An Empirical Analysis of Feature Engineering for Predictive Modeling

Jeff Heaton
McKelvey School of Engineering
Washington University in St. Louis
St. Louis, MO 63130
Email: jtheaton@wustl.edu

***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 we provide feature vectors mapped to the expected output. It is 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 kinds of engineered features. This paper reports empirical research to demonstrate what kinds of engineered features are best suited to various machine learning model types. We provide this recommendation by generating several datasets that we designed to benefit from a particular type of engineered feature. The experiment demonstrates to what degree the machine learning model can synthesize the needed feature on its own. If a model can synthesize a planned 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 essential 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, data scientists spend much of their effort designing preprocessing pipelines and data transformations [1].

To utilize feature engineering, the model must preprocess its input data by adding new features 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 the equations that human analysts design. They construct new features such as body mass index (BMI), wind chill, or Triglyceride/HDL cholesterol ratio to help understand existing features' interactions.

Kaggle and ACM's KDD Cup have seen feature engineering play an essential part in several winning submissions. Individuals applied feature engineering to the winning KDD Cup 2010 competition entry [3]. Additionally, researchers won the Kaggle Algorithmic Trading Challenge with an ensemble of models and feature engineering. These individuals created these engineered features 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 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 non-normally distributed linear regression inputs. Such a 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 technique became known as the Box-Cox transformation.

The alternating conditional expectation (ACE) algorithm [9] works similarly to the Box-Cox transformation. An individual can apply a mathematical function to each component of the feature vector outcome. However, unlike the Box-Cox transformation, ACE can 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, researchers demonstrated that feature engineering could enhance rules learning performance 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 ability to synthesize various types of mathematical expressions. 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. Demonstrating empirically a model's ability to synthesize a particular type of expression shows if engineered features of this type might be useful to that model. To explore these relations, we created ten datasets 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.

For this research, only considered regression machine learning models for this experiment. We chose the following four machine learning models because of the relative popularity and differences in approach.

- 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] and TensorFlow[12]. Using this combination of packages, model types of support vector machine (SVM) [13][14], deep neural network [15], random forest [16], and gradient boosting machine (GBM) [17] were evaluated against the following sixteen selected engineered features:

- Counts
- Differences
- Distance Between Quadratic Roots
- Distance Formula
- Logarithms
- Max of Inputs
- Polynomials
- Power Ratio (such as BMI)
- Powers
- Ratio of a Product
- Rational Differences
- Rational Polynomials
- Ratios
- Root Distance
- Root of a Ratio (such as Standard Deviation)
- 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 [18] or Kaggle [19].

A. Counts

The count engineered feature counts the number of elements in the feature vector that satisfies a certain condition. For example, the program might generate a count feature that counts other features above a specified threshold, such as zero. Equation 1 defines how a count feature might be engineered.

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

The x-vector represents the input vector of length n. The resulting y contains an integer equal to the number of x values above the threshold (t). The resulting y-count was uniformly sampled from integers in the range $[1, 50]$, and the program creates the corresponding input vectors for the program to generate a count dataset. Algorithm 1 demonstrates this process.

Algorithm 1 Generate count test dataset

INPUT: The number of rows to generate r.
OUTPUT: A dataset where y contains random integers sampled from $[0, 50]$, and x contains 50 columns randomly chosen to sum to y.
METHOD:
$x \leftarrow$ [...empty set...]
$y \leftarrow$ [...empty set...]
for $n \leftarrow 1\ TO\ r$ **do**
    $v \leftarrow$ zeros(50) ▷ Vector of length 50
    $o \leftarrow$ uniform_random_int$(0, 50)$ ▷ Outcome(y)
    $r \leftarrow$ o ▷ remaining
    while $r \geq 0$ **do**:
        $i \leftarrow$ uniform_random_int$(0, \text{len}(x) - 1)$
        if $x[i] = 0$ **then**
            $v[i] \leftarrow 1$
            $r \leftarrow r - 1$
    x.append(x)
    y.append(o)
    return $[x, y]$

TABLE I
COUNTS TRANSFORMATION

x_1	x_2	x_3	...	x_{50}	y_1
1	0	1	...	0	2
1	1	1	...	1	12
0	1	0	...	1	8
1	0	0	...	1	5

Several example rows of the count input vector are shown in Table I. 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.

B. Differences and Ratios

Differences and ratios are common choices for feature engineering. To evaluate this feature type a dataset is generated with x observations uniformly sampled in the real number range $[0, 1]$, a single y prediction is also generated that is various differences and ratios of the observations. When sampling uniform real numbers for the denominator, the range $[0.1, 1]$ is used to avoid division by zero. The equations chosen are simple difference (Equation 2), simple ratio (Equation 3), power ratio (Equation 4), product power ratio (Equation 5) and ratio of a polynomial (Equation 6).

$$y = x_1 - x_2 \tag{2}$$

$$y = \frac{x_1}{x_2} \tag{3}$$

$$y = \frac{x_1}{x_2^2} \tag{4}$$

$$y = \frac{x_1 x_2}{x_3^2} \tag{5}$$

$$y = \frac{1}{5x + 8x^2} \tag{6}$$

C. Distance Between Quadratic Roots

It is also useful to see how capable the four machine learning models are at synthesizing ordinary mathematical equations. We generate the final synthesized feature from a distance between the roots of a quadratic equation. 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 7, in its unsimplified form.

$$y = \left| \frac{-b + \sqrt{b^2 - 4ac}}{2a} - \frac{-b - \sqrt{b^2 - 4ac}}{2a} \right| \tag{7}$$

The dataset for the transformation represented by Equation 7 is generated by uniformly sampling x values from the real number range $[-10, 10]$. We discard any invalid results.

D. Distance Formula

The distance formula contains a ratio inside a radical, and is shown in Equation 8. The input are for x values uniformly sampled from the range $[0, 10]$, and the outcome is the Euclidean distance between (x_1, x_2) and (x_3, x_4).

$$y = \sqrt{(x_1 - x_2)^2 + (x_3 - x_4)^2} \tag{8}$$

E. Logarithms and Power Functions

Statisticians have long used logarithms and power functions to transform the inputs to linear regression [8]. Researchers have shown the usefulness of these functions for transformation for other model types, such as neural networks [20]. The log and power transforms used in this paper are of the type shown in Equations 9,10, and 11.

$$y = \log(x) \tag{9}$$

$$y = x^2 \tag{10}$$

$$y = x^{\frac{1}{2}} \tag{11}$$

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 $[1, 10]$. 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.

F. Max of Inputs

Ten random inputs are generated for the observations $(x_1 - x_10)$. These random inputs are sampled uniformly in the range $[1, 100]$. The outcome is the maximum of the observations. Equation 12 shows how this research calculates the max of inputs feature.

$$y = \max(x_1 ... x_{10}) \tag{12}$$

G. 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 13.

$$y = 1 + 5x + 8x^2 \tag{13}$$

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 $[0, 2)$. The y_1 value is simply calculated based on x_1 as input to Equation 13.

H. Rational Differences and Polynomials

Useful features might also come from combinations of rational equations of polynomials. Equations 14 & 15 show the types of rational combinations of differences and polynomials tested by this paper. We also examine a ratio power equation, similar to the body mass index (BMI) calculation, shown in Equations 16.

$$y = \frac{x_1 - x_2}{x_3 - x_4} \tag{14}$$

$$y = \frac{1}{5x + 8x^2} \tag{15}$$

$$y = \frac{x_1}{x_2^2} \tag{16}$$

To generate a dataset containing rational differences (Equation 14), 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 $[1, 10]$.

IV. Results Analysis

To evaluate the effectiveness of the four model types over the sixteen different datasets we must account for the differences in ranges of the y values. As Table II demonstrates, the maximum, minimum, mean, and standard deviation of the datasets varied considerably. Because an error metric, such as root mean square error (RMSE) is in the same units as its corresponding y values, some means of normalization is needed. To allow comparison across datasets, and provide this normalization, we made use of the normalized root-mean-square deviation (NRMSD) error metric shown in Equation 17. We capped all NRMSD values at 1.5; we considered values higher than 1.5 to have failed to synthesize the feature.

$$\text{NRMSD} = \frac{1}{\sigma} \sqrt{\frac{\sum_{t=1}^{T} (\hat{y}_t - y_t)^2}{T}} \tag{17}$$

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

TABLE II
DATASET OBSERVATION (Y) STATISTICS AND RANGES

name	min	max	mean	std
bmi	6.25	87.77	37.22	18.22
counts	0.0	49.0	24.60	14.47
dev	7.56	41.70	27.01	4.57
diff	-0.99	0.98	-0.007	0.40
dist	0.07	11.55	4.71	2.25
log	0.00	4.60	3.66	0.87
max	37.89	99.99	90.87	8.38
poly	1.00	42.99	16.79	12.37
pow	1.00	99.99	37.29	29.25
quad	0.0	11.66	1.59	2.03
r_diff	-24,976.28	4,205.68	-2.83	259.28
r_poly	0.001	0.07	0.009	0.01
ratio	2.36e-05	93.92	2.33	5.32
rel	0.015	88.38	3.15	6.63
sqrt	1.00	9.99	6.75	2.28
sum	16.82	81.38	49.95	9.16

TABLE III
MODEL SCORES FOR DATASETS

Feature	Score SVM	Score RF	Score GBM	Score Neural
bmi	0.00	0.00	0.00	0.03
counts	0.00	0.08	0.10	0.00
dev	0.03	0.45	0.32	0.05
diff	0.07	0.00	0.01	0.00
dist	0.03	0.17	0.12	0.01
log	0.07	0.00	0.00	0.00
max	0.36	0.06	0.12	0.07
poly	0.00	0.00	0.00	0.00
pow	0.00	0.00	0.00	0.00
quad	0.56	0.04	0.04	0.03
r_diff	0.99	1.39	1.5	1.00
r_poly	1.5	0.00	0.00	0.02
ratio	0.28	0.03	0.05	0.03
rel	0.07	0.05	0.09	0.02
sqrt	0.03	0.00	0.00	0.00
sum	0.00	0.33	0.28	0.00

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. Table III provides the scores for each equation type and model. The model specific results from this experiment are summarized in the following sections.

A. Neural Network Results

For each engineered feature experiment, create an ADAM [21] trained deep neural network. We made use of a learning rate of 0.001, β_1 of 0.9, β_2 of 0.999, and ϵ of 1×10^{-7}, the default training hyperparameters for Keras ADAM.

The deep neural network contained the number of input neurons equal to the number of inputs needed to test that engineered feature type. Likewise, a single output neuron provided the value generated by the specified engineered feature. When viewed from the input to the output layer, there are five hidden layers, containing 400, 200, 100, 50, and 25 neurons, respectively. Each hidden layer makes use of a rectifier transfer function [22], making each hidden neuron a rectified linear unit (ReLU). We provide the results of these deep neural network engineered feature experiments in Figure 1.

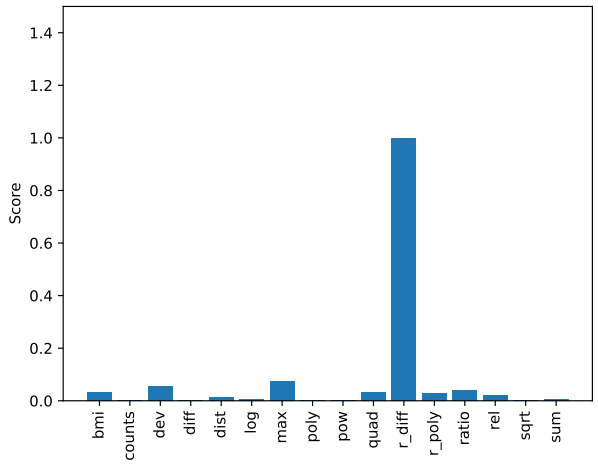

Fig. 1. Deep Neural Network Engineered Features

The deep neural network performed well on all equation types except the ratio of differences. The neural network also performed consistently better on the remaining equation types than the other three models. 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 18.

$$f(x, w, b) = \phi\left(\sum_{n}(w_i x_i) + b\right) \tag{18}$$

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. This paper's experiments used the rectifier transfer function [22] 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 fact allows the neural network to be adjusted to perform many mathematical operations and explain some of the results shown in Figure 1. The neural network can easily add, sum, and multiply. This fact made the counts, diff, power, and rational polynomial engineered features all relatively easy to synthesize by using layers of Equation 18.

B. Support Vector Machine Results

The two primary hyper-parameters of an SVM are C and γ. It is customary to perform a grid search to find an optimal combination of C and γ [23]. We tried 3 C values of 0.001, 1, and 100, combined with the 3 γ values of 0.1, 1, and 10. This selection resulted in 9 different SVMs to evaluate. The experiment results are from the best combination of C and γ for each feature type. A third hyper-parameter specifies

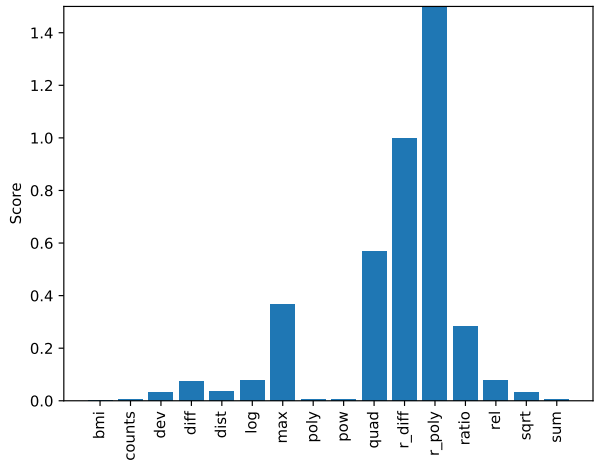

Fig. 2. SVM Engineered Features

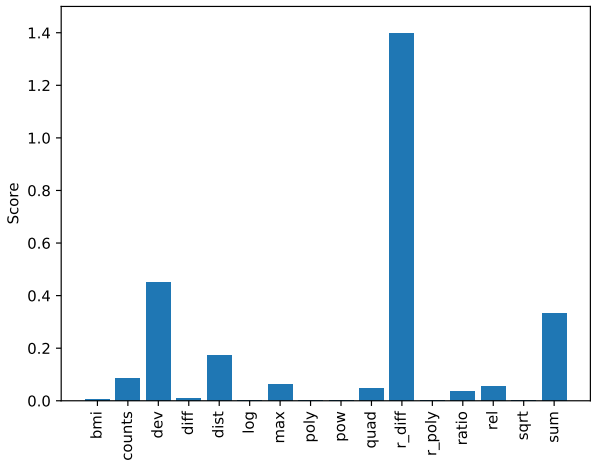

Fig. 3. Random Forest Engineered Features

the type of kernel that the SVM uses, which is a Gaussian kernel. Because support vector machines benefit from their input feature vectors normalized to a specific range [23], we normalized all SVM input to [0,1]. This required normalization step for the SVM does add additional calculations to the feature investigated. Therefore, the SVM results are not as pure of a feature engineering experiment as the other models. We provide the results of the SVM engineered features in Figure 2.

The support vector machine found the max, quadratic, ratio of differences, a polynomial ratio, and a ratio all difficult to synthesize. All other feature experiments were within a low NRMSD level. Smola and Vapnik extended the original support vector machine to include regression; we call the resulting algorithm a support vector regression (SVR) [24]. A full discussion of how an SVR is fitted and calculated is beyond the scope of this paper. However, for this paper's research, the primary concern is how an SVR calculates its final output. This calculation can help determine the transformations that an SVR can synthesize. The final output for an SVR is given by the decision function, shown in Equation 19.

$$y = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(x_i, x) + \rho \tag{19}$$

The vector x represents the input vector; the difference between the two alphas is called the SVR's coefficient. 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 SVR intercept, which is somewhat analogous to the bias of a neural network.

Like the neural network, the SVR can perform multiplications and summations. Though there are many differences between a neural network and SVR, the final calculations share many similarities.

C. Random Forest Results

Random forests are an ensemble model made up of decision trees. We randomly sampled the training data 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. We show the result of the random forest model's attempt to synthesize the engineered features in Figure 3.

The random forest model had the most difficulty with the standard deviation, a ratio of differences, and sum.

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 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 ten levels, the number of estimators was 100, and the learning rate was 0.05. We provide the results of the GBM engineered features in Figure 4.

Like the random forest model, the gradient boosted machine had the most difficulty with the standard deviation, the ratio of differences, and sum.

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 typically benefit from the same set of engineered features. The results of this research allow us to make recommendations for both the types of features to use

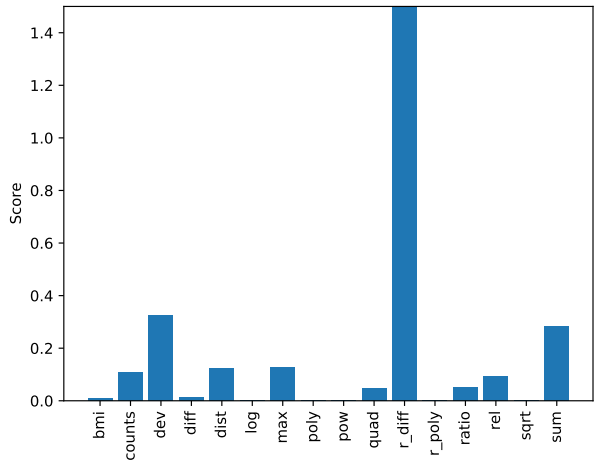

Fig. 4. Figure 4: GBM Engineered Features

for a particular machine learning model type and 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 these ratios of difference might be useful to a wide array of models, all models explored here might benefit from engineered features based on ratios with differences.

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.

We did not spend significant time tuning the models for each of the datasets. Instead, we made reasonably generic choices 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.

This paper examined 16 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. We might also examine additional types of engineered features. It would be useful to see how more complex classes of features affect machine learning models' performance.

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] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

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

[14] 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.

[15] 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.

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

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

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

[19] Heaton, J., "Tabular Feature Engineering Dataset," 2020. [Online]. Available: http://doi.org/10.34740/KAGGLE/DSV/1600809

[20] 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.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

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

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

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