Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features

Florian Pargent*[,a], Florian Pfisterer[b], Janek Thomas[c], Bernd Bischl[b]

[a] *Department of Psychology, Psychological Methods and Assessment, LMU Munich, Leopoldstraße 13, 80802 Munich, Germany*
[b] *Department of Statistics, Statistical Learning and Data Science, LMU Munich, Ludwigstraße 33, 80539 Munich, Germany*
[c] *Fraunhofer Institute for Integrated Circuits, Hansastraße 32, 8068 Munich, Germany*

Abstract

Because most machine learning (ML) algorithms are designed for numerical inputs, efficiently encoding categorical variables is a crucial aspect during data analysis. An often encountered problem are high cardinality features, i.e. unordered categorical predictor variables with a high number of levels. We study techniques that yield numeric representations of categorical variables which can then be used in subsequent ML applications. We focus on the impact of those techniques on a subsequent algorithm's predictive performance, and – if possible – derive best practices on when to use which technique. We conducted a large-scale benchmark experiment, where we compared different encoding strategies together with five ML algorithms (lasso, random forest, gradient boosting, k-nearest neighbours, support vector machine) using datasets from regression, binary- and multiclass- classification settings. Throughout our study, regularized versions of target encoding (i.e. using target predictions based on the feature levels in the training set as a new numerical feature) consistently provided the best results. Traditional encodings that make unreasonable assumptions to map levels to integers (e.g. integer encoding) or to reduce the number of levels (possibly based on target information, e.g. leaf encoding) before creating binary indicator variables (one-hot or dummy encoding) were not as effective.

Key words: supervised machine learning, benchmark, high-cardinality categorical features, target encoding, impact encoding, dummy encoding

1. Introduction

While increasing sample size is usually considered the most important step to improve the predictive performance of a machine learning (ML) model, using effective feature engineering comes as a close second. One remaining challenge is

*Corresponding Author
Email address: florian.pargent@psy.lmu.de (Florian Pargent)

how to handle high cardinality features – categorical predictor variables with a high number of different levels but without any natural ordering. While categorical variables with only a small number of possible levels can often be efficiently dealt with using standard techniques such as one-hot encoding, this approach becomes inefficient as the number of levels increases. Although domain knowledge can sometimes be used to reduce the number of theoretically relevant levels, finding strategies that work well on a large variety of problems is highly important for many applications as well as in automated ML (Feurer et al., 2015; Thomas et al., 2018; Thornton et al., 2013). Optimally, strategies should be model-agnostic because benchmarking encoding methods together with ML algorithms from different classes is often necessary for applications. While a variety of strategies exist, there are very few benchmarks that can be used to decide which technique is expected to yield good predictive performance. Furthermore, there has recently been increasing attention on scientific benchmark studies that compare different methods to provide a clearer picture in light of a large number of methods available to practitioners (Bommert et al., 2020; Fernández-Delgado et al., 2014), as they can provide at least partial answers to such questions. The goal of this study is to provide an overview of existing approaches for encoding categorical predictor variables and to study their effect on a model's predictive performance.

1.1. Notation

We consider the classical setting of supervised learning from an *i.i.d.* tabular dataset $\mathcal{D}$ of size N sampled from a joint distribution $\mathcal{P}(\boldsymbol{x}, y)$ of a set of features $\boldsymbol{x}$ and an associated target variable y. Here, $\boldsymbol{x}$ consists of a mix of numeric (real-valued or integer-valued) features and categorical features, the latter of which we seek to transform feature-wise to numeric features using a categorical encoding technique. Let x be a single unordered categorical feature from a feature space $\mathcal{X}$ with cardinality $card(\mathcal{X}) \leq card(\mathbb{N})$. It holds either $y \in \mathbb{R}$ (regression), $y \in \mathcal{C}$ from a finite class space $\mathcal{C} = \{c_1, ..., c_C\}$ with $C = 2$ (binary classification) or $C > 2$ (multiclass classification). We always assume to observe all C classes in our training sample, however we might only observe a subset $\mathcal{L}^{train} \subseteq \mathcal{X}$ of a feature's available L levels, $\mathcal{L}^{train} = \{l_1, ..., l_L\}$ for categorical features. We denote the observed frequency of class c in the training set with N_c and the observed frequency of a level l in the training set with N_l. As most ML algorithms require numerical features, we use categorical encoding techniques to transform each nominal feature x^{train} into numerical features $\hat{x}^{train}$ which are then used for training. If clear from the context, we use $\hat{x}_l$ as the encoded value for an observation with level l. Although datasets might contain multiple high cardinality features, we encode each feature separately but with the same strategy.

1.2. Related Work

We broadly categorize feature encoding techniques into *target-agnostic* methods and *target-based* methods (Micci-Barreca, 2001). Figure 1 contains a taxonomy

of the encoding methods we consider in this work. *Target-agnostic* methods do not rely on any information about the target variable and can therefore also be used in unsupervised settings. Simple strategies from this domain e.g. one-hot or dummy encoding are widely used – in the scientific literature (Hancock and Khoshgoftaar, 2020; Kuhn and Johnson, 2019) but also on Kaggle[1] – to embed variables for classical ML algorithms as well as (deep) neural networks. Such indicator methods map each level of a categorical variable to a set of dichotomous features encoding the presence or absence of a particular level. An obvious drawback of indicator encoding is that it adds one additional feature per level of a categorical variable. When the number of levels increases to a point where indicator encoding leads to an unreasonable number of features, levels are often simply mapped to integer values with random order (integer encoding). Alternatively, the "hashing trick" (Weinberger et al., 2009) can be used to randomly collapse feature levels into a smaller number of indicator variables (Kuhn and Johnson, 2019), or levels can be encoded by using the observed frequency of a given level in the dataset (frequency encoding).

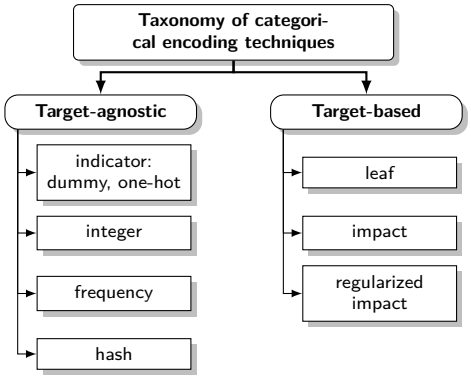

Figure 1: Taxonomy of common categorical variable encoding techniques.

Target-based methods try to incorporate information about the target values associated with a given level. Early strategies aimed to reduce the number of levels by methods like hierarchical clustering or decision trees based on statistics of the target variable, although this has been rarely described in the scientific literature (for a brief mention, see Micci-Barreca, 2001). The basic idea of more advanced methods called target, impact, mean, or likelihood encoding is to use the training set to make a simple prediction of the target for each level of the categorical feature, and to use the prediction as the numerical feature value $\hat{x}_l$ for the respective level. One of the earliest formal descriptions of this strategy is Micci-Barreca (2001). A very similar approach is called *weights of evidence* in the credit scoring literature Hand and Henley (1997). In simple target encoding for regression problems, the mean target value in the training

[1] https://www.kaggle.com

set from all observations with a certain feature level is used as the numeric value to encode that level for all observations: $\hat{x}_l = \frac{\sum_{i:x_i^{train}=l} y_i^{train}}{N_l}$. Simple target encoding often does not perform well with rare levels, where it tends to overfit to the training data and fails to generalize well for new observations. In the extreme case of a categorical feature with unique values (e.g. some hashed ID variable) studied in Prokhorenkova et al. (2018), the mean target for each level of this feature in simple target encoding is similar to the true target value of a single observation. Based on the encoded feature, all observations can be predicted perfectly in the training set, even if the original variable did not contain any useful information. ML models would place a high priority on such an encoded feature during training but would perform badly on test data. To avoid this, practitioners often use regularized target encoding by adding a smoothing parameter ϵ which shrinks those effects towards the global mean (Micci-Barreca, 2001). An alternative strategy is to combine target encoding with cross-validation (CV) techniques (Prokhorenkova et al., 2018).

1.3. Previous Encoding Benchmarks

Several small-scale studies that investigate aspects of our research questions have been previously conducted. However, those works do not yield conclusive results due to narrower or different scopes or not considering high cardinality variables. One benchmark (6 datasets) on encoding high cardinality features (cardinality between 103 and 9095) in combination with gradient boosting has been published on the Kaggle forums (Prokopev, 2018). Different versions of target encoding are compared with indicator, integer, and frequency encoding. They recommend combining the smoothed version of target encoding with 4- or 5-fold CV, never using simple target encoding and use indicator encoding only for small datasets. Interestingly, frequency encoding did perform well in many cases. Coors (2018) performed a benchmark (12 datasets) on encoding high cardinality features (maximum number of levels per dataset between 10 and 25847) when developing the automatic gradient boosting (autoxgboost) library (Thomas et al., 2018). They compared different variants of target encoding with integer and indicator encoding. In their benchmarks, target encoding only improved over target-agnostic methods on 2 datasets, while it led to worse results on 4 datasets. For a smaller number of levels, indicator and integer encoding yielded similar results. As those studies only consider gradient boosting and a limited amount of datasets, it is unclear whether results generalize to other datasets and ML algorithms. Several other publications studied encoding text data based on similarity (Cerda et al., 2018; Cerda and Varoquaux, 2020) and employed indicator or target encoding as baselines. Another line of work studies variable encodings employed within specific ML models. Wright and König (2019) study treatments for categorical variables in random forests together with dummy and integer encoding (18 datasets, cardinality between 3 and 38), concluding that indicator and integer encoding perform subpar in comparison to methods that re-order levels according to the target variable. Prokhorenkova et al. (2018) compared their new CatBoost variant of target

encoding to smoothed target encoding with no CV, hold-out, and leave-one-out CV on 8 datasets. While the CatBoost method performed best, hold-out came second and target encoding without CV performed worst. An overview of encoding techniques tailored to be used together with Neural Networks is provided in Hancock and Khoshgoftaar (2020). Amongst others, they survey indicator-based methods as well as embedding-based methods but do not conduct a benchmark study. One widely adopted approach, in particular, has been proposed by Guo and Berkhahn (2016) under the term *entity embeddings*. In contrast, our work focuses on high cardinality variables and techniques that are agnostic to the subsequent ML method, which has not been done before. We furthermore study this problem on a sufficiently large variety of datasets and settings to allow for more compelling conclusions.

The main goal of our study is to assess the impact different categorical encoding techniques have on subsequent models' predictive performance. As the optimal encoding might differ depending on the ML algorithm, we consider various state-of-the-art algorithms, *regularized linear models*, *random forests*, *gradient tree boosting*, *k-nearest neighbors*, and *support vector machines*. To find default settings for high cardinality features, we analyze a variety of datasets with different characteristics including regression, binary classification and multiclass classification problems. As our methods vary in runtime and complexity, we aim to learn whether more complex methods, in general, are to be preferred, or if simpler approaches suffice. Furthermore, the relationship between a feature's cardinality and the choice of encoding technique has not been investigated, which we study by varying the minimum number of levels above which features are transformed.

1.4. Contributions

We first survey various categorical encoding techniques and conduct a comprehensive benchmark study of a broad set of categorical encoding techniques with a focus on high cardinality features. We study 7 different encoding techniques in conjunction with 5 commonly used ML algorithms across 24 datasets, both from a classification and a regression regime. We give a detailed description of our study design to highlight important considerations for studying high cardinality features. Our results provide an overview of the performance of various approaches heavily used in the literature. After a discussion of results concerning predictive performance, we provide further analyses seeking to inform practitioners which methods to apply. This includes an important discussion on runtimes.

2. Encodings

Pseudocode is presented in the main text (for non-standard encodings), or the Supplementary Material (for standard methods). An important detail is how encoding techniques treat new levels during the prediction phase, as this is often not obvious.

2.1. Integer Encoding

The simplest strategy for categorical features is integer encoding, which is also often referred to as ordinal encoding. To make sure that the order of the observed levels does not affect our results, we randomly map the observed levels from the training set to the integers 1 to L. Although new levels could be mapped to $L + 1$ or 0, model predictions would be completely arbitrary as the order of the integers does not contain any information. Thus, we encode new levels as missing values and use mode imputation to obtain the integer which matches the most frequent level in the training set. Integer encoding should be an acceptable strategy for tree-based models, which can separate all original levels with repeated splits, but less so for other algorithms.

2.2. Frequency Encoding

Frequency encoding simply maps each level to its observed frequency in the training set ($\hat{x}_l = N_l$). On the one hand, this assumes a functional relationship between the frequency of a level and the target. On the other hand, it implicitly reduces the number of levels, as the subsequent model can best differentiate between levels with dissimilar frequencies. This approach is typically used e.g. in natural language processing to encode token or n-gram counts. We encode new levels with a frequency of 1, although choosing the minimal observed frequency for any level during training might be another reasonable strategy.

2.3. Indicator Encoding

We use indicator encoding as an umbrella term for the two most common strategies to encode categorical features with a small to moderate number of levels: one-hot and dummy encoding. One-hot encoding transforms the original feature into L binary indicator columns, each representing one original level. An observation is coded with 1 for the indicator column representing its level ($x_i^{train} = l$) and 0 for all other indicators. Dummy encoding results in only $L - 1$ indicator columns. A reference feature level is chosen that is encoded with 0 in all indicator columns. For one-hot encoding, the zero vector can be used to encode new levels which were not observed during training. For dummy encoding, this is not useful as it collapses new levels to the (often arbitrary) reference category. In our study, the first level in alphabetical order is used as the reference category for dummy encoding. We replace new levels in the prediction phase with the most frequent level in the training set. In datasets with a high total number of feature levels, constructing all indicator variables can be detrimental to the predictive performance or at least increase the computational load. As indicator encoding is practically infeasible for high cardinality variables (a feature with 10^5 levels would add 10^5 new columns), we limit the number of indicator coded features by collapsing rare levels beyond a varying threshold to a single *other* category before encoding. Closely related to dummy encoding in classical analysis of variance and regression are Helmert and Polynomial contrasts (Chambers and Hastie, 1992). However, these encodings focus on ordinal categorical features, which we did not consider in our study.

2.4. Hash Encoding

Hash Encoding can be used to compute indicator variables based on a hash function (Weinberger et al., 2009). The basic idea is to transform each feature level l into an integer $hash(l) \in \mathbb{N}$, based on its label. This integer is then transformed into an indicator representation, with 1 in indicator column number ($hash(l)$ mod $hash.size) + 1$ and 0 in all remaining columns (Kuhn and Johnson, 2019). Some levels will be hashed to the same indicator representation. The smaller the $hash.size$, the higher the number of collapsed levels. The number of indicators is often effectively lower than $hash.size$, as some indicators can be constant in the training set (we remove those columns for both training and prediction). Although hashing would allow us to jointly hash multiple features, we hash each feature separately to improve comparability with the other encoders.

2.5. Leaf Encoding

In leaf encoding, a decision tree is fitted on the training set to predict the target based on the categorical feature. Each level is encoded by the number of the terminal node, in which an observation with the respective level ends up. In that way, leaf encoding combines feature levels with similar target values. We use the rpart package in R (Therneau and Atkinson, 2018) which grows CARTs with categorical feature support that can be pruned based on internal performance estimates from 10-fold CV. Thus, our leaf encoder automatically uses an "optimal" number of new levels. To speed up the computation for multiclass classification, our implementation uses the ordering approach presented in Wright and König (2019). New levels are encoded with the arbitrary number of the terminal node with most observations during training. Encoded values are treated as a new categorical feature and encoded by one-hot encoding. Our leaf encoder can be thought of as a simplification of the approach suggested by Grąbczewski and Jankowski (2003).

Algorithm 1 Leaf Encoding

Training: require number of cross-validation folds $K \in \mathbb{N}$
fit CART tree on $\mathcal{D}^{train}$ with complexity pruning based on K-fold cross-validation
for all $x_i^{train} \in \boldsymbol{x}^{train}$ **do** $\tilde{x}_i = t$ with x_i^{train} in terminal node t
Prediction:
for x^{new} **do**
  if $x^{new} \in \mathcal{L}^{train}$ **then** $\tilde{x}^{new} = t$ with x^{new} in terminal node t
  else $\tilde{x}^{new} = b$ where b indicates the biggest terminal node

2.6. Impact Encoding

The first formal description of a so-called impact or target encoder was provided by Micci-Barreca (2001). The basic idea is to encode each feature level with the conditional target mean (regression) or the conditional relative frequency of one

or more target classes (classification). The impact encoder for classification uses a logit link and transforms the original feature into C numeric features, each representing one target class. A smoothing parameter ϵ is introduced to avoid division by zero. This parameter could be further used to regularize towards the unconditional mean. To allow comparing regularized (introduced below) with simple target encoding, we always choose a small $\epsilon = 0.0001$ for the impact encoder. Similar techniques have been proposed in the literature: Weights of evidence encoding from the credit scoring classification literature (Hand and Henley, 1997) is almost identical to impact encoding, but does not consider any regularization or centering. Impact encoding is also referred to as a target or James-Stein encoding.

2.7. GLMM Encoding

Smoothed target encoding (Micci-Barreca, 2001) can be interpreted as a simple (generalized) linear mixed model (glmm) in which the target is predicted by a random intercept for each feature level in addition to a fixed global intercept. This connection is described in Kuhn and Johnson (2019). To achieve regularized impact encoding, we constructed our own glmm encoders for regression, binary, and multiclass classification which are described in Algorithm 2 (regression) and in the Supplementary Material (classification). The encoded value for each level is based on the spherical conditional mode estimates. In regression, the conditional modes are similar to the mean target value for each level, weighted by the relative observed frequency of that level in the training set (Gelman and Hill, 2006). For technical details, see Bates (2018). Additionally, the estimate of the fixed intercept can be used during the prediction phase to encode new feature levels that were not observed in the training set. In multiclass classification, we fit C one vs. rest glmms resulting in one encoded feature per class. An important advantage of using a glmm over impact encoding with a smoothing parameter is that a reasonable amount of regularization is determined automatically and tuning the complete ML pipeline is not necessary. A further strategy to avoid overfitting in target encoding is to combine it with CV to train the encoder on independent observations without limiting the data to train the ML model. To our knowledge, we provide the first implementation which combines target encoding based on glmms with CV. During the training phase, we partition the data using CV into $n.folds$ and fit a glmm on each resulting training set. For each observation, there is exactly one glmm that did not use that observation for model fitting and can be safely used for encoding. Note that the $n.folds$ CV models (for $n.folds > 1$) are only used during the training phase. In the prediction phase, feature values are always encoded by a single glmm fitted to the complete training set. We study this method in three different settings: without CV (noCV), with 5− (5CV) and with 10− fold CV (10CV). In our study, we use the lmer (regression) and glmer (classification) functions from the lme4 package in R (Bates et al., 2015) as an efficient way to fit glmms.

Algorithm 2 GLMM Encoding Regression

Training: require $n.folds \in \mathbb{N}$
fit simple random intercept model: $y_i^{train} = \beta_{0l} + \epsilon_i = \gamma_{oo} + u_l + \epsilon_i$ on $\mathcal{D}^{train}$
with $u_l \overset{iid}{\sim} N(0, \tau^2)$, $\epsilon_i \overset{iid}{\sim} N(0, \sigma^2)$ and $x_i^{train} = l$, $l \in \mathcal{L}^{train}$
if $n.folds = 1$ **then**
  for all $x_i^{train} \in \boldsymbol{x}^{train}$ **do**
    $\hat{x}_i^{train} = \hat{\beta}_{0l}^{\mathcal{D}^{train}}$ with $x_i^{train} = l$
else use $n.folds$ cross-validation scheme to make training sets $\mathcal{D}_1^{train}, \ldots, \mathcal{D}_{n.folds}^{train}$
  and fit simple random intercept model on each $\mathcal{D}_m^{train}$
  for all $x_i^{train} \in \boldsymbol{x}^{train}$ **do**
    $\hat{x}_i^{train} = \hat{\beta}_{0l}^{\mathcal{D}^{train}}$ with $x_i^{train} = l$ based on the model m
    with $(x_i^{train}, y_i^{train}) \notin \mathcal{D}_m^{train}$
Prediction:
for x^{new} **do**
  if $x^{new} \in \mathcal{L}^{train}$ **then**
    $\hat{x}^{new} = \hat{\beta}_{0l}^{\mathcal{D}^{train}}$ with $x^{new} = l$ based on full model fitted on $\mathcal{D}^{train}$
  else $\hat{x}^{new} = \hat{\gamma}_{00}$ based on full model fitted on $\mathcal{D}^{train}$

2.8. Control Conditions

We include three control conditions to better understand the effectiveness of the investigated encoders: The performance of a featureless learner (**FL** condition) was estimated as a conservative baseline for each dataset. In regression problems, FL predicts the mean of the target variable in the training set for each observation in the test set. In classification problems, the most frequent class of the target within the training set is predicted. For each dataset, we also consider a random forest model without encoding (**none** condition), to compare the use of encoding methods in a random forest with a natural categorical splitting approach. We use the random forest in ranger (Wright and Ziegler, 2017) which provides efficient categorical feature support by ordering levels once before starting the tree growing algorithm (Wright and König, 2019). In the **remove** high cardinality features control condition, we remove features with a high number of levels above some threshold and use one-hot encoding (without collapsing rare levels) for the remaining features. This condition reflects on whether including high cardinality features does indeed improve predictive performance. Otherwise, the best encoding might just provide the least impairment compared to not including any high cardinality features. We include an overview of available implementations in widely used machine learning frameworks for R and python in the Supplementary Material.

3. Benchmark Setup

3.1. Datasets

Table 1 summarizes the datasets used in our benchmark study. We specifically investigate datasets that contain categorical variables with a large number of levels, including many well-known datasets from previous studies (Cerda et al., 2018; Coors, 2018; Kuhn and Johnson, 2019; Prokhorenkova et al., 2018). All datasets can be downloaded from the `OpenML` platform (Vanschoren et al., 2013) based on the name or the displayed OmlId. The datasets include 8 regression, 10 binary classification, and 6 multiclass classification problems (between 3 and 12 classes). To assess the imbalance in categorical variables we computed the normalized entropy for each categorical variable. The maximum normalized entropy is 1, which corresponds to a uniform distribution, while a lower number indicates a larger imbalance. Sample sizes range between 736 and 1224158 observations. The total number of features ranges between 5 and 208. Datasets contain between 1 and 20 categorical features with more than 10 levels, with the maximum number of levels for one feature in a dataset ranging between 14 and 30114. Missing values are present in about half of the datasets.

Table 1: Benchmark Datasets and Dataset Characteristics

OmlId	Name	Cl	N	NA%	Num	Bin	Cat	HighCardLevels	Entropy
41211	ames-housing	0	2930	0.00	34	2	44	10,10,16,16,17,28	
41445	employee_salaries	0	9228	0.03	1	2	3	37,385,694	
41210	avocado-sales	0	18249	0.00	8	1	2	54	
41437	wine-reviews	0	129971	6.22	1	0	6	19,43,425,707,1229,16757	
41444	medical_charges	0	163065	0.00	1	0	5	51,100,306,1977,3201	
41267	particulate-matter-ukair-2017	0	394299	0.00	4	0	5	12,30,53	
41251	flight-delay-usa-dec-2017	0	457892	0.00	2	0	7	12,31,52,52,294,294	
41255	nyc-taxi-green-dec-2016	0	1224158	0.00	4	3	7	241,260	
41283	churn	2	5000	0.00	14	2	3	10,51	
981	kdd_internet_usage	2	10108	0.39	0	48	20	10,11,46,77,119,129	
4135	Amazon_employee_access	2	32769	0.00	0	0	9	67,128,177,343,343,449,2358,4243,7518	
41434	Click_prediction_small	2	39948	0.00	3	0	6	6064,19228,19803,22381,25321,30114	
1590	adult	2	48842	0.95	6	1	7	14,16,41	
1114	KDDCup09_upselling	2	50000	68.50	174	4	30	14,...,5073,5713,13990,15415,15415	
41162	kick	2	72983	6.39	14	3	15	12,16,33,37,74,134,153,863,1063	
41442	open_payments	2	73354	22.61	0	1	4	513,1460,2255,4365	
41447	road-safety-drivers-sex	2	233964	10.78	2	1	3	380,20397	
41224	porto-seguro	2	595212	2.45	27	23	8	10,12,18,104	
188	eucalyptus	5	736	3.20	14	0	5	12,14,16,27	
41446	Midwest_survey	10	2778	1.66	0	21	5	1008	
41212	hpc-job-scheduling	4	4331	0.00	5	0	2	14	
41216	video-game-sales	12	16598	0.25	6	0	2	31,578	
41440	okcupid-stem	3	50789	15.97	2	1	16	12,12,15,15,18,32,45,48,184,208,7019	
4541	Diabetes130US	3	101766	0.00	13	9	25	10,10,18,73,717,749,790	

Note:
OmlId = Id on OpenML, Name = name on OpenML, Cl = classes (0: regression), N = observations, NA% = percentage of missing values, Num = numeric features, Bin = binary features, Cat = categorical features, HighCardLevels = number of levels for each categorical feature with at least 10 levels (some levels are not displayed for KDDCup09_upselling), Entropy = box-plot of normalized Shannon-entropy across levels (smaller = larger imbalance).

3.2. High Cardinality Threshold

It is widely assumed that more involved encoding methods are only advantageous for variables with a high number of levels, while simple indicator encoding is more appropriate for a small number of levels. To make our benchmark results more realistic, a high cardinality threshold (HCT) parameter with values of 10, 25, and 125 was introduced which determined varying configurations for the different encoders. For indicator encoding, the $HCT-1$ most frequent levels are encoded together with a single collapsed category for the remaining levels. For integer, frequency, hash, leaf, impact, and glmm encoders, only features with more than HCT levels in the training set were encoded with the respective strategy, while the remaining categorical features were one-hot encoded. HCT is used as the hash size in hash encoding. In the remove control condition, features with more than HCT levels in the training set are removed from the feature set. Based on the number of categorical features and levels per feature, some HCT settings were removed from the benchmark for some combinations of dataset $\times$ encoder. This made sure that encoders always affect at least one feature and that the remove condition always removes at least one feature. If several HCT settings of an encoder would lead to identical encoding strategies for all features of a dataset, we only kept the condition with the smallest HCT value.

3.3. Machine Learning Pipeline and Algorithms

Because we investigate strategies that can be used together with different models, we separately report results for different ML algorithms. We kept tuning the hyperparameters of the subsequent ML methods to a minimum because we were interested in the effect of the encoding techniques instead of a comparison of the ML models. In total, we investigate 5 ML methods: Regularized linear models (LASSO) were fitted with `glmnet` (Friedman et al., 2010), internally tuning the regularization using 5-fold CV. Random forests (RF) were trained using `ranger` (Wright and Ziegler, 2017). Because the random forest algorithm can be expected to give reasonable results without tuning (Probst et al., 2019), we only fixed the number of fitted trees to 500. Gradient boosting models (GB) were trained using `xgboost` (Chen et al., 2018). We set the learning rate to 0.01 and determine the number of iterations using early stopping on a 20% holdout set. K-nearest neighbours (KNN) was taken from package `kknn` (Schliep and Hechenbichler, 2016) standardizing all features and using a constant $k = 15$ for the number of nearest neighbours, together with an information gain filter (Brown et al., 2012) to limit the number of features to 25. Support vector machines (SVM) with radial basis function kernel were trained with `liquidSVM` (Steinwart and Thomann, 2017). The bandwidth and regularization parameters were internally tuned using 5-fold CV, and we used a one-vs-all approach for multiclass-settings.

The ML pipeline outlined below was used for all experimental conditions. It is not trivial and was carefully designed to ensure consistent results for extreme conditions (e.g. some levels only existing in the training data).

- **Imputation I**: Create a new factor level for missing values in categorical features with more than two categories. Impute missing values in binary features using the mode and missing values in numerical features using the mean feature value in the training data.
- **Encoding**: Transform the complete categorical features by the respective encoder. In the *no encoding* condition, the encoder simply passes on its input. The leaf and remove conditions still return categorical features, while the remaining encoders return only numerical features. Encoders only affect categorical variables above the specified HCT value.
- **Imputation II**: To handle new levels observed during prediction, impute missing values obtained during encoding.
- **Drop constants**: Drop features that are constant during training. As none of the original datasets includes constant columns, this step only removes constant features that are produced by the encoders or the CV splitting procedure.
- **Final one-hot encoding**: Transform all remaining categorical features via one-hot encoding (skipped for no encoding condition).
- **Learner**: Use the transformed data from each training set to fit the respective ML algorithm. In the prediction phase, transformed feature values (based on the trained encoder) for new observations in each test set are fed into the trained model to compute predictions.

3.4. Performance Evaluation

Throughout our analysis, we use implementations of the various methods for the open-source statistical software R (R Core Team, 2018). To enable a fair and reliable comparison in our study, we implemented all encoding methods on top of the `mlrCPO` package (Binder, 2018). The pre-processing, as well as the final ML algorithm described in 3.3, were trained and resampled using the `mlr` framework (Bischl et al., 2016) together with the `batchtools` package (Lang et al., 2017) to scale the benchmark analysis to HPC compute infrastructure. All materials for this study, including reproducible code for this manuscript and the necessary result objects can be downloaded from our **online repository** [2].
Throughout our experiments, we use 5-fold CV to obtain estimates of the predictive performance. Depending on the target variable, we report root mean squared error (RMSE) for regression, area under the curve (AUC) for binary classification or its extension AUNU (Ferri et al., 2009) for multiclass problems:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2},\ AUC = \frac{U}{N_{pos} \cdot N_{neg}},\ AUNU = \frac{\sum_{c=1}^{C} AUC_c}{C}$$

N: number of observations, N_{pos} and N_{neg}: number of true labels in both classes, U: test statistic of the Mann-Whitney-U test, and AUC_c: one vs. rest AUC of class c.

[2] https://github.com/compstat-lmu/paper_2021_categorical_feature_encodings

Throughout the benchmark study, each encoding method listed in Section 2 is applied together with all ML algorithms (c.f. Section 3.3) across high cardinality thresholds (HCT) 10, 25, 125. While we only report results for the best HCT setting for each ML algorithm × dataset combination in Section 4, we study the effect of the HCT parameter in more detail in section 4.5.

4. Benchmark Results

The overarching question we aim to answer in our study is which encoding methods generally work well across various datasets. We report results for regression and classification datasets separately, as the associated metrics differ in scale.
For 6 datasets, some conditions with the SVM led to unexpected crashes due to memory problems or numerical errors. We completely removed those datasets for the SVM when computing ranks or other statistics to compare encodings across datasets.

4.1. Encoder Performance on Each Dataset

Mean performance estimates along with minimum and maximum performance from the 5-CV folds are reported for all datasets in Figures 2 (regression), 3 (binary classification), and 4 (multiclass classification). To reduce the complexity induced by the hyperparameter HCT we only display the parameter condition with the best performance for each combination of dataset × encoding × ML algorithm. The y-axis differs for all datasets and is reversed for the RMSE for better visual comparison. For some datasets, the remove condition performed very similar to the other encodings (e.g. *ames-housing*, *porto-seguro*), suggesting that categorical features were not informative. Performance in some folds was below the FL learner for *flight-delay-usa-dec-2017* ($RMSE_{FL} = 48.8052624$) and *nyc-taxi-green-dec-2016* ($RMSE_{FL} = 2.2152442$).
On datasets where substantive performance differences could be observed, target encoding with the glmm encoder was generally **most** effective. The **worst** encoder depended on the ML algorithm and respective dataset. For datasets *Click_prediction_small*, *KDDCup09_upselling*, *kick*, and *okcupid-stem* some combinations of encoder × ML algorithm performed worse than simply removing high cardinality features.

4.2. Meta Rankings and Dataset Clustering

To further summarize the results, we computed meta rankings for each ML algorithm: First, we defined an encoder relation within each dataset, based on corrected resample t-tests (Nadeau and Bengio, 2003). An encoding was defined to beat another encoding if the one-sided p-value of the t-test was $< .05$. This allowed us to compute a weak-order consensus ranking R defined by the optimization problem:

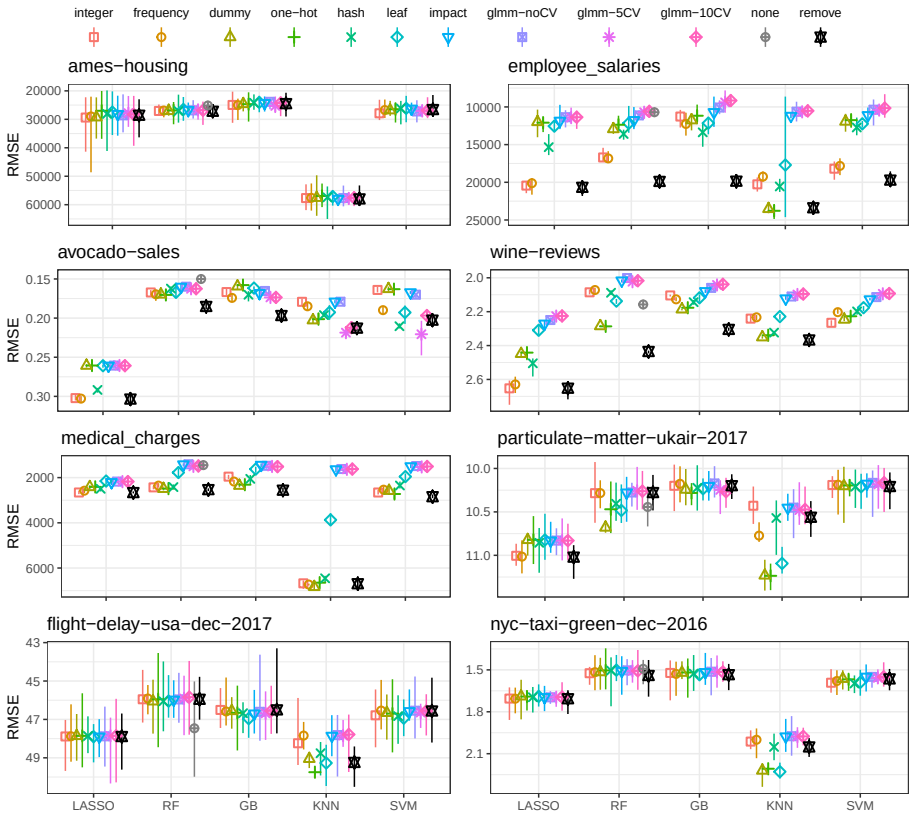

Figure 2: Performance estimates from 5-CV for regression (mean, min, max). For each combination, only the best HCT condition is displayed. Note the reversed y-axis to ease visual interpretation.

$$\underset{R \in \mathcal{C}}{arg\ min} \sum_{b=1}^{B} d(R_b, R)$$

where d is the symmetric difference distance and R_b is the relation for dataset b (Hornik and Meyer, 2007; Meyer and Hornik, 2018). The symmetric difference between two relations is the number of cases one encoding beats another encoding in one relation but not in the other one. This procedure resulted in the meta rankings based on the consensus ranking for each ML algorithm detailed in Figure 5.

Although the presented solutions of the optimization problem are not unique, rankings were highly stable for the high and low ranks. Meta rankings seem to be highly consistent with the individual patterns of encoder performances reflected in Figures 2 to 4. Looking at meta-rankings, approaches based on GLMM's

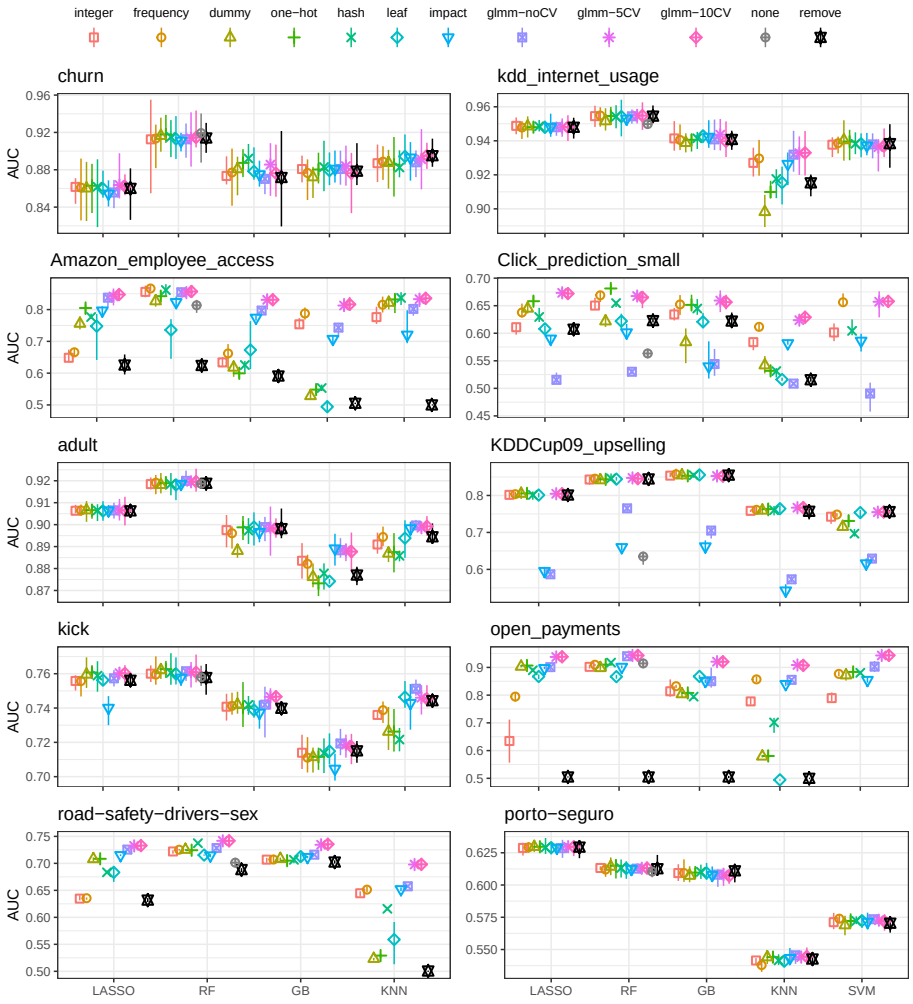

Figure 3: Performance estimates from 5-CV for binary classification (mean, min and max). For each combination, only the best HCT condition is displayed.

in combination with cross-validation outperform all other approaches across all algorithms. A further interesting detail omited in Figure 5 for clarity is that the *none* encoding strategy for the random forest was beaten by all strategies except for the *remove* condition. This implies, that even if the algorithm includes a mechanic for treating categorical variables, it might often be optimal to use a different strategy instead.

In an additional exploratory analysis, we tried to find clusters of datasets, which show systematic patterns of encoder performance (independent of the employed

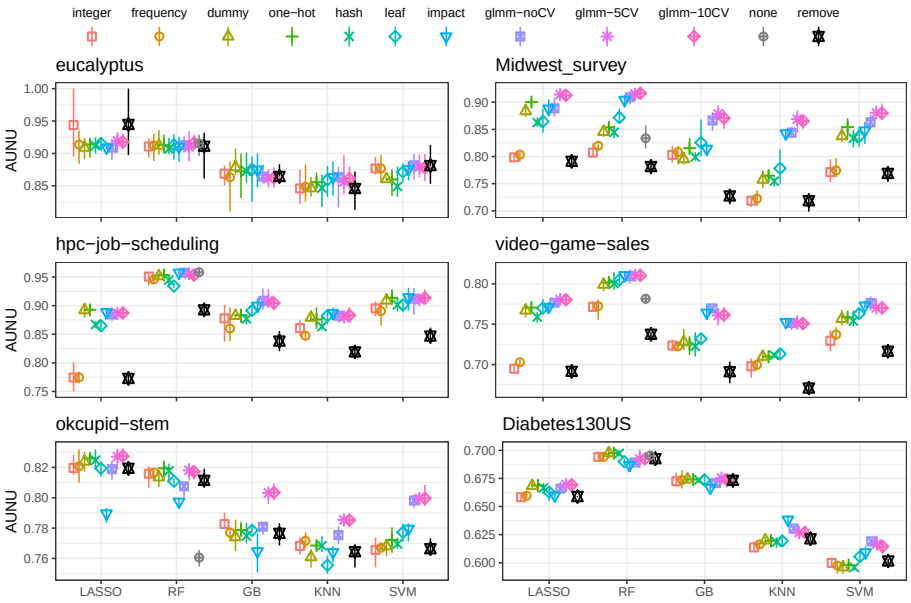

Figure 4: Performance estimates from 5-CV for multiclass classification (mean, min and max). For each combination, only the best HCT condition is displayed.

ML algorithm). We computed a partial-order consensus relation across ML algorithms for each dataset and hierarchically clustered the dataset consensus relations using the symmetric difference distance in combination with the complete linkage agglomeration method. The resulting dendrogram is displayed in Figure 6. Although the cluster structure is somewhat ambiguous, roughly three clusters can be described: The first 9 datasets from the top of the dendrogram are characterized by a medium to a high number of levels, low performance of the remove condition (indicating the importance of high cardinality features) and clear performance advantage of target encoders. For the next 13 datasets which contain the smallest number of levels, traditional encodings can compete with target encoding. This biggest cluster also includes 9 datasets with zero distances, in which no significant performance differences could be observed between any encoding conditions (nor with the remove condition, indicating that high cardinality features are not very informative). The last two datasets with the highest number of levels formed a separate cluster, in which target encoding without strong regularization (impact, glmm-noCV) showed severe overfitting. Note that clusters were not determined by problem type, again suggesting that encoder rankings are somewhat similar for regression and classification settings.

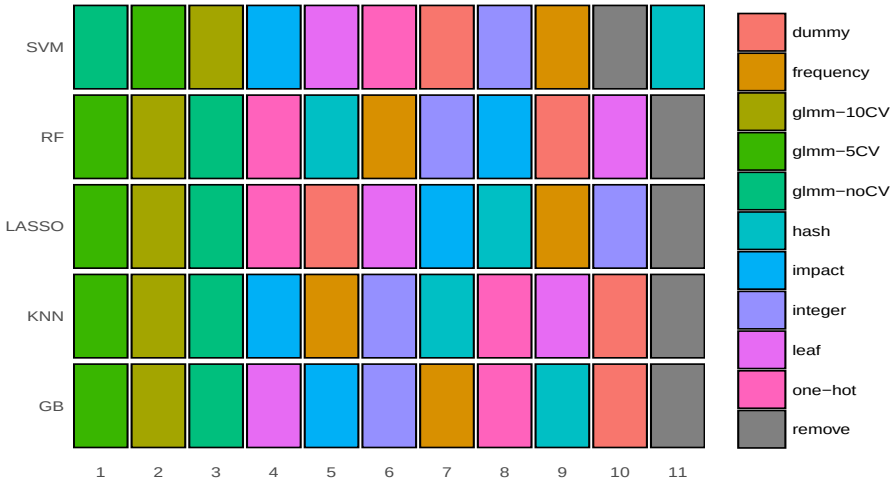

Figure 5: Consensus rankings across all datasets for each algorithm. Lower ranks indicate better performance. The rank of the *none* control condition of the random forest (rank 11) was omitted from the figure.

4.3. Summary of Encoder Performance

Our results indicate that regularized target encoding was superior or at least competitive on all datasets. Especially effective was glmm encoding with 5-fold-CV, which ranked first place for all ML algorithms except SVM. While the impact encoding used no regularization, we compared different amounts of regularization for the glmm encoder. When performance differences between target encoders were observed, more regularization seemed beneficial. In those datasets, impact encoding not only performed worse than glmm with CV but was sometimes also inferior compared to other encoders. We could not observe a setting in which regularized target encoding was convincingly beaten by target agnostic methods. When looking for a simple default method, indicator encoding in combination with collapsing small levels seems a robust alternative, although the glmm strategies should be preferred.

We observed several interesting patterns, which we aim to summarize: Integer encoding did not perform well together with XGBoost in our benchmark in contrast to what is stated in the documentation of other boosting libraries[3]. In contrast, target-based encoders (especially the glmm encoder) seem to be a compelling strategy. For LASSO, previous studies have suggested that indicator encoding works well, even with a very high number of levels (Cerda et al., 2018). Although the glmm encoders ranked first in our benchmark for the LASSO, it

[3]https://lightgbm.readthedocs.io/en/latest/Advanced-Topics.html#categorical-feature-support

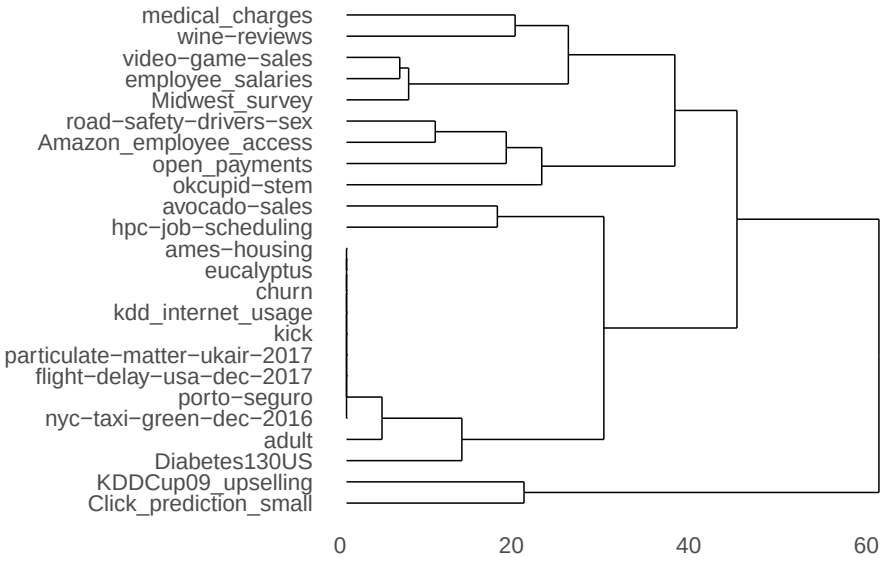

Figure 6: Hierarchical cluster analysis of benchmark datasets. The symmetric difference distance between two datasets reflects differences in performance patterns between encodings (independent of the employed ML algorithm).

was the only algorithm where the indicator encoders achieved the next best ranking. Note that for computational reasons we limited the maximum amount of indicator variables to 125 in our experimental design. The HCT = 125 setting performed best for LASSO with indicator encoding in a large number of datasets, indicating that performance might have further improved with higher values. Both KNN and SVM rely on numerical distances in feature space and tend to perform poorly in the presence of high dimensionality. Thus, we expected that target encoding should work well here as it transforms categories into a single, smooth numerical feature. This was backed by our benchmark results. For some datasets (*medical_charges*, *road-safety-drivers-sex*, and *Midwest_survey*) KNN (without tuning of the optimal number of nearest neighbours) could compete with the more sophisticated ML algorithms when combined with target encoding, but performed poorly with other encoders. Although consistent with the big picture, SVM results have to be considered with care, as some experimental conditions resulted in unexpected computational errors. In RF, a widely used strategy to deal with categorical features is to order levels by average target statistics for a given level. For a small number of levels, this approach has been reported superior to indicator and integer encoding (Wright and König, 2019), while we observed poor performance for datasets with a larger number of levels. In line with earlier research (Micci-Barreca, 2001; Prokhorenkova et al., 2018),

Table 2: Proportional Increase in Runtime Compared to One-Hot Encoding

Encoding	LASSO	RF	GB	KNN	SVM
integer	$0.4_{0}^{1.4}$	$0.59_{0}^{1.1}$	$0.67_{0}^{0.9}$	$0.85_{0.4}^{1.5}$	$1.01_{0.6}^{1.6}$
frequency	$0.34_{0}^{2.1}$	$0.54_{0}^{1.7}$	$0.54_{0}^{1.5}$	$0.79_{0.2}^{1.1}$	$1.11_{0.5}^{1.7}$
dummy	$1.13_{0.8}^{2.8}$	$0.91_{0.5}^{1.7}$	$0.97_{0.4}^{5.7}$	$1.05_{0.6}^{1.6}$	$1.02_{0.7}^{1.5}$
one-hot	1_{1}^{1}	1_{1}^{1}	1_{1}^{1}	1_{1}^{1}	1_{1}^{1}
hash	$0.87_{0.2}^{2.6}$	$1.03_{0.4}^{3.3}$	$0.98_{0.4}^{7.3}$	$0.93_{0.5}^{2.2}$	$1.16_{0.5}^{2}$
leaf	$0.46_{0}^{1.7}$	$0.67_{0}^{2.3}$	$0.85_{0}^{9.1}$	$0.97_{0.5}^{2.2}$	$1.01_{0.6}^{1.9}$
impact	$0.5_{0.1}^{1.9}$	$0.6_{0.1}^{1.5}$	$0.82_{0}^{120.5}$	$1.11_{0.3}^{24}$	$1.06_{0.7}^{1.8}$
glmm-noCV	$0.58_{0}^{2.8}$	$0.57_{0.1}^{3.8}$	1.65_{0}^{12}	$1.09_{0.3}^{10.3}$	$1.1_{0.7}^{2.1}$
glmm-5CV	$0.83_{0.1}^{2.2}$	$0.66_{0.1}^{15.3}$	$7.21_{0}^{69.3}$	$2.77_{0.7}^{50.1}$	$1.41_{0.6}^{4.6}$
glmm-10CV	$1.07_{0.1}^{4.2}$	$0.96_{0.1}^{28.4}$	$12.71_{0}^{127.6}$	$5.37_{0.8}^{97.9}$	$1.55_{0.5}^{4.6}$
none		$0.2_{0}^{1.4}$			
remove	$0.4_{0}^{1.7}$	$0.48_{0}^{1.1}$	$0.58_{0}^{1.3}$	$0.82_{0.2}^{1.2}$	$1.09_{0.6}^{2}$

Note:
Median $_{min}^{max}$ across datasets of the proportional increase in runtime from 5-CV, when comparing the respective encoder with one-hot encoding. Only the best HCT conditions are reported.

target encoding with regularization (glmm) performed better or equally well in comparison with the unregularized impact encoder. The glmm method was the overall winner based on the encoder rankings across datasets and outperformed impact encoding on some datasets. Our results show that combining target encoding based on glmms with 5-fold CV led to an improved predictive performance on many datasets. Performance often did not improve further when using 10 folds, suggesting that 5 folds might be a good default in practice. The glmm encoder has a clear advantage over target encoding with a smoothing hyperparameter (Micci-Barreca, 2001), as costly tuning the whole ML pipeline with different smoothing values is not required.

On a side note, we found that one-hot encoding usually gave a slightly better performance than dummy encoding. This phenomenon has also been observed by Chiquet et al. (2016). Tutz and Gertheiss (2016) commented that in generalized linear models "the naive combination of usual dummy coding with reference category and standard penalties is often a bad idea" (p.254) unless the reference category is of special interest. In our benchmark, the reference category was chosen arbitrarily as is usually the case in ML. Our results suggest that one-hot encoding is the better standard compared to dummy encoding when applying nonlinear regularized models like RF, GB or SVM with (high cardinal) categorical features. We provide a more in-detail analysis of the performances of one-hot and dummy encoding in the Supplementary Material.

4.4. Runtime Analysis

To determine whether traditional encodings might be worth it when facing limited computational resources, we further analyzed the runtimes of the whole analysis pipeline for different encoders and ML algorithms. Aggregated results are shown in Table 2. To enable a meaningful comparison, we report runtime as the fraction of a full pipeline's strategy compared to running simple one-hot encoding for the same pipeline and then further aggregate across datasets using the median. We again only report the **best** HCT setting. Absolute runtimes are hard to interpret and aggregate because runtime distributions across datasets are heavily skewed as proportionally larger runtimes are observed for big datasets. Furthermore, while a pipeline's runtime can be dominated by the encoder for small datasets, the training of the consecutive ML algorithm is dominating for large datasets, which might render differences during encoding irrelevant. Therefore we aim to report what is important in practice, the time differences for training the **full pipeline**. The results clearly show that regularized target encoding does not consistently yield slower runtimes compared to simple strategies like indicator encoding. Reasons for this might be, that more efficient representations yielded by target encoders lead to faster runtimes of subsequent ML algorithms. This suggests that a possible runtime vs. predictive performance trade-off might also be in favour of target encoding, especially for large datasets where a high number of indicator variables increases computational load. We did observe a substantial increase in runtimes when using regularized glmm with GB (with little tuning), where runtimes are relatively short in comparison to the time required to fit categorical encoders. In other settings (LASSO, RF), the glmm encoders have been observed to be even faster than indicator encoding.

4.5. Analysing High Cardinality Thresholds

Until now, we ignored the HCT parameter by reporting only the condition with the best performance. An interesting question was whether target encoding is only useful for features with a very high number of levels or if it might also have advantages for fewer levels, where most practitioners would routinely use indicator encoding. We tested this using HCT thresholds of 10, 25 and 125, but the results are not easy to interpret. In general, the optimal threshold seemed to strongly depend on the dataset, but we also observed some weak patterns: LASSO improved for larger HCT, indicating that its internal regularization can efficiently deal with the sparseness induced by indicator encoding. In comparison, other methods generally yielded better performance if features above the very low HCT of 10 were encoded using one of the target encoding strategies. Furthermore, different regularization strategies for the target based encoders seemed not to prefer strongly different HCT values.

5. Discussion

In our benchmark, we compared encoding strategies for high cardinality features on a variety of regression, binary and multiclass classification datasets with dif-

ferent ML algorithms. Regularized target encoding (glmm) was superior across most datasets and ML algorithms. Although the performance of other encoding strategies was comparable in some conditions, target encoding was never outperformed. In general, our results suggest that regularized target encoding works well for all kinds of algorithms and could be considered by default when working with high cardinality features. In terms of runtime, using target encoding together with 5-fold CV sometimes leads to slightly longer runtimes (in comparison to indicator encoding), but especially for larger datasets this is often offset by the more efficient representation yielded from target encoding.
What constitutes a "high" cardinality problem is a difficult question that might not only depend on the number of levels but also on other characteristics of the dataset and its features. Unfortunately, the number of datasets in our study was too small to discover consistent patterns between encoder performance and dataset characteristics. Target encoding features with as little as 10 levels seemed to be effective in a substantive number of conditions, but for other datasets, higher HCT values performed better. Thus, some form of hyperparameter tuning seems necessary to decide the level threshold for target encoding at this point, as no suitable defaults seem to be available. Note that we compared different HCT values but then used the same encoding strategy for all affected features alongside one-hot encoding for the remaining ones. Although this strategy is reasonable, it might be beneficial to decide whether to use target encoding on a feature by feature basis. ML pipelines could introduce a categorical hyperparameter for each feature that represents which encoding is used. To make this complicated meta optimization problem feasible, only a small number of encoders can be included. Our study can help to decide which encoders could be safely omitted from consideration.

5.1. Limitations

In our study, several decisions were necessary to make answering our research questions technically and computationally feasible. First, we only used minimal tuning for our ML algorithms, as we were not interested in comparing their performance against each other. This probably led to suboptimal performance for the GB, KNN, and SVM learners. When interpreting our results, we assume that the encoder rankings are comparable when more extensive tuning is used. This seems plausible, considering the high stability of our meta rankings between algorithms. We only use 5-fold-CV without repetitions to estimate predictive performance, but due to a small variance between folds, we could confidently detect performance differences of encoders for many datasets. As we are interested in model-agnostic methods that can be combined with any supervised ML algorithm, we do not investigate several strategies proposed in the recent literature (Guo and Berkhahn, 2016; Prokhorenkova et al., 2018). Additionally, we only investigate univariate encodings, i.e. we always encode each variable separately. Levels with comparable main effects can therefore not be distinguished based on the transformed feature, which prevents the consecutive ML algorithm from learning interactions for specific levels. Methods that can

jointly encode several variables and therefore leverage correlation structures between the different variables are an interesting avenue of research in the future. Another interesting avenue is using target-based encoding strategies to learn a vector-valued representation of a level, similar to entity embeddings (Guo and Berkhahn, 2016). We furthermore only investigate traditional categorical variables and do not extend our analysis to multi-categorical variables or text-strings (Cerda et al., 2018; Cerda and Varoquaux, 2020), where other encoding techniques can harvest additional information.

5.2. Conclusion

This benchmark study compared the predictive performance of a variety of strategies to encode categorical features with a high number of unordered levels for different supervised machine learning algorithms. Regularized versions of target encoding, which uses predictions of the target variable as numeric feature values, performed better than traditional strategies like integer or indicator encoding. Most effective, with a consistent superior performance across Ml algorithms and datasets, was a target encoder which combines simple generalized linear mixed models with cross-validation and does not require tuning hyperparameters.

Acknowledgements

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A and by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics – Data – Applications (ADA-Center) within the framework of „BAYERN DIGITAL II“ (20-3410-2-9-8). The authors of this work take full responsibilities for its content.

Declarations of interest

none

References

Bates, D., 2018. Computational methods for mixed models. Vignette for lme4.

Bates, D., Mächler, M., Bolker, B., Walker, S., 2015. Fitting linear mixed-effects models using lme4. Journal of Statistical Software 67, 1–48. doi:10.18637/jss.v067.i01

Binder, M., 2018. mlrCPO: Composable preprocessing operators and pipelines for machine learning.

Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., Jones, Z.M., 2016. mlr: Machine learning in r. Journal of Machine Learning Research 17, 1–5.

Bommert, A., Sun, X., Bischl, B., Rahnenführer, J., Lang, M., 2020. Benchmark for filter methods for feature selection in high-dimensional classification data. Computational Statistics & Data Analysis 143.

Brown, G., Pocock, A., Ming-Jie, Z., Luján, M., 2012. Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. Journal of Machine Learning Research 13, 27–66.

Cerda, P., Varoquaux, G., 2020. Encoding high-cardinality string categorical variables. IEEE Transactions on Knowledge and Data Engineering 1–1. doi:10.1109/TKDE.2020.2992529

Cerda, P., Varoquaux, G., Kégl, B., 2018. Similarity encoding for learning with dirty categorical variables. Machine Learning 107, 1477–1494. doi:10.1007/s10994-018-5724-2

Chambers, J., Hastie, T., 1992. Statistical models. Chapter 2 of statistical models in s. Wadsworth & Brooks/Cole.

Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., Chen, K., Mitchell, R., Cano, I., Zhou, T., Li, M., Xie, J., Lin, M., Geng, Y., Li, Y., 2018. Xgboost: Extreme gradient boosting.

Chiquet, J., Grandvalet, Y., Rigaill, G., 2016. On coding effects in regularized categorical regression. Statistical Modelling 16, 228–237. doi:10.1177/1471082X16644998

Coors, S., 2018. Automatic gradient boosting (Master's thesis). LMU Munich.

Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D., 2014. Do we need hundreds of classifiers to solve real world classification problems? Journal of Machine Learning Research 15, 3133–3181.

Ferri, C., Hernández-Orallo, J., Modroiu, R., 2009. An experimental comparison of performance measures for classification. Pattern Recognition Letters 30, 27–38. doi:https://doi.org/10.1016/j.patrec.2008.08.010

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F., 2015. Efficient and robust automated machine learning, in: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 28. Curran Associates, Inc., pp. 2962–2970.

Friedman, J., Hastie, T., Tibshirani, R., 2010. Regularization paths for generalized linear models via coordinate descent. Journal of Statistical Software 33, 1–22.

Gelman, A., Hill, J., 2006. Data analysis using regression and multilevel/hierarchical models. Cambridge university press.

Grąbczewski, K., Jankowski, N., 2003. Transformations of symbolic data for continuous data oriented models, in: Kaynak, O., Alpaydin, E., Oja, E., Xu, L. (Eds.), Artificial Neural Networks and Neural Information Processing — ICANN/ICONIP 2003. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 359–366.

Guo, C., Berkhahn, F., 2016. Entity embeddings of categorical variables. arXiv preprint arXiv:1604.06737.

Hancock, J.T., Khoshgoftaar, T.M., 2020. Survey on categorical data for neural networks. Journal of Big Data 7, 1–41.

Hand, D.J., Henley, W.E., 1997. Statistical classification methods in consumer credit scoring: A review. Journal of the Royal Statistical Society: Series A (Statistics in Society) 160, 523–541. doi:10.1111/j.1467-985X.1997.00078.x

Hornik, K., Meyer, D., 2007. Deriving consensus rankings from benchmarking experiments, in: Advances in Data Analysis. Springer, pp. 163–170. doi:10.1007/978-3-540-70981-7_19

Kuhn, M., Johnson, K., 2019. Feature engineering and selection: A practical approach for predictive models.

Lang, M., Bischl, B., Surmann, D., 2017. Batchtools: Tools for r to work on batch systems. The Journal of Open Source Software 2. doi:10.21105/joss.00135

Meyer, D., Hornik, K., 2018. Relations: Data structures and algorithms for relations.

Micci-Barreca, D., 2001. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. SIGKDD Explor. Newsl. 3, 27–32. doi:10.1145/507533.507538

Nadeau, C., Bengio, Y., 2003. Inference for the generalization error. Machine Learning 52, 239–281. doi:10.1023/A:1024068626366

Probst, P., Wright, M.N., Boulesteix, A.-L., 2019. Hyperparameters and tuning strategies for random forest. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 0, e1301. doi:10.1002/widm.1301

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V., Gulin, A., 2018. CatBoost: Unbiased boosting with categorical features, in: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 31. Curran Associates, Inc., pp. 6638–6648.

Prokopev, V., 2018. Mean (likelihood) encodings: A comprehensive study. Kaggle Forums.

R Core Team, 2018. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

Schliep, K., Hechenbichler, K., 2016. Kknn: Weighted k-nearest neighbors.

Steinwart, I., Thomann, P., 2017. liquidSVM: A fast and versatile SVM package. ArXiv e-prints 1702.06899.

Therneau, T., Atkinson, B., 2018. Rpart: Recursive partitioning and regression trees.

Thomas, J., Coors, S., Bischl, B., 2018. Automatic gradient boosting.

Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K., 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms, in: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13. ACM, New York, NY, USA, pp. 847–855. doi:10.1145/2487575.2487629

Tutz, G., Gertheiss, J., 2016. Rejoinder: Regularized regression for categorical data. Statistical Modelling 16, 249–260. doi:10.1177/1471082X16652780

Vanschoren, J., N. van Rijn, J., Bischl, B., Torgo, L., 2013. OpenML: Networked science in machine learning. SIGKDD Explorations 15, 49–60. doi:10.1145/2641190.2641198

Weinberger, K.Q., Dasgupta, A., Langford, J., Smola, A.J., Attenberg, J., 2009. Feature hashing for large scale multitask learning, in: ICML.

Wright, M.N., König, I.R., 2019. Splitting on categorical predictors in random forests. PeerJ 7. doi:10.7717/peerj.6339

Wright, M.N., Ziegler, A., 2017. ranger: A fast implementation of random forests for high dimensional data in C++ and R. Journal of Statistical Software 77, 1–17. doi:10.18637/jss.v077.i01