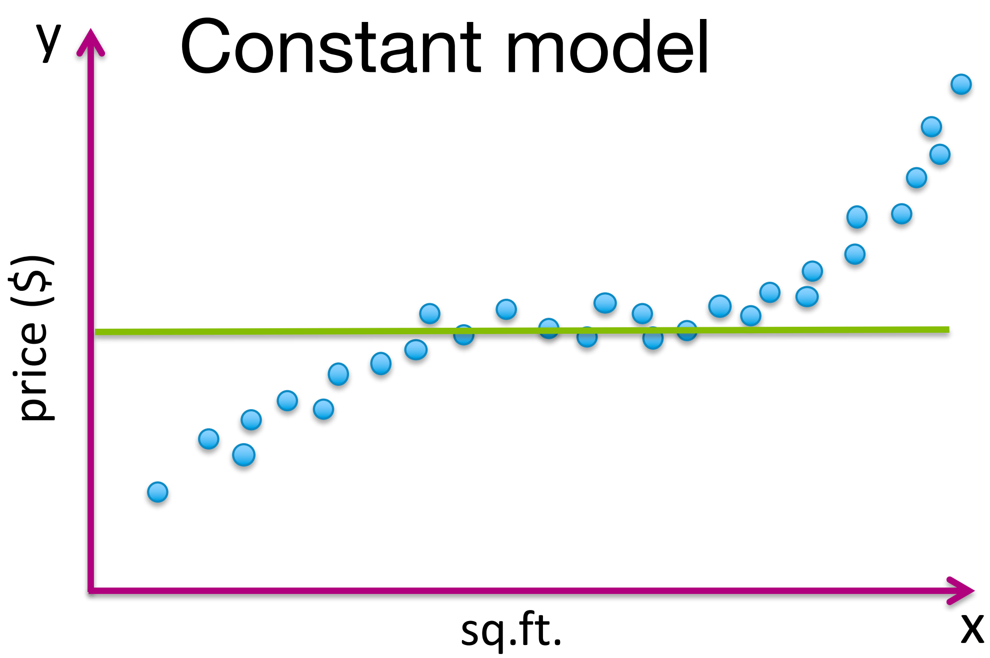
Nearest Neighbor Methods

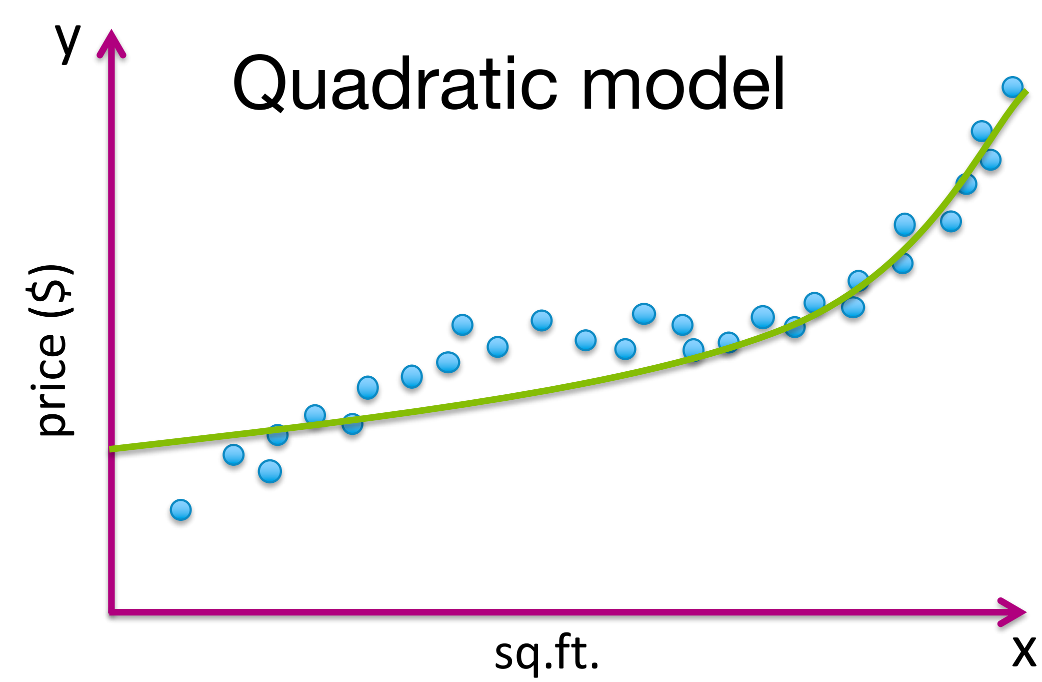
Sewoong Oh

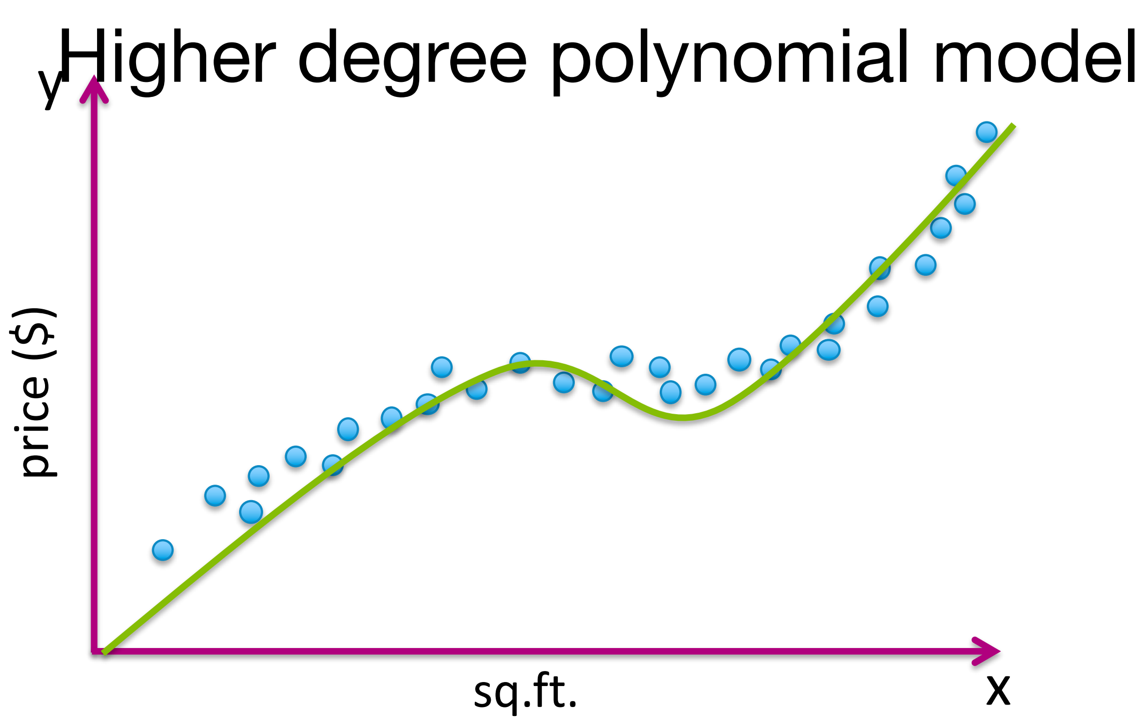
CSE/STAT 416
University of Washington

Recall Regression

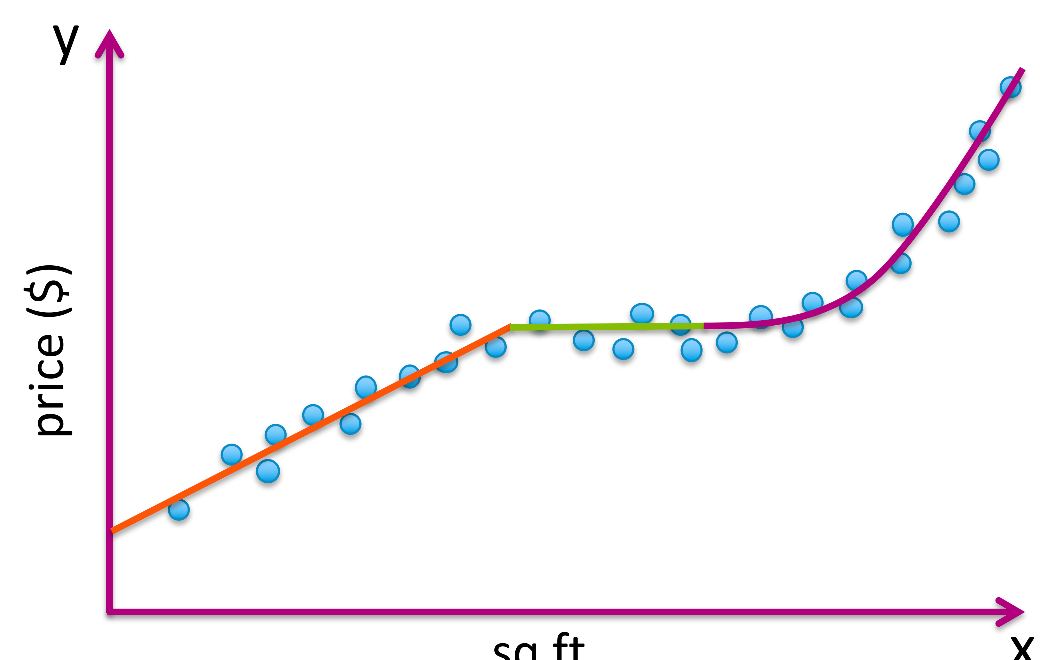
- Recall **parametric** models for regression
  - A parametric model is fitting data with a model defined by a **fixed** number of parameters, independent of data size







When real data is not a polynomial, and polynomial fit can be mis-leading



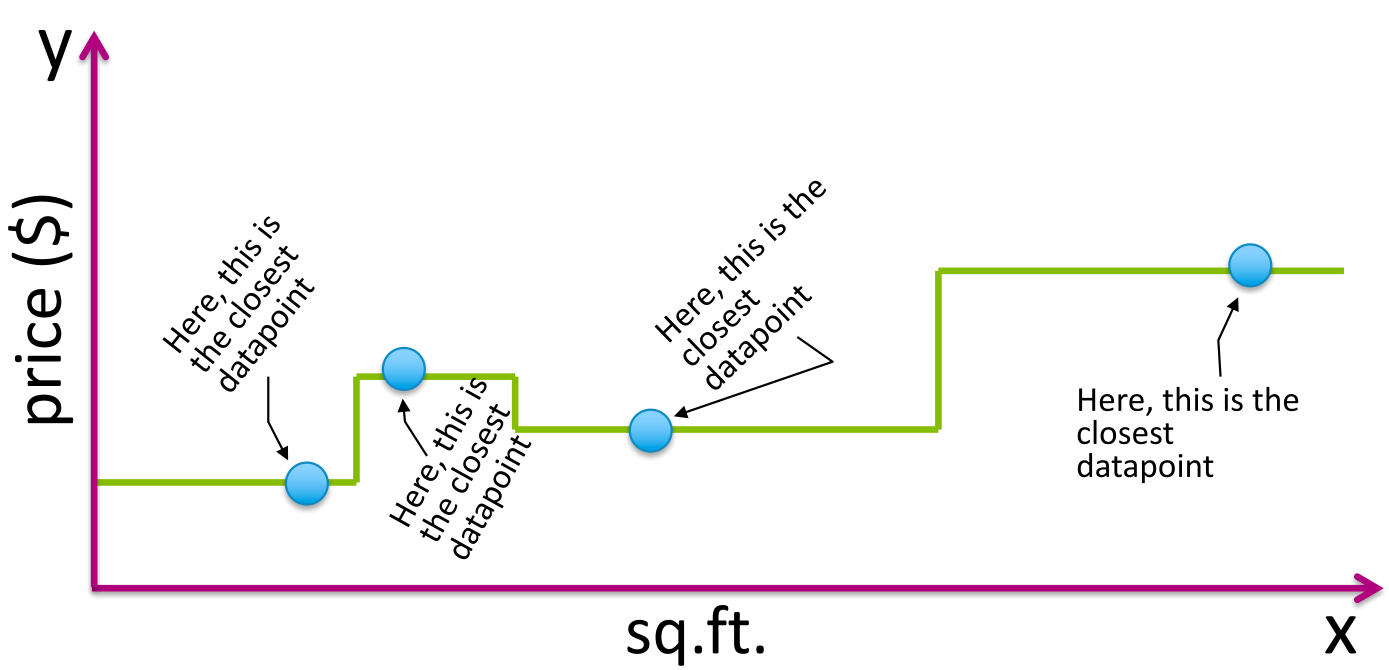
Oftentimes local structures better capture the trends

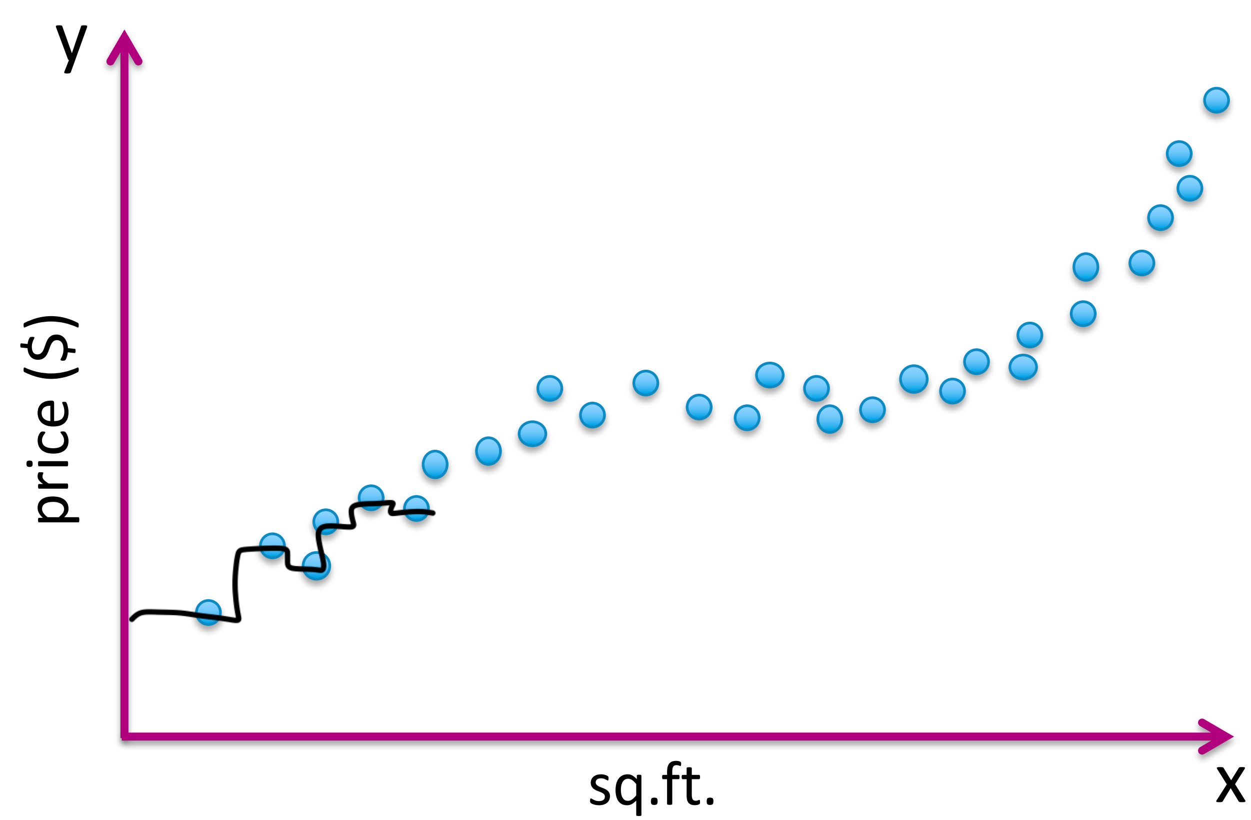
- How can we capture **local structures ?**
  (similarities and patterns among near-by data points)
- Use nearest neighbors

Nearest Neighbor methods for regression

Fit locally to training data

- 1-nearest neighbor regression
  - Predict a value ***y*** using the nearest neighbor's label

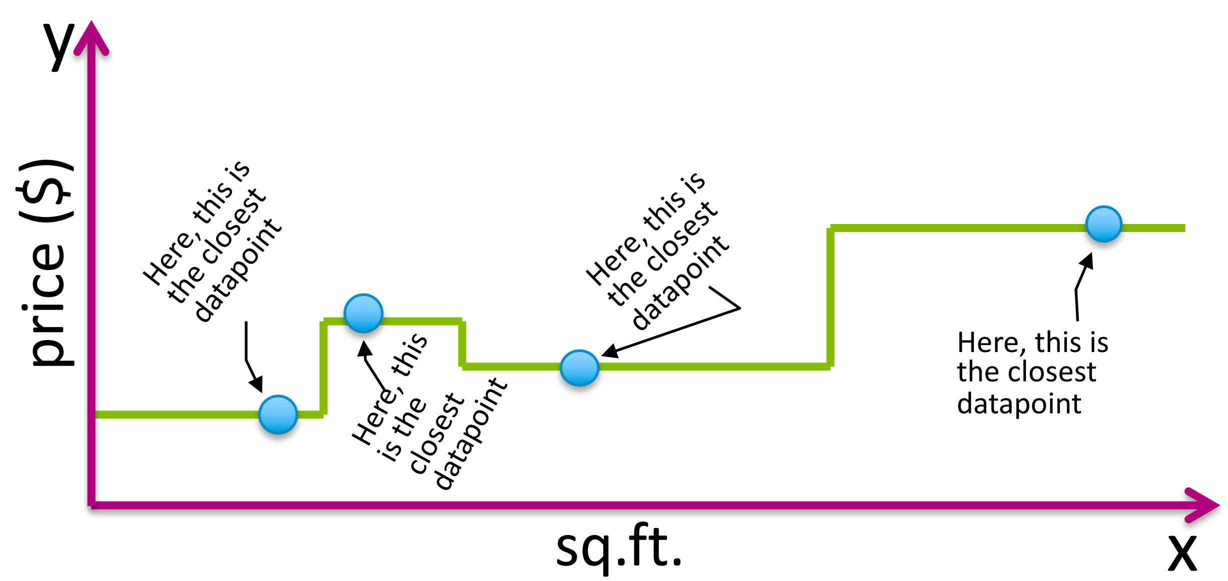




- This is what people naturally do all the time
  - Real estate agents assess value of home using recent houses sale prices on similar houses

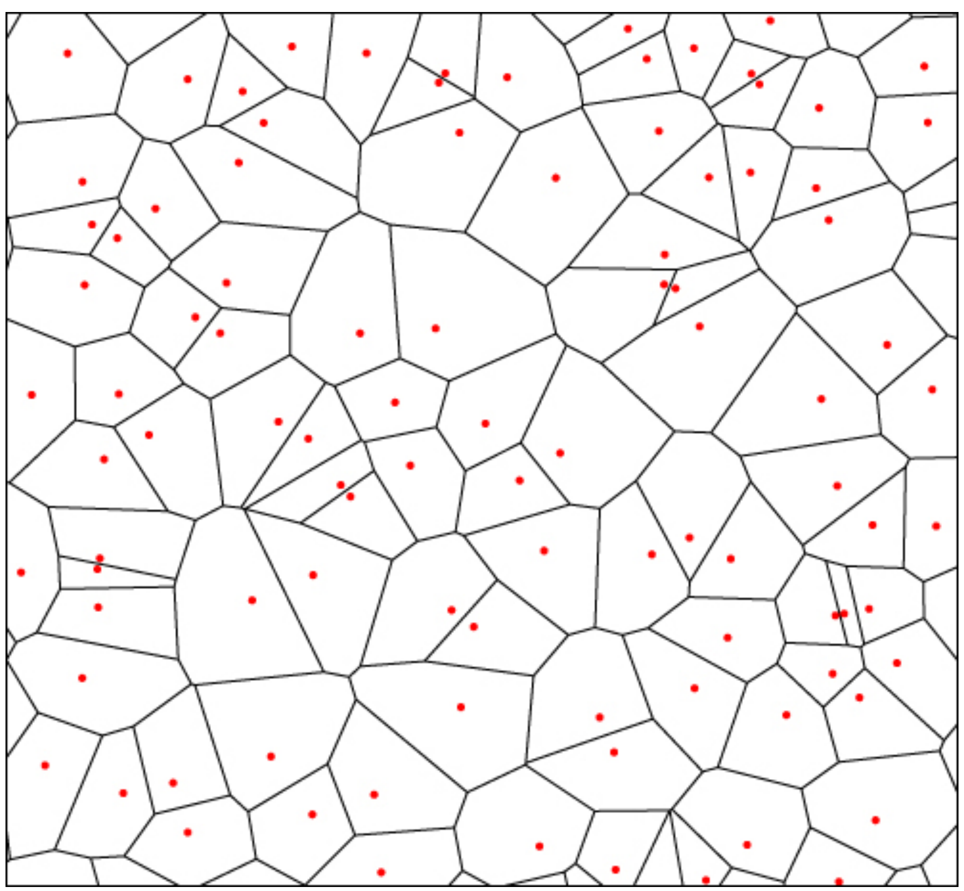
1-nearest neighbor regression

- input:
  - Training data $(x_1,y_1), \ldots , (x_N,y_N)$
  - Query point x_q
- output: prediction y_q
- 1. Find the nearest neighbor x_{nn} of x_q
- 2. Predict using y_{nn}



1-nearest neighbor regression visualized

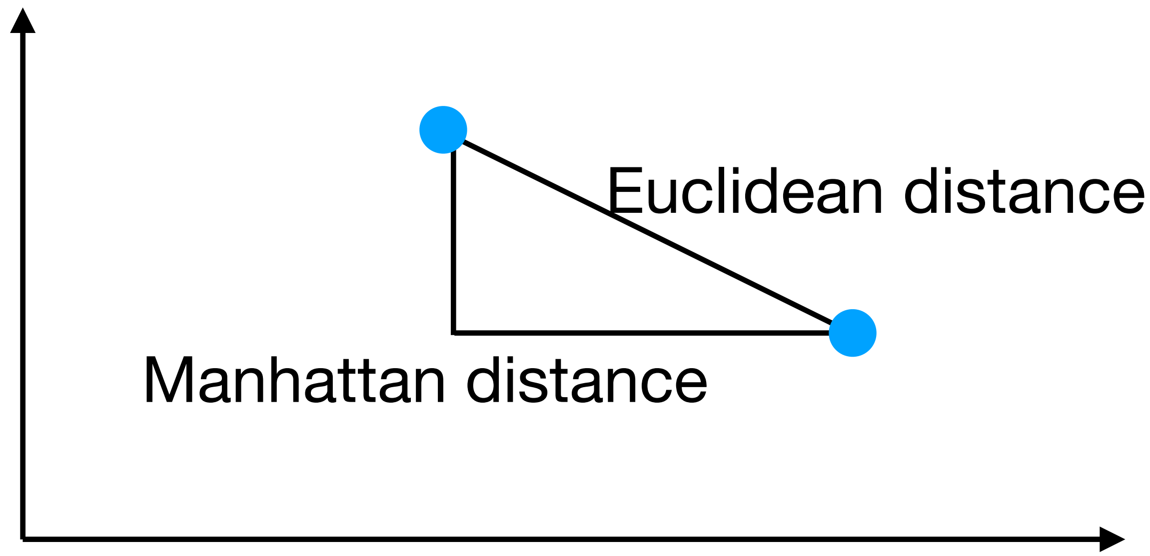
- Decision rules of 1-Nn regression can be visualized as a Voronoi tesselation
- This is never explicitly computed when using -NN regression for prediction
- But good for understanding what is going on

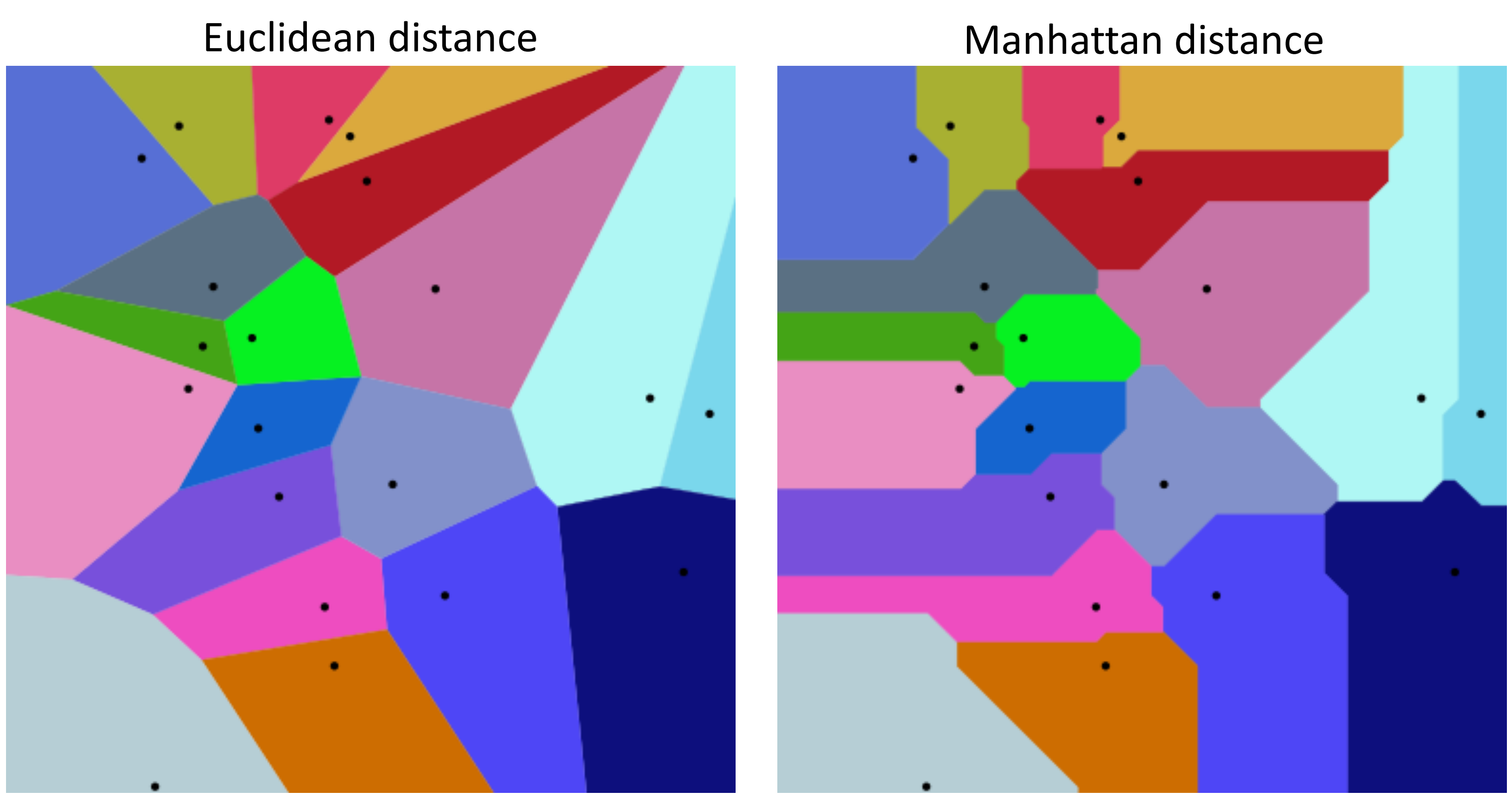


Voronoi tesselation (or diagram):

- Divide space into N regions, each containing 1 datapoint
- Defined such that any x in region is “closest” to region’s datapoint

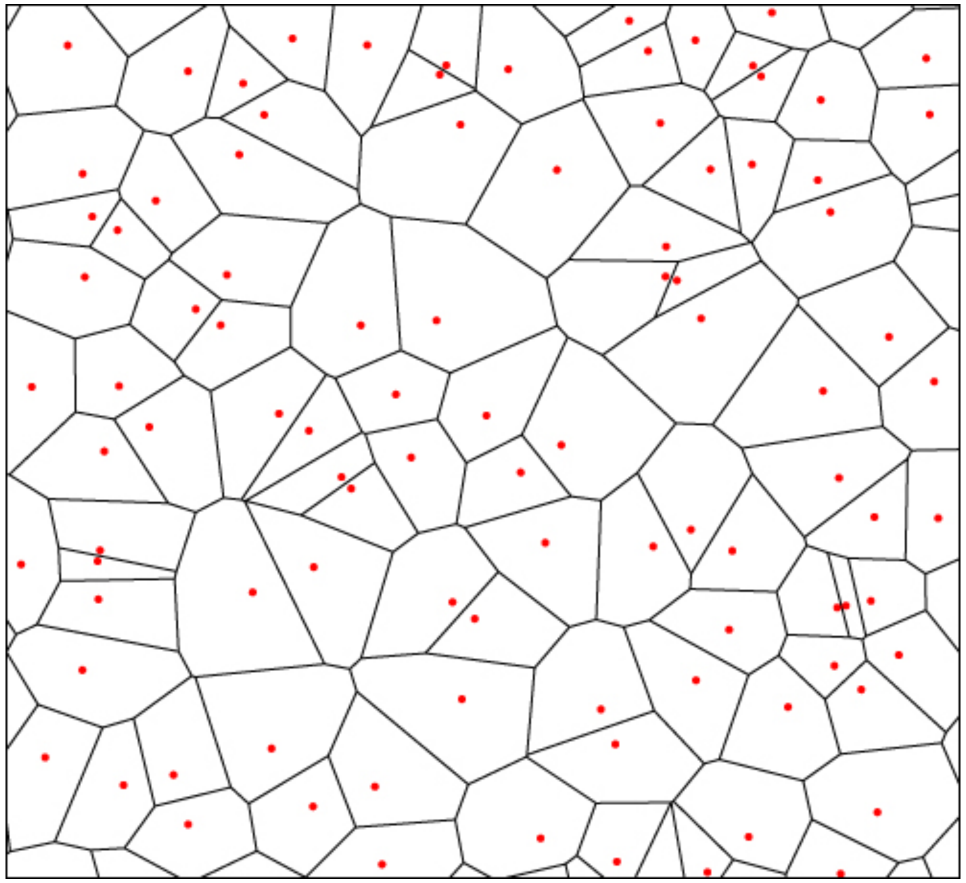
Different distance metrics lead to different prediction surfaces





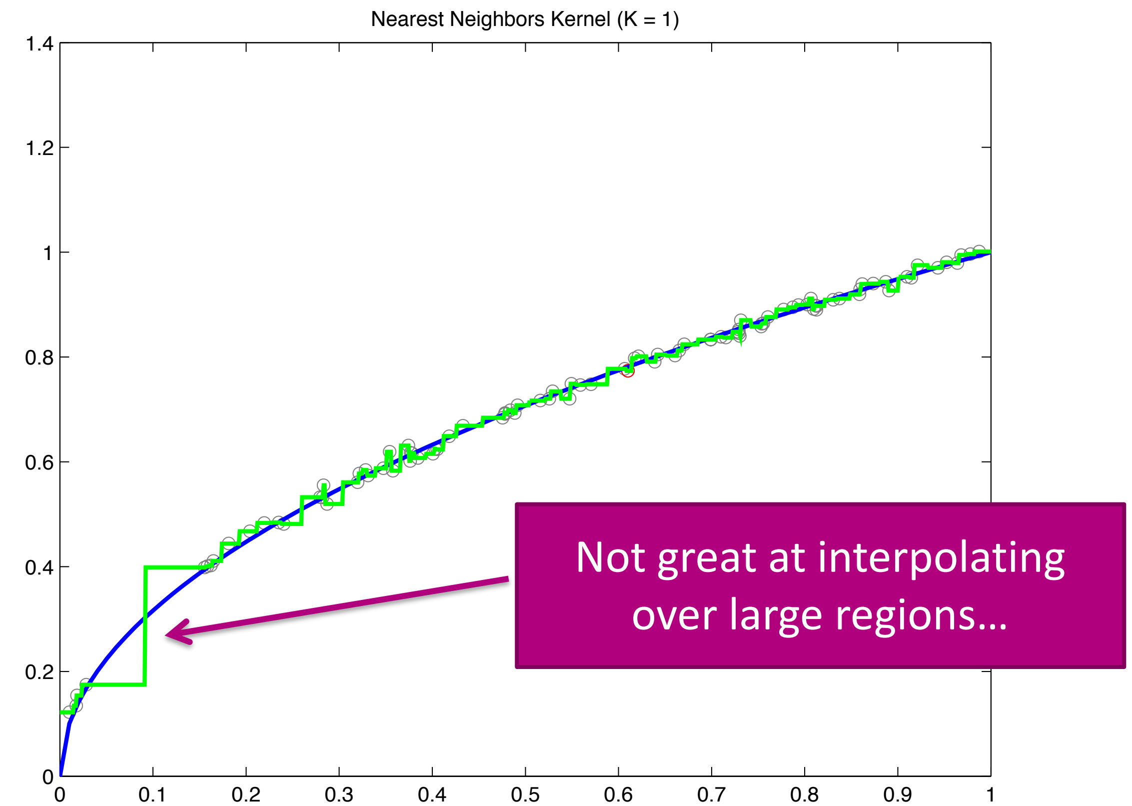
1-nearest neighbor classification

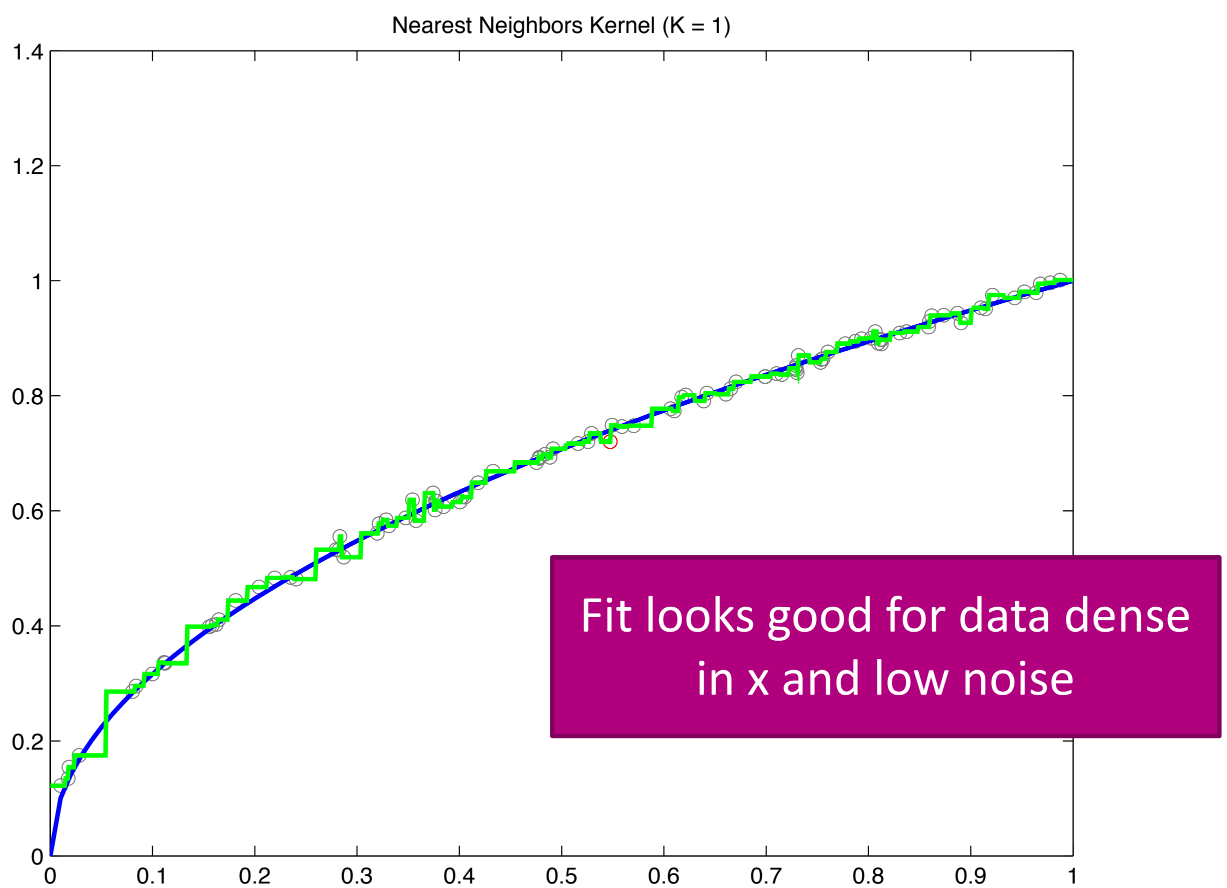
- Exactly same algorithm for 1-nearest neighbor classification

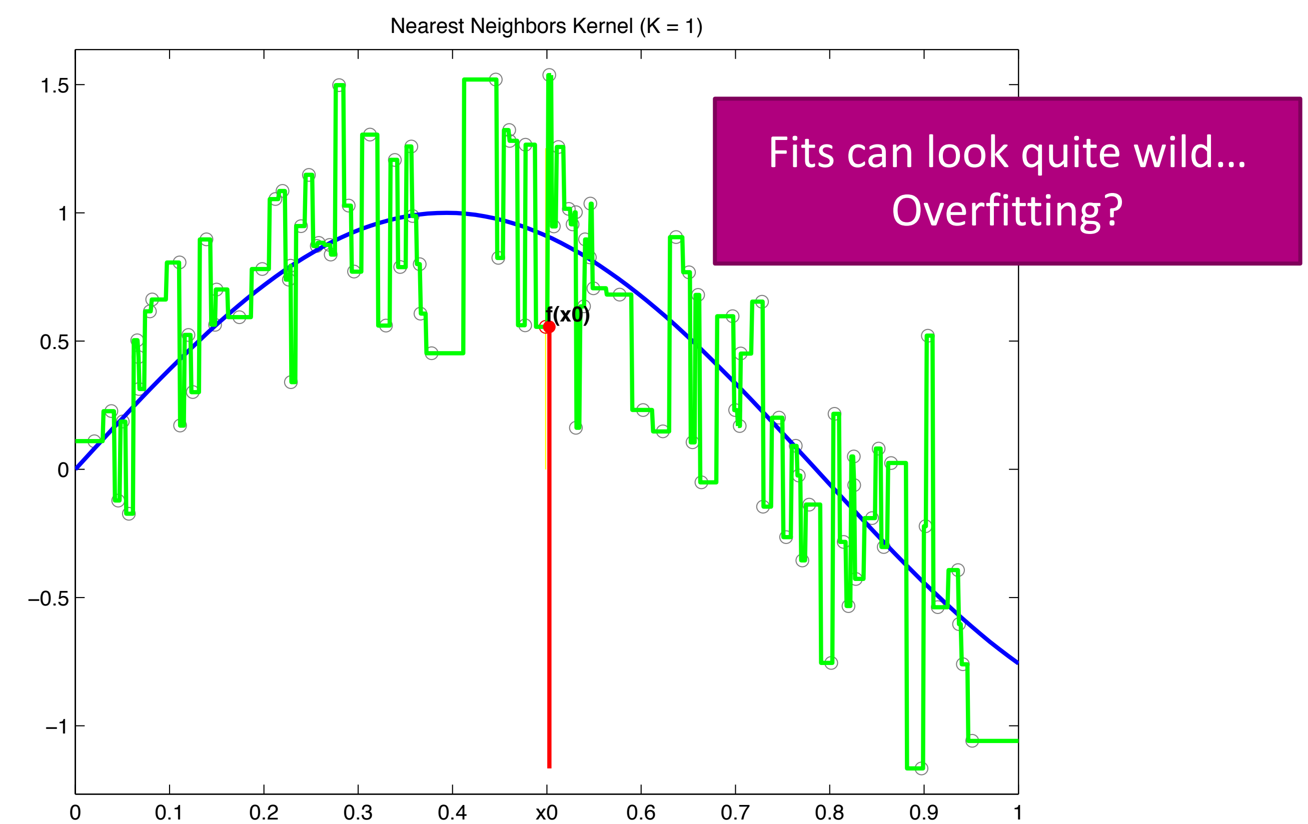


1-nearest neighbor regression

- Weaknesses
  - Inaccurate if sparse data
  - Can wildly overfit



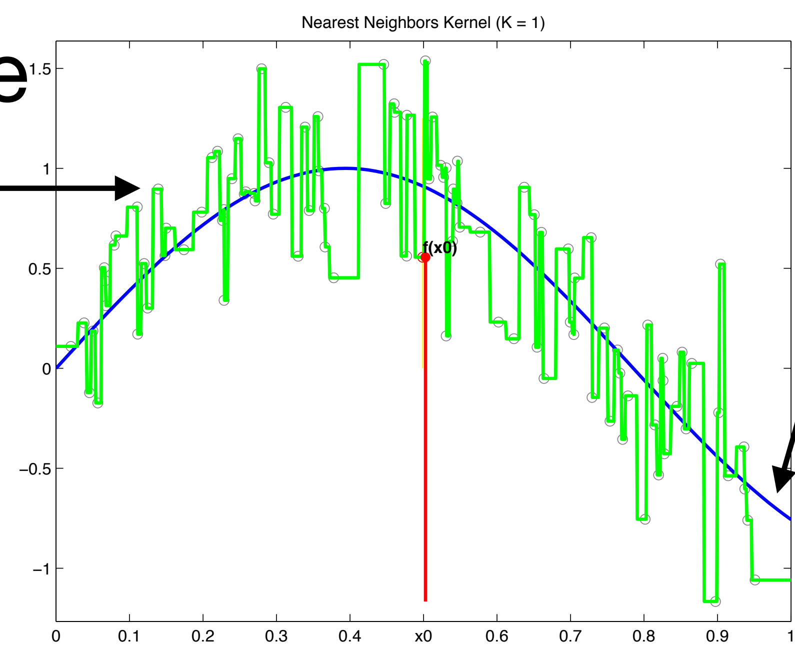




Model complexity

- A pretty good guess for complexity of a model is
  - How many real values do I need to tell you in order to explain my model?
- For example, a degree 5 polynomial requires 6 numbers (= the number of parameters, if it is a **parametric model**)

How do we regularize **non-parametric models?**



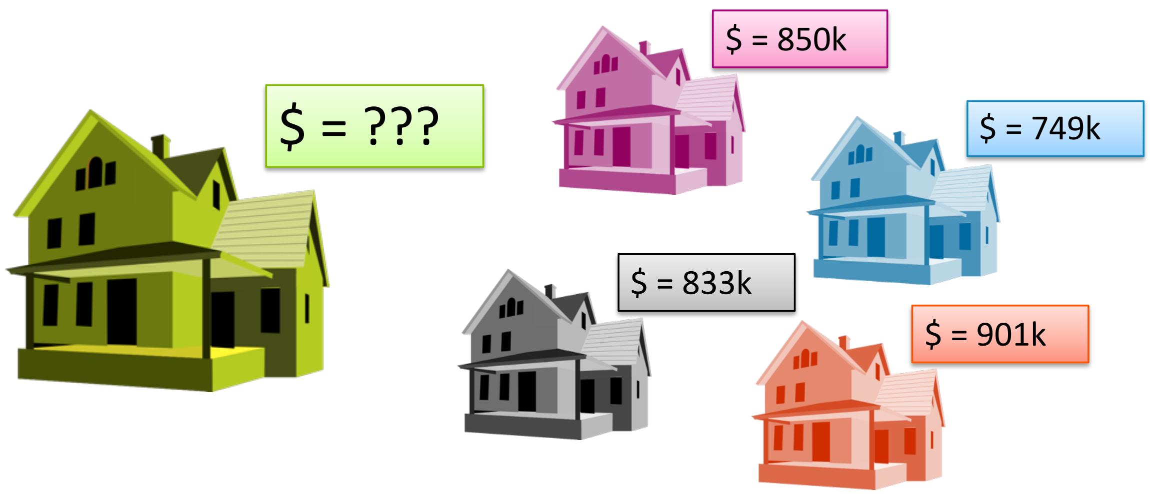
parametric models can over fit too, and we used regularization

- What is the “complexity” of a 1-nearest neighbor regression?
  - I have to give you all ***N*** data points
  - The complexity grows with ***N***
  - Such models are called **non-parametric models**

k-Nearest Neighbor methods

k-nearest neighbor methods

- Insight:
  - using more nearest neighbor should be more robust to noise



- Input:
  - Train data $(\boldsymbol{x_1},\boldsymbol{y_1}),\ldots,(\boldsymbol{x_N},\boldsymbol{y_N})$
  - Query point $\boldsymbol{x_q}$
- 1. Find $\boldsymbol{k}$ closest $\boldsymbol{x_i}$ to $\boldsymbol{x_q}$
- 2. Predict using the average of the labels of those points
-

k-nearest neighbor search

- Query house:

- Dataset:

- Specify: Distance metric
- Output: Most similar houses

k-nearest neighbor algorithm

sort first k houses by distance to query house

Initialize Dist2kNN = sort($\delta_1,\ldots,\delta_k$) ← list of sorted distances

= sort($_1$.., $_k$ ← list of sorted houses

For i=k+1,…,N

query house

Compute: δ = distance($_i$ $_q$

If δ < Dist2kNN[k]

find j such that δ > Dist2kNN[j-1] but δ < Dist2kNN[j]

remove furthest house and shift queue:

[j+1:k] = [j:k-1]

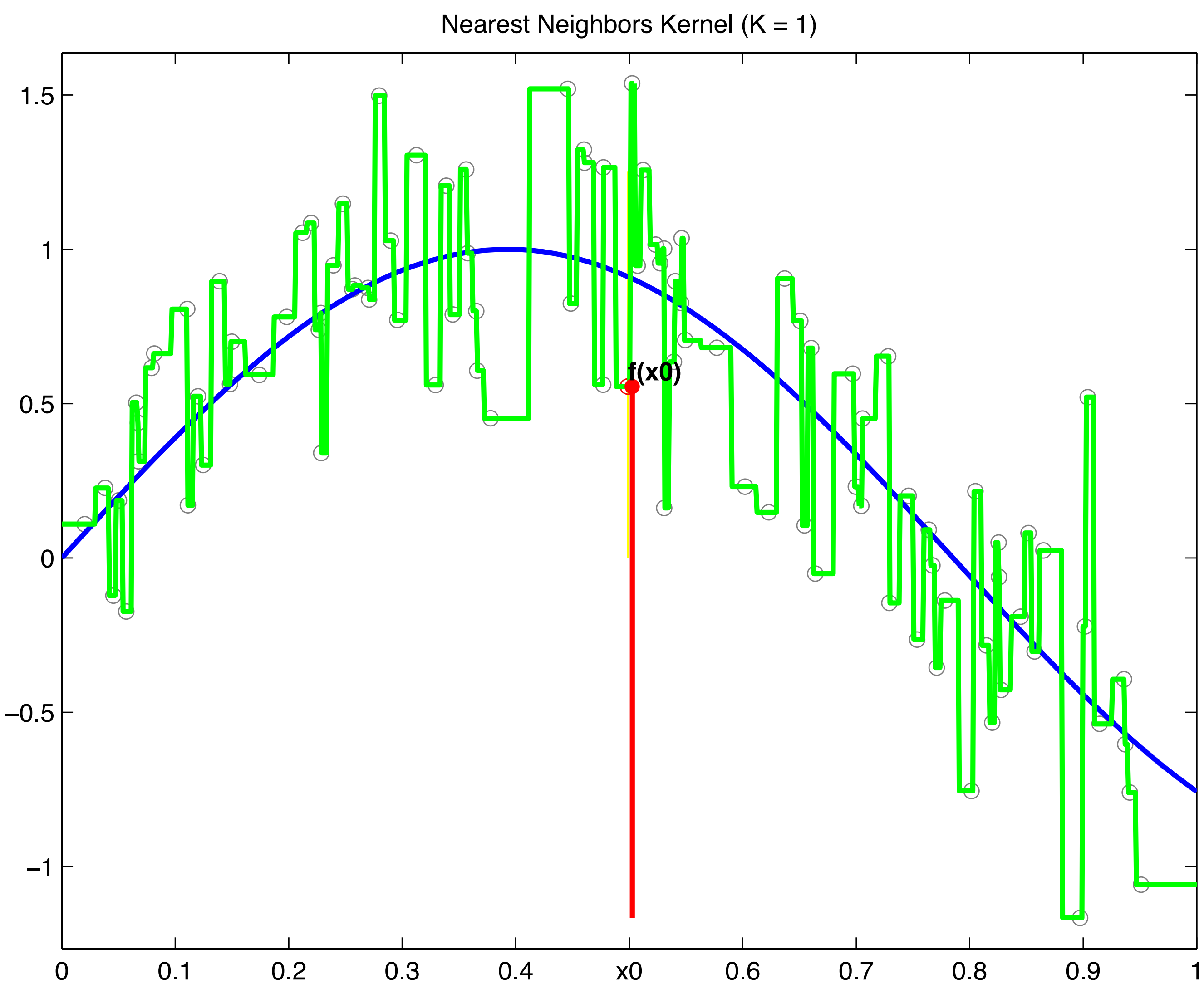
Dist2kNN[j+1:k] = Dist2kNN[j:k-1]

set Dist2kNN[j] = δ and [j] = $_i$

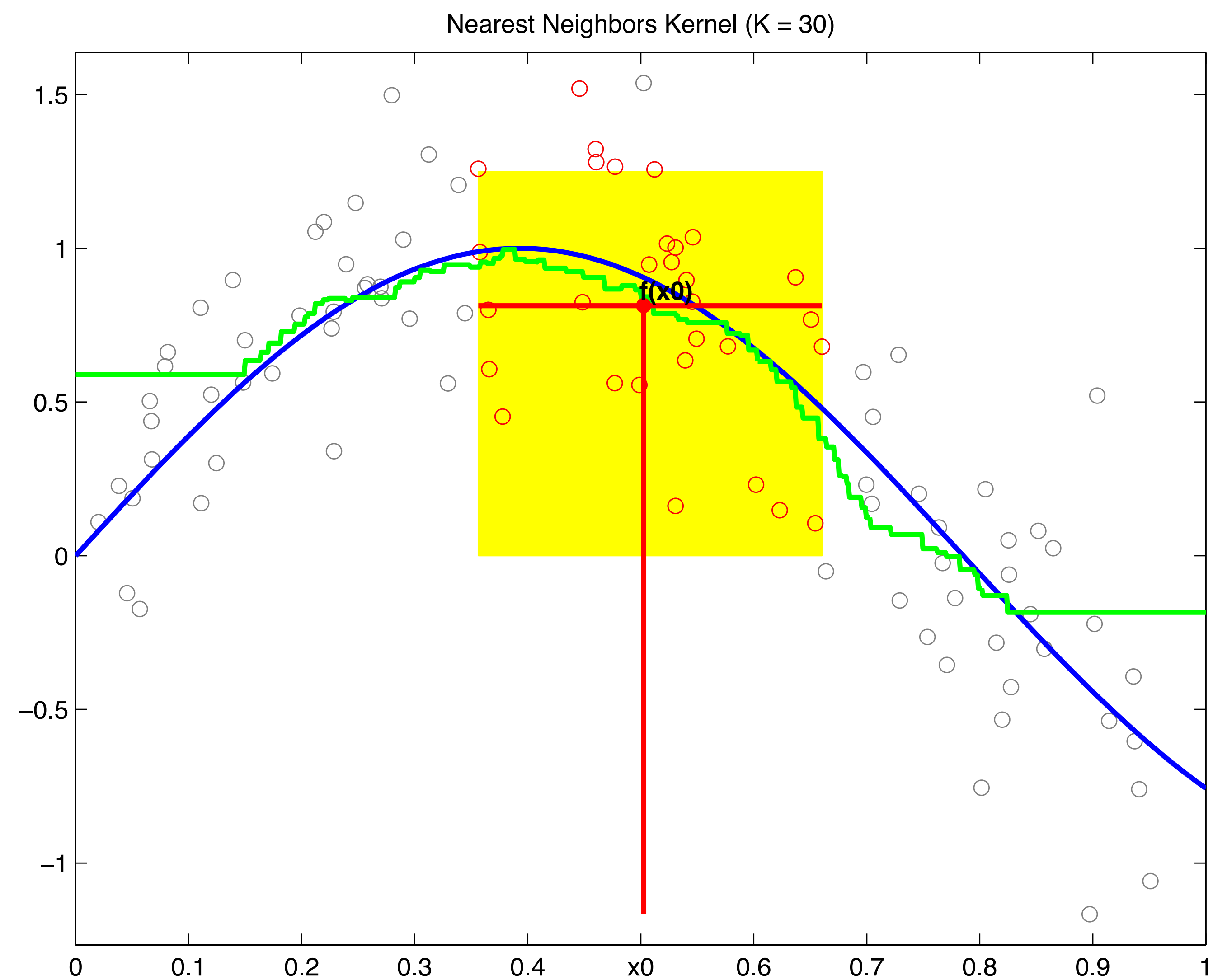
Return k most similar houses ← closest houses to query house

k-nearest neighbor in practice

- 1-nearest neighbor predictor



- 30-nearest neighbor predictor



- Averaging over larger ***k*** reduces **variance** making it robust to noise
- But increases **bias** which is particularly prominent at the boundaries and for large ***k***
- still discontinuous (as a neighbor is in or out)

Discontinuous predictions are bad…

- If you care about accuracy, it does not matter that much
- but, if you are pricing your house, then it is very sensitive at the discontinuous point, for example 2640sq.ft. vs 2641sq.ft
- This seems unrealistic or unintuitive

Solution to discontinuity

- Weighted k-nearest neighbors
- idea:
  - Weigh each neighbor according to how similar it is to the query

weights on NN

$$\hat{y}_q = \frac{c_{qNN1} y_{NN1} + c_{qNN2} y_{NN2} + c_{qNN3} y_{NN3} + \ldots + c_{qNNk} y_{NNk}}{\sum_{j=1}^{k} c_{qNNj}}$$

  - We want the weights to satisfy

Want weight c_{qNNj} to be small when distance(x_{NNj},x_q) large

and c_{qNNj} to be large when distance(x_{NNj},x_q) small

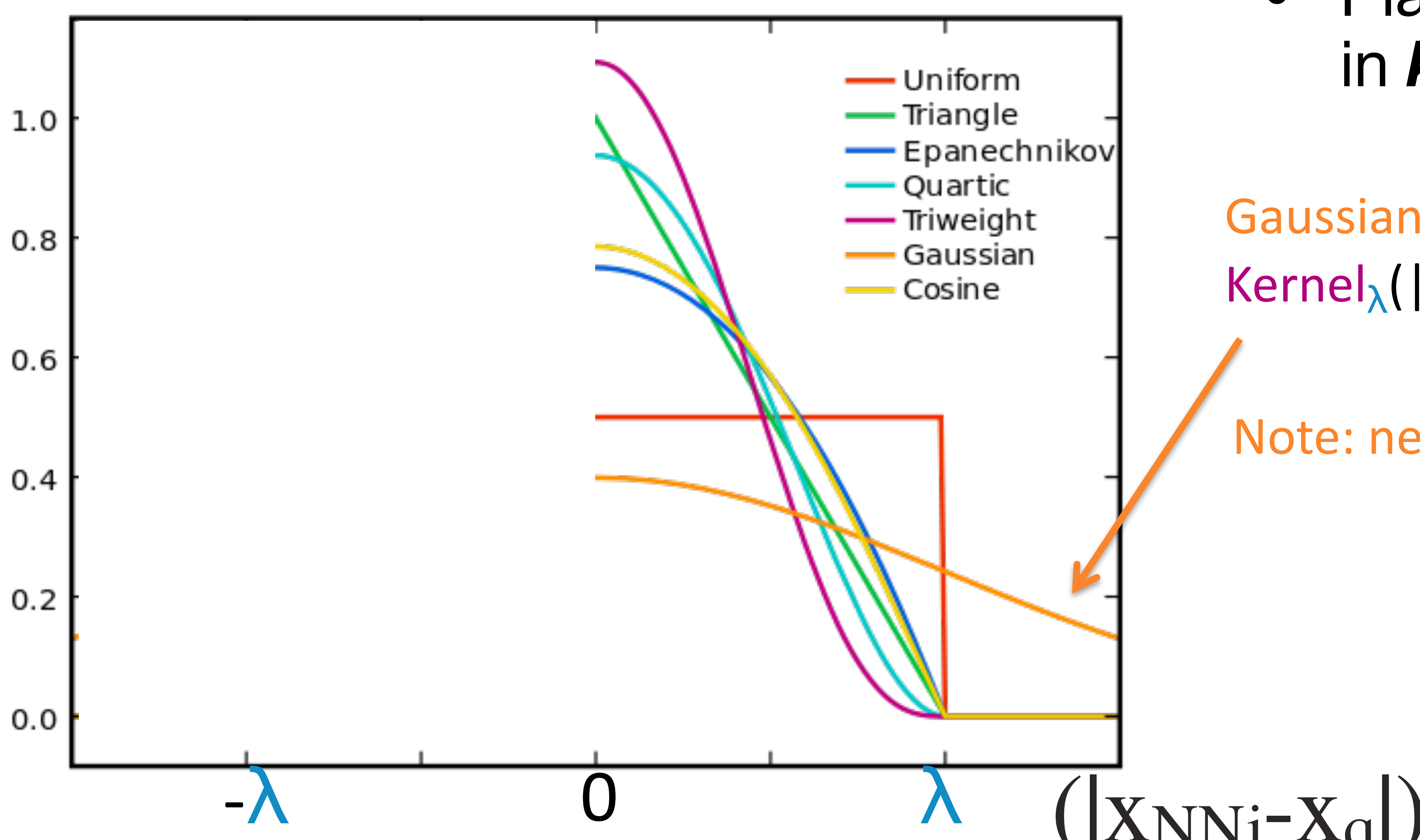
  - What would be a good choice?

Kernel methods

- Give weight according to some function fo the distance, which is inversely related with the distance
- Such functions are called **kernel functions**
- Example with 1-dimensional ***x***

Define: $c_{qNNj} = \text{Kernel}_{\lambda}(|x_{NNj}-x_q|)$

- λ is called bandwidth and is a hyper parameter controlling the width of the kernel
- Play similar role as ***k*** in ***k***-nearest neighbor



Gaussian kernel:

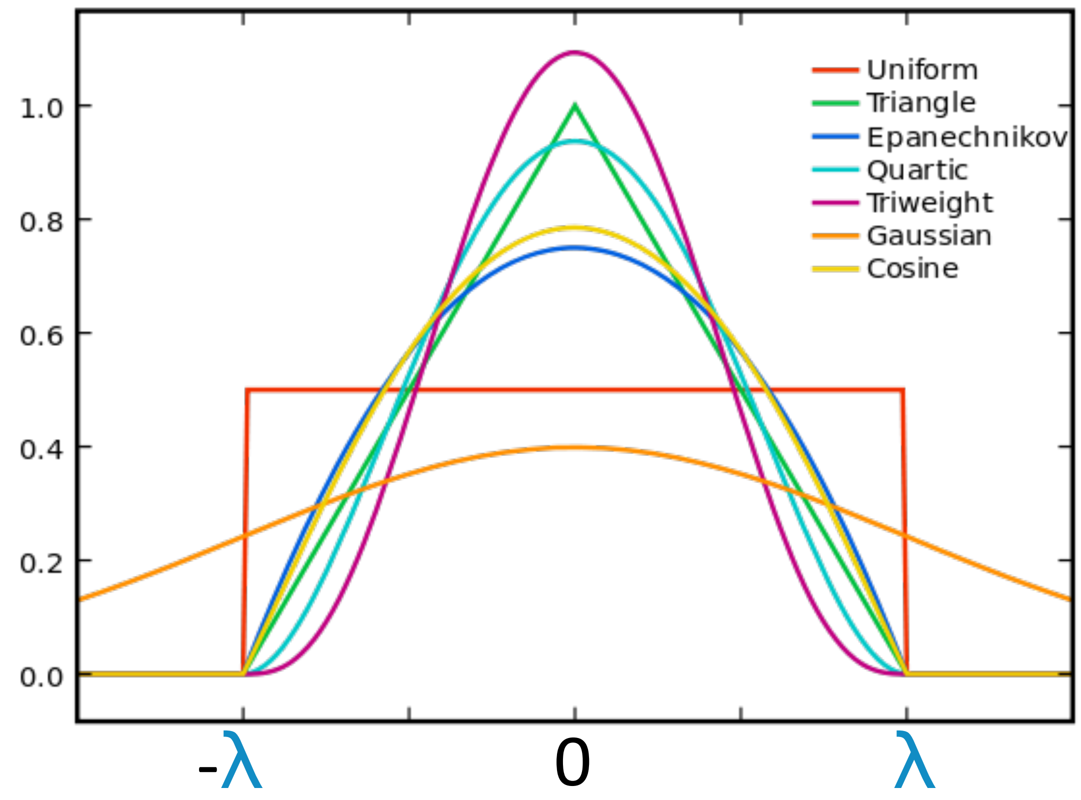
$\text{Kernel}_{\lambda}(|x_i-x_q|) = \exp(-(x_i-x_q)^2/\lambda)$

Note: never exactly 0!

Kernel with d>1

- Use a choice of distance as input to the kernel

Define: c_{qNNj} = $\text{Kernel}_{\lambda}(\text{distance}(x_{NNj}, x_q))$



Kernel regression

k-NN vs. kernel

- Weighted k-nearest neighbor
  - Take only k-nearest neighbors
  - Weigh them according to similarity

  prediction:

  weights on NN

$$\hat{y}_q = \frac{c_{qNN1}y_{NN1} + c_{qNN2}y_{NN2} + c_{qNN3}y_{NN3} + \ldots + c_{qNNk}y_{NNk}}{\sum_{j=1}^{k} c_{qNNj}}$$

- Kernel regression
  - Take all points
  - Weigh them with kernel

  prediction:

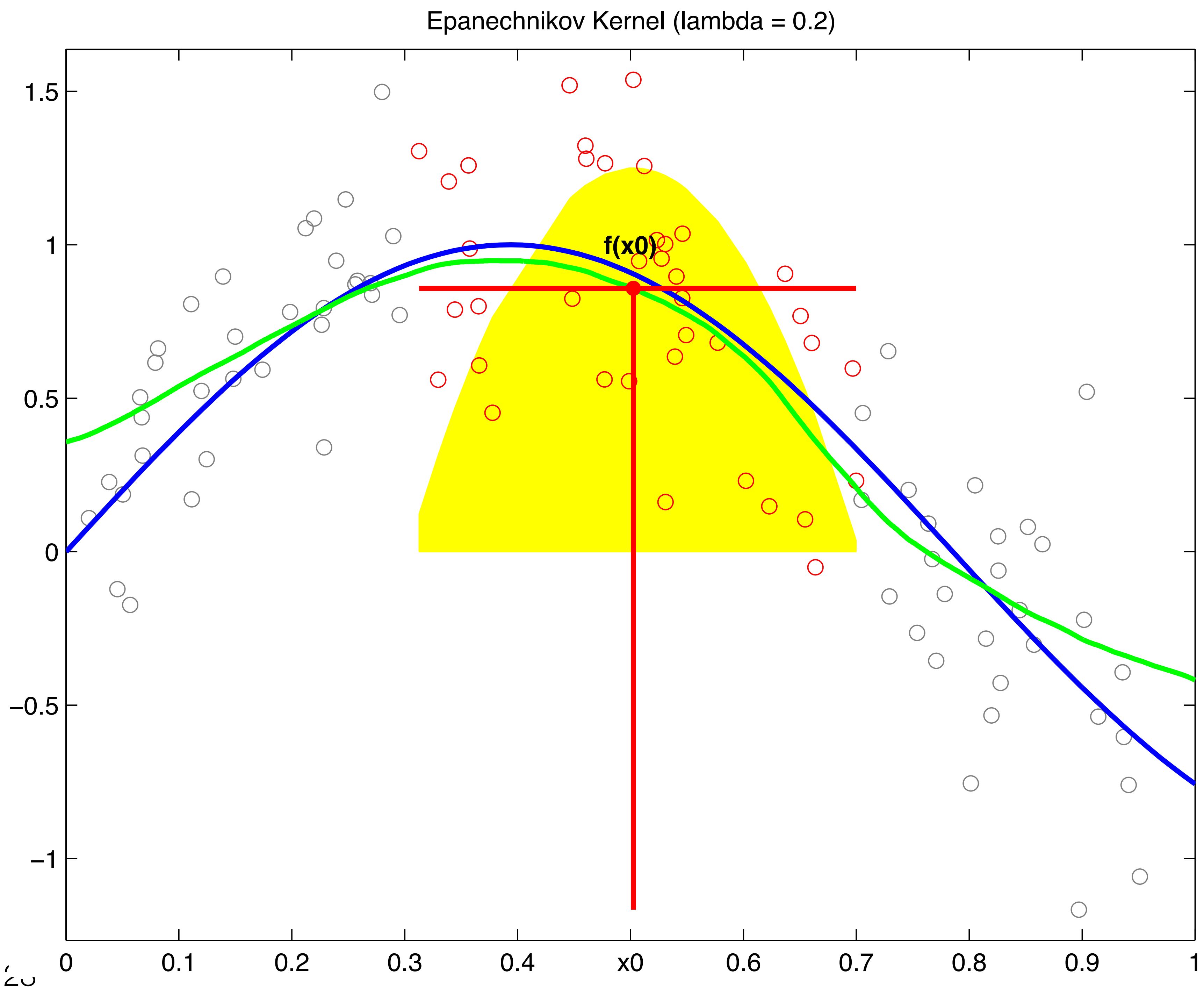
  weight on each datapoint

$$\hat{y}_q = \frac{\sum_{i=1}^{N} c_{qi}y_i}{\sum_{i=1}^{N} c_{qi}} = \frac{\sum_{i=1}^{N} \mathrm{Kernel}_{\lambda}(\mathrm{distance}(x_i,x_q)) * y_i}{\sum_{i=1}^{N} \mathrm{Kernel}_{\lambda}(\mathrm{distance}(x_i,x_q))}$$

Nadaraya-Watson kernel weighted average

Kernel regression in practice

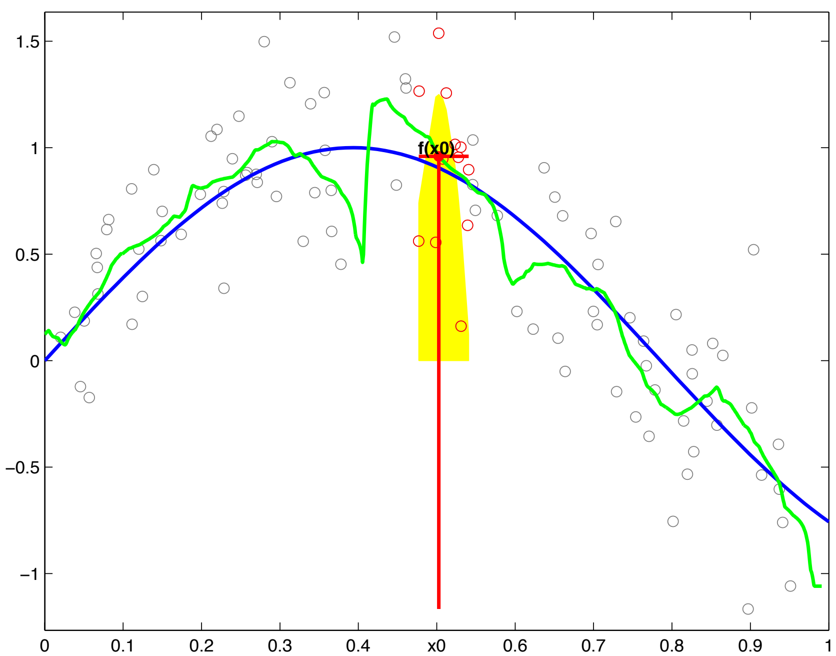
- Bandwidth lambda is 0.2
- The kernel has bounded support



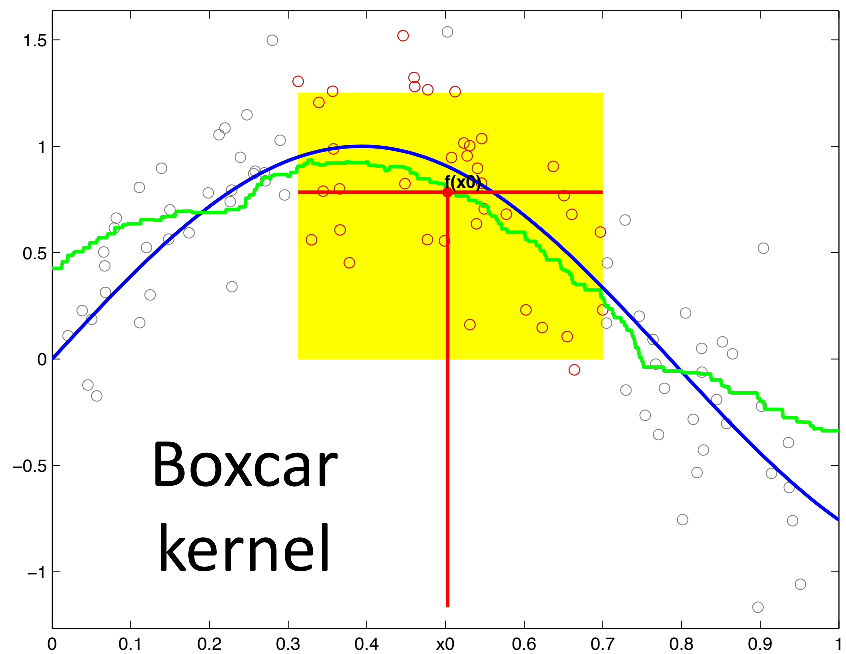
How to choose bandwidth lambda

- Often, choice of kernel matters much less than choice of lambda

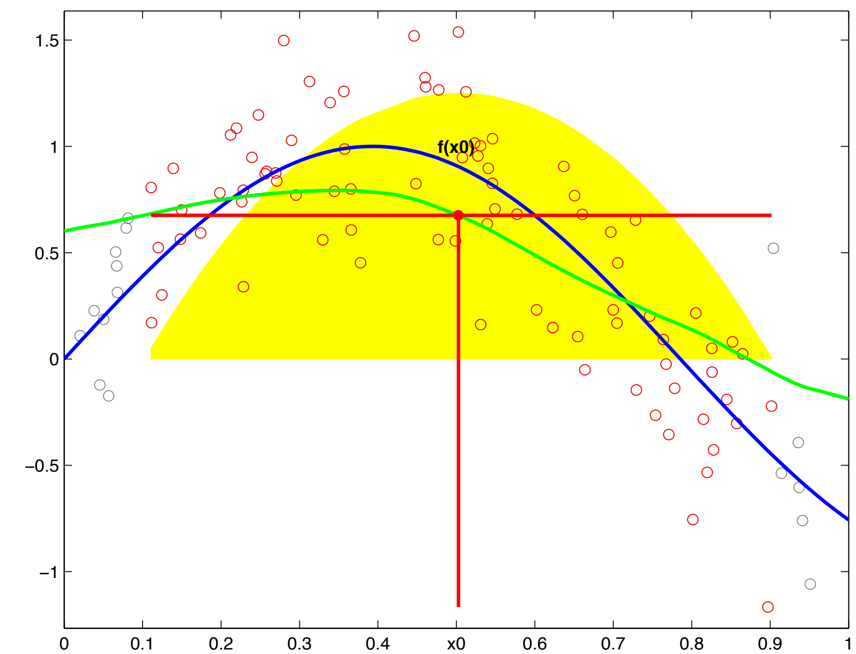
$\lambda = 0.04$



$\lambda = 0.2$



$\lambda = 0.4$



- Small bandwidth results in fluctuations and sensitivity to noise

- Large bandwidth results in oversmoothing and large bias

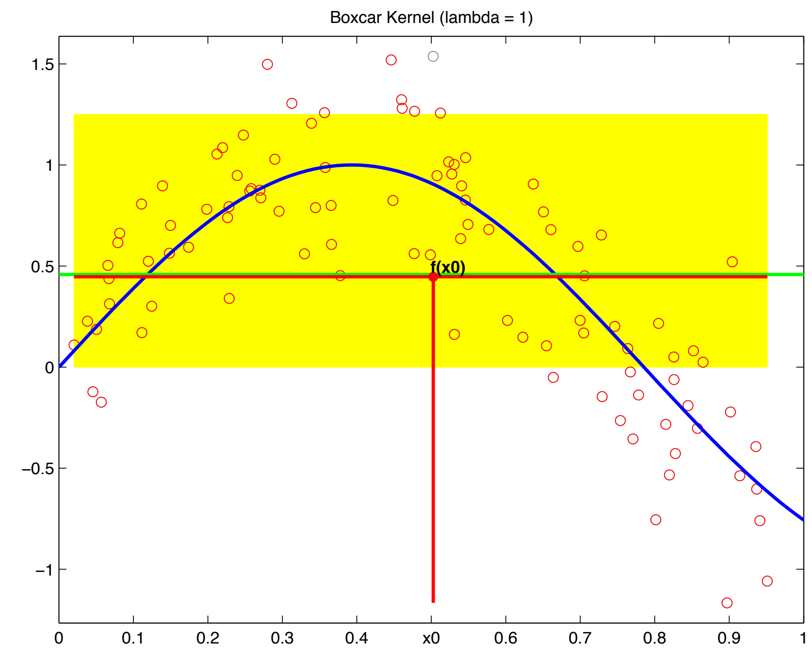
- Use cross validation to choose bandwidth lambda and/or ***k*** in ***k***-nearest neighbor

Local fit

- Both k-NN and kernel regression are embodying a idea of **local fit**
- For example, a **global constant fit** will be

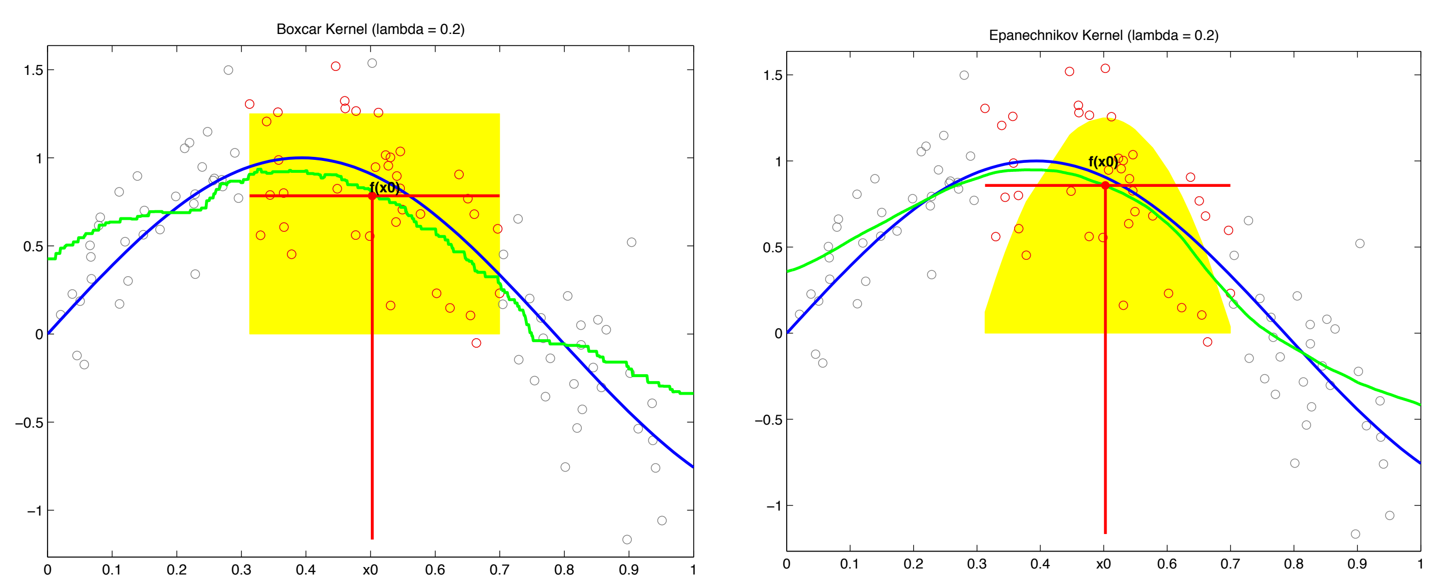
$$\hat{y}_q = \frac{1}{N}\sum_{i=1}^{N} y_i = \frac{\sum_{i=1}^{N} c\, y_i}{\sum_{i=1}^{N} c}$$

equal weight on each datapoint



- We can use **kernel** to do a **local constant fit,** for example (and make it smooth by using smooth kernels)

$$\hat{y}_q = \frac{\sum_{i=1}^{N} \text{Kernel}_{\lambda}(\text{distance}(x_i,x_q)) * y_i}{\sum_{i=1}^{N} \text{Kernel}_{\lambda}(\text{distance}(x_i,x_q))}$$



You can take this idea of local fit further

- And combine local methods (k-NN or kernel regression) and global methods () we learned so far

- So far, we fit constant function locally at each point
  -> **locally weighted average**

- We can instead fir a polynomial locally at each point
  -> **locally weighted linear regression (with polynomial features)**

-**Local linear fit** reduces bias at boundaries with minimum increase in variance

-**Local quadratic fit** doesn’t help at boundaries and increases variance, but does help capture curvature in the interior

Recommended default choice:
local linear regression

Non-parametric regression

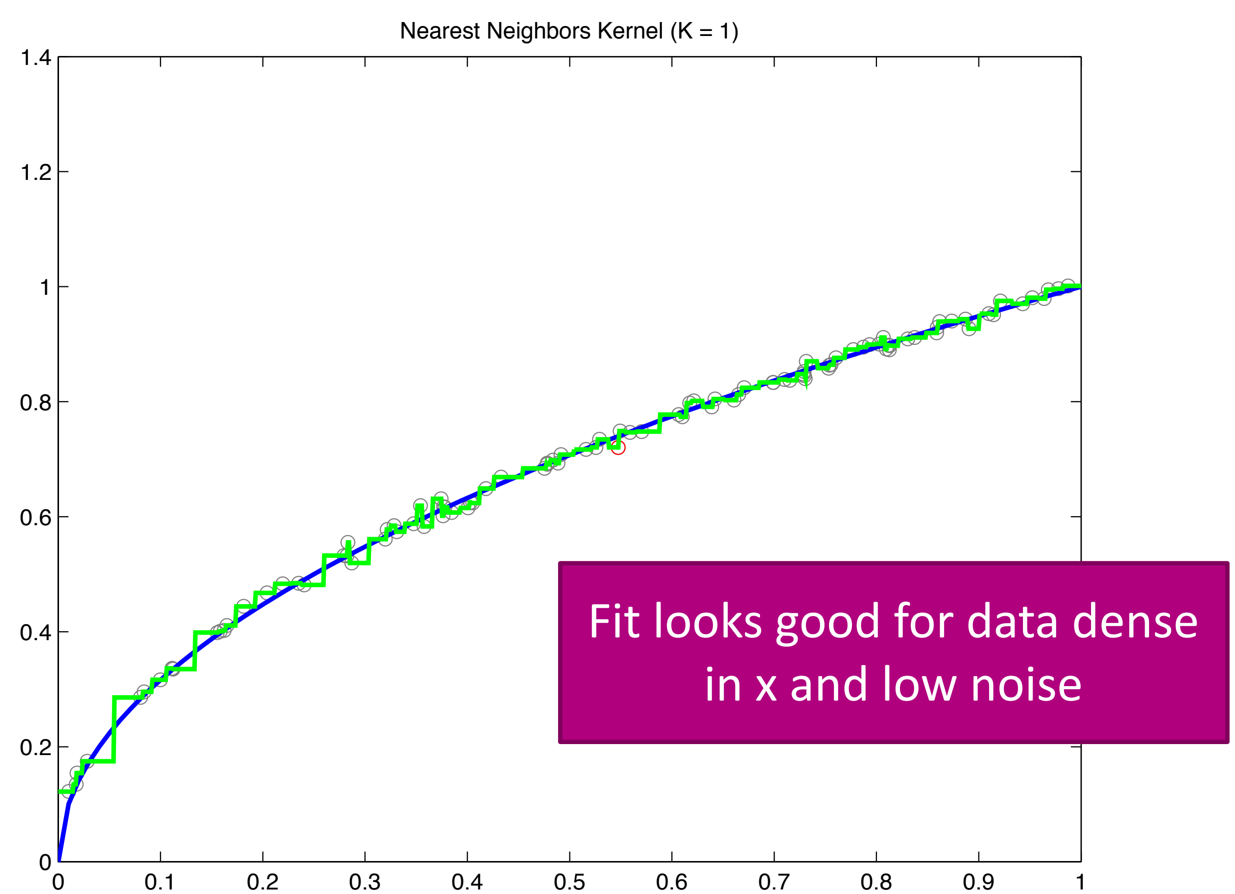
Non-parametric approaches

- K-nearest neighbor method and kernel regression requires one to store all training data points to store the predictor
- This requires storage space scaling proportional to ***N***, the number of samples in training data
- Such models are called **non-parametric**
- They are
  - Flexible
  - Make few assumptions about the true f(x)
  - Complexity of storing the predictor and making prediction grows with ***N***

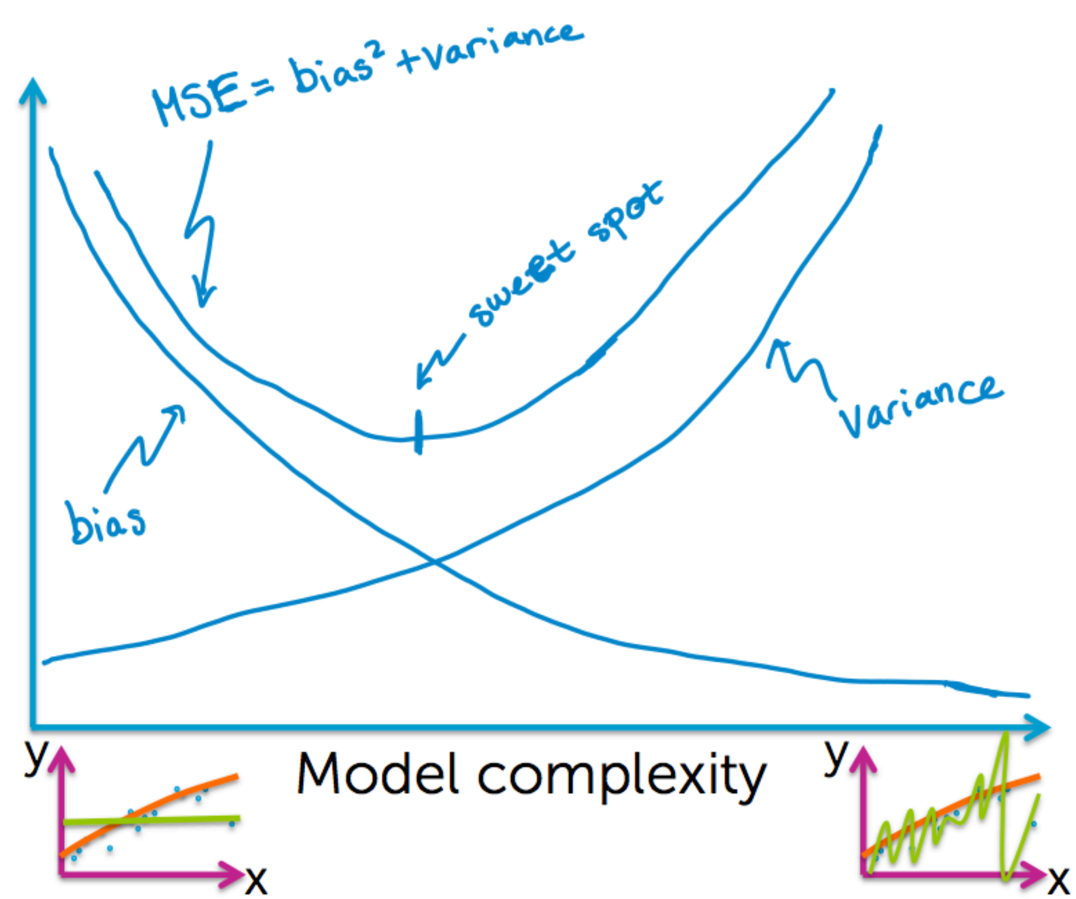
- There are many other examples:
  - splines, locally weighted structures, etc

How does nearest neighbor method behave?

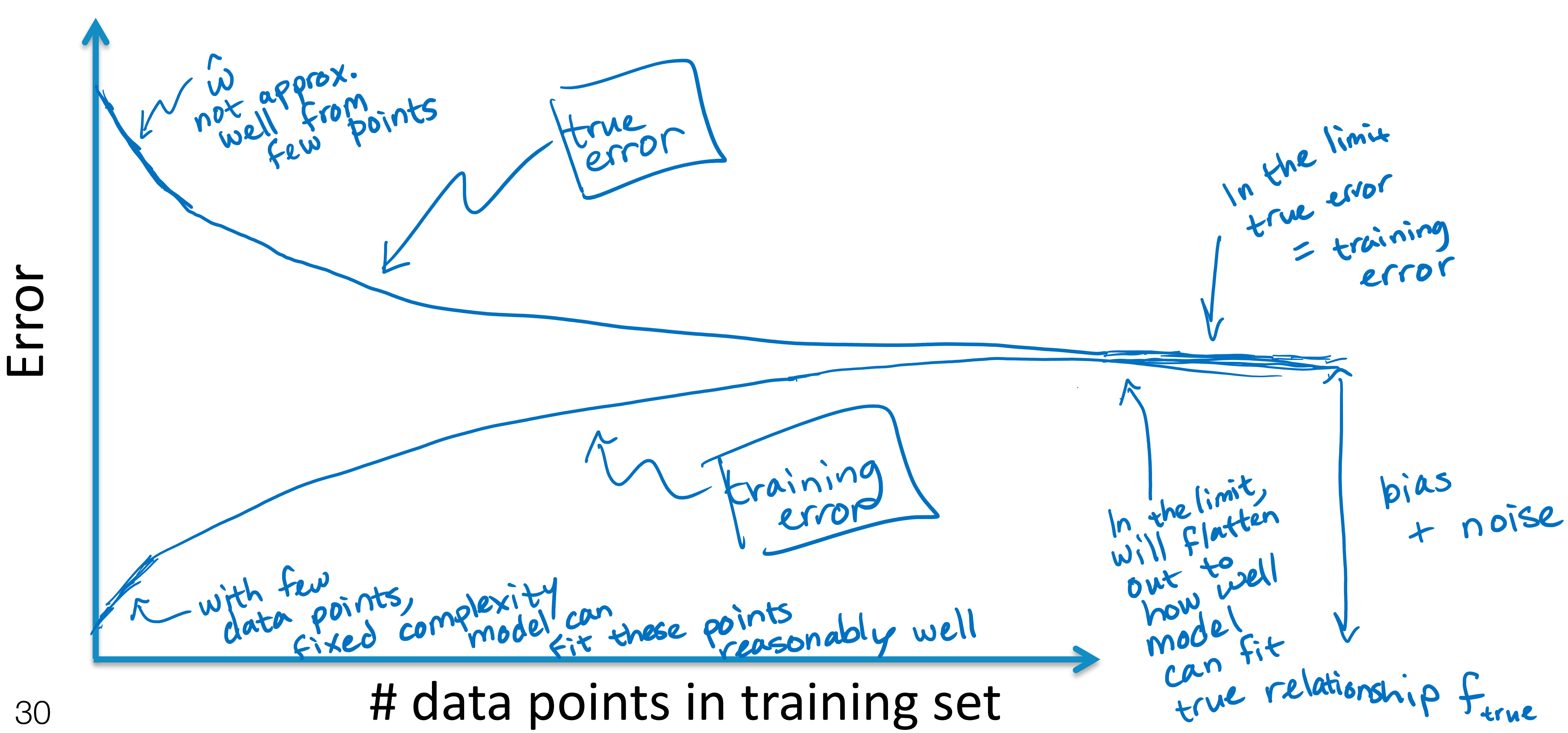
- To answer this question, people looked at the case where the number of training examples N grows to infinity
- Such process of analyzing in the limit is called **asymptotic analysis**
- For example, even with **k=1**, as N goes to infinity, and let's say there is no noise in the training data, i.e. y=f(x) for some nice function f(x)
  - Then the MSE (Mean Squared Error) goes to zero as N grows



- This is not true for parametric models
- Parametric models have non-zero test error even when there is no noise in training data and ***N*** goes to infinity

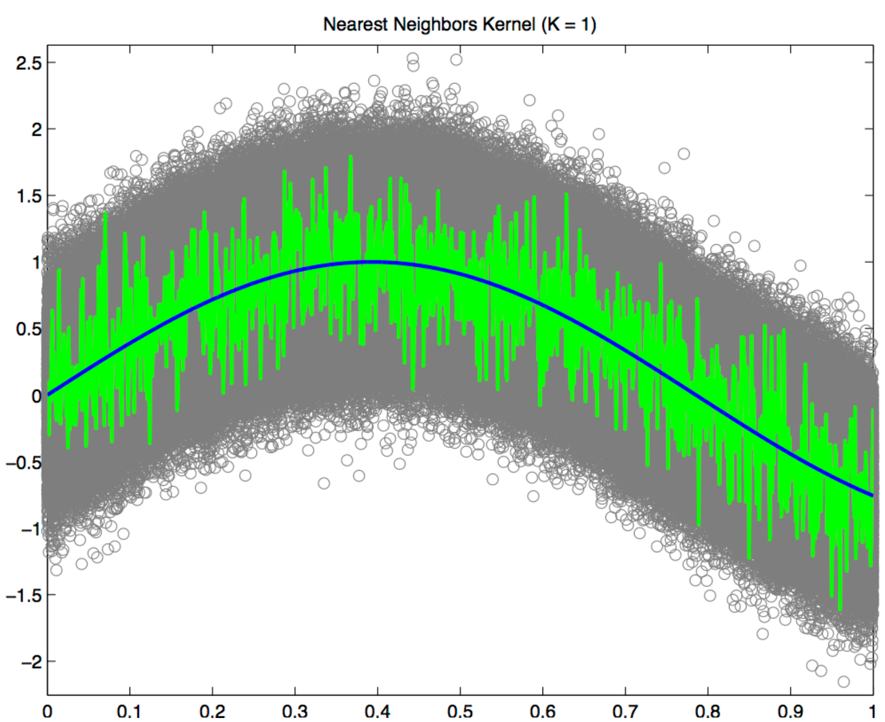


For a fixed model complexity

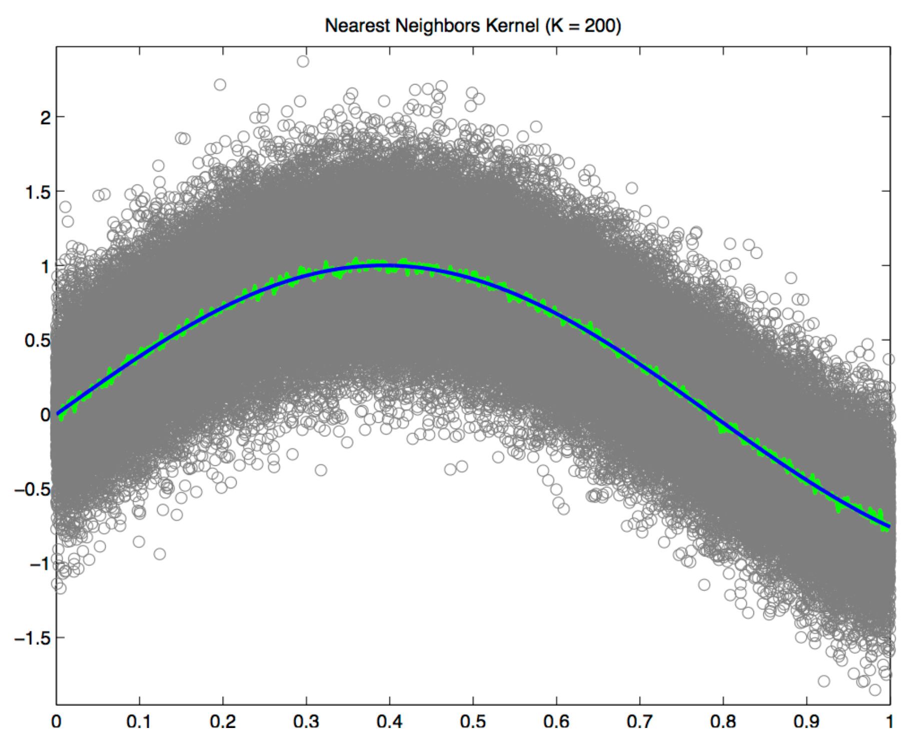


When there is noise,

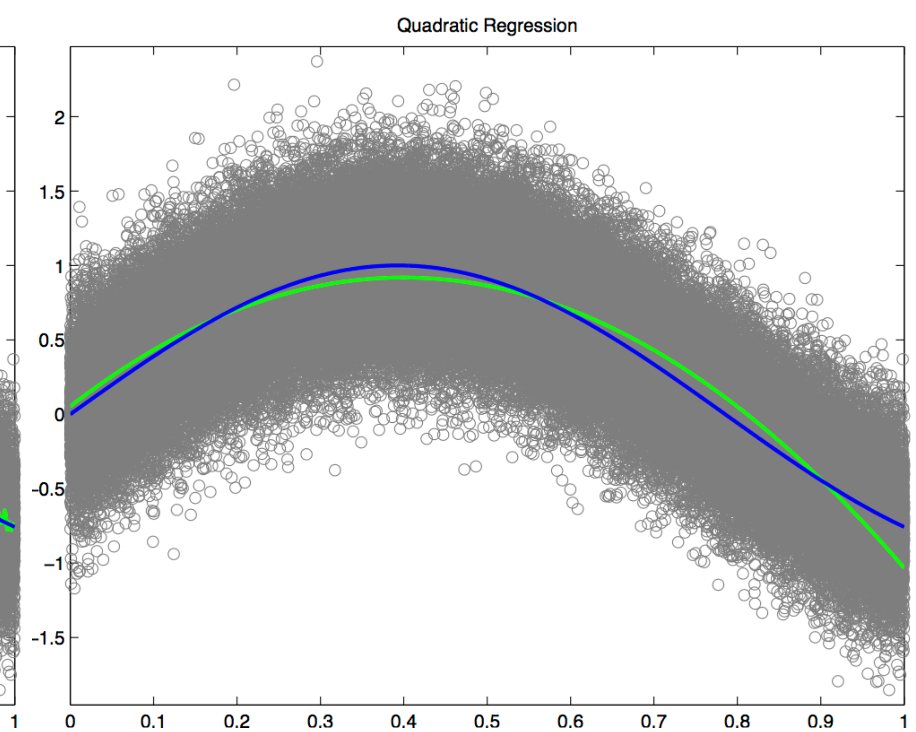
- In the limit of getting infinite data,
  MSE (Mean Squared Error) goes to zero,
  if k grows with N (usually choose k= log N)



1-NN fit



200-NN fit



Quadratic fit

- Non-parametric model with small k have non-vanishing error

- Non-parametric model with large enough k has vanishing error

- Parametric model had non-vanishing error

Is non-parametric perfect?

- Non-parametric methods require sample size N>exp(d), when data ***x*** is in ***d*** dimensions
- because, samples have to cover the volume of the space

- So depending on the sample size
  - If it is less, parametric models work better
  - If it is plenty, non-parametric models work well

- Non-parametric methods build upon local structure
  - Nearest neighbor search is central building block
  - Exact k-NN search takes N log(k) time
  - Can be improved with
    - KD-trees
    - Locality Sensitive Hashing