Data Structures for Fast Kernel Regression and Locally Weighted Regression

The Auton Lab

Carnegie Mellon University

www.autonlab.org

What is LWPR?

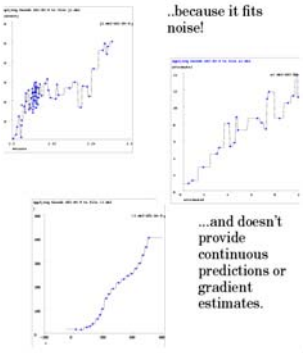
Locally weighted polynomial regression is an algorithm for learning non-linear numeric functions.

Here, we will...

- Review LWPR (travelling via Nearest Neighbor and Kernel Regression).
- Praise its virtues.
- Mourn its computational expense.
- Discuss recently proposed methods for making it faster.
- Introduce and evaluate a new multiresolution data structure.

2

Why not just use 1-nearest neighbor?

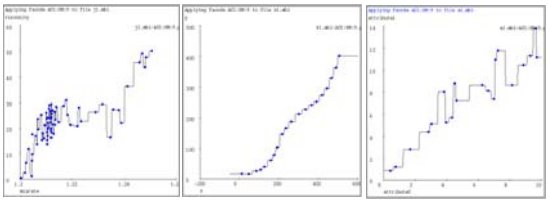


..because it fits noise!

...and doesn't provide continuous predictions or gradient estimates.

3

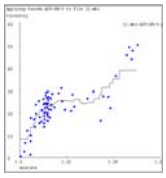
Why not just use one-Nearest Neighbor?



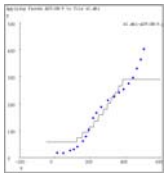
- Discontinuities
- Fits the noise

4

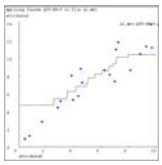
k-Nearest Neighbor (here k=9)



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?

5

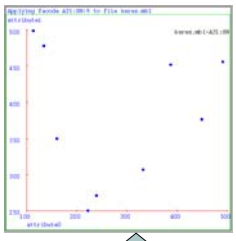
Kernel Regression

Four things make a memory based learner:

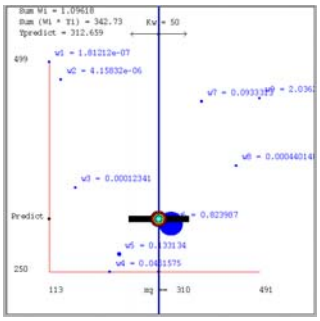
1. *A distance metric*
   Scaled Euclidian
2. *How many nearby neighbors to look at?*
   All of them
3. *A weighting function (optional)*
   $w_i = exp(-D(x_i, query)^2 / K_w^2)$
   Nearby points to the query are weighted strongly, far points weakly. The K_W parameter is the **Kernel Width**. Very important.
4. *How to fit with the local points?*
   Predict the weighted average of the outputs:
   predict = $\Sigma w_i y_i / \Sigma w_i$

6

Kernel Regression in Pictures



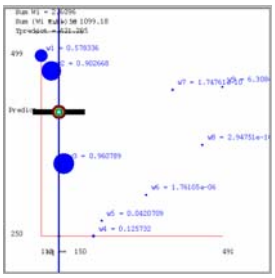
Take this dataset…



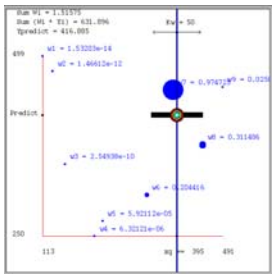
..and do a kernel prediction with x_q (query) = 310, K_w = 50.

7

Varying the Query



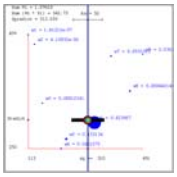
x_q = 150



x_q = 395

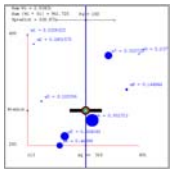
8

Varying the kernel width



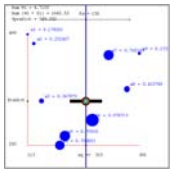
x_q = 310

K_W = 50 (see the double arrow at top of diagram)



x_q = 310 (the same)

K_W = 100



x_q = 310 (the same)

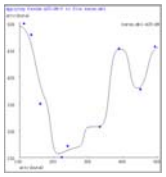
K_W = 150

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

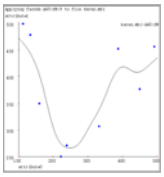
As K_w→infinity, the prediction tends to the global average.

9

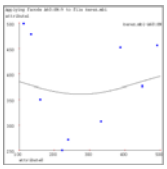
Kernel Regression Predictions



K_W=10



K_W=20



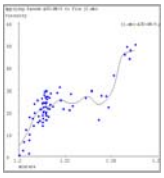
K_W=80

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.

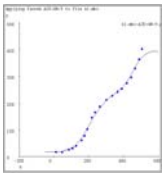
10

Kernel Regression on our test cases



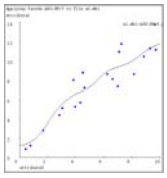
KW=1/32 of x-axis width.

It's nice to see a smooth curve at last. But rather bumpy. If Kw gets any higher, the fit is poor.



KW=1/32 of x-axis width.

Quite splendid. Well done, kernel regression. The author needed to choose the right K_w to achieve this.



KW=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 we're about to see.

11

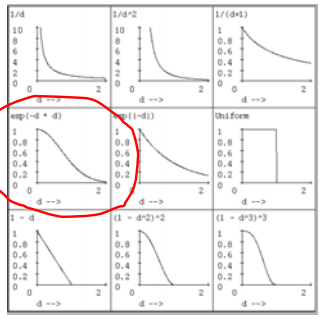
Weighting functions

Let

$d=D(x_i,x_{query})/K_W$

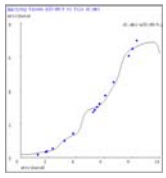
Then here are some commonly used weighting functions…

(we use a Gaussian)



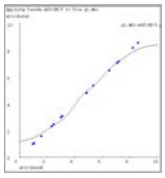
12

Kernel Regression can look bad



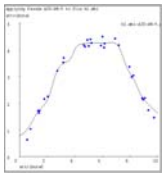
KW = Best.

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



KW = Best.

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



KW = Best.

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

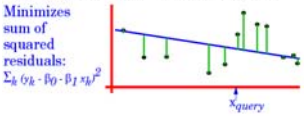
Time to try something more powerful…

13

Locally weighted linear regression

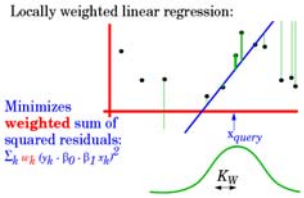
Global Regression: $(y = \beta_0 + \beta_1 x + noise)$:

Minimizes sum of squared residuals: $\Sigma_k (y_k - \beta_0 - \beta_1 x_k)^2$



Locally weighted linear regression:

Minimizes **weighted** sum of squared residuals: $\Sigma_k w_k (y_k - \beta_0 - \beta_1 x_k)^2$



14

Locally Weighted Regression

Four things make a memory-based learner:

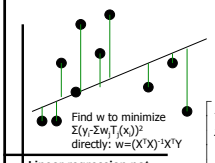
1. *A distance metric*
   Scaled Euclidian
2. *How many nearby neighbors to look at?*
   All of them
3. *A weighting function (optional)*
   $w_k = \exp(-D(x_k, x_{query})^2 / K_w^2)$
   Nearby points to the query are weighted strongly, far points weakly. The K_w parameter is the **Kernel Width**.
4. *How to fit with the local points?*
   First form a local linear model. Find the β that minimizes the locally weighted sum of squared residuals:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{k=1}^{N} w_k^2 \left(y_k - \beta^T x_k\right)^2$$

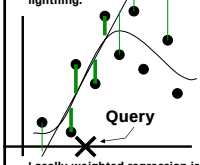
Then predict $y_{predict} = \underline{\beta}^T x_{query}$

15

How LWR works



Linear regression not flexible but trains like lightning.



Locally weighted regression is very flexible and fast to train.

1. For each point $(\mathbf{x}_i, y_i)$ compute w_k.
2. Let $WX = Diag(w_1..w_N)X$

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \rightarrow \begin{bmatrix} w_1 & w_1 x_{11} & w_1 x_{12} & \cdots & w_1 x_{1D} \\ w_2 & w_2 x_{21} & w_2 x_{22} & \cdots & w_2 x_{2D} \\ \vdots & \vdots & \vdots & & \vdots \\ w_N & w_N x_{N1} & w_N x_{N2} & \cdots & w_N x_{ND} \end{bmatrix}$$

X - - > WX

3. Let $WY = Diag(w_1..w_N)Y$, so that $y_k \rightarrow w_k y_k$
4. $\beta = (WX^T WX)^{-1}(WX^T WY)$

16

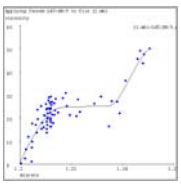
Input X matrix of inputs: X[k] [i] = i'th component of k'th input point.
Input Y matrix of outputs: Y[k] = k'th output value.
Input xq = query input. Input kwidth.

WXTWX = empty (D+1) x (D+1) matrix
WXTWY = empty (D+1) x 1 matrix

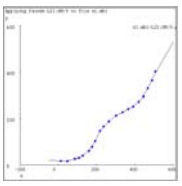
```
for ( k = 1 ; k <= N ; k = k + 1 )
    /* Compute weight of kth point  */
    wk = weight_function( distance( xq , X[k] ) / kwidth )

    /* Add to (WX) ^T (WX) matrix */
     for ( i = 0 ; i <= D ; i = i + 1 )
            for ( j = 0 ; j <= D ; j = j + 1 )
                    if ( i == 0 ) xki = 1 else xki = X[k] [i]
                    if ( j == 0 ) xkj = 1 else xkj = X[k] [j]
                    WXTWX [i] [j] = WXTWX [i] [j] + wk * wk * xki * xkj

    /*  Add to (WX) ^T (WY) vector */
    for ( i = 0 ; i <= D ; i = i + 1 )
          if ( i == 0 ) xki = 1 else xki = X[k] [i]
          WXTWY [i] = WXTWY [i] + wk * wk * xki * Y[k]

/* Compute the local beta.  Call your favorite linear equation solver.  Recommend Cholesky
     Decomposition for speed.  Recommend Singular Val Decomp for Robustness. */
beta = (WXTWX)^-1 (WXTWY)
ypredict = beta[0] + beta[1]*xq[1] + beta[2]*xq[2] + … beta[D]*xq[D]
```

17

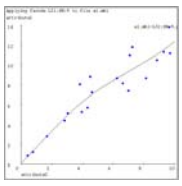
LWR on our test cases



KW = 1/16 of x-axis width.



KW = 1/32 of x-axis width.

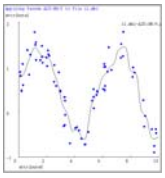


KW = 1/8 of x-axis width.

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

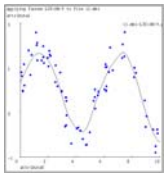
18

Locally weighted Polynomial regression



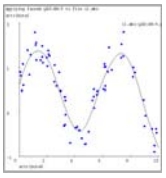
Kernel Regression
Kernel width K_W at optimal level.

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

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.

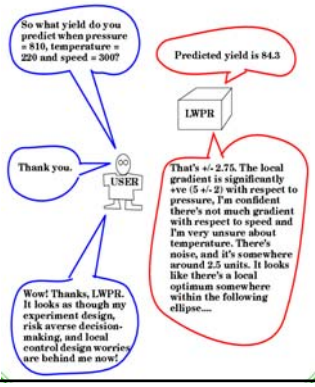
19

Nice things about LWPR

1. It naturally extends to multiple inputs / multiple outputs.
2. There's no interference: new data doesn't cause unlearning of old parts of the model.
3. Incorporating new data is very cheap!
4. Handles functions where the output is linear in some variables and non-linear in others very naturally.
5. Leave-One-Out cross-validation is cheap
6. **You don't just get a prediction. You can get a full regression analysis!**

20

LWPR Regression Analysis



21

LWPR: The Dark Side

Cost of each prediction grows linearly with the number of datapoints.

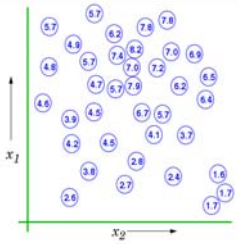
With N datapoints, and with M polynomial terms....

- training is FREE
- each individual prediction costs $O(NM^2 + M^3)$

What can we do????

22

Idea One: Editing

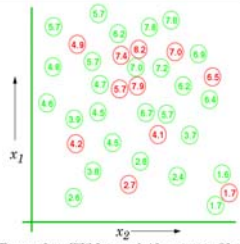


Here we have $N=36$ examples, each with two inputs (x_1,x_2) and one output.

Instead of retaining and searching over all N, we could choose to keep only a few "representative" points called **prototypes**.

23

Idea One: Editing



- Examples: [Kibler and Aha, 1988; Skalak, 1994].
- Other editing tricks: maintain averages in the prototypes [Aha, 1992].

BUT: Lose accuracy. Lose ability to give different queries different kernel widths. Lose cheap L-O-O. Hard to estimate noise.

24

Idea 2: cache output surface

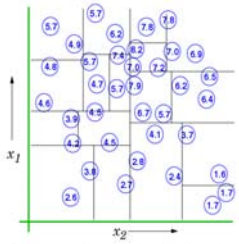
- First, decide which instance-based algorithm you'll use, fix the kernel width etc.
- **Caching step:** perform lots of (slow) predictions but save the results in some interpolative table.
  - Simple (but expensive): cache in a multidimensional array.
  - Fancy (still expensive): cache in a *k*d-tree and predict with multilinear interpolation [Grosse, 1992].
  - Simple, cheaper, but discontinuous: cache linear models inside leaves of a tree. e.g. [Quinlan, 1993].

BUT: Must commit to a kernel width during caching. Lose regression analysis. Caching step can be very very very expensive. Incrementally adding points can be hard.

25

Idea 3: Range Searching

Embed your data in the leaves of a *k*d-tree [Friedman et al, 1977]

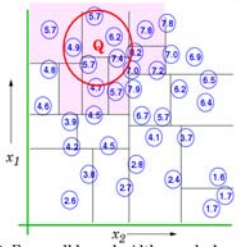


If you have a small kernel width, then beyond a certain distance from your query, all weights will be very close to zero.

You can quickly search the tree to get the set of points with non-zero-weight.

26

Idea 3: Range Searching



PRO: For small kernel widths much cheaper than linear search. Different queries can have different kernels. And you can do full regression analysis.

CON: For large kernel widths or very dense datasets, you're still in trouble.

27

This talk: Multiresolution Approach

What do we really need in order to do a locally weighted regression query?

$$\beta = \left(\Sigma_k w_k^2 \mathbf{x}_k^T \mathbf{x}_k\right)^{-1} \left(\Sigma_k w_k^2 \mathbf{x}_k^T y_k\right)$$

Let's look at how we can build the

$\left(\Sigma_k w_k^2 \mathbf{x}_k^T \mathbf{x}_k\right)$ matrix...

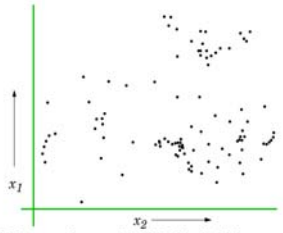
$$\begin{bmatrix} \sum w_k^2 x_{k1}^2 & \sum w_k^2 x_{k1} x_{k2} & \cdots & \sum w_k^2 x_{k1} x_{kN} \\ \sum w_k^2 x_{k1} x_{k2} & \sum w_k^2 x_{k2}^2 & \cdots & \sum w_k^2 x_{k2} x_{kN} \\ \vdots & \vdots & & \vdots \\ \sum w_k^2 x_{k1} x_{kN} & \sum w_k^2 x_{k2} x_{kN} & \cdots & \sum w_k^2 x_{kN}^2 \end{bmatrix}$$

The slow way:

```
wwxtx = MxM matrix of zeros.
for ( k = 1 ; k <= N ; k := k + 1 )
   wwxtx := wwxtx + w_k^2 x_k^T x_k
```

28

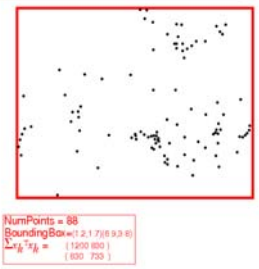
A multiresolution structure



Take your data and build a kind of *k*d-tree structure with extra cached information...

29

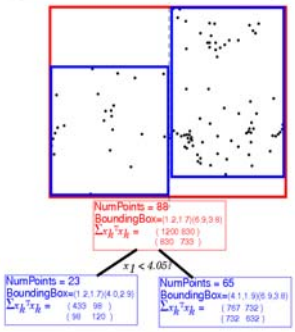
Top level of tree



NumPoints = 88
BoundingBox = (1.2 1.7)(6.9 3.6)
$\sum w_k^2 x_k^T x_k$ = (1200 600) (600 733)

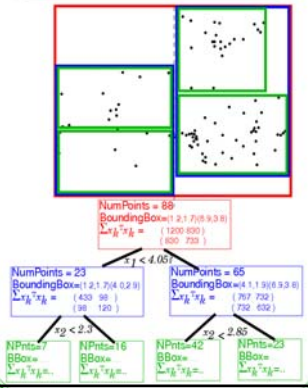
30

Second level of tree



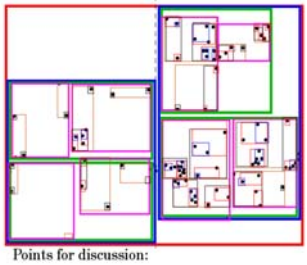
31

Third level of tree



32

Further levels of tree



Points for discussion:

- The tree can be added to incrementally
- Subtract out centroids from $\Sigma_k x_k^T x_k$ matrices for numerical stability
- You must similarly compute $\Sigma_k x_k^T y_k$ and $\Sigma_k y_k^T$

This slide is beautiful but took a long time to create. Appreciate it.

33

Using the tree.

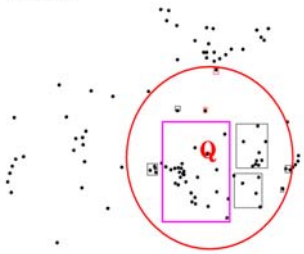
If our weighting function was... $W(Dist) = 1$ *if* $Dist < KW$ $= 0$ *otherwise*

Instead of... $W(Dist) = \exp(-dist^2 / Kw^2)$

34

Using the tree

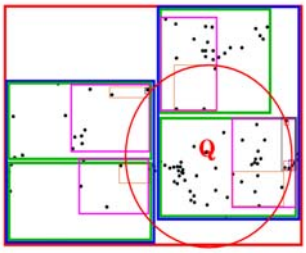
If we had a step weighting function then we would merely add together matrices of all nodes that were entirely within the critical distance.



35

Using the tree

To find this set of nodes, we'd merely need to visit all nodes that intersect the critical distance..

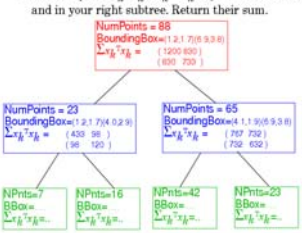


36

With a Gaussian weight fn...

Recurse down the tree.

- At each node you must compute the weighted sum $\sum_k w_k^2 \mathbf{x}_k^T \mathbf{x}_k$ of all points below in the tree.
- If node has only one point $\mathbf{x}_j$ return $w_j^2 \mathbf{x}_j^T \mathbf{x}_j$
- Else compute $\sum_k w_k^2 \mathbf{x}_k^T \mathbf{x}_k$ of your left subtree and in your right subtree. Return their sum.

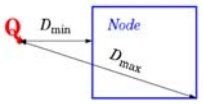


- But ***SOMETIMES*** you may discover that you can avoid searching a node....

37

Cutoff Criterion

You're about to compute $\sum_k w_k^2 \mathbf{x}_k^T \mathbf{x}_k$ for a node.



You know that all weights inside the node must lie in the range

$w_{min}=weight(D_{max}) <= w_k <= weight(D_{min})=w_{max}$

So if w_{max} and w_{min} are very similar, then we know that

$\sum_k w_k^2 \mathbf{x}_k^T \mathbf{x}_k = (w_{max}+w_{min})/2 * \sum_k \mathbf{x}_k^T \mathbf{x}_k$

That information is inside the current node.

w_{max} and w_{min} are very similar when (1) the weight function is flat or (2) the node is small.

38

Cutoff Rule

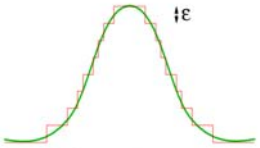
To compute $\sum_k w_k^2 \mathbf{x}_k^T \mathbf{x}_k$ for a non-leaf node:

- **If** $w_{max} - w_{min} < \varepsilon$ then return

  $(w_{max}+w_{min})/2 * \sum_k \mathbf{x}_k^T \mathbf{x}_k$

- **Else**

  recursively compute $\sum_k w_k^2 \mathbf{x}_k^T \mathbf{x}_k$ for left and right children and add them.



This is equivalent to using some approximate weighting function that never deviates from desired weighting function by more than ε

39

One final little detail...

- $\sum_k w_k^2$ can get as small as $O(\varepsilon)$ if we are far from data.
- So approximations of $O(\varepsilon)$ can become dangerously significant.

  Thus we alter our criterion to:

  Cutoff if $w_{max} - w_{min} < \tau W_{TOTAL}$

  where τ is a small constant and where W_{TOTAL} is the total sum of squared weights of all datapoints in this query.

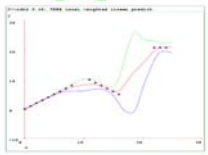
- W_{TOTAL} is not known in advance, but has a lower bound W_{SOFAR}. So in actuality...

  Cutoff if $w_{max} - w_{min} < \tau W_{SOFAR}$

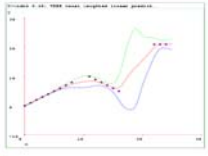
40

Slight Approximation

Examine the difference for locally weighted linear regression.



The linear, exact, method.



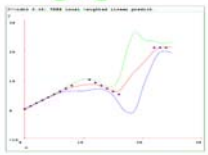
Multires, using $\tau = 10^{-7}$

..it's identical

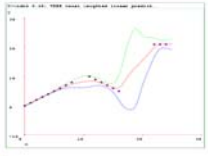
41

Medium Approximation

Examine the difference for locally weighted linear regression.



The linear, exact, method.



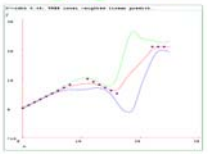
Multires, using $\tau = 0.05$

..it's indistinguish able

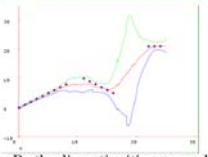
42

Gross Approximation

$KW = 4$



The linear, exact, method.



Multires, using $\tau = 0.5$

..it's close but with tiny discont-inuities

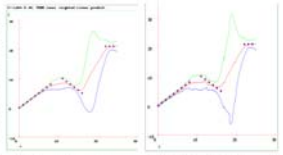
Do the discontinuities pose a danger for algorithms that compute derivatives?

43

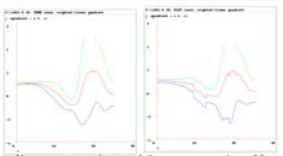
Gradient Estimation

Exact Method | τ =0.5 fast method

Predicted value



Predicted gradient



Gradients are estimated from regression. Not by numerical differencing.

44

Speed and Accuracy

Using locally weighted regression with kernel width 1/32 in unit-scaled coordinates.

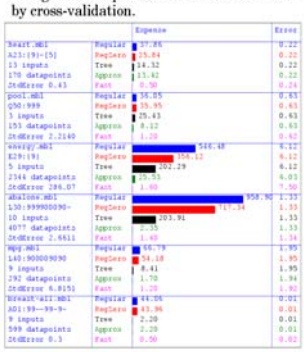
		Expense	Error
Bkart.mbl	Regular	42.16	0.27
L40:(9)	RegZero	32.95	0.28
13 inputs	Tree	21.23	0.28
170 datapoints	Approx	18.93	0.28
2tdError 0.45	Fast	14.12	0.28
pool.mbl	Regular	34.65	0.63
L40:(9)	RegZero	33.45	0.63
3 inputs	Tree	22.33	0.63
151 datapoints	Approx	4.41	0.63
2tdError 2.2140	Fast	0.80	0.62
energy.mbl	Regular	525.87	11.93
L40:(9)	RegZero	484.30	11.93
5 inputs	Tree	323.37	11.93
2344 datapoints	Approx	9.44	15.15
2tdError 286.07	Fast	1.10	21.60
abalone.mbl	Regular	964.00	1.65
L40:(9)	RegZero	864.00	1.65
10 inputs	Tree	469.00	1.65
4077 datapoints	Approx	5.80	1.67
2tdError 2.66	Fast	1.70	1.71
Wpg.mbl	Regular	70.10	1.92
L40:(9)	RegZero	55.18	1.92
9 inputs	Tree	34.35	1.92
292 datapoints	Approx	11.61	1.92
2tdError 6.8151	Fast	2.00	1.93
breast-all.mbl	Regular	141.40	0.03
L40:(9)	RegZero	124.10	0.03
9 inputs	Tree	79.88	0.03
599 datapoints	Approx	13.82	0.03
2tdError 0.3	Fast	4.21	0.02

Regular: standard linear method. RegZero: Linear method, but ignores zero-weights. Tree: Multires with $\tau = 1e\text{-}7$. Approx: $\tau = 0.08$. Fast: $\tau = 0.8$. Expense is seconds for 100 predictions.

45

Speed and Accuracy

Using dataset-specific LWPR model selected by cross-validation.

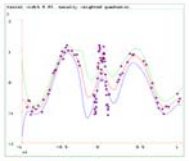


		Expense	Error
Bkart.mbl	Regular	37.86	0.22
A23:(9)-[5]	RegZero	25.84	0.22
13 inputs	Tree	14.32	0.22
170 datapoints	Approx	13.42	0.22
2tdError 0.45	Fast	2.50	0.24
pool.mbl	Regular	35.85	0.63
q50:999	RegZero	25.95	0.63
3 inputs	Tree	25.43	0.63
151 datapoints	Approx	8.12	0.63
2tdError 2.2140	Fast	1.20	0.62
energy.mbl	Regular	516.48	6.12
E29:(9)	RegZero	356.12	6.12
5 inputs	Tree	302.29	6.12
2344 datapoints	Approx	15.75	6.03
2tdError 286.07	Fast	1.50	7.50
abalone.mbl	Regular	958.90	1.33
L30:999900090-	RegZero	715.34	1.33
10 inputs	Tree	203.91	1.33
4077 datapoints	Approx	2.35	1.33
2tdError 2.6611	Fast	1.40	1.34
Wpg.mbl	Regular	86.79	1.95
L40:900009090	RegZero	54.18	1.95
9 inputs	Tree	8.41	1.95
292 datapoints	Approx	1.70	1.94
2tdError 6.8151	Fast	1.20	1.92
breast-all.mbl	Regular	44.06	0.01
A01:99--99-9-	RegZero	43.96	0.01
9 inputs	Tree	2.20	0.01
599 datapoints	Approx	2.20	0.01
2tdError 0.3	Fast	0.50	0.02

Regular: standard linear method. RegZero: Linear method, but ignores zero-weights. Tree: Multires with $\tau = 1e\text{-}7$. Approx: $\tau = 0.08$. Fast: $\tau = 0.8$. Expense is seconds for 100 predictions.

46

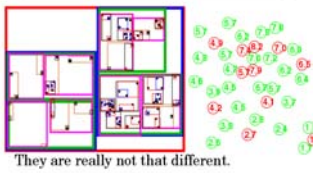
Variable kernel-widths



Using Fixed Kernel width		Using Variable K-width		Using Variable K-width Goal Weight 8.0		
K-width	Mean Error	Goal Weight	Mean Error	Method	Mean Error	MSec / Predict
0.25000	0.41	64	0.19			
0.12500	0.24	32	0.13	Linear	0.104	1400
0.06250	0.24	16	0.11	$\tau=10^{-7}$	0.104	395
0.03125	0.22	8	0.10	$\tau=0.05$	0.103	181
0.01562	0.29	4	0.10	$\tau=0.5$	0.107	165
0.00781	0.37	2	0.11			
0.00391	0.41	1	0.15			
0.00195	0.51	0.5	0.51			

47

Multiresolution versus Prototypes



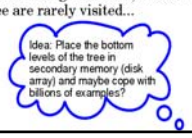
They are really not that different.

Nodes in tree can be seen as prototypes summarizing all the instances below them.

48

Comments

- Multiresolution trees make LWPR much faster without significant sacrifices.
- But predictions are still not free!
- This work extends [Deng+Moore 1995]'s kernel regression work.
- Similar multiresolution tricks are used in computer graphics and galaxy simulations
- Ongoing work plays similar tricks for local classification algorithms:
  e.g. Locally weighted logistic regression [Deng+Moore, 1997]
- Note that with large datasets, the leaves of the tree are rarely visited...



49