Higher Order Functions for Kernel Regression

Alexandros Agapitos[1], James McDermott[2], Michael O'Neill[1], Ahmed Kattan[3], and Anthony Brabazon[2]

[1] School of Computer Science and Informatics, University College Dublin, Ireland
[2] School of Business, University College Dublin, Ireland
[3] Um Al Qura University, Dept. of Computer Science, Kingdom of Saudi Arabia

Abstract. Kernel regression is a well-established nonparametric method, in which the target value of a query point is estimated using a weighted average of the surrounding training examples. The weights are typically obtained by applying a distance-based kernel function, which presupposes the existence of a distance measure.This paper investigates the use of Genetic Programming for the evolution of task-specific distance measures as an alternative to Euclidean distance. Results on seven real-world datasets show that the generalisation performance of the proposed system is superior to that of Euclidean-based kernel regression and standard GP.

1 Introduction

One of the oldest and most commonly used nonparametric methods for function estimation is kernel regression [12]. It achieves flexibility in estimating a regression function $F(\mathbf{x})$ over the domain $\mathbb{R}^d$ by fitting a different, *local* model at each query point x_0. This is achieved by using only those observations close to x_0 in such a way that the resulting estimated function $F(\mathbf{x})$ is *smooth* in $\mathbb{R}^d$. The value of $F(x_0)$ is then computed as a weighted average of the function values observed at training inputs.

We note three substantial drawbacks of standard methods for kernel regression. First, they require an *a priori* well-defined distance metric on the input space, which may preclude their usage in datasets where such metrics are not meaningful. For example, the well-known Boston housing dataset [5] contains 13 input features representing completely disparate quantities such as population levels, crime rates, pupil-teacher ratios, etc. Similar difficulties can arise in cases of a mixture of qualitative, ordinal and numerical features. Secondly, a pre-defined distance metric may not be particularly relevant to the regression task at hand. The typical Euclidean distance is calculated on all features defining a point in $\mathbb{R}^d$. In a high-dimensional input space, the distance metric may become dominated by a large number of irrelevant features, as it ascribes to them identical weight to that of the most significant ones. The irrelevant features ideally should not contribute at all to the distance calculation. Thirdly, in cases of input spaces of high-dimensionality, most neighbours of a point can be

very far away, causing bias and degrading the performance of the kernel function. As a simple example [4] (Figure 1.22, page 36), consider a sphere of radius $r = 1$ in a space of D dimensions, and ask what is the fraction of a volume of the sphere that lies between radius $r = 1 - \epsilon$ and $r = 1$. It is shown that as D grows (i.e. $D > 20$), most of the volume of the sphere is concentrated in a thin shell near the surface. This causes most of the points in the feature space to be neighbours, and renders the determination of the kernel width problematic. An additional manifestation of the curse of the dimensionality for kernel regression is that it is impossible to maintain localness (low bias) and a sizeable sample in the neighbourhood (low variance) as D increases, without the training sample size increasing exponentially in D [12].

We propose a novel method to learn a problem-specific distance measure over an input space in which small distances between two vectors imply similar target values, and so we can exploit local interpolation-like techniques to allow us to make predictions of the target variables for new values of the input variables. We employ Genetic Programming (GP) [10] to learn such distance measures by searching the space of programs composed of general-purpose higher-order functions, which allow for implicit iteration over lists of feature values. Typical distance functions, such as Euclidean and other l_p distances, involve *iteration* over the multiple dimensions of the pair of input points. This feature is likely to be useful in new evolved distances also. Including the ability to iterate in our GP language makes it far more general than the constant-time numerical language typical of GP symbolic regression. Success or failure in our work therefore has implications for the broader project of evolutionary synthesis of general computer programs. It also raises the issue of halting, to be addressed by our choice of language.

The reader's guide to the rest of the paper is as follows. Section 2 formalises the method of kernel regression. Section 3 introduces the higher-order functions that will be used in the experiments. Section 4 presents the proposed method and details the experiment design. Section 5 analyses the empirical results, and finally Section 6 wraps up and sketches future research directions.

2 Kernel Regression

In the general function estimation problem, one is given a set of training examples $\{x_i, y_i\}$, $i = \{1, \ldots, N\}$, where y is the response variable and $\mathbf{x} \in \mathbb{R}^d$ is a vector of explanatory variables. The goal is to find a function $F^*(\mathbf{x})$ that maps $\mathbf{x}$ to y, such that over the joint distribution $P(\mathbf{x}, y)$ the expected value of some specified loss function $L(y, F(\mathbf{x}))$ is minimised:

$$F^*(\mathbf{x}) = \arg\min_{F(\mathbf{x})} \mathbb{E}_{x,y}[L(y, F(\mathbf{x}))] \tag{1}$$

Kernel regression or kernel smoothing [12] (page 192) uses the so called Nadaraya-Watson kernel-weighted average to fit a constant locally as follows:

$$F^*(x_0) = \frac{\sum_{i=1}^{N} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{N} K_\lambda(x_0, x_i)} \tag{2}$$

with a *kernel* K_λ, which is typically a probability density function, defined as:

$$K_\lambda(x_0, x) = D\left(\frac{||x - x_0||}{\lambda}\right) \tag{3}$$

where $||\cdot||$ is the Euclidean norm, and λ is the smoothing parameter called the kernel *width*. The smoothing parameter λ determines the width of the local neighbourhood and is usually set by means of cross-validation. Large λ implies lower variance (averages over more observations) but higher bias. Constant values for λ tend to keep the bias of the estimate constant, while the variance is inversely proportionate to the local density.

The function $D(\cdot)$ is typically a positive real-valued function, which decays with increasing distance between x_0 and x. The optimal rate of decay depends on the noisiness and smoothness of the target function, the density of training examples, and the scale of the input features. A wide variety of kernel functions can be found in statistics, see [1].

The application of kernel regression to model a noisy sinusoidal function (green curve) is illustrated in Figure 1. The example uses the Epanechnikov kernel with λ set to 0.2. The fitted function (red curve) is continuous and quite smooth. As we move the target from left to right, points enter in the neighbourhood initially with weight zero, and then their contribution to the weighted average of Equation 2 slowly increases.

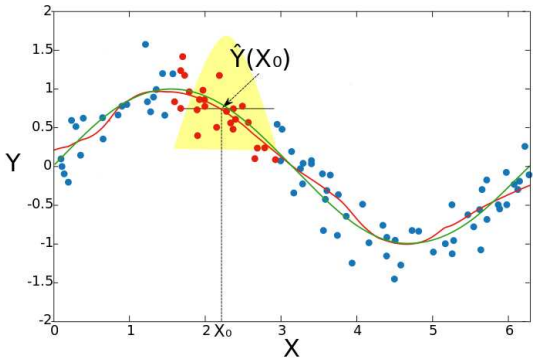

Fig. 1. Example of kernel smoothing. $\hat{Y}(x_0)$ is the fitted constant (calculated using Equation 2), and the red circles indicate those observations contributing to the fit at x_0. The solid yellow region indicates the weights assigned to observations. The green curve is the resulting kernel-weighted average using an Epanechnikov kernel with $\lambda = 0.2$. The figure is adapted from the one found in *http* : *//en.wikipedia.org/wiki/Kernel_smoother*.

.

A general approach for constructing a task-specific distance metric in order to overcome some of the difficulties outlined in the introductory section is to use a *Mahalanobis* metric instead of the Euclidean norm in Equation 5, in which the distance between vectors x and x_0 is defined as:

$$d(x_0, x) = \sqrt{(x - x_0)^T \mathbf{A}(x - x_0)} \tag{4}$$

where $\mathbf{A}$ can be any symmetric positive semi-definite matrix (setting $\mathbf{A}$ to identity results in the standard Euclidean distance). $\mathbf{A}$ is then used to weight different features [12]. Entire coordinates can be downgraded or omitted by imposing appropriate restrictions on $\mathbf{A}$. For example, if $\mathbf{A}$ is diagonal, then we can increase or decrease the influence of individual features x_j by increasing or decreasing $\mathbf{A}_{jj}$. Various methods for adapting $\mathbf{A}$ in the Mahalanobis distance are presented in the studies of [6–8, 11, 13]. An additional method for adapting the feature weights in the calculation of Euclidean distance was originally developed for nearest-neighbour classification [3], and can be directly applied to kernel regression.

3 Higher Order Functions

This paper adopts a different approach to the work cited in the previous section for adapting the metric used in the kernel function. There, the underlying structure of the metric remains fixed. That is, the overall distance between two multi-dimensional input vectors is based on the sum of weighted squared pair-wise distances for each dimension, with the adaptation concerning only the weights. Herein, our aim is to simultaneously learn the underlying computation of similarity as well as the weighting of different features for a particular problem. He have hypothesised that the ability to iterate over the multiple dimensions of an input vector is essential to the evolution of similarity measures, thus we will allow GP to search the space of programs populated by high-level iteration constructs, named *higher-order* functions.

Higher-order functions are functions that take other functions as arguments or produce other functions as results. They are a powerful method of abstraction and re-use, and have been the subject of research in GP for evolving even-n-parity programs [14], music and architectural designs [9], and recursive sorting algorithms [2]. Higher-order functions can be used as general iteration schemata that are bound to a finite-sized list of elements. The iterative behaviour is encapsulated within the body of the function, bypassing the problem of non-halting programs that arises when GP operates on a search space of programs with unbounded iteration/recursion capabilities. Bellow we present the higher-order functions *Reduce*, *Mapcar*, and *Filter*.

`Reduce(list, body).` It is a function that uses a combining operation to recursively process the constituent elements of an argument list, building up a return value. It requires two expression-trees as arguments; the first being of type `list`, whereas the second being the combining operation of type

`double`. It returns a scalar value of type `double`. `Reduce` functions are allowed to nest. The *body* argument is repeatedly evaluated, once per element of the *list* argument. The result returned after each evaluation of the body is stored as the value of a local variable, and when the list is exhausted, the value of this local variable is returned as the value of `Reduce`.
Bellow is an example of a function that computes the sum of `double` elements of `listA`. `res_var_listX` is the result local variable that is returned once the argument list is exhausted. `elt_var_listX` is a local variable that is bound to the current element of the argument list throughout iteration.

```
(Reduce (listA)
   (+  res_var_listA  elt_var_listA))
```

`Mapcar(list, body).` It is a higher-order function that applies a given operation to each element of an argument list, returning a list of results. It requires two expression-trees as arguments, the first being of type `list`, and the second being of type `double`. It returns a list of elements of the same size as the argument list. As an example:

```
(Mapcar (listA)
   (*  elt_var_listA   elt_var_listA))
```

returns a list of elements, where each element is the respective element of `listA` raised to the power of two.

`Filter(list, body).` This higher-order function applies the *body* expression-tree, which is a predicate expression (i.e. returns a boolean value) to each element in the *list* argument to return a list containing items that satisfy the predicate expression. The size of the returned list is less than or equal to the size of the argument list. For example:

```
(Filter (listA)
   (>  elt_var_listA  0))
```

returns a list containing only the positive elements of `listA`.

4 Method

4.1 Wrapper Approach to the Evolution of Distance Measures

The proposed method for evolving a task-specific distance measure is based on a wrapper approach, in which kernel regression is wrapped around an evolved distance measure, with the mean squared error (MSE) that accrues from the regression serving as the fitness of the distance measure.

While kernel regression can be performed with many types of kernel functions, we hereafter focus our research on a particular instance of a logistic kernel that takes the following form:

$$K_\lambda(x_0, x) = D\left(evo(x_0, x)\right) \tag{5}$$

where width λ is absorbed in the evolved distance measure $evo(x_0, x)$, with

$$D(t) = \frac{1}{e^t + 2 + e^{-t}} \tag{6}$$

We decided on the use of the logistic kernel after some initial runs with a range a different kernel functions. Table 1 presents the strongly-typed representation language that was designed for the experiments. The signature of an evolved program is `double measure(list x, list x0)`. The function `zip` is a standard Lisp function that takes two lists and creates a list of pairs, i.e. `zip`({1, 2, 3}, {4, 5, 6}) returns {{1, 4}, {2, 5}, {3, 6}}. In our version, `zip` is defined to return its first argument if not both of its arguments are lists of `double` elements.

Table 1. Representation Language

Function set		
Function	**Argument(s) type**	**Return type**
Reduce	`list, double`	`double`
Mapcar	`list, double`	`list`
Filter	`list, boolean`	`list`
zip	`list, list`	`list`
add, sub, mul	`double, double`	`double`
exp, log, sqrt, sin	`double`	`double`
$\|a-b\|$, $(a-b)^2$	`double, double`	`double`
$\geq$, $<$	`double, double`	`boolean`
Terminal set		
Terminal	**Type**	
x_0, x	`list`	
local vars (used in higher-order funcs)	`double`	
random constants $\in$ [0.0, 1.0]	`double`	
input features (in case of standard GP)	`double`	

4.2 Experiment Design

In this study we use seven real-world datasets obtained from the UCI Machine Learning repository [5], and the Dow Chemical dataset which was the subject of the Symbolic Regression EvoCompetitions event of the 2010 EvoStar conference [4]. Table 3 presents the details of the benchmarks. In all datasets, feature values were standardised to have zero mean and unit variance. Each dataset was randomly split into training and test sets with proportions of 70%-30%. Currently, no validation set is used to select the best-of-run individual.

We perform a comparison between standard Euclidean-based kernel regression using different kernels found in [1], the method of evolutionary-distance-based kernel regression ($KernelGP$) presented in Section 4.1, and standard GP ($StdGP$) that evolves a multi-variate model to predict a response variable. In the case of Euclidean-based kernel regression, width λ is set via 10-fold cross-validation performed on the training set. We cross-validated $2,000$ values for λ

[4] `http://casnew.iti.upv.es/index.php/evocompetitions/105-symregcompetition`

in the range of $\{0.01, \ldots, 20.0\}$ with a step-size of 0.01. Tables 2 and 4 show the setup of the evolutionary systems. Previous research has shown generalisation improvements accruing from the use of small, dynamically-sampled sets of training examples, thus for $KernelGP$ training is based on 20 cases drawn at random from the complete training set in each generation. For $StdGP$ we tried two different variations; $StdGP_{(N=20)}$ trains using 20 random cases dynamically drawn in every generation similarly to $KernelGP$, whereas $StdGP_{(N=all)}$ uses the complete training set.

Table 2. GP systems under comparison

Name	Primitives	Constraints	Max depth	Fitness func.
$KernelGP_1$	Reduce, zip, add, sub, mul, exp, log, sqrt, sin, $\|a-b\|$, $(a-b)^2$, x_0, x, local vars, constants	1) Reduce at the root 2) No nesting of Reduce	6	MSE of kernel regression
$KernelGP_2$	Reduce, Mapcar, Filter, zip, add, sub, mul, exp, log, sqrt, sin, $\|a-b\|$, $(a-b)^2$, x_0, x, local vars, constants	n/a	6	MSE of kernel regression
$StdGP$	add, sub, mul, exp, log, sqrt, sin, features, constants	n/a	10	MSE of evolved program

Since the iterations performed implicitly in the higher-order functions are bounded by the size of the argument lists, we do not have to worry about non-halting programs, however it is reasonable to anticipate programs with deep nesting of higher-order functions. Preliminary runs devoted to configure the parameters of the evolutionary systems suggested that in the case of $KernelGP_2$ configuration, a constraint on the depth of the expression-tree was not enough to keep the run-time within a reasonable frame. We thus decided to impose an *iteration-monitor* for fitness evaluation. This is simply a counter on the number of times a higher-order function is called within an expression-tree. When the limit of this monitor is exceeded, an individual is assigned a high error and its evaluation is abandoned. In these experiments we used the limit of $10,000$ higher-order function calls. To put this number into context, two nested `Reduce` functions would result in $3,249$ `Reduce` function calls in the case of the Dow Chemical dataset of dimensionality 57.

Table 3. Datasets.

Dataset	Training set size	Test set size	Dimensionality
Dow Chemical	747	319	57
Concrete compressive strength	721	309	8
Energy efficiency (heating load)	538	230	8
Parkinsons (motor UPDRS)	4,113	1,762	16
Wine quality (red)	1,120	479	11
Yacht hydrodynamics	216	92	6
Boston housing (price)	355	151	13

In addition to performing regression, we analyse the fitness landscape. To this end, we perform a *perturbation analysis*, in which random walks (using subtree mutation) are taken from a fit individual, plotting the average *Canberra distance* (between targets and predictions) of consecutive neighbours versus the number of mutations. Given a model F and a training example $\{x_i, y_i\}$, the Canberra distance (CD) between prediction $F(x_i)$ and target y_i is given by $|F(x_i) - y_i|/(|F(x_i)| + |y_i|)$, which is implicitly normalised within the [0.0, 1.0] interval. Its average is simply calculated on N examples in the respective training dataset. For this type of analysis, CD was preferred over MSE because it is bounded (a very bad individual can be clearly indicated), and it makes the results of different random walks and different datasets directly comparable.

Table 4. Setup shared by all GP systems

Evolutionary algorithm	elitist (k=1), generational, expression-tree representation
No. of generations	51
Population size	1,000
Tournament size	4
Tree creation	ramped half-and-half (depths of 2 to 4)
Subtree crossover	20% (90% inner-nodes, 10% leaf-nodes)
Subtree mutation	50% (max. depth of subtree: uniform randomly in [1, 4])
Point mutation	30% (probability of a node to be mutated: 10% or 30% or 40%)

5 Results Analysis

We performed 50 independent runs for each evolutionary system. Table 6 compares the test-set MSE. Each best-of-50-runs individual is determined as the one out of 50 final-generation elitists having the lowest training MSE, then its test-set MSE is shown in Table 6. Mean MSE is similarly calculated on 50 final-generation elitists. In every dataset, we used the best-of-50-runs individuals in order to compare the evolutionary methods against the deterministic, Euclidean-based kernel regression. Results suggest that the evolutionary-distance based kernel regression is outperforming the euclidean-based one in all datasets. We also note that the exponential kernels (Gaussian, Logistic) based on the Euclidean distance consistently performed the best.

Comparing Euclidean-based kernel regression against standard GP, we observe that $StdGP_{(N=20)}$ outperforms the former in 1 out of 7 datasets, with the opposite being the case in 4 datasets. On the other hand, $StdGP_{(N=all)}$ outperforms standard kernel regression in 4 out of 7 datasets. Interestingly, a comparison between the two different $StdGP$ setups suggests that the complex evolved models trained on all available data consistently generalised better than those trained on random data samples, a result that is inline with theoretical results in the ML field about matching the model complexity with the amount of training resources available. We identified pathologies (MSE $\geq$ 1,000 and MSE

Boston housing

```
(Reduce
    (zip
       x0
       x
    )
    (+
       (sin
          (sqrt
             (exp
                elt_var_x0
             )
          )
       )
       (+
          (+
             0.577
             res_var
          )
          (absdiff
             elt_var_x
             elt_var_x0
          )
       )
    )
)
```

Parkinsons

```
(Reduce
   (zip
      x0
      x
   )
   (-
      (-
         (+
            (log
               res_var
            )
            res_var
         )
         (absdiff
            elt_var_x0
            elt_var_x
         )
      )
      (absdiff
         elt_var_x0
         elt_var_x
      )
   )
)
```

Concrete

```
(Reduce
    (zip
       x0
       x
    )
    (+
       (sqrdiff
          (-
             -0.962
             (absdiff
                elt_var_x
                elt_var_x0
             )
          )
          (exp
             (absdiff
                elt_var_x0
                elt_var_x
             )
          )
       )
       (*
          res_var
          0.524
       )
    )
)
```

Table 5. Best-of-50-runs simplified distance measures using $KernelGP_1$ configuration.

$= \infty$) in final-generation individuals, and calculated the percentage of these individuals in 50 runs. We observe that the MSE loss function can lead to severe pathologies partly indicative of overfitting in case where complex models are trained on small-sized, even dynamically changed, sets of examples. In the case of $StgGP_{(N=all)}$ no pathologies were observed.

It is interesting to note that despite the fact that $KernelGP$ uses the same training examples sampling configuration as $StdGP_{(N=20)}$, there were no pathologies in the final-generation models. The synergy between the evolved-distance-based kernel and the kernel-weighted average seems to have created a fitness landscape, where search was able to locate models with relatively smooth response surfaces – even in the case where search was guided by a kernel regression MSE estimate based on a limited-sized set of examples. There is a clear superiority in the out-of-sample performance of $KernelGP$ as opposed to $StdGP_{(N=20)}$. Compared against $StdGP_{(N=all)}$, $KernelGP$ generalises better, and in most problems the differences in out-of-sample performance are statistically significant.

Figure 5 presents the simplified best-of-50-runs evolved distances for the Boston, Parkinsons and Concrete datasets, using the $KernelGP_1$ configuration. The evolved solutions are quite neat and comprehensible. All of them are using the $|a-b|$ primitive (shown as `absdiff` in Table 5) operating on the returned list of pairs from the `zip` function. Also, they all rely on some kind of transformation of the `result_var` or `element_var` that is linearly combined with the output of $|a-b|$ to update the `result_var` is each iteration of `Reduce`.

Finally, Figure 2 presents the results of the perturbation analysis for two of the problems. The graphs for the rest of the problems are omitted, but exhibit the same trend. It is evident that higher-order functions craft a neighbourhood in which very bad individuals (avg. CD of approx. 1.0) can be reached using a single mutation step. This is the case in all problems, and it becomes particularly pronounced in the Energy dataset (Figures 2(c), 2(d)), where the performance of a fit individual can be severely degraded during the first step of the random walk. This indicates that while higher-order functions are a powerful addition to the GP paradigm, they can result in very difficult to search program spaces, where gradient quickly diminishes and gradient-based methods are left hopeless. We suspect that it was the implicit parallelism of the evolutionary algorithm that enabled search to counteract this issue to some degree, and allowed for the induction of good-performing individuals. Also, an observation that is consistent across all problems is that on average there is more gradient in the space of programs that is based on all three higher order functions, than there is in the space of programs composed of a single `Reduce` function serving as the root-node of an expression-tree. This can be seen by comparing Figure 2(a) (using `Reduce` at the root) and Figure 2(b) (using all three higher-order functions).

Table 6. Test-set MSE of different regression methods. λ values in parentheses for standard Euclidean-based kernel regression. Statistics for the evolutionary methods were calculated on 50 runs. Std. deviation in parentheses for mean. An asterisk * indicates that the difference in mean values between $KernelGP$ and $StdGP_{N=all}$ are statistically significant at the 5% level (two-tailed Student's t-test, 98 degrees of freedom).

	Boston	**Concrete**	**Dow**	**Energy**	**Parkinsons**	**Wine**	**Yacht**
Euclidean-based kernel regression							
Biweight	0.39 (2.82)	0.36 (2.13)	0.38 (5.61)	0.05 (1.46)	0.98 (0.23)	0.68 (3.10)	0.43 (2.07)
Cosine	0.34 (2.61)	0.38 (2.13)	0.40 (5.61)	0.05 (1.34)	0.98 (0.23)	0.69 (3.09)	0.44 (1.81)
Epanechnikov	0.35 (2.61)	0.39 (2.13)	0.40 (5.61)	0.05 (1.34)	0.98 (0.23)	0.69 (3.08)	0.44 (1.81)
Gaussian	0.27 (0.63)	0.33 (0.58)	0.26 (1.47)	0.05 (0.40)	0.97 (3.75)	0.66 (0.95)	0.22 (0.28)
Logistic	0.27 (0.19)	0.32 (0.17)	0.24 (0.30)	0.05 (0.13)	0.97 (2.08)	0.61 (0.37)	0.19 (0.05)
Triangular	0.41 (2.76)	0.37 (2.13)	0.39 (5.61)	0.05 (1.34)	0.98 (0.23)	0.68 (3.10)	0.41 (1.81)
Tricube	0.39 (2.82)	0.37 (2.30)	0.38 (5.61)	0.05 (1.49)	0.98 (0.23)	0.69 (3.10)	0.44 (2.07)
Triweight	0.38 (3.93)	0.38 (2.76)	0.36 (5.61)	0.05 (1.49)	0.98 (0.23)	0.67 (3.10)	0.40 (2.07)
Evolutionary-distance-based kernel regression							
$KernelGP_1$							
best-of-50-runs	0.24	0.18	0.22	0.002	0.68	0.55	0.007
mean	0.37 (0.14)	*0.24 (0.04)	*0.31(0.08)	*0.009 (0.007)	*0.89 (0.25)	*0.62 (0.07)	*0.01 (0.007)
$KernelGP_2$							
best-of-50-runs	0.26	0.19	0.22	0.003	0.90	0.61	0.007
mean	0.30 (0.06)	0.38 (0.14)	0.44 (0.20)	0.04 (0.07)	0.94 (0.05)	0.79 (0.22)	0.02 (0.04)
Standard GP							
$StdGP_{(N=20)}$							
best-of-50-runs	0.31	0.36	0.60	0.05	0.97	0.72	0.12
mean	0.58 (0.51)	0.77 (1.11)	0.84 (0.18)	0.13 (0.04)	97.25 (577.4)	1.09 (0.47)	0.07 (0.09)
Pathologies:							
MSE $\geq 1,000$	0%	0%	0%	0%	4%	0%	0%
MSE $= \infty$	0%	0%	2%	0%	6%	2%	0%
$StdGP_{(N=all)}$							
best-of-50-runs	0.25	0.28	0.35	0.05	0.88	0.67	0.009
mean	0.45 (0.74)	0.38 (0.04)	0.54 (0.09)	0.08 (0.02)	0.95 (0.01)	0.72 (0.02)	0.06 (0.01)

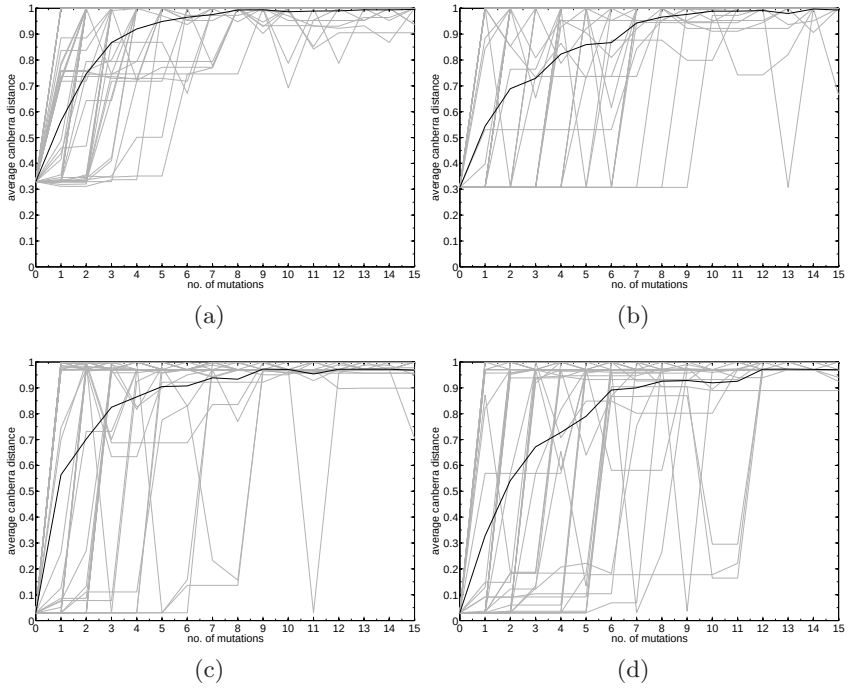

Fig. 2. Random walks. We start from a best-of-50-runs individual and perform 50 different random walks of length 15 – average shown in bold. (a) DowChem $KernelGP_1$; (b) DowChem $KernelGP_2$; (c) Energy $KernelGP_1$; (d) Energy $KernelGP_2$.

6 Conclusion and Future Work

The generalisation performance of interpolation-like function estimation methods can be significantly improved when the distance measure utilised in the kernel function is adapted to the data. We presented a successful, hybrid ML technique that combines kernel regression with the evolutionary learning of the distance measure used in the kernel function. As an additional advantage, the width of the kernel is absorbed in the evolved distance, thus there is no need to set this parameter via cross-validation.

There are a number of avenues for extending this research. First, the use of a validation set to designate a best-of-run individual is believed to further boost the out-of-sample performance of the system. In addition, we plan to perform experiments using the complete set of training examples. Secondly, we note that the configuration of $KernelGP_2$ is of greater generality that the one of $KernelGP_1$. However the performance of the former was somewhat dissatisfying in the sense that it did not outperform the latter. We suspect that this is due

to an insufficient search effort devoted in $KernelGP_2$, and expect that a new experiment based on a more extended search will reveal the true potential of spaces populated by programs composed of these high-level iteration constructs.

The evolution of general computer programs that maintain state and utilise iteration/recursion is still an under-explored area in GP. We wish to stimulate interest in this exciting niche of research.

Acknowledgement

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant Number 08/SRC/FM1389.

References

1. http://en.wikipedia.org/wiki/Kernel(statistics)
2. Agapitos, A., Lucas, S.M.: Evolving efficient recursive sorting algorithms. In: Proceedings of the 2006 IEEE Congress on Evolutionary Computation. pp. 9227–9234. IEEE Press, Vancouver (6-21 Jul 2006)
3. Agapitos, A., O'Neill, M., Brabazon, A.: Adaptive distance metrics for nearest neighbour classification based on genetic programming. In: Krawiec, K., Moraglio, A., Hu, T., Uyar, A.S., Hu, B. (eds.) Proceedings of the 16th European Conference on Genetic Programming, EuroGP 2013. LNCS, vol. 7831, pp. 1–12. Springer Verlag, Vienna, Austria (3-5 Apr 2013)
4. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)
5. Frank, A., Asuncion, A.: UCI machine learning repository (2010), http://archive.ics.uci.edu/ml
6. Goldberger, J., Roweis, S., Hinton, G., Salakhutdinov, R.: Neighbourhood components analysis. In: Advances in Neural Information Processing Systems 17. pp. 513–520. MIT Press (2004)
7. Goutte, C., Larsen, J.: Adaptive metric kernel regression. Journal of VLSI Signal Processing (26), 155–167 (2000)
8. Huang, R., Sun, S.: Kernel regression with sparse metric learning. Journal of Intelligent and Fuzzy Systems 24(4), 775–787 (2013)
9. McDermott, J., Byrne, J., Swafford, J.M., O'Neill, M., Brabazon, A.: Higher-order functions in aesthetic EC encodings. In: 2010 IEEE World Congress on Computational Intelligence. pp. 2816–2823. IEEE Computation Intelligence Society, IEEE Press, Barcelona, Spain (18-23 Jul 2010)
10. Poli, R., Langdon, W.B., McPhee, N.F.: A Field Guide to Genetic Programming. Lulu Enterprises, UK Ltd (2008)
11. Takeda, H., Farsiu, S., Milanfar, P.: Robust kernel regression for restoration and reconstruction of images from sparse, noisy data. In: Proceeding International Conference on Image Processing (ICIP. pp. 1257–1260 (2006)
12. Trevor, H., Robert, T., Jerome, F.: The Elements of Statistical Learning. Springer, 2nd edn. (2009)
13. Weinberger, K.Q., Tesauro, G.: Metric learning for kernel regression. In: Eleventh international conference on artificial intelligence and statistics. pp. 608–615 (2007)
14. Yu, T.: Hierachical processing for evolving recursive and modular programs using higher order functions and lambda abstractions. Genetic Programming and Evolvable Machines 2(4), 345–380 (Dec 2001)