Why not just use Linear Regression?

\[ y = w_0 + w_1 x \]
Using data to predict new data

Nearest neighbor
Univariate 1-Nearest Neighbor

Given datapoints \((x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\), where we assume \(y_i = f(x_i)\) for some unknown function \(f\).

Given query point \(x_q\), your job is to predict \(\hat{y} \approx f(x_q)\).

Nearest Neighbor:
1. Find the closest \(x_i\) in our set of datapoints
   \[ i(nn) = \arg\min_i |x_i - x_q| \]
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 \( ||x_j - x_q||_2 \)
   Euclidian (and many more)

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.

\[ i = \operatorname{argmin} \; ||x_j - x_q||_2 \]

\[ y_i = \text{predict } y \]

Multivariate 1-NN examples

Classification

Regression
Multivariate distance metrics

Suppose the input vectors \( x_1, x_2, \ldots, x_n \) are two dimensional:
\[
x_1 = (x_{11}, x_{12}), \quad x_2 = (x_{21}, x_{22}), \ldots, x_N = (x_{N1}, x_{N2}).
\]
One can draw the nearest-neighbor regions in input space.

\[
\text{Dist}(x_i, x_j) = (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2
\]

The relative scalings in the distance metric affect region shapes.

Euclidean distance metric

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

where
\[
D(x, x') = \sqrt{(x - x')^T \Sigma^{-1} (x - x')}
\]

Other Metrics…
- Mahalanobis, Rank-based, Correlation-based, …
Notable distance metrics 
(and their level sets)

- **L₁ norm (absolute)**
- **L₁ (max) norm**
- **Scaled Euclidian (L₂)**
- **Mahalanobis** (here, Σ on the previous slide is not necessarily diagonal, but is symmetric)

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:
    \[
    \lim_{n \to \infty} MSE(f_n) = 0
    \]
- Regression is not consistent!
  - Representation bias
- **1-NN is consistent** (under some mild fineprint)

What about variance???
1-NN overfits?

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.

\[ \hat{y}_i \sim \frac{1}{k} \sum_{i \in \mathcal{N}(x_i)} y_i \]
k-Nearest Neighbor (here $k=9$)

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

Intuitively, $w_i$ decreases with distance.

E.g., $w_i = \frac{1}{\|x_i - x\|^2}$
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)
   \( w_i = \exp\left(-\frac{D(x_i, \text{query})^2}{K_w^2}\right) \)
   Nearby points to the query are weighted strongly, far points
   weakly. The \( K_w \) parameter is the Kernel Width. Very
   important.

4. How to fit with the local points?
   Predict the weighted average of the outputs:
   \[ \text{predict} = \frac{\sum w_i y_i}{\sum w_i} \]

Weighting functions

\[ w_i = \exp\left(-\frac{D(x_i, \text{query})^2}{K_w^2}\right) \]

Typically optimize \( K_w \)
using gradient descent

(Our examples use Gaussian)
Kernel regression predictions

Increasing the kernel width $K_w$ means further away points get an opportunity to influence you. As $K_w \to \infty$, the prediction tends to the global average.

Kernel regression on our test cases

Choosing a good $K_w$ is important. Not just for Kernel Regression, but for all the locally weighted learners we’re about to see.
Kernel regression can look bad

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
  \[ w_i = \exp(-D(x_i, \text{query})^2 / K w_i^2) \]

How to fit with the local points?

General weighted regression:

\[
\hat{\beta} = \arg\min_{\beta} \sum_{k=1}^{N} w_k (y_k - \beta^T x_k)^2
\]

For every test point \( y_k \):
- Compute \( w_k \)
  - Use Epanechnikov kernel
  - Compute \( \hat{\beta} \)
- Solve weighted linear regression
  - \( \hat{\beta} \) for every query

Linear regression

- Same parameters for all queries

\[
\hat{\beta} = \left( X^T X \right)^{-1} X^T Y
\]

Locally weighted regression

- Solve weighted linear regression for each query

\[
\beta = \left( W X \right)^T \frac{1}{W} W X \frac{1}{W} Y
\]

Weight matrix \( W \):

\[
W = \begin{pmatrix}
w_1 & 0 & 0 & 0 \\
0 & w_2 & 0 & 0 \\
0 & 0 & \ddots & 0 \\
0 & 0 & 0 & w_n
\end{pmatrix}
\]
Another view of LWR

KW = 1/16 of x-axis width.

KW = 1/32 of x-axis width.

KW = 1/8 of x-axis width.

Locally weighted polynomial regression

Kernel Regression
Kernel width $K_W$ at optimal level.

$K_W = 1/100$ x-axis

LW Linear Regression
Kernel width $K_W$ at optimal level.

$K_W = 1/40$ x-axis

LW Quadratic Regression
Kernel width $K_W$ at optimal level.

$K_W = 1/15$ x-axis

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

Curse of dimensionality for instance-based learning

- Must store and retrieve all data!
  - Most real work done during testing
  - For every test sample, must search through all dataset – very slow!
  - We’ll see fast methods for dealing with large datasets

- Instance-based learning often poor with noisy or irrelevant features
  
  every point is equally far in high
  
  dims $\Rightarrow$ typically need exponentially
  
  many data in dim
  
  (or at least very
  
  cleverly selected features)
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](http://www.cs.cmu.edu/~awm/tutorials)