MIT 6.867 Machine Learning
Final Project Milestone 5: Iterate
Improvement of neural network performance with feature engineering?

Irina Degtiar, Greyson Liu, Aaron Sonabend

I. Introduction

Feature engineering transforms the original inputs onto a new feature space that hopefully improves data separability. Non-linear classifiers such as neural networks allow for features to be adaptively created by transforming the original input data. However, proper feature engineering has the potential to further improve algorithm performance. In many machine learning settings, a-priori knowledge of the relationships between features is limited. Exploratory data analysis, however, can uncover covariate relationships that promote target separation. In this report, we explore the impact of feature engineering on neural network prediction and speed in the setting of no a-priori knowledge of feature relationships. We explore whether performance can be improved in various simulations and in an auto insurance claims database.

II. Related work

Feature engineering has been shown to win prediction competitions,[1] but prior work has focused on settings in which domain knowledge can be utilized to create meaningful features. We will explore the setting in which prior knowledge is limited. Simulation studies have shown that neural networks are not able to perform well when the target is determined by certain transformations of covariates such as differences and ratios.[2]

III. Study setup

A. Auto insurance claims data description

In the Kaggle competition Porto Seguro's Safe Driver Prediction, Porto Seguro, a Brazilian insurance company, seeks to improve predictions of a driver filing an auto insurance claim (kaggle.com/c/porto-seguro-safe-driver-prediction). Porto Seguro provides a dataset of 595,212 training and 892,816 test individuals with 59 variables; 3.6% of which filed claims. Variables are de-identified and labeled with type ("cat", "bin", and ordinal/continuous), and with category ("ind", "car", "calc"). 79% of individuals are missing at least one variable, with one of the variables missing for 69% of individuals and all but 5 variables having <1% missingness. No one variable is strongly correlated with the target, but there is up to 89% correlation between variables (Figure 1).

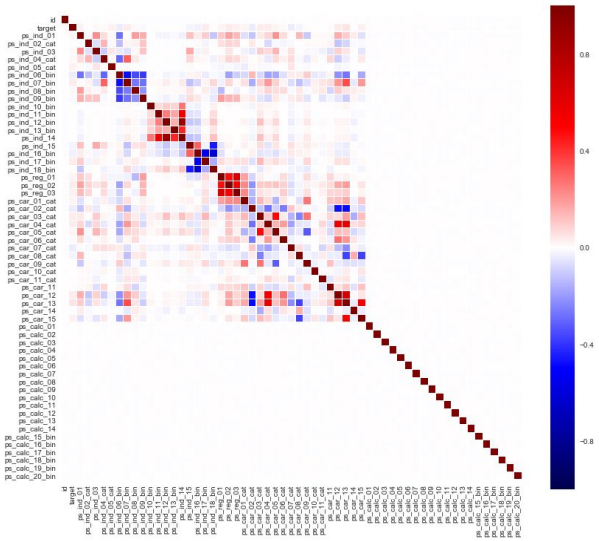

Figure 1. Pearson correlation between original variables

Both correlated and uncorrelated variables originally provided show meaningful interactions that induce target separability (Figure 2), and thus might potentially improve algorithm performance.

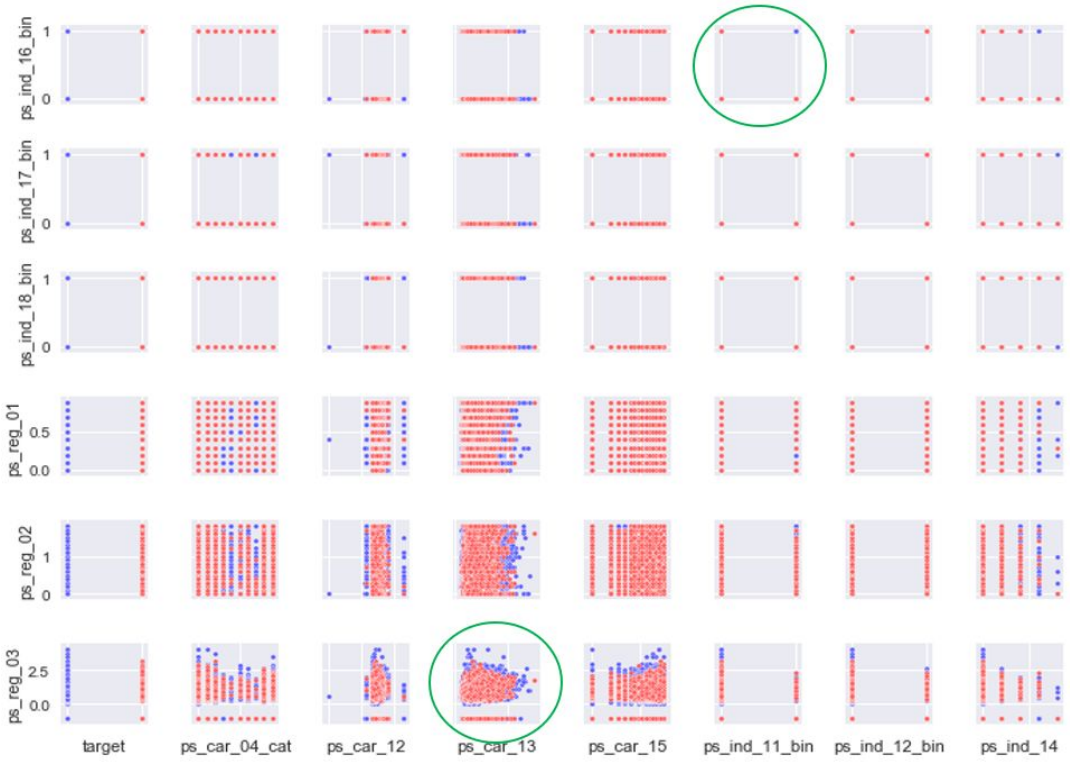

Figure 2. Select pairwise distributions for original variables. Red: target = 1; blue: target = 0. Note separability induced by ps_ind_16_bin*ps_ind_11_bin interaction and ps_reg_03*ps_car_13 interaction.

B. Auto insurance claims feature engineering

Preliminary feature engineering consisted of binary interaction terms between variables that induced data separability visually. We will explore the following additional transformations:

- Count of the number of missing values
- Powers and logarithms
- Ratios and differences

To avoid overfitting, with the new dataset of original and engineered features, we performed feature selection using lasso and random forest, as example linear and nonlinear algorithms. For lasso, we will select a features corresponding to the λ that minimized categorical cross-entropy loss. For random forest, we selected features with a feature importance greater than 0.15.[3] We used features chosen by either algorithm for use in our neural network model.

C. Data cleaning and missing data imputation

We added indicators for missingness for each variable. Missing data was imputed using 5-nearest-neighbors using the “bnstruct” package in R. KNN has pre-built packages for implementation and performs robustly for imputation in large datasets;[4] 5 nearest neighbors were chosen for computational feasibility. Furthermore, for computational efficiency, we randomly shuffled the rows of our data and split the training dataset into 10 pieces. Ten 5-nearest-neighbors algorithm were applied separately within each piece to improve computation time. Each row in the testset was assigned to one of the training set splits and imputated using 5 nearest training neighbors. To standardize data, categorical variables were converted to one-hot encoding; binary variables to +/-1 encoding, which has been shown to be optimal for neural networks;[5] continuous variables were Gaussian normalized (subtracting means, dividing by standard deviations).

D. Simulation

D1. Data generation

To assess under which scenarios feature engineering improves neural network performance, we began with a simulation in which target values depended on a linear combination of variables and 2-way variable interactions, with 2-way interaction features having high weights. In the final report, we will also explore nonlinear relationships.

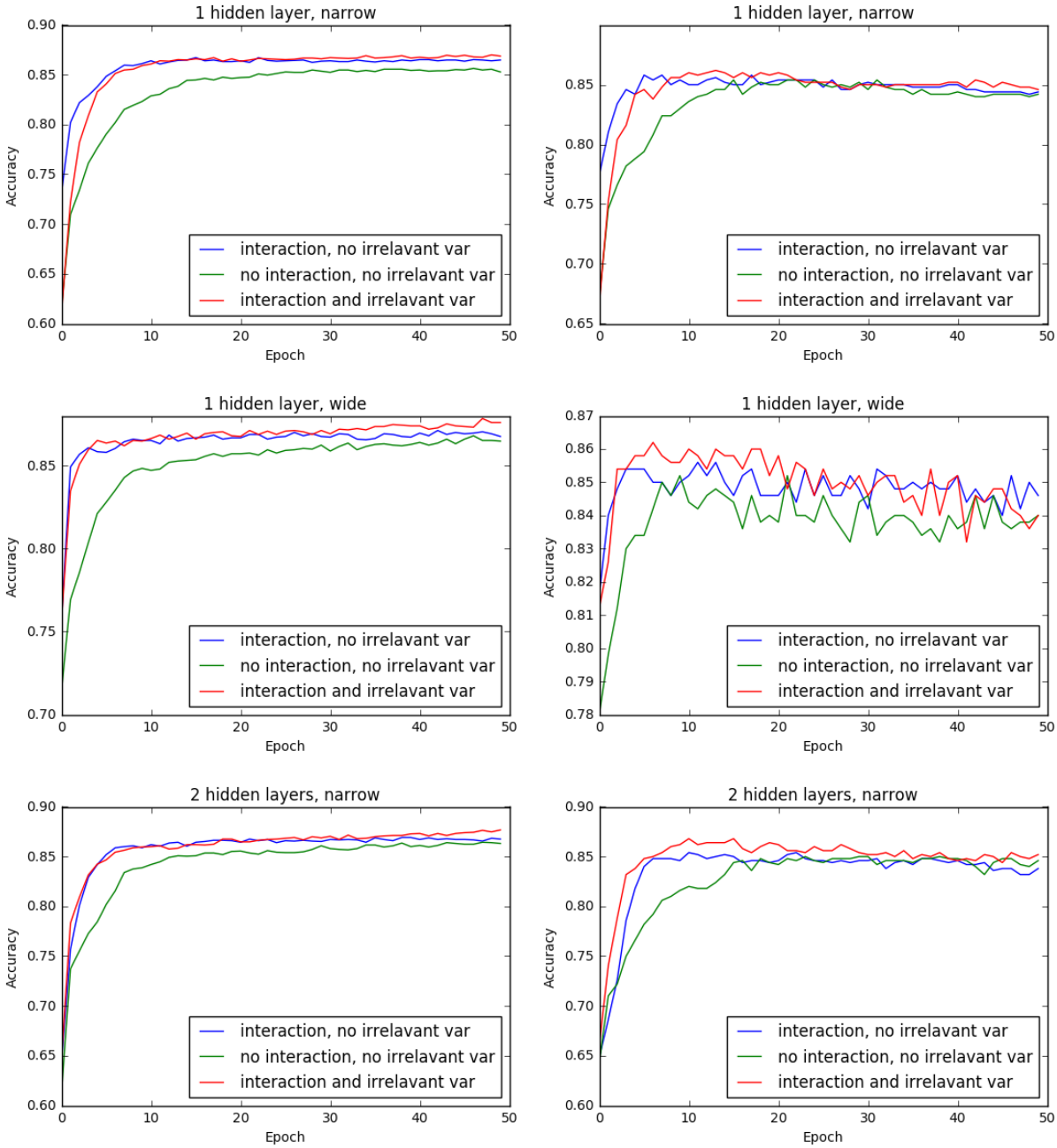

Figure 3. Training (left) and validation (right) accuracies comparing neural networks with different architectures and different inputs

3,000 *iid* samples of a binary outcome were generated by applying the sigmoid function to a linear sum of covariate functions defined as follows:

$X_1, X_2, X_3, X_5 \sim N(0,1)$, X_4, X_6 are four-category variables with equal probability, and $\varepsilon \sim N(0,1)$. Then let

$$\eta = -0.5 + 0.1X_1 - 0.1X_2 + 0.1X_3 - 0.1I(X_4 = 1) + 0.1I(X_4 = 2) - 0.1I(X_4 = 3) + 0.1I(X_4 = 4) + 2X_1X_2 + 0.5X_3I(X_4 = 1) + X_3I(X_4 = 2) + 1.5X_3I(X_4 = 3) + 2X_3I(X_4 = 4) + \varepsilon$$

$$Y = sigmoid(\eta)$$

Note that X_5, X_6 do not appear in the above equation and are generated to study whether adding unrelated variables would affect the performance of neural networks. In this simulation, the main effect of each variable is very weak but the interaction terms are dominant in predicting Y.

D2. Neural network architecture

Three network architectures were experimented with to study the effect of adding interactions: (1) one hidden layer with 20 units, (2) one hidden layer with 100 units, and (3) two hidden layers with 20 and 10 units. We used a ReLU activation function in each hidden layer, sigmoid activation in the outcome layer, and cross-entropy loss.

D3. Simulation study results

Convergence rates and final accuracies for models with interactions were higher than those without interactions under various network architectures (Figure 3). The accuracy of models with irrelevant features is higher than the one only including the true features because even though ε was generated independently, it still had weak correlations with those irrelevant predictors. These results suggest that adding interactions as additional features may speed up the training procedure and improve the prediction accuracy.

E. Auto insurance claims data: neural network model & tuning

In the auto insurance claims data, we also applied neural networks to study (1) whether adding interactions as additional features can speed up the training procedure or (2) improve the Gini coefficient,[6] a measure of prediction accuracy.

We adopted a neural network with four hidden layers, where each layer is activated by a ReLU function and the number of units within each layer is 100, 80, 20, 10, respectively. Those parameters are chosen to maximize the validation Gini coefficient; a thorough exploration has yet to be done and the parameters will be tuned for the final report. The output layer is linked by a softmax function and the loss function is binary cross-entropy.

The algorithm was implemented by the keras package in Python, using the default mini-batch gradient descent method (rmsprop) with batch size 5,000. The dataset is balanced by weighing the claims (originally 3.56% of the sample size) to be 50% of the data in order to train the network and once the training is finished the weights are rebalanced again for the original dataset proportions.

F. Auto insurance claims results

As seen in Figure 4, for the first few epochs, the model that uses interaction terms has a slower learning rate since it is tuning higher-dimension parameters relative to the raw data model. However, as training moves forward, interaction terms add a clear advantage in terms of speed.

On the other hand, Figure 5 shows that both models reach the best validation set Gini coefficient in the beginning of training, after which there is overfitting. It is also clear that the raw-data model outperforms the model with interactions. We are exploring regularization via drop out for both models, as well as additional engineered features.

IV. Conclusions

With the initial simulation results, we found that using the true interactions as features vs. using only raw covariates improves both speed of convergence measured in epochs and accuracy. However, the effect of including true interactions is small, and we do not have a sense of the accuracy

distribution in order to determine if the difference is significant - we plan to explore significance for the final report. Clearly we do not have knowledge of true interactions between features for the real data; thus, including interaction terms led to overfitting and validation accuracy suffered. We will explore additional features for the final report to see if performance can be improved. Currently, results suggest that without domain knowledge to guide feature creation, feature engineering is not worth the added cost; one should instead focus on model architecture tuning to improve performance.

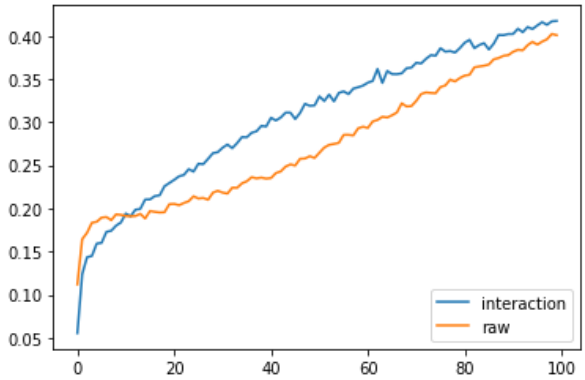

Figure 4. Training set Gini coefficient as a function of epochs for a neural network using interaction and raw features.

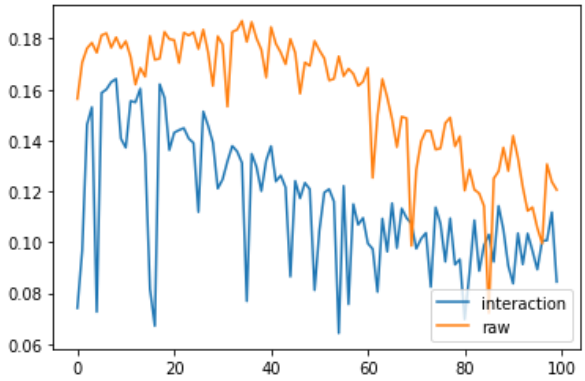

Figure 5. Validation set Gini coefficient as a function of epochs for a neural network using interaction and raw features.

Author contributions:

- ID: data exploration, feature engineering, & standardization, report drafting
- GL: missing data imputation, neural net & resnet training, report results drafting
- AS: simulated data generation, neural net & resnet training, report results drafting

Risks

- Not seeing improvement in results in the auto insurance claim data. Solution: explore in simulations under what settings feature engineering can vs. can't improve results & how strong interaction needs to be to impact performance

Citations

- Project advice and feedback: Curtis Northcutt
- Data exploration: https://www.kaggle.com/arthurtok/introduction-to-ensembling-stacking-in-python https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python https://github.com/mwaskom/seaborn
- Data type extraction: https://www.kaggle.com/felipeleiteantunes/introductory-eda-baseline-with-cv

References

1. Yu, H. F. *et al.* Feature engineering and classifier ensemble for KDD cup 2010. *ntur.lib.ntu.edu.tw* (2010).
2. Heaton, J. An Empirical Analysis of Feature Engineering for Predictive Modeling. *arXiv [cs.LG]* (2017).
3. Albon, C. Feature Selection Using Random Forest - Machine Learning. Available at: https://chrisalbon.com/machine-learning/feature_selection_using_random_forest.html. (Accessed: 21st November 2017)
4. Mandel J, S. P. A Comparison of Six Methods for Missing Data Imputation. *J. Biom. Biostat.* **06,** (2015).
5. McCaffrey, J. How To Standardize Data for Neural Networks -- Visual Studio Magazine. *Visual Studio Magazine* Available at: https://visualstudiomagazine.com/articles/2014/01/01/how-to-standardize-data-for-neural-networks.aspx. (Accessed: 8th November 2017)
6. Moser, J. Code to Calculate NormalizedGini. Available at: https://www.kaggle.com/c/ClaimPredictionChallenge/discussion/703. (Accessed: 21st

November 2017)