PharmaSUG 2020 - Paper XX-25

Gradient Descent Using SAS™ for Gradient Boosting

Karen E. Walker, Walker Consulting LLC Chandler Arizona

README

Please don't be put off by the amount of math equations here; they are used for illustration only. Further explanation can be easily found in any calculus or linear algebra text book as same are for the DOT PRODUCT, and CHAIN RULE. This also goes for backwards propagation of the linear regression equation. SAS™ PROCS: **LOGSELECT, FOREST, GRADBOOST**, **PROC ADAPTIVREG** are found in SAS™ Viya installation of SAS™ Software.

ABSTRACT

Much of applied mathematics is centered on equations and evaluation of ERRORS in effort to reduce them during computations. Optimization of ERRORS is the focus of this topic. This paper will illustrate mathematics equations applied to thousands of calculations. Then show algorithms that are proven to optimize these calculations through scoring. One such algorithm is a gradient descent. It will show ways to reduce the errors by letting the gradient approach zero and adjusting for that. Next, this paper will use the gradient descent SAS™ program to find the smallest ERROR when matching two models and subsequently introduce the **SAS™ GRADBOOST** procedure. Finally it will prove that more precision is achieved using gradient descent with gradient boosting algorithms and show why this is so important to machine learning. Scientific methods like: Machine Learning and Deep Learning utilize mathematical functions that are rendered to algebraic computations are then programmed. Through "back propagation" of equations that compute directional derivatives, finite differences between models are scored. When we know the difference between equations, we can use that information to estimate which is more correct. That difference, if allowed to approach to a minimum enables a computer program to think without human intervention. In this paper we'll discuss gradient descent to demonstrate how machine learning programs are broken down, executed, and the data is stored. The stored data is passed iteratively to Deep Learning programs that make artificial intelligence possible.

INTRODUCTION

Scientific methods for machine learning begin with a mathematical modeling. Models contain variables, equations, possibly some expectations of how the model will perform, past data, categorical or conditional assignments, and groupings. Data scientist, statisticians, application developers and the like will use the tools described herein to train, evaluate, and predict their models.

The purpose and scope of this paper is to optimize models for machine learning using SAS™ PROCS, data step, estimates, optimization algorithms, and graphs.

Machine Learning applications are powerful tools that can solve problems before they are known to us. Machine learning draws from statistics, calculus, linear algebra and optimization theory for functions. To develop effective tools we must concentrate on modeling activation functions and normalization routines for them. The combination of activation functions working with normalization routines are known as "neural networks". Well known models for machine learning include: mathematical functions of shapes, artificial neural networks, convolutional neural networks, and recurrent neural networks. Neural networks are designed to collect data, and then use the data to build instructions. Both the results and ERRORS are collected from the instructions to be examined by how close they are to a given expectation. As the Machine Learning Application finds patterns in the data it passes them to the activation function. The normalization routines adjust for ERRORS then loop back and subsequently optimize the activation function as more data is passed to it. For example consider the activation function y = mx+b. Many will recognize this function as the general model for the slope of a line. In our world we know where our activation function begins, and we can guess where it's going. So we initialize y=, x=, and use that to calculate m, let b do what it do. Now the question as we step through moment by moment of our actual values for X and Y, how close is our guess to what is observed? The Loss is the difference between our observed points on the slope of the line and our guess for N number of points.

$$\text{“ loss } = \frac{1}{N}\sum_{i=0}^{N-1}(y_{step} - (mx + b))^2 \text{ “}$$

x=point along the x-axis i=1,2,3….N N=number of points y_{step}=observed data

m=slope of the line, b=some constant.

This is the statistical equation for the mean squared error.

So we feed forward points on a line for both y and x and collected values that can be used to improve our activation model by decreasing the loss between what is observed and our initial guess. Improving the model by reducing the loss is called optimization. The simplest way to improve the activation model by decreasing the loss is called Gradient Descent.

Rendering Mathematical formula to PSEUDO Code

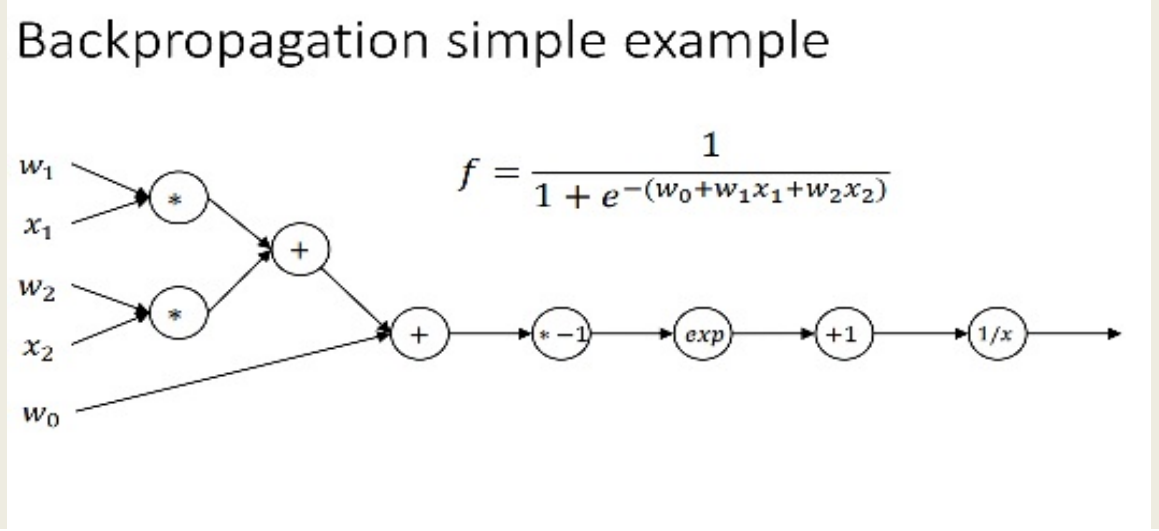
Numerical differentiation uses first partial derivative of z = f(x,y) with respect to x and to y: $\frac{\partial z}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x+\nabla x, y) - f(x,y)}{\Delta x}$ while (y is held constant).

Algorithmic differentiation is using back propagation of linear regression function: $f = \frac{1}{1+e^{-(w_0+w_1x_1+w_2x_2)}}$

Mathematical Formula	**Backwards Propagation PSEUDO CODE**
Z = wx + b	t_1=wx
Y = $\frac{1}{1+\exp(-z)}$	t_2=t_1 + b
Lambda = $\frac{1}{2}(y - t)^2$	t_3=-t_2
	t_4= exp(t_3)
	t_5=1 + t_4
	t_6=$\frac{1}{t_5}$
	t_7=t_6-t
	t_8=$t_6{}^2$
	t_9=$\frac{8}{2}$

Think of backwards propagation as using the hierarchy of mathematical operators on an equation in reverse. Do the multiplications for :w_1 times x_1 and w_2 times x_2 values then add and loaded to variable "wx" and set to " t_1". Next Add w_0 to wx and set to variable " t_2". Multiply by negative 1 and set to variable "t_3". Take the exponent of "t_3" and set to " t_4". One plus " t_4" is "t_5". The reciprocal of " t_5 " is set to " t_6". For Lambda adjust for a point in time "t". The difference is t_7, the square is t_8, one half is t_9.

This is how computational differentiation or automatic differentiation breaks down equations for processing. Automatic differentiation (AD) is a way to get these gradients efficiently without having to do anything but write the objective function in computer code. Automatic differentiation can be implemented in a variety of ways, via run-time abstractions and also via source code transformation.



GRADIENT DESCENT ALGORITHM

The Gradient Decent for a given function is found at a point expressed as…

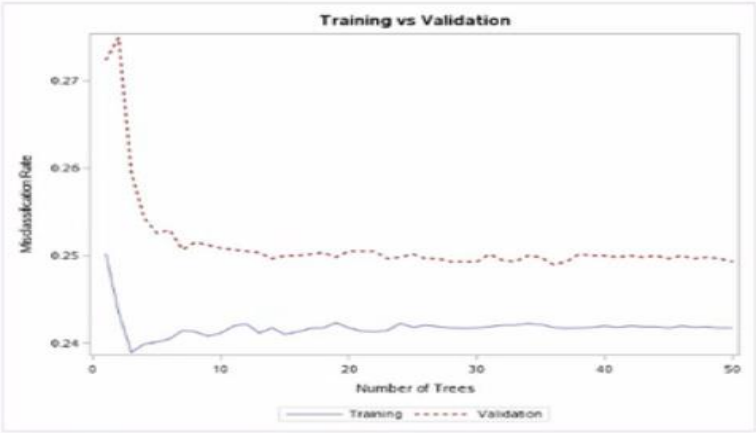
$$\nabla f(x,y,z) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right]$$

Let's check out an example… just a bit more complicated. Let's take what we know for continuous functions from calculus involving a function f(x,y,z). A point equals the derivative with respect to x, and a point equals the derivative with respect to y and a point equals the derivative with respect to z. A function decreases fastest at a point in the direction determined by it negative gradient at that point$\left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right]$. To find all the partial derivatives for x, y, and z. SAS™ gives us proc **LOGSELECT** so we can build the logistic regression model. For our neural network we need to build two sets: 1 for training, and 1 for validation. Proc **LOGSELECT** provides an option "partition rolevar=partind_1 for train and partition rolvar=partind_0 to validate. The goal of training is to determine which variable produced the smallest lost and select it.

```
proc logselect data=mycaslib.pkparms_raw_partind;
class target_b &class_var.;
model target_b (event='1')=&class_var. &interval_var.;
selection method=forward /*(choose=validate stop=validate)*/
partition rolevar=_partind_(train='1' validate='0')
code file="&outdir./logselect1.sas";
output data=mycaslib.logselect_scored copyvars=(_ALL_);
ods output FitStatistics=fitstats(rename=(Ntrees=Trees));
run;

/* Random Forest Modeling */
proc forest data=mycaslib.pkparms_raw_partind ntrees=50 intervalbins=20
minleafsize=5;
input &interval_var. / level = interval;
input &class_var. / level = nominal;
target target_b / level = nomial;
partition rolevar=_partind_ (train='1' validate='0');
code file="&outdir./forest1.sas";
output data=mycaslib.forest_scored copyvars=(_ALL_);
ods output FitStatistics=fitstats(rename=(Ntrees=Trees));
run;
```

Partial results are done one by one for x,y,z, each lost is computed through a thread of multi-threaded processes. Therefore we have also validate and train threads for x, y, and z.



Partial Results of the lost between training verses validation results.

GRADIENT BOOSTING

```
/* Score the data using the generated model */
data mycaslib.pkparms_scored_forest;
set mycaslib.pkparms_raw_partind;
%include "&outdir./forest1.sas";
p_target_b0=fdaemea_; /*training verses validation*/
run;
```

This algorithm computes the gradients of the loss and updates the model's variables (x,y,z) until the gradient of the loss equals = zero.

The values of the model's variables at a point in time are denoted as θ_t where theta "θ" is the set of guesses.

The difference between the observed points and the guesses over time are made into a new function of theta. Let's call it "diff(θ)".

The gradient of that new function optimizes it so the function will converge to a minimum is called;

"$\nabla diff(\theta)$"

The Learning rate; let's call this "θ_{diff}" is the step to step collection of difference results.

The step by step optimization equation is given by…

$$\boldsymbol{\theta_t}=\boldsymbol{\theta_{t-1}} - \theta_{diff}\boldsymbol{\nabla diff(\theta)}$$

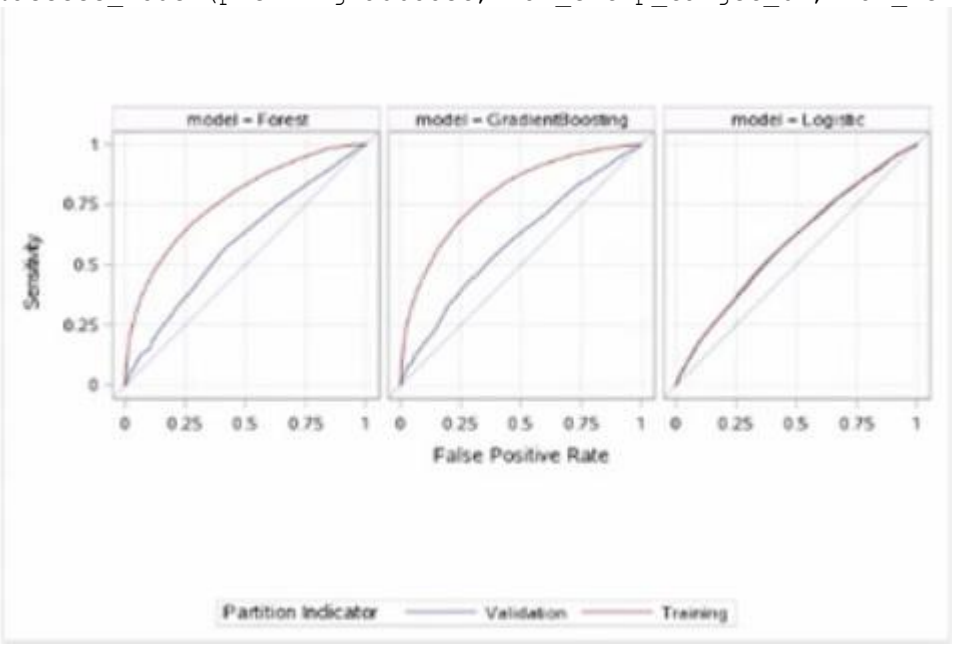
Now that we know what it is, how do we use it? In a perfect world we will always find convergence at a point when the derivative of the activation functions reaches their minimum. If diff(θ) is too large this algorithm could fail because it over-shoots the minimum and never converges. If diff(θ) is too small it could find multiple minimum values everywhere, and fail to converge to the right point. How about those functions not continuous and break over periods where a pattern is difficult to compute? If we consider stochastic functions that go on and on over time, with all the peaks and valleys, where can we find the true minimum? Not only are they slow to converge but may cause an infinite loop trying to find the point closest to zero without going negative. For all these cases we have to "BOOST" the gradient descent algorithm. In other words modify the equation in certain ways by its partial derivatives. SAS™ comes equipped with procedures that leverage dictionaries and boost performance like**: PROC GRADBOOST**, and **PROC ADAPTIVREG**. Notice that each time we re-score the model; we generate score code that can be used to score future data. **PROC ADAPTIVREG** provides the predictions only for a specified event level, so we calculate the non-event probability and add it to the scored data table. This score data are the brains (inputs) for our neural network application. Scored data is the aim of our topic of discussion today.

```
/*Score the data once again using the generated model */
proc gradboost data=mycaslib.pkparms_raw_partind ntrees=50 intervalbins=20
maxdepth=5;
input &interval_var. / level = interval;
input &class_var. / level = nominal;
target target / level=nominal;
partition rolevar=_partind_(train='1' validate='0');
code file="&outdir./gradboost.sas";
run;

/*Score the data using the generated model */
data mycaslib.pkparms_scored_gradboost;
set mycaslib.pkparms_raw_partind;
%include "&outdir./gradboost1.sas";
run;

%macro assess_model(prefix=, var_evt=, var_nevt=);
proc assess data=mycaslib.pkparms_scored_&prefix.;
input &var_evt.;
target target_b / level=nominal event='1';
fitstat pvar=&var_nevt. / pevent='0';
by _partind_;
ods output
```

```
fitstat=mylib.&prefix._fitstat
rocinfo=mylib.&prefix._rocinfo
liftinfo=mylib.&prefix._liftinfo;
run;
%mend assess_model;
%assess_model(prefix=logistic, var_evt=p_target_b, var_nevt=p_target_b0);
%assess_model(prefix=forest, var_evt=p_target_b1, var_nevt=p_target_b0);
%assess_model(prefix=gradboost, var_evt=p_target_b1, var_nevt=p_target_b0);
```




```
/*****************************/
/* ROC and Lift Charts       */
/*****************************/
ods graphics on;
proc format;
value partind1b1
0 = 'Validation'
1 = 'Training'
;
run;
data mylib.all_ROCinfo;
set mylib.logistic_ROCinfo (keep=sensitivity fpr _partind_ in=l)
    mylib.forest_ROCinfo(keep=sensitivity fpr _partind_ in=f)
    mylib.gradboost_ROCinfo(keep=sensitivity fpr _partind_ in=g);
length model $ 16;
select;
when (l) model='Logistic';
when (f) model='Forest';
when (g) model='GradientBoosting':
end;
run;
data mylib.all_liftinfo;
set mylib.logistic_liftinfo(keep=depth lift cumlift _partind_ in=l)
   mylib.forest_liftinfo(keep=depth lift cumlift _partind_ in=f)
   mylib.gradboost_liftinfo(keep=depth lift cumlift _partind_ in=g);
run;
```

VARIABLE METADATA

Given non-compartmental pharmacokinetic parameters: Area Under the Curve from time 0 to infinite (AUCinf), maximum concentration(Cmax), time at the Cmax(Tmax). What are the limits for: the derived parameters clearance/F, and volume of distribution/F and half-life? How far away from the prescribed path are these values? Based on the FDA and EMEA guidelines (trained data) what are the differences in values? What adjustments should we make to hit our target values? TEST formulation, TRAINING formulation, Validation FDA EMEA formulation. As per the relevant FDA and EMEA Guidelines, the statistical analysis should be based on the non- compartmental
Parameters AUCinf and Cmax, derived from the drug concentration-time curve where plasma is the matrix, with whole blood, or free concentration. These parameters are compared by means of an ANOVA in which the variance is partitioned into components due to subjects, periods and treatments. The validation is 90% confidence intervals for the mean difference of the parameters after logarithmic transformation should fall in the acceptance interval 0.8-1.25.

Table 1 is a sample table using NONMEM data format.

Always introduce a table by inserting a cross-reference. For instructions, see the section "**Error! Reference source not found.**."

Sourced Data	DB2 Data Type
VALIDATION or "fdaemea" = Nominal	VECTOR
TEST DATA= Interval	MATRICIES
TRAINING or "Reference"= Nominal score data	VECTOR

Table 1. DBLOAD Procedure: Default DB2 Data Types for SAS Variable Formats

```
PROC GRADBOOST
ods noproctitle;
proc gradboost data=MYCASLIB.PKPARAMS_DATA  ntrees=200 learningrate=0.01;
partition fraction(validate=0.4 seed=12345);
target FDAEMEA_acceptance_interval_values / level=nominal;
input
Subject AUCl AUCinf Cmax lnAUCl  lnAUCinf lnCmax, time_since_dose, Period, Treatment, Time, Concentration,
Dose
/ level=interval;
input AUCinf lnAUCinf  sd cv DF  Type IIISS MeanSquare  FValue  Pr _F
/ level=nominal;
run;
```

The Momentum Algorithm

If you need to include a numbered (ordered) list, copy the following list, paste it, and modify the text. Be sure to introduce your list with a complete sentence that ends with a colon. For example, "Here are the required steps:".

Let's attack the problem of oscillation of a pair of points and modify our gradient descent equation. To address it, consider the moment by moment we discussed before. As we step through time collecting and considering our next point. We can "look back" at previous point to determine how much influence the preceding step will be given to the current step. To do this we update our optimization equation to include interrogation to see if the previous step is a repeat of the last one, or two steps before or three steps before… Or how ever many steps we want to examine. Let's look at the math to fully appreciate all the hard work that has gone into development of these procs For added precision we can accumulate the results of θ_t over time. Let's call it"*cum*".

1. The gradient algorithm is modified as follows. $\boldsymbol{\theta_t = cum * \theta_{t-1}}$ - $\boldsymbol{\theta_{diff} \nabla diff(\theta)}$
2. If $cum * \theta_{t-1}$ is large the equation converges to a minimum quickly.
3. If the gradient descent was stuck oscillating $cum * \theta_{t-1}$ will reduce variable updates for subsequent iterations.

The Adaptive Gradient Descent Algorithm

If you need to include a bulleted (unordered) list, copy the following list, paste it, and modify the text. Be sure to introduce your list with a complete sentence that ends with a colon. For example, "Systems that are supported by the product include the following:".

Should we notice that the derivative of the activation function is undefined or taking lots of time and iterations to converge to it's minimum. Then we can boost the gradient descent function so as to adjust the learning rate as we step though iterations. For the steps that remain resistant to converge, we'll break out a new thread so that its partial derivative can be computed separately. A sub gradient is a generalization of a gradient that applies to non differentiable functions. The learning rate "θ_{diff}" can be changed to a two dimensional matrix involving time =t and iteration =i. Such that the ith element of the diagonal of the matrix formed by taking the outer product of the sub-gradient of the loss with itself. ALERT! Calculations now involve dot product, and chain rule.) . To make gradient updates with respect to θ, we're going to have to differentiate the objective with respect to θ and that will require the chain rule.

- The gradient algorithm is modified as follows: $\boldsymbol{\theta_{t,i} = \theta_{t-1,i}}$ - $\boldsymbol{\theta_{diff\ t,diff\ i} \nabla diff(\theta_i)}$
- Equations are generalized, broken apart and partial derivatives are taken to find their gradients.
- If a sub-gradient equation is non-differentiable if will be adapted to the closest library dictionary relative in the ith element in the 'outer product' of the sub-gradient against the loss equation.

Adaptive Momentum Gradient Descent Algorithm

Always introduce a display by inserting a cross-reference. For instructions, see the section "**Error! Reference source not found.**."

In cases where the previous step must be interrogated to gauge its influence on the current step we need to use the momentum optimization. However this same case resulted from a step that was broken away and is a partial derivative. It's situations like this when we boost a version of the gradient descent algorithm that accommodates values that change step by step at the same time interrogates "look ahead" results for undefined values. Separate equations into sub-gradients, at the same time deal with functions that are not differentiable. As thus use functions that are modifiable on the Euclidean plane by letting the MOMENTS be vectors. Making it Adaptive Moment Estimation.

- First moment = 1-β_1 gradient equation is $\boldsymbol{\theta_t = \beta_1 \theta_{t-1}}$ + (1-β_1) $\theta_{diff} \boldsymbol{\nabla diff(\theta)}$ set to $\boldsymbol{m_1}$
- Second moment = 1-β_2 gradient equation $\boldsymbol{\theta_t = \beta_2 \theta_{t-1}}$ + (1-β_2) $[\theta_{diff} \boldsymbol{\nabla diff(\theta)}]^2$ set to $\boldsymbol{m_2}$
- After some algebra the new step to step collection of difference results over time moment descent algorithm is. $\boldsymbol{\theta_{diff}} = \frac{\boldsymbol{\theta_{diff}} \sqrt{1-\beta_2^t}}{1-\beta_1^t}$
- And the gradient descent Algorithm is… $\boldsymbol{\theta_t = \theta_{t-1}}$ - $\frac{\boldsymbol{\theta_{diff} m_1}}{\sqrt{\boldsymbol{m_2}}+\varepsilon}$ where "ε" is some small value to prevent the denominator from being zero.
- Same as "Adaptive" Equations are generalized, broken apart and partial derivatives are taken to find their gradients.

- Same as “Adaptive” If a sub-gradient equation is non differentiable if will be adapted to the closest…

Stochastic Gradient Descent Algorithm (SGD)

For the situation where there are many points of converging as will a local minimum or a global minimum. The guess points must be split in to smaller buckets, and designated as local or global.

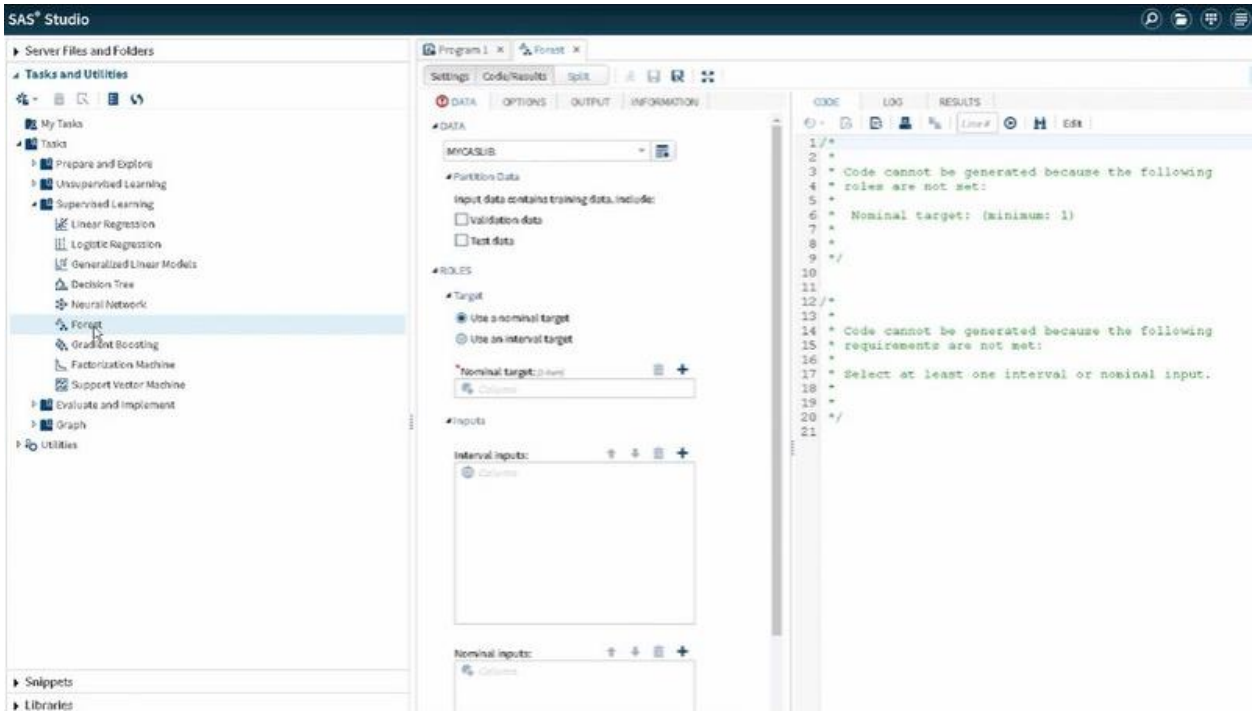
The buckets are shuffled randomly fed though the activation function to compute their derivatives. This forces the computer math co-processor to take larger jumps back and forth, and will improve the chances of finding a “global minimum”. This data is captured as index data and fed back through the activation function. For this kind of model the Maximum likelihood estimation is popular because of the jumping size random. We consider our buckets as category 0 and category 1. A typical SGD update rule would: 1) iterate and in each step take a small subset of the data (or even just a single example), compute the gradient of the loss, and then 2) take a step in the direction of the negative gradient.

- Computing loss with maximum likelihood estimation Probability of point being classified in category 1 is $\sigma(mx + b)$
- Probability of point being classified in category 0 is 1 - $\sigma(mx + b)$
- After some algebra the new step to step collection of difference results over time moment descent algorithm is. $\boldsymbol{\theta_{diff}} = \frac{\boldsymbol{\theta_{diff}} \sqrt{1-\beta_2^t}}{1-\beta_1^t}$
- And the gradient descent Algorithm is… $\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \frac{\boldsymbol{\theta_{diff}} m_1}{\sqrt{m_2}+\varepsilon}$ where "ε" is some small value to prevent the denominator from being zero.
- Same as “Adaptive” Equations are generalized, broken apart and partial derivatives are taken to find their gradients.
- Same as “Adaptive” If a sub-gradient equation is non differentiable if will be adapted to the closest library dictionary relative in the ith element in the ‘outer product’ of the sub-gradient against the loss equation.

RE-SCORING LOOP WITH LOOP-BACK FOR NEW DATA

```
data mycaslib.pkparms_score_data;
set mylib.pkparms_score_data;
run;
data mycaslib.pkparms_scored;
set mycaslib.pkparms_score_data;
if gradient not zero
%include “outdir./impute1.sas”;
%*include “outdir./gradboost_momentum.sas”;
If -1
%*include “outdir./gradboost_adapt.sas”;
%*include “outdir./gradboost_adaptmom.sas”;
%*include “outdir./gradboost_SGD.sas”;
run;
```

SUPERVISED MODEL TRAINNG WITH SAS™ STUDIO

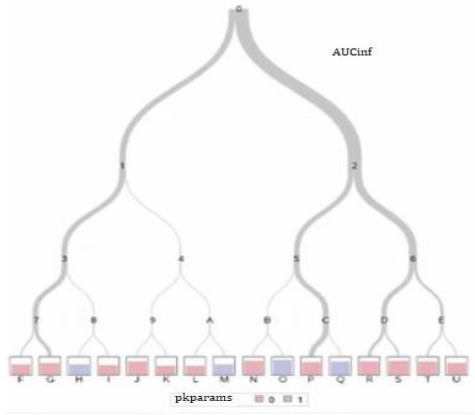


Display 1. SAS™ Studio interface

In SAS Studio you can develop a supervised MODEL and watch each iteration as it tunes in on best optimization. What we want to do is configure the Forest Model and programmatically edit Forest options. On the DATA tab, in the

DATA property, click the **Select a table** button to select the table that we are going to use in this example.

Your table should be available in Libraries " My Libraries MYCASLIB". Select the table named "PKPARAMS_RAW_DATA" then Click OK to close the Select a Table window. Keep **Validation data** and **Test data** de-selected on the **Partition Data** option until you're ready to run the full model. This is your NOMINAL target. Next select your INTERVAL target data for VALIDATION, and TRAINING. After executing this, SAS studio will show all training trees created. In the Results, we see the summary of model information, which contains the Out of Bag (OOB) Misclassification Rate of 0.24958. In addition, we can observe a summary of observation counts. As you scroll down through the Results, you will see the Variable Importance Table. The last table is the Fit Statistics Table, which provides a summary of common fit statistics based on the number of trees in the forest model.



Output 1. Output from SUPERVISED Optimization of forest

CONCLUSION

The simplest algorithm for gradient descent is rarely the most effective. For deploying effective machine learning algorithms the gradient descent must also, estimate, train, evaluate and predict the next move. To get there we boost the gradient descent algorithm with methods like: momentum, adaptive gradient descent, adaptive momentum gradient descent, and stochastic gradient descent. This will make it easy to train and test machine learning algorithms. When we combined with computer looping and datasets, we take advantage of deep learning.

Ties to Data Science

Data science is an inter-disciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data. Data science is related to data mining and big data." Wikipedia In this paper you'll be introduced to Artificial Intelligence systems that track and process big data. You'll be familiarized with common AI processes to see "under the hood" where scientific methods, and algorithms do the work of training AI applications to think

The conclusion summarizes your paper and ties together any loose ends. You can use the conclusion to make any final points such as recommendations, predictions, or judgments.

REFERENCES

Scarpino, Matthew. 2018. TensorFlow for dummies. Hoboken, NJ : John Wiley & Sons Inc.

Larson, Roland E,Hostetler Robert P. 1987. Brief Calculus with Applications Alternate Second Edition. Lexington, Massachusetts : D. C. Heath and Company.

SAS Institute Inc. 2019. *Exploring SAS® Viya®: Data Mining and Machine Learning*. Cary, NC: SAS Institute Inc.
Exploring SAS® Viya®: Data Mining and Machine Learning
Copyright © 2019, SAS Institute Inc., Cary, NC, USA

Website github.com. "sas-viya-machine-learning." Accessed March 27,2020. https://github.com/sassoftware/sas-viya-machine-learning/tree/master/data.

Website Support.SAS.com 2020. "SAS™ Visual Data Mining and Machine Learning" Accessed March 27, 2020. https://support.sas.com/en/software/visual-data-mining-and-machine-learning-support.html#documentation

ACKNOWLEDGMENTS

This paper is based on the "**Data Mining and Machine Learning on SAS™ Viya**" videos in SAS® Viya®

Enablement is a free course available from SAS Education.

Special thanks to Fred Balch, Fal Diabate and Gary Sayingim and Walker Consulting LLC.

"Every step of progress the world has made has been from scaffold to scaffold and from stake to stake."

--Wendell Phillips, Speech for Women's rights (1851)

RECOMMENDED READING

- *Base SAS® Procedures Guide*
- *SAS® For Dummies®*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Karen Walker
Walker Consulting LLC
480.206.7196
Karen@walker.consulting
KarenErneWalker@twitter.com
https://www.linkedin.com/in/karen-walker-434b801/detail/contact-info/