Package `mactivate`
About

Dave Zes

January 6, 2021

1 Introduction

We start by imagining a system,

$$Y_i = \mathrm{b}_0 + \mathbf{x}_i\,\mathbf{b} + \varepsilon_i\ , \quad \varepsilon_i \sim \mathcal{N}[0, \Sigma] \tag{1}$$

Using data notation, we can write

$$\mathbf{y} = \mathrm{b}_0 + \mathbf{X}\,\mathbf{b} + \boldsymbol{\varepsilon} \tag{2}$$

where $\mathbf{y}$ is an $n \times 1$ column vector, b_0 is a scalar, $\mathbf{X}$ is our $n \times d$ matrix of inputs — our "design" matrix; $\mathbf{b}$ is a $d \times 1$ column vector; and $\boldsymbol{\varepsilon}$ is an $n \times 1$ column vector of observed system shocks, or errors.

Let us further specify (2):

$$\mathbf{y} = \mathrm{b}_0 + \mathbf{X}\,\mathbf{b} + \mathbf{X}^\star\,\mathbf{c} + \boldsymbol{\varepsilon} \tag{3}$$

Here, our inputs, $\mathbf{X}$, and our responses, $\mathbf{y}$, are *directly* observed. Our $n \times m$ matrix $\mathbf{X}^\star$ contains interactions constructed from $\mathbf{X}$. For example, the first column of $\mathbf{X}^\star$ might contain the three-term element-wise product of, say, columns 1, 2, and 17 of $\mathbf{X}$.

Our goal, then, is to identify $\mathbf{X}^\star$ such that the true coefficient vector, $\mathbf{c}$, contains all non-zero elements. Or, in the practical spirit of inferential prediction, to construct $\mathbf{X}^\star$ so that, $\widehat{y}_0 = \widehat{f}(x_0, x_0^\star)$ will be as precise a predictor of y_0 as possible.

This enterprise of identifying interaction terms falls broadly under "model selection" and more

specifically under "feature selection". When d, the number of columns of $\mathbf{X}$ is large, finding $\mathbf{X}^\star$ is traditionally non-trivial. The matter is one of combinatorics. The total number of possible first-or-greater-order interactions is $2^d - d$.

In the simplest sense, activation layers in artificial neural networks (ANNs) are comprised of sibling *units*. The *state* or value of a unit is some function of units' states or values in another layer. It could be said that multiplicative activation functions have been known, at least in concept, throughout the 6-or-so-decade history of ANNs in much the same way that one might say that arithmetic has been known throughout the history of mathematics. This is so, if for no other reason, because when units' states are 0 or 1, i.e, $u \in \{0, 1\}$, then combining units' values by multiplication serves as a boolean AND gate. However, the ANN literature is absolutely dominated by additive functions between units. That is, a unit's value is modeled as some function of a weighted sum of other units's values. About 20 years ago, there was a bump in interest in multiplicative functions with the conjecture that some biological neurons actually possessed a "multiplicative-like" behavior [10]. For an example involving barn owls: [11]. Recently there has been murmurings of renewed interest within the ANN community. One compelling example couches the matter of detecting interactions much in language friendly to a more statistical audience [14] — here the researchers uncover potential interactions by processing the weights (coefficients) of specially formulated linear units' input functions. Alternately, towards the same end, [6] extend layers' modeling flexibility by extending the dimensionality of the weight tensor.

For a more statistical audience, on the topic of detecting interaction effects, in their excellent book ([7]), currently available for online viewing, Kuhn and Johnson point out that tree-based methods can be effective at identifying interaction effects (citations include [2, 4, 8]). Trees's leafs are essentially bins in the input space, and so can naturally implicate non-linear relationships, including interactions. The Additive Groves method [13] is something of a milestone for efficiency, flexibility, and model diagnoses.

More germane to multivariate regression, textbook methods for identifying interactions include *stepwise selection* and *best subsets*. There is a fairly substantial literature of clever multivariate regression extensions/adaptations, including *backtracking* [12]; *backward dropping algorithm (BDA)* [15]; *VANISH* [9]; *Elastic Net*, [16]; the *Dantzig Selector*, especially suited when the input dimensionality is much greater than the number of observations [1]; *iFOR*, [5]. These multivariate regression adaptations all possess the commonality that they *discourage* model complexity during model parameterization, or fitting. For example, both stepwise selection and best subsets utilize information criteria, which discourages complex model parameterizations by imposing a *penalty* that is a function of the total number of proposed parameters. Other methods discourage complexity by imposing a *penalty* that is a function of the parameter estimates themselves. It is important to point out that in addition to forwarding a conceptual framework, much of the relevant literature provide technique, technical details concerning implementation. For example, the LARS-EN

algorithm for Elastic Net, and the iFORT and iFORM algorithms for iFOR, and, as the very title suggests, the *Penalized Likelihood Maximization Algorithm*, [3].

Tree-based methods are extremely powerful and useful. From a nuts-and-bolts perspective, attempting to provide a lucid interpretation as to why, for example, a fitted tree might implicate an interaction down two branches at two particular nodes at different points in the branches' hierarchy, or, in the case of fitting multiple trees, as with random forests, why one fitted tree may implicate some interaction whereas another doesn't, can be challenging — especially when the input dimensionality is large.

Most all the MLR adaptations suited for detecting interactions possess another commonality: They require either full model specification upfront, e.g., the Dantzig Selector, or at least enumeration of candidate interaction terms. Our method under present consideration, m-activation, requires neither.

2 Method

Our method, *m-activation*, is inspired by dense activation layers in neural nets. This method is directly understood by understanding a single "activation" function, g.

$$\mathbf{X}^* = g\left(\mathbf{X}\,;\,\mathbf{W}\right) \tag{4}$$

where $\mathbf{X}^*$ and $\mathbf{X}$ are the familiar input matrices from above, and $\mathbf{W}$ is a $d \times m$ matrix. Specifically,

$$\mathrm{x}^*_{i,k} = g(\mathrm{x}_{i,j}; \mathrm{w}_{j,k}) = \prod_{j=1}^{d} \left(\mathrm{x}_{i,j}\, \mathrm{w}_{j,k} + 1 - \mathrm{w}_{j,k} \right) \tag{5}$$

where, notationally, $\mathrm{x}_{i,j}$ refers to the value at row i and column j of $\mathbf{X}$.

For example, looking at (5), suppose the k-th column of $\mathbf{W}$ contains entirely zeroes. Then every element in the k-th column of $\mathbf{X}^*$ will contain 1 — i.e., unit value. If, at the other extreme, the k-th column of $\mathbf{W}$ contains entirely ones, then every element in the k-th column of $\mathbf{X}^*$ will contain the grand product of the respective row entries of $\mathbf{X}$ — i.e., the *full* interaction.

2.1 Locating W

For some user-selected m, our task is to locate (or *estimate*) $\mathbf{W}$.

2.1.1 Uniqueness

The notion of uniqueness arises in linear solutions as methods for actually solving systems given data involve algebraic linear operations that are sensitive to parameter uniqueness. It could and should be noted that uniqueness is neither necessary, nor, even, sufficient to assure a tractable solution.

The model representation (3) does not necessarily possess a unique parameterization. As an example, suppose the first column of $\mathbf{W}$ contains a 1 at the first row, and zeroes otherwise. Then $\boldsymbol{x}_1$ (an $n \times 1$ column vector), would be equal to the $n \times 1$ column vector $\boldsymbol{x}_1^\star$ — the two vectors, $\mathrm{b}_1\boldsymbol{x}_1$ and $\mathrm{c}_1\boldsymbol{x}_1^\star$ would be perfectly co-linear.

To better understand $\mathbf{W}$ and its ultimate role in our solution, let us briefly consider another simple specific example. Suppose the first column of $\mathbf{W}$ is

$$\mathbf{w}_{\cdot,1} = (0.5, 1, 1, 0, 0, 0)^T \tag{6}$$

The first column of $\mathbf{X}^*$ would then contain

$$\mathbf{x}^*_{\cdot,1} = \tfrac{1}{2}\,\mathrm{c}_1\,(\mathrm{x}_1\,\mathrm{x}_2\,\mathrm{x}_3 + \mathrm{x}_1\,\mathrm{x}_2) = \tfrac{1}{2}\,\mathrm{c}_1\,(\mathrm{x}_1\,\mathrm{x}_2\,\mathrm{x}_3) + \tfrac{1}{2}\,\mathrm{c}_1\,(\mathrm{x}_1\,\mathrm{x}_2) \tag{7}$$

Values in $\mathbf{W}$ that are not equal to zero or one implicate a polynomial term(s) in $\mathbf{X}^*$. While $\mathbf{x}^*_{\cdot,1}$ from our example is additive — the sum of a second order interaction (between x_1, x_2, x_3) and a first order interaction (between x_1, x_2) — the individual contributions of these two terms are tied by the commonality of their shared coefficient, c_1.

2.1.2 Polynomial Space

When each element of $\mathbf{W}$ is in $\{0, 1\}$, the class of polynomials defined by $\mathbf{X}^*\mathbf{c}$ created from (5) includes only the class of polynomials of unshifted inputs — (7) serves an example. The more realistic and practical setting is one in which our polynomial effects arise from shifted inputs, for example, $\mathrm{c}_1\,(\mathrm{x}_1 - a_1)\,(\mathrm{x}_2 - a_2)$.

The key insight here is that, inasmuch as nature might on rare occasion materialize data where our response just so happens to be solely driven by interactions of unshifted inputs (where we can ideally imagine the contents of $\mathbf{W}$ to be in $\{0, 1\}$), thinking more broadly we can easily allow the elements $\mathbf{W}$ to reside in $[0, 1]$. Or, more broadly still, we may allow the elements $\mathbf{W}$ to reside in $\mathbb{R}$.

In this way, the *dual personality* of m-activation is revealed. In the former case, when $\mathbf{W}$ is an indicator, it serves as a hard-and-fast object for messaging interactions in $\mathbf{X}$; in the latter cases, $\mathbf{W}$ becomes another parameter, and the behavior of m-activation is much more akin to a neural net layer.

2.2 Exploration & Confirmation

No matter what the actual values of $\mathbf{W}$, ultimately $\mathbf{X}^*$ is a column-wise collection of inputs — just like $\mathbf{X}$. While the process of locating $\mathbf{W}$ is clearly a matter of model "fitting", it also possesses a spirit of EDA as such a process is essentially searching over the data trying to explain our response, $\mathbf{y}$. However, given some estimate of $\mathbf{W}$ — call it $\widehat{\mathbf{W}}$ — all the familiar tools of confirmatory analysis are available to us. For example, considering our original system (3), if our goal is to minimize our response sum of square errors, then, for $\mathbf{X}^* = g\left(\mathbf{X}\,;\,\widehat{\mathbf{W}}\right)$, everything reduces to simple multivariate regression where our design matrix, $\mathbf{\Xi}$, is constructed

$$\mathbf{\Xi} = [\mathbf{1}, \mathbf{X}, \mathbf{X}^*] \tag{8}$$

bringing with it the usual luxury of confirmatory indicators, t-stats, Information Criteria, diagnostics, etc. — and very importantly, additional exploratory extensions such as *regularization*.

2.3 Statistical Learning

Quite simply, our lone fitting hyper-parameter is m — the number of columns in $\widehat{\mathbf{W}}$ and $\mathbf{X}^*$. Fortuitously, since it seems locating $\mathbf{W}$ is best done sequentially one column after the next, resampling methods such as k-fold CV are efficient, as there is no need to re-locate $\mathbf{W}$ for each possible value of m. For example, suppose $\widehat{\mathbf{W}}^{m=1}_{\text{fold}-1}$ is our estimate when testing for $m = 1$ (and therefore has one column), for CV fold 1. Then we may simply append a column to it and use it for fold-1 to test $m = 2$ (perhaps holding the first column fixed while fitting the values of the second column). And so on for $m = 3, 4, 5, ...$

Since $\mathbf{X}$ is fixed, and $\mathbf{X}^*$ is a function of $\mathbf{X}$, the importance of the consequence of (8) can be emphasized by noting that in a descriptive setting, the quality of a candidate fit is uniquely and solely determined by $\widehat{\mathbf{W}}$; in an inferential setting, our "linear" solution is amenable to the usual extensions, such as regularization.

References

[1] E. Candes, T. Tao, et al. The dantzig selector: Statistical estimation when p is much larger than n. *The annals of Statistics*, 35(6):2313–2351, 2007.

[2] J. Elith, J. R. Leathwick, and T. Hastie. A working guide to boosted regression trees. *Journal of Animal Ecology*, 77(4):802–813, 2008.

[3] J. Fan and R. Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American statistical Association*, 96(456):1348–1360, 2001.

[4] M. García-Magariños, I. López-de Ullibarri, R. Cao, and A. Salas. Evaluating the ability of tree-based methods and logistic regression for the detection of snp-snp interaction. *Annals of human genetics*, 73(3):360–369, 2009.

[5] N. Hao and H. H. Zhang. Interaction screening for ultrahigh-dimensional data. *Journal of the American Statistical Association*, 109(507):1285–1301, 2014.

[6] S. M. Jayakumar, W. M. Czarnecki, J. Menick, J. Schwarz, J. Rae, S. Osindero, Y. W. Teh, T. Harley, and R. Pascanu. Multiplicative interactions and where to find them. In *International Conference on Learning Representations*, 2019.

[7] M. Kuhn and K. Johnson. *Feature Engineering and Selection: A Practical Approach for Predictive Models*. Chapman & Hall/CRC Data Science Series. Chapman and Hall/CRC, 2019.

[8] E. Lampa, L. Lind, P. M. Lind, and A. Bornefalk-Hermansson. The identification of complex interactions in epidemiology and toxicology: a simulation study of boosted regression trees. *Environmental health*, 13(1):1–17, 2014.

[9] P. Radchenko and G. M. James. Variable selection using adaptive nonlinear interaction structures in high dimensions. *Journal of the American Statistical Association*, 105(492):1541–1553, 2010.

[10] M. Schmitt. On the complexity of computing and learning with multiplicative neural networks. *Neural Computation*, 14(2):241–301, 2002.

[11] J. W. Schnupp and A. J. King. Neural processing: the logic of multiplication in single neurons. *Current Biology*, 11(16):R640–R642, 2001.

[12] R. D. Shah. Modelling interactions in high-dimensional data with backtracking. *Journal of Machine Learning Research*, 17(207):1–31, 2016.

[13] D. Sorokina, R. Caruana, and M. Riedewald. Additive groves of regression trees. In *European Conference on Machine Learning*, pages 323–334. Springer, 2007.

[14] M. Tsang, D. Cheng, and Y. Liu. Detecting statistical interactions from neural network weights. *arXiv preprint arXiv:1705.04977*, 2017.

[15] H. Wang, S.-H. Lo, T. Zheng, and I. Hu. Interaction-based feature selection and classification for high-dimensional biological data. *Bioinformatics*, 28(21):2834–2842, 09 2012.

[16] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.

3 Appendix

3.0.1 Metaheuristics

Metaheuristic stochastic search (MSS) is often slow, and commonly regarded as inelegant. I'd be surprised if, over the last 50 years, anyone attained any acclaim over an MSS implementation. However, there is a certain poetry to MSS. First, it *works*. MSS happily dances around iteratively over the parameter space in a semi-directed fashion, constantly testing tries against the objective function, sidling towards the best and away from the worst. Not surprisingly — at least to the author — so too in our present setting. Working with simulated examples both large and small, MSS happily located our model parameters. The real charm of MSS is that it is completely indifferent to *properties* the objective function — it only requires the objective function itself. There's no need to calculate any derivatives, posterior distributions, or the like, which may be why it is often ignored by hardened academicians.

Another nice property of MSS is that the search is amenable to asynchronous iteration and hence parallelization. MSS does possess drawbacks. Most notoriously, it's slow. Moreover, MSS comes in many variants, and its success on a particular model can be highly dependent on the particular variant applied and tuning parameters such as search radius, sequence or order of model parameters over which to search, stopping rules, and the like.

3.0.2 First-Order Descent

Again, working with a number simulated examples both large and small, recursive first-order descent,

$$\boldsymbol{\theta}^{+} = \boldsymbol{\theta} - a \, \frac{\partial f}{\partial \boldsymbol{\theta}} \tag{9}$$

where f is our objective function, and a is our step size, reliably located our parameters. This method is sensitive to the step size, a.

3.0.3 Hybrid

Each recursive step comprises two distinct parts. The first, estimate b_0 and $\mathbf{b}$. Then estimate $\mathbf{c}$ and $\mathbf{W}$.

For the sake of notational simplicity, let's have $\mathbf{b}$ include b_0, $\mathbf{X}_{\mathrm{int}}$ respectively include a column of

1s.

$$^{(i)}\mathbf{X}^{*} = g(\mathbf{X}, {}^{(i)}\mathbf{W}) \tag{10}$$

$$^{(i)}\mathbf{y}_{\text{nocw}} = {}^{(i)}\mathbf{X}^{*}\ {}^{(i)}\mathbf{c} \tag{11}$$

$$^{(i)}\mathbf{e}_{\text{nocw}} = \mathbf{y} - {}^{(i)}\mathbf{y}_{\text{nocw}} \tag{12}$$

$$^{(i)}\mathbf{b} = \underset{{}^{(i)}\mathbf{b}}{\mathsf{argmin}} \left\{ \left\| {}^{(i)}\mathbf{e}_{\text{nocw}} - \mathbf{X}_{\text{int}}\ {}^{(i)}\mathbf{b} \right\| \right\} \qquad \text{(Least Squares Solution)} \tag{13}$$

$$^{(i)}\widehat{\mathbf{y}}_{\mathbf{b}} = \mathbf{X}_{\text{int}}\ {}^{(i)}\mathbf{b} \tag{14}$$

$$- - - - - - - - - - - - \tag{15}$$

$$^{(i)}\mathbf{e}_{\text{nob}} = \mathbf{y} - {}^{(i)}\widehat{\mathbf{y}}_{\mathbf{b}} \tag{16}$$

$$^{(i)}\mathbf{c}\,,\ {}^{(i)}\mathbf{W} = \underset{{}^{(i)}\mathbf{c}\,,\ {}^{(i)}\mathbf{W}}{\mathsf{argmin}} \left\{ \left\| {}^{(i)}\mathbf{e}^{2}_{\text{nob}} - {}^{(i)}\mathbf{X}^{*}\ {}^{(i)}\mathbf{c} \right\| \right\} \qquad \text{(Gradient Descent)} \tag{17}$$