Adaptive Distance Metrics for Nearest Neighbour Classification Based on Genetic Programming

Alexandros Agapitos, Michael O'Neill, and Anthony Brabazon

Financial Mathematics and Computation Research Cluster
Complex and Adaptive Systems Laboratory
University College Dublin, Ireland
{alexandros.agapitos,m.oneill,anthony.brabazon}@ucd.ie

Abstract. Nearest Neighbour (NN) classification is a widely-used, effective method for both binary and multi-class problems. It relies on the assumption that class conditional probabilities are locally constant. However, this assumption becomes invalid in high dimensions, and severe bias can be introduced, which degrades the performance of the method. The employment of a locally adaptive distance metric becomes crucial in order to keep class conditional probabilities approximately uniform, whereby better classification performance can be attained. This paper presents a locally adaptive distance metric for NN classification based on a supervised learning algorithm (Genetic Programming) that learns a vector of feature weights for the features composing an instance query. Using a weighted Euclidean distance metric, this has the effect of adaptive neighbourhood shapes to query locations, stretching the neighbourhood along the directions for which the class conditional probabilities don't change much. Initial empirical results on a set of real-world classification datasets showed that the proposed method enhances the generalisation performance of standard NN algorithm, and that it is a competent method for pattern classification as compared to other learning algorithms.

1 Introduction

In a classification problem, we are given C classes and N training observations. Each training observation x is usually a vector of d features $x = (x_1, \ldots, x_d) \in R^d$ along with the known class labels $y \in \{1, 2, \ldots, C\}$. The task is to predict the class label of a given query instance. The k *Nearest Neighbour* (kNN) classification technique, a popular *instance-based learning* method [14], was originally proposed by Fix and Hodges in 1951 [6]. It determines the k nearest neighbours ("closeness" is usually defined in terms of a distance metric on the Euclidean space) of instance query q, and then predicts the class label of q as the most frequent one occurring among the k neighbours. In contrast to learning methods that induce a function approximation designed to perform well in the entire

K. Krawiec et al. (Eds.): EuroGP 2013, LNCS 7831, pp. 1–12, 2013.

instance space, the kNN method simply stores the training examples (*memory-based* classification); generalisation beyond these examples is postponed until a new instance must be classified.

An important issue that hinders the application of kNN to high-dimensional datasets is the learning algorithm's *inductive bias* – the set of assumptions that a learner uses to predict outputs given inputs that it has not encountered [14] – which assumes that the class conditional probabilities are roughly locally constant, that is, the classification of an instance query q will be most similar to the classification of other instances that are nearby in Euclidean space. This assumption becomes false in high-dimensional spaces, where the nearest neighbours of a point can be very far away, introducing severe bias in the estimates [18].

The method we are developing in this paper deals with the problem of kNN's inductive bias, and falls into the family of methods that employ locally adaptive metrics in order to maintain the class conditional probabilities approximately uniform in the neighbourhood of an instance query. Genetic programming (GP) is employed to learn a model that outputs a real-valued vector, whose components represent individual feature relevances for single features composing a query pattern. This vector is then transformed into a vector of feature weights allowing for a weighted Euclidean distance metric computation, thus enabling a kNN neighbourhood to adapt its shape in different parts of the feature space. This results in enhanced classification performance. We would like to point out that while there exists a plethora of methods for dealing with the *generalisation* of models induced by GP alone (some studies are found in [1–3, 13, 17, 19, 20], this work focusses on hybridising GP and kNN in an attempt to learn even better-generalising models that exploit the power of both learning algorithms.

The rest of the paper is organised as follows. Section 2 formalises the inefficiency that can arise from kNN's inductive bias, and motivates the need to introduce adaptive distance metrics when forming neighbourhoods. Hence, it outlines previous research efforts towards that goal. Section 3 presents the proposed method for dealing with the problem of locally adaptive distance metrics, outlines the experiment setup, the real-world application datasets, and the learning algorithms used to compare against the proposed method. Section 4 analyses the experimental results, and finally Section 5 draws our conclusions and sketches future work.

2 The Need for Distance Metric Adaptation

Formally, in a kNN classification problem, the learner is presented with N training examples $x \in R^d$, each mapped to a corresponding class label y, $y \in \{1, 2, \ldots, C\}$. It is assumed that there exists an unknown probability distribution $P(x, y)$ that generated the training data. In order to predict the class label of an instance query q, we need to estimate the class posterior probabilities $\{P(c|q)\}_{c=1}^{C}$. kNN methods are based on the assumption that the target function is smooth, meaning that the class posterior probabilities $P(c|q)$ are locally constant [4]. That is: $P(c|q + \delta q) \simeq P(c|q)$, for $\|\delta q\|$ small enough. Then,

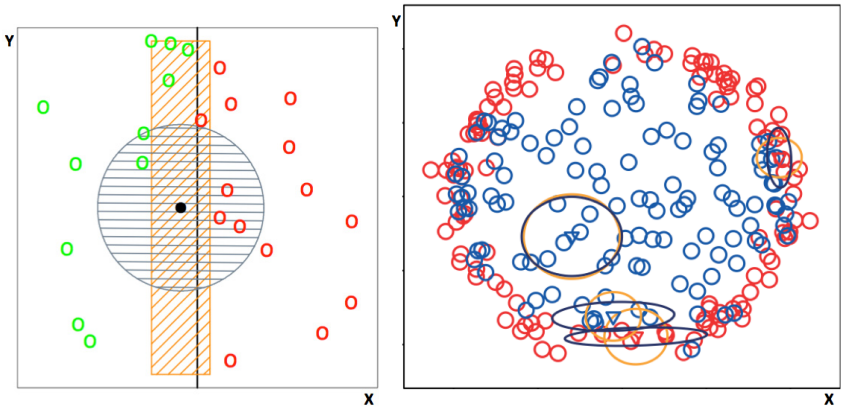
$P(c|q) \simeq ((\sum_{x \in N(q)} P(c|x)/|N(q)|))$, where $N(q)$ is a neighbourhood of q that contains points x that are "close" to q, and $|N(q)|$ denotes the number of points in $N(q)$. This motivates the estimate:

$$\hat{P}(c|q) = \frac{\sum_{i=1}^{N} 1(x_i \in N(q)) 1(y_i = c)}{\sum_{i=1}^{N} 1(x_i \in N(q))} \quad (1)$$

where $1(\cdot)$ in an indicator function that returns 1 if its argument is true, and 0 otherwise.

The assumption of locally uniform class conditional probabilities becomes false when the instance query approaches the class boundaries. We present an example that explains how the choice of a distance measure becomes crucial in determining the outcome of kNN classification. Consider the binary, linearly separable dataset in Figure 1(a), where patterns from each class are represented by the green and yellow circles respectively. Each input pattern resides in a 2-dimensional feature space formed by the horizontal and vertical axes X and Y. The class boundary is represented by the black vertical line and is parallel to the Y axis. The new query to be classified using a 5-NN classifier is shown with the black solid dot. The commonly used Euclidean distance metric assigns equal weight to individual pair-wise feature squared differences, implying that the input space is isotropic or homogenous [14]. This distance metric results in hyper-spherical neighbourhoods – in our 2-dimensional feature space is denoted by the circular strip. We note that the 5-NN neighbourhood has extended into the red-class region, and is dominated by points of the wrong class (3 from the red class and 2 from the green class), thereby causing a misclassification.

If we carefully inspect the dataset in Figure 1(a), we will observe that the class conditional probabilities vary only in the horizontal direction (i.e. a slight move along the horizontal axis may change the class label). In lieu of this knowledge, we should constrict the neighbourhood in the horizontal direction, and elongate it along the vertical direction (direction where the class conditional probabilities do not change), as shown by the vertical strip in the example. This will reduce the bias of the estimate, and leave the variance the same (the neighbourhood is still based on the same number of 5 points). As a result, we observe that the distance metric should not assign equal weights or the same proportions in all directions of the feature space; the weights/proportions during distance computation are query-specific. Capturing such information is of great importance to kNN classification in high-dimensional feature spaces. Figure 1(b) shows examples of different neighbourhood shapes required in different parts of the input space, ranging from circular neighbourhoods, to elliptical ones, and contrasts them against kNN neighbourhoods formed using a standard, unweighted, Euclidean distance metric. Note that the amount of elongation/restriction decays as the instance query moves further away from areas where a decision boundary would lie. The above examples call for locally adapting the distance metric so that the resulting neighbourhood is elongated along the axis direction that provides less class-discrimination information, and is constricted in the opposite case.



(a) (b)

Fig. 1. (a) The vertical line represents the class boundaries between classes red and green. The vertical strip denotes the 5-NN region of a neighbourhood for the query instance (solid black dot), which is constricted along the horizontal axis of the feature space. Figure(a) is adapted from the figure found in page 476 in [18]. (b) Different neighbourhood shapes required to minimise the bias of estimates. The little triangles are the instance query points to be classified. The navy-blue ellipses represent the adaptive 6-NN neighbourhoods, while the orange circles are the standard 6-NN neighbourhoods. Note how the shape varies with instance query locations in the 2-dimensional feature space.

2.1 Previous Work

There has been a variety of studies aiming at locally adapting the distance metric so that a neighbourhood of approximately constant a posteriori probability can be produced. The techniques proposed in [5, 8, 10, 12, 15, 23] are based on estimating feature relevance locally at each instance query. The locally estimated feature relevance leads to a weighted metric for computing the distance between the instance query and the training data. As a result, neighbourhoods get constricted along most relevant dimensions, and elongated along less important ones. Although these methods improve upon the original kNN algorithm, the time-complexity cost of such improvement is high due to the local feature relevance being estimated on the fly with costly procedures whenever a new query is to be classified. This makes it difficult to scale up in large datasets.

An improvement to this time inefficiency is presented in the work of [4] that utilises support vector machines (SVMs) to estimate local feature weighting. The global decision boundary is determined offline, leaving only local refinements to be performed online. The proposed technique offers accuracy improvements over the SVMs alone. Additional work [21] attempted to address the time inefficiency issue in online local feature relevance estimation by introducing a very simple locally adaptive distance metric that normalises the ordinary Euclidean or Manhattan distance from an instance query to each training example by the closest distance between the corresponding training example to training examples of a different class. Results showed comparable performance to SVMs.

In addition to the works for determining local distance metrics, there has been considerable research interest in directly learning distance metrics from training examples. The work of [9] proposed a method for learning a Mahalanobis distance measure by directly maximising a stochastic variant of the leave-one-out kNN classification performance on the training data. In [22] the authors developed a method for inducing a Mahalanobis distance metric using semidefinite programming. Both of these methods induce a global distance metric that is employed in every kNN application irrespective of the location of the instance query.

3 Methods

3.1 Supervised Learning of Local Feature Weights

The method we are proposing revolves around the general notion of adapting the shape of the neighbourhood via the computation of a weighted Euclidean distance metric. As discussed in Section 2, a "closeness" criterion that is based on a weighted Euclidean distance metric computation has the effect of stretching/elongating the axis of the feature space. The proposed technique has the potential of scaling up to large datasets, by learning offline a model that outputs a real-valued vector, whose components represent individual feature relevances for every feature describing a query pattern. These relevance values can then be transformed to weights associated with each pair-wise squared feature-value difference in a weighted Euclidean distance computation. Note that the technique is query-based because the learned model outputs a vector of feature relevances for a particular instance query.

Formally, assume that we want to classify patterns defined in a d-dimensional feature space. Each pattern x is a d-dimensional vector of real-valued features, that is, $x = (x_1, \ldots, x_d)$. For every pattern there is an associated class label y, $y \in \{1, 2, \ldots, C\}$. The learning task is to approximate a function $h(x)$ that maps an input vector representing a pattern into an output vector of feature relevances denoted as $x' = (x'_1, \ldots, x'_d)$ driven by an error measure that concerns the classification accuracy as described below. Using the output vector $(x'_1, \ldots, x'_d)$ from $h(x)$, a measure of relative relevance can be given using the following *exponential weighting scheme*:

$$w_i(x) = \frac{exp(x'_i)}{\sum_{i=1}^{d} exp(x'_i)} \tag{2}$$

We follow [4] and adopt the exponential weighting scheme as it has been shown to be more sensitive in local feature relevance, and in general results in better performance improvement. The weights from Equation 2 can then be associated with features in a weighted Euclidean distance computation:

$$D(x, y) = \sqrt{\sum_{i=1}^{d} w_i (x_i - y_i)^2} \tag{3}$$

This adaptive distance metric can then be used in the kNN algorithm to form a neighbourhood around query pattern x, and classify it accordingly. The learning algorithm needs to induce a model that uncovers the relationship between an output vector of feature relevances $x' = (x'_1, \ldots, x'_d)$ and the classification accuracy (defined as the number of correct classifications divided by the number of examples in a learning set) of the kNN algorithm that employs the adaptive distance metric accruing from the use of $x' = (x'_1, \ldots, x'_d)$. The goal is to learn to output feature relevance vectors x' that result in high classification accuracy. In summary, once a model for assigning feature weights has been learned, the proposed system is a two-layer classifier: given input x, in the first layer we use the learned model to induce feature weights for x, and in the second layer we invoke the standard kNN classifier that employs the weighted Euclidean distance.

3.2 Multiple-Output Program Representation for GP

We used a supervised learning algorithm, Genetic Programming (GP) [16], to learn such a model. The model needs to output a vector of feature relevances, and for that we used a program representation that was introduced in [24] by the name of a *modi expression-tree*. A *modi* program representation can simulate the effect of a *directed acyclic graph*, and consists of two main parts: (a) an expression-tree, and (b) an associated vector for holding outputs, as shown in Figure 2. Similar to standard GP, a *modi* tree has function nodes representing operations (i.e. arithmetic, conditionals, trigonometry), and terminal nodes representing variables and constants. However, unlike the standard expression-tree structure, which outputs a single value through the root, a *modi* program utilises its output vector, hence producing multiple values, each of which corresponds to a single feature relevance in our case. The two parts of a *modi* program are connected through some special function nodes, called *modi* nodes (grey nodes in Figure 2). A *modi* node has two roles: (1) it updates an element in the output vector that the node is pre-associated with, by adding its node value to the value of the vector element; (2) it passes the value of its right child node to its parent node, so the expression-tree structure can be preserved.

The output vector is in effect an array of memory locations where *modi* nodes are allowed to write into. Figure 2 shows what happens when an example *modi* program is evaluated. Before the evaluation starts, the output vector's elements are all initialised with ones. During the evaluation, each non-*modi* node passes its value to its parent, exactly the same way as in standard GP. On the other hand, each *modi* node firstly uses its node value to update the output vector (shown as curved solid arrows), and then passes on the value of its right child to its parent node (shown as dashed arrows). The side-effect of program evaluation is the update of the output vector – we are not concerned with the value returned at the root of the tree. The value of each output vector's element corresponds to a pattern feature's relevance, so starting from the value of one, the higher a value at the end of the program evaluation procedure, the higher the feature's

relevance for a particular input pattern. Once the values of the output vector are set, the exponential weighting scheme of Equation 2 is used to transform each vector element into a weight that will be subsequently used in a weighted Euclidean distance.

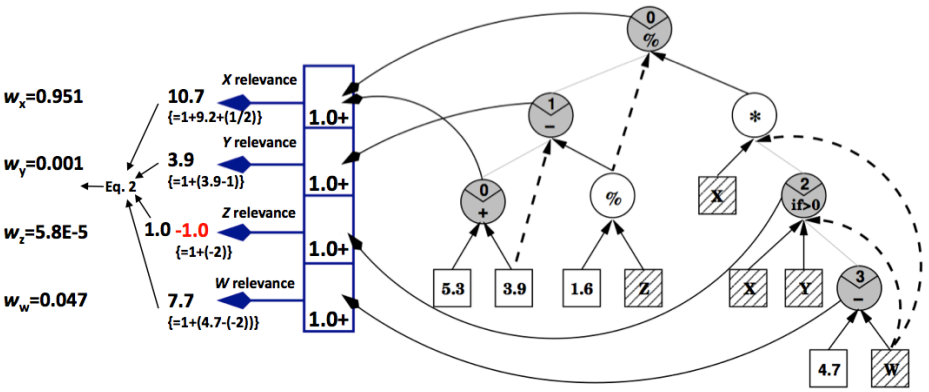


Fig. 2. Example illustration of the way the output vector, representing feature relevances, is updated through the evaluation of a *modi* program, and how this is transformed into a set of feature weights w that will be used in the weighted Euclidean distance computation (Equation 3) for determining the neighbourhood during kNN classification. Here we are considering a pattern with four features $[X, Y, Z, W] = [-1.0, 2.0, 1.6, -2.0]$. The output vector allocates a cell index for each of the four features; these are indices 0, 1, 2, 3 for features X, Y, Z, W respectively. Prior to program evaluation, all vector elements are initialised to 1.0. Feeding the input vector $[-1.0, 2.0, 1.6, -2.0]$ into the *modi* program produces the output vector [10.7, 3.9, 1.0, 7.7]. Note that the value of the third vector element resulted in a negative number (-1.0 shown in red font), and has been set back to the lower bound of 1.0. Invoking Equation 2 using the output vector we get a vector of weights w=[0.951, 0.001, 5.8E-5, 0.047] for features X, Y, Z, W respectively. (The figure is adapted from the figure found in page 4 of [24]).

The detailed method for initialising *modi* trees can be found in [24]. It is worth noting that for the case of intermediate nodes, the probability of a node being set to a *modi* node is governed by a parameter $\mu \in [0, 1]$, the *modi rate*. In addition, we are constraining the values that can be held by vector elements within the range of [1.0, 100.0]. During program execution, vector values are incremented by the value returned from a *modi* node, and this can result in certain vector cell indices being assigned very large values – a situation that can degrade the performance of the exponential weighting scheme by zeroing certain feature weights. A very simple check is employed that sets the value of a vector cell back to the lower bound of 1.0 if the last update resulted in a value that was less than the lower bound, and sets the value to the upper bound of 100.0 if the last update resulted in a value that was bigger than the upper bound.

3.3 Experiment Design

In the experiments we used eight real-world datasets (Table 2) obtained from the UCI Machine Learning repository [7]. These were carefully picked to test our method in problems with high-dimensional input spaces. In all datasets, input values were standardised to have zero mean and unit variance. For each dataset we compared the performance of our system *AdaptiveKNN* against the performance of other learning algorithms via stratified 10-fold or 5-fold cross validation (in case data were limited for a particular dataset – see Table 2).

Table 1. GP system setup

EA used in GP system	elitist, generational, *modi* expression-tree representation
***modi* rate**	0.4
Function set	$+$, $-$, $*$, % (protected), sin, cos, e^x, log, $sqrt$
Terminal set	feature values, 10 random constants in [0.0, 1.0]
No. of generations	51
Population size	100
Tournament size	2
Tree creation	ramped half-and-half (depths of 2 to 6)
Max. tree depth	20
Subtree crossover	30% (90% inner nodes, 10% leaf-nodes)
Subtree mutation	40%
Point mutation	30%
Fitness function	classification accuracy

Table 2. UCI Machine Learning Datasets

Dataset	Size	Classes	Input dimensionality	Cross-validation folds
Australian credit appr. (Statlog)	690	2	14	10
Sonar (Mines vs. Rocks)	208	2	60	10
Ionosphere	351	2	33	10
Vehicle Silhouettes (Statlog)	946	4	18	10
Heart (Stallog)	270	2	13	10
Hepatitis	155	2	19	5
Vote	435	2	16	10
Glass	214	7	9	5

The first stage of applying AdaptiveKNN to classification consists of learning a model of feature weights using GP. For each test fold we treat the remaining folds as our learning set. This learning set is further divided into two disjoint sets of *training* and *validation* with proportions of 80% and 20% respectively. The training set is used to fit the model, while the validation set is used for model selection at the end of an evolutionary run. In the second stage, after learning the model, we can assess the generalisation error using the instance queries of the test fold, and using the data in the remaining folds as a *memory* in the standard kNN methodology. Table 1 summarises the setup of the GP system.

We contrasted the performance of AdaptiveKNN against several classification algorithms implemented in the WEKA software [11]:

1. kNN (StdKNN) using the standard Euclidean distance metric. Value of parameter k for number of neighbours was determined via cross validation.

2. SVM with Radial Basis Function kernel (SVM-RBF) trained with sequential minimal optimisation. Values of parameters γ in $K(x,c) = e^{-\gamma\|x-c\|^2}$, and c for soft-margin were determined via cross validation.
3. SVM with Polynomial kernel (SVM-POLY) trained with sequential minimal optimisation. Values of parameters c for soft-margin, and n for polynomial order were determined via cross validation.
4. Naive Bayesian classifier (NaiveBayes).
5. Gaussian Radial Basis Function Network (RBFN). Values of parameters σ^2 in the Gaussian kernel, and k in k-means clustering were determined via cross validation.
6. Feed forward multilayer perceptron (MLP) trained with back-propagation. Network structure determined via cross validation.
7. C4.5 decision tree method (with post-pruning).
8. Classification and Regression Tree (CART) method (with post-pruning).

4 Results

For training the AdaptiveKNN, we performed 30 independent cross-validated runs with each dataset in order to account for the stochastic nature of the GP learning algorithm. The same number of runs were performed for MLP (after parameter tuning) that also exhibits a stochastic element. Thus, is order to calculate cross-validated performance in Table 3, we used the best models out of 30 models learned for each fold, and then average their test-fold performances. For the remaining of the learning algorithms, we first performed parameter tuning and then reported their cross-validated accuracies. Table 3 shows that AdaptiveKNN achieves the best performance in five out of eight datasets. In two cases (Ionosphere and Sonar datasets) it obtained the second best performance following the SVM-RBF. However, in the case of Vehicles dataset, AdaptiveKNN achieved the lowest performance as compared to SVMs and MLP, and it was only comparable with the tree-based methods C4.5 and CART. Looking at the generalisation performance enhancement that AdaptiveKNN offers over StdKNN, we found that this reaches the level of 13.6% (averaged among datasets), with the lowest percentage increases of 3% and 1% obtained for Australian credit and Vehicles datasets respectively. Finally, Figure 3 contrasts the cross-validated

Table 3. Cross-validated Classification Accuracies

	Australian credit	**Sonar**	**Ionosphere**	**Vehicles**	**Heart**	**Hepatitis**	**Vote**	**Glass**
AdaptiveKNN	**89.1**	88.6	94.5	73.4	**88.5**	**97.8**	**99.2**	**78.3**
StdKNN	86.5	63.6	84.0	72.3	84.4	84.4	92.9	61.7
NaiveBayes	77.1	67.8	82.6	44.7	83.3	87.5	92.7	49.5
RBFN	85.8	87.5	93.7	71.5	84.1	92.5	97.0	70.1
MLP	84.9	82.2	91.4	82.5	78.1	81.2	94.7	67.3
SVM-POLY	86.4	84.6	91.7	84.5	84.8	86.2	97.0	71.5
SVM-RBF	85.5	**89.4**	**94.8**	**84.9**	83.7	86.2	97.0	70.1
C4.5	85.4	71.1	91.4	72.6	76.7	86.2	96.5	67.3
CART	85.6	71.1	89.7	69.4	78.5	82.5	97.0	70.6

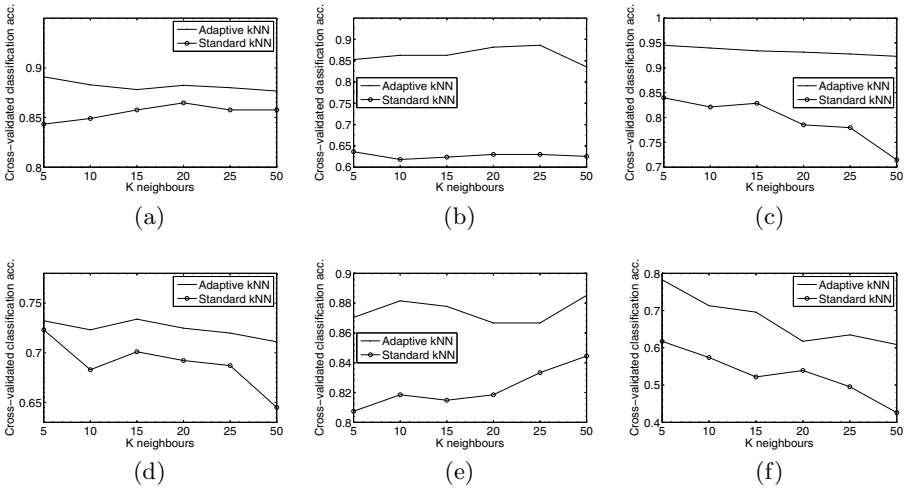


Fig. 3. Cross-validated classification accuracies at different values of k neighbours. (a) Australian credit; (b) Sonar; (c) Ionosphere; (d) Vehicles; (e) Heart; (f) Glass.

classification accuracies at different values of k neighbours for AdaptiveKNN and StdKNN in a sample of datasets. We observe that AdaptiveKNN performs better that StdKNN for all values of k considered.

5 Conclusion

In this work we combined (a) kNN, an instance-based learning algorithm that constructs a local approximation to the target function which then applies to the neighbourhood of each individual query instance, with (b) GP, a very powerful global (i.e. fits a model to the entire instance space) function approximator that is able to learn local relative feature relevances from examples. Transforming these into adaptive distance metrics for use with kNN allows a complex target function to be described as a collection of less complex approximations that are locally tuned to achieve better classification performance.

While there is a cost associated with effective training of GP models (i.e. evolutionary algorithm's parameter tuning through cross-validation, actual cost of performing a significant amount of runs to account for GP's stochastic nature), this process is performed offline once (in contrast to other locally adaptive algorithms that require a considerable amount of online computation), and subsequently allows for a time-efficient computation of local weights. This enhances scalability to large datasets.

Initial empirical results on a collection of real-world datasets showed that (a) the gain in performance over the simple kNN method outweighs this extra cost of offline model learning, and that (b) the proposed method is competent in pattern classification as opposed to other learning algorithms.

There are several avenues for further development of AdaptiveKNN. First, we are planing to compare it against other locally adaptive kNN methods found in literature. Our GP system uses a program representation that hasn't received much attention from the GP community – it falls under the general category of programs with side-effects. We are currently working on optimising two crucial aspects of the system (i.e. *modi* rate, variation operators tailored to this program presentation).

On a more general note, the local feature relevances implicitly touched on the issue of *feature selection*, which can be essentially performed by zeroing certain feature weighs. Adaptive distance metrics for kNN classifiers consist an approach to ameliorate the problem arising from the curse of dimensionality by performing local dimensionality reduction. We plan to investigate this in our future 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. Agapitos, A., Brabazon, A., O'Neill, M.: Controlling Overfitting in Symbolic Regression Based on a Bias/Variance Error Decomposition. In: Coello Coello, C.A., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 438–447. Springer, Heidelberg (2012)
2. Agapitos, A., O'Neill, M., Brabazon, A.: Evolutionary Learning of Technical Trading Rules without Data-Mining Bias. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI, Part I. LNCS, vol. 6238, pp. 294–303. Springer, Heidelberg (2010)
3. Agapitos, A., O'Neill, M., Brabazon, A., Theodoridis, T.: Maximum Margin Decision Surfaces for Increased Generalisation in Evolutionary Decision Tree Learning. In: Silva, S., Foster, J.A., Nicolau, M., Machado, P., Giacobini, M. (eds.) EuroGP 2011. LNCS, vol. 6621, pp. 61–72. Springer, Heidelberg (2011)
4. Domeniconi, C., Gunopulos, D., Peng, J.: Large margin nearest neighbor classifiers. IEEE Transactions on Neural Networks 16(4), 899–909 (2005)
5. Domeniconi, C., Peng, J., Gunopulos, D.: Locally adaptive metric nearest-neighbor classification. IEEE Transactions on Pattern Analysis and Machine Intelligence 24(9), 1281–1285 (2002)
6. Fix, E., Hodges Jr., J.L.: Discriminatory analysis. nonparametric discrimination: Consistency properties. International Statistical Review 57(3), 238–247 (1989)
7. Frank, A., Asuncion, A.: UCI machine learning repository (2010), `http://archive.ics.uci.edu/ml`
8. Friedman, J.H.: Flexible metric nearest neighbour classification. Tech. rep., Department of Statistics. Stanford University (1994)
9. 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)
10. Guo, R., Chakraborty, S.: Bayesian adaptive nearest neighbor. Stat. Anal. Data Min. 3(2), 92–105 (2010)

11. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The weka data mining software: An update. SIGKDD Explorations 11(1) (2009)
12. Hastie, T., Tibshirani, R.: Discriminant adaptive nearest neighbor classification. IEEE Trans. Pattern Anal. Mach. Intell. 18(6), 607–616 (1996)
13. Kattan, A., Agapitos, A., Poli, R.: Unsupervised Problem Decomposition Using Genetic Programming. In: Esparcia-Alcázar, A.I., Ekárt, A., Silva, S., Dignum, S., Uyar, A.Ş. (eds.) EuroGP 2010. LNCS, vol. 6021, pp. 122–133. Springer, Heidelberg (2010)
14. Mitchell, T.: Machine Learning. McGraw-Hill (1997)
15. Peng, J., Heisterkamp, D.R., Dai, H.K.: Lda/svm driven nearest neighbor classification. IEEE Transactions on Neural Networks 14(4), 940–942 (2003)
16. Poli, R., Langdon, W.B., McPhee, N.F.: A Field Guide to Genetic Programming. Lulu Enterprises, UK Ltd (2008)
17. Theodoridis, T., Agapitos, A., Hu, H.: A gaussian groundplan projection area model for evolving probabilistic classifiers. In: Genetic and Evolutionary Computation Conference, GECCO 2011, Dublin, July 12-16. ACM (2011)
18. Trevor, H., Robert, T., Jerome, F.: The Elements of Statistical Learning, 2nd edn. Springer (2009)
19. Tuite, C., Agapitos, A., O'Neill, M., Brabazon, A.: A Preliminary Investigation of Overfitting in Evolutionary Driven Model Induction: Implications for Financial Modelling. In: Di Chio, C., Brabazon, A., Di Caro, G.A., Drechsler, R., Farooq, M., Grahl, J., Greenfield, G., Prins, C., Romero, J., Squillero, G., Tarantino, E., Tettamanzi, A.G.B., Urquhart, N., Uyar, A.Ş. (eds.) EvoApplications 2011, Part II. LNCS, vol. 6625, pp. 120–130. Springer, Heidelberg (2011)
20. Tuite, C., Agapitos, A., O'Neill, M., Brabazon, A.: Early stopping criteria to counteract overfitting in genetic programming. In: Genetic and Evolutionary Computation Conference, GECCO 2011, Dublin, July 12-16. ACM (2011)
21. Wang, J., Neskovic, P., Cooper, L.N.: Improving nearest neighbor rule with a simple adaptive distance measure. Pattern Recogn. Lett. 28(2), 207–213 (2007)
22. Weinberger, K.Q., Saul, L.K.: Distance metric learning for large margin nearest neighbor classification. J. Mach. Learn. Res. 10, 207–244 (2009)
23. Zhang, G.-J., Du, J.-X., Huang, D.-S., Lok, T.-M., Lyu, M.R.: Adaptive Nearest Neighbor Classifier Based on Supervised Ellipsoid Clustering. In: Wang, L., Jiao, L., Shi, G., Li, X., Liu, J. (eds.) FSKD 2006. LNCS (LNAI), vol. 4223, pp. 582–585. Springer, Heidelberg (2006)
24. Zhang, Y., Zhang, M.: A multiple-output program tree structure in genetic programming. In: Mckay, R.I., Cho, S.B. (eds.) Proceedings of the Second Asian-Pacific Workshop on Genetic Programming, Cairns, Australia, p. 12