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.
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?

©2005-2007 Carlos Guestrin
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? $k$
3. A weighting function (optional)
   - Unused
4. How to fit with the local points?
   - Just predict the average output among the $k$ nearest neighbors.
   \[
   \hat{y}_{i} = \frac{1}{k} \sum_{x_i \in N(x_q)} y_i
   \]

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

\[ y_q = \frac{1}{d_i} y_i \]

\[ \sum_{x_i \in N(q)} \frac{1}{d_i} y_i \]

(normalize to get a convex combination)

---

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

   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:
     \[ \hat{y}_q = \frac{\sum w_i y_i}{\sum w_i} \]
Weighting functions

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

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.

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
    - $w_i = \exp(-D(x_i, \text{query})^2 / Kw^2)$
- How to fit with the local points?
  - General weighted regression:
    $$\hat{\beta} = \arg \min_{\beta} \sum_{k=1}^{N} w_k \left(y_k - \beta^T x_k\right)^2$$
How LWR works

Linear regression
- Same parameters for all queries

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

Locally weighted regression
- Solve weighted linear regression for each query

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

Another view of LWR

LWR on our test cases

Kernel Regression
Kernel width $K_W$ at optimal level.

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

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

Local quadratic regression is easy: just add quadratic terms to the $WX^T WX$ 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

Curse of the irrelevant feature

`NN can perform poorly with irrelevant features. Can address by feature selection or learning shifted.`
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)
Linear classifiers – Which line is better?

Data:
\[ \langle x_1^{(1)}, \ldots, x_1^{(m)}, y_1 \rangle \]
\[ \langle x_2^{(1)}, \ldots, x_2^{(m)}, y_2 \rangle \]
\[ \langle x_n^{(1)}, \ldots, x_n^{(m)}, y_n \rangle \]

Example i:
\[ \langle x_1^{(1)}, \ldots, x_1^{(m)} \rangle \quad \text{— m features} \]
\[ y_i \in \{-1, +1\} \quad \text{— class} \]

\[ w \cdot x = \sum_j w^{(j)} x^{(j)} \]
Pick the one with the largest margin!

\[
\mathbf{w} \cdot \mathbf{x} = \sum_j w^{(j)} x^{(j)}
\]

confidence = \((\mathbf{w} \cdot \mathbf{x}_j + b) y_j\)

Maximize the margin

\[
\gamma = \max \min \left( \mathbf{w} \cdot \mathbf{x}_i + b \right) y_i
\]

\[
\gamma = \max \min \left( \mathbf{w} \cdot \mathbf{x}_i + b \right) y_i \geq \gamma
\]

\[
\max \gamma, \mathbf{w}, b
\]

\[
(w \cdot x_j + b) y_j \geq \gamma, \forall j \in \text{Dataset}
\]
But there are many planes...

\[ w \cdot x + b = 0 \]

What if:
- New plane
  - \( \tilde{w} = 2w \)
  - \( \tilde{b} = 2b \)
- Now \( \tilde{y} = 2y \)
- I can be as confident as I want by increasing \( w \)

Review: Normal to a plane

\[ x_j = \bar{x}_j + \lambda_j \frac{w}{||w||} \]
Normalized margin – Canonical hyperplanes

\[ x_j = \bar{x}_j + \lambda \frac{w}{\|w\|} \]

\[ x^+ = x^- + \lambda \frac{w}{\|w\|} \]

\[ w \cdot x^+ + b = 1 \]

\[ w \cdot (x^- + \lambda \frac{w}{\|w\|}) + b = 1 \]

\[ \gamma = \frac{1}{\sqrt{w \cdot w}} \]

\[ \lambda = \frac{2}{\|w\|} \]
Margin maximization using canonical hyperplanes

\[ \gamma = \frac{1}{\sqrt{w.w}} \]

Maximize \( \gamma, w, b \) \( \gamma, y \leq 1 \)
\((w.x_j + b) y_j \geq \gamma, \forall j \in \text{Dataset})

\[ \max \frac{1}{\sqrt{w.w}} \]
\( (w.x_j + b) y_j \geq \gamma, \gamma \leq 1 \)
\[ \min w.w \]
\( (w.x_j + b) y_j \geq \gamma, \gamma \leq 1 \)

Support vector machines (SVMs)

Minimize \( w, b \)
\( w.w \)
\((w.x_j + b) y_j \geq 1, \forall j \)

- Solve efficiently by quadratic programming (QP)
- Well-studied solution algorithms
- Hyperplane defined by support vectors

Points where \((w.x_j + b) y_j = 1\) (support vectors)