Why not just use Linear Regression?
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 \(\hat{y} = \hat{f}(x^q)\)

1. Find the closest \(x_i\) in our set of datapoints

\[
j(nn) = \arg\min_j |x^j - x^q|
\]

2. Predict \(\hat{y} = y^{\hat{(m)}}\)

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)
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.

\[ i = \arg\min_j \|x_i - x_j\| \]

\[ \text{project } \hat{y}_i = y_i \]

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.

The relative scalings in the distance metric affect region shapes.

Euclidean distance metric

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

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

where
\[
\Sigma = \begin{bmatrix}
\sigma_1^2 & 0 & \cdots & 0 \\
0 & \sigma_2^2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \sigma_N^2
\end{bmatrix}
\]

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

- **L$_1$** norm (absolute)
- **L$_1$** (max) norm
- Scaled Euclidean (L$_2$)
- Mahalanobis (here, $\Sigma$ 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?

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.

\[
\hat{y}_k(x) \in \text{k nearest neighbors}
\]

\[
\hat{y}_k = \frac{1}{k} \sum_{j \in \text{NN}(x)} y_j
\]

regression

classification

majority vote

( or most common class)
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
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)
   \[ \pi^i = \exp(-D(x^i, \text{query})^2 / \rho^2) \]
   Nearby points to the query are weighted strongly, far points weakly. The \( \rho \) parameter is the Kernel Width. Very important.
4. How to fit with the local points?
   Predict the weighted average of the outputs:
   \[ \text{predict} = \frac{\Sigma \pi^i y^i}{\Sigma \pi^i} \]

Weighting functions

\[ \pi^i = \exp(-D(x^i, \text{query})^2 / \rho^2) \]

Typically optimize \( \rho \) using gradient descent (Our examples use Gaussian)
Kernel regression predictions

Increasing the kernel width $\rho$ 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

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

\( \rho = \text{Best.} \)

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
      \[ \pi_i = \exp\left(-\frac{D(x_i, \text{query})^2}{\rho^2}\right) \]
  - How to fit with the local points?

General weighted regression:

\[
\hat{w}^q = \arg\min_w \sum_{i=1}^{N} \pi_i \left( y^i - \hat{w}^T x^i \right)^2
\]

How LWR works

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

\[ \Pi = \begin{pmatrix}
\pi_1 & 0 & 0 & 0 \\
0 & \pi_2 & 0 & 0 \\
0 & 0 & \ddots & 0 \\
0 & 0 & 0 & \pi_n
\end{pmatrix} \]
Another view of LWR

- Kernel too wide - includes nonlinear region
- Kernel just right
- Kernel too narrow - excludes some of linear region

LWR on our test cases

\[ \rho = \frac{1}{16} \text{ of x-axis width.} \quad \rho = \frac{1}{32} \text{ of x-axis width.} \quad \rho = \frac{1}{8} \text{ of x-axis width.} \]
Locally weighted polynomial regression

![Graphs showing kernel regression, linear regression, and quadratic regression with different kernel widths](image)

- **Kernel Regression**
  - Kernel width $\rho$ at optimal level.
  - $\rho = 1/100$ x-axis

- **LW Linear Regression**
  - Kernel width $\rho$ at optimal level.
  - $\rho = 1/40$ x-axis

- **LW Quadratic Regression**
  - Kernel width $\rho$ at optimal level.
  - $\rho = 1/15$ 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 retrieve 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

  "Thou art able to be smart about picking dimensionality"
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)