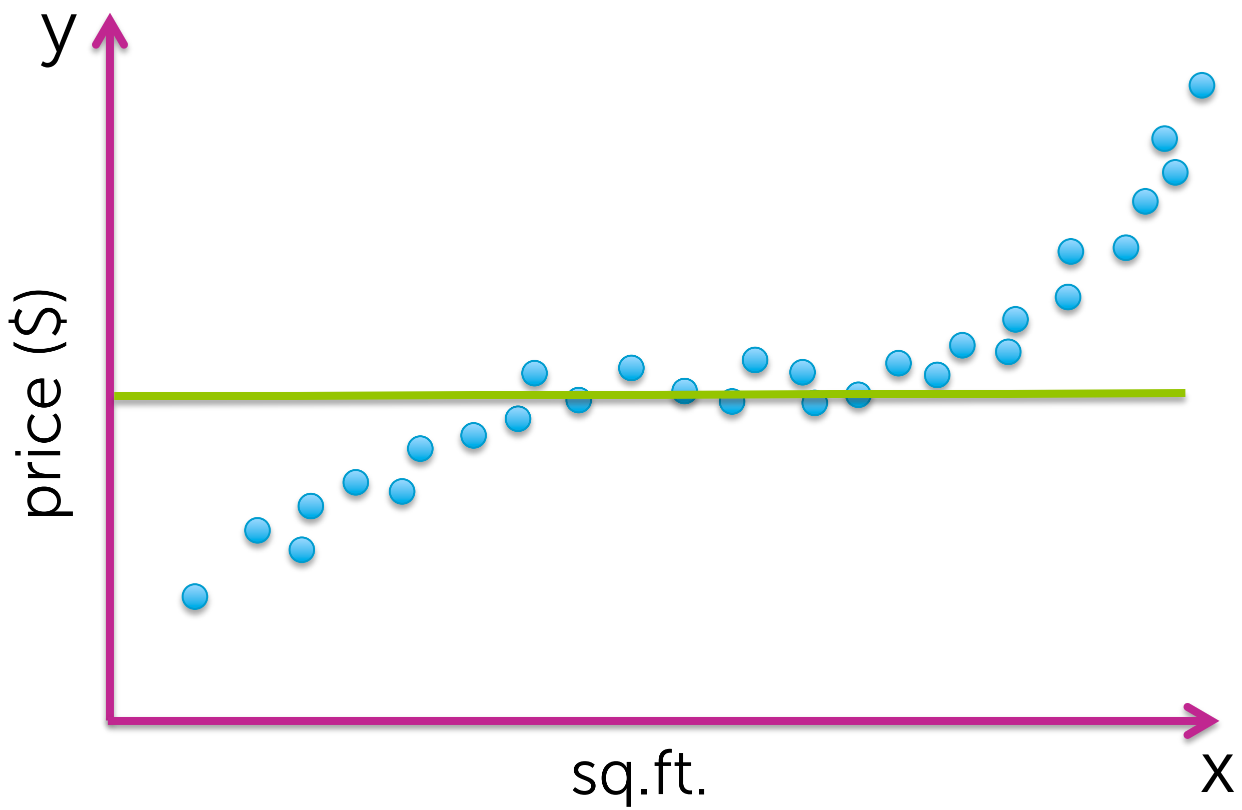
Going nonparametric:

Nearest neighbor and kernel regression

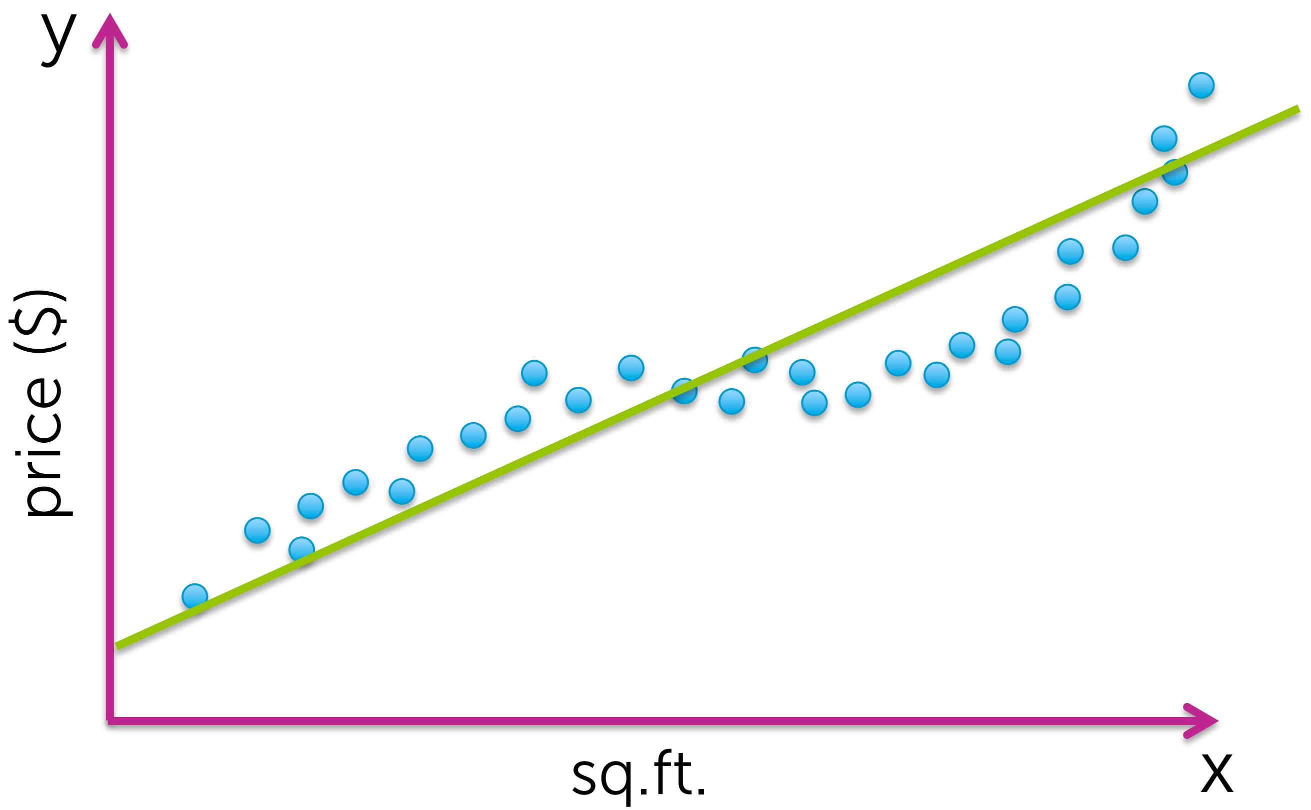
Emily Fox & Carlos Guestrin
Machine Learning Specialization
University of Washington

Fit globally vs. fit locally

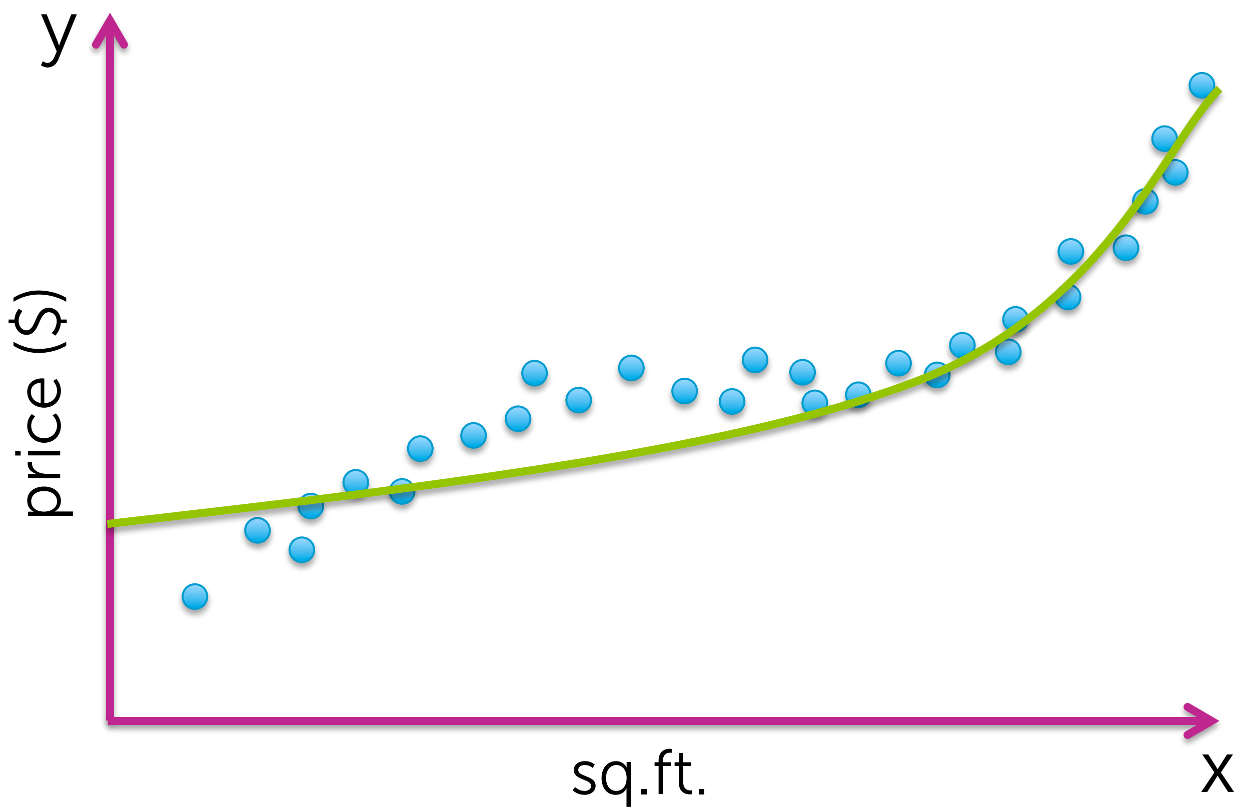
Parametric models of f(x)



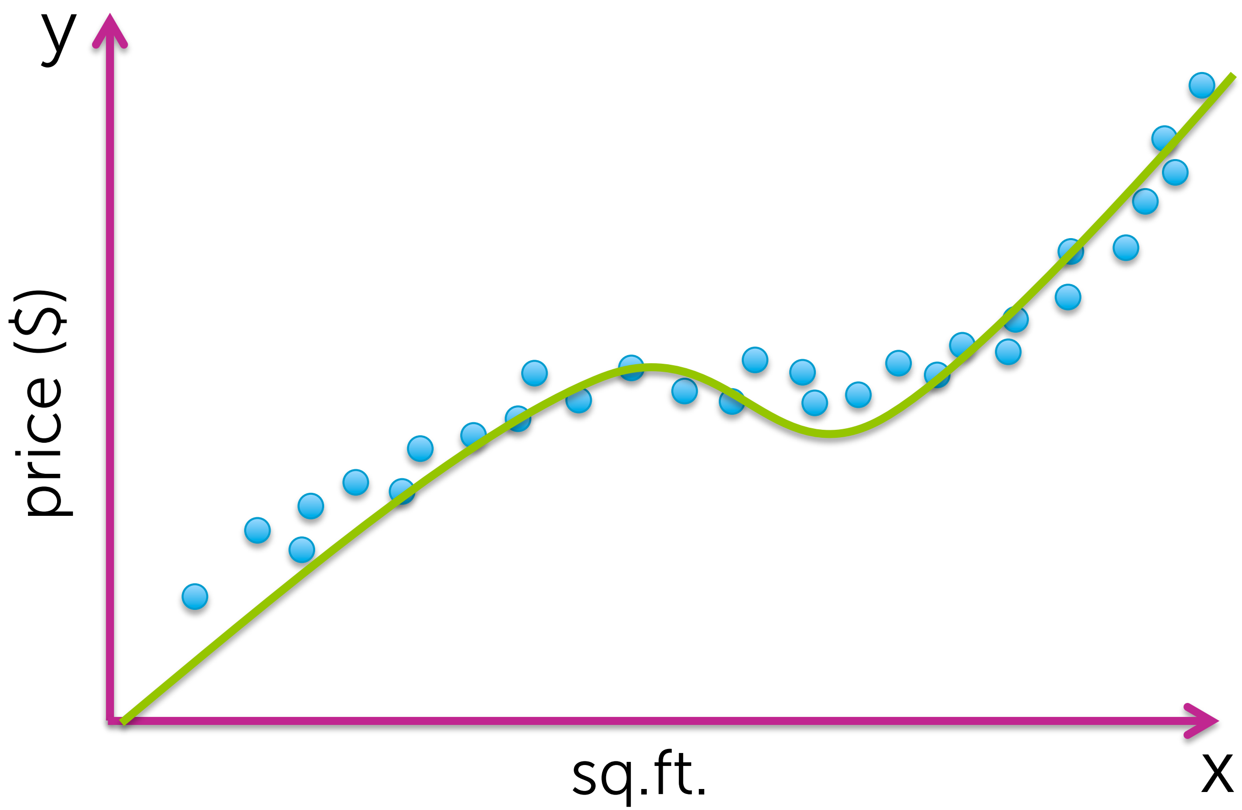
Parametric models of f(x)



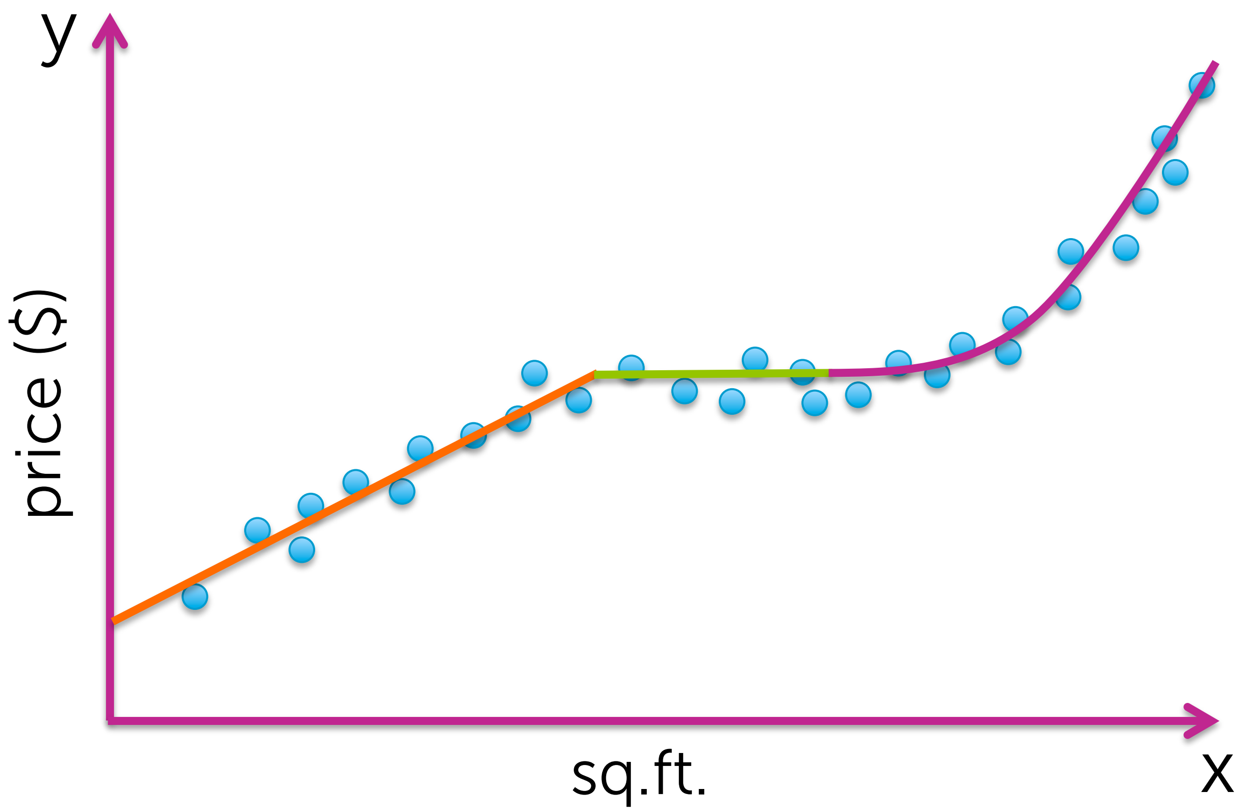
Parametric models of f(x)



Parametric models of f(x)



f(x) is not really a polynomial



What alternative do we have?

If we:

- Want to allow flexibility in f(**x**) having local structure
- Don't want to infer "structural breaks"

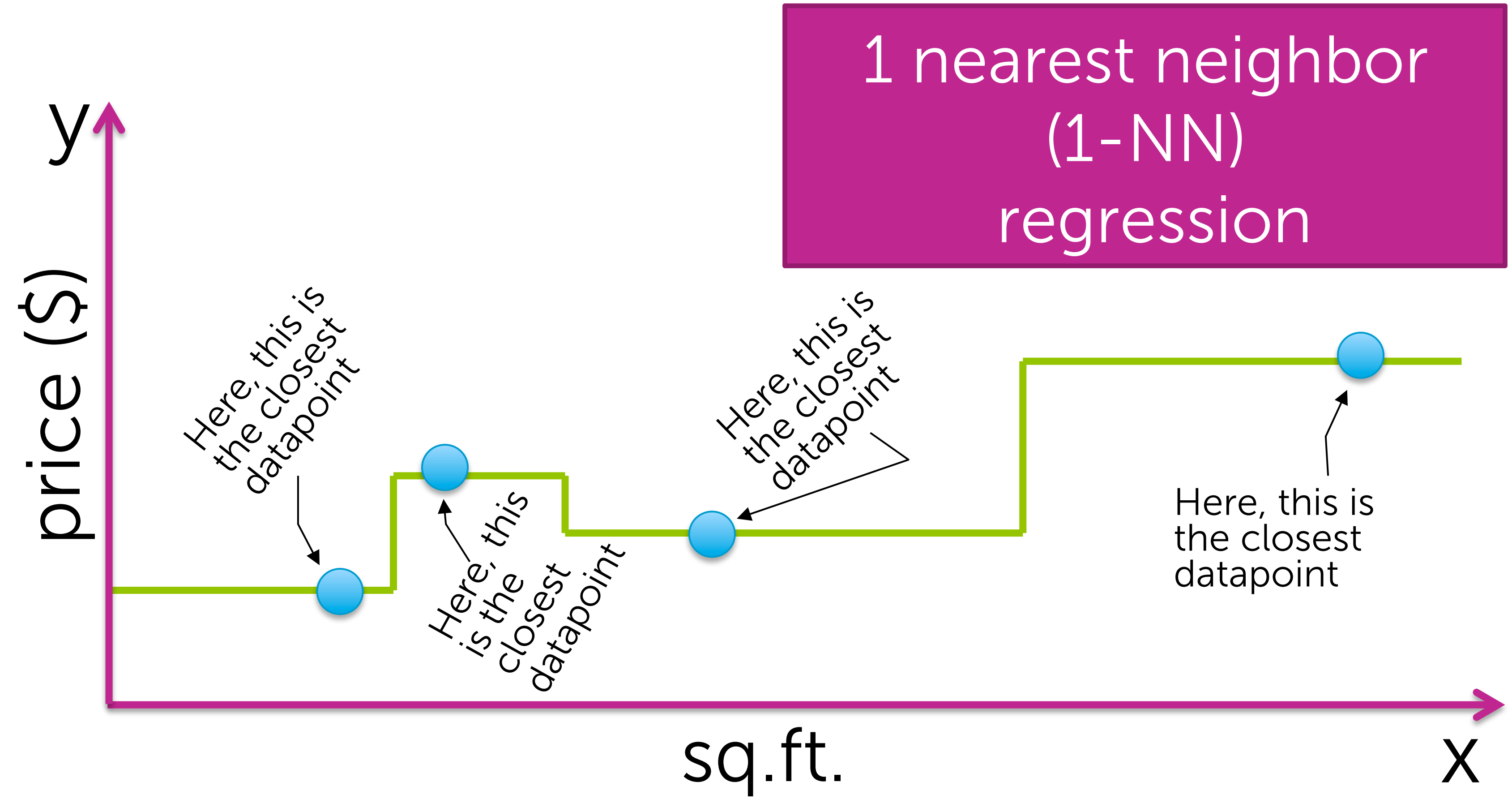
What's a simple option we have?

- Assuming we have plenty of data…

Simplest approach: Nearest neighbor regression

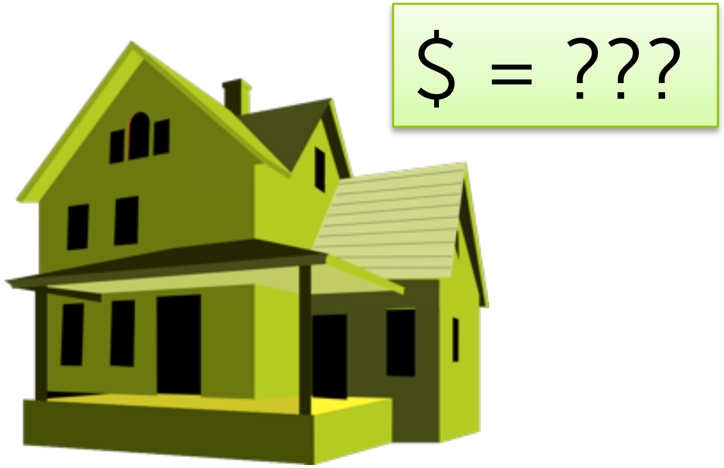
Fit locally to each data point

Predicted value = "closest" y_i



What people do naturally…

Real estate agent assesses value by finding sale of most similar house



1-NN regression more formally

Dataset of (🏠,\$) pairs: $(\mathbf{x}_1,y_1), (\mathbf{x}_2,y_2),\ldots,(\mathbf{x}_N,y_N)$

Query point: $\mathbf{x}_q$ ← \$?

big lime green house

1. Find "closest" $\mathbf{x}_i$ in dataset

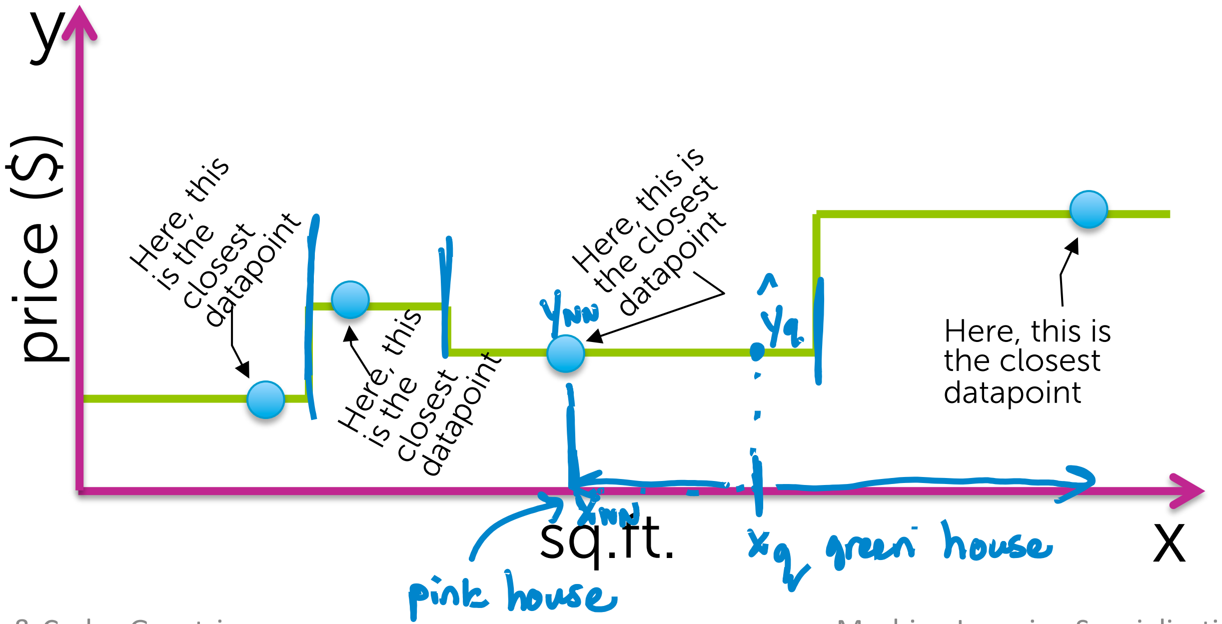
$$x_{NN} \leftarrow \min_i \text{distance}(x_i, x_q)$$

big pink house

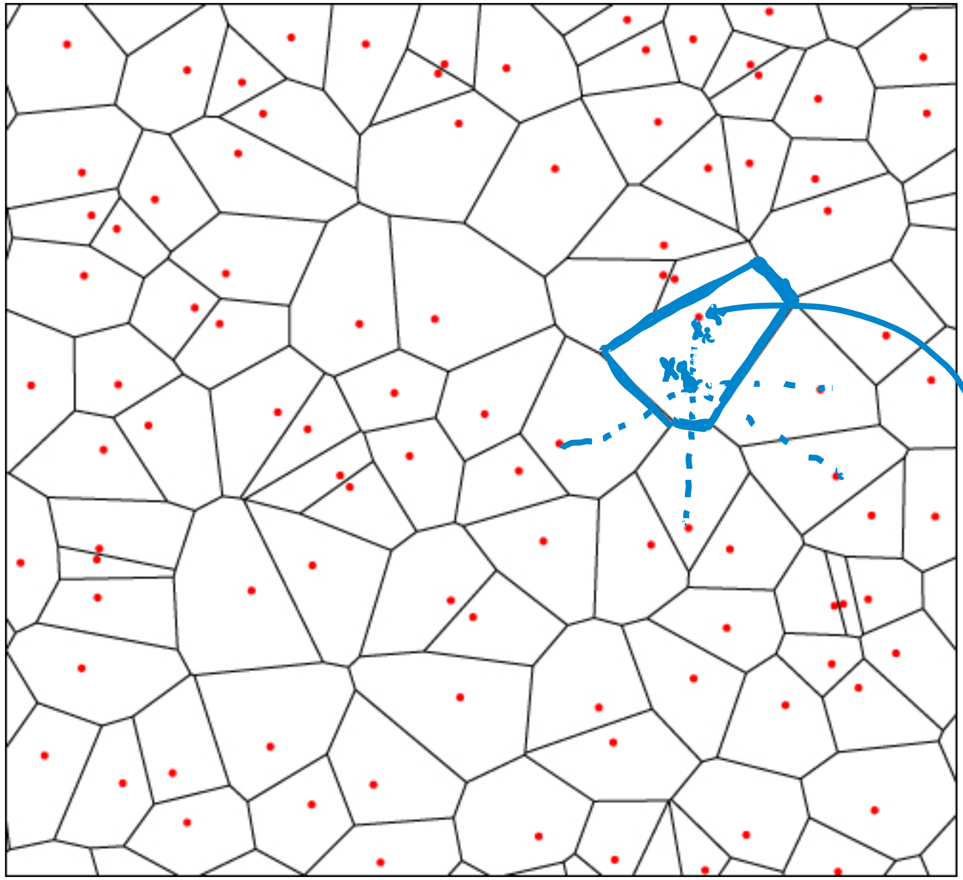
2. Predict

$$\hat{y}_q = y_{NN}$$

sales price of big pink house



Visualizing 1-NN in multiple dimensions



observation

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

x_q closer to x_i than any other x_j for $j \neq i$.

Don't explicitly form!

Distance metrics: Defining notion of "closest"

In 1D, just Euclidean distance:

$$\boxed{\text{distance}(x_j, x_q) = |x_j - x_q|}$$

In multiple dimensions:

- can define many interesting distance functions
- most straightforwardly, might want to weight different dimensions differently

Weighting housing inputs

Some inputs are more relevant than others

 # bedrooms
 # bathrooms
sq.ft. living
sq.ft. lot
floors
year built
year renovated
waterfront

Scaled Euclidean distance

Formally, this is achieved via

$$\text{distance}(\mathbf{x}_j, \mathbf{x}_q) = \sqrt{a_1(\mathbf{x}_j[1]-\mathbf{x}_q[1])^2 + \ldots + a_d(\mathbf{x}_j[d]-\mathbf{x}_q[d])^2}$$

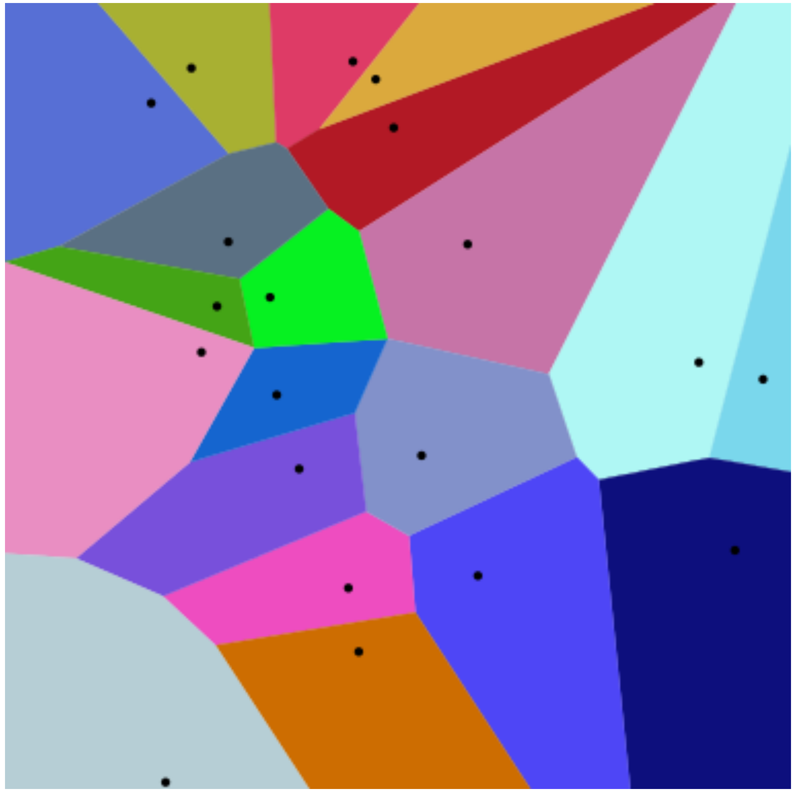
weight on each input
(defining relative importance)

Other example distance metrics:

- Mahalanobis, rank-based, correlation-based, cosine similarity, Manhattan, Hamming, …

Different distance metrics lead to different predictive surfaces

Euclidean distance



Manhattan distance

1-NN algorithm

Performing 1-NN search

- Query house: 

- Dataset: 

- **Specify:** Distance metric
- **Output:** Most similar house 

1-NN algorithm

closest house

Initialize **Dist2NN** = ∞,  = ∅

query house

For i=1,2,...,N

Compute: δ = **distance(** $_i$,  $_q$ **)**

If δ < **Dist2NN**

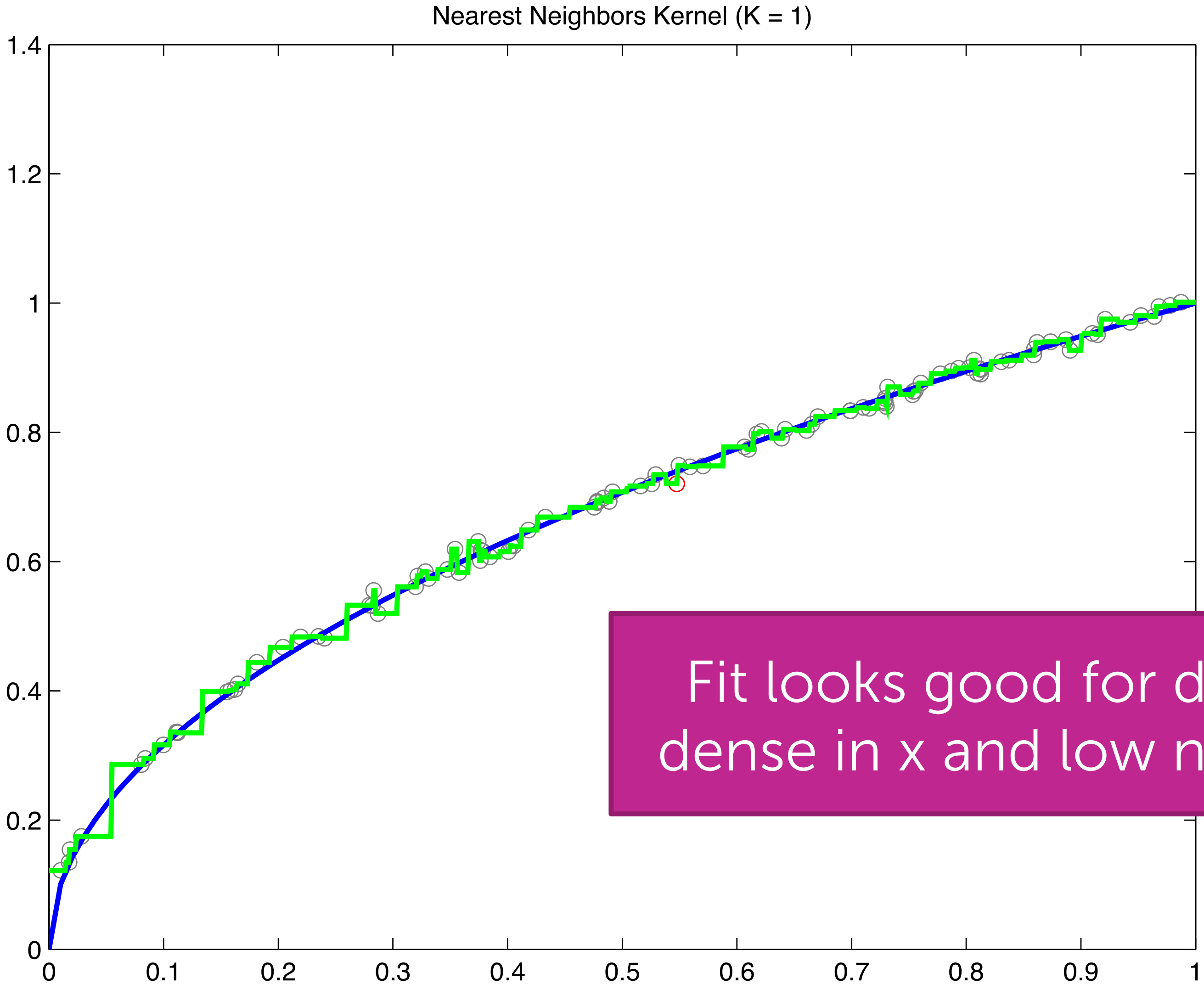
set   $_i$

set **Dist2NN** = δ

Return most similar house 

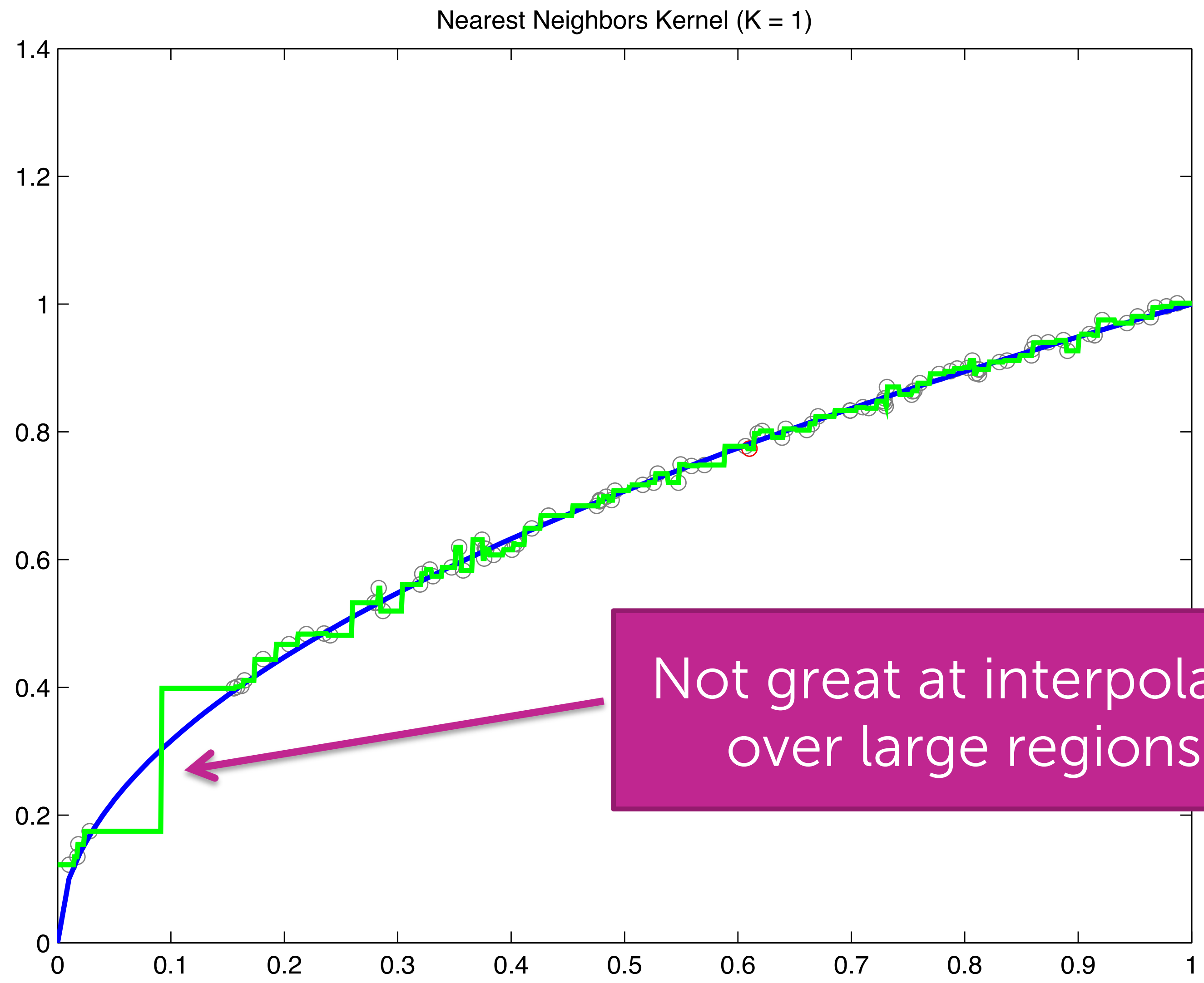
closest house
to query house

1-NN in practice



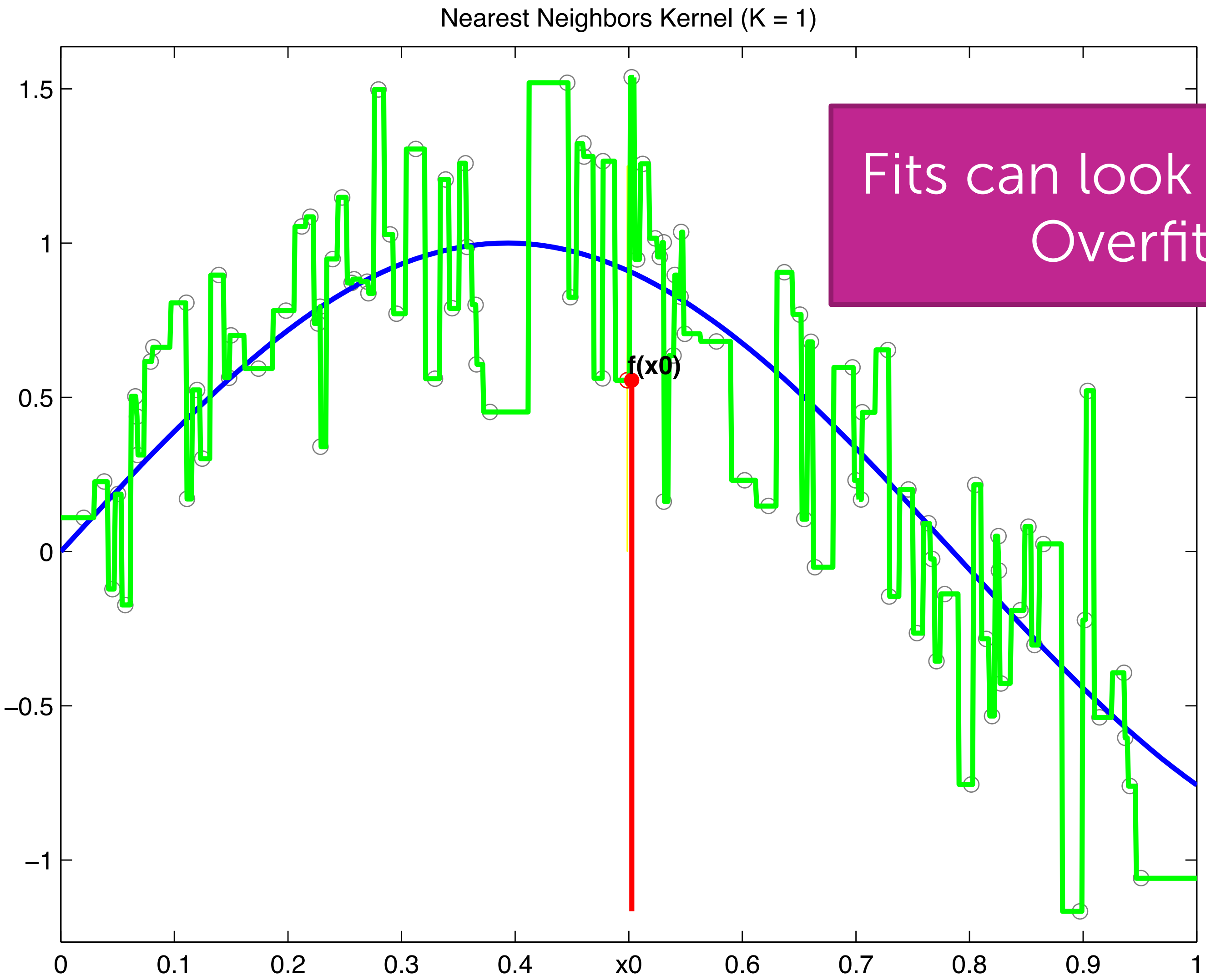
Fit looks good for data dense in x and low noise

Sensitive to regions with little data



Not great at interpolating over large regions…

Also sensitive to noise in data

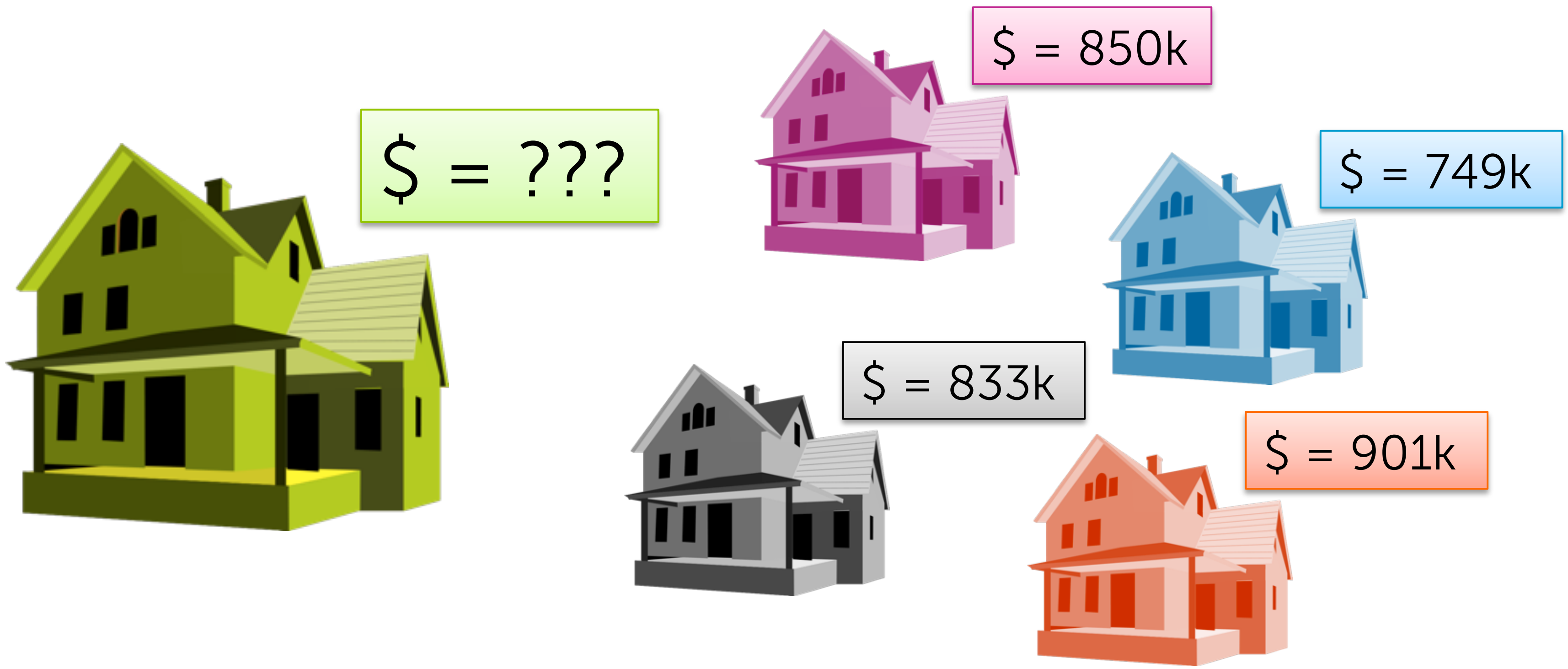


Fits can look quite wild…
Overfitting?

k-Nearest neighbors

Get more "comps"

More reliable estimate if you base estimate off of a larger set of comparable homes



k-NN regression more formally

Dataset of (,$) pairs: $(\mathbf{x}_1,y_1), (\mathbf{x}_2,y_2),\ldots,(\mathbf{x}_N,y_N)$

Query point: $\mathbf{x}_q$

1. Find k closest $\mathbf{x}_i$ in dataset

   $(x_{NN_1}, x_{NN_2}, \ldots, x_{NN_k})$ such that for any x_i not in nearest neighbor set,

   $$\text{distance}(x_i, x_q) \geq \text{distance}(x_{NN_k}, x_q)$$

2. Predict

   $$\hat{y}_q = \frac{1}{k}(y_{NN_1} + y_{NN_2} + \ldots + y_{NN_k})$$

   $$= \frac{1}{k}\sum_{j=1}^{k} y_{NN_j}$$

Performing k-NN search

- Query house: 
- Dataset: 
- **Specify:** Distance metric
- **Output:** Most similar houses

k-NN algorithm

sort first **k houses**
by distance to query house

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

= so ,…, $_1$ $_k$ ← 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] = [j:k

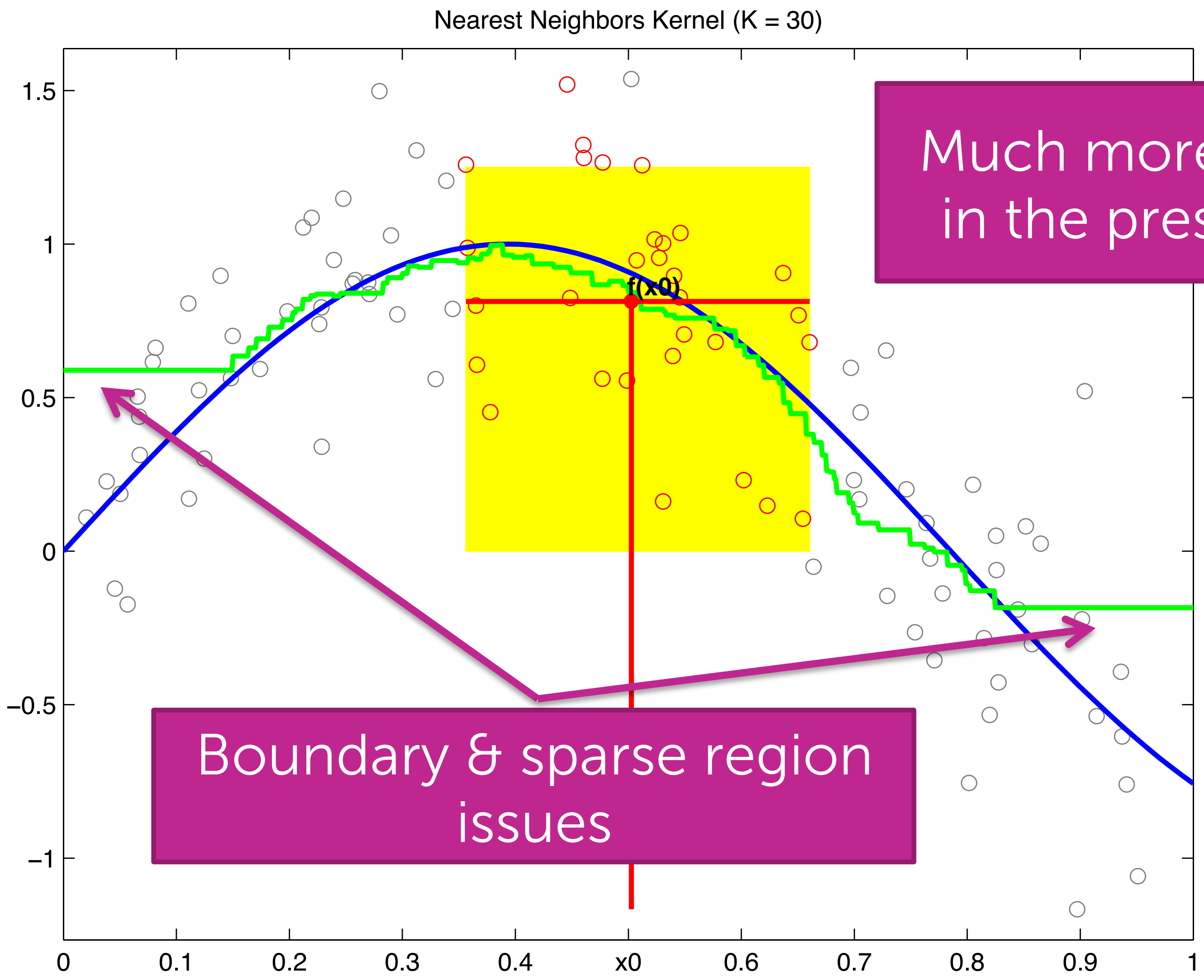
insert new NN

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

set **Dist2kNN**[j] = δ and = $_i$

Return k most similar houses ← closest houses to query house

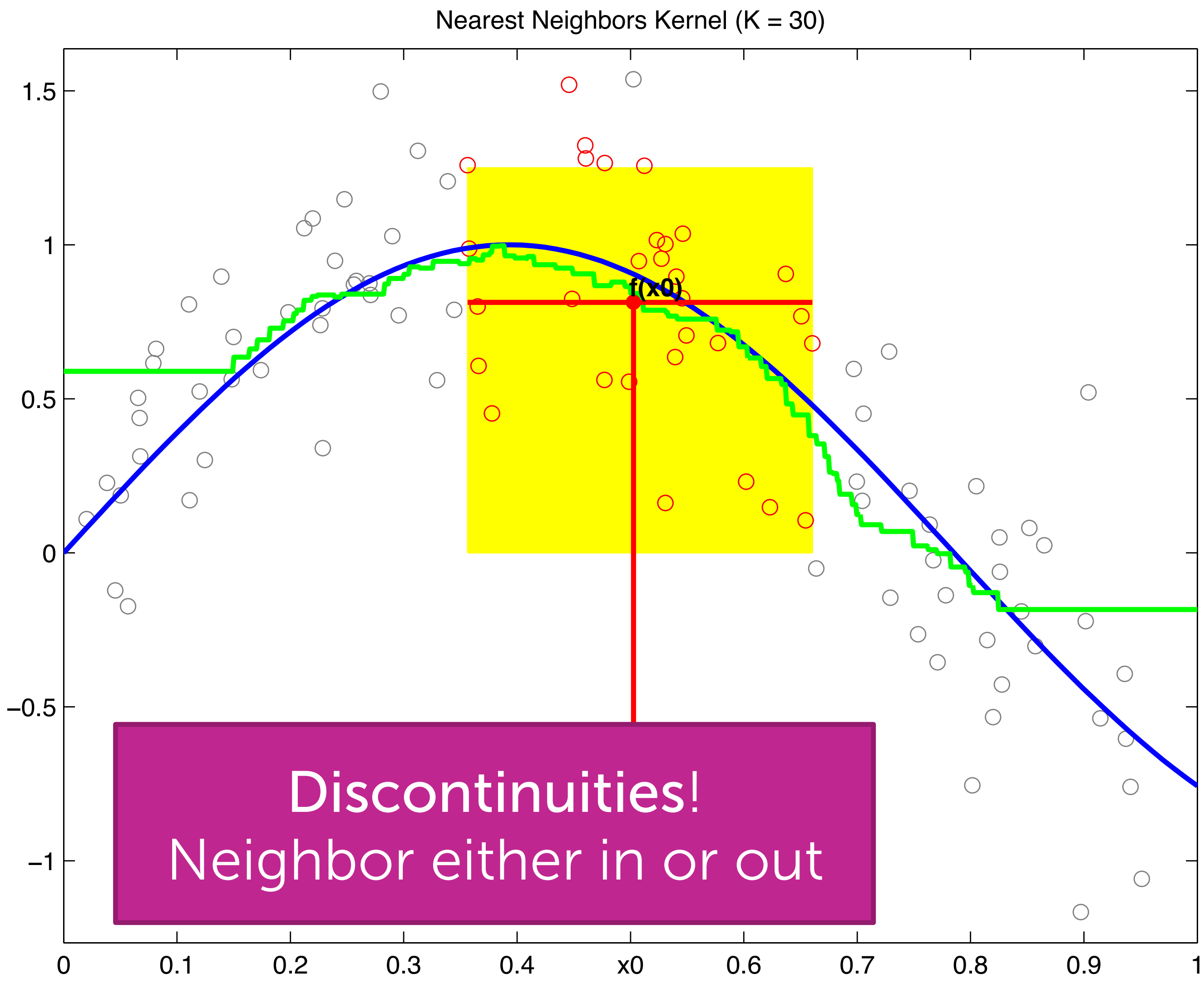
k-NN in practice



Much more reasonable fit in the presence of noise

Boundary & sparse region issues

k-NN in practice



Discontinuities!
Neighbor either in or out

Issues with discontinuities

Overall predictive accuracy might be okay, but…

For example, in housing application:

- If you are a buyer or seller, this matters
- Can be a jump in estimated value of house going just from 2640 sq.ft. to 2641 sq.ft.
- Don't really believe this type of fit

Weighted k-nearest neighbors

Weighted k-NN

Weigh more similar houses more than those less similar in list of k-NN

Predict:

weights on NN

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

How to define weights?

Want weight c_{qNNj} to be small when
  $\text{distance}(\mathbf{x}_{NNj},\mathbf{x}_q)$ large

and c_{qNNj} to be large when
  $\text{distance}(\mathbf{x}_{NNj},\mathbf{x}_q)$ small

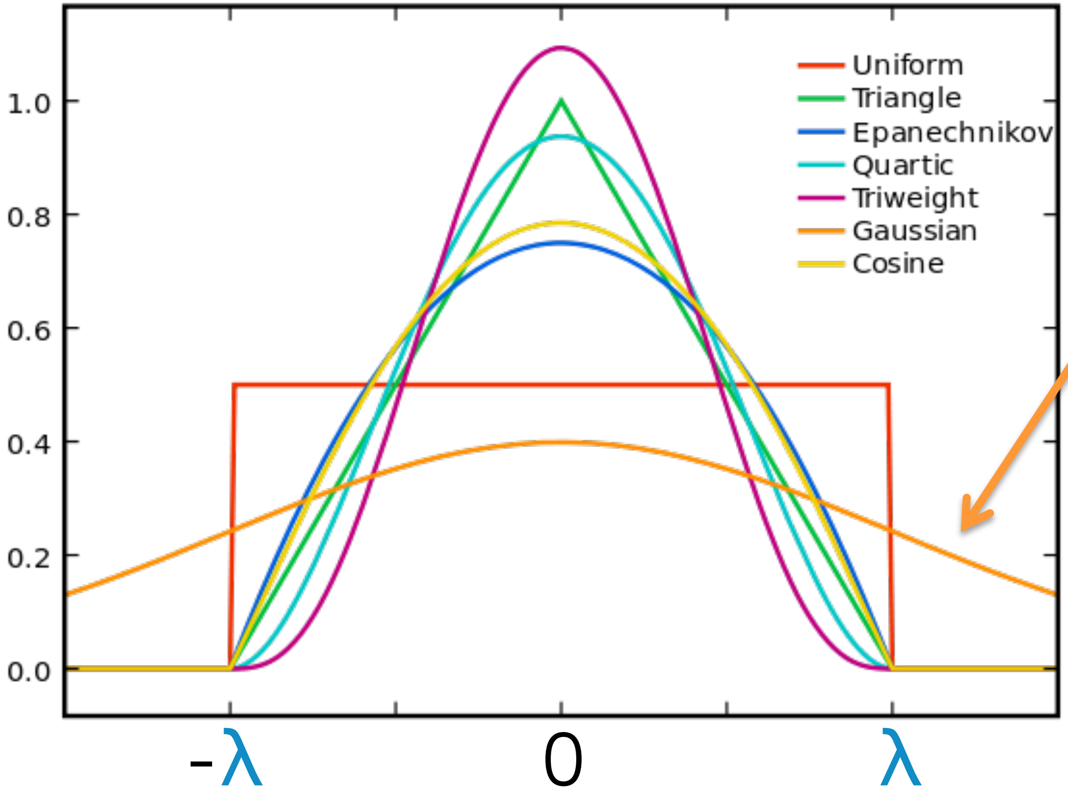
Simple method:

$$c_{qNN_j} = \frac{1}{\text{distance}(x_j, x_q)}$$

Kernel weights for d=1

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

simple isotropic case



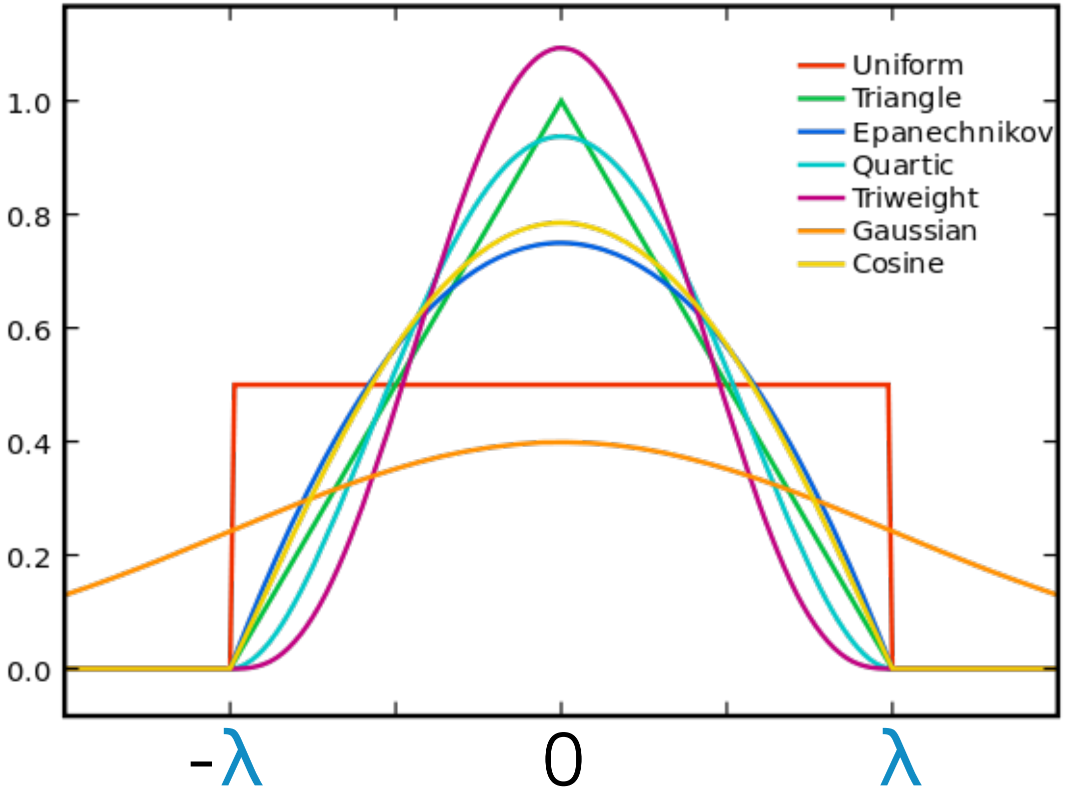
Gaussian kernel:

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

Note: never exactly 0!

Kernel weights for d≥1

Define: $c_{qNNj} = \text{Kernel}_{\lambda}(\text{distance}(\mathbf{x}_{NNj},\mathbf{x}_q))$



Kernel regression

Weighted k-NN

Weigh more similar houses more than those less similar in list of k-NN

Predict:

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

Nadaraya-Watson kernel weighted average

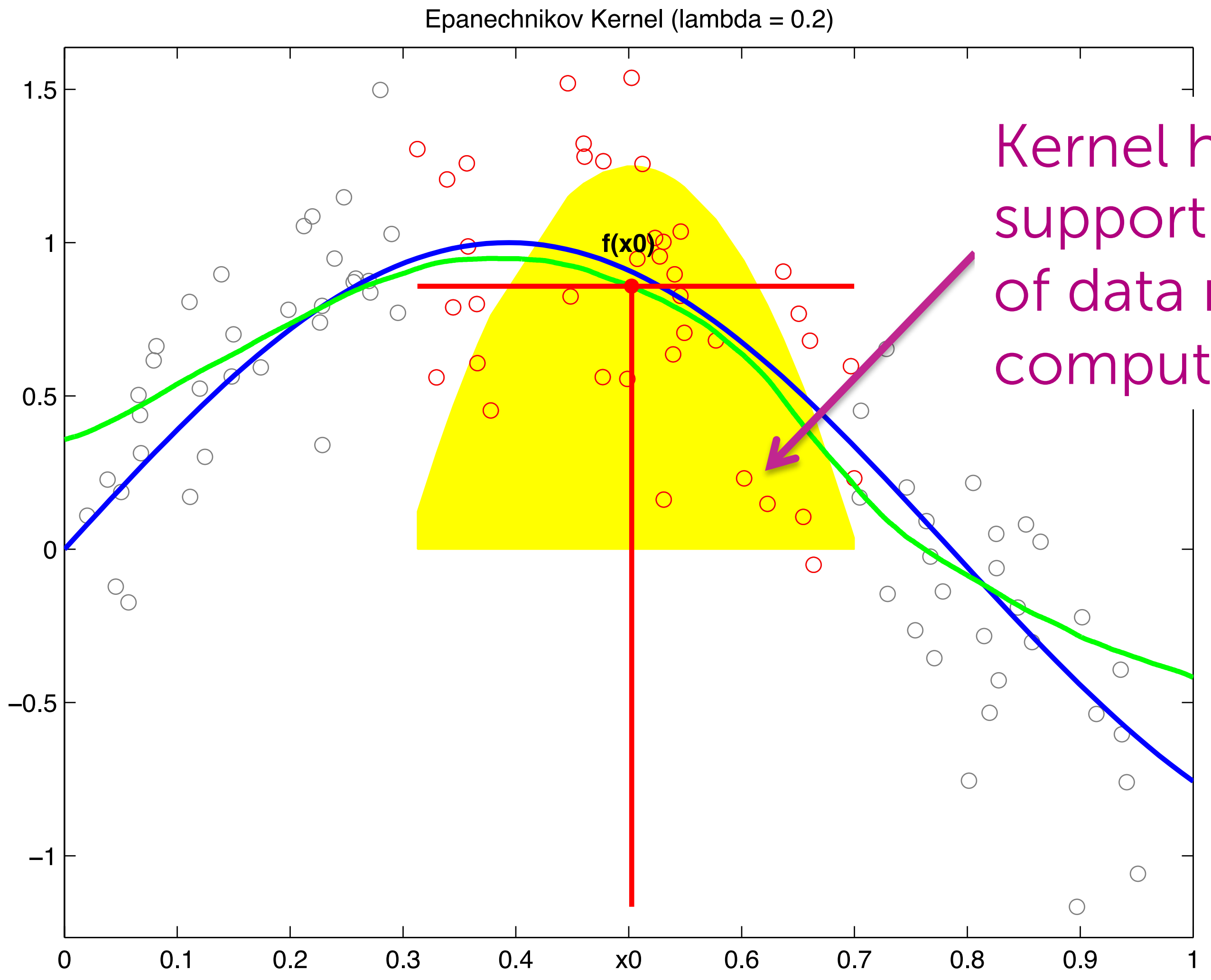
Instead of just weighting NN, weight *all* points

Predict:

weight on each datapoint

$$\hat{y}_q = \frac{\sum_{i=1}^{N} c_{qi} y_i}{\sum_{i=1}^{N} c_{qi}} = \frac{\sum_{i=1}^{N} \text{Kernel}_\lambda(\text{distance}(\mathbf{x}_i,\mathbf{x}_q)) * y_i}{\sum_{i=1}^{N} \text{Kernel}_\lambda(\text{distance}(\mathbf{x}_i,\mathbf{x}_q))}$$

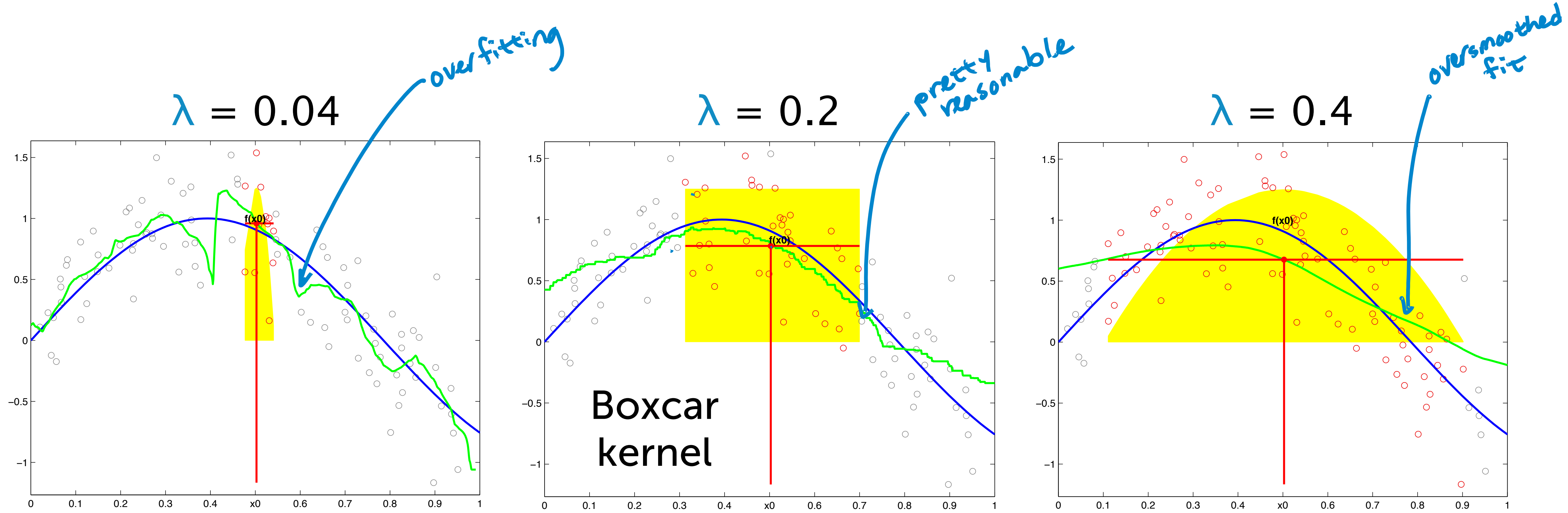
Kernel regression in practice



Kernel has bounded support…Only subset of data needed to compute local fit

Choice of bandwidth λ

Often, choice of kernel matters much less than choice of λ



Choosing λ (or k in k-NN)

How to choose? Same story as always…

Cross Validation

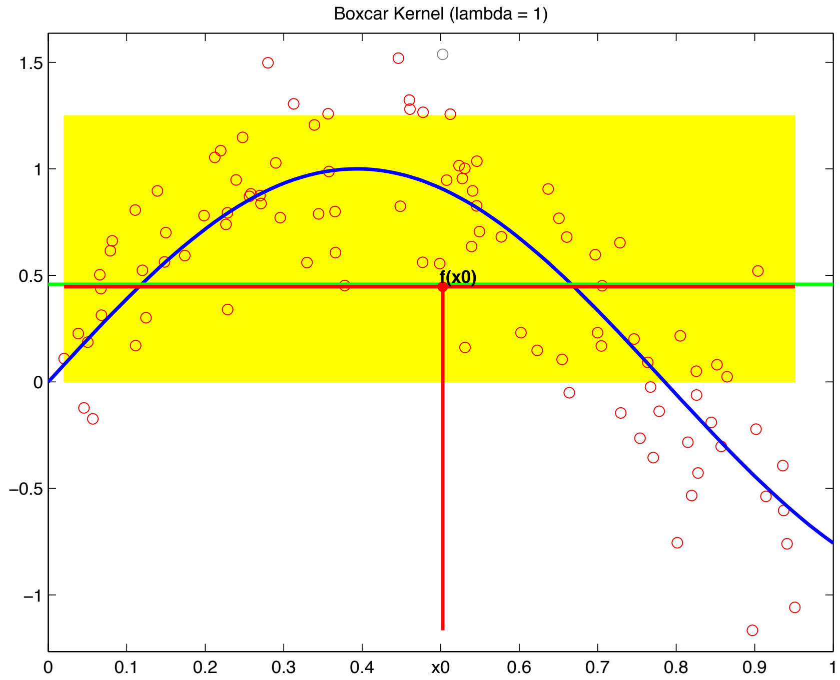
Formalizing the idea of local fits

Contrasting with global average

A globally constant fit weights all points equally

equal weight on each datapoint

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

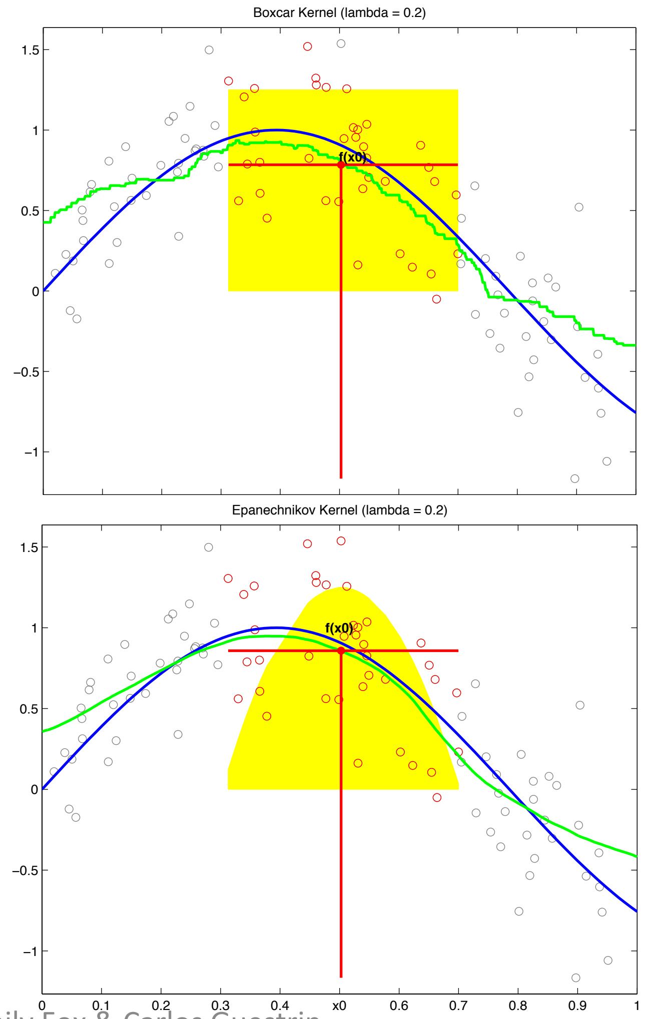


Contrasting with global average

Kernel regression leads to locally constant fit

– slowly add in some points and and let others gradually die off

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



Local linear regression

So far, discussed fitting constant function locally at each point

→ "locally weighted averages"

Can instead fit a line or polynomial locally at each point

→ "locally weighted linear regression"

Local regression rules of thumb

- 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
- With sufficient data, local polynomials of odd degree dominate those of even degree

Recommended default choice:

local linear regression

Discussion on k-NN and kernel regression

Nonparametric approaches

k-NN and kernel regression are examples of nonparametric regression

General goals of nonparametrics:

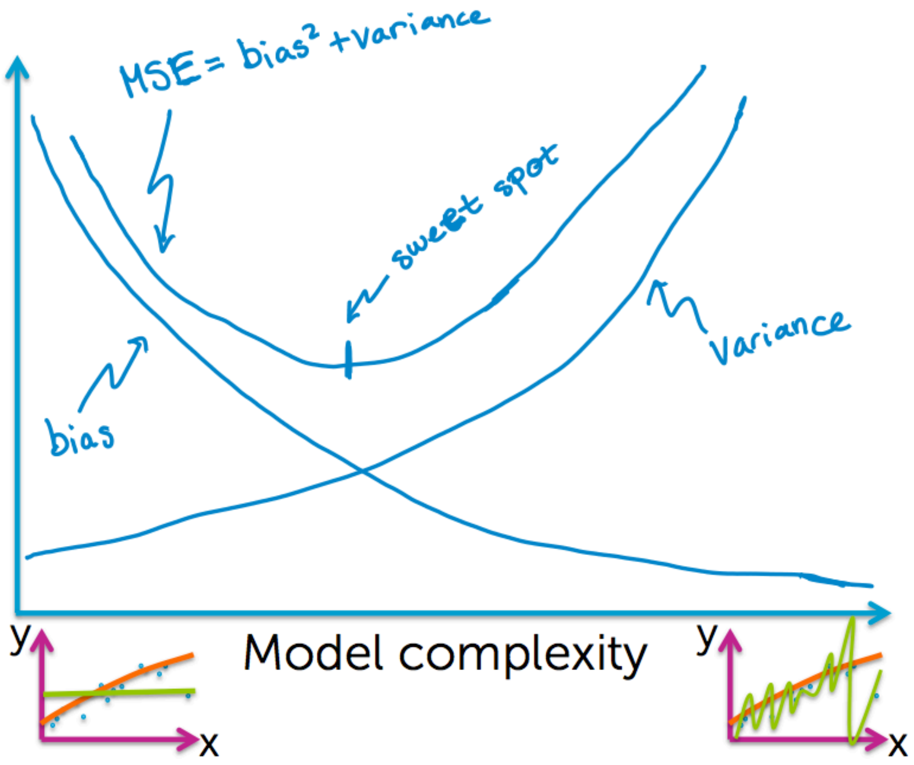
- Flexibility
- Make few assumptions about f(**x**)
- Complexity can grow with the number of observations N

Lots of other choices:

- Splines, trees, locally weighted structured regression models…

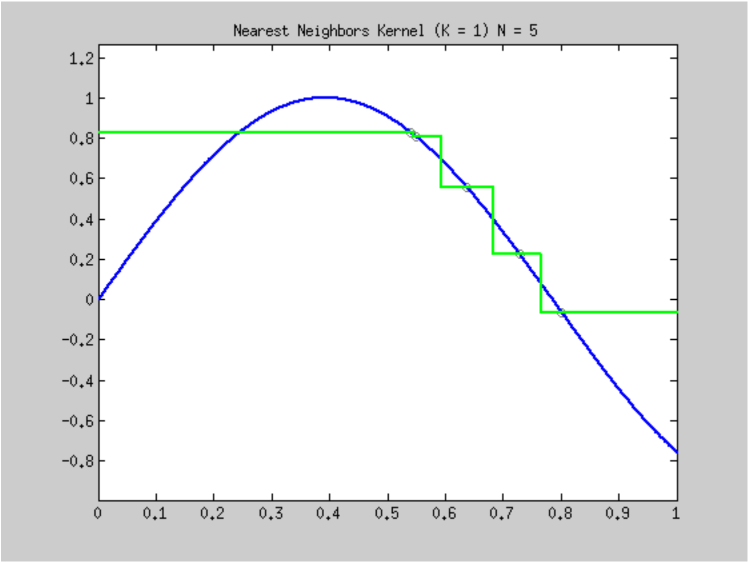
Limiting behavior of NN: Noiseless setting (ε_i=0)

In the limit of getting an infinite amount of noiseless data, the MSE of 1-NN fit goes to 0

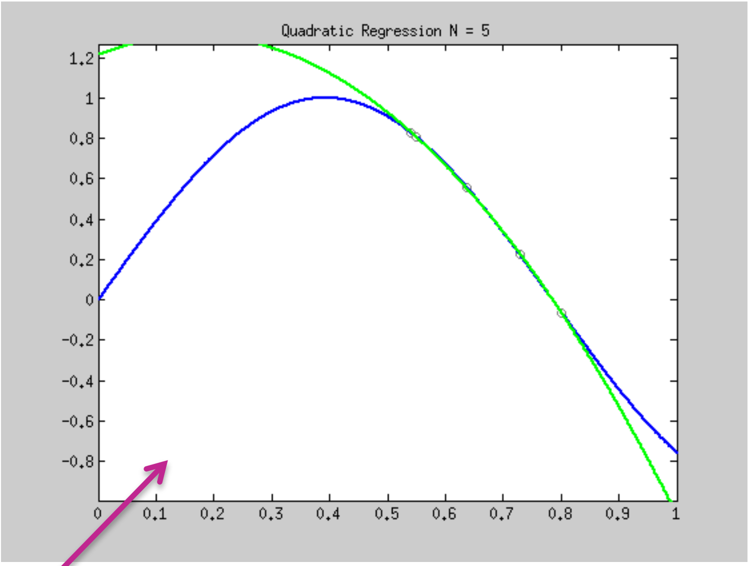


Limiting behavior of NN: Noiseless setting (ε_i=0)

In the limit of getting an infinite amount of noiseless data, the MSE of 1-NN fit goes to 0



1-NN fit

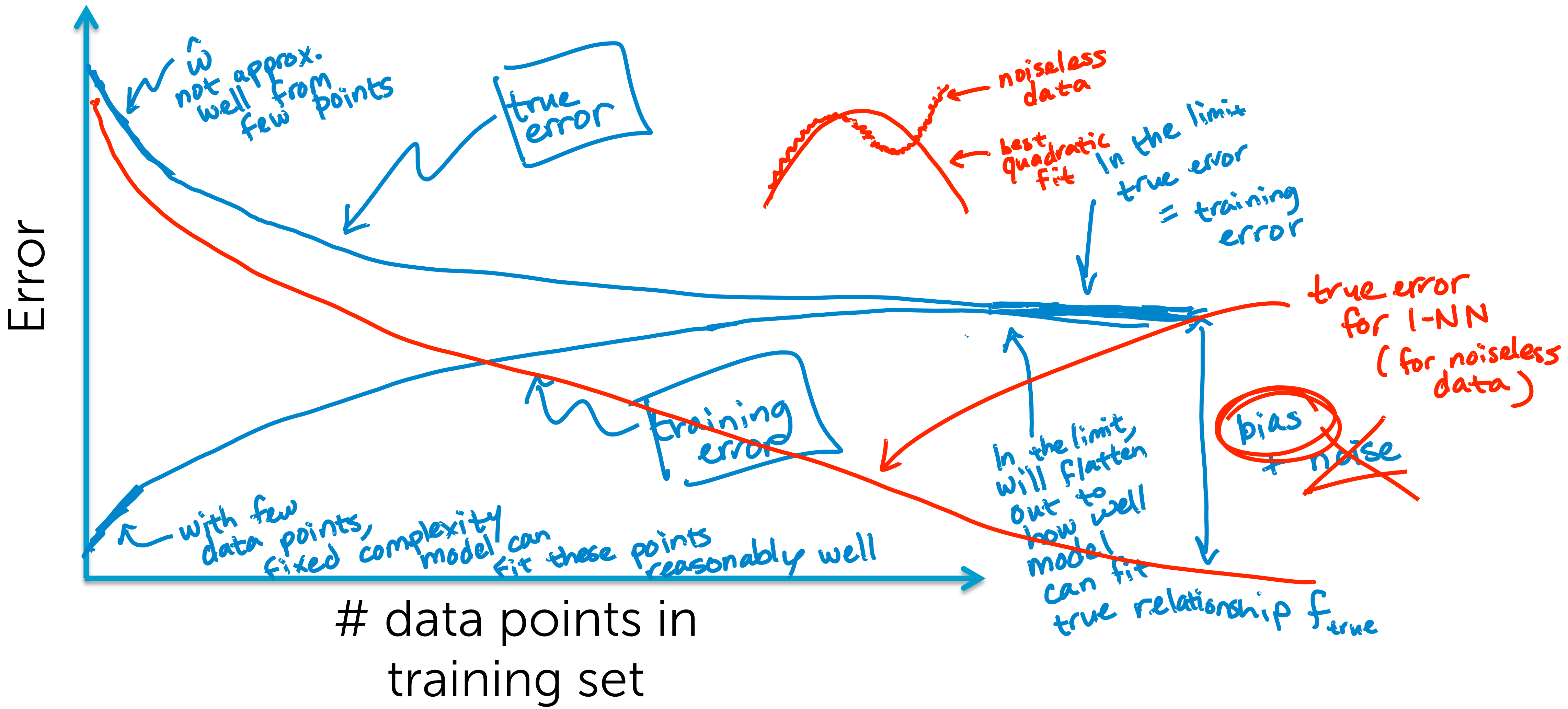


Quadratic fit

Not true for parametric models!

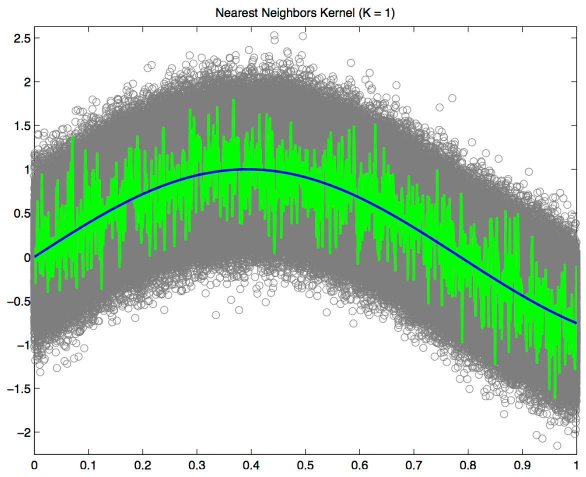
Error vs. amount of data

for a fixed model complexity

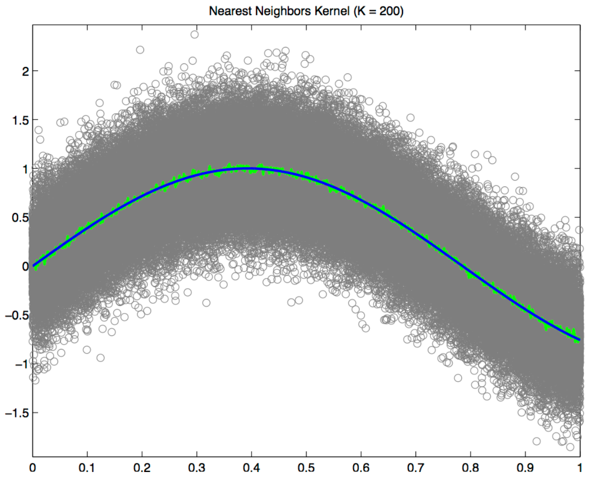


Limiting behavior of NN: Noisy data setting

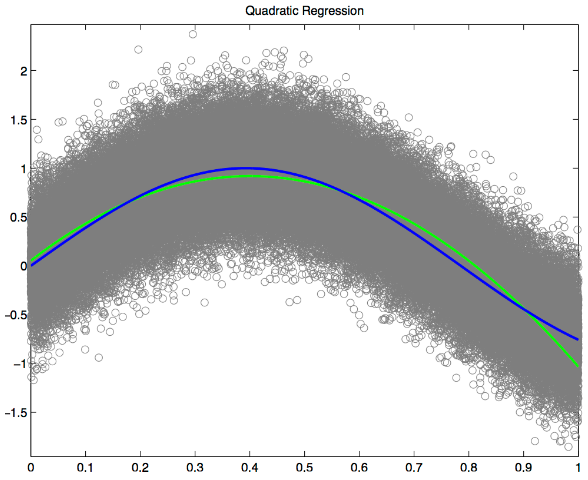
In the limit of getting an infinite amount of data, the MSE of NN fit goes to 0 if k grows, too



1-NN fit



200-NN fit



Quadratic fit

NN and kernel methods for large d or small N

NN and kernel methods work well when the data cover the space, but…

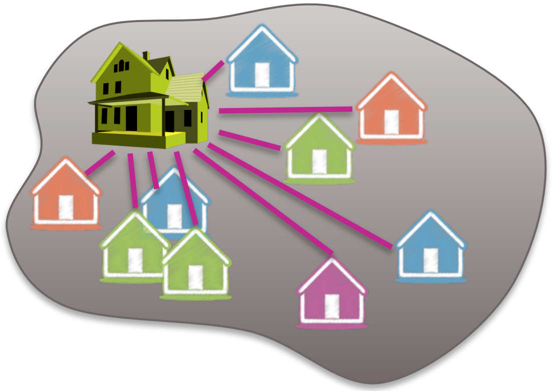
- the more dimensions d you have, the more points N you need to cover the space
- need $N = O(\exp(d))$ data points for good performance

This is where parametric models become useful…

Complexity of NN search

Naïve approach: **Brute force search**

- Given a query point $\mathbf{x}_q$
- Scan through each point $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$
- O(N) distance computations per 1-NN query!
- O(Nlogk) per k-NN query!

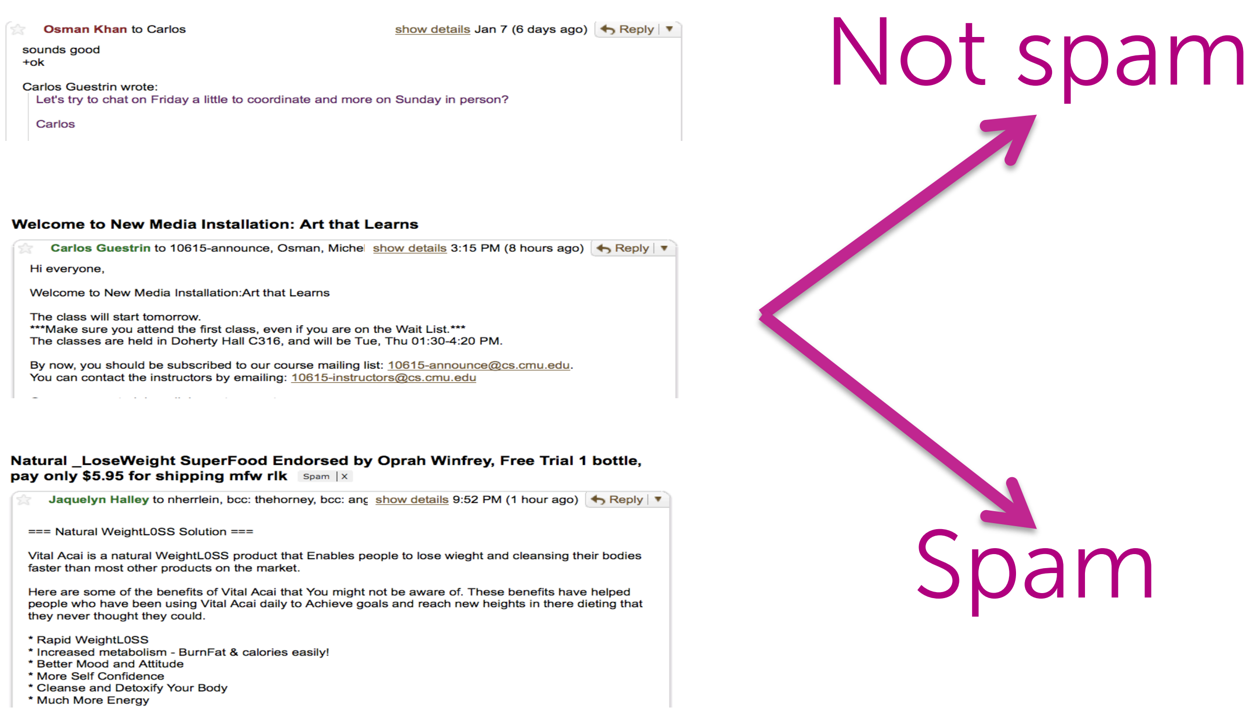


What if N is huge???
(and many queries)

Will talk more about efficient methods in
Clustering & Retrieval course

k-NN for classification

Recall classification task: Spam filtering example



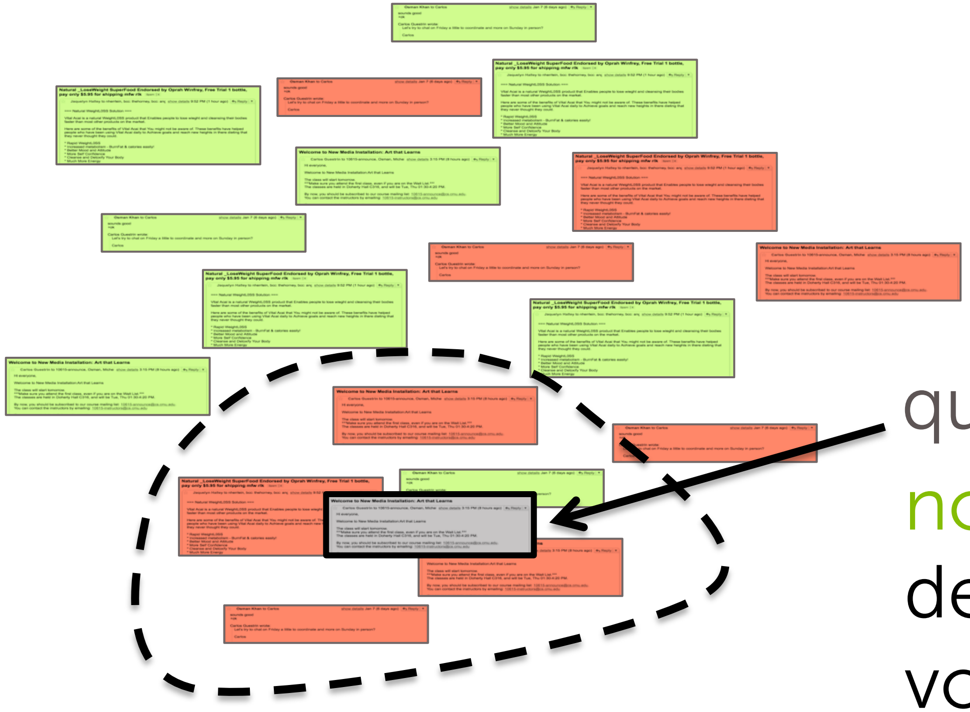
Input: **x**

Output: y

Text of email, sender, IP,...

Using k-NN for classification

Space of labeled emails (not spam vs. spam), organized by similarity of text



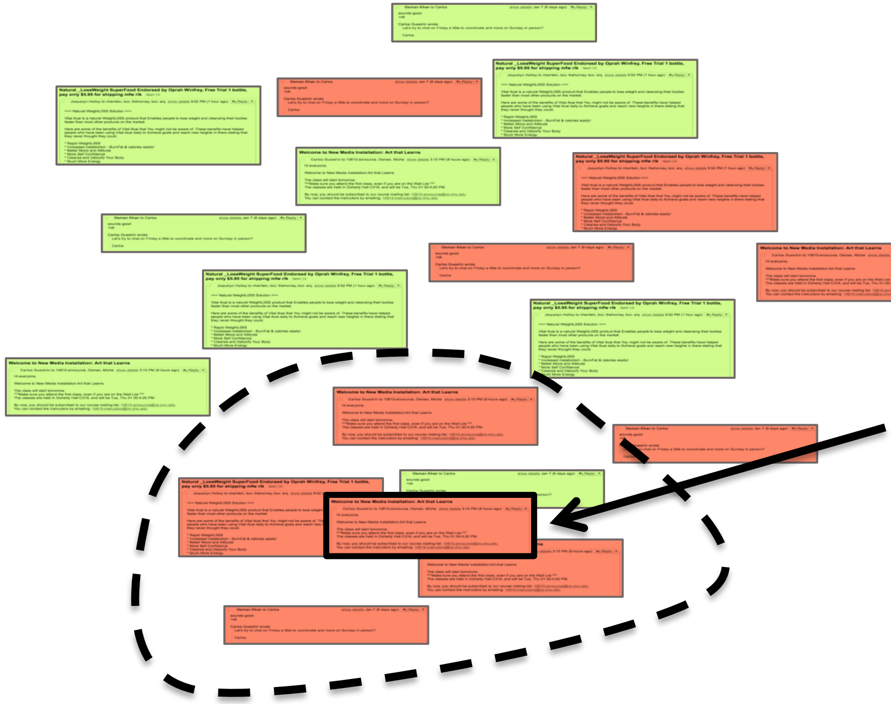
query email

not spam vs. spam

decide via majority vote of k-NN

Using k-NN for classification

Space of labeled emails (not spam vs. spam), organized by similarity of text



query email

not spam vs. $\boxed{\text{spam}}$

decide via majority vote of k-NN

Summary for nearest neighbor and kernel regression

What you can do now…

- Motivate the use of nearest neighbor (NN) regression
- Define distance metrics in 1D and multiple dimensions
- Perform NN and k-NN regression
- Analyze computational costs of these algorithms
- Discuss sensitivity of NN to lack of data, dimensionality, and noise
- Perform weighted k-NN and define weights using a kernel
- Define and implement kernel regression
- Describe the effect of varying the kernel bandwidth λ or # of nearest neighbors k
- Select λ or k using cross validation
- Compare and contrast kernel regression with a global average fit
- Define what makes an approach nonparametric and why NN and kernel regression are considered nonparametric methods
- Analyze the limiting behavior of NN regression
- Use NN for classification