ZGP ALGORITHM

DESIGN CONSIDERATIONS AND DEVELOPMENT

Revised: October, 9, 2018

Contents

Introduction and Forward **3**

Background **3**

Other Evolutionary Algorithms **4**

Genetic Algorithms (GA) 4

Genetic Programming (GP) 4

Linear Genetic Programming (LGP) 4

Common Challenges 5

ZGP Core Design Criteria **6**

Approach **9**

Design and Implementation **10**

Phenotype and Genotype 10

Candidate Fitness Assessment 12

Final Model Selection 12

Analysis Types 13

Model Form 13

Numerical Considerations 14

Extension to Higher Variable Usage 15

Data Random Permutation 15

MyDataModels Implementation 15

Target Audience 15

Design Objectives 16

Variable reduction 16

Fitness and selection Functions 18

Overview 18

Binary Classification 19

Model Fitness Function 19

Model Selection Function 20

Regression 21

Model Fitness Function 21

Model Selection Function 21

FUSION FUNCTION FORMAL DEFINITION 22

Introduction 22

Fusion function definition 23

Definition of Mathematical Operations 24

Introduction and Forward

This document serves to describe the design objectives for the ZGP modeling engine. The ZGP engine was designed to overcome a number of challenges faced in the implementation of other evolutionary algorithms. The design criteria were derived from commercial experience developing and deploying other evolutionary algorithms. ZGP was intended to address barriers and reduce sub-optimalities experienced in those earlier efforts.

The ZGP engine is currently used in the reference program known as MyDataModels. MyDataModels was designed to address specific market concerns and opportunities. Relevant objective and design decisions made in the creation of MyDataModels are also discussed. ZGP has application far beyond that expressed in MyDataModels. Some of the more significant key design opportunities and applications will be discussed as well.

Background

ZGP was the product of a development effort that arose after 12 years of experience designing and implementing evolutionary algorithms as components in commercial technology. For the majority of those efforts machine learning technologies were supportive and not the primary function of the commercial product. They provided a function necessary for attaining required functionality of those products but the products' primary function was not to make available machine learning technology.

Several of remaining efforts did involve products whose primary function was the modeling of data. In those cases however, the technology was operated in house and not distributed as commercial software for external use. This was due to the complexity of the implementations and specialized expertise required for optimal results. These technologies were commercially successful but the reliance on internal operation limited the opportunity size.

In short, in a commercial sense, they didn't scale well. Additionally, some of these implementations required exotic computing hardware such as expensive GPU processors in order to process larger amounts of data in a reasonable time with results that met requirements.

When the effort to produce the ZGP engine began, all the lessons learned from those efforts along and the previous barriers encountered were identified and considered. A design was sought to mitigate these issues and eliminate the major obstacles encountered in the past.

Other Evolutionary Algorithms

ZGP was designed on the back of significant operational experience with existing evolutionary algorithms. Some of the barriers encountered in earlier efforts with some of the more relevant earlier algorithms are described below.

Genetic Algorithms (GA)

Genetic algorithms are a very interesting category of algorithms. The designer's commercial experience with GA dates back to 2002 when it was employed to optimize update binary size for the first commercial 3G mobile phone over the air firmware updates. GA was quite innovative and impressive at its advent. However, the imposed mathematical structure for resulting models greatly limits the applications for which it can be optimally applied.

Genetic Programming (GP)

GP was quite innovative when it was first introduced. It afforded a quite versatile tree structure which can express a wide variety of mathematical forms. One of the limitations found with GP is the effect evolutionary operators have upon GP's tree structure. Operators such a crossover and macro mutation could introduce huge discontinuities of convoluted search space in the mathematical evolution of candidate solutions. This was observed to introduce inefficiencies in model convergence to optimal solutions.

Linear Genetic Programming (LGP)

LGP was exceptionally interesting. It produced a model in a form akin to computer assembly language. Expressing a model through a virtual state machine was quite a compelling idea and quite engaging to observe. Somewhat surprisingly, LGP did not appear to have the same issues with evolutionary operators as GP. Using LGP, the evolution of model logic was able to be modulated more precisely. This somewhat mitigated the discontinuity issues seen with GP.

From an operational perspective, there were three primary issues with respect to LGP.

The first was model complexity. LGP could create exceptionally complicated and intricate models (from a human comprehension and interpretation perspective). LGP was also extremely resourceful. There were occasions where it was observed to exploit numerical implementation artifacts as "shortcuts" to more complicated computations. When denied access to the trigonometry functions and the value of PI as a constant, it would produce impressively terse logic to approximate PI from the available constants.

The issue was that while the core logic in as model was always deliberately crafted, a great deal of ineffective operations would always be produced in tandem. While there were some techniques to reduce this such as detection and removal of semantic introns, for all but the most simple of problems, it was extremely difficult to definitively differentiate between essential and non-essential logic. This characteristic limited the ability to produce solutions which could be efficiently and thoroughly examined by domain experts.

The second issue was the expertise required to optimally operate the tool. The LGP as implemented exposed a number of control parameters ranging from evolutionary parameters to model complexity bounds. The values often required considerably tuning for each analysis problem. This required appropriate staff to successfully operate the tool. It also required a considerable amount of man hours to perform an entire project.

The third problem was that the algorithm would aggressively attempt to over fit the data. This required a significant amount of parameter experimentation to mitigate for each project.

Another issue that may be implementation specific was a dependency on costly hardware. This specific implementation was forced to rely upon expensive NVidia GPU compute units in order to model data sets of even modest size.

These factors limited the external usability of the tool and inflated the internal cost to operate.

Common Challenges

Algorithms of this category also face common well known challenges.

One of these challenges is model selection. It is trivial to fabricate a large number of models that perfectly explain a set of records. It is difficult to determine which of those models will perform best when evaluated with respect to new data. A lot of diverse research has been invested into the mechanics of the modeling aspects themselves but there seemed to be a lesser amount of effort invested in model selection using such algorithms.

Another common challenge is the selection of a fitness function. The common approaches to crafting a fitness function usually optimize a model with respect to a training partition of data using simple statistical metrics. Such an approach continually drives the pool of candidate models towards over fit models. This can lead to the effect that a population of candidate solutions becomes less and less useful in deployed form. More sophisticated fitness functions which guide model solutions toward optimal generalized solutions are needed.

For many of these algorithms, including GP and LGP, the genotype to phenotype mapping is constructed in a way that does not result in a search space topology that is expediently traversed by these algorithms. Certainly, many of these algorithms overcome this and still converge on reasonable solutions. Ideally, the design of a genotype/phenotype implementation should inherently result in a search space with smooth topological gradients which can be quickly and efficiently explored using evolutionary operators coupled with well crafted fitness functions.

For such algorithms to be competitive with other machine learning technologies some advancements in these areas were highly desirable.

ZGP Core Design Criteria

The principles described below represent the ideal objective sought in the development of the ZGP algorithm. Discussion of the degree of attainment of the final ZGP algorithm will be discussed in a subsequent section.

Understandable

Models produced should be in the form of a human readable mathematical equation. In scientific fields, having specific equations describing system behaviors enables both wide application and detailed exploration of the underlying phenomena.

Concise

The mathematics employed by generated models should be concise without ineffective terms or logic. At the minimum, in cases where models do contain such ineffective components, such ineffective components should be easily identifiable.

Controllable

In order to impose constraints upon the tendency of certain algorithms to produce logic “bloat”, model complexity should be able to be absolutely bounded and the algorithm should aggressively seek to optimize a given complexity allotment with respect to optimality.

Minimalistic

Models produced should be minimalistic in the sense of having a minimum reliance upon larger quantities of independent variables. The algorithm should intrinsically seek to aggressively limit the number of variables required by generated models while still seeking to obtain a solution that is within close proximity of optimality for a given data set. As an initial goal, the model should have a peak maximum efficiency when employing 5-7 independent variables.

Extendable

While the algorithm may be initially optimized to produce models with a small set of independent variables, there should exist practical methods by which problems with a dependency on a larger number of variables can be optimally modeled using the ZGP algorithm. Such methods should be resistant to degradation due to issues such as the curse of dimensionality and reluctant to overfit due to an abundance of variables.

Resilient

Poor or insufficient data should result is a model that is optimal with respect to the information contained within the data set under consideration. Such deficient data should not result in overly curve fit solutions or unrealistically optimistic performance estimations. Poor data should simply result in poor accuracy so that the consumer of the model can properly consider a model's merit and act accordingly.

Discernable/Efficient

The algorithm should be able to discern signals in small amounts of data. Data sets often contain large amounts of redundant information. In such cases, resource benefits may be realized by algorithms that can identify appropriate information in small samples by permitting successful analysis of smaller subsamples. Ideally, analysis of a small sample which includes all descriptive behavior should produce models on par with models generated from a full data set.

Discriminating

The modeling algorithm should be able to consider a large numbers of variables during analysis and automatically select a minimal subset comprised of the most useful variables.

Facilitative

Any genotype, phenotype and evolution operator implementations should intrinsically produce “search friendly” gradients when considering alternative mathematical operators, factors and independent variables. Limitations in degrees of freedom with respect to search space mobility should be aggressively minimized.

Effective

Fitness function methodology that guides evolution of candidate models to generalized solutions and intrinsically avoids overfit solutions. Evolutionary assessments such as tournament selection methodology should be sophisticated such that fitness scores represent an estimation of performance with respect to future data not simply an evaluation of accuracy with respect to a training partition.

Insightful

The algorithm should produce accurate estimation of future model performance. The accuracy realized when a model is deployed should closely match that estimated by the modeling process (assuming any implicit constraints imposed upon the modeled data set are no less restrictive than implicit constraints experienced in deployment).

Automated

The modeling process should require minimal configuration and control by a software operator. Ideally, modeling parameters should be autonomously determined by the implementation with automated analysis producing equal or superior results to those obtained by expert manual guided operation.

Practical

The implementation should be practical, not exotic. It should afford reasonable analysis irrespective of platform whether that platform is a powerful compute cluster or an inexpensive notebook computer.

Ubiquitous

The implementation should be able to be run on Windows, MacOS, and Linux. It should be equally suitable for local or remote operations (e.g. desktop or cloud). The implementation should be accessible from multiple programming environments. While model evaluation runtime SDKs may be made available for ease of use, model logic should be automatically generated and/or easily implemented natively in capable environments without the requirement of an SDK.

Transparent

Regardless of whether the modeling process itself is disclosed to end users, the resulting models should be completely disclosed and transparent such that the mathematics of the model are easily inspected, understood, and able to be implemented mathematically, completely independent of any specific implementation of the modeling engine.

Approach

With the principles outlined above in mind, a base algorithm was sought that strove to attain these objectives. A great deal of literature was reviewed and a large number of existing algorithms considered. The algorithms identified all had deficiencies with respect to one or more of the above constraints. Effort began to design and construct a novel algorithm which better met the design criteria.

It was decided that the category of algorithm would be an evolutionary algorithm. Evolutionary algorithms exhibit a tremendous ability to construct diverse and complex mathematical solutions. It was thought that this characteristic would be beneficial in producing model composition that meets the design criteria.

There are several well known challenges involved when working with evolutionary algorithms.

The first is that the tremendous ability of such algorithms to precisely describe a series of numbers makes it quite challenging to confine results to a general solution and not obtain an extremely overzealous and overfit solution. It was believed that this challenge was manageable with new approaches to fitness assessment, evolutionary guidance, and performance estimation.

The second is the stochastic and typically non-deterministic output of such algorithms. One challenge with respect to this is the notion that every analysis attempt is very likely to result in a completely different solution. This can become a barrier to adoption for analysts accustomed to deterministic algorithms as this is an unsettling situation for some.

With techniques such as regression, analysts are accustomed to having a single best answer that results from any and all modeling attempts with similar parameters and data. It is acknowledged that this is definitely worthy of consideration. However, while a deterministic algorithm that always produces the same answer is reassuring to some, for others the ability to generate a diverse set of dissimilar models is an immensely useful capability.

Upon consideration, it was decided that having an algorithm that can produce an infinite number of solutions of similar efficacy was a far more compelling and opportunistic capability. Consider for example the ability to generate an infinite number of mathematical models on demand that precisely describe quantum mechanical observations using completely dissimilar mathematics. Such a capability would likely lead to much greater insights or inspirations into the fundamental nature of matter than just a single known model.

In this view, this diversity of result is a strength, not a weakness. There would be practical considerations for some with respect to how to operationally employ such an algorithm but in terms of value to be gained, it held a strong advantage over algorithms which could only ever produce a single explanation for each problem.

At this point, design of the new evolutionary algorithm began.

Design and Implementation

As this was new research and an experimental effort, there were not separate and distinct design and implementation phases. Ideas were considered, the design of certain aspects was created; they were implemented and then evaluated. Test cases were assessed, results considered and numerous repetitions of this cycle undertaken. This section will discuss some of the key design and implementation points in this process.

Phenotype and Genotype

The first consideration was how to design a genotype and associated phenotype that would express the desired mathematical model structure. The resulting design also had to be amenable to suitable, effective and efficient evolutionary permutation.

The design criteria also called for improved search space mobility and improved search space gradients. It was desired that the implementation would create a contiguous gradient for the algorithm to transition between, for example, the mathematical expressions x^y and $x - y$. One of the problems observed with other evolutionary algorithms capable of such expressions is that when the algorithm jumped from one expression to the other, it often involved large discontinuities or exceptionally noisy topology. Such characteristics limited the ability of those algorithms to expediently explore the search space and converge on useful solutions. This can result in inefficient computation and diminished results.

Phenotypical requirements were considered and a phenotype structure established. Various structures to represent the genotype were considered such as trees and arrays. In the end, a matrix structure was chosen with a sophisticated mapping process from which the phenotype would be derived.

The matrix implementation provided that for each “node” multiple mathematical operations would be expressed. Each node also contained a real numbered value which governed the relationship of the two specified mathematical operators. Evolution operators had the ability to permute the mathematical operators as well as this real number. This system created continuous gradients between dissimilar mathematical operators and permitted the algorithm to seamlessly search between what would typically be convoluted or discontinuous search space.

The dimensions of the matrix provide complexity control. One dimension of the matrix controls the maximal number of input variables a model may reference. The other dimension bounds the mathematical complexity.

Values supplied to the nodes for evaluation were governed by the organization of the matrix and the choice of alternate phenotype interpretations as described below. These argument values may be explicit constant real valued numbers, independent variable values or the value produced by another node. Constant and source variables may be permuted by evolution operators. The algorithm takes a set of independent variable series and randomly employs them in candidate solutions during the analysis process.

Variables series are selected randomly and variables may be used more than once or not at all. This approach permits the algorithm to be given a set of hundreds of variables but ultimately automatically chose to use only the 4 variables it found most useful.

Due to the design of the genotype to phenotype mapping process, a given genotype could be interpreted as multiple phenotypes with relatively inexpensive computation. One desirable characteristic of this ability was that a single evolutionary operation upon a given genotype would simultaneously alter numerous additional phenotypes in different ways.

This has two benefits.

The first is that for a single change to a genotype, the fitness of numerous related phenotypes could be assessed and the optimal phenotype mapping chosen to represent that specific genotype. This essentially permits sampling of the search space at more than a single point for a given amount of computation resource. This translates into advantages in analysis time and computational resource requirements.

The second is that since a given phenotype interpretation rarely uses all of the permitted complexity, introns are permitted to exist. LGP offers a similar capability in that logic can exist without experiencing fitness pressure at every generation. This permits the retention of logic that was useful in the past, might not be useful in a given generation but might provide useful again in the future.

One common issue with evolutionary algorithms is that typically, the diversity of genetic material in a given candidate model population is in a constant state of decline as an analysis proceeds. This usually means that genetic material that is not constantly valuable is quickly eliminated from a population and is not easily synthesized again as dominant genetic material continues to displace all less dominant material. Algorithms which retain useful genetic constructs have the potential to produce superior results and have a lesser tendency to rapidly converge on the first strong candidate. Such early candidates have the potential to be suboptimal.

Candidate Fitness Assessment

Common approaches to candidate fitness assessment typically focus on relatively simple statistical measures. They often attempt to simply determine which candidate model best describes a training partition. The actual objective however is just to distinguish the relative merit of two different candidate solutions with respect to future data.

The approach taken in this effort was to develop a method to estimate the relative performance of a two models with respect to a validation partition using only information present in the training partition.

The solution devised was achieved by using a combination of sampling, statistical measures and the characteristics of distributions, as well as heuristics to estimate the relative performance of one model to another with respect to a validation set while only considering data in the training partition.

Final Model Selection

With evolutionary algorithms, it is usually not difficult to produce a sizeable population of candidate models which well describe the training partition. The challenge is arriving at a generalized solution of acceptable accuracy and then determining which of the candidate models will perform best on additional new data.

Based on prior experience, identifying the best performing model from a population of models produced by an evolutionary algorithm is not a trivial task. It is not uncommon when deeply analyzing the population of such an algorithm at the terminus of an analysis run to discover that a better solution existed in the population but was not selected by the algorithm.

The issue of selecting the “best” model from a set of candidates is challenging. Even beyond considerations of accuracy there are other potentially relevant factors such as costs to acquire the required variables, etc.

After considerable experimentation, a model selection method was chosen that was somewhat similar to the fitness assessment solution in that it employed sampling, heuristics and statistical principles. It was more complex however because it was permitted to use information obtained from both a training and validation partition. Also, instead of comparing the relative merit of just two models, it has to establish the relative merit of all models present in the candidate population.

Analysis Types

With experience applying other evolutionary algorithms to analysis types including binary classification, regression, time series, ranking, and clustering among others, it was required that the new algorithm would have the potential to be applied to these same problem types. The initial priority of analysis types to be implemented was decided to be (in descending order of importance):

1. Binary Classification
2. Regression
3. Time Series

Binary classification was the predominate focus for much of the development as most envisioned use cases involve decision making and most decision making processes can ultimately be reduced to a binary classification problem.

Regression was implemented but it did not receive the same amount of effort to tune the fitness and selection heuristics to optimally handle cases with disproportionate representation of values. Further efforts had commenced but not yet completed to better tune the process for optimal modeling without the need for data preprocessing methods such as stratified sampling.

The same core evolutionary logic is used for all analysis types. The difference is in the presentation of observations and computation of fitness. The core engine was designed such that additional problem types are implemented by deriving common abstract classes and customizing presentation, fitness, selection and assessment logic as needed for a given type of analysis. This approach has been used in the past with other core algorithms to great success with analysis types such as generic time series, financial time series, ranking, and clustering. There are no known issues applying these same approaches with this new algorithm.

Model Form

The final form of generated models was chosen to be a tree representation such that basic elements of a model can be expressed in a nested form such as $f(x,\ g(y,z))$. All such base element functions take two values as arguments. The argument values can take values from independent variables, real valued constants, or the result of a similarly formed nested function.

As each function in a model node contains two mathematical operators, a special mechanism was required to express this special mathematical construct. This notation is referred to by this analysis engine as the FUSION operator. The FUSION operator provides a concise syntax to represent and communicate all the necessary information to reproduce the logic of a model node. For the formal definition of the FUSION function, see the paragraph "Fusion Function Formal Definition."

Models therefore take the form of FUSION functions where arguments are independent variables, constants, or the result of another FUSION function. Binary Classification analysis models consist of two FUSION functions each of which produces a value. Each FUSION function produces a real number which is interpreted as a propensity for either the positive or negative predictive case. The greater value represents the predicted case and the magnitude of difference between the two values is interpreted as a confidence in prediction measure.

Regression models are of a slightly different form. Regression models consist of 1 or more FUSION functions added together. The number of FUSION functions employed has an upper bound imposed by one dimension of the genotype matrix. The optimal phenotype mapping identified for a given genotype dictates the actual number of FUSION functions that comprise the final model.

Numerical Considerations

Models are evaluated using IEEE 754 compliant double precision floating point numerical operations. Computations that produce undefined values are interpreted as NaN (Not a Number).

In order to minimize the development of models that are not mathematically robust, during fitness, all nodes of candidate models are tested against all observations in the fitness partition. If any node is found to be exotic, that candidate is assigned the minimum possible value of fitness.

Nodes are considered exotic if they are given or produce NaN, if their numeric representation is degenerate, or if any values given or produced are within a specified proximity of the limits of precision. Models exhibiting any of these behaviors at any time have been found to be less generalized, less portable, more brittle or less reliable in practice when deployed.

Producing a fitness pressure to minimize the existence of genotypical content that produces such behavior improves model development and deployment success. It also manifests performance benefits by quickly disregarding lower value regions of search space. Limiting edge case behavior in this way also minimizes algorithmic exploitation of platform implementation artifacts which improves model consistency when models are deployed on multiple or different platforms.

One additional numerical consideration with respect to model using the FUSION function notation pertains to the real number on the interval [0,1] that governs the relationship of the two mathematical operators specified in a FUSION function expression. The real number that governs this relationship is stored as a rational number consisting of two integers to ensure portability and consistency across deployment platforms.

Extension to Higher Variable Usage

The default configuration of the algorithm is optimized to produce models reliant upon 1-7 independent variables as model inputs.

Evolutionary algorithms are particularly well suited for the production of ensemble models. The correlation of candidate models with respect to other existing models in an ensemble can be employed by fitness assessment to encourage the development of ensemble component models that are minimally corrected. This capability is not currently implemented but adding such functionality is in our short term roadmap.

Data Random Permutation

One feature built into the core engine which is not currently used in the commercial product is the ability to randomly permute the values of independent variables by a specified magnitude. The ultimate objective from this capability was to produce a set of data quality assessment tools. Research is still preliminary at this point but there is a good expectation that by performing a series of analyses while permuting independent variable values by varying degrees, an assessment can be made regarding the quality of the data under consideration and the algorithms ability to derive pertinent knowledge from that information.

MyDataModels Implementation

Target Audience

While the algorithm was created from the beginning in the form of a software engine, a reference application was required to enable non-programmatic use and evaluation. This first reference application using the analysis engine became known as MyDataModels.

As there are many existing modeling tools which require substantial data science expertise to operate yet few that were relatively simple and useful to those without data science experience, it was decided that the MyDataModels application should strive to be as automated and simple to use as possible to provide utility to the widest possible audience. Due to the design of the algorithm, considerations such as avoiding correlated independent variables were unnecessary and the algorithm afforded many attributes useful to such automation.

Design Objectives

The primary design objectives for MyDataModels were to produce a data modeling tool which could be learned and successfully operated by professionals who may not have specific knowledge and expertise in data analysis or statistics.

To this end, a workflow was designed upon a software wizard like model comprised of Steps 1, 2, 3, etc. Nomenclature used in the application was derived from statistical principles where terms were thought to be informative to less sophisticated end users. New descriptive terms were adopted where it was thought to make usage easier to learn and apply for those new to data science and modeling.

While some base algorithm parameters were exposed to the end user to configure if desired, the default parameters were suitable for a large majority of use cases and afforded worthwhile results for very little effort. MyDataModels contains useful data preprocessing functionalities - randomization, stratified sampling, missing values handling, time series - and only supports today ingestion of data in csv format. More data formats will be supported as planned in our roadmap.

Variable reduction

One interesting feature of MyDataModels is an automated feature reduction process. This process has been successfully used to identify and rank the variables most useful in terms of predicting a dependent variable. One case study applied the feature reduction process to a gene expression dataset comprised of approximately 55,000 variables and 183 observations. Of the 20 most significant variables found by MyDataModels to be most predictive of outcome, 15 of the probes were confirmed by other research to be highly relevant. Of particular interest was the fact that for one of the highest ranked genes, 5 variables existed in the dataset representing different base pair sequences for that gene. MyDataModels ranked 4 of the 5 probes measuring expression for that gene in the most significant 20 genes (with the 5th probe ranked somewhere in the top few hundred probe IDs). This demonstrates that the variable reduction process is correctly assessing the most important variables and not just identifying a subset of variables that has predictive value.

The feature reduction process is based on the concept of relative merit not absolute accuracy. Since the goal is to rank, not to quantify, this is all that is required. To accomplish this, analysis only needs to proceed far enough to make a relative determination between variables. The process used performs a large number of “micro-analyses” considering only a random subset of the available variables. These “micro-analyses” are analysis runs similar to standard full modeling but of much shorter duration. Statistics from these analysis runs is collected and a relative ranking of the full set of variables is produced. MyDataModels manages these processes by storing cumulative metrics in such a way that the process can be interrupted and resumed at any time, even between application instances. This permits an iterative approach were preliminary feature reduction can be performed to assess final model performance and then additional reduction processing to incrementally improve the resolution of the ranking. The process provides a number of controls such as to specific the maximum number of variables that can be considered in a single “micro-analysis”. This enables performance benefits by rapidly producing ranking for the full set of variables, excluding a proportion of the least useful variables and then performing additional reduction which focuses only on the remaining variables known to be of higher value. This process is useful because there is no value in ensuring accuracy of relative rank for variables which will never be used in modeling.

Fitness and selection Functions

Overview

The ZGP engine implementation uses an abstracted and modular architecture for implementing analysis types on top of the core ZGP genotype engine (binary classification, regression, etc.). With this architecture multiple problem types can be presented to the underlying engine to leverage the base evolutionary algorithm to evolve models that fulfill various different analysis objectives. One of the abstract concepts in this abstraction is the concept of assessment functions. This is primarily used to provide evolutionary and selection guidance to the core ZGP genotype engine.

While the ZGP codebase contains numerous experimental implementations of these functions, the reference analysis application exposes single specific defaults for each required purpose. These defaults were chosen based on experimental results for the following considerations.

- Consistent behavior
- Quality of fully automated results
- Usefulness with respect to a maximal variety of analysis use cases

This document provides the definitions of the model evolution (fitness) and model selection functions as currently employed for default mechanisms in the Databolics reference ZGP application. As of this writing, the two analysis types supported by the reference application are binary classification and regression. The default evolution and selection functions for these two types of analyses are described below.

Binary Classification

Model Fitness Function

The ZGP engine contains numerous variants of fitness and validation functions used for model evolution and selection. The current default fitness and validation functions for binary classification are described below. These particular functions were found to be the most consistent of the present variants in terms of consistent convergence characteristics and resulting model efficacy.

Fitness evaluation is performed by evaluating the model with respect to the training partition. The training partition is divided into a configurable number of distinct folds. Each record in the training is a member of one and only one fold. Area under the curve (ROC) is evaluated with respect to each fold. True positive rate and true negative rate are computed for the entire training partition. Prediction confidence measures are computed for all records in the training partition.

This model fitness function is then evaluated as $f(m)$ defined below. The resulting value is within the closed interval [0,1]. Higher values indicate a model that is expected to better perform with respect to future data.

$$g(m) = \left(\overline{A} - \frac{s_{AUC}}{2}\right)(1.0 - |P - N| \cdot 0.1)$$

$$f(m) = \begin{cases} \overline{C_{correct}} \leq \overline{C_{correct}}, & g(m) \cdot 0.9 \\ \overline{C_{correct}} > \overline{C_{incorrect}}, & g(m) \end{cases}$$

Where:

m is the model descriptor of the model to be evaluated.

$\overline{A}$ is the arithmetic mean of the area under the curve for all folds.

s_{AUC} is the standard deviation of the area under the curve with respect to all folds.

P is the true positive rate obtained from evaluation of the model with respect to the entire training partition.

N is the true negative rate obtained from evaluation of the model with respect to the entire training partition.

$\overline{C_{correct}}$ is the arithmetic mean of the confidence estimations for correct predictions made by the model with respect to the entire training partition.

$\overline{C_{incorrect}}$ is the arithmetic mean of the confidence estimations for incorrect predictions made by the model with respect to the entire training partition.

The two piecewise functions defined for $f(m)$ serve to provide a fitness pressure that favors models with a correlation between confidence estimation and prediction accuracy.

The multiplier produced using P and N serve to encourage parity with respect to true positive rate and true negative rate. While models can be biased to favor either of these measures in deployment, it was observed that models developed with this characteristic tended to produce models well suited for a wide variety of general purpose analytical situations.

Model Selection Function

Model selection is performed by evaluating the model with respect to both the training partition and the validation partition. For both partitions, area under the ROC curve, true positive rate and true negative rate is computed.

This model selection function is then evaluated as $v(m)$ defined below. The resulting value is within the closed interval [0,1]. Higher values indicate a model that is expected to better perform with respect to future data.

$$v(m) = AUC_{Training} \bullet AUC_{Validation} \bullet \left[1.0 - 0.1 \cdot \left(\left|TPR_{Training} - TNR_{Training}\right| + \left|TPR_{Validation} - TNR_{Validation}\right|\right)\right]$$

Where :

m is the model descriptor of the model to be evaluated.

$AUC_{Training}$ is the area under the ROC curve for the specified model with respect to the entire training partition.

$AUC_{Validation}$ is the area under the ROC curve for the specified model with respect to the entire validation partition.

$TPR_{Training}$ is the true positive rate obtained from evaluation of the model with respect to the entire training partition.

$TNR_{Training}$ is the true negative rate obtained from evaluation of the model with respect to the entire training partition.

$TPR_{Training}$ is the true positive rate obtained from evaluation of the model with respect to the entire validation partition.

$TNR_{Training}$ is the true negative rate obtained from evaluation of the model with respect to the entire validation partition.

Regression

While multiple experimental functions are implemented in the ZGP code base, the reference application uses simple measures for the purposes of fitness assessment and model selection. These simple options were chosen for this purpose due to the general purpose behavior for a large number of analysis use cases was desired for the reference application.

Experimental evidence suggests that better convergence and generalized fit can be realized by employing regression fitness and selections functions that employ sampling and consistency measures similar to those used in binary classification above.

Limited initial development resources were primarily focused on binary classification problems as at the time of conception, existing business use cases were primarily concerned with decision making and classification. Further development on the regression fitness and selection functions is likely to be quite worthwhile.

Model Fitness Function

Model fitness for regression analysis types is based on root mean square error (RMSE). The actual value used is simply RMSE for the model with respect to the entire training set normalized to the interval [0,1] and multiplied by -1. The negative multiplier is employed so that greater values represent higher accuracy.

Model Selection Function

The regression model selection metric uses the same statistic as described for the model fitness function above but evaluated for both the training and validation partitions. The model selection score is the computed as the arithmetic mean of these two values.

FUSION FUNCTION FORMAL DEFINITION

Introduction

This document describes and defines the FUSION function as used by ZGP Machine Learning Platform.

The FUSION function serves the purpose of creating gradients within search topology that are more easily spanned and explored by evolutionary machine learning algorithms.

The concept is quite simple; given two different mathematical functions, obtain a function that is a linear interpolation between the two functions, with a bias controlled by a single real number of the range 0 to 1.

The mathematical definition is described in the following sections.

Fusion function definition

Where the fusion function is expressed in the form:

$$fusion(a,b,d,x,y) \in \mathbb{R}$$

and given the following definitions:

$$s = \left\{ \begin{array}{l} "ADD","SUB","MUL","DIV","POW","ABS","SIN","COS", \\ "EXP","NLG","SQU","SRT","MOD","CEI","FLO","INT" \end{array} \right\}$$

$$a \in s, \quad b \in s$$

$$d \in \mathbb{R} \ \left| 0 \le d \le 1 \right.$$

$$x \in \mathbb{R}, \ y \in \mathbb{R}$$

a and b each represent a mathmatical operation from the set s.
$f_a(x,y)$ represents the mathematical operation specified by a.
$f_b(x,y)$ represents the mathematical operation specified by b.

The fusion function is evaluated as:

$$fusion(a,b,d,x,y) = f_b(x,y) - d\left[f_b(x,y) - f_a(x,y) \right]$$

Definition of Mathematical Operations

Definition of mathematical operations of s :

$$"ADD" \rightarrow f(x,y) = x + y$$

$$"SUB" \rightarrow f(x,y) = x - y$$

$$"MUL" \rightarrow f(x,y) = xy$$

$$"DIV" \rightarrow f(x,y) = \begin{cases} 0, & \text{if } y = 0 \\ \dfrac{x}{y}, & \text{if } y \neq 0 \end{cases}$$

$$"POW" \rightarrow f(x,y) = x^y$$

$$"ABS" \rightarrow f(x,y) = |x|$$

$$"SIN" \rightarrow f(x,y) = \sin x$$

$$"COS" \rightarrow f(x,y) = \cos x$$

$$"EXP" \rightarrow f(x,y) = e^x$$

$$"NLG" \rightarrow f(x,y) = \ln x$$

$$"SQU" \rightarrow f(x,y) = x^2$$

$$"SRT" \rightarrow f(x,y) = \sqrt{|x|}$$

$$"MOD" \rightarrow f(x,y) = x \bmod y$$

$$"CEI" \rightarrow f(x,y) = \lceil x \rceil$$

$$"FLO" \rightarrow f(x,y) = \lfloor x \rfloor$$

$$"INT" \rightarrow f(x,y) = \begin{cases} \lfloor x \rfloor, & \text{if } x \geq 0 \\ \lceil x \rceil, & \text{if } x<0 \end{cases}$$