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
1. A weighting function (optional)
   Unused
2. How to fit with the local points?
   Just predict the average output among the k nearest neighbors.

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)
   \[ w_i = \exp(-D(x_i, \text{query})^2 / K^2) \]
   Nearby points to the query are weighted strongly, far points weakly. The \( K \) parameter is the Kernel Width. Very important.
4. How to fit with the local points?
   Predict the weighted average of the outputs:
   \[ \text{predict} = \frac{\Sigma w_i y_i}{\Sigma w_i} \]
Weighting functions

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

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

Linear regression
- Same parameters for all queries
\[ \hat{a} = (X^TX)^{-1}X^TY \]

Locally weighted regression
- Solve weighted linear regression for each query
\[ \hat{y} = (WX)^TWX \]
\[ W = \begin{pmatrix} w_1 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & w_n \end{pmatrix} \]

Another view of LWR

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

LWR on our test cases

![Graphs showing LWR results with different kernel widths.]

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

![Graphs showing LW regression results with different kernel widths.]

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.

KW = 1/100 x-axis

KW = 1/40 x-axis

KW = 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

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)
Announcements

- Recitation this week: Neural networks

- Project proposals due next Wednesday
  - Exciting data:
    - Swivel.com - user generated graphs
    - Recognizing Captchas
    - Election contributions
    - Activity recognition
    - \ldots

Support Vector Machines

Machine Learning – 10701/15781
Carlos Guestrin
Carnegie Mellon University

October 15th, 2007
Linear classifiers – Which line is better?

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

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

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

Pick the one with the largest margin!

“confidence” = \[(w \cdot x_j + b) y_j \]

\[
w \cdot x = \sum_j w^{(j)} x^{(j)}
\]
Maximize the margin

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

maximize \( \gamma, w, b \)

\[ (w \cdot x_j + b) y_j \geq \gamma, \ \forall j \in \text{Dataset} \]

But there are a many planes…
Review: Normal to a plane

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

Normalized margin – Canonical hyperplanes

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

margin \(2\gamma\)
Normalized margin – Canonical hyperplanes

\[ x^+ = x^- + \lambda w \]

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

\[ \lambda = \frac{2}{||w||} \]
\[ \gamma = \frac{1}{\sqrt{w \cdot w}} \]

Margin maximization using canonical hyperplanes

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

maximize_{\gamma, w, b} \gamma \cdot (w \cdot x_j + b) y_j \geq \gamma, \ \forall j \in \text{Dataset} \]

minimize_{w, b} w \cdot w \cdot w \cdot w \cdot \sum \left( (w \cdot x_j + b) y_j \geq 1, \ \forall j \in \text{Dataset} \right)
Support vector machines (SVMs)

$$\begin{align*}
\text{minimize}_{w,b} & \quad w \cdot w \\
& (w \cdot x_j + b) y_j \geq 1, \quad \forall j
\end{align*}$$

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

What if the data is not linearly separable?

Use features of features of features of features….
What if the data is still not linearly separable?

\[
\min_{w,b} \quad w \cdot w \\
\left( w \cdot x_j + b \right) y_j \geq 1, \quad \forall j
\]

- Minimize \( w \cdot w \) and number of training mistakes
  - Tradeoff two criteria?

- Tradeoff \#(mistakes) and \( w \cdot w \)
  - 0/1 loss
  - Slack penalty \( C \)
  - Not QP anymore
  - Also doesn’t distinguish near misses and really bad mistakes

Slack variables – Hinge loss

\[
\min_{w,b} \quad w \cdot w \\
\left( w \cdot x_j + b \right) y_j \geq 1, \quad \forall j
\]

- If margin \( \geq 1 \), don’t care
- If margin \( < 1 \), pay linear penalty
Side note: What’s the difference between SVMs and logistic regression?

SVM:

\[
\begin{align*}
\text{minimize}_{w, b} & \quad w \cdot w + C \sum_j \xi_j \\
& \quad (w \cdot x_j + b) y_j \geq 1 - \xi_j, \quad \forall j \\
& \quad \xi_j \geq 0, \quad \forall j
\end{align*}
\]

Logistic regression:

\[
P(Y = 1 \mid x, w) = \frac{1}{1 + e^{-(w \cdot x + b)}}
\]

Log loss:

\[
-\ln P(Y = 1 \mid x, w) = \ln (1 + e^{-(w \cdot x + b)})
\]

What about multiple classes?
One against All

Learn 3 classifiers:

Learn 1 classifier: Multiclass SVM

Simultaneously learn 3 sets of weights

\[ w(y_j) x_j + b(y_j) \geq w(y') x_j + b(y') + 1, \ \forall y' \neq y_j, \ \forall j \]
Learn 1 classifier: Multiclass SVM

\[
\begin{align*}
\text{minimize}_{w,b} & \quad \sum_y w(y) \cdot w(y) + C \sum_j \xi_j \\
& \quad w(y_j) \cdot x_j + b(y_j) \geq w(y') \cdot x_j + b(y') + 1 - \xi_j, \quad \forall y' \neq y_j, \forall j \\
& \quad \xi_j \geq 0, \forall j
\end{align*}
\]

What you need to know

- Maximizing margin
- Derivation of SVM formulation
- Slack variables and hinge loss
- Relationship between SVMs and logistic regression
  - 0/1 loss
  - Hinge loss
  - Log loss
- Tackling multiple class
  - One against All
  - Multiclass SVMs