Instance-based Learning

Nearest Neighbors/Non-Parametric Methods

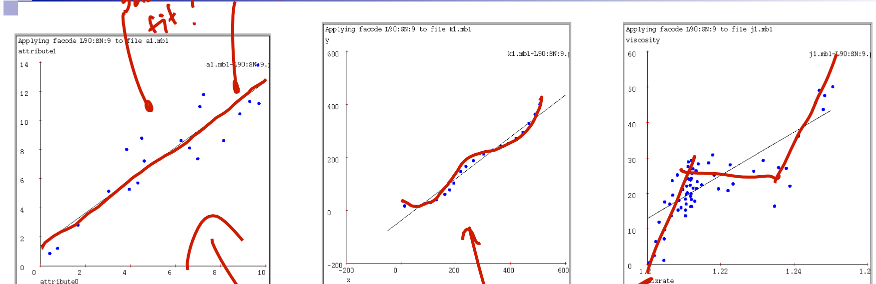
Machine Learning – CSE446
Carlos Guestrin
University of Washington

April 24, 2013

Why not just use Linear Regression?

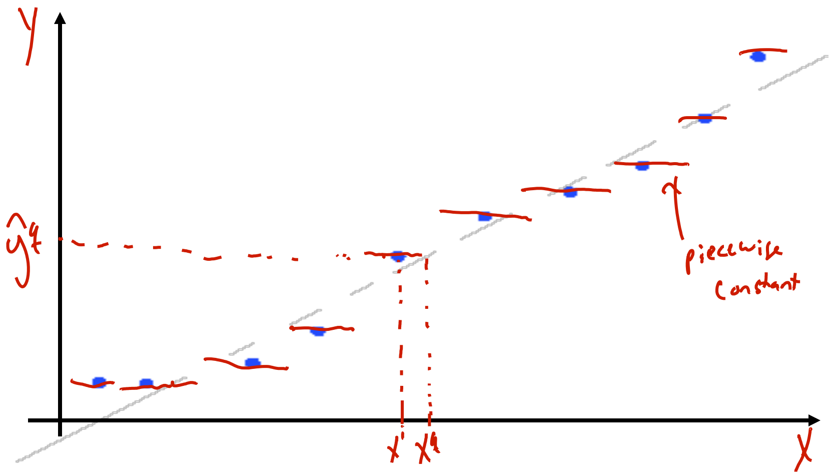
bad fit?

$w_0 + w_1 x$

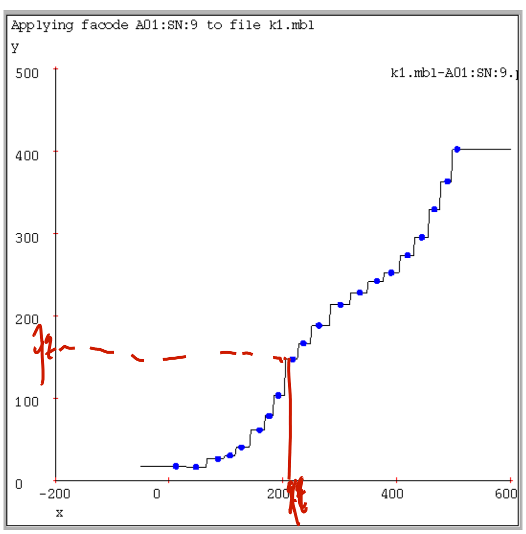


fit?

Using data to predict new data



Nearest neighbor



Univariate 1-Nearest Neighbor

Given datapoints (x^1,y^1) (x^2,y^2)..(x^N,y^N),where we assume $y^i=f(x^i)$ for some unknown function f.

Given query point x^q, your job is to predict $\hat{y} \approx f\left(x^q\right)$

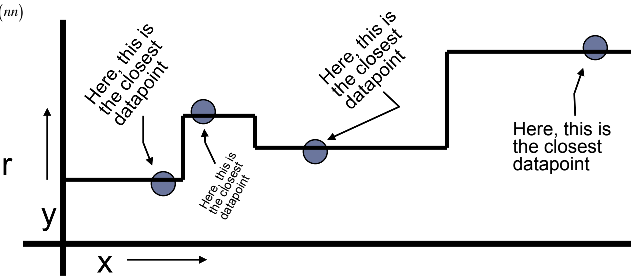
Nearest Neighbor:

1. Find the closest x_i in our set of datapoints

$$j(nn) = \underset{j}{\operatorname{argmin}} \left| x^j - x^q \right|$$

2. Predict $\hat{y} = y^{i(nn)}$

Here's a dataset with one input, one output and four datapoints.

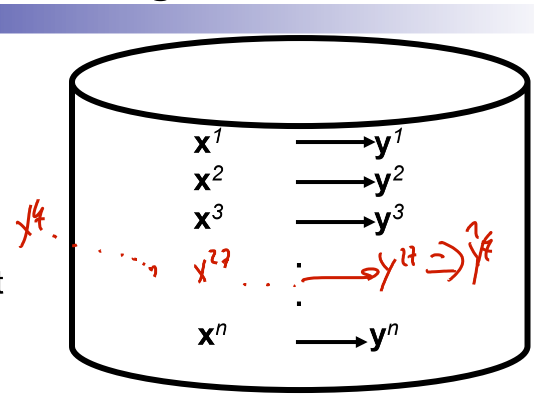


1-Nearest Neighbor is an example of….

Instance-based learning

A function approximator that has been around since about 1910.

To make a prediction, search database for similar datapoints, and fit with the local points.



Four things make a memory based learner:

- A distance metric
- How many nearby neighbors to look at?
- A weighting function (optional)
- How to fit with the local points?

1-Nearest Neighbor

Four things make a memory based learner:

1. *A distance metric*
   Euclidian (and many more) $\|\cdot\|_2$ ←
2. *How many nearby neighbors to look at?*
   One
3. *A weighting function (optional)*
   Unused
4. *How to fit with the local points?*
   Just predict the same output as the nearest neighbor.

$$i = \arg\min_j \|x^j - x^q\|$$

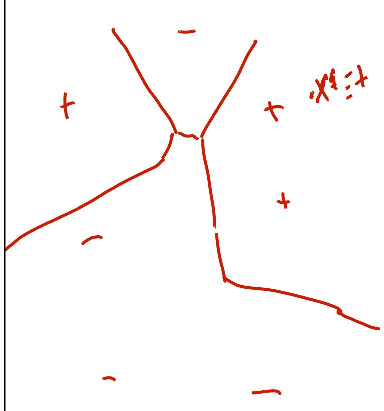
predict $\hat{y}^q = y^i$

Euclidean

Multivariate 1-NN examples

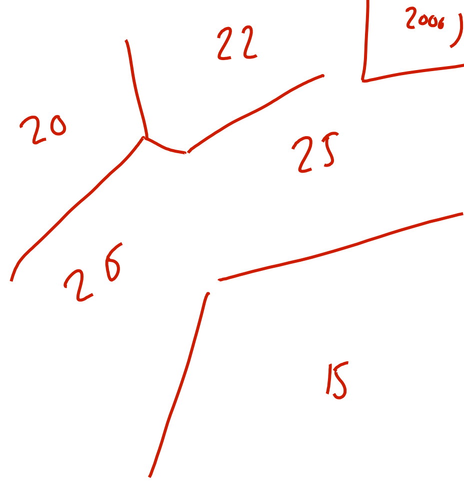
fast nearest neighbor search
hot topic (hottest in 2006)

Classification



Voronoi diagram

Regression



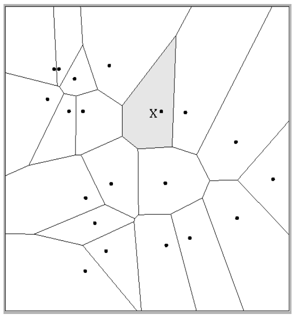
Multivariate distance metrics

$\|x^i - x^j\|_2^2 = (x_1^i - x_1^j)^2 + (x_2^i - x_2^j)^2$

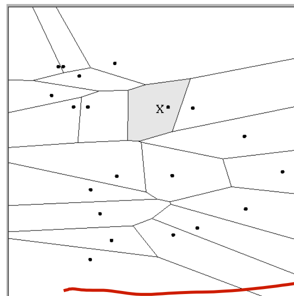
Suppose the input vectors x^1, x^2, …x^N are two dimensional:

$\mathbf{x}^1 = (x^1_1 , x^1_2)$, $\mathbf{x}^2 = (x^2_1 , x^2_2)$, …$\mathbf{x}^N = (x^N_1 , x^N_2)$.

One can draw the nearest-neighbor regions in input space.



x_2



x_1

I care more about changes in x_2

$Dist(\mathbf{x}^i,\mathbf{x}^j) = (x^i_1 - x^j_1)^2 + (x^i_2 - x^j_2)^2$ $Dist(\mathbf{x}^i,\mathbf{x}^j) = (x^i_1 - x^j_1)^2 + (3x^i_2 - 3x^j_2)^2$

The relative scalings in the distance metric affect region shapes

scalling x_2 in my distance

Learning the parameters Σ is called "distance metric learning" e.g., gradient on training data (or cross-validation)

Euclidean distance metric

scalling

Or equivalently,
$$D(\mathrm{x},\mathrm{x}') = \sqrt{\sum_i \sigma_i^2 (x_i - x'_i)^2}$$

where
$$D(\mathrm{x},\mathrm{x}') = \sqrt{(\mathrm{x}-\mathrm{x}')^T \sum (\mathrm{x}-\mathrm{x}')}$$

arbitrary matrix scalling

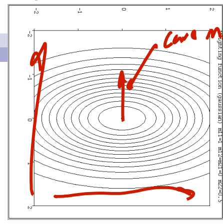
$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \sigma_N^2 \end{bmatrix}$$

diagonal

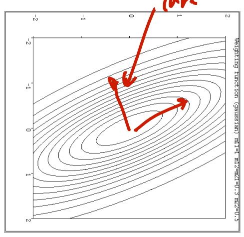
Other Metrics… Σ not diagonal

- Mahalanobis, Rank-based, Correlation-based,…

Notable distance metrics (and their level sets)

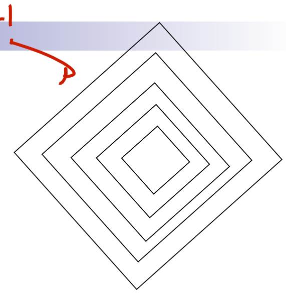


Scaled Euclidian (L_2)



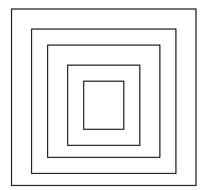
Mahalanobis (here, Σ on the previous slide is not necessarily diagonal, but is symmetric

L1



L_1 norm (absolute)

many others



L1 *(max)* norm

Consistency of 1-NN

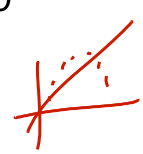
- Consider an estimator f_n trained on n examples
  - e.g., 1-NN, neural nets, regression,...
- Estimator is *consistent* if true error goes to zero as amount of data increases
  - e.g., for no noise data, consistent if:

mean squared error

$$\lim_{n \to \infty} MSE(f_n) = 0$$

standard linear

- Regression is not consistent!
  - Representation bias

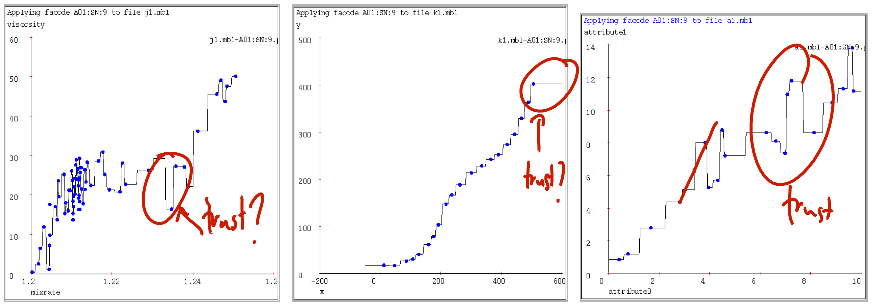


- **1-NN is consistent** (under some mild fineprint)

danger: overfitting

What about variance???

1-NN overfits?



need more smoothness

⇒ e.g. avg over k neighbors

k-Nearest Neighbor

Four things make a memory based learner:

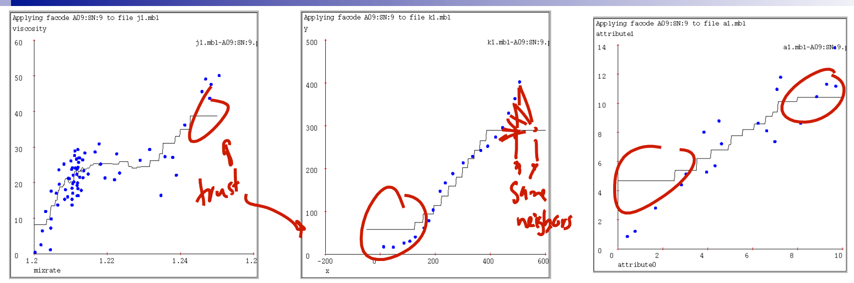
1. *A distance metric*
   Euclidian (and many more)
2. *How many nearby neighbors to look at?*
   k
1. *A weighting function (optional)*
   Unused
2. *How to fit with the local points?*
   Just predict the average output among the k nearest neighbors.

$NN(x^q) \leftarrow k$ nearest neighbors

$\hat{y}^q = \frac{1}{k} \sum_{j \in NN(x^q)} y_j \leftarrow$ regression

classification
majority vote
(most common class)

k-Nearest Neighbor (here k=9)



smoother but still hard to trust

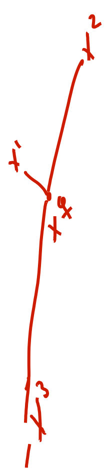
K-nearest neighbor for function fitting smoothes away noise, but there are clear deficiencies.

What can we do about all the discontinuities that k-NN gives us?

Weighted k-NNs

- Neighbors are not all the same

k-NN



$$\hat{y}^q = \frac{y^1 + y^2 + y^3}{3}$$

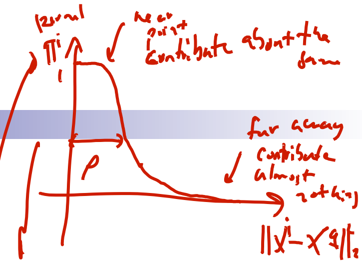
weighted NN

$$\hat{y}^q = \frac{\sum_{i \in NN(x^q)} \pi^i y^i}{\sum_i \pi^i}$$

π^i intuitively decreases with distance

e.g., $\pi^i = \dfrac{1}{\|x^i - x^q\|}$

Kernel regression



Four things make a memory based learner:

1. *A distance metric*
   Euclidian (and many more)
2. *How many nearby neighbors to look at?*
   All of them
3. *A weighting function (optional)*
   π^i = exp(-D(x^i, query)^2 / ρ^2)

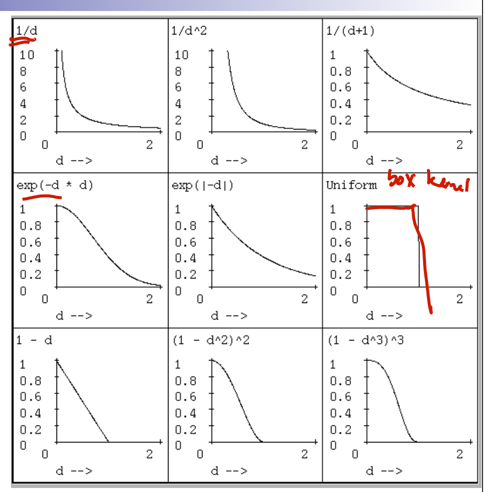
   $e^{-\frac{\|x^i - x^q\|_2^2}{\rho^2}}$

   Nearby points to the query are weighted strongly, far points weakly. The ***ρ*** parameter is the **Kernel Width**. Very important.
4. *How to fit with the local points?*
   Predict the weighted average of the outputs:
   predict = $\Sigma\pi^i y^i$ / $\Sigma\pi^i$

Weighting functions

π^i = exp(-D(x^i, query)^2 / ρ^2)

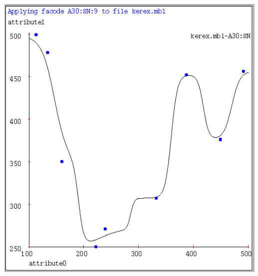
Square exponential or Gaussian kernel

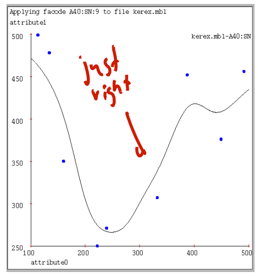


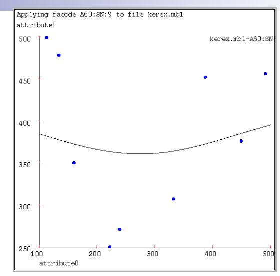
Typically optimize ***ρ*** using gradient descent or cross validation

(Our examples use Gaussian)

Kernel regression predictions







***ρ*=10** only care about nearby points ⇒ more variance

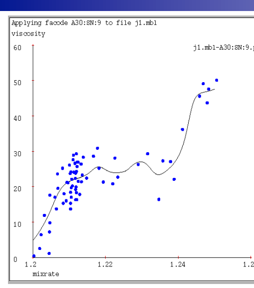
***ρ*=20**

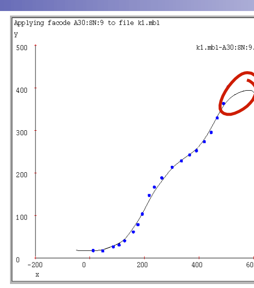
***ρ*=80** care about points that are far away ⇒ more bias

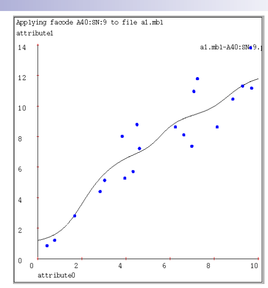
Increasing the kernel width ρ means further away points get an opportunity to influence you.

As $\rho \to \infty$, the prediction tends to the global average.

Kernel regression on our test cases







ρ=1/32 of x-axis width.

ρ=1/32 of x-axis width.

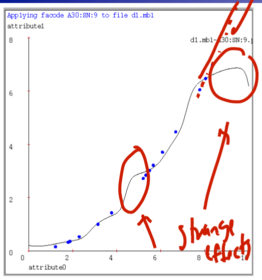
ρ=1/16 axis width.

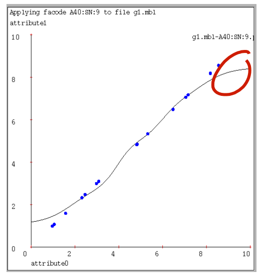
better fits

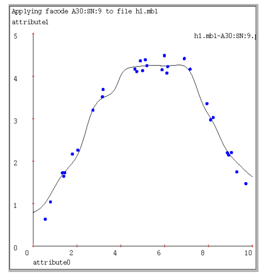
Choosing a good ρ is important. Not just for Kernel Regression, but for all the locally weighted learners we're about to see.

Kernel regression can look bad

locally fit regression curves







ρ = Best. ρ = Best. ρ = Best.

Time to try something more powerful…

Locally weighted regression

Kernel regression:

Take a very very conservative function approximator called AVERAGING. Locally weight it.

Locally weighted regression:

Take a conservative function approximator called LINEAR REGRESSION. Locally weight it.

Locally weighted regression

- **Four things make a memory based learner:**
- *A distance metric*
  - **Any**
- *How many nearby neighbors to look at?*
  - **All of them**
- *A weighting function (optional)*
  - **Kernels**
    - $\pi^i = \exp(-D(x^i, \text{query})^2 / \rho^2)$
- *How to fit with the local points?*
  - **General weighted regression:**

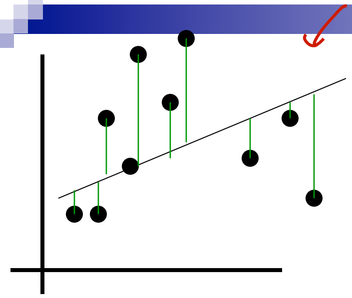
$$\hat{w}^q = \underset{w}{\operatorname{argmin}} \sum_{k=1}^{N} \pi_q^k \Big(\underbrace{\overbrace{y^k}^{\text{standard least square}} - \underbrace{w^T x^k}_{\text{prediction}}}_{\text{truth}}\Big)^2$$

fit regression over weighted data

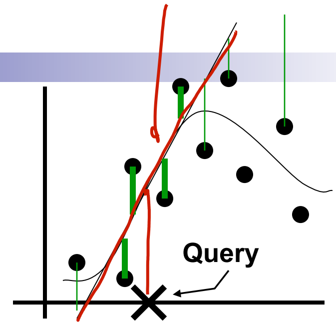
$\hat{y}^q = \hat{w}^q \cdot x^q$ just like in regression

How LWR works

care about all points the same



care more about errors in nearby points



Linear regression
- Same parameters for all queries

$$\hat{w} = \left(X^T X\right)^{-1} X^T Y$$

Locally weighted regression
- Solve weighted linear regression for each query

simple matrix inversion again

$$w^q = \left((\Pi X)^T \Pi X\right)^{-1} (\Pi X)^T \Pi Y$$

$$\Pi = \begin{pmatrix} \pi_1 & 0 & 0 & 0 \\ 0 & \pi_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \pi_n \end{pmatrix}$$

Another view of LWR

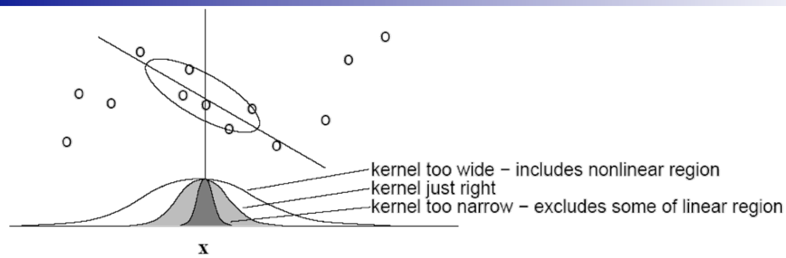
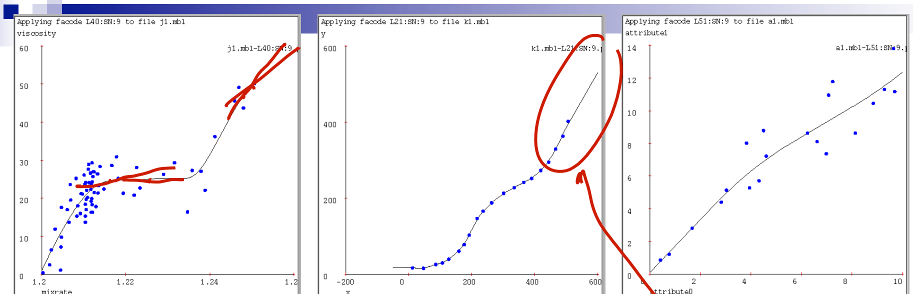


Image from Cohn, D.A., Ghahramani, Z., and Jordan, M.I. (1996) "Active Learning with Statistical Models", JAIR Volume 4, pages 129-145.

LWR on our test cases

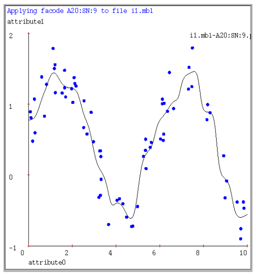


ρ = 1/16 of x-axis width. ρ = 1/32 of x-axis width. ρ = 1/8 of x-axis width.

we like it nice fits

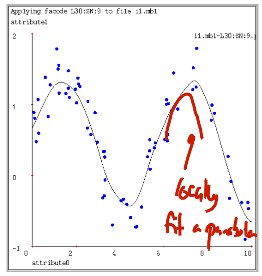
Locally weighted polynomial regression

different query points can use different basis functions



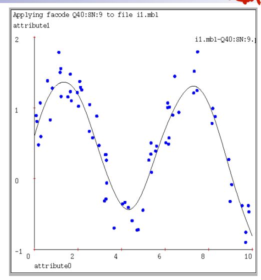
Kernel Regression
Kernel width ρ at optimal level.

ρ = 1/100 x-axis



LW Linear Regression
Kernel width ρ at optimal level.

ρ = 1/40 x-axis



LW Quadratic Regression
Kernel width ρ at optimal level.

ρ = 1/15 x-axis

Local quadratic regression is easy: just add quadratic terms to the X matrix. As the regression degree increases, the kernel width can increase without introducing bias.

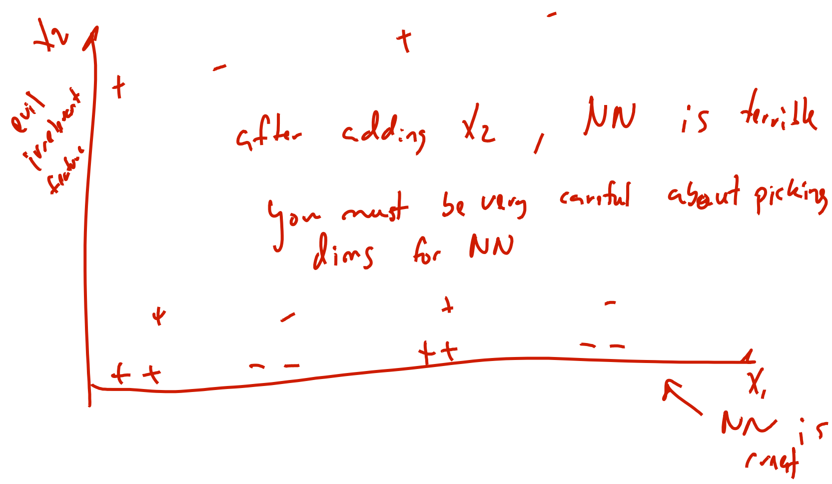
Curse of dimensionality for instance-based learning

- Must store and retreve all data!
  - Most real work done during testing
  - For every test sample, must search through all dataset – very slow!
  - There are (sometimes) fast methods for dealing with large datasets
- Instance-based learning often poor with noisy or irrelevant features

thm: amount data you need is exp in dim

often need to be smart about picking dimensions

Curse of the irrelevant feature



What you need to know about instance-based learning

- k-NN
  - Simplest learning algorithm
  - With sufficient data, very hard to beat "strawman" approach
  - Picking k?
- Kernel regression
  - Set k to n (number of data points) and optimize weights by gradient descent
  - Smoother than k-NN
- Locally weighted regression
  - Generalizes kernel regression, not just local average
- Curse of dimensionality
  - Must remember (very large) dataset for prediction
  - Irrelevant features often killers for instance-based approaches

Acknowledgment

- This lecture contains some material from Andrew Moore's excellent collection of ML tutorials:
  - http://www.cs.cmu.edu/~awm/tutorials