2015 Brazilian Conference on Intelligent Systems

# Predicting high-performance concrete compressive strength using features constructed by Kaizen Programming

Vinícius Veloso de Melo
Institute of Science and Technology
Federal University of São Paulo
São José dos Campos, São Paulo, Brazil
Email: vinicius.melo@unifesp.br

Wolfgang Banzhaf
Department of Computer Science
Memorial University of Newfoundland
St. John's, NL, A1B 3X5, Canada
Email: banzhaf@mun.ca

***Abstract*—The compressive strength of high-performance concrete (HPC) can be predicted by a nonlinear function of the proportions of its components. However, HPC is a complex material, and finding that nonlinear function is not trivial. Many distinct techniques such as traditional statistical regression methods and machine learning methods have been used to solve this task, reaching considerably different levels of accuracy. In this paper, we employ the recently proposed Kaizen Programming coupled with classical Ordinary Least Squares (OLS) regression to find high-quality nonlinear combinations of the original features, resulting in new sets of features. Those new features are then tested with various regression techniques to perform prediction. Experimental results show that the features constructed by our technique provide significantly better results than the original ones. Moreover, when compared to similar evolutionary approaches, Kaizen Programming builds only a small fraction of the number of prediction models, but reaches similar or better results.**

***Index Terms*—Kaizen Programming, Prediction, Linear regression, High performance concrete, Compressive strength**

## I. Introduction

High-performance concrete (HPC) is a material that has been widely used in various structural applications such as bridges, high buildings, and pavement construction. HPC replaced high-strength concrete [1] in several applications because it presents good workability, high-strength and low permeability, which is directly related to long-term durability [2].

HPC's compressive strength is considered its most important quality, and accurate prediction models are extremely useful in industry as they can save time and costs. Concrete compressive strength (CCS) has been predicted by linear or non-linear regression methods [3], but given its non-linear characteristics, machine-learning methods - mainly artificial neural networks - have been investigated [4], [1]. However, artificial neural networks are known to result in a black-box, meaning that the derived model is hard to understand. Therefore, evolutionary algorithms have been employed to generate models easier to interpret [5], [6], [7], [8].

Two issues regarding those evolutionary algorithms are the large number of models that must be tested to reach good results and the size of these solutions, which can make them uninterpretable. Kaizen Programming (KP) was proposed in [9] and tested on solving benchmark symbolic regression problems functions. Here we decided to test their approach to construct high-level features from the original CCS dataset and compare with the results using the original features employing several well-known regression techniques.

The contribution of this paper is two-fold. The first one is use KP to perform regression on a real-world dataset, showing its computational efficiency in terms of number of models evaluated. The second one is to show its efficacy in finding high-quality features, which are able to improve prediction quality of many well-known regression techniques by providing better input to them.

The rest of the paper is organized as follows. Section II presents related work. Section III introduces the Kaizen Programming technique developed in this work. Experimental results are shown in Section IV. Finally, Section V has the Summary and Conclusions.

## II. Related work

In this section we first present some work related to KP and then introduce related work from the literature that investigated feature construction to improve prediction of HPC compressive strength. Most work is based on evolutionary algorithms such as Genetic Programming (GP, [10]). Unfortunately, the other authors did not use exactly the same dataset, but some important characteristics of the methods will be highlighted.

The team-based approach has been investigated in the literature by several authors [11], [12], [13], but for such techniques a team means a set of solutions, while in KP a team is a set of agents that create solutions. Therefore, a team in those work corresponds to a complete solution in KP.

In a Michigan-style Learning Classifier Systems (LCS, [14]) each individual is a rule generated by a Genetic Algorithm, and the classifier is composed of various individuals of the population. The fitness of each rule is calculated by a reinforcement learning technique that estimates the reward the system would receive if the action of that rule was performed. LCS evolves rules to group the samples in the dataset. On the other hand, the partial solutions evolved by KP are taken as inputs to the method that will actually solve the problem.

In Parisian Genetic Programming [15] a niching mechanism is employed to maintain diversity and to define the set of individuals selected for a global solution. The local fitness of an individual solution is constituted by how much of the problem it solves, while the global fitness is based on how much the combined set of individuals solves the problem. On the other hand, KP may select partial solutions that are not good at solving the problem by themselves, whereas in Parisian GP an individual that solves only a small part of the problem will likely be discarded.

Regarding HPC compressive strength prediction, Baykasoğlu et al. [5] applied Gene Expression Programming (GEP) to construct a single feature of a dataset with 104 instances and six input variables. There is no indication of separation into folds nor into training and test sets. GEP was configured with a population size of 100 solutions and ran for an uncertain number of generations between 3,000 and 20,000, evaluating at least 300,000 prediction models. They reported a Mean Squared Error (MSE) of 4.

Tsai and Lin [8] studied a GP in which every term is weighted by a Genetic Algorithm. The authors used the same dataset as [5], taking 84 examples as training set and 20 as test set. The reported statistics for the test set were $R^2 = 0.957$ and $RMSE = 1.86$, reached after testing 1,000,000 models (5,000 generations and 200 individuals).

Chen and Wang [7] modeled the strength of HPC using Grammatical Evolution combined with a Genetic Algorithm (GEGA). They reported a dataset with 1140 concrete samples with nine input variables, split into 760 examples for training and 380 for testing. GEGA was run for 1,000 generations with a population of 200 solutions, giving a total of 200,000 models. They reported a test RMSE of 9.949.

Castelli et. al [6] used Genetic Programming with geometric semantic genetic operators (GS-GP). Their dataset contains 1028 instances and eight variables, with a split of 70%/30% for training and testing respectively. GS-GP was executed for 2,000 generations with 200 individuals, resulting in approximately 400,000 models. No maximum tree depth limit was imposed during the evolution, meaning that the best solution could be a huge tree. For the test set the authors reported a median $RMSE = 5.926$ over 50 runs.

## III. Kaizen Programming

Kaizen Programming is a recent tool based on the concepts of the Kaizen methodology. KP is a computational implementation of a Kaizen event with Plan-Do-Check-Act (PDCA [16]) methodology to guide the continuous improvement process. As KP is not linked to particular techniques to solve problems, it may also be seen as a methodology or framework.

In PDCA, modifications in a business process are planned, executed, checked, and new actions are taken based on the results. The cycle is repeated until a mission, such as cost reduction, is complete. As every action can be evaluated according to its effectiveness in helping solving the issue, at each cycle the team acquires more knowledge on the problem, meaning more information to avoid bad actions and guide the search towards a better solution.

Three basic modules are necessary to solve problems using KP. For solving regression, KP performs 1) *Feature Construction* to expand the current feature set; 2) *Feature Selection*; and 3) *Model generation*.

The first module contains the experts, which are the agents (data structure + procedures) responsible for proposing the ideas to solve the problem using, for instance, recombination and variation of the current best solution (the standard, which contains various partial solutions). In [9] and in this work, the experts could use only traditional evolutionary procedures, but non-evolutionary methods such as dynamic programming may be employed in a different implementation of KP.

The second module executes the ideas and joins these new partial solutions with those from the current standard into a single set. Then it calculates the importance of each partial solution. The importance measure *must* consider that the partial solutions are not complete, independent solutions. The importance of a partial solution is not how well it solves the problem but, for example, the probability of increasing the quality of the complete solution when a particular partial solution is included into the complete solution.

Finally, the third module is responsible for calculating the quality of the complete solution. A single complete solution is created after selecting the most important partial solutions from the set, based on their importance. Then, the problem is solved and a scoring function returns the solution quality. Once again, the procedure or technique used to solve the problem must be able to use all partial solutions at once, but it will automatically choose how to do that.

To improve efficiency, it is advised that the second and third modules are linked, for example, the method used to solve the problem is also employed to calculate the importances of the partial solutions. This way, the experts will provide better ideas that make sense to that method because it is the importance measurements that are used for guiding the search, not the quality of the complete solution. On the other hand, if those two modules are independent, then the ideas are important to the procedure employed in the second module, but not to the method that solves the problem. Moreover, KP implementations are encouraged to be hybrid, employing high-quality and high-efficiency statistical and machine learning methods.

KP is a collaborative problem-solving approach where the experts have to contribute by providing better ideas at each cycle. When compared to GP, such property produces less bloat and requires smaller population sizes and lower number of function evaluations because it is simpler to evolve several smaller partial solutions than a single big one.

In order to solve a regression task, KP searches for a set of partial solutions that consider the current "knowledge" and improve on it. Given that the initial knowledge is the original set of features of the dataset, a partial solution is an arbitrary expression using one or more original features, resulting in a new combined feature. The final solution generated by KP is therefore a set of new features, that are non-linear recombinations of the original ones.

Here, the experts employ evolutionary operators of a Genetic Programming algorithm to generate their ideas, and Multiple Linear Regression via Ordinary Least Squares (OLS) to build a complete solution (regression model), to generate the predictions, and to calculate the importances. An expert uses recombination, recombination and variation, or only variation. Algorithm 1 presents a high-level version of the KP algorithm used in this work, and the following subsections explain more details.

**Algorithm 1** High-level algorithm of KP with OLS.

```
• Generate $s_t$ initial random ideas as CurrentStandard
• Evaluate CurrentStandard
• BestStandard ← CurrentStandard
• Loop while target is not achieved
    – PLAN: Construct new Features from the ones in the CurrentStandard
      through evolutionary operators. Even the worst idea from CurrentStandard
      might have offspring
    – DO: Execute the ideas and join them with the CurrentStandard
    – CHECK:
        * Build a model (OLS) considering CurrentStandard and the new features,
          and calculate the importance of each feature ($p$-values)
        * Select the $s_t$ most important features
        * Perform cross-validation to build models using the selected features
    – ACT:
        * Update CurrentStandard if the new model is better
        * Update BestStandard if the new model is better
• Return BestStandard, BestStandardQuality
```

## A. PLAN

At the beginning, a set of partial solutions is randomly created and taken as the standard that will be improved in the next cycles. Then, the team of experts propose ideas based on the current standard to be tested.

In regression terms, regressors (independent variables) are used to predict the response (the dependent variable). An idea is then an arbitrary expression that must be calculated into a *new* regressor. As every expert must provide at least one idea, we linked the number of experts (size of the team, $s_t$) to the number of ideas that will be in the standard. As an example, for one regressor ($x$) and three experts one may have three ideas $I_1 = -log(x)$; $I_2 = sin(\sqrt{x})$; $I_3 = -3 + 1/x$.

## B. DO

Here the ideas are converted into the regressors to create the matrix $TRIAL_{n,w}$, where $n$ is the number of observations in the sample dataset used for training, and $w$ is the number of proposed regressors, which must be a multiple of $s_t$ as each expert may propose more than one idea. The new regressors are appended to the current standard $STD_{n,s_t}$ and all of them are used at the same time to build a new model, allowing the identification of the important ones in a *single* iteration. Below is an example of the new features matrix $F_{n,(s_t+w)}$.

$$F = \begin{bmatrix} STD_{1,1} & \dots & STD_{1,s_t} & TRIAL_{1,1} & \dots & TRIAL_{1,w} \\ \vdots & \dots & \vdots & \vdots & \dots & \vdots \\ STD_{n,1} & \dots & STD_{n,s_t} & TRIAL_{n,1} & \dots & TRIAL_{n,w} \end{bmatrix}.$$

## C. CHECK

To identify the important partial solutions, in this work KP uses the well-known Ordinary Least Squares (OLS) to build a multiple linear model using $F$ and the set of expected outputs for the problem ($y$). Using the previous example with $F_1$, $F_2$, and $F_3$, the model is created in the form:

$$\hat{y_i} = \hat{\beta}_0 + \hat{\beta}_1 F_{i,1} + \hat{\beta}_2 F_{i,2} + \hat{\beta}_3 F_{i,3}, \tag{1}$$

where $\hat{y}_i, i = 1, ..., n$ is the calculated output for a specific input and $\hat{\beta}_1$, $\hat{\beta}_2$, and $\hat{\beta}_3$ are the coefficients estimated by OLS. The coefficients are not included in the solution during the search, only in the final solution. The final model (final solution) includes all the estimated parameters and formulas. For instance:

$$\hat{y_i} = -1.93 + 2e\text{-}3(-log(x)) + 982.13(sin(\sqrt{x}) - 57.12(-3 + 1/x). \tag{2}$$

All models generated by KP are linear in the parameters ($\hat{\beta_i}$). However, as the features are non-linear combinations of the original regressors, models generated from $F$ are supposed to approximate any non-linear function. After building the model, the traditional $p\text{-}value$ of each regressor is used to calculate the importance as

$$contribution(F_j) = 1.0 - \begin{cases} 1.0, & if\; p\text{-}value(F_j) = NA \\ 1.0, & if\; p\text{-}value(F_j) > \alpha \\ 1.0, & if\; |\hat{\beta_j}| < \theta \\ p\text{-}value(F_j), & otherwise \end{cases}. \tag{3}$$

Significant $p\text{-}values$ tend to zero, but one wants to maximize contribution. Here, only the significant solutions, limited by $s_t$, are then selected to build a reduced model to be evaluated against the current standard, avoiding exponential growth of the number of regressors. Reducing the number of regressors was not explored in this work, but it is a possibility. However, it was observed that the contribution of all regressors increase over the cycles, meaning that all of them will likely be significant. As the final solution is a linear regression model, the user may easily perform either feature selection or any other applicable procedure.

The solution quality of the reduced model is given by a goodness of fit measure by evaluating prediction error. In this work, the scoring function is the Root Mean Squared Error (RMSE). A model is built on the training set and RMSE is calculated on the validation set, which contains only instances not present in the training set.

## D. Act

Action is taken to update the standard if it has been improved. Therefore, in this work it is simply the selection step to update the solution in a hill-climbing approach, meaning that worse standards are not accepted.

## IV. Experimental results

This section presents experimental results of Kaizen Programming coupled with Ordinary Least Squares to perform feature construction for regression.

## *A. The dataset*

Here we investigate the technique on the HPC data available from the UCI machine learning repository, first presented by Yeh [1]. A description of the dataset is shown in Table I. The dataset contains eight numerical attributes and 1030 examples. There is no indication of missing values.

Table I
CONCRETE STRENGTH DATASET DESCRIPTION.

| Attribute | Unit | Minimum | Maximum | Average |
|---|---|---|---|---|
| Cement | kg/m3 | 102 | 540 | 276.5 |
| Blast-furnace slag | kg/m3 | 0 | 359.4 | 74.27 |
| Fly ash | kg/m3 | 0 | 260 | 62.81 |
| Water | kg/m3 | 121.8 | 247 | 182.98 |
| Super-plasticizer | kg/m3 | 0 | 32.2 | 6.42 |
| Coarse aggregate | kg/m3 | 708 | 1145 | 964.83 |
| Fine aggregate | kg/m3 | 594 | 992.6 | 770.49 |
| Age of testing | Day | 1 | 365 | 44.06 |
| Concrete compressive strength | Mpa | 2.3 | 82.6 | 35.84 |

## *B. Implementation and configuration of the algorithms*

KP was implemented in Python 2.7.6 using DEAP (Distributed Evolutionary Algorithms in Python, [17]), an evolutionary computation framework for rapid prototyping and testing of ideas. DEAP provides the evolutionary operators and data structure required by KP. The statistical parts of KP (the OLS method, $p\text{-}values$, among others) used the *lm* (linear model) function in R programming language. Rpy2 Python package was used to connect R and Python. All issues related to the model building (multicollinearity, singularity, overflow, etc) are simply caught as exceptions and treated as poor quality models/features. KP configuration is shown in Table II.

The experiments were run on a system using Arch Linux kernel 3.18.6-1 as operating system, an Intel i7-920 CPU, and 6Gb DDR3 RAM. A comparison with other techniques was performed on Weka machine learning tool [18] 3.6.11 running on Java OpenJDK Runtime Environment (IcedTea 2.4.7) (ArchLinux build 7.u55_2.4.7-1-x86_64).

## *C. Evaluation*

A descriptive analysis of the results is presented with the best, median, worst, mean, and standard-deviation using the following measures: $R^2$, RMSE, MAE, and computation time. The measurements are performed on 30 independent trials (10-fold cross-validation). For $R^2$, the highest values are the best, while for other measures the lowest values are the best.

We also investigate whether the features constructed by KP could be useful to other regression techniques. To do that, regression methods from Weka were executed with their default configuration to generate models for comparison (except for IBk that was configured with *k*=3 instead of *k*=1). Also, the Linear Regression in Weka is not the traditional one, thus the results are different from those obtained by KP.

To perform a hypothesis test and verify whether the new features improve prediction quality, we selected the best dataset out of 30 (lowest RMSE) generated by each KP configuration (each distinct number of features). Each method from Weka was tested in 10 independent runs of 10-fold cross-validation

Table II
RUN AND EVOLUTIONARY PARAMETERS.

| Parameter | Value |
|---|---|
| Ideas (# of new features) | 1, 2, 3, 5, 10 |
| Initial ideas generator | Ramped half-and-half |
| Iterations | 100 |
| Recombination probability ($RP$) | U(0, 1) |
| Recombination operator | One-point crossover |
| Variation probability | $1 - RP$ |
| Variation operator | 10% Uniform Sub-tree Mutation<br>50% Random Node Replacement<br>10% Random Node Insertion<br>15% Random Leaf Shrink<br>15% ERC Mutation (all constants) |
| Initial tree depth | 2 |
| Maximum tree depth | 5 |
| Non-terminals (functions) | $+, \times, -, /(safe), sin, cos, exp, log(safe), \frac{1.0}{x}, -x, \|x\|, \sqrt{x}(protected), max(x,y), min(x,y), less(x,y), greater(x,y)$ |
| Terminals | $x$, Constants (ERC) are random values from $N(\mu = 0, \sigma = 5)$. |
| Solution quality | average RMSE on 10-fold cross-validation |
| Independent trials | 30 |
| $\alpha$ (for KP Check) | 0.05 |
| $\theta$ (for KP Check) | Skipped, not used |

with the original dataset, and then with only those best feature sets. A paired *t*-test with correction (Weka's default) was subsequently executed to detect statistically significant differences among the means, taking the original dataset as control. It is important to notice that the new features were not used in addition to the original features in the same dataset.

## *D. Results and Discussion*

Table III presents the results of running KP to evolve better features for the concrete dataset investigated in this paper. As one may notice, prediction quality increases with the number of features. As expected, the computation time also increases considerably. This is the biggest drawback of our current implementation. However, it is important to remember that Python is not a high-performance programming language and that this is a proof-of-concept implementation only, not production ready.

Considering the linear regression model, $R^2$ is supposed to increase with the number of features as overfitting gets stronger. However, the results are averages from the test sets on 10-fold cross-validation. For the three remaining measures, the lower the value the better. As occurred to $R^2$, more complex models resulted in lower prediction error (RMSE and MAE). Therefore, it is noticeable that the model's generalization ability was directly proportional to its complexity, and no overfitting was detected.

Due to page limit restrictions, below we present only the best three-idea set, so one may reproduce the dataset and run experiments. We omitted the estimated parameters so the other regression techniques may search for them. $safeLog(x)$ is { if ($x == 0$) return 0; else return $log(abs(x))$; } and $safeDiv(a, b)$ is { if ($b == 0$) return 0; else return $a/b$; }. ARG from 0 to 7 are the original features.

Table III
RESULTS OF KP ON 10-FOLD CROSS-VALIDATION AVERAGED ON 30 INDEPENDENT RUNS (DESCRIPTIVE ANALYSIS). BEST VALUES ARE IN BOLD FACE.

| # of new features | Statistic | $R^2$ | RMSE | MAE | Time (s) |
|---|---|---|---|---|---|
| 1 | Best | 0.5176 | 11.5977 | 9.4637 | **7.7576** |
| | Median | 0.3064 | 13.9059 | 11.1409 | **8.7677** |
| | Worst | 0.0520 | 16.2581 | 13.1369 | 18.6947 |
| | Mean | 0.3126 | 13.8126 | 11.1309 | **9.4999** |
| | Std.-dev. | 0.0928 | 0.9413 | 0.8449 | 2.3255 |
| 2 | Best | 0.7013 | 9.1261 | 7.1530 | 11.9507 |
| | Median | 0.5554 | 11.1334 | 8.8511 | 13.8315 |
| | Worst | 0.1551 | 15.3484 | 12.0688 | 26.3916 |
| | Mean | 0.5267 | 11.3942 | 9.1127 | 14.5601 |
| | Std.-dev. | 0.1264 | 1.4843 | 1.2276 | 3.0127 |
| 3 | Best | 0.7858 | 7.7275 | 6.0728 | 15.6247 |
| | Median | 0.6777 | 9.4790 | 7.4367 | 17.2392 |
| | Worst | 0.4802 | 12.0386 | 9.6203 | 42.9885 |
| | Mean | 0.6713 | 9.5193 | 7.4948 | 18.2586 |
| | Std.-dev. | 0.0723 | 1.0325 | 0.8457 | 4.8962 |
| 5 | Best | 0.8383 | 6.7151 | 5.2468 | 23.6944 |
| | Median | 0.8068 | 7.3391 | 5.6371 | 28.8253 |
| | Worst | 0.7585 | 8.2053 | 6.6106 | 34.9846 |
| | Mean | 0.8074 | 7.3189 | 5.7383 | 28.8273 |
| | Std.-dev. | 0.0203 | 0.3804 | 0.3550 | 3.0339 |
| 10 | Best | **0.8582** | **6.2870** | **4.9153** | 48.0683 |
| | Median | **0.8479** | **6.5129** | **5.1299** | 54.9636 |
| | Worst | 0.8252 | 6.9811 | 5.4621 | 77.3004 |
| | Mean | **0.8464** | **6.5440** | **5.1278** | 57.4262 |
| | Std.-dev. | 0.0064 | 0.1349 | 0.1185 | 7.9236 |

$I_1$: $safeLog((min(ARG1, ARG6)))$
$I_2$: $safeLog((safeDiv(ARG3, ARG0)))$
$I_3$: $safeLog((safeDiv(ARG3, (safeDiv((min(ARG3, ARG7)), ARG0)))))$

The next tables compare the performance of various Weka regression techniques using the best feature set evolved for each experiment. The results shown in Table IV correspond to the Correlation Coefficient, where 1.0 means a perfect fit. The techniques are organized into Traditional Statistical Methods (from Linear Regression to Gaussian Processes), Lazy Learning (IBk), Machine Learning (from Support Vector Regression to RBF neural network), Trees (the remaining). For the original dataset (baseline), one may observe that the first group did present fitting quality below 0.9, what may be considered poor fitting, but that actually depends on the dataset being investigated. On the other hand, MLP and the two regression tree methods reached $R^2 >= 0.9$. The RBF network showed a very poor quality fit on the default configuration. We detected that it considers only two clusters and that increasing such value improves quality. However, one wants to investigate whether a better set of features may be useful even with the default configuration.

The original dataset has eight features. With a single feature generated by KP the significance test shows improvement for the Isotonic Regression and RBF network, but the other methods had a significant drop in prediction quality. For the latter, the improvement was large as it builds two clusters using a single feature. The correlation coefficient tends to increase with the number of features. Nonetheless, the regression tree methods did not benefit from the new features; in fact, their performance was equal or worse. A possible explanation is that the importance of the new features was not calculated by a regression tree, thus they are not necessarily important to such methods.

Most methods reached higher $R^2$ with the new features. For some of them, the more features the higher $R^2$, whereas for others there was oscillation (see Isotonic Regression), and for some others (tree methods) few features were simply inadequate.

Results of RMSE in Table V are directly related to prediction error where smaller numbers are better, and the conclusions are similar to those drawn for $R^2$. Large reductions in prediction error may be obtained for methods of different categories (see Linear Regression, Gaussian Processes, IBk, SMOreg) while equal or worse performance for others (MLP and trees). Finally, one may notice that the Linear Regression method with five features generated by KP shows lower prediction error than using the original dataset with more advanced methods such as Gaussian Processes, SVM (SMOreg), MLP, RBF, or REPTree.

## V. SUMMARY AND CONCLUSIONS

Kaizen Programming (KP) is a hybrid algorithm that uses a collaborative problem solving approach in which partial solutions are put together to result in a complete solution. It employs concepts of agent-based algorithms, evolutionary algorithms, statistics, and machine learning. The partial solutions are created by the experts (agents), that generate ideas based on knowledge obtained in the iterative improvement process. As the ideas have their contribution to the problem measured when a complete solution is evaluated, one may have more confidence in which ideas are useful for the next improvement cycle.

KP was coupled with Ordinary Least Squares to generate better features from the original ones in the concrete compressive strength dataset, aiming to improve the prediction performance of distinct regression techniques. It was shown that most regression techniques investigated here may reach lower prediction error when a better feature set is employed, and that simpler techniques may even outperform more complex ones. Another important aspect is that those complex techniques, such as neural networks, result in black-box models, whereas the simpler techniques may be seen as grey-box models because they can be easier to interpret. On the other hand, it was observed that regression tree methods did not make proper use of the new features as they operate in a different way.

It was not possible to directly compare with related work from the literature as the dataset is not the exactly same. However, some of them are almost the same, with a small number of different instances. Results using KP are similar to those of other methods from the literature, but requiring just a fraction of the number of models, i.e., 100 by KP versus hundreds of thousands by the other methods.

As future work we intend to improve the computational performance of our code, but trying to preserve both simplicity and flexibility. Also, KP will be tested on other datasets employed in the literature.

Table IV
COMPARISON OF $R^2$ (MEAN $\pm$ STANDARD-DEVIATION) WITH METHODS IN WEKA.

| Method / Dataset | Original | 1 new feature | 2 new features | 3 new features | 5 new features | 10 new features |
|---|---|---|---|---|---|---|
| Linear Regression | $0.78 \pm 0.04$ | $0.69 \pm 0.05$ ● | $0.84 \pm 0.02$ ○ | $0.89 \pm 0.02$ ○ | $0.91 \pm 0.02$ ○ | $0.92 \pm 0.01$ ○ |
| Least Median Squares | $0.67 \pm 0.08$ | $0.69 \pm 0.05$ | $0.84 \pm 0.02$ ○ | $0.89 \pm 0.02$ ○ | $0.91 \pm 0.02$ ○ | $0.91 \pm 0.02$ ○ |
| Isotonic Regression | $0.59 \pm 0.05$ | $0.71 \pm 0.05$ ○ | $0.69 \pm 0.04$ ○ | $0.58 \pm 0.07$ | $0.59 \pm 0.05$ | $0.65 \pm 0.05$ ○ |
| Pace Regression | $0.78 \pm 0.04$ | $0.69 \pm 0.05$ ● | $0.84 \pm 0.02$ ○ | $0.89 \pm 0.02$ ○ | $0.91 \pm 0.02$ ○ | $0.92 \pm 0.01$ ○ |
| Gaussian Processes | $0.88 \pm 0.02$ | $0.70 \pm 0.05$ ● | $0.84 \pm 0.02$ ● | $0.89 \pm 0.02$ | $0.93 \pm 0.01$ ○ | $0.93 \pm 0.01$ ○ |
| IBk (k=3) | $0.85 \pm 0.03$ | $0.73 \pm 0.05$ ● | $0.86 \pm 0.03$ | $0.90 \pm 0.02$ ○ | $0.93 \pm 0.02$ ○ | $0.93 \pm 0.01$ ○ |
| SMOreg | $0.77 \pm 0.04$ | $0.69 \pm 0.05$ ● | $0.84 \pm 0.02$ ○ | $0.89 \pm 0.02$ ○ | $0.91 \pm 0.02$ ○ | $0.92 \pm 0.02$ ○ |
| MultilayerPerceptron | $0.91 \pm 0.02$ | $0.70 \pm 0.05$ ● | $0.84 \pm 0.03$ ● | $0.89 \pm 0.02$ ● | $0.92 \pm 0.02$ ○ | $0.93 \pm 0.01$ ○ |
| RBFNetwork | $0.10 \pm 0.10$ | $0.64 \pm 0.05$ ○ | $0.49 \pm 0.06$ ○ | $0.21 \pm 0.09$ ○ | $0.28 \pm 0.07$ ○ | $0.05 \pm 0.11$ |
| REPTree | $0.90 \pm 0.02$ | $0.73 \pm 0.05$ ● | $0.84 \pm 0.03$ ● | $0.90 \pm 0.02$ | $0.90 \pm 0.02$ | $0.89 \pm 0.02$ |
| M5P | $0.92 \pm 0.03$ | $0.71 \pm 0.05$ ● | $0.85 \pm 0.02$ ● | $0.90 \pm 0.02$ ● | $0.93 \pm 0.02$ | $0.92 \pm 0.02$ |

○ statistically significant improvement, ● statistically significant degradation

Table V
COMPARISON OF RMSE (MEAN $\pm$ STANDARD-DEVIATION) WITH METHODS IN WEKA.

| Method / Dataset | Original | 1 new feature | 2 new features | 3 new features | 5 new features | 10 new features |
|---|---|---|---|---|---|---|
| Linear Regression | $10.46 \pm 0.73$ | $12.07 \pm 0.78$ ● | $9.13 \pm 0.60$ ○ | $7.74 \pm 0.57$ ○ | $6.74 \pm 0.56$ ○ | $6.67 \pm 0.54$ ○ |
| Least Median Squares | $16.47 \pm 4.05$ | $12.09 \pm 0.81$ ○ | $9.17 \pm 0.63$ ○ | $7.77 \pm 0.58$ ○ | $6.81 \pm 0.58$ ○ | $7.01 \pm 0.61$ ○ |
| Isotonic Regression | $13.50 \pm 0.89$ | $11.68 \pm 0.76$ ○ | $12.12 \pm 0.85$ ○ | $13.62 \pm 0.72$ | $13.50 \pm 0.89$ | $12.74 \pm 0.91$ ○ |
| Pace Regression | $10.47 \pm 0.73$ | $12.07 \pm 0.78$ ● | $9.13 \pm 0.60$ ○ | $7.74 \pm 0.57$ ○ | $6.74 \pm 0.56$ ○ | $6.36 \pm 0.48$ ○ |
| Gaussian Processes | $7.88 \pm 0.62$ | $11.93 \pm 0.77$ ● | $9.00 \pm 0.59$ ● | $7.53 \pm 0.54$ ○ | $6.31 \pm 0.54$ ○ | $5.98 \pm 0.54$ ○ |
| IBk (k=3) | $8.91 \pm 0.78$ | $11.39 \pm 0.92$ ● | $8.68 \pm 0.78$ | $7.21 \pm 0.66$ ○ | $6.06 \pm 0.60$ ○ | $6.03 \pm 0.60$ ○ |
| SMOreg | $10.90 \pm 1.08$ | $12.13 \pm 0.82$ ● | $9.18 \pm 0.64$ ○ | $7.75 \pm 0.58$ ○ | $6.77 \pm 0.57$ ○ | $6.73 \pm 0.57$ ○ |
| MultilayerPerceptron | $7.91 \pm 1.54$ | $13.72 \pm 2.40$ ● | $10.24 \pm 1.48$ ● | $8.78 \pm 1.28$ | $7.33 \pm 1.24$ | $6.76 \pm 1.09$ ○ |
| RBFNetwork | $16.59 \pm 0.98$ | $12.86 \pm 0.79$ ○ | $14.51 \pm 0.92$ ○ | $16.34 \pm 0.97$ | $16.05 \pm 0.90$ ○ | $16.62 \pm 1.00$ |
| REPTree | $7.27 \pm 0.78$ | $11.43 \pm 0.83$ ● | $8.97 \pm 0.75$ ● | $7.31 \pm 0.69$ | $7.35 \pm 0.71$ | $7.45 \pm 0.71$ |
| M5P | $6.35 \pm 0.96$ | $11.77 \pm 0.75$ ● | $8.75 \pm 0.61$ ● | $7.30 \pm 0.62$ ● | $6.29 \pm 0.62$ | $6.54 \pm 0.61$ |

○ statistically significant improvement, ● statistically significant degradation

## ACKNOWLEDGMENTS

This paper was supported by the Brazilian Government CNPq (Universal) grant (486950/2013-1) and CAPES (Science without Borders) grant (12180-13-0) to V.V.M., CAPES grant (CNPq PIBIC 165293/2013-6) to V. C. C, and Canada's NSERC Discovery grant RGPIN 283304-2012 to W.B.

## REFERENCES

[1] I.-C. Yeh, "Modeling of strength of high-performance concrete using artificial neural networks," *Cement and Concrete research*, vol. 28, no. 12, pp. 1797–1808, 1998.

[2] J.-S. Chou and C.-F. Tsai, "Concrete compressive strength analysis using a combined classification and regression technique," *Automation in Construction*, vol. 24, pp. 52–60, 2012.

[3] S. Bhanja and B. Sengupta, "Investigations on the compressive strength of silica fume concrete using statistical methods," *Cement and Concrete Research*, vol. 32, no. 9, pp. 1391–1394, 2002.

[4] I. B. Topcu and M. Sarıdemir, "Prediction of compressive strength of concrete containing fly ash using artificial neural networks and fuzzy logic," *Computational Materials Science*, vol. 41, no. 3, pp. 305–311, 2008.

[5] A. Baykasoğlu, A. Öztaş, and E. Özbay, "Prediction and multi-objective optimization of high-strength concrete parameters via soft computing approaches," *Expert Systems with Applications*, vol. 36, no. 3, pp. 6145–6155, 2009.

[6] M. Castelli, L. Vanneschi, and S. Silva, "Prediction of high performance concrete strength using genetic programming with geometric semantic genetic operators," *Expert Systems with Applications*, vol. 40, no. 17, pp. 6856–6862, 2013.

[7] L. Chen and T.-S. Wang, "Modeling strength of high-performance concrete using an improved grammatical evolution combined with macrogenetic algorithm," *Journal of Computing in Civil Engineering*, vol. 24, no. 3, pp. 281–288, 2009.

[8] H.-C. Tsai and Y.-H. Lin, "Predicting high-strength concrete parameters using weighted genetic programming," *Engineering with Computers*, vol. 27, no. 4, pp. 347–355, 2011.

[9] V. V. De Melo, "Kaizen programming," in *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, ser. GECCO '14. New York, NY, USA: ACM, 2014, pp. 895–902. [Online]. Available: http://doi.acm.org/10.1145/2576768.2598264

[10] J. R. Koza, *Genetic programming - on the programming of computers by means of natural selection*, ser. Complex adaptive systems. MIT Press, 1993.

[11] M. Brameier and W. Banzhaf, "Evolving teams of predictors with linear genetic programming," *Genetic Programming and Evolvable Machines*, vol. 2, no. 4, pp. 381–407, 2001.

[12] T. Soule and P. Komireddy, "Orthogonal evolution of teams: A class of algorithms for evolving teams with inversely correlated errors," in *Genetic Programming Theory and Practice IV*. Springer, 2007, pp. 79–95.

[13] S. X. Wu and W. Banzhaf, "Rethinking multilevel selection in genetic programming," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, 2011, pp. 1403–1410.

[14] J. H. Holland, "Adaptation," in *Progress in theoretical biology*, R. Rosen and F. Snell, Eds. New York: Academic Press, 1976, vol. 4, pp. 263–293.

[15] P. Collet, E. Lutton, F. Raynal, and M. Schoenauer, "Polar ifs+ parisian genetic programming= efficient ifs inverse problem solving," *Genetic Programming and Evolvable Machines*, vol. 1, no. 4, pp. 339–361, 2000.

[16] H. Gitlow, S. Gitlow, A. Oppenheim, and R. Oppenheim, *Tools and Methods for the Improvement of Quality*, ser. Irwin series in quantitative analysis for business. Taylor & Francis, 1989.

[17] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[18] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009.