Article from

Predictive Analytics & Futurism

December 2018

Issue 19

A Math Test for Models

By Jeff Heaton

In my opinion, feature engineering is one of the most critical components of any predictive analytics project. Feature engineering is a preprocessing step where additional features, or input columns, are calculated to augment or replace the original features. This technique is closely related to the similar strategies of interaction terms and column transformations.

Feature engineering is often cited as one of the most important components in many winning Kaggle competition entries. There have been many attempts to automate feature engineering. Techniques such as auto encoders, deep feature synthesis and various dimensionality reduction algorithms can provide new features that provide additional lift to models. However, feature engineering remains one of the key areas where a human data scientist excels over their mechanical automated machine learning counterparts. I suspect that when this tide shifts, we will see Kaggle leaderboards dominated by automatic machine learning entries.

MY STRATEGIES FOR FEATURE ENGINEERING

At the highest level, there are two types of features that can be engineered. The first type are simple transformations and interactions between the existing features. For this type, you might take the log of one of the features or divide one feature by another. All of the information needed to create these features is contained entirely within the data set itself. The second type of engineered feature is an augmentation. For this feature you tap external data sources to bring more meaning to features you already have. Consider a column that contains applicant's zip codes. Alone, a zip code is difficult to use in a model. However, you might use a table that contains the coordinates of the center of these zip codes to calculate distance to a major metropolitan area.

The first type of engineered feature is the focus of this article. We will examine what types of transformations and interactions will be the most effective for a variety of models. Some model types have the ability to automatically incorporate interactions. Neural networks and tree-based models in particular have this capability. I published a paper with the IEEE that explored the effectiveness of various models at self-engineering certain types of features on their own. For example, if a given model is often effective at engineering a particular form of feature, you will probably not increase the lift of that model by adding that feature type. By feature types I mean ratios, power transformations, log transformations and others.

To explore the automatic feature engineering capabilities of these model types, I created a "math test for models." I selected four model types and 10 different equation formats. For each of these equation types, I generated training data where the outcome was the result of the given equation. No noise was used. I was interested only in how well the given model could approximate the selected equation type. The results showed two interesting results. The first is that some models were more effective at certain equation types than others. The second is that certain equations are indeed much more difficult for these models to approximate. This can serve as a guide to the structure of engineered features to consider for a particular model type. The models examined in this research were: neural networks, support vector machines, random forests and gradient boosting machines (GBM). The generalized linear model (GLM) family was not considered due to their inability to automatically express interactions and transformations.

DESIGNING A MATH TEST FOR MODELS

To evaluate the automatic feature engineering effectiveness of the four model types a total of 10 different equation forms were used. The models were tested on their ability to approximate these 10 functions. If the model can easily approximate a function then it can probably automatically engineer a similar feature. The 10 different equation types are provided in Table 1.

Table 1
Ten Different Equation Types

#	Name	Expression
1	Difference	$x_1 - x_2$
2	Log	$\log(x_1)$
3	Polynomial	$8x_1^2 + 5x_1 + 1$
4	Polynomial2	$5x_1^2x_2^2 + 4x_1x_2 + 2$
5	Power	x_1^2
6	Ratio	$\frac{x_1}{x_2}$
7	Ratio Difference	$\frac{x_1 - x_2}{x_3 - x_4}$
8	Ratio Polynomial	$\frac{1}{8x_1^2 + 5x_1 + 1}$
9	Ratio Polynomial2	$\frac{1}{5x_1^2x_2^2 + 4x_1x_2 + 2}$
10	Square Root	$\sqrt{x_1}$

Simple transformations, such as log, contain only one value for x, whereas expressions like difference and polynomial contain two. The most complex equation, the ratio difference, contains four x values.

RESULTS OF THE TEST

The first step is to generate the training data. This is done by uniformly generating 10,000 random inputs for all of the x values of each equation. The y values are the result of each equation. There is no noise generated.

The math test was conducted by running each of the 10 equations against each of the four model types a total of five cycles. This results in 200 total runs. This entire process takes approximately 30 minutes to complete. The complete source code for this experiment can be found at the author's GitHub repository.[1] The five cycles are due to the fact that some of these models make use of random numbers in their training. The best (lowest) root mean square error (RMSE) score of all five cycles for each model and equation type are given by Table 2.

Table 2
Best RMSE Scores

	SVM	RF	GBM	Neural
Diff	0.03	0.00	0.01	0.00
Log	0.09	0.00	0.00	0.01
Poly	0.06	0.00	0.00	0.01
Poly2	0.05	0.02	0.02	0.01
Power	0.07	0.01	0.01	0.07
Ratio	0.66	0.14	0.21	0.15
R.Diff	28.57	100.20	204.40	28.27
R.Poly	0.03	0.00	0.00	0.00
R.Poly2	0.06	0.00	0.00	0.00
Sqrt	0.10	0.00	0.00	0.01

All errors are measured in RMSE. I did consider normalizing the output of these equations. While the domain of each equation is intentionally set to between -10 and +10, the range varies depending on the equation. Some of the equations have much larger ranges than others. A common normalization in this case is to divide the RMSE by either the mean or difference of the maximum and minimum y values. Because RMSE is in the same units as the y-value, a function with a large range will likely always have a larger RMSE than one with a small range.

I decided not to normalize because I care how closely the model approximates the function. The actual range of the function does not matter. I simply care how close the approximation is. If the approximation is perfect then the RMSE should approach zero, regardless of how large the range is.

This is shown graphically by Figures 1, 2, 3 and 4. The taller bars in each graph indicate a particular equation type that is more difficult for a given model to approximate. As can be seen from the figures, all of the models had difficulty with the ratio of differences (equation 7). The ratio (equation 6) was impossible for the support vector machine (SVM), but only somewhat more difficult for the other three model types. All other equation types were trivial for the various models to approximate.

Figure 1
Neural Network Results

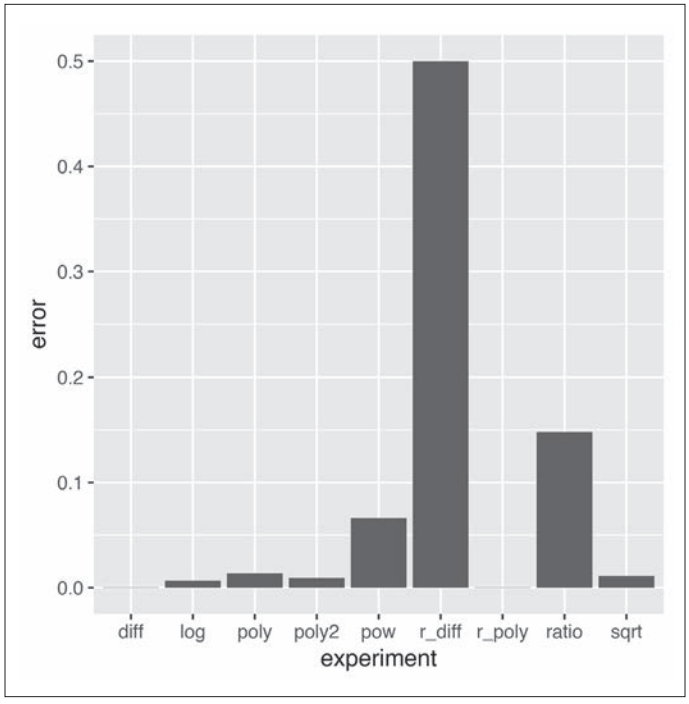


Figure 2
Gradient Boosting Machine (GBM) Results

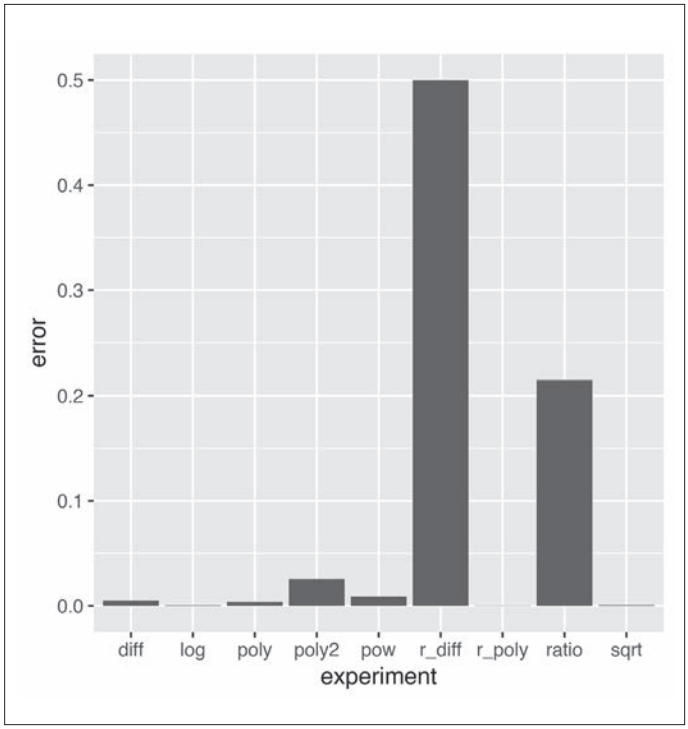


Figure 3
Random Forest Results

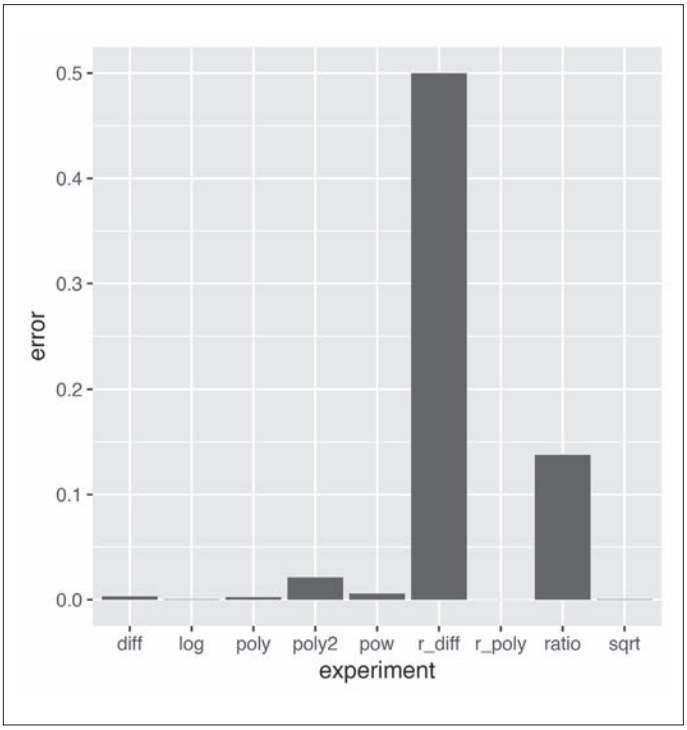
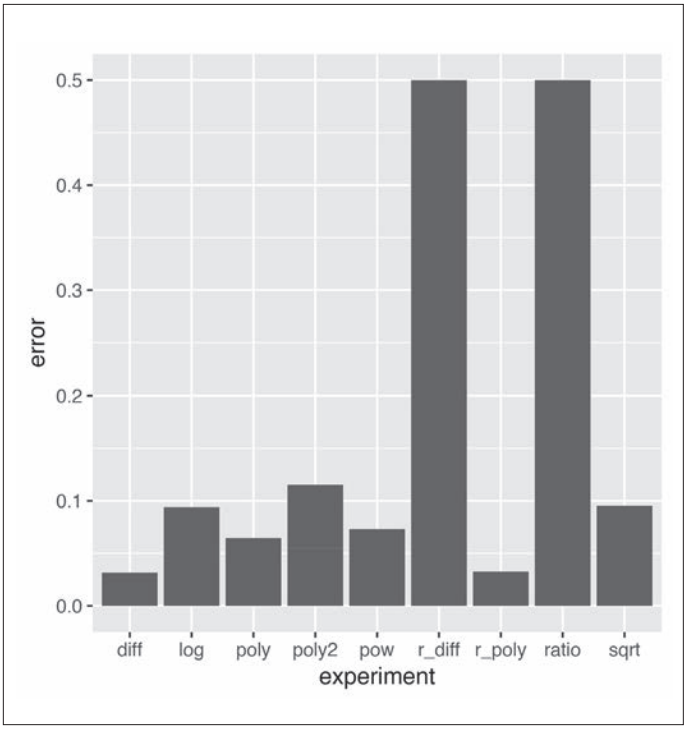


Figure 4
Support Vector Machine (SVM) Results



CONCLUSIONS

The ratio of differences was found to be a difficult feature for all models. When I am engineering my own features I frequently use ratios or a ratio of differences. The ratio by itself is a normalizer. Adding the difference causes it to be a normalizer with thresholds. For example, an engineered feature that I recently created is as follows:

(Home price – mean home price) / (age – mean age)

This is the ratio of two differences, so it has the potential to help any of the four model types. This essentially looks at how much above or below an individual's house is from the mean. However, this is then normalized by how old the individual is relative to their zip code. This reflects the fact that older individuals typically have more expensive houses than younger. Adding this calculated feature provided lift to my model.

As I continue this line of research I will introduce additional equation types to see which ones prove the most challenging for each model. I will also look at how the models might be augmented to perhaps have a chance of engineering a feature of this form. ■

Jeff Heaton, Ph.D. is VP, Data Science, RGA Reinsurance Company, in Chesterfield, Mo. He can be reached at *jheaton@rgare.com.*

ENDNOTE

1 *https://github.com/jeffheaton/present/tree/master/SOA/paf-mathtest*

REFERENCES

Heaton, J. (2016, March). An empirical analysis of feature engineering for predictive modeling. In SoutheastCon 2016 (pp. 1-6). IEEE.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016, November). Tensorflow: a system for large-scale machine learning. In OSDI (Vol. 16, pp. 265-283).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research,* 12(Oct), 2825-2830.