Building Multiclass Nonlinear Classifiers with GPUs

Ignacio Arnaldo
CSAIL, MIT
Cambridge, MA 02139
iarnaldo@mit.edu

Kalyan Veeramachaneni
CSAIL, MIT
Cambridge, MA 02139
kalyan@csail.mit.edu

Una-May O'Reilly
CSAIL, MIT
Cambridge, MA 02139
unamay@csail.mit.edu

Abstract

The adoption of multiclass classification strategies that train independent binary classifiers becomes challenging when the goal is to retrieve nonlinear models from large datasets and the process requires several passes through the data. In such scenario, the combined use of a search and score algorithm and GPUs allows to obtain binary classifiers in a reduced time. We demonstrate our approach by training a ten class classifier over more than 400K exemplars following the exhaustive Error Correcting Output Code strategy that decomposes into 511 binary problems.

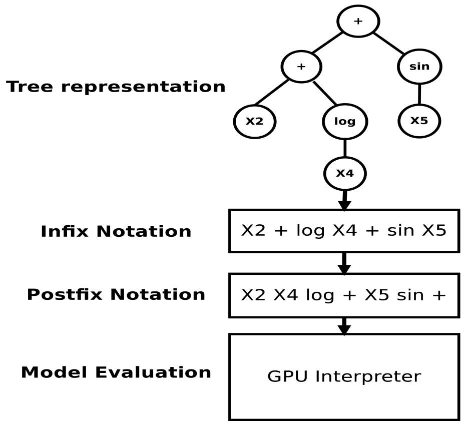
1 Introduction

In this paper, our focus is to design a nonlinear function of the features $f(\bar{X})$ such that the resulting distributions of the function output when conditioned on the two classes $p(f(\bar{X})|H_0)$ and $p(f(\bar{X})|H_1)$ are best separated. This approach is extremely beneficial when the variables are believed to have non-linear relationships when conditioned upon a class. Given a set of non-linear operators, the optimization to obtain the best possible non-linear function for binary classification requires a search and score methodology that performs several passes over the data.

An example search and score algorithm, known as Genetic Programming (GP) [6], relies on a population of models or *individuals*, generally represented with trees (see Figure 1). These trees are iteratively improved by means of an evolutionary sampling process. The algorithm evaluates a population of models in each iteration, each requiring the evaluation of the non linear function on the entire dataset. Since the goal is to find the model with highest discriminatory power between the two classes, the area under the ROC curve is used as cost function to guide the search. This adds additional complexity because each point on the ROC curve is obtained by applying a moving decision threshold to the non-linear functions output and evaluating the type I and type II errors.

Finally, most real world problems are multi-class classification problems that are often tackled by training a series of binary classifiers and combining their predictions to classify new exemplars. With increasing number of classes and data size, the training time of these search and score approaches can overwhelm a CPU-based system.

This work focuses on the exploitation of Graphics Processing Units (GPUs) to build these non-linear classifiers. To achieve this we first decompose a non-linear function into a sequence of functions and evaluate them on the GPUs. We then obtain their areas under the ROC curves in a massively parallel fashion. Our approach thus enables the search and score methodology to evaluate many models at really fast speeds. Reduced time for building a single classifier allows us to build many classifiers sequentially to solve multi-class classification problems following the strategies of One-vs-All and Error Correcting Output Codes (ECOC).



```
function CUDAINT($E_{postfix}$)
    loadVariablesX(threadId)
    while not end of expression do
        read token from expression
        if token is a variable then
            push
        else if token is operator then
            pop right operand
            if unary operator then
                evaluate(right)
            else if binary operator then
                pop left operand
                evaluate(left,right)
            end if
            push result
        end if
    end while
    pop result
    output[threadId] = result
end function
```

Figure 1: GP Tree, infix, and postfix forms (left) and pseudocode of the postfix interpreter (right)

2 Designing a non-linear classifiers with GPUs

Given a binary classification problem and a training data D, we first split the training data into D_T and D_λ. Then, we employ a search and score methodology Genetic Programming to *iteratively* search for a nonlinear function $f(\bar{X})$ such the distributions $p(f(\bar{X})|H_0)$ and $p(f(\bar{X})|H_1)$ are best separated with respect to D_T . The objective function for the Genetic programming is the maximization of the area under the ROC curve. The adopted approach presents a major challenge since it requires $p * g$ passes over the data, where p is the number of models maintained during an iteration and g represents the number of iterations. The elevated learning cost can be dramatically alleviated with GPUs that enable many passes over large datasets in a reduced time. The computation required to score a model is broken in two steps.

Step 1: Evaluation of non linear model with GPUs

In this step, non-linear model $f(\bar{X})$ is evaluated for the n data points in D_T resulting in $y_{1...n}$. We implement a GPU interpreter inspired on [4], [5], and [8] capable of evaluating any possible arithmetic expression in postfix notation. As depicted in Figure 1, several steps are required to obtain the postfix expression from a GP tree. We first traverse the tree in a depth-first in-order manner to generate the respective infix expression. We then apply Dijkstra's Shunting Yard algorithm [3] to obtain the postfix notation. The interpreter will evaluate the expression and produce an output value for each data point in the dataset. This task is computed via GPUs for datasets of size hundreds of thousands or even millions of independent test cases.

In our GPU implementation, a CUDA thread is declared for each data point in the dataset. Thus multiple CUDA threads will execute the interpreter function shown in Figure 1 simultaneously on different data points, ensuring that all threads will follow the exact same execution path. Note that conditional instructions such as *if* or *while* statements are pernicious for the performance of CUDA programs only when they trigger a divergence in the execution of threads within a *warp*. In such case, their execution is serialized. To benefit from coalesced memory accesses, we transpose the input matrix before storing it in global memory in such way that exemplars are displayed in columns while each line corresponds to an explanatory variable. This way, contiguous threads will access adjacent memory positions, thus reducing the number of expensive global memory accesses. At the end of this step we obtain the output of the model $y_{1...n}$ for all the n data points in D_T.

Step 2: Computation of the area under the ROC curve

To compute the area under the ROC curve, we need to vary the threshold λ in the decision rule $\hat{L}_i = \left\{ \begin{smallmatrix} 1, \, if \; y_i \geq \lambda \\ 0, \, if \; y_i < \lambda \end{smallmatrix} \right\}$ that determines whether a given output represents a class 0 or class 1 prediction. For each threshold and data point, we obtain the predicted class label. We evaluate the two errors and compute the model's area under the ROC curve. We proceed as follows:

a: Retrieve the maximum y_{max} and minimum y_{min} values for the model outputs $y_{1...n}$.
b: Normalize the outputs of the model with the obtained boundaries.
c: Vary the threshold $\lambda \in [0;1]$ and apply the decision rule as above.
d: Obtain the False Positive and True Positive rates for each value of λ
e: Compute the area under the ROC curve with the obtained rates.

a, b, c, d are computed in the GPU while e is computed in CPU. We accomplish very high speeds in these computations by exploiting a a very high level of parallelism. In (a), a CUDA parallel reduction is employed to obtain the maximum and minimum values. The GPU is also used to normalize the output values and a third kernel computes the predictions of a model for a given threshold λ. Finally, a parallel reduction is employed again in (e) to count the true and false positives. Once (e) is finished the computation moves on to the next model and repeats Steps 1 and 2.

Step 3: Model and operating point selection During the search process, the best model of each iteration is saved. Once the search is finished, we select from the best G models (one per iteration). Then for each model we follow steps 1 and 2 and evaluate its area under the ROC curve for the the model given the unseen data D_λ. We select the model with the highest area under the ROC curve. This allows us to control for over fitting.

Given the model $f(\bar{X})$ and D_λ the last step consists in identifying the the threshold λ in the decision rule $\hat{L_i} = \left\{ \begin{smallmatrix} 1, \; if \; y_i \geq \lambda \\ 0, \; if \; y_i < \lambda \end{smallmatrix} \right\}$ such that the Bayesian risk function given by

$$\left\{ \begin{array}{l} R = P(H_0) C_{H_1|H_0} P(H_1|H_0) + P(H_1) C_{H_0|H_1} P(H_0|H_1) \\ P(H_i|H_j) = \frac{|\hat{L}=i|L=j|}{|L=j|} \end{array} \right. \tag{1}$$

is minimized, where $C_{H_i|H_j}$ represents the cost associated with declaring class i when the true class is j, $P(H_i)$ is the prior of class i and $P(H_i|H_j)$ are the type I and type II errors. This is achieved by performing a grid search over λ. This process is carried out post-hoc after the classifier is trained.

Step 4: Extending to Multi class problem For a multi class problem, we repeat the steps 1, 2, 3 sequentially each time changing the binary classification problem we are attempting to solve. Since the training time for each non-linear classifier is brought down significantly this process is still manageable on a desktop in a reasonable amount of time.

3 Experimental setup and results

The proposed approach is demonstrated with a modified version of the Million Song Dataset year prediction challenge [1], generally described as a regression problem. We change this problem to be one in which the goal is to predict the decade in which a given song was released. As shown in Table 1, the resulting problem is a highly imbalanced ten class classification problem. The dataset is divided into D_T, D_λ, and D_{test} accounting for 70%, 10%, and 20% of the data respectively. Note that the *producer effect* issue has been taken into account to perform the split. To compute the areas under the ROC curves, 11 points are obtained by moving the threshold λ in $[0;1]$ with a step of 0.1. All the experiments are run on the same computer, equipped with a NVIDIA Geforce GTX 690 that counts 1536 CUDA cores.

3.1 One-vs-All Strategy

We first adopt the one-vs-All approach, which decomposes into 10 problems in a one-versus-rest fashion, to attack the decade prediction problem. The reduced number of binary problems allows us to compare the execution time of an optimized C++ approach running on CPU and the proposed GPU

1920	1930	1940	1950	1960	1970	1980	1990	2000	2010	Total
225	252	356	3104	11741	24748	41827	124808	299117	9386	515564
0.04%	0.05%	0.07%	0.60%	2.28%	4.80%	8.11%	24.21%	58.02%	1.82%	100%

Table 1: Class Distribution of the MSD decade prediction challenge

	Tr time	speedup
Opt. C++	136m31s	1×
CUDA	6m 40s	20.46×

Table 2: Training time per binary classifier and speedup obtained with the GPU implementation

approach	accuracy	Tr time	model eval.	num classif.
DT	0.494	4m12s	-	1
OVA	0.430	4m2s	2×10^{10}	10
ECOC	0.579	4m56s	2×10^{10}	511
ECOC$_f$	0.598	4m56s	2×10^{10}	511

Table 3: Accuracy, average training time and model evaluations per classifier, and number of trained classifiers

implementation introduced in the previous section. The seed employed to generate random numbers is fixed to ensure the fairness of the comparison. Each of the GP instances (one per binary problem) is run with a population of 1000 models and a generation limit of 100. Table 2 presents the training times per binary classifier of the two implementations and the speedup obtained with the GPU implementation. Note that the optimized CPU implementation already provides a 15× speedup [7] as compared to the standard model evaluation generally adopted in Genetic Programming.

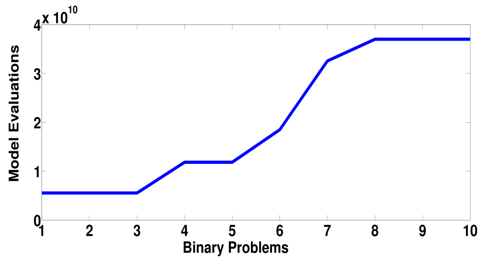
3.2 Exhaustive Error Correcting Output Code

We implement the exhaustive Error Correcting Output Code (ECOC) [2] strategy that decomposes a k class classification problem into $2^{k-1} - 1$ binary problems. The referred work proposes to reduce the number of binary problems when the number of classes k is greater than seven. However, we demonstrate that our approach can tackle problems with an elevated number of classes by adopting the exhaustive approach and training all the 511 binary classifiers. GP runs are stopped if convergence is reached or if a model presents an area under the ROC curve greater than 0.95.

Table 3 shows the accuracy, the average training time and number of model evaluations per classifier, and the number of trained classifiers when approaching the decade prediction problem with a Decision Tree (DT), One-vs-All (OVA), Error Correcting Output Codes (ECOC), and (ECOC$_f$). The latter is a refined version of the Error Correcting Output Codes strategy that filters the binary predictors that present an area under the ROC curve less than 0.7 with respect to D_λ. The number of model evaluations is a factor of the exemplars in the training set (370K), the population size (1000), and the average number of iterations iterations (55.7 for OVA and 57.4 for ECOC). As shown in Figure 2, the number of model evaluations varies from a GP run to the other due to the stop criteria.

4 Conclusions

We have introduced a methodology that allows to retrieve nonlinear multiclass classifiers from large datasets. The training procedure is decomposed into a series of binary classification problems, each approached as the search of the nonlinear function $f(\bar{X})$ that best separates the distributions of the two classes $p(f(\bar{X})|H_0)$ and $p(f(\bar{X})|H_1)$. Genetic Programming is employed to perform the search in the binary classifier space by targeting the maximization of the area under the ROC curve. The comparison against an optimized CPU model evaluation has shown that the proposed GPU implementation provides a 20.46× speedup, enabling our approach to iterate through the data in a reduced time. In fact, a total of 1×10^{13} model evaluations are performed when adopting the exhaustive ECOC strategy to train a classifier on a 10-class dataset composed of 370K exemplars.



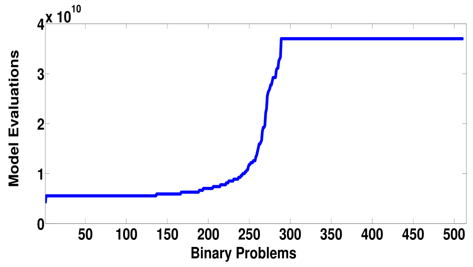


Figure 2: Model evaluations per binary problem with OVA (left) and ECOC (right)

References

[1] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[2] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *J. Artif. Int. Res.*, 2(1):263–286, January 1995.

[3] E. W. Dijkstra. Algol 60 translation. *Supplement, Algol 60 Bulletin*, 10, 1960.

[4] W. B. Langdon and Wolfgang Banzhaf. A SIMD interpreter for genetic programming on GPU graphics cards. In *Proceedings of the 11th European conference on Genetic programming*, EuroGP'08, pages 73–85, Berlin, Heidelberg, 2008. Springer-Verlag.

[5] Denis Robilliard, Virginie Marion-Poty, and Cyril Fonlupt. Population parallel GP on the G80 GPU. In Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, AnnaIsabel Esparcia Alcázar, Ivanoe Falco, Antonio Cioppa, and Ernesto Tarantino, editors, *Genetic Programming*, volume 4971 of *Lecture Notes in Computer Science*, pages 98–109. Springer Berlin Heidelberg, 2008.

[6] Michael Schmidt and Hod Lipson. Symbolic regression of implicit equations. In Rick Riolo, Una-May O'Reilly, and Trent McConaghy, editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, pages 73–85. Springer US, 2010.

[7] Dylan J Sherry. FlexGP 2.0: Multiple levels of parallelism in distributed machine learning via Genetic Programming, 2013.

[8] G. Wilson and W. Banzhaf. Linear genetic programming GPGPU on Microsoft Xbox 360. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 378–385, 2008.