■ OBJECT AND MATERIAL EIGENFUNCTIONS – Cont'd.

• Laplacian eigenfunctions and neural networks:...

Simplified overall classification decision pipeline:

• Given, calculated probability values for an unclassified segment:

$$\begin{matrix} p_0^1, \dots, p_0^H \\ \vdots \\ p_C^1, \dots, p_C^H \end{matrix}$$

• Computation of CLASS-specific probability values using optimized mathematical modeling functions P_{cl}, $cl = 0, \dots, C$:

$$\begin{aligned} P_0 &= P_0\left(p_0^1, \dots, p_C^H, \alpha_0, \beta_0, \gamma_0, \dots\right)^* \\ &\vdots \\ P_C &= P_C\left(p_0^1, \dots, p_C^H, \alpha_C, \beta_C, \gamma_C, \dots\right)^* \end{aligned}$$

• Threshold-based decision-making and determining index set of class(es) recognized:

$$I \subseteq \{0, 1, \dots, C\}$$

* The parameters α_{cl}, β_{cl}, γ_{cl} etc. merely serve as "placeholders" of model function parameters – ideally having values "learned" from real image data.

In summary, the presented methods make it possible to calculate probability values for a new, unclassified image segment. These probability values define to what degree it is likely, probable, that the unclassified segment reflects specific scale characteristics (for H scales) for classes $0, 1, \dots, C$. We call these probabilities p_{cl}^{sc}, where $0 \le p_{cl}^{sc} \le 1$ and $sc = 1, \dots, H$, $cl = 0, \dots, C$. Therefore, the goal at this point is best-possible image classification using these multi-scale probabilities:

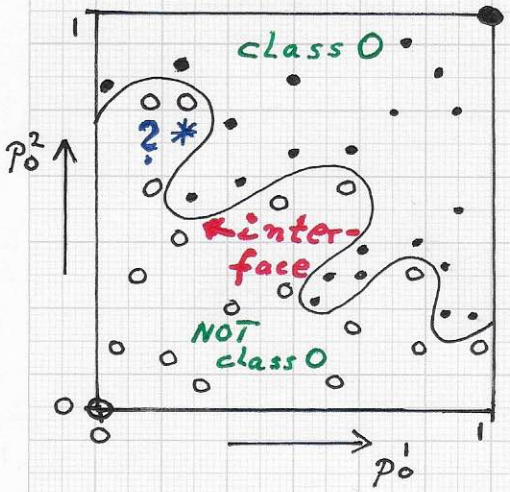
i) Select an appropriate model for the data of a complexity that is NECESSARY AND SUFFICIENT as far as parameters, i.e., degrees of freedom, are concerned. This model is defined via a mathematical function(s).

...

■ OBJECT AND MATERIAL EIGENFUNCTIONS - Cont'd.

• Laplacian eigenfunctions and neural networks: ...

Simple synthetic scenario for the problem of constructing a CLASS-specific probability function P_o, based on only two probabilities p_o^1 and p_o^2 (two scales):



For example, the function P_o could be defined as

$$P_o(p_o^1, p_o^2) = \begin{cases} 0, & \text{if } p_o^1 = p_o^2 = 0 \\ 1, & \text{if } p_o^1 = p_o^2 = 1 \\ \text{"Local radial basis function-based approximation"}, & \text{otherwise} \end{cases}$$

The tuples shown as '0' should return 0 for P_o ("not class O"), and the tuples shown as '●' should return 1 for P_o ("class O").

Since the "interface" sketched in the figure cannot be defined precisely, one must rely on "training samples" to represent classification close to the interface.

ii) Perform the necessary number of "training, testing and validation experiments," required for the values of the model functions' parameters – with the objective of minimizing the resulting "classification error."

{iii) Update and adjust the model functions' parameter values as part of the use of the classification software.}

The figure (left) shows a very simple synthetic scenario, where only one class (class O) is considered and class-O membership is determined by only two probability values, p_o^1 and p_o^2, i.e., data for two scales, are used for classification. Based on the classification of the samples in the domain (class-O and NOT-class-O samples), one must compute a class membership value for '*' – a probability value in [0,1]. ...

■ OBJECT AND MATERIAL EIGENFUNCTIONS – Cont'd.

• Laplacian eigenfunctions and neural networks:...

Using SHEPARD's method as basic building block for local approximation of (probability) function values:

• **General definition:**

"Given sites x_i and associated values f_i, $i = 1,\dots,N$, in some multi-dimensional domain, one can approximate a value f for any location x by using inverse distance values as the weights for the given f_i-values."

• **Specific scheme:**

$$f(x) = \frac{\sum_{i=1}^{N} \frac{1}{d_i^2} f_i}{\sum_{i=1}^{N} \frac{1}{d_i^2}},$$

where

$$d_i^2 = \| x - x_i \|^2,$$

$$f_i = f(x_i).$$

(For evaluation, one must simply define that $f(x_i) = f_i$ to avoid dividing by zero.)

• **Simple univariate case:**

$$f(x) = \frac{\sum_{i=1}^{N} f_i / (x - x_i)^2}{\sum_{i=1}^{N} 1 / (x - x_i)^2},$$

where $x \neq x_i$.

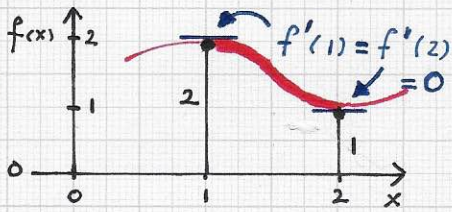
For the computation of the needed membership probability value – a value for $P_o = P_o(p_o^1, p_o^2)$ in the example – one must employ a simple, efficient, local and scalable approximation method. Primarily, a method suitable for our application must be computationally efficient, in terms of time and data structure complexity, and must be usable in a high-dimensional setting. SHEPARD'S METHOD, and several of its modifications, is a viable method to consider for our purpose: It is simple, can be used efficiently via its local definition and a multi-dimensional data structure supporting rapid data search (like a BSP and binary tree). Further, it is possible to add parameters to the basic Shepard approximation function, to achieve best-possible classification results.

...

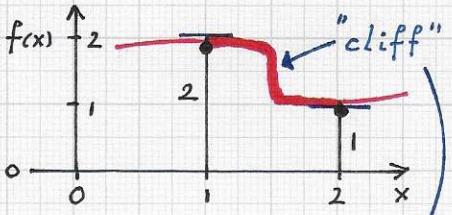
■ OBJECT AND MATERIAL EIGENFUNCTIONS - Cont'd.

- Laplacian eigenfunctions and neural networks:...
  - Univariate examples:

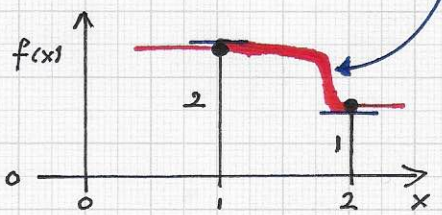
"gradual change"



"abrupt change - cliff"



"moved cliff"



Interpolating functions for two values to be interpolated, $f_1 = 2$ and $f_2 = 1$. All functions satisfy the interpolation conditions, i.e., $f(1) = 2$ and $f(2) = 1$.

The sketched functions use different exponents for the distance function:

- Top:

$$f(x) = \sum_i f_i / d_i^2 \Big/ \sum_i 1/d_i^2$$

- Middle:

$$f(x) = \sum_i f_i / d_i^{10} \Big/ \sum_i 1/d_i^{10}$$

- Bottom:

$$f(x) = \frac{2/d_1^{10} + 1/d_2^2}{1/d_1^{10} + 1/d_2^2}$$

Before discussing the adaptation of Shepard's method to our application, classification, we summarize some relevant characteristics of this method. One can write the Shepard formula in the univariate case as follows:

$$f(x) = \sum_{i=1}^{N} \frac{1/d_i^2}{\sum_{j=1}^{N} 1/d_j^2} \cdot f_i = \sum_{i=1}^{N} w_i \cdot f_i .$$

Since the "weight functions" $w_i = w_i(x)$ satisfy (i) $w_i \geq 0$ and (ii) $\sum w_i = 1$, $f(x)$ is a CONVEX COMBINATION of the values $f_1, \ldots, f_N$. Thus,

$$\min\{f_i\} \leq f(x) \leq \max\{f_i\} .$$

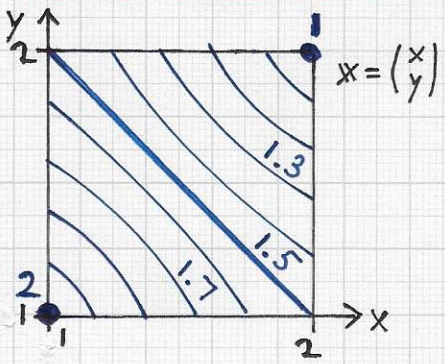
This fact is important especially for our application, as it ensures that "undesirable and wrong" under- and overshoots cannot happen when using Shepard's method. Further, $f'(x_i) = 0$, $i = 0, \ldots, N$, when using squared distance functions d_i^2 in Shepard's method. "f has flat spots/plateaus at x_i."

...

■ OBJECT AND MATERIAL EIGENFUNCTIONS – Cont'd.

• Laplacian eigenfunctions and neural networks: ...

• Sketches of simple bivariate examples:

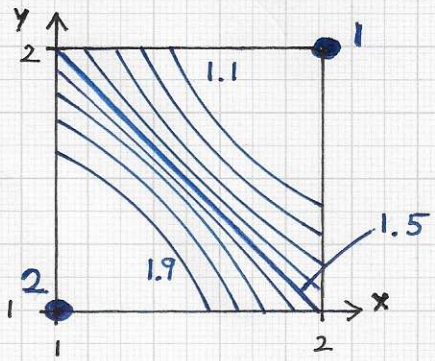


Qualitative behavior of contours of the function

$$f(\mathbf{x}) = \left(\sum_{i=1}^{2} f_i / d_i^2 \right) \Big/ \left(\sum_{i=1}^{2} 1 / d_i^2 \right),$$

where $f(1,1) = 2$ and $f(2,2) = 1$; the squared distance function is

$$d_i^2 = d_i^2(x,y) = (x - x_i)^2 + (y - y_i)^2.$$



Qualitative "cliff behavior" of contours of

$$f(\mathbf{x}) = \left(\sum_{i=1}^{2} f_i / d_i^{10} \right) \Big/ \left(\sum_{i=1}^{2} 1 / d_i^{10} \right),$$

where $f(1,1) = 2$ and $f(2,2) = 1$.

⇒ High values of distance function exponents induce "cliff behavior."

The examples shown in the three figures on the previous page demonstrate that one can use even-integer values (2, 4, 6, ...) as distance function exponents to affect the shape of a Shepard function's graph:

• Increasing the distance function's exponent increases the "width" of the flat spots/plateaus of the Shepard function's graph at the sites $\mathbf{x}_i$. As a consequence, the Shepard function changes its value between sites increasingly abruptly, and "cliffs" result.

• By using different even-integer values as distance function exponents, one can affect the locations of the resulting "cliffs" (third figure).

⇒ INCLUDE EXPONENTS IN FORMULA:

$$f(\mathbf{x}) = \left(\sum_{i=1}^{N} f_i / (d_i^2)^{n_i} \right) \Big/ \left(\sum_{i=1}^{N} 1 / (d_i^2)^{n_i} \right),$$

where $n_i \in \{1, 2, 3, \ldots\}$. The n_i-values can be used as parameters for the optimization of $f(\mathbf{x})$. ...