An Empirical Analysis of Feature Engineering for Predictive Modeling

Jeff Heaton
College of Engineering and Computing
Nova Southeastern University
Ft. Lauderdale, FL 33314
Email: jeffheaton@acm.org

Abstract—**Machine learning models, such as neural networks, decision trees, random forests and gradient boosting machines accept a feature vector and provide a prediction. These models learn in a supervised fashion where a set of feature vectors with expected output is provided. It is very common practice to engineer new features from the provided feature set. Such engineered features will either augment, or replace portions of the existing feature vector. These engineered features are essentially calculated fields, based on the values of the other features.**

Engineering such features is primarily a manual, time-consuming task. Additionally, each type of model will respond differently to different types of engineered features. This paper reports on empirical research to demonstrate what types of engineered features are best suited to which machine learning model type. This is accomplished by generating several datasets that are designed to benefit from a particular type of engineered feature. The experiment demonstrates to what degree the machine learning model is capable of synthesizing the needed feature on its own. If a model is capable of synthesizing an engineered feature, it is not necessary to provide that feature. The research demonstrated that the studied models do indeed perform differently with various types of engineered features.

I. INTRODUCTION

Feature engineering is an important but labor-intensive component of machine learning applications [1]. Most machine-learning performance is heavily dependent on the representation of the feature vector. As a result, much of the actual effort in deploying machine learning algorithms goes into the design of preprocessing pipelines and data transformations [1].

To make use of feature engineering a model's feature vector is expanded by adding new features that are calculations based on the other features [2]. These new features might be ratios, differences, or other mathematical transformations of existing features. This process is similar to transformations that human analysts perform as they construct new features such as body mass index (BMI), wind chill, or Triglyceride/HDL cholesterol ratio to help understand the interactions of existing features.

Kaggle and ACM's KDD Cup have seen feature engineering play a very important part in several winning submissions. Feature engineering was successfully applied to the winning KDD Cup 2010 competition entry [3]. Additionally, the Kaggle Algorithmic Trading Challenge was won with an ensemble of models and feature engineering. The features engineered for these competitions were created by hand.

Technologies such as deep learning [4] can benefit from feature engineering. Most research into feature engineering in the deep learning space has been in the areas of image and speech recognition [1]. Such techniques are successful in the high-dimension space of image processing and often amount to dimensionality reduction techniques [5] such as PCA [6] and auto-encoders [7].

II. BACKGROUND AND PRIOR WORK

Feature engineering grew out of the desire to transform linear regression inputs that are not normally distributed. Such transformation can be helpful for linear regression. The seminal work by George Box and David Cox in 1964 introduced a method for determining which of several power functions might be a useful transformation for the outcome of linear regression [8]. This became known as the Box-Cox transformation.

The alternating conditional expectation (ACE) algorithm [9] works similarly to the Box-Cox transformation, in that a mathematical function is applied to each component of the feature vector and outcome. However, unlike the Box-Cox transformation, ACE is able to guarantee optimal transformations for linear regression.

Linear regression is not the only machine-learning model that can benefit from feature engineering and other transformations. In 1999, it was demonstrated that feature engineering could enhance the performance of rules learning for text classification [10]. Feature engineering was successfully applied to the KDD Cup 2010 competition using a variety of machine learning models.

III. EXPERIMENT DESIGN AND METHODOLOGY

Different machine learning model types have varying degrees of mathematical ability. If the model can learn to synthesize an engineered feature on its own, there was no reason to engineer the feature in the first place. This empirically determines what type of engineered feature performs best with which machine-learning model. To accomplish this, ten datasets were generated that each contain the inputs and outputs that correspond to a particular type of engineered feature. If the machine-learning model can learn to reproduce that feature, with a low error, it means that that particular model could have learned that engineered feature without assistance. The following regression machine learning models were examined in this experiment.

- Deep Neural Networks (DANN)
- Gradient Boosted Machines (GBM)
- Random Forests
- Support Vector Machines for Regression (SVR)

To mitigate the stochastic nature of some of these machine learning models, each experiment was run 5 times, and the best run's outcome was used for the comparison. These experiments were conducted in the Python programming language, using the following third-party packages: Scikit-Learn [11], Lasange, and Nolearn. Using this combination of packages, model types of support vector machine (SVM) [12][13], deep neural network [14], random forest [15], and gradient boosting machine (GBM) [16] were evaluated against the following ten selected engineered features:

- Counts
- Differences
- Logarithms
- Polynomials
- Powers
- Ratios
- Rational Differences
- Rational Polynomials
- Root Distance
- Square Roots

The techniques used to create each of these datasets are described in the following sections. The Python source code for these experiments can be downloaded from the author's GitHub page [17].

A. Logarithms and Power Functions

Logarithms and power functions have long been used to transform the inputs to linear regression [8]. The usefulness of these functions for transformation has been shown for other model types, such as neural networks [18]. The log and power transforms used in this paper are of the type shown in Equations 1, 2, and 3.

$$y = \log(x) \tag{1}$$

$$y = x^2 \tag{2}$$

$$y = x^{\frac{1}{2}} \tag{3}$$

This paper investigates using the natural log function, the second power, and the square root. For both the log and root transform, random x values were uniformly sampled in the real number range $[1, 100]$. For the second power transformation, the x values were uniformly sampled in the real number range $[1, 10]$. The partial dataset from the resulting log transformations (Equation 1) is shown in Table I.

A single x_1 observation is used to generate a single y_1 observation. The x_1 values are simply random numbers that produce the expected y_1 values by applying the logarithm function.

x_1	y_1
59.37163	4.08382
14.54385	2.67717
66.54086	4.19782
98.72570	4.59235

x_1	x_2	y_1
0.58961	0.14544	0.44417
0.66203	0.98726	-0.32523
0.89746	0.58317	0.31429
0.06489	0.11745	-0.05256
0.44753	0.34509	0.10244

B. Differences and Ratios

Differences and ratios are common choices for feature engineering. To evaluate this feature type a dataset is generated with two x observations uniformly sampled in the real number range $[0, 1]$, a single y prediction is also generated that is either the difference or ratio of the two observations. When sampling uniform real numbers for the denominator, the range $[0.1, 1]$ is used to avoid division by zero. The differences and ratio transformations are shown by Equations 4 and 5.

$$y = x_1 - x_2 \tag{4}$$

$$y = \frac{x_1}{x_2} \tag{5}$$

Several rows that demonstrate the difference transformation (Equation 4) are shown in Table II. The x_2 value is simply subtracted from the x_1 value resulting in y_1.

C. Counts

The count engineered feature counts the number of elements in the feature vector that satisfies a certain condition. For example, a count feature might be generated that gives the count of other features that are above a specified threshold, such as zero. Equation 6 defines how a count feature might be engineered.

$$y = \sum_{i=1}^{n} 1 \text{ if } x_i > t \text{ else } 0 \tag{6}$$

The x-vector represents the input vector of length n. The resulting y contains an integer equal to the number of x values that were above the threshold (t). To generate a count dataset the resulting y-count was uniformly sampled from integers in the range $[1, 50]$, and corresponding input vectors are created. This process is demonstrated by Algorithm 1.

Several example rows of the count input vector are shown in Table III. The y_1 value simply holds the count of the number of features x_1 through x_{50} that contain a value greater than 0.

Algorithm 1 Generate count test dataset

1: **INPUT:** The number of rows to generate r.
2: **OUTPUT:** A dataset where y contains random integers sampled from $[0, 50]$, and x contains 50 columns randomly chosen to sum to y.
3: **METHOD:**
4:     $x \leftarrow [...\text{empty set}...]$
5:     $y \leftarrow [...\text{empty set}...]$
6: **for** $n \leftarrow 1\ TO\ r$ **do**
7:     $v \leftarrow \text{zeros}(50)$ ▷ Vector of length 50
8:     $o \leftarrow \text{uniform\_random\_int}(0, 50)$ ▷ Outcome(y)
9:     $r \leftarrow o$ ▷ remaining
10:     **while** $r \geq 0$ **do**:
11:        $i \leftarrow \text{uniform\_random\_int}(0, \text{len}(x) - 1)$
12:        **if** $x[i] = 0$ **then**
13:           $v[i] \leftarrow 1$
14:           $r \leftarrow r - 1$
15:     $x.\text{append}(x)$
16:     $y.\text{append}(o)$
      return $[x, y]$

TABLE III
COUNTS TRANSFORMATION

x_1	x_2	x_3	$\ldots$	x_{50}	y_1
1	0	1	$\ldots$	0	2
1	1	1	$\ldots$	1	12
0	1	0	$\ldots$	1	8
1	0	0	$\ldots$	1	5

TABLE IV
POLYNOMIAL TRANSFORMATION

x_1	y_1
1.17922	18.02069
1.97426	42.05279
0.12978	1.78364
0.67695	8.05078

D. Polynomials

Engineered features might take the form of polynomials. This paper investigated the machine learning models' ability to synthesize features that follow the polynomial given by Equation 7.

$$y = 1 + 5x + 8x^2 \tag{7}$$

An equation such as this shows the models' ability to synthesize features that contain several multiplications and an exponent. The data set was generated by uniformly sampling x from real numbers in the range $[0, 2]$. Example observations from this dataset are shown in Table IV. The y_1 value is simply calculated based on x_1 as input to Equation 7.

E. Rational Differences and Polynomials

Useful features might also come from combinations of rational equations of polynomials. Equations 8 & 9 show the types of rational combinations of differences and polynomials tested by this paper.

TABLE V
RATIONAL DIFFERENCE TRANSFORMATION

x_1	x_2	x_3	x_4	y_1
6.30651	2.23126	6.95826	9.88415	-1.39282
9.07714	6.21059	1.58401	1.97679	-7.29794
5.02777	4.04626	3.90232	9.15452	-0.18688
1.90746	7.76275	8.44665	7.09478	-4.33127

TABLE VI
RATIONAL POLYNOMIAL TRANSFORMATION

x_1	y_1
6.30651	0.00286
2.23126	0.01961
6.95826	0.00237
9.88415	0.00120
9.07714	0.00142

TABLE VII
DISTANCE OF QUADRATIC ROOTS

x_1	x_2	x_3	y_1
4.00000	8.00000	-2.00000	2.44949
8.00000	-3.00000	-4.00000	1.46309
3.00000	-1.00000	-9.00000	3.48010
-2.00000	-7.00000	-5.00000	-1.50000

$$y = \frac{x_1 - x_2}{x_3 - x_4} \tag{8}$$

$$y = \frac{1}{5x + 8x^2} \tag{9}$$

To generate a dataset containing rational differences (Equation 8), four observations are uniformly sampled from real numbers of the range $[1, 10]$. Generating a dataset of rational polynomials, a single observation is uniformly sampled from real numbers of the range $[1, 10]$. Several example observations from this training set are shown in Table VI.

F. The Quadratic Equation

It is also useful to see how capable the four machine learning models are at synthesizing common mathematical equations. The final synthesized feature is based on the distance between the roots of a quadratic equation [19]. The distance between roots of a quadratic equation can easily be calculated by taking the difference of the two outputs of the quadratic formula, as given in Equation 10, in its unsimplified form.

$$y = \left| \frac{-b + \sqrt{b^2 - 4ac}}{2a} - \frac{-b - \sqrt{b^2 - 4ac}}{2a} \right| \tag{10}$$

The dataset for the transformation represented by Equation 10 is generated by uniformly sampling x values from the real number range $[-10, 10]$. Such a range will generate some invalid results, which can be discarded. Table VII demonstrates the appearance of this dataset.
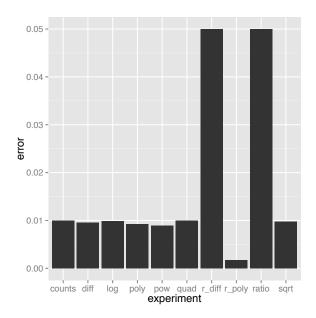
Fig. 1.   Deep Neural Network Engineered Features

IV. RESULTS ANALYSIS

The results obtained by the experiments performed in this paper clearly indicate that some model types perform much better with certain classes of engineered features than other model types. The simple transformations that only involved a single feature were all easily learned by all four models. This included the log, polynomial, power, and root. However, none of the models were able to successfully learn the ratio difference feature. The model specific results from this experiment are summarized in the following sections.

A. Neural Network Results

For each engineered feature experiment a stochastic gradient descent [20] trained deep neural network was used. A learning rate of 1×10^{-5} and a momentum of 0.9 were used. The same set of hyper-parameters were used for each engineered feature experiment. The deep neural network contained a number of input neurons equal to the number of inputs needed to test that engineered feature type. Likewise, a single output neuron was used to provide the value generated by the specified engineered feature. There are 5 hidden layers that, when viewed from the input to the output layer contain 400, 200, 100, 50, and 25 neurons respectively. Each hidden layer makes use of a rectifier transfer function [21], making each hidden neuron a rectified linear unit (ReLU). The results of these deep neural network engineered feature experiments are provided in Figure 1.

All mean square (MSE) error values for Figures 1-4 are clamped at 0.05. For all MSE values above 0.05, the model is considered to have failed the feature engineering experiment. The deep neural network failed to synthesize the ratio and ratio-difference features. All other feature experiments were within an acceptable MSE level.

An examination of the calculations performed by a neural network will provide some insight into this performance. A single-layer neural network is essentially a weighted sum of the input vector transformed by a transfer function, as shown in Equation 11.

$$f(x, w, b) = \phi \left(\sum_n (w_i x_i) + b \right) \tag{11}$$

The vector x represents the input vector, the vector w represents the weights, and the scalar variable b represents the bias. The symbol ϕ represents the transfer function. The experiments in this paper used the rectifier transfer function [21] for hidden neurons and a simple identity linear function for output neurons. The weights and biases are adjusted, as the neural network is trained. A deep neural network contains many layers of these neurons, where each layer can form the input (represented by x) into the next layer. This allows the neural network to be adjusted to perform many mathematical operations, and can explain some of the results shown in Figure 1. The neural network can easily add, sum and multiply. This made the counts, diff, power, and rational polynomial engineered features all relatively easy to synthesize, by using layers of Equation 11.

B. Support Vector Machine Results

The two primary hyper-parameters of a SVM are C and γ. It is customary to perform a grid search to find an optimal combination of C and γ [22]. The 3 C values of 0.001, 1, and 100 were tried, in combination with the 3 γ values of 0.1, 1, and 10. This resulted in 9 different SVMs to evaluate. The experiment results given are from the best combination of C and γ for each feature type. A third hyper-parameter specifies the type of kernel that the SVM uses, which in this case is a Gaussian kernel. Because support vector machines benefit from their input feature vectors being normalized to a specific range [22], we normalized all SVM input to [0,1]. This required normalization step for the SVM does add additional calculations on to the feature being investigated. Therefore, the results obtained for the SVM are not as pure of a feature engineering experiment as the other models. The results of the SVM engineered features are provided in Figure 2.

The support vector machine failed to synthesize the quadratic, ratio, and ratio-difference features. All other feature experiments were within an acceptable MSE level. Smola and Vapnik extended the original support vector machine to include regression; the resulting algorithm is called a support vector regression (SVR) [23]. A full discussion of how a SVR is fitted and calculated is beyond the scope of this paper. However, for the purposes of this paper's research, the primary concern is how a SVR calculates its final output. This calculation can help determine the transformations that a SVR can synthesize. The final output for a SVR is given by the decision function, shown in Equation 12.

$$y = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(x_i, x) + \rho \tag{12}$$
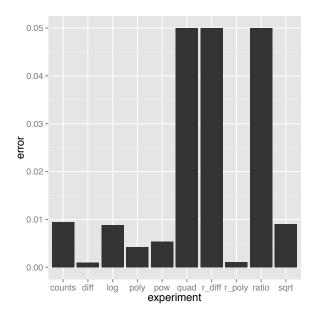
Fig. 2.　SVM Engineered Features

Fig. 3.　Random Forest Engineered Features

The vector x represents the input vector; the difference between the two alphas is called the coefficient of the SVR. The weights of the neural network are somewhat analogous to the coefficients of an SVR. The function K represents a kernel function that introduces non-linearity. This paper used a radial basis function (RBF) kernel based on the Gaussian function. The variable ρ represents the intercept of the SVR, which is somewhat analogous to the bias of a neural network.

Like the neural network, the SVR has the ability to perform multiplications and summations. Though there are many differences between a neural network and SVR, the final calculations share many similarities. Because of these similarities it is not too surprising that the neural network and SVR both fail to synthesize some of the same types of engineered features.

C. Random Forest Results

Random forests are an ensemble model made up of decision trees. The training data is randomly sampled to produce a forest of trees that together, will usually outperform the individual trees. The random forests used in this paper all use 100 classifier trees. This tree count is a hyper-parameter for the random forest algorithm. The result of the random forest model's attempt to synthesize the engineered features is shown in Figure 3.

The random forest model failed to synthesize the counts and ratio-difference features. It is not too surprising that the random forest fails on the synthesized count feature. Trees do not have any inherent way to handle multiple inputs simultaneously other than branching. While a neural network or SVM can produce a count by a simple summation, a tree would need to create branches for every combination of the 50 binary input variables. This would have terrible scalability and would be bounded according to a Catalan number.
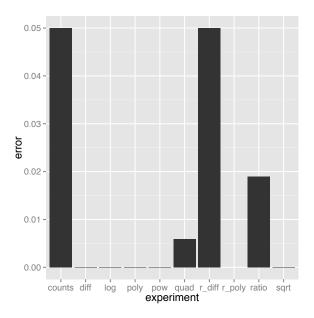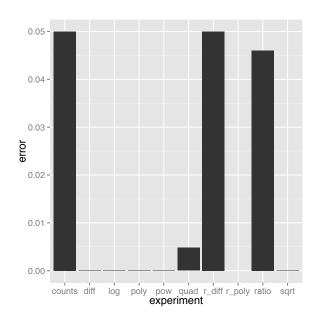

Fig. 4.　Figure 4: GBM Engineered Features

D. Gradient Boosted Machine

The gradient boosted machine (GBM) model operates very similarly to random forests. However, the GBM algorithm uses the gradient of the training objective to attempt to produce optimal combinations of the trees. This additional optimization sometimes gives GBM a performance advantage over random forests. The gradient boosting machines used in this paper all used the same hyper-parameters. The maximum depth was 10 levels, the number of estimators was 100 and the learning rate was 0.05. The results of the GBM engineered features are provided in Figure 4.

Like the random forest model, the gradient boosted machine failed to synthesize the counts and ratio-difference features. Though the gradient boosting machine achieved a satisfactory result on the ratio feature, it performed worse than the random forest.

V. Conclusion & Further Research

Figures 1-4 clearly illustrate that machine learning models such as neural networks, support vector machines, random forests, and gradient boosting machines benefit from a different set of synthesized features. Neural networks and support vector machines generally benefit from the same types of engineered features; similarly, random forests and gradient boosting machines also generally benefit from the same set of engineered features. The results of this research allow for recommendations to be made for both the types of features to use for a particular machine learning model type, as well as the types of models that will work well with each other in an ensemble.

Based on the experiments performed in this research, the type of machine learning model used has a great deal of influence on the types of engineered features to consider. Engineered features based on a ratio of differences were not synthesized well by any of the models explored in this paper. Because of this ratios of difference might be useful to a wide array of models. Neural networks and support vector machines, might also have benefited from engineered features based on ratios. For random forests, and gradient boosting machines, engineered features based on counts could be very useful.

The research performed by this paper also empirically demonstrates one of the reasons why ensembles of models typically perform better than individual models. Because neural networks and support vector machines can synthesize different features than random forests and gradient boosting machines, ensembles made up of a model from each of these two groups might perform very well. A neural network or support vector machine might ensemble well with a random forest or gradient boosting machine.

Significant time was not spend tuning the models for each of the datasets. Rather, reasonably generic choices were made for the hyper-parameters chosen for the models. Results for individual models and datasets might have shown some improvement for additional time spent tuning the hyper-parameters.

Future research will focus on exploring other engineered features with a wider set of machine learning models. Engineered features that are made up of multiple input features seem a logical focus. Possible candidate engineered features for future research include maximums, sums, minimums, means, standard deviations and other functions that could be used over part or all of the input feature vector.

This paper examined 10 different engineered features for four popular machine learning model types. Further research is needed to understand what features might be useful for other machine learning models. Such research could help guide the creation of ensembles that use a variety of machine learning model types. Additional types of engineered features should also be examined. It would be useful to see how more complex classes of features affect the performance of machine learning models.

References

[1] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[2] A. Coates, A. Y. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *International conference on artificial intelligence and statistics*, 2011, pp. 215–223.

[3] H.-F. Yu, H.-Y. Lo, H.-P. Hsieh, J.-K. Lou, T. G. McKenzie, J.-W. Chou, P.-H. Chung, C.-H. Ho, C.-F. Chang, Y.-H. Wei *et al.*, "Feature engineering and classifier ensemble for kdd cup 2010," *KDD Cup*, 2010.

[4] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[5] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[6] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.

[7] B. A. Olshausen *et al.*, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[8] G. E. Box and D. R. Cox, "An analysis of transformations," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 211–252, 1964.

[9] L. Breiman and J. H. Friedman, "Estimating optimal transformations for multiple regression and correlation," *Journal of the American statistical Association*, vol. 80, no. 391, pp. 580–598, 1985.

[10] S. Scott and S. Matwin, "Feature engineering for text classification," in *ICML*, vol. 99. Citeseer, 1999, pp. 379–388.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[12] V. N. Vapnik and A. J. Chervonenkis, "Theory of pattern recognition," 1974.

[13] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[14] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[15] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[16] L. Mason, J. Baxter, P. Bartlett, and M. Frean, "Boosting algorithms as gradient descent in function space." NIPS, 1999.

[17] J Heaton, "Jeff Heaton's GitHub Repository - Conference/Paper Source Code," https://github.com/jeffheaton/papers, accessed: 2016-01-31.

[18] S. D. Balkin and J. K. Ord, "Automatic neural network modeling for univariate time series," *International Journal of Forecasting*, vol. 16, no. 4, pp. 509–515, 2000.

[19] Y. Bugeaud and M. Mignotte, "On the distance between roots of integer polynomials," *Proceedings of the Edinburgh Mathematical Society (Series 2)*, vol. 47, no. 03, pp. 553–556, 2004.

[20] J. C. Spall, *Introduction to stochastic search and optimization: estimation, simulation, and control*. John Wiley & Sons, 2005, vol. 65.

[21] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[22] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," 2003.

[23] A. Smola and V. Vapnik, "Support vector regression machines," *Advances in neural information processing systems*, vol. 9, pp. 155–161, 1997.