Machine Learning

Topic 3: Instance(memory)-based learning

(with ideas and a few images from Andrew Moore
Check out his site at http://www.autonlab.org/)
Classification vs. Regression

• Classification:
  Learning a function to map from a n-tuple to a \textit{discrete} value from a finite set

• Regression:
  Learning a function to map from a n-tuple to a \textit{continuous} value
Nearest Neighbor Classifier

• Example of memory-based (a.k.a instance-based, a.k.a case-based) learning

• The basic idea:
  1. Get some example set of cases with known outputs e.g diagnoses of infectious diseases by experts
  2. When you see a new case, assign its output to be the same as the most similar known case.

  Your symptoms most resemble Mr X.
  Mr X had the flu.
  Ergo you have the flu.
General Learning Task

There is a set of possible examples

\[ X = \{\vec{x}_1, \ldots, \vec{x}_n \} \]

Each example is an k-tuple of attribute values

\[ \vec{x}_1 = \langle a_1, \ldots, a_k \rangle \]

There is a target function that maps \( X \) onto some finite set \( Y \)

\[ f : X \rightarrow Y \]

The DATA is a set of tuples <example, target function values>

\[ D = \{\langle \vec{x}_1, f(\vec{x}_1) \rangle, \ldots, \langle \vec{x}_m, f(\vec{x}_m) \rangle \} \]

Find a hypothesis \( h \) such that...

\[ \forall \vec{x}, h(\vec{x}) \approx f(\vec{x}) \]
Eager vs. Lazy

**Eager learning**

- Learn model ahead of time from training data

- Explicitly learn $h$ from training data

- E.g. decision tree, linear regression, svm, neural nets, etc.

**Lazy learning**

- Delay the learning process until a query example must be labeled

- $h$ is implicitly learned from training data

- E.g. Nearest neighbor, kNN, locally weighted regression, etc.
Single Nearest Neighbor

Given some set of training data...

\[ D = \{ \langle \hat{x}_1, f(\hat{x}_1) \rangle, \ldots, \langle \hat{x}_m, f(\hat{x}_m) \rangle \} \]

...and query point \( \hat{x}_q \), predict \( f(\hat{x}_q) \)

1. Find the nearest member of the data set to the query

\[ \hat{x}_{nn} = \arg \min_{\hat{x} \in D} (d(\hat{x}, \hat{x}_q)) \]

2. Assign the nearest neighbor’s output to the query

\[ h(\hat{x}_q) = f(\hat{x}_{nn}) \]

Our hypothesis

Bryan Pardo, Machine Learning: EECS 349 Fall 2012
A Univariate Example

- Find closest point: \( \vec{x}_{nn} = \arg \min_{\vec{x} \in D} (d(\vec{x}, \vec{x}_q)) \)
- Give query its value: \( f(\vec{x}_q) = f(\vec{x}_{nn}) \)
A Univariate Example

• Find closest point. \( \vec{x}_{nn} = \arg\min_{x \in D} (d(\vec{x}, \vec{x}_q)) \)

• Give query its value \( f(\vec{x}_q) = f(\vec{x}_{nn}) \)
A Two-dimensional Example

• Voronoi diagram
What makes a memory based learner?

• A distance measure
  
  Nearest neighbor: typically Euclidean

• Number of neighbors to consider
  
  Nearest neighbor: One

• A weighting function (optional)
  
  Nearest neighbor: unused (equal weights)

• How to fit with the neighbors
  
  Nearest neighbor: Same output as nearest neighbor
K-nearest neighbor

- A distance measure
  \textit{Euclidean}
- Number of neighbors to consider \( K \)
- A weighting function (optional)
  \textit{Unused (i.e. equal weights)}
- How to fit with the neighbors
  \textit{regression: average output among } K \textit{ nearest neighbors.}
  \textit{classification: most popular output among } K \textit{ nearest neighbors}
Choosing K

• Too small of K fits the output to the noise in the dataset (overfit)

• Too large of K can make decision boundaries in classification indistinct (underfit)

• Choose K empirically using cross-validation
Ex. of KNN for classification

http://demonstrations.wolfram.com/KNearestNeighborKNNClassifier/
Ex. Of kNN regression where K=9

Reasonable job
Did smooth noise

Screws up on the ends

OK, but problem on the ends again.
Kernel Regression

- A distance measure: *Scaled Euclidean*
- Number of neighbors to consider: *All of them*
- A weighting function:

\[
w_i = \exp \left( -d(x_i, x_q)^2 \right) \frac{1}{K_W^2}
\]

Nearby points to the query are weighted strongly, far points weakly. The \( K_W \) parameter is the Kernel Width.

- How to fit with the neighbors:

\[
h(x_q) = \frac{\sum_i w_i \cdot f(x_i)}{\sum_i w_i}
\]

A weighted average
Kernel Regression

Kernel Weight = 1/32 of X-axis width

A better fit than KNN?

Kernel Weight = 1/32 of X-axis width

Definitely better than KNN! Catch: Had to play with kernel width to get this result

Kernel Weight = 1/16 of X-axis width

Nice and smooth, but are the bumps justified, or is this overfitting?
Weighting dimensions

- Suppose data points are two-dimensional
- Different dimensional weightings affect region shapes

\[ d(x, y) = (x_1 - y_1)^2 + (x_2 - y_2)^2 \]

\[ d(x, y) = (x_1 - y_1)^2 + (3x_2 - 3y_2)^2 \]
kNN and Kernel Regression

Pros

• Robust to noise
• Very effective when training data is sufficient
• Customized to each query
• Easily adapts when new training data is added

Cons

• How to weight different dimensions?
• Irrelevant dimensions
• Computationally expensive to label a new query
• High space requirements (must retain each training example)
Locally Weighted (Linear) Regression

- **Linear regression**: global, linear
  
  
  $$Err = \sum_{\tilde{x} \in D} \frac{1}{2} (f(\tilde{x}) - h(\tilde{x}))^2$$

  $$h(\tilde{x}) = \tilde{a}^T \tilde{x} + \tilde{b}$$

- **kNN**: local, constant

- **LWR**: local, linear

  $$\tilde{x}_{nn} = \arg \min_{\tilde{x} \in D} (d(\tilde{x}, \tilde{x}_q))$$

  $$Err(x_q) = \sum_{x \in D} \frac{1}{2} (f(x) - h(x))^2 \exp\left(\frac{-d(x, x_q)^2}{K_w^2}\right)$$
Locally Weighted (Linear) Regression

\[ KW = \frac{1}{16} \text{ of } x\text{-axis width.} \]

\[ KW = \frac{1}{32} \text{ of } x\text{-axis width.} \]

\[ KW = \frac{1}{8} \text{ of } x\text{-axis width.} \]

Nicer and smoother, but even now, are the bumps justified, or is this overfitting?
Locally Weighted Polynomial Regression

- Use a polynomial instead of a linear function to fit the data locally
  - Quadratic, cubic, etc.

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.

$KW = 1/100 \ x\text{-axis}$

$KW = 1/40 \ x\text{-axis}$

$KW = 1/15 \ x\text{-axis}$
Memory-based Learning

Pros

- Easily adapts to new training examples (no-retraining required)
- Can handle complex decision boundaries and functions by considering the query instance when deciding how to generalize

Cons

- Requires retaining all training examples in memory
- Slow to evaluate a new query
- Evaluation time grows with the dataset size
Summary

• Memory-based learning are “lazy”
  – Delay learning until receiving a query

• Local
  – Training data that is localized around the query contribute more to the prediction value

• Non-parametric

• Robust to noise

• Curse of dimensionality
  – Irrelevant dimensions
  – How to scale dimensions
Summary

• Nearest neighbor
  – Output the nearest neighbor’s label

• kNN
  – Output the average of the k NN’s labels

• Kernel regression
  – Output weighted average of all training data’s (or k NN’s) labels

• Locally weighted (linear) regression
  – Fit a linear function locally