Kaizen Programming for Feature Construction for Classification

Vinícius Veloso de Melo and Wolfgang Banzhaf

Abstract A data set for classification is commonly composed of a set of features defining the data space representation and one attribute corresponding to the instances' class. A classification tool has to discover how to separate classes based on features, but the discovery of useful knowledge may be hampered by inadequate or insufficient features. Pre-processing steps for the automatic construction of new high-level features proposed to discover hidden relationships among features and to improve classification quality. Here we present a new tool for high-level feature construction: Kaizen Programming. This tool can construct many complementary/dependent high-level features simultaneously. We show that our approach outperforms related methods on well-known binary-class medical data sets using a decision-tree classifier, achieving greater accuracy and smaller trees.

Keywords Kaizen programming • Genetic programming • Classification • Decision-tree

1 Introduction

The objective of a classification algorithm is to predict the class (label) of a record given the values of its attributes. In order to do that, it employs knowledge obtained from a tagged data set, composed of pre-classified records. The information contained in the attribute set (also known as feature set) and in the labels is used to build a model able to accurately differentiate the classes present in the data. This

V.V. de Melo (✉)
Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada A1B 3X5

Institute of Science and Technology, Federal University of São Paulo – UNIFESP, São Paulo, Brazil
e-mail: vinicius.melo@unifesp.br

W. Banzhaf
Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada A1B 3X5

R. Riolo et al. (eds.), *Genetic Programming Theory and Practice XIII*, Genetic and Evolutionary Computation, DOI 10.1007/978-3-319-34223-8_3

model will then be tested using unseen data which have labels (for quality assurance) and other data without labels (for use the classifier).

The classification process can therefore be seen to consist of two phases. The first phase, which corresponds to model building or training, employs a data set of n records with known labels for each record. The model has to correctly identify the class $(y_i, i = 1, \ldots, n)$ of each record $x_{i,j}, j = 1, \ldots, m$, where m is the number of features. The second phase consists of using this classifier to predict classes of unknown records that were not employed in the training phase. Obviously, to evaluate the classifier's performance in the second phase, the test records must have a known class, which is not used in the prediction but is used for comparison with the class predicted by the model.

The method proposed in this paper aims at an improvement of predictive quality by discovering useful knowledge from data in the pre-processing stage. Such extracted knowledge is inserted into the data set in the form of new attributes and can be used subsequently by the classifier to build new models. This strategy is known as feature construction or feature generation (Liu and Motoda 1998), which can also be employed for dimensionality reduction (Guo et al. 2008).

While many feature construction methods are deterministic (Schölkopf et al. 1997; Nguyen and Rocke 2004; Jolliffe 2005), stochastic approaches have also been proposed (Guo et al. 2008; Neshatian et al. 2012; Wu and Banzhaf 2011). Deterministic methods rely on greedy heuristics that are supposed to work on any kind of data, but have been shown to not always effective (Wolpert and Macready 1997). Stochastic approaches are more flexible in this aspect: They can generate and evaluate non-linear features that would be discarded by deterministic methods. It is easy to see that, by being more rigid, greedy deterministic methods tend to be much faster but less capable of exploring the search-space, while stochastic methods will be slower but may generate better features.

The method reported in this chapter combines a stochastic and a deterministic method into a hybrid method. Its stochastic part performs knowledge extraction to generate high-level features, and its deterministic part builds a classification model on top of that. In this work we aim at a grey-box classifier, which is a human-readable model that may not be fully understandable ("gray" instead of "white") because some formulas in the new features could be complex and opaque, though clearer than results produced by black-box approaches such as Artificial Neural Networks.

In the present contribution we employ a collaborative approach to search for high-quality features. There are many aspects that differentiate our method from others found in the literature that use a team-based approach, see, for instance (Brameier and Banzhaf 2001). Those differences will be explained later. For now, to say that our approach evolves a set of features instead of evolving individual features to be used as an ensemble, as commonly explored in the literature.

In order to have a good ensemble, it is important that all classifiers have high predictive quality. Therefore, it is reasonable to suppose that there is a high chance of having very similar classifiers that do not augment each other. The methodology

of the present study builds a single classifier whose features are likely to be complementary to each other. It is similar in behavior to PCA (Jolliffe 2005), where the features generated show a decreasing degree of variance the data. The difference is, however, that our method discovers non-linear features using arbitrary formulas.

In order to perform an efficient search, our approach is based on the Kaizen methodology (Imai 1986), that will be briefly introduced below. The main idea of Kaizen is the continuous improvement of a process through the PDCA (Plan-Do-Check-Act, Gitlow et al. 1989) cycle, generating a new solution based on the knowledge obtained in previous cycles. This new solution can be divided to conquer, allowing individual analysis and improvement of each part. Therefore, a solution is actually composed of partial solutions. Our approach, called Kaizen Programming (KP, de Melo 2014) is an implementation of the PDCA cycle. KP can generate a feature set, build and evaluate the model, extract the importance of each feature from the set, and evolve useful attributes to extract high-quality knowledge from the training data.

The rest of this chapter is organized as follows. Section 2 introduces the concept of Feature Construction, Section 3 describes related algorithms for feature constructions that were used for comparison in this work. Section 4 presents Kaizen Programming applied to feature construction. Computational experiments are presented in Sect. 5 with Sect. 6 providing some conclusions.

2 Feature Construction

Feature construction (Liu and Motoda 1998; Guyon et al. 2006; Kantardzic 2011) is a process employed to discover useful knowledge regarding hidden relationships among features in a set of data. The newly constructed features can then be either used alone or to augment the existing data set. When used alone, the new features may be smaller in number than the original feature set, acting as a dimensionality reduction method; thus named Feature Extraction (Liu and Motoda 1998; Guyon et al. 2006; Kantardzic 2011). It is expected that the new features perform a better separation of classes, facilitating the data mining task.

As pointed out by Freitas (2008), when compared to feature selection, the construction of new useful attributes can be a much more difficult task. This can be explained by the fact that feature selection is a binary combination problem (a feature is *selected* or *not selected*, giving 2^k possibilities, where k is the number of features), while construction is a multi-valued combination task because the new feature is a function composition.

Many methods have been proposed in the last decades to perform feature construction for dimensionality reduction. Deterministic approaches such as Principal Component Analysis (PCA, Jolliffe 2005), and Partial Least Squares (PLS, Nguyen and Rocke 2004) are usually fast and able to find useful features. However, these techniques do not search for a global optimum. Stochastic techniques such as evolutionary algorithms have been investigated to that end. As examples of distinct

methods from the area of Genetic Programming (Banzhaf et al. 1998) we refer to Smith and Bull (2005), Gavrilis et al. (2008), and Drozdz and Kwasnicka (2010).

After generating a feature one must measure its quality. As we are aiming to achieve the best prediction quality, we selected the *wrapper* approach (Miner et al. 2009), where a predictive model (a classifier, for instance) is built using the input feature set, and the prediction results are compared to the expected results. The percentage of correct predictions made on the test set is used as a quality measure. Since the wrapper does not provide feature importance, it is necessary to investigate how the classifier used the features to build the model in. This information about feature importance is then used to efficiently guide the search.

Our models are built using a random sample from the data set (the training set), and tested using a distinct sample from the same data set. Wrappers may require a large computational effort, but their use tends to result in high-quality features that will be tuned to the specific classifier. Finding this best solution implies a global search and a stochastic algorithm is a reasonable choice for this task. The next Section presents some related work.

3 Evolutionary Algorithms for Feature Construction

While there are many using evolutionary algorithms for feature construction, here we briefly introduce only those techniques that are used for comparison in the experimental section.

GPMFC+CART (Neshatian et al. 2012): This technique is a GP-based system for construction of multiple features for classification problems. It uses a filter approach instead of a wrapper, to evaluate the quality of the constructed features during the evolution. The multiple features are sequentially constructed, one by one, for each class of the dataset, maximizing the purity of class intervals. After evolution, the features were tested using the CART decision-tree technique.

MLGP (Wu and Banzhaf 2011): In this contribution, the multilevel selection framework (Wu and Banzhaf 2010) served as inspiration to the development of a multilevel genetic programming approach (MLGP). Multilevel selection tries to encourage cooperation among individuals based on biological group selection theory. The authors developed a cooperation operator in order to build solutions hierarchically. The fitness of a group (or individual) is calculated through direct evaluation, without external classifiers. Therefore, the individual, or group, is used to classify the data.

GP-EM (Guo et al. 2010): This method uses GP to generate a single feature, and an expectation maximization algorithm (EM) to model the new features as a Gaussian mixture (GM). The objective is to evolve features that better separate the classes when modeled as GM. The fitness measure, in this case, considers both the within-class scatter and the between-class scatter values.

GP+C4.5 (Neshatian et al. 2007): In this contribution, classical GP is used to evolve multiple features, and a class-dispersion and entropy-based measure is employed to calculate a feature's quality. A feature is independently constructed for each class in the dataset. Therefore, the distribution of classes in a particular feature must be well separated. After evolving the features, experiments were performed using the well-known C4.5 classifier.

GP+CART (Muharram and Smith 2004): These authors employ two distinct fitness measures to evolve features using GP: Information Gain and Gini Index. The constructed feature is assumed to be a node in a decision-tree, and fitness is calculated using the result of a split in that node. A single feature is evolved in each GP run, and four classifiers are tested on the features. We selected the results obtained by CART to compare to the results herein.

In the next section we describe KP and our proposal for feature construction.

4 Kaizen Programming Applied to Feature Construction

Kaizen Programming (KP), proposed by de Melo (2014), is a novel tool inspired by the concepts of the Kaizen method (Imai 1986). KP is a computational implementation of a Kaizen event with the Plan-Do-Check-Act (PDCA) methodology employed to guide a process continuous improvement. However, KP is an *abstraction* of the main components of PDCA.

Compared with classical GP, KP follows a different method for the automated design of algorithms. KP individuals are not complete solutions, only parts of solutions that have to combine together. As a result, evolution becomes a collaborative approach with the expectation that more than one partial solution is improved to help other partial solutions.

In KP, a team of experts is formed to propose ideas to solve a problem, which then are joined to become a solution. The quality of a solution measures how well it solves the problem, and the quality of an idea quantifies its contribution/importance to the solution. KP first builds a model, and then calculates the importance of each feature. Therefore, different from general GP and other evolutionary algorithms that perform trial-and-error search guided by natural selection, in KP can determine, exactly which parts of the solution should be removed or improved because they were important to the method that built the model. Consequently, the experts contribute by providing better ideas in each cycle. This results in a reduction in bloat, population size, and number of function evaluations. A further difference to other team-based approaches is that the team in KP is a set of agents (data structure + procedures), while for other methods a team is a set of solutions (individuals).

Conceptually, the *knowledge* acquired during the search is shared with the team to improve everyone's ideas. Thus, there is a single set of ideas accessible to all experts, not multiple populations. Not all experts may provide useful contributions all the time that is, the search mechanism does not guarantee that every cycle will

give a better solution. However, it is expected that better ideas are generated over the cycles. A brief explanation of the PDCA cycle is presented next.

REPEAT

> **PLAN**: assuming the current ideas (called *standard*) the team performs a *brainstorming*, and each expert proposes one or more ideas to solve part of the problem;
> **DO**: the standard and new ideas are applied (executed/parsed/evaluated/calculated) to the problem and put together to become a complete solution;
> **CHECK**: evaluate the proposed solution, then each single idea (considering the standard and the new ones) is analyzed and its contribution to solve the problem is measured. Create a new solution using only the important ideas and measure its quality;
> **ACT**: if the solution quality has improved, then the standard is updated, which is presented to the team along with each contribution, improving the knowledge of the problem. Create another kaizen event with a new team if the current one doesn't improve the standard after a certain number of cycles;

WHILE target not achieved

In this chapter, KP is employed to perform high-level feature construction to improve prediction quality of a particular classifier. Various features can be generated at the same time, being improved over PDCA cycles. As opposed to what happens in traditional approaches, in KP those features are dependent on each other, therefore the result is a feature set for a single model, not an ensemble.

4.1 Implementation

Algorithm 1 presents the pseudo-code of the KP method implemented for this contribution. The experts work on a tree-based representation, i.e., as a traditional GP, and may perform only recombination (crossover), only variation (mutation), or both.

The ideas proposed by the experts are non-linear combinations of the original features (formulas) using the terminals and non-terminals defined by the user. The ideas are randomly selected for improvement (there is no tournament) as all of them are supposed to be important. To facilitate implementation, we assumed that the number of experts is the same as the number of features to be constructed, but they are actually distinct parameters. The Expansion Factor to increase the size of the team is a mechanism that may be used when stagnation is detected.

The method selected for building the model was the Classification and Regression Tree algorithm (CART, Breiman et al. 1984). Also, our CART implementation (Pedregosa et al. 2011) provides the Gini Importance (Breiman 2001) for each feature of the dataset, which is used as the importance measure. Thus, one may notice that CART must be used twice: first with all features to measure

Algorithm 1 Pseudo-code of Kaizen Programming for feature construction

1. **Read** the dataset and set n as the number of instances
2. **Set** *CurrentStandardQuality* $\leftarrow$ 0, *MaxStagnated*, *Stagnated* $\leftarrow$ 0, Size of the Team (s_t), number of New Ideas per Expert (*NIE*), Expansion Factor (*EF*), $w \leftarrow s_t * NIE$
3. **Define** the target and set it as not achieved
4. **Generate** s_t initial random ideas as *CurrentStandard*
5. **Apply** the *CurrentStandard* (calculate the results from the expressions) and create the feature set STD_{n,s_t}
6. *BestStandard* $\leftarrow$ *CurrentStandard*
7. *BestStandardQuality* $\leftarrow$ *CurrentStandardQuality* on k-fold cross-validation
8. **Do**
   a. **Generate**, via GP operators, the *TrialIdeas*, which are *NIE* variations (ideas) of the *CurrentStandard* through multiple crossover and mutation. Even the worst idea from *CurrentStandard* might have offspring
   b. **Apply** each new idea, resulting in the $TRIAL_{n,w}$ feature set
   c. **Create** the expanded feature set F_{n,s_t+w} containing $TRIAL_{n,w}$ and STD_{n,s_t}
   d. **Create** new k stratified folds from F to reduce bias in the search
   e. **For** each fold
      i. **Induce** a decision tree via CART
      ii. **Calculate** the array *FoldImportances* as the importance of each feature from F using *Gini Importance*
   f. **End For**
   g. **Set** *TrialImportances* as the average of all *FoldImportances*
   h. *MostImportantTrialIdeas* is the subset of the s_t most important *TrialIdeas* (considering *TrialImportances*)
   i. **Create** $MITI_{n,s_t}$ as a subset of F, and calculate *MostImportantTrialIdeasQuality* using the current k-folds
   j. **If** *MostImportantTrialIdeasQuality* is better than *CurrentStandardQuality* then
      i. *CurrentStandard* $\leftarrow$ *MostImportantTrialIdeas*
      ii. *CurrentStandardQuality* $\leftarrow$ *MostImportantTrialIdeasQuality*
      iii. $STD_{n,s_t} \leftarrow MITI_{n,s_t}$
      iv. **If** *CurrentStandardQuality* is better than *BestStandardQuality* then
         A. *BestStandard* $\leftarrow$ *CurrentStandard*
         B. *BestStandardQuality* $\leftarrow$ *CurrentStandardQuality*
      v. **End If**
   k. **Else**
      i. *Stagnated* $\leftarrow$ *Stagnated* + 1
   l. **End If**
   m. **If** *Stagnated* $>$ *MaxStagnation* then
      i. *Stagnated* $\leftarrow$ 0
      ii. $s_t \leftarrow s_t + \lceil s_t * EF \rceil$ to increase the team of experts' size
      iii. **Generate** s_t initial random ideas as *CurrentStandard*
      iv. **Apply** the *CurrentStandard* (calculate the results from the expressions) and create STD_{n,s_t}
      v. **Calculate** *CurrentStandardQuality* on k-fold cross-validation
   n. **End If**
9. **While** target is not achieved
10. **Return** *BestStandard*, *BestStandardQuality*

their importance, and then with the reduced feature set to measure the actual solution quality. Therefore there is an expansion of the feature set, followed by feature selection. Finally, to reduce the risk of overfitting we used cross-validation in the training.

5 Experiments

This section presents our experiments performed to evaluate KP for classification. KP was tested using publicly available two-class medical datasets from the UCI online repository (Lichman 2013). Some characteristics of the datasets are presented in Table 1. The datasets were chosen after selecting papers from literature that will be used for comparison.

5.1 Pre-processing

Given that KP generates mathematical expressions using features from the dataset, it is necessary to prepare the data. The Weka machine learning tool (Hall et al. 2009) was used to replace missing values with the means from the training data, instead of removing incomplete instances. No other transformation, normalization, or standardization was performed on the data.

5.2 Computational Environment

KP was implemented in the Python programming language (version 2.7.6), using GP from DEAP (Distributed Evolutionary Algorithms in Python) library (version 1.0.1), and scikit-learn library (version 0.14.2) for CART. To evaluate the features discovered by KP, tests were performed using CART in Weka (version 3.6.11) running on Java (version 1.7.0_55) via OpenJDK Runtime Environment (IcedTea version 2.4.7). The experiments were executed on an Intel i7 920 desktop, with 6Gb of RAM, Archbang Linux (kernel version 3.14.5-1), GCC (version 4.9.0 20140521).

Table 1 Summary of the two-class datasets employed in the experiments

Dataset	Continuous attributes	Instances
Breast-w (Winsconsin)	9	699
Diabetes (PIMA)	8	768
Liver-disorders (BUPA)	6	345
Parkinson	22	195

5.3 Organization of the Experiments

During the discovery phase (training), a k-fold stratified cross-validation was performed to calculate both the importance of ideas and the solution quality of selected ideas. It is important to be clear that KP did not evolve features for a specific k-fold configuration, because every time the objective function was called k new stratified folds were generated.

For each dataset of Table 1, KP was run 32 independent times with a different random seed.[1] All runs used the configuration shown in Tables 2, 3, and 4. KP was configured to search for the same number of ideas (10 new features), independently of the number of features in the original dataset. However, not all may be used in the final classifier.

In the expert configuration, GP evolutionary operators, *pdiv*, *plog*, and *psqrt* are protected versions of these operations. $pdiv(a, b)$ returns zero whenever b is zero; $plog(a)$ returns zero whenever the a is zero, and $log(abs(a))$ otherwise; $psqrt(a)$ returns $1e100$ if $a \leq 0$; and $hypot(a, b) = sqrt(a * a + b * b)$.

Since the CART implementation in scikit-learn is not exactly the same as in Weka, it was necessary to use two parameters to achieve greater similarity between the results of different implementations: maximum tree-depth and minimum objects in the leaf node. Features were then tested, in the second phase, with distinct configurations of the CART method (in Weka), which also performed the statistical analysis. This experiment was to evaluate the decision-tree's performance using the original feature set (O), the new feature set (N) discovered by KP, and the

Table 2 KP and CART configuration

Parameter	Value
Initial experts (s_t)	10
Initial ideas generator	GP ramped half-half
Initial ideas max. depth	2
New ideas per expert (*NIE*)	5
Cycles	2000
Stagnation	2.5 % of the cycles
Factor (*EF*) to increase experts	0, disabled
Independent runs	32
Model builder (decision-tree)	CART
CART Max. depth	5
CART Min. instances at a leaf	10
k (folds)	10
Solution quality/fitness	*Accuracy*
Idea importance	*Gini Importance*

[1] Thirty-two runs were performed because it is a multiple of 8, and the runs were done in parallel on a quad-core machine with hyper-threading, so we employed all available processing units.

Table 3 Experts configuration (GP operators)

Parameter	Value
Crossover probability	0.2
Idea combinator/crossover operator	One-point
Mutation probability	1.0
Idea improver/mutation operator	GP subtree replacement
Max. depth	10
Non-terminals	$+, -, \times, pdiv(a,b), plog, psqrt, neg, cos, sin, tan, tanh, square, cube, sigmoid, hypot(a,b), max(a,b), min(a,b), avg2(a,b), avg3(a,b,c)$
Terminals	$x_i, i = 1, \ldots, nf$ (features of the original dataset)

Table 4 CART configuration in Weka for the Test phase

Config. name	Min.Number.Obj	Prune	Use OneSE rule
CART_1	2	No	No
CART_2	2	Yes	No
CART_3	2	Yes	Yes
CART_4	10	No	No
CART_5	10	Yes	No
CART_6	10	Yes	Yes

The other parameters were the default values

combination of new and original feature sets (*NO*). In Weka, CART was configured in six different ways (see Table 4) to verify the influence of the pruning mechanism.

The second analysis is the comparison of the best results obtained by CART experiments versus other feature construction techniques, mainly using Genetic Programming, whose results are reported in the literature. We selected only those that performed ten-fold cross-validation.

5.4 Method of Analysis

The results presented here are only from the test phase. Given that KP was run 32 times on each dataset, we have 32 new feature sets for each of them. A CART decision-tree was induced for each feature set using tenfold cross-validation. Therefore, the original dataset gives 10 results, while each new feature set gives $32 \times 10 = 320$ results.

The evaluated measures were Accuracy, Weighted F-Measure, and Tree size. Accuracy considers the whole dataset, while the Weighted F-Measure is the sum of all F-measures, each weighted according to the number of instances with that particular class label. Tree size is used to evaluate the complexity of the final solution; however, it does not take into consideration the complexity of a feature.

The relevance of this information can be decided by the user when defining the maximum tree-depth used by KP when generating new ideas (features).

In order to verify if there are differences between the feature sets (*O* versus *N*, and *O* versus *NO*), we executed Welch's *t*-test at a significance level $\alpha = 0.05$. If the new features result in statistically different means, a mark '*' is inserted after the standard-deviation in the tables showing the results.

5.5 Evaluation of the Discovered Features

For each dataset investigated here, one has a table with a short descriptive analysis (mean and standard-deviation) of the results for each CART configuration and feature sets, with the significant differences (via Welch's test) marked when necessary. The discussion on the results is as follows.

For the Breast Cancer dataset, one can see the short descriptive analysis in Table 5. Accuracy when using either the New features (*N*) or the combination of New and Original features (*NO*) improved significantly, as shown by the symbol '*'. It is interesting to notice that for both configurations CART_5 and CART_6, the accuracies using *N* and *NO* were identical. This suggests that CART used only the new features from the *NO* dataset; therefore, the Original features (*O*) were not very useful anymore. This hypothesis gets stronger when configurations CART_2, CART_3, and CART_4 are analyzed, in which the mean accuracy of using *N* is bigger than using *NO*. The highest mean accuracy was achieved using a minimum of 2 instances for leaf and the pruning mechanism without the OneSE rule (CART_2).

The second classification quality measure is the Weighted *F*-Measure, which considers the correct classification of each class separately. Again, all CART configurations presented statistically better results when using *N*. For unbalanced datasets, where one class has considerably more instances than the other, these two measures may not have the same statistical interpretation.

The third measure is the tree size. Given that *N* is more representative than *O*, a significant reduction is expected. As shown in the corresponding table, this reduction was bigger than 50 % for CART_1, CART_2, and CART_3, all of them using minimum number of leaves set 2. A relevant comparison can be made between the results of CART_1 and CART_3: there was an increment in the accuracy (from 93.7 to 97.28 %) and a reduction in the tree size (from 41 to 4.41). Consequently, by the results present in this table, the features discovered by KP for the Wisconsin Breast Cancer dataset helped CART in finding better and smaller trees.

Regarding the PIMA diabetes dataset (Table 6), the lowest accuracy occurred using CART_1 on the *O* dataset, while the highest accuracy was obtained with CART_2 on the *N* dataset. All *N* datasets improved over *O* and were also better than all *NO*. Similar behavior is present in the Weighted *F*-Measure results. A posterior application of feature selection on *NO* could help improving the accuracy. With respect to the trees sizes, large reductions can be seen from CART_1 to CART_2, with corresponding increase in Accuracy and Weighted *F*-Measure. However, in

Table 5 Short descriptive analysis (mean and standard-deviation) for the breast-w dataset

Metric	Feat	CART_1	CART_2	CART_3	CART_4	CART_5	CART_6
Accuracy	O	93.70 (2.80)	94.42 (3.53)	94.27 (3.93)	93.99 (4.03)	94.13 (3.96)	93.56 (4.73)
Accuracy	N	97.21 (2.16)*	97.43 (2.03)*	97.28 (2.02)*	97.44 (1.99)*	97.12 (2.66)*	97.04 (2.64)*
Accuracy	NO	97.22 (2.16)*	97.38 (2.07)*	97.25 (2.05)*	97.43 (1.99)*	97.12 (2.66)*	97.04 (2.64)*
W. F-Meas.	O	0.94 (0.028)	0.94 (0.035)	0.94 (0.039)	0.94 (0.041)	0.94 (0.04)	0.94 (0.047)
W. F-Meas.	N	0.97 (0.021)*	0.97 (0.02)*	0.97 (0.02)*	0.97 (0.02)*	0.97 (0.032)*	0.97 (0.032)*
W. F-Meas.	NO	0.97 (0.022)*	0.97 (0.02)*	0.97 (0.02)*	0.97 (0.02)*	0.97 (0.032)*	0.97 (0.032)*
Tree size	O	41.00 (3.53)	16.80 (7.33)	9.00 (5.08)	15.40 (1.26)	5.60 (0.97)	5.20 (1.14)
Tree size	N	18.89 (4.19)*	7.02 (3.58)*	4.41 (2.35)*	8.00 (2.80)*	3.76 (1.58)*	3.18 (0.86)*
Tree size	NO	18.34 (4.22)*	6.74 (3.36)*	4.25 (2.21)*	7.96 (2.78)*	3.76 (1.58)*	3.18 (0.86)*

Table 6 Short descriptive analysis (mean and standard-deviation) for the diabetes dataset

Metric	Feat	CART_1	CART_2	CART_3	CART_4	CART_5	CART_6
Accuracy	O	71.75 (3.39)	75.13 (4.09)	74.36 (4.07)	75.26 (6.20)	75.27 (3.69)	74.09 (4.43)
Accuracy	N	75.54 (4.51)*	79.65 (4.65)*	78.58 (4.73)*	79.48 (4.59)*	79.45 (4.79)*	78.36 (4.79)*
Accuracy	NO	74.62 (4.70)	79.17 (4.89)*	78.19 (4.83)*	79.02 (4.53)*	79.08 (4.67)*	78.05 (4.76)*
W. F-Meas.	O	0.71 (0.035)	0.74 (0.048)	0.73 (0.043)	0.75 (0.061)	0.74 (0.041)	0.73 (0.044)
W. F-Meas.	N	0.75 (0.046)*	0.79 (0.049)*	0.78 (0.051)*	0.79 (0.047)*	0.79 (0.05)*	0.78 (0.052)*
W. F-Meas.	NO	0.75 (0.047)	0.79 (0.052)*	0.78 (0.052)*	0.79 (0.047)*	0.79 (0.05)*	0.77 (0.051)*
Tree size	O	131.80 (11.93)	16.20 (13.54)	4.40 (1.35)	27.80 (5.98)	8.60 (8.37)	4.20 (1.40)
Tree size	N	105.51 (14.74)*	15.27 (7.87)	9.19 (4.67)*	26.66 (5.69)	13.84 (5.33)*	8.57 (4.47)*
Tree size	NO	104.69 (14.45)*	14.36 (8.02)	8.01 (4.51)*	26.18 (5.13)	13.38 (5.81)*	7.50 (4.15)

contrast to what happened to the breast-w dataset, in this case the sizes were bigger when *N* and *NO* were used by CART_3, CART_5, and CART_6. We are still investigating the results to propose a reasonable explanation for this issue.

In the BUPA liver-disorders dataset (Table 7) both Accuracy and Weighted *F*-Measure improved more than $11.94\,\% = (75\,\% - 67\,\%)/67\,\%$ when the discovered features were employed. As will be seen in the comparison with results from the literature, this improvement is very relevant. Finally, the increase in the trees sizes is present for the same CART configurations as in the previous dataset. Nevertheless, smaller trees showed similar or better quality than the bigger ones.

The last dataset contains information of Parkinson's disease. The most noticeable characteristic in Table 8 is that even though both the mean Accuracy and Weighted *F*-Measure improved, the standard-deviations were large, reflecting non-significant differences for configurations CART_4, CART_5, and CART_6, that had as a termination criterion a minimum of 10 instances per leaf. Therefore, it was better to let the tree grow deeper and prune it afterwards, taking the risk of overfitting. This means that, for this dataset, for a significant number of times KP did not discover features capable of reducing entropy in the leaf nodes. A possible explanation is that, as shown in Table 1, this dataset has not only more attributes than the other three datasets, but also fewer instances. Therefore, either a longer run would be necessary or one would need more than 10 features. Nevertheless, the new features led to an increase in mean Accuracy from 87.68 % (best solution using *O*) to 93.85 % (best solution using *N* or *NO*).

5.6 Comparison Against Other Feature Construction Techniques

In this section, KP's results are compared with those from the literature. In order to have a fairer comparison, we selected only works using GP (or a similar technique) to evolve features, with ten-fold cross-validation in the test phase. The comparison is performed with techniques presented in the literature review: GPMFC+CART, MLGP, GP-EM, GP+C4.5, and GP+CART. The results on other methods were taken from the other authors' original works.

From each dataset in the previous section, we have selected the highest mean Accuracy among the CART configurations (see Table 9). Not all datasets used in this work were found in other papers.

As one can see, the features discovered by KP led to more accurate classifiers than all the other feature construction techniques. An important characteristic the number of feature sets created by the techniques. For KP, two feature sets have to be tested at each generation: the first one using the current ideas (features) and the new ideas simultaneously to calculate the importance of each idea; the second one using only the s_t most important ideas to finally calculate the solution quality. Given that KP was run for 2000 cycles, 4000 feature sets were generated in the

Table 7 Short descriptive analysis (mean and standard-deviation) for the liver-disorders dataset

Metric	Feat	CART_1	CART_2	CART_3	CART_4	CART_5	CART_6
Accuracy	O	67.54 (8.32)	67.57 (7.94)	65.85 (8.88)	68.66 (6.37)	65.26 (7.59)	65.25 (8.22)
Accuracy	N	76.23 (7.55)*	78.86 (7.27)*	77.10 (7.24)*	78.80 (7.51)*	77.64 (7.08)*	74.95 (7.15)*
Accuracy	NO	75.71 (7.69)*	78.34 (7.32)*	76.50 (7.24)*	78.28 (7.42)*	77.18 (7.23)*	74.77 (7.18)*
W. F-Meas.	O	0.67 (0.083)	0.66 (0.087)	0.64 (0.089)	0.68 (0.065)	0.64 (0.081)	0.62 (0.10)
W. F-Meas.	N	0.76 (0.076)*	0.79 (0.073)*	0.77 (0.075)*	0.79 (0.076)*	0.77 (0.072)*	0.74 (0.075)*
W. F-Meas.	NO	0.75 (0.077)*	0.78 (0.074)*	0.76 (0.075)*	0.78 (0.075)*	0.77 (0.074)*	0.74 (0.075)*
Tree size	O	79.40 (12.64)	20.80 (14.92)	8.00 (5.83)	22.20 (3.43)	10.00 (4.55)	5.80 (3.16)
Tree size	N	53.48 (9.96)*	17.76 (7.48)	11.43 (4.20)*	18.16 (4.07)*	12.62 (4.17)	8.57 (4.14)
Tree size	NO	50.71 (9.29)*	17.26 (8.02)	10.94 (4.53)	17.62 (3.87)*	12.21 (4.39)	8.18 (4.16)

Table 8 Short descriptive analysis (mean and standard-deviation) for the parkinsons dataset

Metric	Feat	CART_1	CART_2	CART_3	CART_4	CART_5	CART_6
Accuracy	O	87.68 (4.27)	85.66 (4.57)	87.26 (8.23)	86.71 (7.12)	86.21 (8.21)	84.71 (8.54)
Accuracy	N	93.32 (5.34)*	93.85 (5.53)*	93.79 (5.64)*	92.25 (6.68)*	91.20 (7.03)	90.40 (7.11)
Accuracy	NO	92.80 (5.64)*	93.17 (5.85)*	93.03 (6.17)*	91.71 (6.86)	90.60 (7.37)	89.57 (7.64)
W. F-Meas.	O	0.88 (0.041)	0.86 (0.047)	0.87 (0.084)	0.86 (0.073)	0.86 (0.08)	0.84 (0.089)
W. F-Meas.	N	0.93 (0.054)*	0.94 (0.056)*	0.94 (0.059)*	0.92 (0.069)*	0.91 (0.074)	0.90 (0.076)
W. F-Meas.	NO	0.93 (0.057)*	0.93 (0.06)*	0.93 (0.065)*	0.91 (0.072)	0.90 (0.078)	0.89 (0.081)
Tree size	O	17.60 (4.43)	10.80 (3.46)	5.40 (2.80)	8.40 (1.35)	5.80 (2.70)	3.80 (1.69)
Tree size	N	14.20 (2.85)*	8.65 (2.85)	7.26 (2.12)*	7.44 (1.80)	5.83 (1.85)	5.02 (1.82)
Tree size	NO	13.52 (3.25)*	8.54 (2.76)*	7.12 (2.16)*	7.28 (1.69)	5.72 (1.86)	4.74 (1.83)

Table 9 Comparison of mean accuracy among feature extraction techniques that use GP

Dataset	KP+CART	GPMFC+CART	MLGP	GP-EM	GP+C4.5	GP+CART
Breast-w	97.44	96.3**	96.8	–	97.2**	–
Diabetes	79.65	–	71.6	–	75.4	–
Liver-disorders	78.86	67.68	67.5	–	70.4	69.71
Parkinsons	93.85	–	–	93.12	–	–
Feature sets	4000	100,000	600,000	11,200	18,000	60,000

Symbol '**' means a reduction in the number of instances due to missing values, and "–" means Not Available

process. Even though a ten-fold cross-validation approach was used in the training phase, the features were the same for all folds. Because the features in KP are partial solutions, they cannot be evaluated separately.

On the other hand, for the other techniques from Table 9 a single individual is a solution to the problem thus they employed more feature sets. As most techniques evolve a single expression per solution/class, more runs are necessary to have a set of features, while KP can evolve many complementary features at the same time. For them, we calculated the number of feature sets as Population size $\times$ number of generations $\times$ number of features generated. An interesting conjecture is that in order to achieve a performance close to that shown by KP, other techniques may need a more complex formula, while KP may generate a set of smaller/simpler formulas allowing for a posterior feature selection procedure, if desired by the user.

6 Conclusions

This chapter presented Kaizen Programming (KP) as a technique to perform high-level feature construction. KP evolves partial solutions that complement each other to solve a problem, instead of producing individuals that encode complete solutions.

Here, KP employed tree-based evolutionary operators to generate ideas (new features for the dataset) and the CART decision-tree technique for the wrapper approach. The gini impurity used by CART as split criterion is used to calculate the importance of each feature, translating into the importance of each partial solution in KP. The quality of complete solutions was calculated using accuracy in a tenfold stratified cross-validation scheme.

Four widely studied datasets were used to evaluate KP, and tests were performed on six distinct CART configurations. Comparisons among different configurations were made in terms of mean and standard deviation of accuracy, weighted f-measure, and tree-size. A hypothesis test was performed to compare the mean performance when using the new features, and the new and original features together. Results show that the new features with or without the original ones, improved performance and reduced tree-sizes significantly.

The second comparison was against five related approaches from the literature. All those approaches employ genetic programming to construct features from the

original dataset and test them using well-known classifiers. It was found that KP was better than all other approaches, while requiring a fraction of the feature sets generated in other work. KP was not only more accurate, but also much faster.

As future work, a deeper sensitivity analysis will be necessary to verify KP's behavior on distinct configurations in order to be able to differentiate poor from good configurations.

Acknowledgements This paper was supported by the Brazilian Government CNPq (Universal) grant (486950/2013-1) and CAPES (Science without Borders) grant (12180-13-0) to Vinícius Veloso de Melo, and Canada's NSERC Discovery grant RGPIN 283304-2012 to Wolfgang Banzhaf.

References

Banzhaf W, Nordin P, Keller R, Francone F (1998) Genetic programming - an introduction. Morgan Kaufmann, San Francisco

Brameier M, Banzhaf W (2001) Evolving teams of predictors with linear genetic programming. Genet Program Evolvable Mach 2(4):381–407

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Breiman L, Friedman J, Stone C, Olshen R (1984) Classification and regression trees. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, London

de Melo VV (2014) Kaizen programming. In: Proceedings of the 2014 conference on genetic and evolutionary computation (GECCO). ACM, New York, pp 895–902

Drozdz K, Kwasnicka H (2010) Feature set reduction by evolutionary selection and construction. In: Agent and multi-agent systems: technologies and applications. Springer, Berlin, Heidelberg, pp 140–149

Freitas AA (2008) A review of evolutionary algorithms for data mining. In: Soft computing for knowledge discovery and data mining, Springer, Berlin, pp 79–111

Gavrilis D, Tsoulos IG, Dermatas E (2008) Selecting and constructing features using grammatical evolution. Pattern Recogn Lett 29(9):1358–1365. doi:10.1016/j.patrec.2008.02.007. http://www.sciencedirect.com/science/article/B6V15-4S01WDH-4/2/aaff3c40c5eca125dfacb426d88fa177

Gitlow H, Gitlow S, Oppenheim A, Oppenheim R (1989) Tools and methods for the improvement of quality. Irwin series in quantitative analysis for business. Taylor & Francis, London

Guo H, Zhang Q, Nandi AK (2008) Feature extraction and dimensionality reduction by genetic programming based on the fisher criterion. Expert Syst 25(5):444–459

Guo PF, Bhattacharya P, Kharma N (2010) Advances in detecting parkinson's disease. In: Zhang D, Sonka M (eds) Medical biometrics. Lecture notes in computer science, vol 6165. Springer, Berlin, Heidelberg, pp 306–314

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. SIGKDD Explor Newsl 11(1):10–18. doi:10.1145/1656274.1656278. http://doi.acm.org/10.1145/1656274.1656278

Imai M (1986) Kaizen (Ky'zen), the key to Japan's competitive success. McGraw-Hill, New York

Isabelle G, André E, An introduction to feature extraction. In: Guyon I, Gunn S, Nikravesh M, Zadeh LA (eds) Feature extraction: foundations and applications (Studies in Fuzziness and Soft Computing). Springer, Berlin/Heidelberg, pp 1–25. doi:10.1007/978-3-540-35488-8

Jolliffe I (2005) Principal component analysis. Wiley Online Library

Kantardzic M (2011) Data mining: concepts, models, methods, and algorithms. Wiley, New York

Lichman M (2013) UCI machine learning repository. http://archive.ics.uci.edu/ml

Liu H, Motoda H (1998) Feature extraction, construction and selection: a data mining perspective. Springer, Berlin

Miner G, Nisbet R, Elder IVJ (2009) Handbook of statistical analysis and data mining applications. Academic Press, New York

Muharram MA, Smith GD (2004) Evolutionary feature construction using information gain and gini index. In: Genetic programming, Springer, pp 379–388

Neshatian K, Zhang M, Johnston M (2007) Feature construction and dimension reduction using genetic programming. In: AI 2007: advances in artificial intelligence. Springer, Berlin, pp 160–170

Neshatian K, Zhang M, Andreae P (2012) A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming. Trans Evol Comp 16(5):645–661

Nguyen DV, Rocke DM (2004) On partial least squares dimension reduction for microarray-based classification: a simulation study. Comput Stat Data Anal 46(3):407–425

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830

Schölkopf B, Smola A, Müller KR (1997) Kernel principal component analysis. In: Artificial neural networks–ICANN 97. Springer, Berlin, pp 583–588

Smith MG, Bull L (2005) Genetic programming with a genetic algorithm for feature construction and selection. Genet Program Evolvable Mach 6(3):265–281

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82

Wu SX, Banzhaf W (2010) A hierarchical cooperative evolutionary algorithm. In: Proceedings of the 12th annual conference on genetic and evolutionary computation, GECCO '10. ACM, New York, pp 233–240

Wu SX, Banzhaf W (2011) Rethinking multilevel selection in genetic programming. In: Proceedings of the 13th annual conference on genetic and evolutionary computation, Dublin, pp 1403–1410