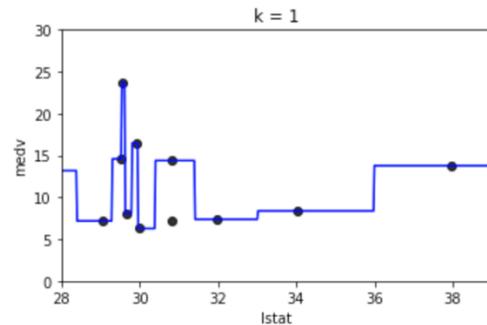
T = np.linspace(28, 39, 500)[:, np.newaxis] # For graphing

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()
```



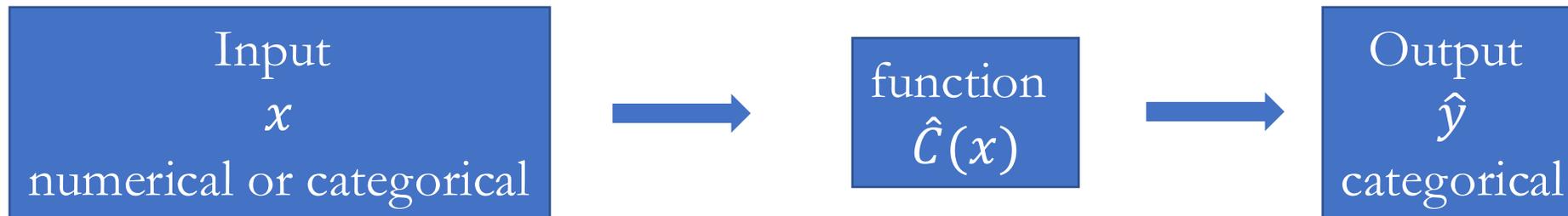
Lecture plan

- K -nearest neighbors (KNN) regression
- K -nearest neighbors (KNN) classification



Classification problem

- Classification is a form of supervised machine learning
- The response variable Y is **categorical**, as opposed to numerical for regression
- Our goal is to find a function \mathcal{C} which takes feature(s), x , as input, and outputs a **category** which is the same as the **true category** as frequently as possible



K -nearest neighbors classification

- Given a value for K and a prediction point x_0
 - The predicted probability of class g is the fraction of responses of K nearest neighbors in class g

$$\hat{P}(Y = g|X = x_0) = \frac{1}{K} \sum_{x_i \in N_K(x_0)} 1(y_i = g)$$

- $1(\cdot)$: indicator function
 - $N_K(x_0)$ is the set of K training observations that are closest to x_0
- The predicted class is the class with maximum predicted probability

$$\hat{C}(x_0) = \operatorname{argmax}_g \hat{P}(Y = g|X = x)$$

K -nearest neighbors classification

- Given a value for K and a prediction point x_0
 - The predicted probability of class g is the fraction of responses of K nearest neighbors in class g

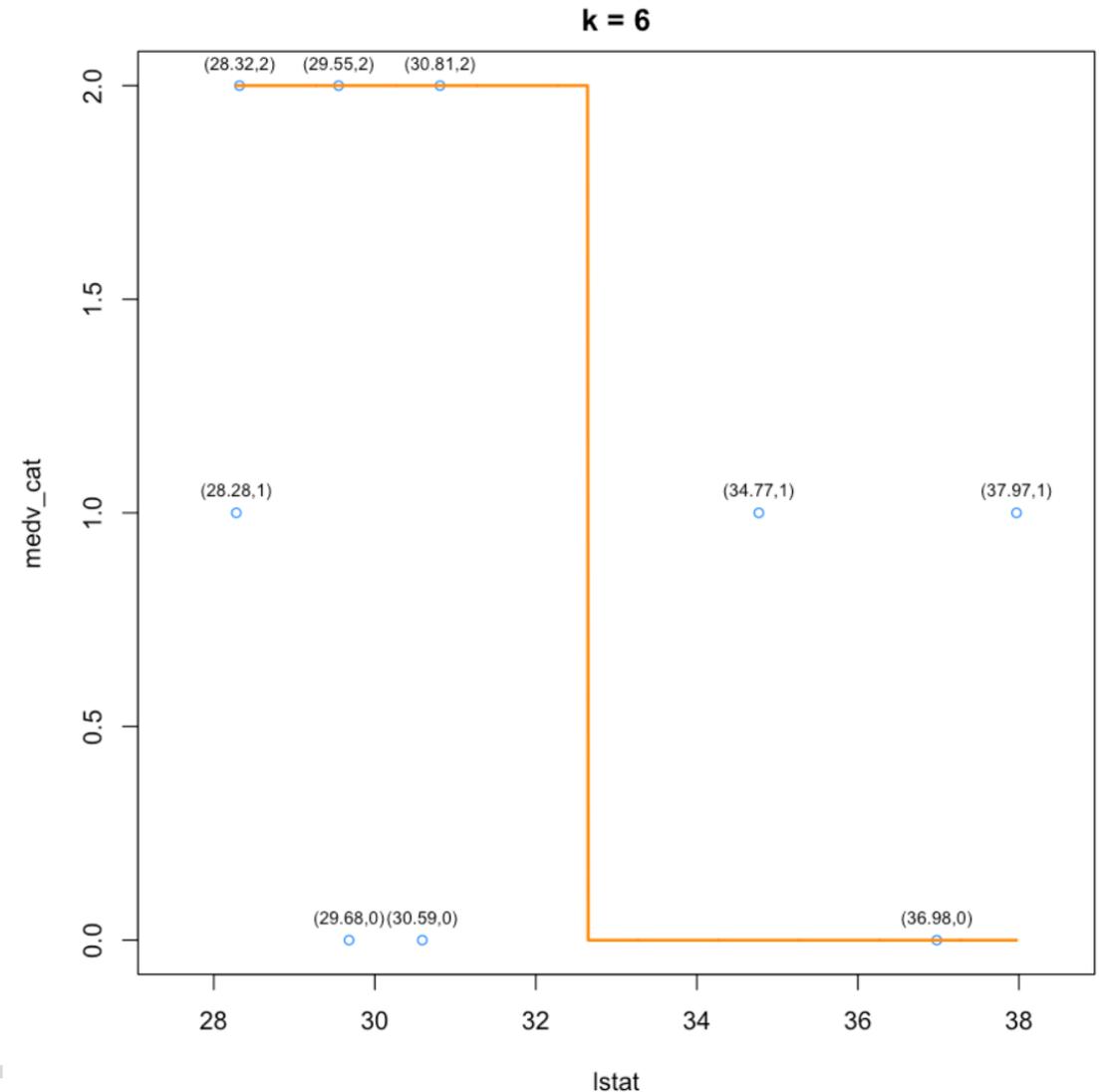
$$\hat{P}(Y = g|X = x_0) = \frac{1}{K} \sum_{x_i \in N_K(x_0)} 1(y_i = g)$$

- $1(\cdot)$: indicator function
 - $N_K(x_0)$ is the set of K training observations that are closest to x_0
- The predicted class is the class with maximum predicted probability

$$\hat{C}(x_0) = \operatorname{argmax}_g \hat{P}(Y = g|X = x)$$

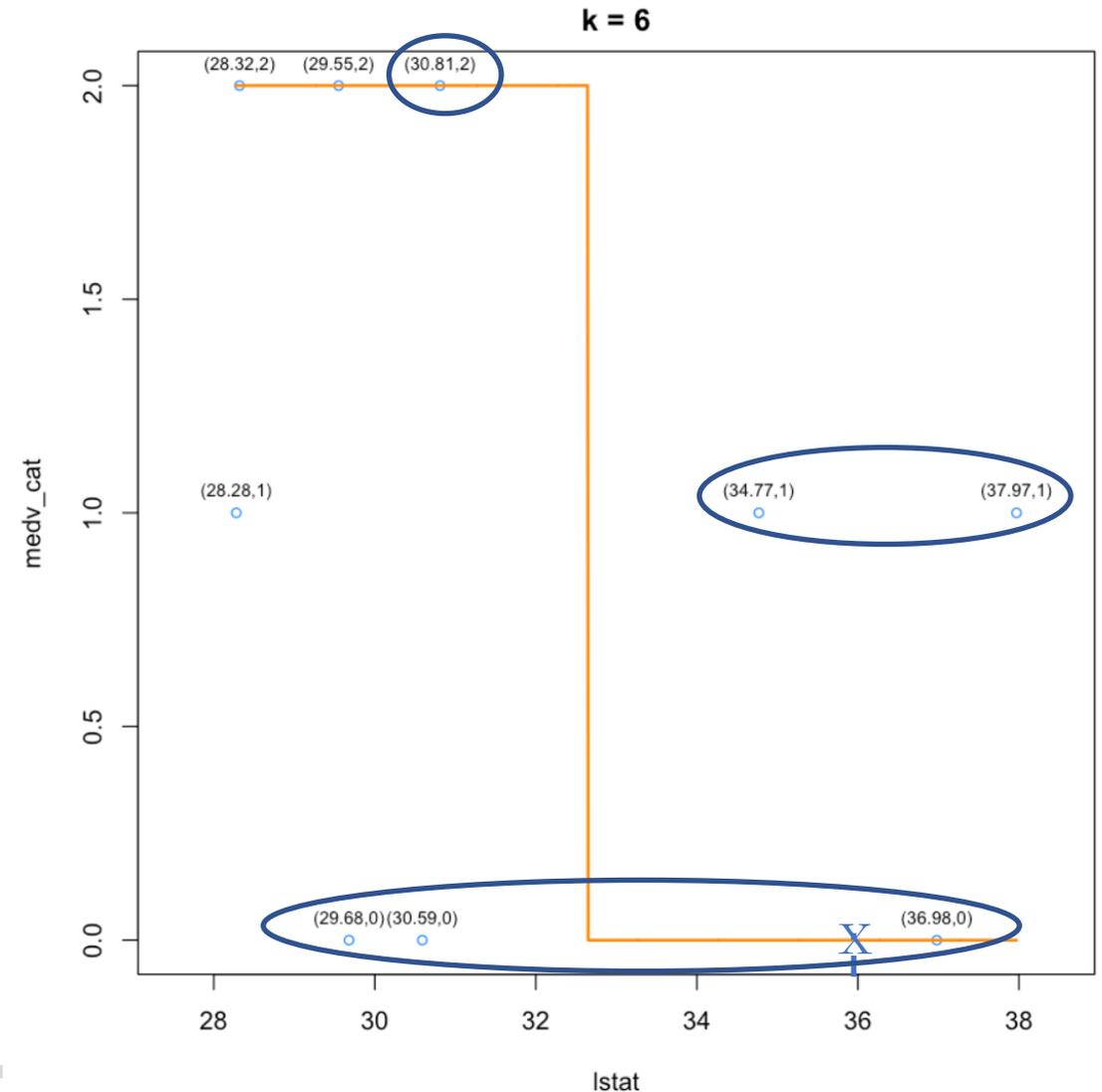
Example: 6-nearest neighbors classification

- Prediction of the category of median house value (**0**: low, **1**: medium, **2**: high) given the percentage of households with low socioeconomic status (**lstat**)
- Orange curve: $\hat{C}(x_0)$



Example: 6-nearest neighbors classification

- $x_0 = 36$
- $N_K(x_0) = \{29.68, 30.59, 30.81, 34.77, 36.98, 37.97\}$
- $\hat{P}(Y = 0|X = x_0) = \frac{1}{2}$
- $\hat{P}(Y = 1|X = x_0) = \frac{1}{3}$
- $\hat{P}(Y = 2|X = x_0) = \frac{1}{6}$
- $\hat{C}(x_0) = 0$



K-nearest neighbors classification in Python

KNN as classification problem

```
In [6]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [7]: df = load_data("Boston")

X_train, X_test, y_train, y_test = train_test_split(df[['lstat']], df[['medv']], train_size=250, random_state=42) # We

X_train = X_train.to_numpy()
y_train = y_train.to_numpy()

y_train_cat = (y_train >= 14) * 2 + ((y_train < 14) * (y_train >= 10)) * 1
y_train_cat = y_train_cat.reshape((-1,))
```

```
In [8]: df_train = pd.DataFrame({'lstat': X_train.reshape(-1,)}, 'medv': y_train_cat.reshape(-1,))

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

n = 2
knn = KNeighborsClassifier(n, weights = 'uniform')
y_pred = knn.fit(X_train, y_train_cat).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')
sns.regplot(data = df_train, x = 'lstat', y = 'medv', fit_reg = False, scatter_kws={"color": "black"}).set(title = f'k

plt.xlim([28, 39])
plt.show()
```

