Logistic Regression Collaborating with AI Beam Search

Tom, Daniel

25 December 2021

Logistic Regression Collaborating with AI Beam Search

Rivals Neural Network and Gradient Boosted Machine

Daniel Tom, Ph.D.
https://orcid.org/0000-0003-4853-2498

DTom Computing
https://www.linkedin.com/in/DanielTom

December 25, 2021†
March 4, 2023 (rev.)

Abstract

We systematically explore the universe of all models using AI search methods. We automate much of the data preparation and testing of each model built along the way. The result is a method and system that generate superior production ready logistic regression models, beating an industry standard consumer credit risk score, GBM and NN ML models. We also incorporate into our system a method to eliminate disparate impact used by the FRB and the FTC.

Keywords: Modeling, Regression, Logistic, AIC, IRLS, AI, ML, NN, GBM, KS, IV, GC, Wald, X2, PSI, VIF, correlation coefficient, condition index, proportion-of-variation, reject inference, FRB, FTC, CRA, disparate impact, BISG, SBC, ARM, Intel, GPU, GPGPU, BLAS, LAPACK, transformation, normalization

Introduction

Logistic regression has been around for over half a century. It has found use in a wide array of binary classification tasks in medical fields, social sciences, and risk management in banking and financial services. Many risk models have been built with logistic regression using modeling practices that are passed down. Being log-linear, a logistic model is easy to explain, and this satisfies regulatory requirements in consumer banking credit decisions.

Machine learning (ML) models are relatively newcomers. While ML models often appear to have an edge in performance, some don't perform well in validation, especially with novel data samples, leading to stability and generalization concerns. Another shortcoming holding back acceptance is that ML models are rather nontransparent with their structure and/or complexity, therefore not lending themselves to explanation.

In this article, we review, redefine, and refine modeling processes and methods to generate superior logistic models rivaling ML models. We automate much of these processes in software in our research and development system. We could describe this system following its processing

† engrxiv.org/vz496

steps from beginning to end, but that would be rather dry. Rather we describe the construction of our research system, beginning with the core implementation of the iterative reweighted least squares (IRLS) algorithm, and explain the motivations behind adding another step, or another component, or enhancing and revamping conventional processes. We take a page out of the playbook of successful consumer products (e.g., the Apple iPhone) with successive refinements in each release. The reader may find this easier to digest than a full-blown system requirements specification.

Iterative Reweighted Least Squares

IRLS is the algorithm that produces a maximum likelihood optimization of a set of binary logistic regression model coefficients. This is our starting place as the algorithm is well-defined and has many implementations. We want to make sure our IRLS implementation is done right and bulletproof, so with every chance we cross-check our model with a leading enterprise statistical and data management software. Our tests begin with very small binary classification data sets and, when matching, progress to more and more observations and larger and larger set of predictor variables.

We envision very early on that our system should handle weighted data samples, not only frequencies, but also fractional weights. This turns out to be crucial in handing Bayesian inferred demographic data for mitigating disparate impact. Another use is reject inference (RI), where a RI model may infer a rejected credit application (hence no performance on the books) to have simultaneously good outcome with a probability, and bad outcome with the complementary probability. Our enterprise statistical software could handle frequencies in whole numbers, but when it does, most output statistics are still based on the observations (i.e., unweighted) while the estimated coefficients do reflect the frequencies in each observation. The only other option is to use the events/trials specification to pick up fractional weights in real-valued data fields, and so that is what we use to cross-check our IRLS implementation.

Using the logistic function (a.k.a. sigmoid),

$$S(\cdot)=\frac{\exp(\cdot)}{\exp(\cdot)+1}=\frac{1}{1+\exp(-\cdot)} \tag{1}$$

we can express in matrix form IRLS for weighted data as:

$$\beta_{t+1}=\beta_t+\left[x^T(W\odot D)x\right]^{-1}x^T W\left(y-S(x\beta_t)\right) \tag{2}$$

where β contains the logistic regression coefficients, x and y are independent (predictor) and dependent (target) variables, respectively, W and D are diagonal matrices representing the weights and $S(x\beta_t)(1-S(x\beta_t))$, respectively, and $\odot$ represents element-wise multiplication. Rather than online (i.e., per observation) update, we update the coefficients after each epoch (iteration) through the data set. Therefore x and y are respectively matrix and vector.

Everywhere we need to be quite conscious of the computing requirements. Typically, IRLS converges within 6 iterations. While this is much faster than training some ML models, e.g., neural networks (NN), this is only estimating one of the many models within a huge universe (explained below). In some situations we managed to reduce the number of iterations to as few as three.

To generate statistics we compute them along the way with each pass through the data. Other than the IRLS convergence criterion, first and foremost we want to know the performance of the logistic model separating the binary targets. We also need to check the signs of all the regression coefficients, as well as the Wald statistics to ensure all coefficients are significant. Our modelers have long complained that in some open source modeling software assessing a model's performance is a function call separate from fitting a model, thus requiring another pass through the data which may be time-consuming with large data sets. We should note that in a computer, central processing unit (CPU) arithmetic computation is orders of magnitude (easily 1000x) faster than data storage access. Therefore, generating model performance statistics should not be much overhead when data is already loaded. This is one of many examples (more below) of design optimizations we engineer into our system.

Also worthy of note is that many linear regression related statistics can be computed with only one pass through the data. In particular, we need the variance inflation factor (VIF), condition index, correlation among predictor variables, etc. With one pass through the data we compute $x^T W x$ for all predictor variables. Before estimating any logistic model we first compute these statistics using a submatrix of $x^T W x$ by selecting relevant rows and columns that contain the model's predictors. We check the statistics, and if any VIF, condition index, or correlation coefficient is beyond acceptable limits, we don't even spend computing time running IRLS as the logistic regression model will not be a viable candidate.

If the data contain multiple vintages, it would be beneficial to compute the population shift/stability index (PSI) for each predictor variable. PSI could be computed in advance and may be used to whittle down the predictors to a subset of stable predictors for logistic regression. Else we would need to check the PSI of every predictor selected, and the logistic model would not be viable if any variable is unstable.

AI Search in the Universe of Models

With a set of p predictors in x, there are 2^p possible logistic models in our search space, where each model selects a distinct subset of predictors. Via combinatorics there are C_k^p different models with k predictors each. Summing C_k^p for k from 0 to p would give a total of 2^p. We liken this to genetics, where each predictor is a characteristic or trait that is dominant when selected in the model, or recessive when not. For example, considering $p=4$ personality traits alone, there are 2^4 =16 unique Myers-Briggs personality types. With thousands of genes and traits in a person, there is amazing diversity in humanity. The larger the set of predictors, the larger is the universe of models.

In banking and financial services data sets we encounter a lot of predictors. These could be a combination of on-us (in-house/account) data as well as off-us data typically obtained from consumer credit reporting agencies (CRA). In this context and according to some regulatory guidance documents raw data are customer characteristics, while aggregated or derived data would be called attributes. Here we refer to characteristics, attributes, predictors, and independent variables interchangeably. The number of attributes we deal with has climbed from 300 or so years ago to over 1000 these days. If there are 10 attributes we could search through all 2^{10} =1024 models. If there are 20 attributes, then universe has well over a million possible models. If there are 30 attributes, then over a billion, etc., etc. With 1000+ attributes, the search space is enormous. Therefore, we need a smart way to navigate efficiently through the universe of models to find the best one. This is where we employ classic artificial intelligence (AI) search techniques.

Beam search is an AI technique that helps manage the process of searching through a vast space of possibilities. The general idea is that like a beam of light, we focus on the possibilities illuminated by the search beam as our next target(s). This technique has many variants. A pencil or laser beam search may refer to picking the best one as the next target. A searchlight beam search may refer to exploring the best N (constant) targets. A flashlight beam search may refer to exploring a progressively larger set of best targets.

Flashlight beam search is the one we engineer into our system. The base of exploration into the universe of all models is the null model. The null model has only an intercept but no predictors. The null model intercept should equal the log odds (logarithm of the odds of an event vs. non-event) of the samples in the data set. This would be a good test for any implementation of IRLS. As a boundary condition the null model is a perfect starting point as we keep building the model by adding predictors.

The first step is to explore the immediate vicinity of the base. Starting from the null model, we reach into the space of models with 1 predictor (there are p of them). The second step is to pick the 2 best performing viable models from these $p+1$ (null model included), and add another predictor to each. We should note that the null model is unlikely to be among the top 2, nevertheless it needs to be on reserve. Say the 2 best viable models at this point are single predictor models. Each would have $p-1$ possibilities of adding another predictor. However, there is overlap, as each may add the predictor in the other model. We could go ahead and estimate these models, even though they are identical. But if we could save some computation time, we would rather skip the duplicate work.

A database containing already built models comes in handy here. In the N-th step of the flashlight beam search we pick the top N viable models from the database, and add another predictor to each provided the expanded model is not already built. We would query the database whether the expanded model is already there. If not, we estimate the model, and insert it into the database.

For simplicity, we rig our database of models out of the computer's file system instead of using proper database software. We note that a file system is a type of database in its own right, providing the two database operations we need – query and insert. Representing a model as a path/file, we can query its existence in the file system before attempting to build the model, or insert it into file system after one is built.

We could go on adding predictors until all possible models are built (if predictors are few), but we need a criterion to terminate searching the vast space of models when the prospects are bleak. We set up the search termination condition as follows: We compare the top N+1 viable models in the N+1st step to the top N+1 viable models in the previous (i.e., N-th) step. If there is no difference, i.e., adding another predictor makes no further improvement, we terminate the search. The best performing viable candidate becomes our final model.

Comparing with Traditional Modeling Practice

The experienced modeler may know that stepwise logistic regression procedure has been around. So how is this IRLS logistic regression + AI search different? Say we put the data into a stepwise logistic regression procedure, and we put the same data into IRLS+AI. The stepwise logistic regression procedure builds and selects a final model based on the Akaike Information Criterion (AIC). Rarely, if at all, does AIC stepwise logistic produce a viable model for us. First, the sign of a coefficient may not be right. It must align to our understanding so the modeler can explain the

model. Second, a coefficient may not be significant by the Wald test, i.e., having a non-negligible likelihood that the coefficient is zero. The Wald statistic may be displayed with the stepwise logistic regression procedure, but it is not a built-in check for model viability at any step. Needless to say, the modeler needs to run separately linear regression as well as correlation procedures just to obtain VIF, condition indices, correlation among variables, etc. to check for viability. IRLS+AI has engineered into the system model viability checks all along the way as it builds up the model.

So the AIC stepwise logistic regression procedure gives us an unviable model, what is a model to do? Many a modeler would start dropping one by one those offending variables that make the model unviable, e.g., a coefficient that is insignificant, or having the wrong sign, or large PSI, VIF, condition index, or correlation coefficient. However, dropping variables could have the unintended consequence of decreasing the model's separation performance. In the limiting case if the modeler ends up dropping all predictors, then the result is the null model which has no separation whatsoever. How about adding predictors to AIC stepwise logistic regression's model? We have not met a modeler who makes this attempt. Having an unviable model to begin with, adding predictors would be perpetuating with another unviable model, so this effort is futile. This is the reason we scrap traditional modeling practice, instead, add predictors only to viable models and checking for model viability along the way when building our IRLS logistic regression model with AI search.

Systems Implementation

We engineer our IRLS+AI search modeling system to implement on widely available computing hardware and operating systems. We make design considerations and choices based on computing technology at present. This could change in the future with rapid advancement in computing.

The reference implementation is a server/background/batch processing software running in a 64-bit linux operating system (OS) on an Intel CPU. Our implementation can be ported to run on some other ubiquitous PC operating system, ARM CPUs, server, or commodity hardware like laptop/notebook or SBC (single-board computer). Output stream and files provide a monitoring mechanism into background batch processes. Most output files can be opened in a generic text editor or a spreadsheet application.

The key processing is one step (the N+1st) in the IRLS+AI search from a base model. The boundary condition being N=1 which is the null model. The base model for this N+1st step is one of the top N performing viable models in the previous (the N-th) step. We build the next set of models

simultaneously (via POSIX threads or pthreads) by adding one predictor not already in the base model. We record/de-duplicate this set of models in our database of already built models. Additional scripts implement the flashlight beam search, and filters thoroughly check for model viability.

Multithreading allows IRLS threads to read modeling data in shared memory without needing separate loads. As mentioned storage access is rather slow, so eliminating unnecessary data loads speeds up the overall process significantly. We do not load the entire data set into memory, rather we use a fixed size buffer so our system can run as well on lowly computing platforms with limited memory. We may need to cycle through the data per epoch, but we still manage to build thousands upon thousands of models with multithreading. We should note that there may be more threads than the actual number of CPUs or hyperthreads in hardware, but that can be handled by a 64-bit linux OS. In contrast, a 32-bit OS can only handle a smaller number of threads and is therefore limited to much fewer predictors. A 32-bit OS is not recommended; it is on the way out anyway.

We considered using graphical processing units (GPUs) and processor intrinsic vector computing. After all this author created a neural network (NN) backpropagation learning shared library over a third of a century ago with inner loop vectorized FORTRAN driving a graphics computer of that era, wrapped it around with a research grade batch NN trainer, and shared it with fellow NN researchers. Incidentally the NN backpropagation training procedure can be compactly expressed in matrix language.

General purpose GPUs (GPGPUs) are amenable to vector and matrix computing. Nevertheless, these are specialty components akin to coprocessors of yesteryear. Even if vector and matrix operations take next to zero time, considerable cycles are taken to copy or move data off-chip to the GPU and bring the results back to the CPU. Communications off-chip on system bus or network is always slower.

Processor intrinsic vector operations need no off-chip data transfer. However, intrinsic vector operations only run in a specific instruction architecture. Intel AVX intrinsics are not supported in ARM CPUs. Vice versa, ARM NEON intrinsics are not supported in Intel silicon. Moreover, newer intrinsic operations support wider and wider vectors with newer CPU releases, which may not be backward compatible.

We estimate there may only be a few percent speed up using processor intrinsic or GPGPU outside of BLAS (basic linear algebra subroutines) and LAPACK (linear algebra package) which handle the IRLS vector/matrix operations. There are plenty of testing and branching (if-then-else code) that would break parallel processing. At a high level, building a set of models concurrently is better served by multithreaded processing. The actual hardware threading capability is transparent to POSIX thread enabled

software, which runs on multithreaded CPUs as well as single threaded CPUs without modification. We are not ruling out processor intrinsics or GPGPU. As computing technology makes advances, e.g., when GPGPU becomes ubiquitous or integrated into the CPU die, we would definitely reevaluate the possibilities.

We have also experimented with multinode distributed or cluster computing. Recall at the N+1st step we have the top N performing viable models as bases of exploration. Each exploration (a key processing step described above) is handled by our multithreaded software. With multiple nodes in a cluster, each node can host a base and launch its exploration simultaneously with other nodes. We have completed runs of our system in distributed mode, launching concurrent jobs via a load sharing facility into a cluster of disparate nodes having Intel CPUs of various ages, rarely with a GPU, and all connected to a centralized file server. Theoretically we should get an N-fold speed up, but that is just our hope. Synchronization, connectivity, and data transfer bottlenecks all play a role in the timing. This is consistent with our prior experience on a pachyderm themed cluster which is touted to sort enormous text files, but that says nothing about technical computing. There are other clusters proven for scientific/technical computing. If we get our hands on a supercomputing cluster, we would surely try out the message passing interface (MPI).

Now that we have described the core IRLS+AI search, let us move on to other associated processing in our system which contribute to increase in performance, give us a unique understanding of the logistic model, and help gain acceptance in the model validation and approval work stream.

Variable Reduction

Given a large number of attributes, putting all of them through any regression would take much computing time. When asked how can we reduce the number of predictors passed into stepwise logistic regression, a senior modeler told us to run a stepwise logistic regression and see what it gives you. Needless to say we got the runaround with this chicken and egg conundrum. If this ever works, it would buck the tradition. Before we describe our own variable reduction methods below, we first review preexisting practices.

A popular variable reduction method involves selecting only predictors with high information value (IV). Jeffreys divergence, a symmetrized version of the information theoretic Kullback-Leibler divergence, is a distance measure between two analytic (idealized) probability distribution functions.

$$\int_{-\infty}^{\infty} \left(p(x)-q(x)\right) \ln\left(\frac{p(x)}{q(x)}\right) dx \tag{3}$$

As our data come in samples, the integral in Jeffreys divergence is replaced by a discrete summation to calculate the IV.

$$\sum_{k} \left(p_k - q_k\right) \ln\left(\frac{p_k}{q_k}\right) \tag{4}$$

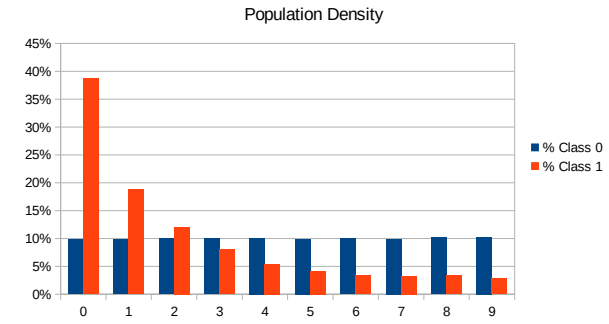
Here p_k and q_k are probability densities of the two target classes, but frequently either could be zero, making the quotient of the term and therefore IV undefined. One remedy creates contiguous bins out of the range, and making them fewer and wider until both the numerator and the denominator are non-zero. It may work with the development data, but this remedy could break again with validation samples. Another dubious practice zeros out the term so it would not ruin the whole summation. Zeroing out the term means we ignore the bin, even when witnessing an extinction. If extinction is not surprising (surprise is information) then what is? Neither approach is satisfactory. We know the number of bins affect the IV quantity – collapsing to one single bin would make IV exactly zero. We also learned that other metrics, e.g., Hosmer and Lemeshow (HL) statistic, are also impacted by binning. Recent software that allows changing the historical 10 bins show highly varying HL statistics as a result.

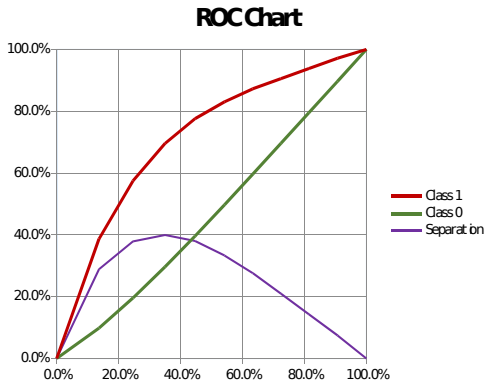
We seek alternatives to computing IV as binning hundreds of predictors is a burdensome chore. Often our enterprise statistical software's decile procedures fails to produce ten bins as many attributes are highly skewed. Fundamentally binning is a noise generating irreversible quantization. We need a metric that requires no binning. Our breakthrough comes when we encounter a risk score among the predictors. So do we compute the IV for this score? Or do we use the KS (Kolmogorov-Smirnov) separation for this score? Well, an attribute or a score is just a number. If we can compute IV, we can compute KS. While we are at it, why don't we also throw in GC (Gini coefficient, same as Somers' D) and compute all three of them at the same time? KS and GC need no binning.

It will become obvious why we end up choosing GC and not IV or even KS. Examples below illustrate the similarities and differences among them. Our first example shown in Exhibit A looks like a fairly good attribute with IV=0.860, KS=0.399, and GC=0.495:

Exhibit A

Attribute	% Class 0	% Class 1
0	9.8%	38.7%
1	9.8%	18.8%
2	10.1%	12.1%
3	10.0%	8.0%
4	10.1%	5.4%
5	9.9%	4.2%
6	10.0%	3.3%
7	9.9%	3.3%
8	10.1%	3.3%
9	10.3%	2.9%
All	100.0%	100.0%

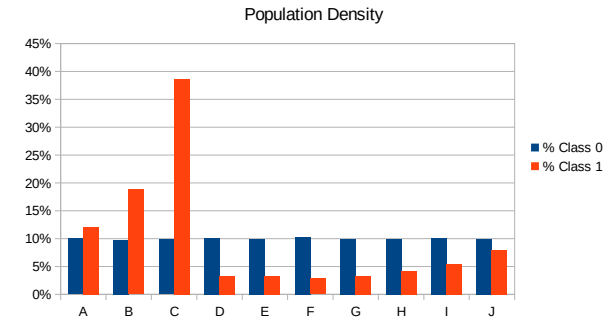


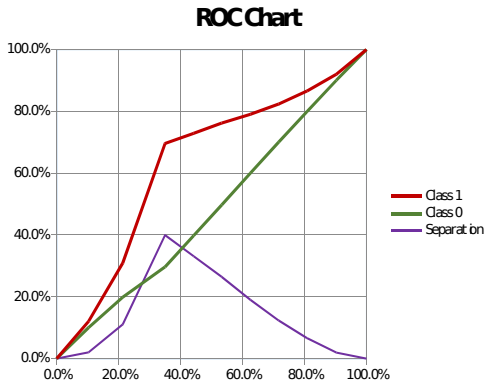


In the next example below the attribute does not have as fat a separation in the ROC chart. GC shows decrease, yet IV and KS remain the same. The second example in Exhibit B has IV=0.860, KS=0.399, and GC=0.306.

Exhibit B

Attribute	% Class 0	% Class 1
A	10.1%	12.1%
B	9.8%	18.8%
C	9.8%	38.7%
D	10.1%	3.3%
E	9.9%	3.3%
F	10.3%	2.9%
G	10.0%	3.3%
H	9.9%	4.2%
I	10.1%	5.4%
J	10.0%	8.0%
All	100.0%	100.0%

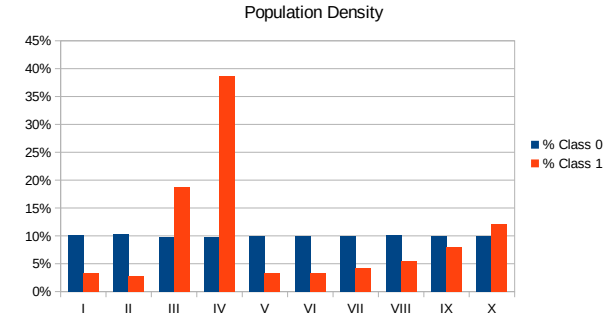


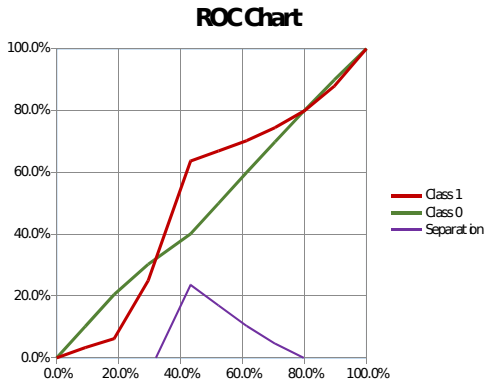


The third example in Exhibit C has IV=0.860, KS=0.236, and GC=0.053.

Exhibit C

Attribute	% Class 0	% Class 1
I	10.1%	3.3%
II	10.3%	2.9%
III	9.8%	18.8%
IV	9.8%	38.7%
V	9.9%	3.3%
VI	10.0%	3.3%
VII	9.9%	4.2%
VIII	10.1%	5.4%
IX	10.0%	8.0%
X	10.1%	12.1%
All	100.0%	100.0%





The attribute in the third example looks even worse in the ROC chart. However, IV remains the same. KS now shows decrease. GC decreases further.

The astute reader will notice the second and third examples are just row-permutations of the first. IV staying the same means it is insensitive to rank ordering anywhere. Between the first and second examples, we see KS is insensitive to rank ordering on either side of the maximum separation. GC is the separation measure that is sensitive to rank ordering everywhere. Thus, for variable reduction, we select attributes with high GC instead of IV.

As with traditional modeling practice, we further reduce the number of predictors by screening variables for shift/instability with PSI, and for high correlation to each other. We should note that the computation of PSI uses the same formula (4) as IV and inherits the same problems. Detecting shifts and instability is a bit more nuanced, and we should have a replacement for PSI in an upcoming article.

We remove variables with high correlation to each other by building and pruning a correlation clustering tree. The Lance and Williams general algorithm with complete linkage is used for hierarchical clustering of the independent variables. The clustering algorithm begins with the calculating the correlation matrix of the independent variables. Then the algorithm

iterates by finding the highest correlation in the matrix. The two variables having this highest correlation become a cluster. The correlation table is revised to include the remaining variables plus the newly formed cluster node. The iteration stops when the correlation matrix contains only one entry, at which point the final clustering tree remains.

Pruning the clustering tree reveals major clusters of variables that are highly correlated among themselves. Pruning starts with highly correlated branches, and progress to moderated correlated branches. With each pruning, the variable having the highest separation is selected, thereby securing a high separation power of the model. This step significantly reduces the number of variables, leaving those with high separation, but not high correlation.

We have deferred until now to disclose the performance measure we use in IRLS+AI search. Based on the observation in the examples above and the same reasoning discussed, the Gini coefficient is our favorite choice for separation performance measure. We have not computed IV or KS at all in IRLS+AI search (though we have them ready for independent validation). Using GC is a key design consideration as we engineer a system that builds superior logistic regression models.

Variable Transformation

Experience like we have with GC, KS, and IV also influence our approach to variable transformation, normalization, and the treatment of missing values. Some modelers do not transform any variable. Those who do transform a predictor so it becomes linear with the log odds, thus satisfying one of the assumptions (requirements) of logistic regression. The square root, logarithmic, double log, reciprocal, and exponential are a few of the transformations. There could be infinitely many functional transformations, some with better fit than others, but we should not get carried away.

For clear explanation, we stick to these traditionally accepted functional transformations, with minor adjustments. First, we would make the transformation function pass through the origin. For example, the log function does not pass through the origin. Instead, we use $\ln(1+x)$ as the logarithmic transformation as this passes through the origin. Second, we make sure it is monotonic increasing. The reciprocal is not, so we use $1-1/(1+x)$ which is monotonic increasing, and passes through the origin as well. These adjustments are fundamental to variable normalization (unrelated to Gaussian; not centering), discussed below.

Normalization

While a functional transformation makes a predictor variable linear with the log odds, normalization further transforms a predictor completely to the log odds. For example if $T(x)$ is a functional transformation, then $N(x)=mT(x)+c$ is a normalized transformation. Normalization can be readily achieved via a simple logistic regression of a functional transformed variable. Of the acceptable transformation functions, we pick the one that has the highest adjusted R-squared coefficient of determination. With the sign of the variable incorporated in the coefficient m, the sign test in the IRLS+AI model build reduces to ensuring all positive coefficients, which greatly simplifies the model viability check.

Another benefit normalization brings is a unit-free multiple logistic regression model. Where x may have a unit (e.g., dollars of delinquent balance), $N(x)$ is log odds (of bad credit outcome when having x dollars of delinquent balance). The multiple logistic regression model expresses the log odds of an event as a linear combination of normalized predictors in log odds. This is just natural or "normal" (hence normalization) with no need to explain away mismatched units among predictors.

Treatment of Missing Values

Our system's treatment of missing values is unique, as it is a natural extension of our design described above. We do not use any missing value imputation. We do not assume the worst (prevalent in risk management) nor the best (preferred in marketing). Instead, the way we treat missing values of a predictor is the same way we treat numeric values – transform them to normalized log odds. Having a unified treatment means we can handle mixed numeric/categorical variables just as well as numeric variables and categorical variables.

This insight comes along as we explore the relationship between IV, KS, and GC. GC is the only one among the three that is sensitive to rank ordering everywhere. Categorical variables have no inherent ordering, so we can rearrange the categories however we want. If we need to divide the categories into a left pile and a right pile with the highest separation, how could we do it? It turns out sorting the categories from left to right by their log odds does the trick. This is akin to the first example above in Exhibit A. Sorting by log odds gives the highest GC separation.

For the missing in a numeric predictor, where would we put it to give the highest separation? Sticking it beyond the minimum or the maximum may not work. We can put it in between numeric values with trial and error. The highest GC always occurs when the missing is in line with the neighboring

numeric values in the log odds space. Therefore, as we create a normalized predictor, any categorical variable or any missing category in an attribute simply gets the log odds of the category.

Relative Normalization

Relative normalization gives us additional advantages. The inspiration comes along as we work with the aforementioned "absolute" normalization. We create a relative normalization $R(x)$ of a predictor x by subtracting the sample population log odds β_0 from the absolute normalization $N(x)$.

$$R(x)=N(x)-\beta_0=mT(x)+c-\beta_0 \tag{5}$$

A simple logistic regression of a relative normalized predictor gives $\beta_0+R(x_k)$. A multiple logistic regression of relative normalized predictors gives $\beta_0+\sum \beta_k R(x_k)$. With no predictor, the logistic model is just β_0, which is the null model. Despite being subtracted in relative normalization of the predictor, the population log odds reappears as the logistic model intercept.

Relative normalization gives us a unique model explanation. Our logistic model expresses the log odds of an event above the population level as a weighted combination of relative normalized predictors, which themselves are log odds above the population level. When all predictors have log odds at the population level, the model log odds are also at the population level. If a predictor shows an increase/decrease in log odds above the population level, the model log odds get a contributed increase/decrease as weighted by the corresponding coefficient. This uniquely explains the logistic model by contributions of relative normalized predictors. If not for relative normalization, a predictor's contribution can be made nonsensically large by adding an arbitrarily constant, and the regression would just absorb the change right into the intercept.

Contributions cannot be used alone in generating reasons for adverse action. When a customer's attribute is at its best, the attribute cannot be used as an adverse action reason. For example, a delinquent balance attribute may make significant contribution to a credit model score. A customer may have zero dollars of delinquent balance, so this cannot be used against the customer. Therefore, there needs to be a gating principle in governing the generation of adverse action reasons. We take this gating principle even further. We rank the drop in contribution from the best value, and take the highest few as reasons for adverse action (no drop, no reason).

Relative normalization helps us simplify model build and viability checks in our system. In our system, relative normalization helps us manage the

logistic regression intercept, which stays close to the population log odds even with a large number of predictors. Congruent with traditional modeling practices, we check the condition index for model viability since the matrix inverse is theoretically involved in IRLS. If we encounter a condition index (intercept-included) that is unacceptably high, we check the proportion of variation for collinearity. Without relative normalization we may find predictors highly collinear with a large intercept. The reason is that the intercept has become a “catch all” term with various transformations or the lack thereof. With relative normalization the intercept is controlled, so is the condition index. With an acceptable condition index, we don't need to compute the proportion of variation, thus streamlining the model build.

Preprocessing and Postprocessing

Preprocessing and postprocessing help integrate our system into the modeling workflow, creating a seamless drop-in replacement for the stepwise AIC logistic regression procedure. We start with exporting the data from the proprietary format in our enterprise statistical software to IEEE 754 binary floating point format. When the final model is completed, we export the model as macro code of the statistical software, which gets expanded into full scoring code for validation and/or production.

With the data in IEEE binary we immediately compute PSI as well as basic variable statistics like sample count, frequency of missing, mean, standard deviation, skewness, kurtosis, percentiles, min and max, etc. which only need to be computed one time. We also include submax, which is the highest value smaller than the maximum. Frequently the nines (e.g., 9999) is a code for a default value, which we would recognize if it is much larger than the submax (e.g., 100). Another likely default value is zero, which may be the result of misspecification or miscoding/uninitialized variables in attribute derivation. This may overload true attribute zeros, in which case it would be hard to disentangle the two. We suspect zero is a default if its log odds are not in line with that of one and above, which we can detect in our system through non-monotonicity of the log odds by a second order (quadratic) normalization. To satisfy the monotonicity logistic regression assumption we can segment the data by the non-monotonic attribute into piecewise monotonic segments. With attributes in abundance we could simply let the system pick up other attributes that are monotonic.

Our reference logistic regression model that beats ML is not segmented. We do not subscribe to heuristics or customary/traditional segmentations. We always ask our modelers to proof segmentation is better than not splitting, by comparing the separation of the unsegmented model vs. the combined segmented models. If there is no benefit segmenting, we need to stop creating unnecessary work. Segmentation creates data sets with fewer

samples each, which lowers the Wald statistics of attribute coefficients. With fewer significant predictors, each viable segment model is bound to have lower separation. This is on top of the segmentation tree carrying away separation (we have the math) like a decision tree. The only exception is segmentation by a random number which has no association to the event, which is what we use to partition data for validation vs. development.

The system's collection of output give us insight into the superiority of the models it builds. The number of models gone through by our system as recorded in model database is typically in the thousands (depends much on sample size). The final model is the highest separation viable model among these thousands upon thousands. We also notice there more predictors than in an AIC stepwise build. Looking at the separation along the way as the system add predictors, we can observe the relationship between separation and the number of predictors. For example, separation with 10–15 predictors (about as many as AIC stepwise in this experiment) is about 2 points down. Separation with 15–20 attributes is still about 1 point down. Separation only begins to flatten out around 30–40 variables. This says AIC stepwise leaves much separation on the table. The traditional approach to drop variables until the model is viable is not going to get better separation. In contrast, a comparably performing NN uses about twice as many predictors.

Eliminating Disparate Impact in Model

As required by the Fair and Accurate Credit Transactions Act (FACTA) of 2003, the Federal Reserve Board (FRB) and the Federal Trade Commission (FTC) both submitted to the US Congress in 2007 their reports on credit and insurance scoring and their effect on disparate impact. In particular the FRB report describes two methods to create demographically neutral models. The first creates separate models for each demographic, using only the demographic's subpopulation data. The second combines all demographics, but includes demographic indicator/control/dummy coded variables to adjust the intercept. Our system implements the second method in the FRB report to create demographically neutral models. Further we use the same FRB method to eliminate disparate impact in attribute normalization, as it too involves simple logistic regression.

Theoretically we could add a third method that combines the benefits of the two FRB methods. We could build a demographically neutral model including all demographics in the data, with demographic indicator/control/dummy variables adjusting not only the model intercept, but also interacting with the attributes/predictor variables. Therefore, the model coefficients as well as the intercept are different by demographic, in effect giving each demographic a model of its own. However, in this third method the number

of coefficients is as many folds as the number of demographics, and this often leads to numerical/convergence issues in estimating such models with a large degree of freedom and/or rare demographics.

In consumer credit fair lending, we consider six racial demographics:

Note: vulnerable demographics in *italics* require protection

Indicator	**Race**
W	White
B	*Black/African American*
H	*Hispanic*
P	*Asian/Pacific Islander*
N	*American Indian/Alaska Native*
O	*Multiracial/Other*

There are two age demographics:

Indicator	**Age**
A	Adult Less Than 62 Years Of Age
S	*Senior Age 62+*

There are three gender demographics:

Indicator	**Gender**
M	Male
F	*Female*
U	*Unknown Gender*

Altogether, there are 36 distinct demographics considering all interactions between race, age, and gender:

Interaction	Demographic
W*A*M	White adult male
W*A*F	*White adult female*
W*A*U	*White adult of unknown gender*
W*S*F	*White senior female*
W*S*M	*White senior male*
W*S*U	*White senior of unknown gender*
B*A*F	*Black/African American adult female*
B*A*M	*Black/African American adult male*
B*A*U	*Black/African American adult of unknown gender*
B*S*F	*Black/African American senior female*
B*S*M	*Black/African American senior male*
B*S*U	*Black/African American senior of unknown gender*
H*A*F	*Hispanic adult female*
H*A*M	*Hispanic adult male*
H*A*U	*Hispanic adult of unknown gender*
H*S*F	*Hispanic senior female*
H*S*M	*Hispanic senior male*
H*S*U	*Hispanic senior of unknown gender*
P*A*F	*Asian/Pacific Islander adult female*
P*A*M	*Asian/Pacific Islander adult male*
P*A*U	*Asian/Pacific Islander adult of unknown gender*
P*S*F	*Asian/Pacific Islander senior female*
P*S*M	*Asian/Pacific Islander senior male*
P*S*U	*Asian/Pacific Islander senior of unknown gender*
N*A*F	*American Indian/Alaska Native adult female*
N*A*M	*American Indian/Alaska Native adult male*
N*A*U	*American Indian/Alaska Native adult of unknown gender*
N*S*F	*American Indian/Alaska Native senior female*
N*S*M	*American Indian/Alaska Native senior male*
N*S*U	*American Indian/Alaska Native senior of unknown gender*
O*A*F	*Multiracial/Other adult female*
O*A*M	*Multiracial/Other adult male*
O*A*U	*Multiracial/Other adult of unknown gender*
O*S*F	*Multiracial/Other senior female*
O*S*M	*Multiracial/Other senior male*
O*S*U	*Multiracial/Other senior of unknown gender*

The white adult male demographic is not culturally considered vulnerable. Correspondingly the white adult male demographic is the base or reference category in modeling. All the other 35 demographics are vulnerable by one or more of race, age, and gender requiring protection. Only the interaction terms of these 35 demographics appear in regression as dummy variables. In regression modeling, indicator/control/dummy variables can be thought of as beacons that attract explanation or association to the outcome in the data. Without them, other predictors fill in the role to explain nuances in the outcome.

We follow the second FRB disparate impact eliminating method: In model estimation these interaction terms create a demographic shift to the intercept from the reference. In validation and scoring production these demographic interaction terms are dropped, creating a degenerate model without intercept shift. The degenerate model scores everyone as white adult male, thus eliminating disparate impact to protected demographics.

A modeler would be inclined to think that dropping the dummy variables post regression would hurt the model's performance. On the contrary, high performance in the final model can still be achieved. Case in point, in the not so distant past students learned to look up the logarithmic table, add the values and look up the antilog to perform a multiplication. The student learned the concept that addition in log scale is multiplication. Then the calculator came along and made it far easier, but calculators were not allowed in exams. With good training and exercise with using the log table, exam performance was not affected at all.

We only need to make one adaptation to use the FRB method. Instead of selecting the highest performing viable models, we select the highest performing viable degenerate models at each step of IRLS+AI search starting from the null model. With this adaptation, our system continues to build demographically neutral models with high performance. We find no appreciable loss in separation at the end.

Conclusion

In this article we describe a system for building superior logistic regression models via IRLS+AI beam search.

Launching from a base we explore the immediate space of models with one additional predictor in simultaneity via multithreaded computation. We select top performing viable models that pass multiple modeling criteria as the new bases for the next explorations, and iterate the step until there is no improvement in the top performers list. At which time the system would

have built thousands and thousands of models, so the best viable model is the best among thousands and thousands.

We discuss the engineering considerations in the system design and implementation. We review separation measures and choose the Gini coefficient (Somers' D) as the sole measure of performance throughout the system. The Gini coefficient drives variable reduction via separation and correlation clustering. Its omnipresent rank ordering inspires our treatment of missing values, transformation, and absolute/relative normalization. As a result we get a better explanation of predictor contributions and generation of adverse action reasons. The system also incorporates tools for data discovery and segmentation, integrates into our enterprise platform as a drop-in replacement, and goes as far as generating model scoring code for validation and production.

The system builds demographically neutral models with no loss of performance. It implements a method used by the Federal Reserve to eliminate disparate impact. This we find no parallels in other machine learning models.

A number of models have already been built and installed and are currently in production. In a recent project our reference logistic regression model beats an industry standard consumer credit risk score by a significant margin. It also beats gradient boosted machine (GBM) models built by our modelers. It performs comparably to the best neural network model built by our team. We find the logistic regression model to be rather stable and much more transparent to explain. All else being equal, having transparency wins. Arbitrary baseline or reference in some ML attribution methods only creates arbitrary explanations. The recent big blue flop lays bare the pitfalls of opacity in ML models. Giving unsafe healthcare advice could put a patient in jeopardy. Not being able to give an elementary explanation of that advice just compounds the problem. Adverse action in consumer credit requires transparency. If the modeler cannot explain the model, we cannot expect the computer to do so.

We are performing modeling research made possible only with this system. Our latest success [3] shows the AI flashlight beam search is very efficient. It is able to find the highest separation/highest likelihood viable solution in a vast search space after building only a small number of candidate models. In one research we achieve higher validation separation by restricting the development data diet. In another we find reject inference may be unnecessary, provided we perform tests with multiple models built with this tool. We hope to document these results in other articles.

References

1. Report to the Congress on Credit Scoring and Its Effects on the Availability and Affordability of Credit (2007, August). Board of Governors of the Federal Reserve System. https://www.federalreserve.gov/boarddocs/rptcongress/creditscore/creditscore.pdf

2. Credit-Based Insurance Scores: Impacts On Consumers Of Automobile Insurance (2007, July). A Report to Congress by the Federal Trade Commission. https://www.ftc.gov/sites/default/files/documents/reports/credit-based-insurance-scores-impacts-consumers-automobile-insurance-report-congress-federal-trade/p044804facta_report_credit-based_insurance_scores.pdf

3. Tom, Daniel, Ph.D. (2023, January 17). Eliminating Disparate Treatment in Modeling Default of Credit Card Clients. https://doi.org/10.31219/osf.io/cfyzv