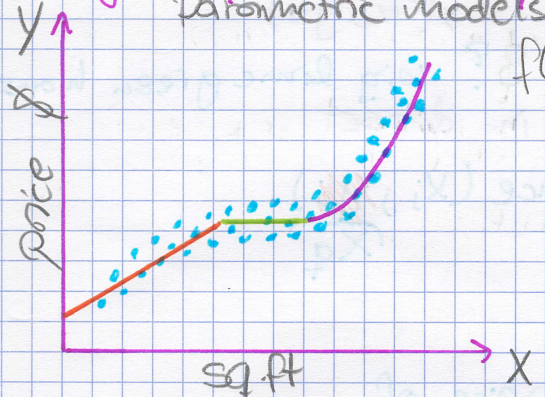
Nearest Neighbors & Kernel Regression

Limitations of parametric regression

Fit globally vs. fit locally

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"

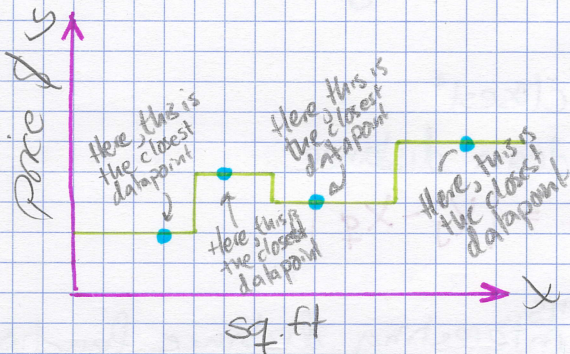
What's a simple option we have?

- Assuming we have plenty of data...

1-Nearest neighbor regression approach

Fit locally to each data point

Predicted value = "closest" y_i



1 nearest neighbor (1-NN) regression

1-NN regression more formally

Dataset of (🏠, $) pairs: $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$

Query point: x_q 🏠 $? big lime green house

1. Find "closest" x_i in dataset

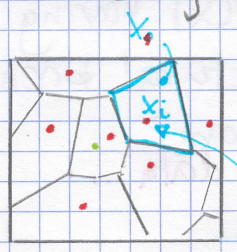
$$x_{NN} \leftarrow \min_j \text{distance}(x_j, x_q)$$

big pink house

2. Predict

$$\hat{y}_q = y_{NN}$$ sales price of big pink house

Visualizing 1-NN in multiple dimensions



Don't explicitly form!

Voronoi tessolation (or diagram)

- Divide space into N regions, each containing 1 datapoint
- Defined such that any x in region i is "closest" to region i's datapoint

observation

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

Distance metrics: Defining notion of "closest"

In 1D, just Euclidean distance:

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

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

Scaled Euclidean distance
Formally, this is achieved via

$$distance(x_j, x_q) = \sqrt{a_1(x_j[1] - x_q[1])^2 + \ldots + a_d[x_j[d] - x_q[d]]^2}$$

- #bedrooms
- #bathrooms
- sqft. living
- sqft. lot
- floors
- year built
- year renovated
- waterfront

a_1: weight on each input (defining relative importance)

Other example distance metrics:
- Mahalanobis, rank-based, correlation-based, cosine similarity, Manhattan, Hamming, ...

1-NN algorithm

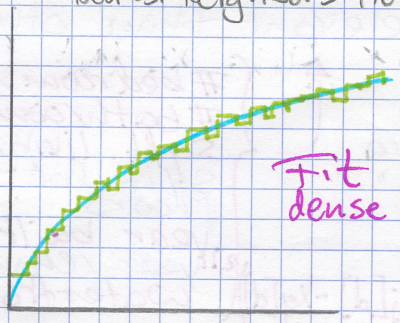
Performing 1-NN search

- Query house:
- Dataset:

- Specify: Distance metric
- Output: Most similar house

Initialize Dist2NN = ∞, = $\emptyset$ (closest house)
For i = 1, 2, ... N (query house)
  Compute δ = distance(,)
    if δ < Dist2NN
      set =
      set Dist2NN = δ
Return most similar house (closest house to query house)

Nearest Neighbors Kernel $K=1$

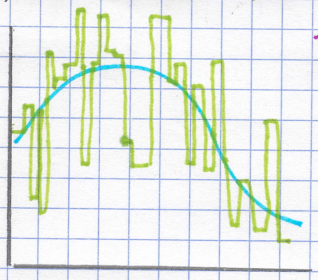


Fit looks good for data dense in x and low noise

Sensitive to regions with little data

Not great at interpolating over large regions

Also sensitive to noise in data

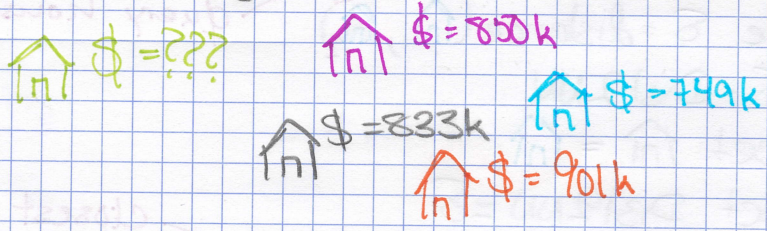


Fits can look quite wild... Overfitting?

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: $(x_1, y_1), (x_2, y_2) \ldots (x_N, y_N)$

Query point: x_q

1. Find k closest 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, distance$(x_i, x_q) \geq$ $\geq$ 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

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

(sort first k houses by distance to query house)

$\square_1 = \text{sort}(\square_1, \ldots, \square_k)$ ← list of sorted houses

for $i = k+1, \ldots, N$

Compute: $\delta = \text{distance}(\square_i, \square_q)$ (query house)

if $\delta <$ Dist2kNN[k]

find j such that $\delta >$ Dist2kNN[j-1] but $\delta <$ Dist2kNN[j] remove furthest house and shift queue:

$\square[j+1:k] = \square[j:k-1]$

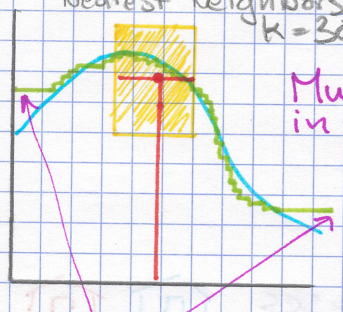
Dist2kNN[j+1:k] = Dist2kNN[j:k-1]

set Dist2kNN[j] = δ and $\square[j] = \square_i$

Return k most similar houses $\square$

closest houses to query house

k-NN in practice



Much more reasonable fit in the presence of noise

Discontinuities! Neighbor either in or out

Boundary & sparse region issues

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 from 2640 sq.ft to 2641 sq.ft.
- Don't really believe this type of fit.

Weighted k-nearest neighbors

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 distance(x_{NNj}, x_q) large

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

Simple method:

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

Kernel weights for d=1

Define: $c_{q,NN_j} = \text{Kernel}_\lambda(|x_{NN_j} - x_q|)$ {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 $d \geq 1$

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

Kernel regression

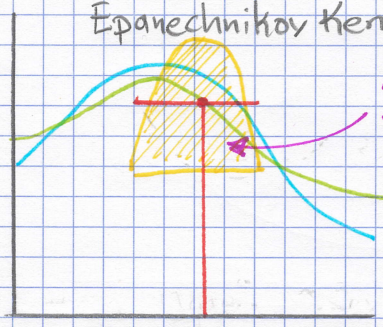
Instead of just weighting NN, weight all points

Predict

weight on each datapoint

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

Nadaraya-Watson Kernel weighted average



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

$$y_p = \frac{1}{N}\sum_{i=1}^{N} y_i = \frac{\sum_{i=1}^{N} c\,y_i}{\sum_{i=1}^{N} c}$$

← equal weight on each datapoint

Kernel regression leads to locally constant fit
- slowly add in some points and let others gradually die off

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

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

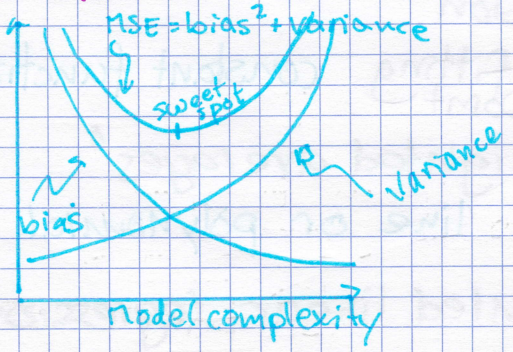
k-NN and kernel regression are examples of **nonparametric** regression

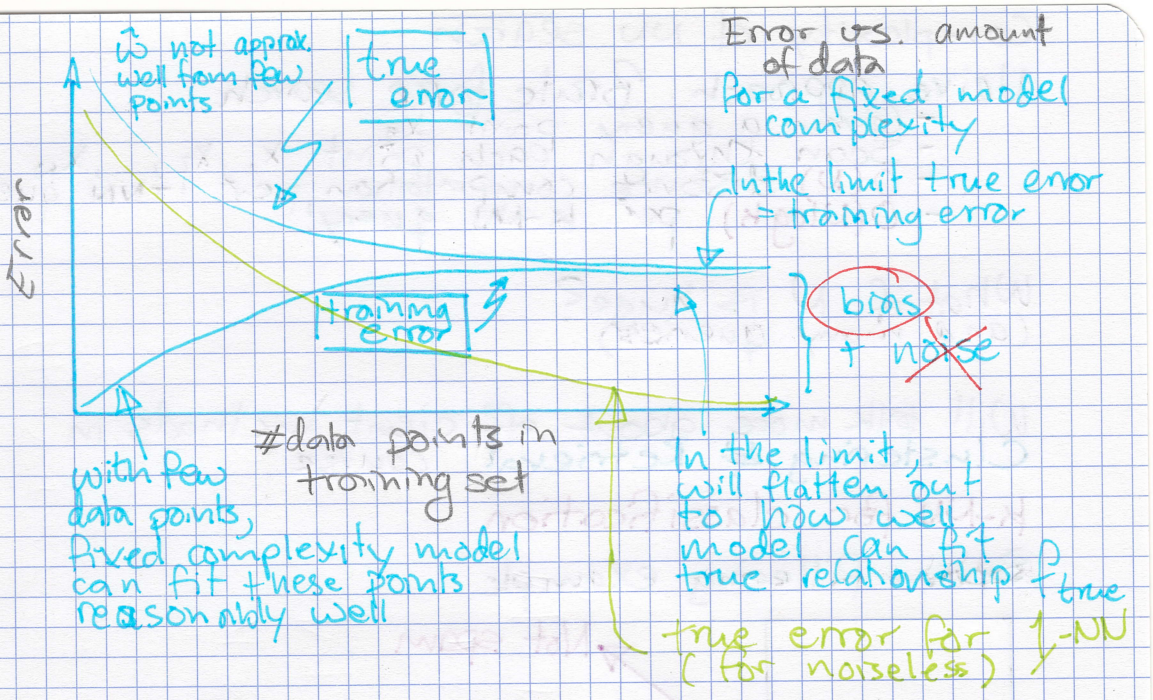
General goals of nonparametrics:
- Flexibility
- Make few assumptions about $f(x)$
- **Complexity can grow with the number of observations N**

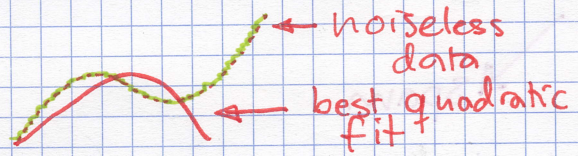
Lots of other choices:
- Splines, trees, locally weighted structured regression models.

Limiting behavior of NN:
Noiseless setting $\varepsilon_i = 0$

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







Limiting behavior of NN:
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

Issues with high-dimensions, data scarcity, and computational complexity

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

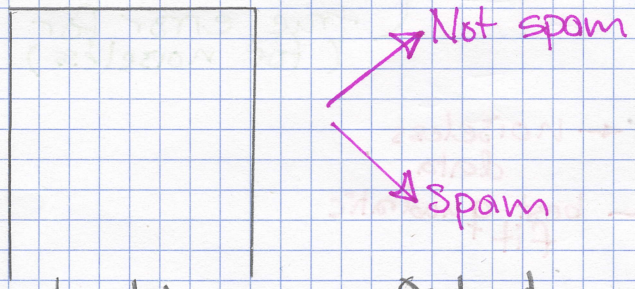
Naive approach. Brute force search
- Given a query point x_q
- Scan through each point $x_1, x_2, \ldots, x_N$
- $O(N)$ distance computation per 1-NN query!
- $O(N \log k)$ per k-NN query!

What if N is huge?
(and many queries)

Will talk more about efficient methods in Clustering & Retrieval course

k-NN for classification

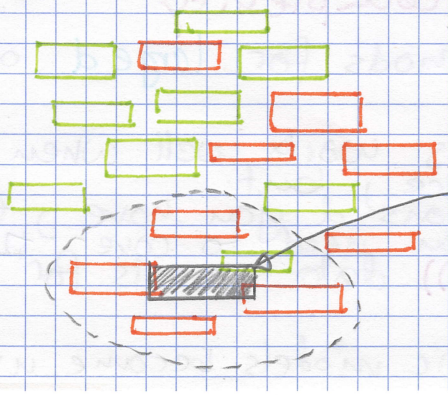
Spam filtering example



Text of email
sender, IP, ...

Using k-NN for classification

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



not spam vs. spam
decide via majority vote of k-NN