Instance Based Methods I

CPS 271

Ron Parr

With content adapted from Lise Getoor
(& Tom Dietterich, Ray Mooney, Andrew Moore)

Parametric Methods

- Supervised learning
  - Linear classifiers
  - Non-linear classifiers, e.g., neural networks
- These methods are *parametric*
- Alternative: Remember stuff
- AKA: Case based or memory based

Overview

- Classification
  - Nearest neighbor
  - K-NN
- Regression

Example

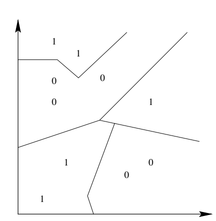
- Flood risk
- Data set:
  - GPS coordinates (features)
  - Flood data for previous hundred years
- Task: predict flood risk for new data points

Nearest Neighbor Algorithm

- Learning Algorithm:
  - Store training examples
  - “But that’s not learning…”
- Prediction Algorithm:
  - To classify a new example **x** by finding the training example ($\mathbf{x}^i$,t^i) that is *nearest* to **x**
  - Guess the class $t = t^i$
  - Learning implicit in query mechanism

Decision Boundaries

- The nearest neighbor algorithm does not explicitly compute decision boundaries. However, the decision boundaries form a subset of the Voronoi diagram for the training data.



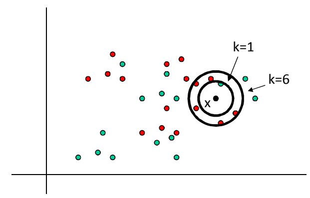
- Each line segment is equidistant between two points of opposite classes. The more examples that are stored, the more complex the decision boundaries can become.

Issues

- Noise
- Efficiency
  - Use K-D Trees
  - Still bad for high dimensions
- Distance measure (critical!)
  - How to pick
  - Questionably useful in high dimensions
- Irrelevant features

Dealing with Noise

- Consider *k nearest neighbors* (K-NN)
- Neighbors vote



common values for k: 3, 5

Picking Distance Measures

- No silver bullet
- Many rules of thumb
- Problem knowledge always helps

Distance: Preprocessing

- What if features don't have same range?
- Normalize feature values
  - Scale to same range
  - Usually –1,+1 scale

Distance Measures

- Two methods for computing similarity:
  1. Explicit similarity measurement for each pair of objects
  2. Similarity obtained indirectly based on vector of object attributes.
- Metric: d(i,j) is a metric iff
  1. d(i,j) ≥ 0 for all i, j and d(i,j) = 0 iff i = j
  2. d(i,j) = d(j,i) for all i and j
  3. d(i,j) ≤ d(i,k) + d(k,i) for all i, j and k

Distance

- Notation: n objects with p measurements

$$x^{(i)} = (x_1{}^{(i)}, x_2{}^{(i)}, \dots, x_p{}^{(i)})$$

- Most common distance metric is *Euclidean* distance:

$$d_E(x^{(i)}, x^{(j)}) = \left(\sum_{k=1}^{p} (x_k{}^{(i)} - x_k{}^{(j)})^2 \right)^{\frac{1}{2}}$$

- Makes sense in the case where the different measurements are commensurate; each is variable measured in the same units. If the measurements are different, say length and weight, it is not clear.

Standardization

When variables are not commensurate, we can standardize them by dividing by the sample standard deviation. This makes them all equally important.

The estimate for the standard deviation of x_k:

$$\hat{\sigma}_k = \left(\frac{1}{n} \sum_{i=1}^{n} \left(x_k^{(i)} - \bar{x}_k \right)^2 \right)^{\frac{1}{2}}$$

where x_k is the sample mean:

$$\bar{x}_k = \frac{1}{n} \sum_{i=1}^{n} x_k^{(i)}$$

But what about correlation???

Mahalanobis distance

$$d_{MH}(x^{(i)}, x^{(j)}) = \left(\left(x^{(i)} - x^{(j)} \right)^T \Sigma^{-1} \left(x^{(i)} - x^{(j)} \right) \right)^{\frac{1}{2}}$$

Inverse covariance matrix
(compare with Gaussian)

1. It automatically accounts for the scaling of the coordinate axes
2. It corrects for correlation between the different features

Price:
1. The covariance matrices can be hard to determine accurately
2. The memory and time requirements grow quadratically rather than linearly with the number of features.

Other Distance Metrics

- Minkowski or L_λ metric:

$$d_\lambda(x^{(i)}, x^{(j)}) = \left(\sum_{k=1}^{p} (x_k^{(i)} - x_k^{(j)})^\lambda \right)^{\frac{1}{\lambda}}$$

- Manhattan, city block or L_1 metric:

$$d_1(x^{(i)}, x^{(j)}) = \sum_{k=1}^{p} \left| x_k^{(i)} - x_k^{(j)} \right|$$

- L_∞

$$d_\infty(x^{(i)}, x^{(j)}) = \max_k \left| x_k^{(i)} - x_k^{(j)} \right|$$

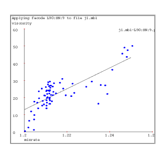
Nearest Neighbor Summary

- Advantages
  - Variable-sized hypothesis space
  - Learning is extremely efficient (low d)
  - Very flexible decision boundaries
- Disadvantages
  - Distance function must be carefully chosen
  - Irrelevant or correlated features must be eliminated
  - Typically cannot handle more than 30 features
  - Memory costs
  - Expensive queries

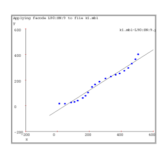
Non-Parametric Regression Methods

- Carry over our intuitions from classificcation
- Look at some set of “neighbors”
- Some issues assumed away (distance metric)
  [Is this valid?]

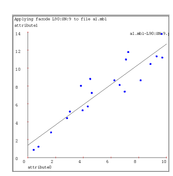
Why not just use Linear Regression?



Here, linear regression manages to capture a significant trend in the data, but there is visual evidence of bias.



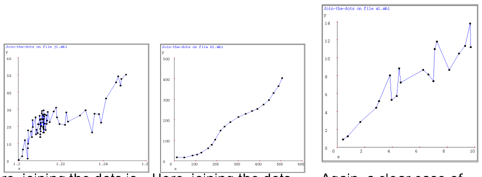
Here, linear regression appears to have a much better fit, but the bias is very clear.



Here, linear regression may indeed be the right thing.

Bias: the underlying choice of model *(in this case, a line)* cannot, with any choice of parameters *(constant term and slope)* and with any amount of data *(the dots)* capture the full relationship.

Why not just Join the Dots?



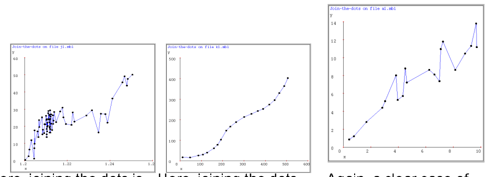
Here, joining the dots is clearly fitting noise.

Here, joining the dots looks very sensible.

Again, a clear case of noise fitting.

Why is fitting the noise so bad?

Why not just Join the Dots?



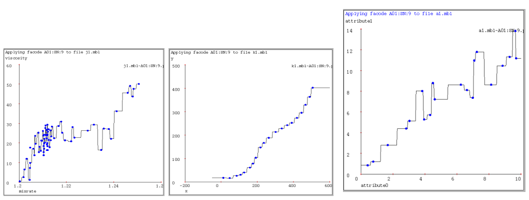
Here, joining the dots is clearly fitting noise.

Here, joining the dots looks very sensible.

Again, a clear case of noise fitting.

Why is fitting the noise so bad?

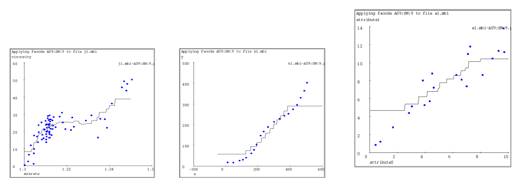
One-Nearest Neighbor



Similar to Join The Dots with Pro and Con.

- PRO: It is easy to implement with multivariate inputs.
- CON: It no longer interpolates locally

k-Nearest Neighbor (here k=9)



A magnificent job of noise-smoothing. Three cheers for 9-nearest-neighbor. But the lack of gradients and the jerkiness isn't good.

Appalling behavior! Loses all the detail that join-the-dots and 1-nearest-neighbor gave us, yet smears the ends.

Fits much less of the noise, captures trends. But still, frankly, pathetic compared with linear regression.

K-nearest neighbor for function fitting smoothes away noise, but there are clear deficiencies.

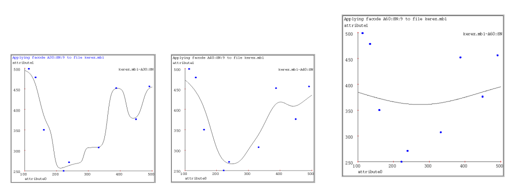
What can we do about all the discontinuities that k-NN gives us?

Kernel Regression
(Not Dual/GP Regression)

Four things make a memory based learner:

1. *A distance metric*
   Scaled Euclidian
2. *How many nearby neighbors to look at?*
   All of them
3. *A weighting function (optional)*
   $w_i = exp(-D(x_i, query)^2 / K_w^2)$
   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:
   predict = $\Sigma w_i y_i / \Sigma w_i$

Kernel Regression Predictions

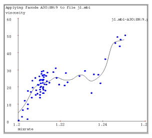


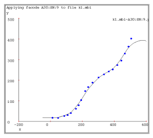
K_W=10 K_W=20 K_W=80

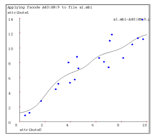
Increasing the kernel width K_w means further away points get an opportunity to influence you.

As K_w→infinity, the prediction tends to the global average.

Kernel Regression on our test cases







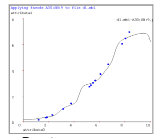
K_W=1/32 of x-axis width. It's nice to see a smooth curve at last. But rather bumpy. If K_w gets any higher, the fit is poor.

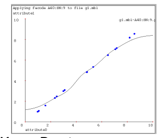
K_W=1/32 of x-axis width. But… AWM needed to choose the right K_W to achieve this.

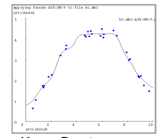
K_W=1/16 axis width. Nice and smooth, but are the bumps justified, or is this overfitting?

Choosing a good K_w is important. Not just for Kernel Regression, but for all the locally weighted learners

Kernel Regression can look bad







K_W = Best.

Clearly not capturing the simple structure of the data.. Note the complete failure to extrapolate at edges.

K_W = Best.

Also much too local fitting. Why wouldn't increasing K_w help? Because then it would all be "smeared".

K_W = Best.

Three noisy linear segments. But best kernel regression gives poor gradients.

Time to try something more powerful…

Locally Weighted Regression

- Like kernel regression LWR uses "neighbors"
- Kernel regression simply averages neighbors
- LWR finds a *locally linear model*

Locally Weighted Regression

Four things make a memory-based learner:

1. *A distance metric*

   Scaled Euclidian
2. *How many nearby neighbors to look at?*

   All of them
3. *A weighting function (optional)*

   $w^{(k)} = exp(-D(x^{(k)}, x_{query})^2 / K_w^2)$

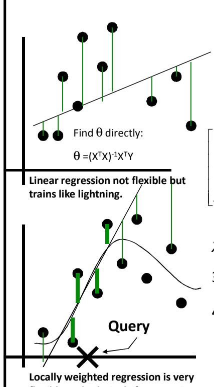
   Nearby points to the query are weighted strongly, far points weakly. The K_w parameter is the **Kernel Width**.
4. *How to fit with the local points?*

   First form a local linear model. Find the θ that minimizes the locally weighted sum of squared residuals:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} w^{(i)^2} \left(y^{(i)} - \theta^T x^{(i)} \right)^2$$

Then predict $y_{predict}=\theta^T x_{query}$

Note: Here **w** are neighbor weights θ are regression weights.



How LWR works

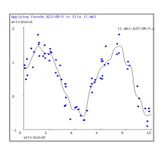
1. For each point ($\mathbf{x}^{(i)}$,$y^{(i)}$) compute $w^{(i)}$.
2. Let $WX = Diag(w^{(i)},..w^{(N)})X$

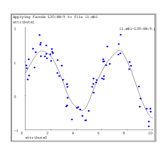
$$\begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_D^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_D^{(2)} \\ \vdots & \vdots & & \vdots \\ x_1^{(N)} & x_2^{(N)} & \cdots & x_D^{(N)} \end{bmatrix} \rightarrow \begin{bmatrix} w^{(1)}x_1^{(1)} & w^{(1)}x_2^{(1)} & \cdots & w^{(1)}x_D^{(1)} \\ w^{(2)}x_1^{(2)} & w^{(2)}x_2^{(2)} & \cdots & w^{(2)}x_D^{(2)} \\ \vdots & \vdots & & \vdots \\ w^{(N)}x_1^{(N)} & w^{(N)}x_2^{(N)} & \cdots & w^{(N)}x_D^{(N)} \end{bmatrix}$$

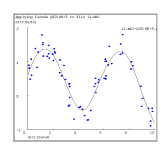
X - - > *WX*

3. Let $WY=Diag(w^{(1)},..w^{(N)})Y$, so that $y^{(i)} \rightarrow w^{(i)}y^{(i)}$
4. $\theta = (WX^TWX)^{-1}(WX^TWY)$

Locally weighted Polynomial regression







Kernel Regression
Kernel width K_W at optimal level.

KW = 1/100 x-axis

LW Linear Regression
Kernel width K_W at optimal level.

KW = 1/40 x-axis

LW Quadratic Regression
Kernel width K_W at optimal level.

KW = 1/15 x-axis

RP: We defer discussion of cost/benefit of extra terms

When's Quadratic better than Linear?

- It can let you use a wider kernel without introducing bias,
- but in higher dimensions is expensive, needs more data.
- Two "Part-way-between-linear-and-quadratic" polynomials:
  - "Ellipses": Add x_i^2 terms to the model, but not cross-terms (no $x_i x_j$ where $i=j$)
  - "Circles": Add only one extra term to the model:

$$x_{D+1} = \sum_{j=1}^{D} x_j^2$$

Locally Weighted Learning: Variants

- Range Searching: Average all neighbors w/in given range
- Range-based linear regression
- Linear Regression on K-nearest-neighbors
- Weighting functions that decay to 0 at the k^{th} nn
- Locally weighted Iteratively Reweighted Least Squares
- Locally weighted Logistic Regression
- Locally weighted classifiers

- Multilinear Interpolation
- Kuhn-Triangulation-based Interpolation
- Spline Smoothers

Copyright © 2001, Andrew W. Moore

Non-Parametric Methods: Conclusions

- Very expressive method for
  - Classification
  - Regression
- Perhaps too powerful
- Can be memory/compute intensive for queries
- Heavy dependence upon distance/kernel
- Good method to use when:
  - Data fills feature space well
  - Good intuitions about distance