Feature Encodings for Gradient Boosting with Automunge

Anonymous Author(s)
Affiliation
Address
email

Abstract

Selecting a default feature encoding strategy for gradient boosted learning may consider metrics of training duration and achieved predictive performance associated with the feature representations. The Automunge library for dataframe preprocessing offers a default of binarization for categoric features and z-score normalization for numeric. The presented study sought to validate those defaults by way of benchmarking on a series of diverse data sets by encoding variations with tuned gradient boosted learning. We found that on average our chosen defaults were top performers both from a tuning duration and a model performance standpoint. Another key finding was that one hot encoding did not perform in a manner consistent with suitability to serve as a categoric default in comparison to categoric binarization. We present here these and further benchmarks.

1 Introduction

The usefulness of feature engineering for applications of deep learning has long been considered a settled question in the negative, as neural networks are on their own universal function approximators Goodfellow et al. (2016). However, even in the context of deep learning, tabular features are often treated with some form of encoding for preprocessing. Automunge Authors (2021b) is a platform for encoding dataframes developed by the authors. This python library was originally built for a simple use case of basic encoding conventions for numeric and categoric features, like z-score normalization and one-hot encodings. Along the iterative development journey we began to flesh out a full library of encoding options, including a series of options for numeric and categoric features that now include scenarios for normalization, binarization, hashing, and missing data infill under automation. Although it was expected that these range of encoding options would be superfluous for deep learning, that does not rule out their utility in other paradigms which could range from simple regression, support vector machines, decisions trees, or as will be the focus of this paper, gradient boosting.

The purpose of this work is to present the results of a benchmarking study between alternate encoding strategies for numeric and categoric features for gradient boosted tabular learning. We were particularly interested in validating the library's default encoding strategies, and found that in both primary performance metrics of tuning duration time and model performance the current defaults under automation of categoric binarization and numeric z-score normalization demonstrated merit to serve as default encodings for the Automunge library. We also found that in addition to our default binarization, even a frequency sorted variant of ordinal encoding on average outperformed one hot encoding.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

2 Gradient Boosting

Gradient boosting Friedman (2000) refers to a paradigm of decision tree learning Quinlan (1986) similar to random forests Breiman (2001) but in which the optimization is boosted by recursively training an iteration's model objective to correct the performance of the preceding iteration's model. It is commonly implemented in practice by the XGBoost library Chen & Guestrin (2016) for GPU acceleration, although there are architecture variations available for different fortes, like LightGPM Ke et al. (2017) which may train faster on CPU's than XGBoost (with a possible performance tradeoff).

Gradient boosting has traditionally been found as a winning solution for tabular modality competitions on the Kaggle platform, and its competitive efficacy has even been demonstrated for more sophisticated applications like time series sequential learning when used for window based regression Elsayed et al. (2021). Recent tabular benchmarking papers have found that gradient boosting may still mostly outperform sophisticated neural architectures like transformers Gorishniy et al. (2021), although even a vanilla multi layer perceptron neural network could have capacity to outperform gradient boosting with comprehensively tuned regularizers Kadra et al. (2021). Gradient boosting can also be expected to have higher latency inference than neural networks Borisov et al. (2021).

Conventional wisdom is that one can expect gradient boosting models to have capacity for better performance than random forests for tabular applications but with a tradeoff of increased probability of overfitting without hyperparameter tuning Howard & Gugger (2020). With both more sensitivity to tuning parameters and a much higher number of parameters in play than random forest, gradient boosting usually requires more sophistication than a simple grid or random search for tuning. One compromise method available is for a sequential grid search through different subsets of parameters Jain (2016), although more automated and even parallelized methods are available by way of black box optimization libraries like Optuna Akiba et al. (2019). There will likely be more improvements to come both in libraries and tuning conventions, this is an active channel of industry research.

3 Feature Encodings

Feature encoding refers to feature set transformations that serve to prepare the data for machine learning. Common forms of feature encoding preparations include normalizations for numeric sets and one hot encodings for categoric, although some learning libraries may accept categoric features in string representations for internal encodings. Before the advent of deep learning, it was common to supplement features with alternate representations of extracted information or to combine features in some fashion. Such practices of feature engineering are sometimes still applied in gradient boosted learning, and it was one of the purposes of these benchmarks to evaluate benefits of the practice in comparison to directly training on the data.

An important distinction of feature encodings can be considered as those that can be applied independent of an esoteric domain profile verses those that rely on external structure. An example could be the difference between supplementing a feature with bins derived based on the distribution of populated numeric values verses extracting bins based on an external database lookup. In the case of Automunge, the internal library of encodings follows almost exclusively the former, that is most encodings are based on inherent numeric or string properties and do not consider adjacent properties that could be inferred based on relevant application domains. (An exception is made for date-time formatted features which under automation automatically extract bins for weekdays, business hours, holidays, and redundantly encodes entries based on cyclic periods of different time scales London (2016).) The library includes a simple template for integrating custom univariate transformations Authors (2021a) if a user would like to integrate into a pipeline alternate conventions.

3.1 Numeric

Numeric normalizations in practice are most commonly applied similar to our default of z-score 'nmbr' (subtract mean and divide by standard deviation) or min-max scaling 'mnmx' (converting to range between 0–1). Other variations that may be found in practice include mean scaling 'mean'(subtract mean and divide by min max delta), and max scaling 'mxab' (divide by feature set absolute max). More sophisticated conventions may convert a distribution shape in addition to the scale, such as the box-cox power law transformation 'bxcx' Box & Cox (1964) or Scikit-Learn's Pedregosa et al.

(2011) quantile transformer 'qttf', which both may serve the purpose of converting a feature set to closer resemble a Gaussian distribution. In general, numeric normalizations are more commonly applied for learning paradigms other than those based on decision trees, where for example in neural networks they serve the purpose of normalizing gradient updates across features. We did find that the type of normalizations applied to numeric features appeared to impact performance, and we will present these findings below.

3.2 Categoric

Categoric encodings are most commonly derived in practice as a one hot encoding, where each unique entry in a received feature is translated to boolean integer activations in a dedicated column among a returned set thereof. The practice of one hot encoding has shortcomings in the high cardinality case (where a categoric feature has an excessive number of unique entries), which in the context of gradient boosting may be particularly impactful as an inflated column count impairs latency performance of a training operation — or when the feature is targeted as a classification label may even cause training to exceed memory overhead constraints. The Automunge library attempts to circumvent this high cardinality edge case in two fashions, first by defaulting to a binarization encoding instead of one hot, and second by distinguishing highest cardinality sets for a hashed encoding Moody (1988) Weinberger et al. (2009) Authors (2021a) which may stochastically consolidate multiple unique entries into a shared ordinal representation for a reduced number of unique entries.

The library default of categoric binarization '1010' refers to translating each unique entry in a received feature to a unique set of zero, one, or more boolean integer activations in a returned set of boolean integer columns. Where one hot encoding may return a set of n columns for n unique entries, binarization will instead return a smaller count of log2(n) rounded up to nearest integer. We have previously seen the practice discussed in the blogging literature, such as Ravi (2019), although without validation as offered herein.

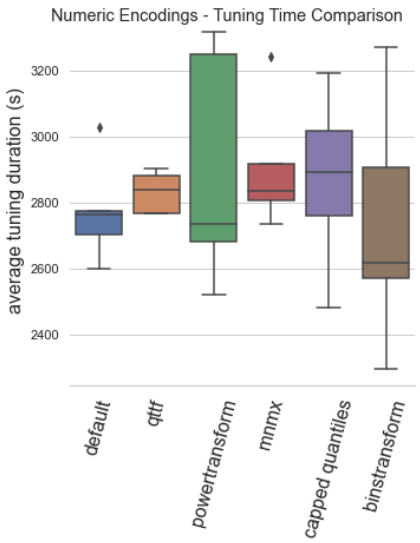
A third common variation on categoric representations includes ordinal encodings, which simply refers to returning a single column encoding of a feature with a distinct integer representation for each unique entry. Variations on ordinal encodings in the library may sort the integer representations by frequency of the unique entry 'ord3' or based on alphabetic sorting 'ordl'.

Another convention for categoric sets unique to the Automunge library we refer to as parsed categoric encodings 'or19' Authors (2020). Parsed encodings search through tiers of string character subsets of unique entries to identify shared grammatical structure for supplementing encodings with structure derived from a training set basis. Parsed encodings are supplemented with extracted numeric portions of unique entries for additional information retention in the form received by training.

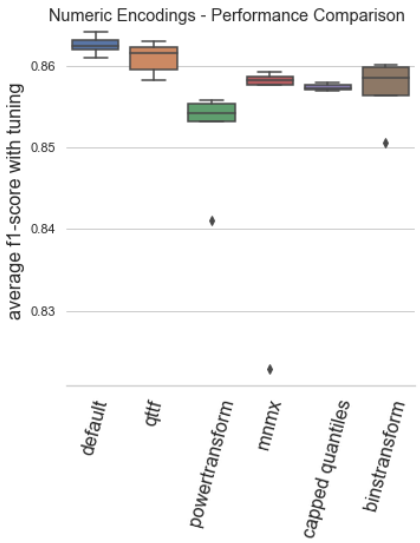
4 Benchmarking

The benchmarking sought to evaluate a range of numeric and categoric encoding scenarios by way of two key performance metrics, training time and model performance. Training was performed over the course of 1.5 weeks on a Lambda workstation with AMD 3970X processor, 128Gb RAM, and two Nvidia 3080 GPUs. Training was performed by way of XGBoost tuned by Optuna with 5-fold fast cross-validation Swersky et al. (2013) and early stopping criteria of 50 tuning iterations without improvement. Performance was evaluated against a partitioned 25% validation set based on a f1 score performance metric, which we understand is a good default for balanced evaluation of bias and variance performance of classification tasks Stevens et al. (2020). This loop was repeated and averaged across 5 iterations and then repeated and averaged across 31 tabular classification data sets sourced from the OpenML benchmarking repository Vanschoren et al. (2014). Rephrasing for clarity, the reported metrics are averages of 5 repetitions of 31 data sets for each encoding type as applied to all numeric or categoric features for training. The distribution bands shown in the figures are across the five repetitions. The data sets were selected for diverse tabular classification applications with in-memory scale training data and tractable label cardinality.

We found that these benchmarks gave us comfort in the Automunge library's defaults of numeric z-score normalization and categoric binarization. An interesting result was the outperformance of categoric binarization in comparison to one-hot encoding, as the latter is commonly used in

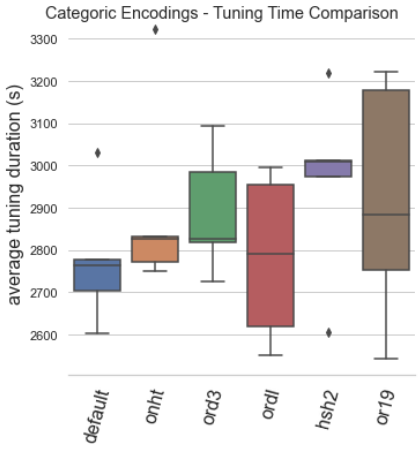


(a) Numeric tuning time comparison

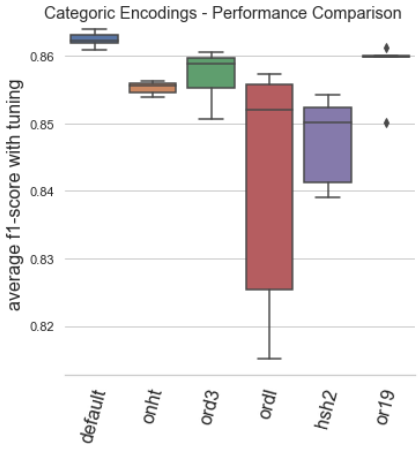


(b) Numeric model performance comparison

Figure 1: Numeric Results



(a) Categoric tuning time comparison



(b) Categoric model performance comparison

Figure 2: Categoric Results

mainstream practice as a default. Further dialogue for the interpretation of the results presented in Figures 1 and 2 are provided in the Appendix.

5 Conclusion

We hope that these benchmarks may have provided some level of user comfort by validating the default encodings applied under automation by the Automunge library of z-score normalization and categoric binarization, both from a training time and model performance standpoint. If you would like to try out the library we recommend the tutorials folder found on GitHub Authors (2021b) as a starting point.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework, 2019. URL https://arxiv.org/abs/1907.10902.

Authors. Parsed categoric encodings with Automunge (anonymous preprint provided in supplemental material), 2020.

Authors. Custom transformations with Automunge (anonymous preprint provided in supplemental material), 2021a.

Authors. Automunge (code repository provided in supplemental material), 2021b.

Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., and Kasneci, G. Deep neural networks and tabular data: A survey, 2021. URL https://arxiv.org/abs/2110.01889.

Box, G. E. P. and Cox, D. R. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964. doi: https://doi.org/10.1111/j.2517-6161.1964.tb00553.x. URL https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1964.tb00553.x.

Breiman, L. Random forests. *Mach. Learn.*, 45(1):5–32, oct 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

Chen, T. and Guestrin, C. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145%2F2939672.2939785.

Elsayed, S., Thyssens, D., Rashed, A., Jomaa, H. S., and Schmidt-Thieme, L. Do we really need deep learning models for time series forecasting?, 2021. URL https://arxiv.org/abs/2101.02118.

Friedman, J. H. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

Goodfellow, I. J., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

Gorishniy, Y., Rubachev, I., Khrulkov, V., and Babenko, A. Revisiting deep learning models for tabular data. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=i_Q1yrOegLY.

Howard, J. and Gugger, S. *Deep Learning for Coders with fastai and PyTorch*. O'Reilly Media, 2020.

Jain, A. Complete guide to parameter tuning in xgboost with codes in python, 2016. URL https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/.

Kadra, A., Lindauer, M., Hutter, F., and Grabocka, J. Well-tuned simple nets excel on tabular datasets. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=d3k38LTDCyO.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

London, I. Encoding cyclical continuous features — 24-hour time, 2016. URL https://ianlondon.github.io/blog/encoding-cyclical-features-24hour-time/.

Moody, J. Fast learning in multi-resolution hierarchies. In Touretzky, D. (ed.), *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. URL https://proceedings.neurips.cc/paper/1988/file/82161242827b703e6acf9c726942a1e4-Paper.pdf.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. URL http://jmlr.org/papers/v12/pedregosa11a.html.

Quinlan, J. R. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

Ravi, R. One-hot encoding is making your tree-based ensembles worse, here's why?, 2019. URL https://towardsdatascience.com/one-hot-encoding-is-making-your-tree-based-ensembles-worse-heres-why-d64b282b5769.

Stevens, E., Antiga, L., and Viehmann, T. *Deep Learning with PyTorch*. 2020. URL https://pytorch.org/assets/deep-learning/Deep-Learning-with-PyTorch.pdf.

Swersky, K., Snoek, J., and Adams, R. P. Multi-task bayesian optimization. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper/2013/file/f33ba15effa5c10e873bf3842afb46a6-Paper.pdf.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. OpenML. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, jun 2014. doi: 10.1145/2641190.2641198. URL https://doi.org/10.1145%2F2641190.2641198.

Weinberger, K., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 1113–1120, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553516. URL https://doi.org/10.1145/1553374.1553516.

A Numeric Results

A.1 default

- defaults for Automunge under automation as z-score normalization ('nmbr' code in the library)
- The default encoding was validated both from a tuning duration and a model performance standpoint as top performing scenario on average.

A.2 qttf

- Scikit-Learn QuantileTransformer with a normal output distribution
- The quantile distribution conversion did not perform as well on average as simple z-score normalization, although it remained a top performer.

A.3 powertransform

- the Automunge option to conditionally encode between 'bxcx', 'mmmx', or 'MAD3' based on distribution properties (via library's powertransform=True setting)
- This was the worst performing encoding scenario, which at a minimum demonstrates that the heuristics and statistical measures currently applied by the library to conditionally select types of encodings could use some refinement.

A.4 mnmx

- min max scaling 'mnmx' which shifts a feature distribution into the range 0–1
- This scenario performed considerably worse than z-score normalization, which we expect was due to cases where outlier values may have caused the predominantly populated region to get "squished together" in the encoding space.

A.5 capped quantiles

- min max scaling with capped outliers at 0.99 and 0.01 quantiles ('mnm3' code in library)
- This scenario is best compared directly to min-max scaling, and demonstrates that defaulting to capping outliers did not benefit performance on average.

A.6 binstransform

- z-score normalization supplemented by 5 one hot encoded standard deviation bins (via library's binstransform=True setting)
- In addition to a widened range of tuning durations, the supplemental bins did not appear to be beneficial to model performance for gradient boosting.

B Categoric Results

B.1 default

- defaults for Automunge under automation for categoric binarization ('1010' code in the library)
- The default encoding was validated as top performing both from a tuning duration and a model performance standpoint.

B.2 onht

- one hot encoding
- The model performance impact was surprisingly negative compared to the default considering this is often used as a default in mainstream practice. Based on this benchmark we recommend discontinuing use of one-hot encoding outside of special use cases (like e.g. for purposes of feature importance analysis).

B.3 ord3

- ordinal encoding with integers sorted by category frequency 'ord3'
- Sorting ordinal integers by category frequency instead of alphabetic significantly benefited model performance, in most cases lifting ordinal above one hot encoding although still not in the range of the default binarization.

B.4 ordl

- ordinal encoding with integers sorted alphabetically by category 'ordl'
- Alphabetic sorted ordinal encodings (as is the default for Scikit-Learn's OrdinalEncoder) did not perform as well, we recommend defaulting to frequency sorted integers when applying ordinal.

B.5 hsh2

- hashed ordinal encoding (library default for high cardinality categoric 'hsh2')
- This benchmark was primarily included for reference, it was expected that as some categories may be consolidated there would be a performance impact for low cardinality sets. The benefit of hashing is for high cardinality which may otherwise impact gradient boosting memory overhead.

B.6 or19

- multi-tier string parsing 'or19' Authors (2020)
- It appears that our recent invention of multi-tier string parsing succeeded in outperforming one-hot encoding and was the second top performer, but did not perform sufficiently to recommended defaulting in comparison to vanilla binarization. We recommend reserving string parsing for cases where the application may have some extended structure associated with grammatical content, as was validated as outperforming binarization for an example in the citation.

C Data Sets

The Benchmarking included the following tabular data sets, shown here with their OpenML ID number. A thank you to Vanschoren et al. (2014) for providing the data sets and Kadra et al. (2021) for inspiring the composition.

- Click prediction / 233146
- C.C.FraudD. / 233143
- sylvine / 233135
- jasmine / 233134
- fabert / 233133
- APSFailure / 233130
- MiniBooNE / 233126
- volkert / 233124
- jannis / 233123
- numerai28.6 / 233120
- Jungle-Chess-2pcs / 233119
- segment / 233117
- car / 233116
- Australian / 233115
- higgs / 233114
- shuttle / 233113
- connect-4 / 233112
- bank-marketing / 233110
- blood-transfusion / 233109
- nomao / 233107
- ldpa / 233106
- skin-segmentation / 233104
- phoneme / 233103
- walking-activity / 233102
- adult / 233099
- kc1 / 233096
- vehicle / 233094
- credit-g / 233088
- mfeat-factors / 233093
- arrhythmia / 233092
- kr-vs-kp / 233091

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [No] (see supplemental material notebooks)
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]