An Evolutionary Algorithm for Big Data Multi-class Classification Problems

Michael F. Korns

Analytic Research Foundation, 2240 Village Walk Drive Suite 2305, Henderson Nevada 89052 mkorns@korns.com.

Abstract.

As symbolic regression (SR) has advanced into the early stages of commercial exploitation, the poor accuracy of SR, still plaguing even the most advanced commercial packages, has become an issue for industrial users. Users expect to have the correct formula returned, especially in cases with zero noise and only one basis function with minimally complex mathematical depth.

At a minimum, users expect the response surface of the SR tool to be easily understood, so that the user can know a priori on what classes of problems to expect excellent, average, or poor accuracy. Poor or *unknown* accuracy is a hindrance to greater academic and industrial acceptance of SR tools.

In several previous papers, we published a complex algorithm for modern symbolic regression which is extremely accurate for a large class of Symbolic Regression problems on noiseless data. The class of problems, on which SR is extremely accurate, is described in detail in these previous papers. This algorithm is extremely accurate, in reasonable time on a single processor, for from 25 up to 3000 features (columns). Further research has shown that the extremely accurate symbolic regression algorithms also convey greater accuracy even in noisy circumstances - albeit not extreme accuracy.

Armed with this level of success in symbolic regression, we naively thought that achieving extreme accuracy applying genetic programming to symbolic multi-class classification would be an easy goal. However, we soon discovered that algorithms which convey extreme accuracy in symbolic regression do not translate directly into algorithms which convey extreme accuracy in symbolic multi-class classification. Furthermore, others have encountered serious issues applying genetic programming to symbolic multi-class classification (Castelli 2013).

This is the first paper in a planned series of papers attempting to develop the algorithms necessary for achieving extreme accuracy in genetic programming applied to symbolic multi-class classification.

In this paper we develop an evolutionary algorithm for optimizing a single symbolic multi-class classification candidate. The algorithm is designed for

big data situations such that the computational effort grows linearly as the number of features and training points increase. The algorithm is tested using statistically correct, out of sample training and testing. The algorithm's behavior is demonstrated on both theoretical problems and on previously published UCI and industry tests cases.

Key words: Symbolic Classification, Abstract Expression Grammars, Grammar Template Genetic Programming, Genetic Algorithms, Particle Swarm.

1 Introduction

The discipline of Genetic Programming (GP)(Koza 1992) (Koza 1994) (Koza 1999) has matured significantly in the last two decades. There are numerous practical successes reported in many application domains (Poli 2008)(Gandomi 2015). A great deal of work has been done to strengthen the theoretical foundations of GP (Langdon 2002). There is at least one commercial package Symbolic Regression (SR) package which has been on the market for several years *http://www.rmltech.com/*. There is now at least one well documented commercial symbolic regression package available for Mathematica *www.evolved-analytics.com*. There is at least one very well done open source symbolic regression package available for free download *http://ccsl.mae.cornell.edu/eureqa*. In addition to our own ARC system (Korns 2010), currently used internally for massive (million row) financial data nonlinear regressions, there are a number of other mature symbolic regression packages currently used in industry including (Smits 2005) and (Kotanchek 2008). Plus there is another commercially deployed regression package which handles up to 50 to 10,000 input features using specialized linear learning (McConaghy 2011).

Yet, despite its increasing sophistication, genetic programming has encountered serious issues addressing multi-class classification applications (Castelli 2013).

An algorithm has been developed, specifically for genetic programming applications in multi-class classification, called M2GP, which has achieved reasonable accuracy in several multi-class classification tests (Ingalalli 2014). One highly attractive attribute of the M2GP algorithm is its solid theoretical machine learning foundations. One unfortunate issue with the M2GP algorithm, is the requirement to compute several matrix multiplications and one matrix inversion per class, and the necessity that the crucial class covariance matrix be non-singular. Furthermore, the M2GP algorithm requires several vector products and one matrix multiply per training point for scoring. As the number of classes and the number of training examples grows larger, in a multi-class classification problem, the computational requirements for the M2GP algorithm increases geometrically. In financial applications there is no a priori guarantee that all of the class covariance matrices will be non-singular. Plus, at least in many financial applications, we have found that the number of classes and training examples can be quite large.

We are interesting in exploring a purely evolutionary alternative to the M2GP algorithm which will not require matrix inversions, and which will operate correctly in conditions wherein the M2GP class covariance matrix is singular.

Before continuing with the discussion of our alternative multi-class classification algorithms for big data problems, we proceed with a basic introduc-

tion and formalization of Genetic Programming Classification (GPC), liberally adapting terminology from M2GP (Castelli 2013) and from Generalized Linear Models (GLMs) (Nelder 1972).

The formalization of genetic programming classification is the class of Genetic Programming Classifier Models (GPCMs). A GPCM is a collection of **K** discriminant functions D^k; k = 1,2,...,K, a dependent unordered categorical variable y with integer values from 1 through K, and an independent data point with M features x = $<x_1, ..., x_M>$: such that

- (*E1*) ey(x) = nomial($c^1{}_0$+($c^1{}_1D^1$(x)),...,$c^K{}_0$+($c^K{}_1D^K$(x)))
- (*E2*) a_k=max(a_1,...,a_K) implies k = nomial(a_1,...,a_K)
- (*E3*) y=ey implies match(y,ey)=0 and y<>ey implies match(y,ey)=1

Given a collection of GPCMs, a collection of independent data points, X, and a collection of dependent categorical variables, Y, the fittest GPCM is the GPCM which minimizes match(Y,EY).

The discriminant functions are a broad generalization and can represent any possible linear or nonlinear formula, as in the following examples:

- (*E4*) $D^1 = x_3$
- (*E5*) $D^2 = x_1+x_4$
- (*E6*) D^3 = sqrt(x_2)/tan(x_5/4.56)
- (*E7*) D^4 = tanh(cos(x_2*.2)*cube(x_5+abs(x_1)))

Viewing the problem in this fashion, we gain an important insight. Genetic programming classification does not add anything to the standard techniques of classification. The value added by GPC lies in its abilities as a search technique: how quickly and how accurately can GPC find an optimal set of discriminant functions D. The immense size of the search space provides ample need for improved search techniques. In basic Koza-style tree-based Genetic Programming (Koza 1992) the genome and the individual are the same Lisp s-expression which is usually illustrated as a tree. Of course the tree-view of an s-expression is a visual aid, since a Lisp s-expression is normally a list which is a special Lisp data structure. Without altering or restricting basic tree-based GP in any way, we can view the individual discriminant functions not as trees but instead as s-expressions such as this depth 2 binary tree s-exp: *(/ (+ x_2 3.45) (* x_0 x_2))*, or this depth 2 irregular tree s-exp: *(/ (+ x_4 3.45) 2.0)*.

In basic GP the non-terminal nodes are all operators (implemented as Lisp function calls), and the terminal nodes are always either real number constants or input features. The maximum depth of a GP individual is limited by the available computational resources; but, it is standard practice to limit the maximum depth of a GP individual to some manageable limit at the start of a genetic programming run.

Given any selected maximum depth **d**, it is an easy process to construct a maximal binary tree s-expression U_d, which can be produced by the GP system without violating the selected maximum depth limit. As long as we are reminded that each f represents a function node while each t represents a terminal node (either a feature v or a real number constant c), the construction algorithm is simple and recursive as follows.

- (U_0): t
- (U_1): (f t t)
- (U_2): (f (f t t) (f t t))
- (U_3): (f (f (f t t) (f t t)) (f (f t t) (f t t)))
- (U_d): (f U_{d-1} U_{d-1})

The basic GP symbolic regression system (Koza 1992), which we will adapt for symbolic classification, contains a set of functions F, and a set of terminals T. If we let t ∈ T, and f ∈ F ∪ ξ, where $\xi(a,b) = \xi(a) = a$, then any basis function produced by the basic GP system will be represented by at least one element of U_d. Adding the ξ function allows U_d to express all possible basis functions generated by the basic GP system *to a depth of d*. **Note to the reader**, the ξ function performs the job of a pass-through function. The ξ function allows a *fixed-maximal-depth* expression in U_d to express trees of varying depth, such as might be produced from a GP system. For instance, the varying depth GP expression $x_2 + (x_3 - x_5) = \xi(x_2,0.0) + (x_3 - x_5) = +(\xi(x_2\ 0.0)\ -(x_3\ x_5))$ which is a *fixed-maximal-depth* expression in U_2.

In addition to the special pass through function ξ, in our system we also make additional slight alterations to improve coverage, reduce unwanted errors, and restrict results from wandering into the complex number range. All unary functions, such as *cos*, are extended to ignore any extra arguments so that, for all unary functions, *cos(a,b) = cos(a)*. The *sqroot* and *ln* functions are extended for negative arguments so that *sqroot(a) = sqroot(abs(a))* and *ln(a) = ln(abs(a))*.

Given this formalism of the search space, it is easy to compute the size of the search space, and it is easy to see that the search space is huge even for rather simple discriminant functions. For our use in this chapter the function set will be the following functions: F = (+ - * / <= >= maximum minimum inv ξ), where inv(x) = (1.0/x), where (x<=y) is 1.0 if true or is 0.0 if false, where (x>=y) is 1.0 if true or is 0.0 if false, and where ξ(a,b) = ξ(a) = a. The terminal set are the features $\mathbf{x}_1$ through $\mathbf{x}_M$ and the real constant **c**, which we shall consider to be 2^{64} in size.

The fitness measure, used in this paper, is the Classification Error Percent (CEP) computed as the average error percent on a per class basis. The formula for the computation of the CEP fitness measure is shown in fitness equation (F1). The term Error_k refers to the total number of unmatched cases for the

class k - i.e. *match(ey,y)=1* for class k. The term Count_k refers to the total number of training points for class k.

- (*F1*) CEP = average($\text{Error}_k/\text{Count}_k$)

In this paper we introduce a Multilayer Discriminant Classifier (MDC) algorithm for big data genetic programming multi-class classification. The MDC algorithm is evolutionary in approach. Each layer of the algorithm requires only one pass over the data, *including scoring*. So MDC's computational complexity grows linearly with larger training data points and classes. Furthermore, MDC's multilayer approach distributes the evolutionary attention across the entire genetic programming run so that more promising candidates receive more evolutionary attention and less promising candidates receive less evolutionary attention.

Prior to writing this chapter, a great deal of *tinker-engineering* was performed on the Lisp code supporting the MDC algorithm. For instance, all generated candidate code was checked to make sure that the real numbers were loaded into Intel machine registers without exception. All vector pointers were checked to make sure they were loaded into Intel address registers at the start of each loop rather than re-loaded with each feature reference. As a result of these engineering efforts, the MDC algorithm is quite practical to run on a personal computer. Of course a cloud configuration can always be used to achieve enhanced performance in much shorter elapsed times.

2 The MDC Algorithm

The Multilayer Discriminant Classification (MDC) algorithm is composed of three main layers of evolutionary activity. These three layers focus on the optimization of the GPCM coefficients for the optimal GPCM candidate, and are a direct alternative to the more mathematically correct M2GP algorithm. None of the MDC algorithm layers require matrix inversion. Therefore the MDC algorithm can operate in training conditions which do not meet the basic requirements of the M2GP algorithm.

The MDC algorithm attempts to optimize the following equation for the K selected discriminant functions $\text{D}^k(\text{x})$.

- (*E1*) ey(x) = nomial($\underline{\mathbf{c}}^1{}_0+(\underline{\mathbf{c}}^1{}_1\text{D}^1(\text{x})),...,\underline{\mathbf{c}}^K{}_0+(\underline{\mathbf{c}}^K{}_1\text{D}^K(\text{x}))$)

The MDC algorithm is evolutionary in approach. Each layer of the algorithm requires only one pass over the data, *including scoring*. The MDC algorithm's multilayer approach distributes the evolutionary attention across the entire genetic programming run so that more promising candidates receive more evolutionary attention and less promising candidates receive less evolutionary attention.

2.1 Partial Bipolar Regression

The first layer of the MDC algorithm attempts to find a quick initial approximation of the optimal coefficients in the selected GPCM.

- (*E1*) ey(x) = nomial($\underline{\mathbf{c}}^1{}_0$+($\underline{\mathbf{c}}^1{}_1$D^1(x)),...,$\underline{\mathbf{c}}^K{}_0$+($\underline{\mathbf{c}}^K{}_1$D^K(x)))

At initialization time, randomly selected coefficients for the current candidate GPCM (E1) will generally score a Classification Error Percent (CEP) between 80% and 99% especially when as K grows larger ***even when the GPCM is an exact match for the dependent variable Y***. We need a quick initialization approach which scores a CEP much closer to 0% even for large K. Partial Bipolar Regression (PBR) is just such a quick GPCM coefficient initialization methodology *especially when the GPCM is an exact match for the dependent variable Y*.

For each new GPCM, the MDC algorithm runs K Partial Bipolar Regression (PBR) single passes through the data followed by a single scoring pass thought the data. To understand how this works, notice that formula (E1) is composed of K simple discriminant formulas as follows.

- (*E8*) $\underline{\mathbf{c}}^k{}_0$+($\underline{\mathbf{c}}^k{}_1$D^k(x))

The PBR layer runs a simple single pass regression on each of the K discriminant formulas against the dependent variable y; however, as each y is loaded its value is temporarily altered on the fly according to the following rule. If y=k, then y becomes +1. If y<>k, then y becomes -1. This partial bipolar regression produces coefficient candidates $\mathbf{c}^k{}_0$ and $\mathbf{c}^k{}_1$ which are approximately in the general ballpark required for an initial guess.

Once all K discriminant formulas have been partially bipolar regressed, a single pass scoring run usually scores a CEP in the approximate range of from 5% to 20%, *in cases where the GPCM is an exact match for the dependent variable Y*, even when K grows larger.

Obviously, in cases where the GPCM is a poor match for the dependent variable Y, the returned CEP will remain in the 80% to 99% range as if the coefficients had been randomly chosen. This initial guess CEP discrepancy been exact matches and poor matches presents an evolutionary activity distribution opportunity which PBR measures by computing the PBR success rate. The PBR success rate is the percent inverse of CEP as follows.

- (*E9*) PBRSuccessRate = 100%-CEP

The PBR success rate will be used in the next layer of the MDC algorithm to distribute evolutionary activity more efficiently.

2.2 Modified Sequential Minimization

The second layer of the MDC algorithm is an opportunistic modification of Platt's sequential minimization optimization algorithm often used to train support vector machines (Platt 1998).

At the start of the modified sequential minimization (MSM) layer, the candidate GPCM contains a swarm pool of a single set of coefficient constants which was produced by the PBR layer. Also the CEP for this single entry in the swarm pool and the PBR success rate are both available. The MSM layer multiplies the PBR success rate times 100 to produce the MSM repetition count, which determines the number of times the MSM will repeat without CEP improvement. The higher the PBR success rate, the greater the count of times the MSM will repeat. So the PBR success rate determines the evolutionary activity spent on the current candidate. Better candidates receive more evolutionary activity. Worse candidates receive less evolutionary activity.

For each repetition of the MSM, on the current candidate GPCM, the most fit entry in the swarm pool is chosen and a single erroneous training point is chosen at random. Since the chosen training point is in error, we know that its estimated dependent variable will not match the actual dependent variable i.e. **ey**<>**y**. Let us assume that **ey=i** and that **y=j**. We therefore know that two discriminant formulas have the following relationship.

- (*E10*) $(c^i{}_0+(c^i{}_1 D^i(x))) > (c^j{}_0+(c^j{}_1 D^j(x)))$

And we also know that, for that single erroneous training point, the maximum of all discriminant formulas, other than **i** and **j**, which we shall name α, will be also be less than $\mathbf{(c^i{}_0+(c^i{}_1 D^i(x)))}$, which value we shall name β. Furthermore, we know that we can convert this single erroneous training point into a successful match if we alter the **i** and **j** coefficients such that the following relationships becomes true.

- (*E11*) $\alpha <= (c^i{}_0+(c^i{}_1 D^i(x))) < (c^j{}_0+(c^j{}_1 D^j(x))) <= \beta$

We select a value between α and β at random then alter $\mathbf{c}^j{}_0$ and $\mathbf{c}^j{}_1$ such that $\mathbf{(c^j{}_0+(c^j{}_1 D^j(x)))}$ is equal to the randomly selected value and then alter $\mathbf{c}^i{}_0$ and $\mathbf{c}^i{}_1$ such that $\mathbf{(c^i{}_0+(c^i{}_1 D^i(x)))}$ is slightly less than the selected value. Of course we do not know what havoc these alterations will create for the other training points, but we do know that this selected training point will be converted from an error to a match.

A single pass scoring run is performed using the newly altered coefficients. The resulting CEP and the altered coefficients are inserted into the swarm pool sorted by CEP. If the new CEP is an improvement then we repeat the MSM layer once again. If the new CEP is NOT an improvement then we decrement the MSM repetition count. If the MSM repetition count is greater

than zero then we repeat the MSM layer once again; otherwise, we terminate the MSM layer.

Upon termination of the final MSM repetition the final CEP is used to compute the MSM success rate as follows.

- (*E12*) MSMSuccessRate = 100%-CEP

The MSM success rate will be used in the next layer of the MDC algorithm to distribute evolutionary activity more efficiently.

2.3 Bees Swarm Optimization

The third layer of the MDC algorithm is the repeated application of the well known bee's swarm optimization algorithm often used in swarm evolution (Karaboga 2009).

At the start of the bees swarm optimization (BSO) layer, the candidate GPCM contains a swarm pool of several sets of coefficient constants which were produced by the MSM layer. Also the CEP for the most fit entry in the swarm pool and the MSM success rate are both available. The BSO layer multiplies the MSM success rate times 100 to produce the BSO repetition count, which determines the number of times the BSO will repeat without CEP improvement. The higher the MSM success rate, the greater the count of times the BSO will repeat. So the MSM success rate determines the evolutionary activity spent on the current candidate. Better candidates receive more evolutionary activity. Worse candidates receive less evolutionary activity.

For each repetition of the BSO, the standard bee's optimization algorithm is applied (Karaboga 2009). Then a single pass scoring run is performed using the new coefficient alterations. The resulting CEP and the altered coefficients are inserted into the swarm pool sorted by CEP. If the new CEP is an improvement then we repeat the BSO layer once again. If the new CEP is NOT an improvement then we decrement the BSO repetition count. If the BSO repetition count is greater than zero then we repeat the BSO layer once again; otherwise, we terminate the BSO layer.

At the end of the BSO layer, the coefficients of the current GPCM candidate are fully optimized *as far as the MDC algorithm is concerned.*

3 Computational Effort Distribution

In this section we create the several theoretical test problems which demonstrate the manner in which the MDC algorithm distributes computational effort as it attempts to optimize the coefficients of a candidate GPCM. All of these theoretical test problems attempt to optimize coefficients for a single

artificially created GPCM against a target which has been explicitly created to demonstrate test cases which are poorly matched versus test cases which are well matched. Using the following formula (E13), we will create a theoretical test data set using the following five discriminant formulas. Therefore we are trying to classify across K = 5 classes.

- (*E13*) y=nominal(3.4+(1.57*x0),2.1-(-39.34*(x4*x1)),(2.13*(x2/x3)),1.0-(46.59*(x3*x7)),(11.54*x4))

Next we will use the MDC algorithm to optimize coefficients for the following candidate GPCMs, each with five proposed discriminants.

- (*C01*) nomial(x0,x1,x2,x3,x4)
- (*C02*) nomial(x0,x1*x4,x2,x3,x4)
- (*C03*) nomial(x0,x1*x4,x2/x3,x3,x4)
- (*C04*) nomial(x0,x1*x4,x2/x3,x3*x7,x4)

The results of using the MDC algorithm to optimize each of the four test GPCMs, on the theoretical test data created with formula (E13), are shown in Table 1. Notice how the fitness Classification Error Percent (CEP) improves as the tests move closer to matching formula (E13). Also notice how the computational effort increases as the tests move closer to matching formula (E13). The MDC algorithm distributes more computational effort to the more promising candidates and less computational effort to the less promising candidates.

Table 1. MDC Evolutionary Effort

Test	*PBR*	*MSM*	*BSO*	*CEP*
C01	6	49	48	0.4550
C02	6	73	121	0.2187
C03	6	119	235	0.2033
C04	6	229	511	0.0000

(**Note1***: the number of PBR layer attempts is listed in the (PBR) column*) (**Note2***: the number of MSM layer attempts is listed in the (MSM) column*) (**Note3***: the number of BSO layer attempts is listed in the (BSO) column*) (**Note4***: the fitness score of the optimized GPCM on test data is listed in the (CEP) column*)

Deterministic algorithms, like M2GP, apply the same amount of computational effort toward optimizing the coefficients of each GPCM equally. Whereas the MDC algorithm distributes the computational effort unevenly throughout the entire evolutionary process with less promising candidates receiving less computational effort and more promising candidates receiving more computational effort.

4 Theoretical Test Problems

The MDC algorithm is NOT adequate for performing a whole symbolic multi-class classification run. MDC is only adequate to optimize the coefficients of a selected GPCM candidate. Therefore we will be required to develop a temporary symbolic multi-class classification strategy wrapped around the MDC algorithm in order to proceed with testing.

Since this is a preliminary paper in a planned series of papers investigating extreme accuracy algorithms vis a vis symbolic multi-class classification, we will create a first draft multi-class classification strategy as a temporary methodology to allow further testing. We do not yet know whether MDC is the most propitious coefficient optimization algorithm, or whether some hybrid of M2GP and MDC, or some other algorithm might be better. We are just getting started in our investigation.

Our temporary symbolic multi-class classification algorithm, to wrap around the MDC algorithm, will be composed of K+1 separate search islands - a general search island and a specific search island for each class. During the classification run, all K+1 search islands will use Pareto front optimization to select GPCM candidates, optimize their discriminant coefficients with MDC, then score the final CEP. Whenever a new global most fit GPCM is discovered - in any of the K+1 search islands - the K specialized search islands are reset to the new global most fit GPCM. Each kth specialized search island then attempts to further optimize the best GPCM by holding all discriminant functions fixed, EXCEPT the kth discriminant function which is evolved using Pareto front evolution.

Our temporary Pareto front search strategy surrounds the MDC algorithm with a general attempt to find the best GPCM (*search island 0*) and K attempts specific to each of the K classes (*search islands 1 thru K*) to find the best GPCM. No assertion is made that this temporary Pareto front strategy will be the best or final search strategy achieving extreme accuracy in symbolic multi-class classification. It is just a temporary search strategy to allow us to proceed with testing.

In this section we create four theoretical training and separate test data sets using discriminant formulas (T1) thru (T4). Each theoretical test problem has K=5 classes. The following tables shows the results for each of the four symbolic classification runs.

- **Theoretical Test Problems**
- (*T1*): y=nomial(1.57*x0,-39.34*x1,2.13*x2,46.59*x3,11.54*x4)
- (*T2*): y=nomial(3.4+(1.57*x0),2.1-(-39.34*(x4*x1)),2.13*(x2/x3),1.0-(46.59*(x5*x5)),11.54*x4)
- (*T3*): y=nomial(3.4+(1.57*x0),2.1+(-39.34*minimum(x4*x1,x6)),2.13*((x2/x3)<=(x7)),1.0-(46.59*(x5*x9)),11.54*x4)
- (*T4*): y=nomial(3.4+(1.57*(x0<=x11)),2.1+(9.34*minimum(x4*x1,x6)),2.13*maximum((x2/x3)<=x7,x19)(46.59*maximum(x15,x8)),11.54*x4)

Table 2. MDC Theoretical Test Problems Results

Test	*WFFs*	*Train-Hrs*	*Train-CEP*	*Test-CEP*
T01	5K	0.59	0.0000	0.0000
T02	10K	2.011	0.0398	0.0411
T03	71K	10.08	0.0606	0.0810
T04	76K	10.08	0.0964	0.0976

(**Note1***: the number of regression candidates tested before finding a solution is listed in the Well Formed Formulas (WFFs) column*) (**Note2***: the elapsed hours spent training on the training data is listed in the (Train-Hrs) column*) (**Note3***: the fitness score of the champion on the noiseless training data is listed in the (Train-CEP) column*) (**Note4***: the fitness score of the champion on the noiseless testing data is listed in the (Test-CEP) column with* **.0880 average fitness**)

The theoretical testing demonstrates the computational ease of using the MDC algorithm for GPCM coefficient optimization. Training is quick and grows only linearly as the number of classes and training points grow larger. The first simple test (T1) achieves extreme accuracy because the temporary Pareto front strategy selected an exact match GPCM during the run. As one would expect the CEP fitness scores get worse as the target discriminant functions grow more mathematically complex (*further from linear*). This is the result of the temporary Pareto front strategy not selecting exact match GPCMs during the run. A problem which will have to be addressed in the future papers if we are to achieve extreme accuracy in symbolic multi-class classification.

5 Real Data Test Problems

In this section we apply the MDC algorithm wrapped in its temporary Pareto front strategy on real world test problems taken from several sources. Some test data sets were downloaded from the University of California at Irvine machine learning repository *https://archive.ics.uci.edu/ml/datasets.html*. Other test data sets were downloaded from the Broad Institute cancer data sets *http://www.broadinstitute.org/cgi-bin/cancer/datasets.cgi*.

Another Volatility data set was constructed from the Yahoo down loadable VIX and UVXY daily historical data sets. This test problem attempted to classify the next day's profit or loss in the UVXY ETF, entirely from the previous day's percent change in the VIX and the percent change in the 140 day moving average of the VIX.

- **Real Data Test Problems**
- (*R01*): Acute Myeloid Leukemia (*from Broad Institute*)
- (*R02*): Iris (*from UCI*)
- (*R03*): Heart Disease (*from UCI*)
- (*R04*): Volatility (*from Yahoo VIX and UVXY historical data*)
- (*R05*): Bank Marketing (*from UCI*)

The results of running the MDC algorithm wrapped in its temporary Pareto front strategy on these real world test problems is shown in the table below.

Table 3. MDC Real Data Test Problems Results

Test	*WFFs*	*Train-Hrs*	*Train-CEP*	*Test-CEP*
R01	9K	0.24	0.0277	0.0000
R02	10K	0.11	0.0133	0.0134
R03	548K	10.00	0.0912	0.0925
R04	9K	0.14	0.0704	0.0705
R05	9K	0.29	0.1077	0.1097

(**Note1***: the number of regression candidates tested before finding a solution is listed in the Well Formed Formulas (WFFs) column*) (**Note2***: the elapsed hours spent training on the training data is listed in the (Train-Hrs) column*) (**Note3***: the fitness score of the champion on the noiseless training data is listed in the (Train-CEP) column*) (**Note4***: the fitness score of the champion on the noiseless testing data is listed in the (Test-CEP) column with* **.0681 average fitness**)

This early in our planned investigation of extreme accuracy in multi-class classification, we did not expect to achieve the results shown above. While we are very far from any industrially usable techniques, the MDC algorithm

wrapped in its temporary Pareto front strategy achieved a perfect CEP score on the Broad Institute's leukemia data. The UCI Iris data also got a very good CEP score. All other real world test data sets got reasonable CEP scores. The Volatility test data achieved a reasonable CEP score and categorized, without any losses, the day's when the UVXY next day profit was 20% or above. There were four estimated trading signals all of which resulted in next day UVXY profits of 20% or more.

6 Conclusion

In a previous series of papers (Korns 2011-2015), significant accuracy issues were identified for state of the art symbolic regression systems and a comprehensive multi-island strategy for achieving extreme accuracy on a large well defined set of theoretical problems was developed and tested both in noiseless and in noisy training environments. Unfortunately these SR techniques do not translate directly into an algorithm which will achieve extreme accuracy on symbolic multi-class classification problems.

A first step in a planned investigation of extreme accuracy in symbolic multi-class classification is taken with the introduction of the Multilayer Discriminant Classification algorithm for optimizing the discriminant coefficients in a multi-class discriminant equation. Distribution pof computational effort Tests on the MDC algorithm demonstrate the desired properties of a high level of accuracy for exact match GPCMs and also well behaved distribution of computational effort with more promising GPCM candidates receiving more computational effort and less promising GPCM candidates receiving less computational effort.

Also both theoretical and real world testing of the MDC algorithm, *wrapped in a temporary Pareto front strategy*, resulted in preliminary but promising CEP scores.

It is now a reasonable suspicion that the same symbolic regression accuracy issues, *due primarily to the poor surface conditions of specific subsets of the problem space*, are also present in and obstructing extreme accuracy in symbolic multi-class classification problems.

Future research must explore the possibility of developing an Extreme Accuracy algorithm for the field of symbolic multi-class classification. Furthermore any such extreme accuracy algorithm would ideally be accompanied by a formal or informal proof of extreme accuracy on a well defined set of theoretical problems.

Finally, to the extent that the reasoning in even an *informal argument* of extreme accuracy can gain academic and commercial acceptance, a climate of *belief* in symbolic multi-class classification can be created wherein SC is increasingly seen as a "**must have**" tool in the scientific arsenal.

Truly knowing the strength's and weaknesses of our tools is an essential step in gaining trust in their use.

References

1. M. Castelli, S. Silva, L. Vanneshi, A. Cabral, M. Vasconcelos, L. Catarino, J. Carrieras (2013). Land cover/land use multiclass classification using gp with geometric semantic operators, in *EvoApplications 13*, pages 334-343, Berlin Springer.
2. Gregory S Hornby (2006). Age-Layered Population Structure For reducing the Problem of Premature Convergence, in *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation.* ACM Press, New York.
3. V. Ingalalli, S. Silva, M. Castelli, L. Vanneschi (2014). A Multi-dimensional Genetic Programming Approach for Multi-class Classification Problems, in *Euro GP 2014*. Springer.
4. D Karaboga, B Akay (2009). A survey: algorithms simulating bee swarm intelligence, in *Artificial Intelligence Review*. Springer, New York.
5. Michael Korns (2010). Abstract Expression Grammar Symbolic Regression, in *Genetic Programming Theory and Practice VIII*. Springer, New York. Kaufmann Publishers, San Francisco California.
6. Michael Korns (2011). Accuracy in Symbolic Regression, in *Genetic Programming Theory and Practice IX*. Springer, New York. Kaufmann Publishers, San Francisco California.
7. Michael Korns (2012). A Baseline Symbolic Regression Algorithm, in *Genetic Programming Theory and Practice X*. Springer, New York. Kaufmann Publishers, San Francisco California.
8. Michael Korns (2013). Extreme Accuracy in Symbolic Regression, in *Genetic Programming Theory and Practice XI*. Springer, New York. Kaufmann Publishers, San Francisco California.
9. Michael Korns (2014). Extremely Accurate Symbolic Regression for Large Feature Problems, in *Genetic Programming Theory and Practice XII*. Springer, New York. Kaufmann Publishers, San Francisco California.
10. Michael Korns (2015). Highly Accurate Symbolic Regression for Noisy Training Data, in *Genetic Programming Theory and Practice XIII*. Springer, New York. Kaufmann Publishers, San Francisco California.
11. Mark Kotanchek, Guido Smits, and Ekaterina Vladislavleva (2008). Trustable Symbolic Regression Models: Using Ensambles, Interval Arithmetic and Pareto Fronts to Develop Robust and Trust-Aware Models, in *Genetic Programming Theory and Practice V*. Springer, New York.
12. John R Koza (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press, Cambridge Massachusetts.
13. John R Koza (1994). Genetic Programming II: Automatic Discovery of Reusable Programs. The MIT Press, Cambridge Massachusetts.
14. John R Koza, Forrest H Bennett III, David Andre, Martin A Keane (1999). Genetic Programming III: Darwinian Invention and Problem Solving. Morgan
15. W Langdon, R Poli (2002). Foundations of Genetic Programming. Springer
16. J Platt, (1998). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. *Technical Report* ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING.
17. Trent McConaghy, (2011). FFX: Fastm Scalable, Deterministic Symbolic Regression Technology, in *Genetic Programming Theory and Practice IX*. Springer, New York.

18. L. Munoz, S. Silva, L. Trujillo (2015). M3GP Multiclass Classification with GP, in *Euro GP 2015*. Springer.
19. J.A., Nelder, and R. W. Wedderburn (1972). Generalized linear Models, in *Journal of the Royal Statistical Society*, Series A, General, 135:370-384.
20. Poli, Riccardo, McPhee, Nicholas, Vanneshi, Leonardo, (2009). Analysis of the Effects of Elitism on Bloat in Linear and Tree-based Genetic Programming, in *Genetic Programming Theory and Practice VI*. Springer, New York.
21. Poli, Langdon, McPhee, (2008). A Field Guide to Genetic Programming.
22. Gandomi, Alavi, Ryan, (2015). Handbook of Genetic Programming Applications. Springer, Zurich Swizerland.
23. Guido Smits, and Mark Kotanchek (2005). Pareto-Front Exploitation in Symbolic Regression, in *Genetic Programming Theory and Practice II*. Springer, New York.
24. Michael Schmidt, Hod Lipson (2010). Age-Fitness Pareto Optimization, in *Genetic Programming Theory and Practice VI*. Springer, New York.