A Benchmark Experiment on How to Encode Categorical Features in Predictive Modeling

Florian Pargent

Master Thesis in Statistics

Ludwig-Maximilians-Universität München

Supervisors:

Bernd Bischl and Janek Thomas

March 2019

Author Note

Correspondence concerning this article should be addressed to Florian Pargent, Leopoldstr. 13, 80802 München. E-mail: Florian.Pargent@psy.lmu.de

Abstract

In predictive modeling, high cardinality features (i.e. unordered categorical predictor variables with a high number of levels) often pose problems, as most supervised machine learning algorithms only accept numerical input. We compared different encoding strategies for high cardinality features in a benchmark with five different machine learning algorithms (lasso, random forest, gradient boosting, k-nearest neighbors, support vector machines) using datasets from the regression, binary and multiclass classification settings. Regularized versions of target encoding (i.e. using target predictions based on the feature levels in the training set as a new numerical feature) performed best for all machine learning algorithms. Ordered splitting which allows the use of categorical features in random forests was not competitive, especially when facing thousands of levels. Traditional encodings which make unreasonable assumptions to map levels to integers (e.g. integer encoding) or reduce the number of levels (possibly based on target information, e.g. cluster encoding) prior to creating binary indicator variables (one-hot or dummy encoding) were not as effective. Most effective was a target encoder which combines a simple generalized linear mixed model (to shrink predictions for rare levels towards the grand mean) with cross-validation (to reduce overfitting to the training data) and does not require excessive hyperparameter tuning. A newly developed target encoder based on random forest predictions showed promising results but needs further improvements.

A Benchmark Experiment on How to Encode Categorical Features in Predictive Modeling

Introduction

Predictive Modeling with High Cardinality Features

While increasing the sample size is usually considered the most important aspect to improve the performance of a predictive model, using effective feature engineering comes in closely in second place. One remaining challenge is how to handle high cardinality features – categorical predictor variables with a high number of different levels but without any natural ordering. Unfortunately, the standard approach of using a binary indicator variable for each level becomes inefficient as the number of categories increases. Although domain knowledge can sometimes be used to reduce the number of theoretically relevant levels, finding strategies which work well on a large variety of problems is highly important for many applications as well as automatic machine learning (Feurer et al., 2015; Thomas, Coors, & Bischl, 2018; Thornton, Hutter, Hoos, & Leyton-Brown, 2013). Optimally, strategies should be model agnostic, as benchmarking machine learning algorithms from different classes is often a necessary part of applied predictive modeling. While a variety of such strategies exists, there are very few benchmarks that can be used to decide which technique is expected to yield good predictive performance under varying circumstances. The present study will try to answer some of these questions comparing common strategies and some extensions, while applying different algorithm classes on high cardinality problems with varying data characteristics.

First, we introduce the general setting alongside some notation. Second, we review a popular class of techniques called target encoding, talk about the special situation of handling categorical features in tree models, and summarize results from earlier benchmark studies. Then, we describe the investigated encoding strategies in greater detail and formulate concrete research questions. We outline the full design of our benchmark study, including datasets, machine learning algorithms, and the experimental conditions with analysis pipeline, resampling strategy, and performance measures. Finally, results will be presented and discussed.

Notation

Let x be an unordered categorical feature from a feature space $\mathcal{X}$ with cardinality $card(\mathcal{X}) \leq card(\mathbb{N})$. For supervised predictive modeling, we use the feature x to predict a target variable y. In regression tasks, $y \in \mathbb{R}$. In classification tasks, y is from a finite class space $\mathcal{C} = \{c_1, ..., c_C\}$ with $C = 2$ (binary classification) or $C > 2$ (multiclass classification). Machine learning models are trained on an *iid* sample $\mathcal{D}^{train} = \{(x_1^{train}, y_1^{train}), \ldots, (x_i^{train}, y_i^{train}), \ldots, (x_N^{train}, y_N^{train})\}$ with N observations.

For a target vector in multiclass classification, $\boldsymbol{y}^{train} = (y_1^{train}, \ldots, y_i^{train}, \ldots, y_N^{train})^T$, $y_i^{train} \in \mathcal{C} = \{c_1, ..., c_C\}$, we always assume to observe all C classes in our training sample. The observed frequency of class c in the training set is N_c. However, for the feature vector $\boldsymbol{x}^{train} = (x_1^{train}, \ldots, x_i^{train}, \ldots, x_N^{train})^T$, we might only observe a subset $\mathcal{L}^{train} \subseteq \mathcal{X}$ of the full space with L levels, $\mathcal{L}^{train} = \{l_1, ..., l_L\}$. The observed frequency of a level l in the training set is N_l. As most machine learning algorithms require numerical features, we have to use feature engineering techniques – which we will refer to as feature encodings – to transform the nominal feature values x_i^{train} into numerical feature values $\hat{x}_i^{train} \in \mathbb{R}$ which are then used to train the machine learning model. If clear within context, we use $\hat{x}_l$ as the encoded value for an observation from level l. Some encoding strategies are based on first transforming x_i^{train} into another categorical feature value $\tilde{x}_i^{train}$ from a space with less than L levels, which is then transformed into $\hat{x}_i^{train} \in \mathbb{R}$ by a simple strategy like dummy encoding. Some encoders use target information from the training set while others do not.

Prior to making a prediction for some new observation x^{new}, the trained encoder is used to compute $\hat{x}^{new}$, which is then fed to the trained machine learning model. New observations might be from an unobserved level ($x^{new} \notin \mathcal{L}^{train}$) which must also be handled by the trained encoder. All presented strategies will encode each categorical feature separately. Thus, our notation does not have to distinguish between different x variables, although datasets often contain more than one high cardinality feature.

Target Encoding

Feature encoding is frequently discussed on the influential machine learning platform Kaggle (https://www.kaggle.com/), where many challenging datasets contain high cardinality features. When the number of levels increases to a point where indicator (i.e. one-hot or dummy) encoding leads to an unreasonable number of features, levels are often simply mapped to integer values (with random order). Other common strategies aim to reduce the number of levels by methods like hierarchical clustering based on statistics of the target variable, although this has been rarely described in scientific publications (for a short mention, see Micci-Barreca, 2001). Alternatively, the "hashing trick" (Weinberger, Dasgupta, Langford, Smola, & Attenberg, 2009) can be used to randomly collapse feature levels into a smaller number of indicator variables (Kuhn & Johnson, 2019).

However, in recent Kaggle competitions with high cardinality features, many high ranked contributions employ a method called target, impact, mean, or likelihood encoding, often in combination with gradient boosting. The basic idea of target encoding is to use the training set to make a simple prediction of the target for each level of the categorical feature, and use the prediction as the numerical feature value $\hat{x}_l$ for the respective level. One of the earliest formal description of this strategy is Micci-Barreca (2001). For similar ideas from the credit scoring literature, see Hand and Henley (1997). In the most basic version for regression problems, the mean target value in the training 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.

Simple target encoding can assign extreme values for rare levels. To place lower trust on extreme target values for rare categories, Micci-Barreca (2001) proposed target encoding with smoothing, which shrinks the mean target within each level towards the mean target of all training observations based on the general formula: $\hat{x}_l = \lambda_l \cdot \frac{\sum_{i:x_i^{train}=l} y_i^{train}}{N_l} + (1-\lambda_l) \cdot \frac{\sum_{i=1}^{N} y_i^{train}}{N}$. Setting $\lambda_l = \frac{N_l}{N_l+\epsilon}$ leads to $\hat{x}_l = \frac{\sum_{i:x_i^{train}=l} y_i^{train} + \epsilon \cdot \bar{y}^{train}}{N_l+\epsilon}$ with regularization parameter $\epsilon > 0$, which is often used in practice (Prokhorenkova, Gusev, Vorobev, Dorogush, & Gulin, 2018).

The amount of shrinkage increases for small N_l and high values of ϵ. The optimal size of ϵ depends on the data and is usually determined by cross-validation. This costly approach requires fitting the encoder in addition to the subsequent machine learning algorithm multiple times. Avoiding hyperparameter tuning would be much preferable. Micci-Barreca (2001) already noticed that the more general version of target encoding can be understood as an empirical Bayes estimate for the random effects β_{0l} of a simple linear mixed model of the form $y_{il} = \beta_{0l} + \epsilon_{il} = \gamma_{oo} + u_l + \epsilon_{il}$ with $u_l \overset{iid}{\sim} N(0, \tau^2)$, $\epsilon_{il} \overset{iid}{\sim} N(0, \sigma^2)$. An efficient version of target encoding, which determines the optimal shrinkage based on the data by fitting a single linear mixed model has recently been implemented in the embed package in R (Kuhn, 2018).

A vivid example for another problem of simple target encoding is presented by Prokhorenkova et al. (2018). In the extreme case of a categorical feature with only unique values (e.g. some hashed ID variable), the mean target for each level of this feature 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. Machine learning models would place high priority on such an encoded feature during training but would perform horribly on test data. A general strategy to avoid this kind of overfitting is to combine target encoding with resampling techniques (Prokhorenkova et al., 2018). In the most simple case, the training set is divided into two parts. Target encoding based on the first part is used to transform the observations from the second part. Only the encoded second part is then used to train the machine learning model. This is not feasible with small datasets where the benefit of reducing overfitting of the encoding does not outweigh the performance reduction introduced by the smaller training set of the final model. Leave-one-out cross-validation could be used to produce an encoded value for each observation in the training data without using that observation to compute the encoding. However, leave-one-out cross-validation also has two important disadvantages. First, it tremendously increases the amount of computation especially for large samples. Second, it can be shown that there are still situations in which perfect predictions are easily possible based on the encoded feature in the training set. One example is a constant categorical feature in binary classification (for details, see Prokhorenkova et al., 2018). The solution for both problems is to use k-fold cross-validation with $k \ll N$, as it limits the computational costs and makes the perfect prediction problem less severe, as the cross-validation estimates for each fold are less correlated. Target encoding with cross-validation is implemented in the vtreat package in R (Mount & Zumel, 2019). A different solution is to perform simple target encoding (possibly without smoothing) on the whole training set and add random noise to the encoded values. This reduces overfitting by effectively limiting the predictive performance on the training set. At the same time it introduces a hyperparameter for the noise variance, which can be expected to be crucial for the effectiveness of the encoding. In contrast, it seems easier to find reasonable default values for the number of folds in the cross-validation version.

Unordered Categorical Features in Trees

Tree based methods are one of the only model classes which have a natural mechanism to deal with unordered categorical features (see Wright & König, 2019 for an introduction). One of the most widespread machine learning algorithms is the random forest (Breiman, 2001) which is an ensemble of several CARTs (Breiman, Friedman, Stone, & Olshen, 1984). In CART, subsets of the training data are consecutively split into two parts based on the observed values of a single feature. For unordered categorical features there are $2^{L-1} - 1$ possible ways to partition the L levels into two groups. To determine an optimal split point, the basic algorithm computes all possible partitions and compares them to the possible partitions for all remaining features. For high cardinality features, this strategy is computationally demanding. First, it takes a long time. Second, when the optimal split is found, the direction for all levels has to be stored. For efficiency, the bit representation of a single integer is often used where 0 and 1 identify left and right. This limits the maximum number of levels in many implementations. In contrast, only numeric split points (stored as a single floating point number) have to be considered for numeric or ordered categorical features. Solutions to avoid computing all possible partitions for regression and binary classification have been available since Breiman et al. (1984) and are implemented in R for single trees (Therneau & Atkinson, 2018) as well as random forest algorithms (Ishwaran & Kogalur, 2019; Liaw & Wiener, 2002; Wright & Ziegler, 2017). When ordering levels for each split based on the target mean (regression) or the frequency of one class (binary classification), numeric spitting on the ordered levels (only $L-1$ candidate splits) leads to the same optimal split as considering all possible combinations. This speeds up the computation but still leads to memory problems as the ordering has to be stored for each split. The ranger package for random forests in R went one step further by ordering the levels only once prior to the splitting process (Wright & König, 2019). After the initial ordering, tree growing is identical to numerical features, although optimal splits are no longer similar to the full partitioning approach. Ranger is also the first package to offer an efficient ordering strategy for multiclass classification, by implementing a method proposed by Coppersmith, Hong, and Hosking (1999). The relative frequency of all classes within each feature level are used to compute a weighted covariance matrix, which respects both the deviation of the class frequencies of each level from the mean class frequencies and the observed frequencies of all levels. Feature levels are ordered based on the first principal component of this weighted covariance matrix (Wright & König, 2019). Note that splits are also not similar to the full partitioning approach (even if the ordering would be computed for each split). Not all implementations of random forests have categorical feature support. As of February 2019, the popular python library scikit-learn (Pedregosa et al., 2011) does not handle categorical features in trees, although this has been discussed for a long time.

Another very successful model class for larger datasets is gradient boosting with shallow trees as the base learner (Friedman, 2001). Popular libraries are XGBoost (T. Chen & Guestrin, 2016), the implementation in scikit-learn (Pedregosa et al., 2011), LightGBM (Ke et al., 2017), and CatBoost (Prokhorenkova et al., 2018). XGBoost and scikit-learn can only handle numerical data, while LightGBM and CatBoost provide categorical feature support. LightGBM uses the ordering approach on each split (Fisher, 1958) to sort the histogram of categorical features based on the training objective. CatBoost takes a different approach by introducing a new version of target encoding, which is integrated into their ordered boosting algorithm. The general idea of ordered boosting is to use a permutation of the training data and maintain N models, with model M_j, $j \leq N$ being only trained on the permuted observations $i = 1, ..., j$. Each model is trained for B boosting iterations:

$M_j^1, \ldots, M_j^b, \ldots, M_j^B$. Although the final model is only model M_N^B, the others are needed in the updating process. To update model M_N^{b-1} to M_N^b, ordered boosting only allows residuals based on a model trained without the respective observation, as this reduces overfitting. Thus, in iteration b the residual for observation i is obtained from M_{i-1}^{b-1}. To ensure that such unbiased residuals are used everywhere, all models have to be updated for each boosting iteration b by using only "previous" models $M_1^{b-1}, ..., M_{j-1}^{b-1}$ from the permutation. In the presence of categorical features, ordered boosting enables an elegant version of target encoding, which is closely related to the leave-one-out version. For each observation i, the encoded feature value is the result of smoothed target encoding trained only on "previous" observations $1, ..., i-1$. Note that CatBoost uses a computationally more efficient modification of ordered boosting, which uses several permutations but does not require to store all models.

For tree based algorithms that can only deal with numerical data, unordered categorical features have to be encoded. Simple integer encoding might be an acceptable strategy for trees, which can separate all original levels with repeated splits. Integer encoding is often considered superior to indicator encoding, as stated in the documentation of LightGBM (https://lightgbm.readthedocs.io/en/latest/Advanced-Topics.html#categorical-feature-support). With indicator encoding, the feature importance of the original variable is distributed among separate binary variables. Tree growing algorithms should be biased against them, as the impurity reduction induced by a single indicator is rarely enough to be selected for splitting. Even if a tree-based algorithm supports categorical partitioning, it is unclear whether this is suitable for high cardinality features. Most tree algorithms are biased towards features with a high number of possible split points (Strobl, Boulesteix, Zeileis, & Hothorn, 2007). In their seminal textbook, Hastie, Tibshirani, and Friedman (2009) recommend not to use features with a large number of levels, especially in combination with the full partitioning approach where the number of partitions grows exponentially with the number of levels. Similarly, the documentation of LightGBM notes that integer encoding is often the best strategy for high cardinality features. Encoding techniques might be preferable for high cardinality features, which is in line with the observed success of combining target encoding with gradient boosting on Kaggle.

Previous Encoding Benchmarks

No previous study has focused on comparing encoding strategies for high cardinality features with different machine learning algorithms on a variety of datasets. For the random forest, Wright and König (2019) tested full partitioning, ordering at each split, and ordering once for each tree with dummy and integer encoding (18 datasets). Note that they did not investigate high cardinality features, as most features had between 3 and 10 levels (38 levels max). The three categorical partitioning methods performed best, with the computationally most efficient order once approach performing equally well than the two others. Integer encoding generally performed worst, and tree sizes were twice as big compared to the order once method. Even though trees can potentially create all possible partitions with repeated splits, integer encoding did not seem to be an effective strategy. Indicator encoding also performed worse than categorical partitioning in most cases (except for survival forests), although differences were smaller.

A small benchmark (6 datasets) on encoding high cardinality features (maximum number of levels per dataset between 103 and 9095) in combination with gradient boosting has been published on the Kaggle forums by Viacheslav Prokopev (https://www.kaggle.com/vprokopev/mean-likelihood-encodings-a-comprehensive-study). Different versions of target encoding are compared with indicator, integer, and frequency ($\hat{x}_l = N_l$) encoding. He recommends to combine the smoothed version of target encoding with 4 or 5-fold cross-validation, never use simple target encoding and use indicator encoding only in small datasets. Interestingly, frequency encoding did perform well in many cases.

Cerda, Varoquaux, and Kégl (2018) developed encoding methods for text data, based on the similarity between the labels of the levels themselves. In their validation study (6 datasets) with ridge regression and gradient boosting, they included indicator encoding and the smoothed version of target encoding as competing methods. All datasets contained one high cardinality feature with between 100 and 4634 levels. Depending on the dataset, smoothed target encoding performed better, worse or similar to indicator encoding. However, it seemed like target encoding works better for gradient boosting, while indicator encoding works better for ridge regression. Sometimes one of both methods could compete with the special encodings constructed for text data. They also employed hash encoding, which performed horribly in most datasets.

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. Superior performance of target encoding was only reported for 2 datasets. In 4 datasets, integer or dummy encoding was superior. If the number of levels was small enough that indicator encoding could be computed, the performance was very similar to integer encoding. In general, highly similar performance was reported for all target encoding conditions.

Prokhorenkova et al. (2018) compared their new CatBoost variant of target encoding to smoothed target encoding with no cross-validation, hold-out, and leave-one-out cross-validation on 8 datasets. While the CatBoost method performed best, hold-out came second and target encoding without cross-validation performed worst. Leave-one-out cross-validation was usually better than using no cross-validation, but not always.

Encodings

Feature encoding techniques are implemented in a number of general preprocessing packages. Examples in R are vtreat (Mount & Zumel, 2019), embed (Kuhn, 2018), and mlrCPO (Binder, 2018). In python, there is scikit-learn (Pedregosa et al., 2011) with its compatible category-encoding library (https://github.com/scikit-learn-contrib/categorical-encoding). We use and extend the mlrCPO package to investigate the encodings outlined below. Pseudocode is presented in the main text (for non-standard encodings), or in Appendix A (for standard methods).

Integer Encoding

The simplest strategy for categorical features is integer encoding (Appendix A). To make sure that the order of the observed levels has no effect on 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.

Frequency Encoding

Frequency Encoding (Appendix A) 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. 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.

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 (Appendix A). 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 with the (often arbitrary) reference category. In our study, the first level in alphabetical order (the default for factor variables in R) 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 by mode imputation prior to dummy encoding. In datasets with a high total number of feature levels, constructing all indicator variables can be detrimental to the predictive performance or at least tremendously increase the computational load. To limit the number of features, we sort all feature levels by frequency in the training set, collapse rare levels beyond some threshold to a single category, and apply indicator encoding to the reduced level set.

Hash Encoding

Hash Encoding (Appendix A) 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 to an integer $hash(l) \in \mathbb{N}$, based on its label. This integer is then transformed to an indicator representation, with a one in indicator column number $(hash(l) \mod hash.size) + 1$ and a zero in all remaining columns (Kuhn & Johnson, 2019). Some levels will be randomly 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). Our implementation wraps the functionality of the FeatureHashing R package (Wu & Benesty, 2018) which uses the 32-bit variant of MurmurHash3. Although FeatureHashing can hash multiple features (which also collapses levels across features), we transform each feature separately to ease the comparison with the other encoders.

Leaf Encoding

In leaf encoding (Algorithm 1), 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 the final tree. In that way, leaf encoding combines feature levels with similar target values. We use the rpart package in R (Therneau & Atkinson, 2018) which grows CARTs with categorical feature support. With rpart, trees can be easily pruned based on internal prediction error estimates from k-fold cross-validation. Here, we always use 10-folds. With pruning, an "optimal" number of new levels is automatically found by the encoder. However, note that this can lead to constant features in extreme situations when the minimal cross-validation error is observed with a constant prediction for all observations. For regression and binary classification, rpart automatically orders the levels at each split (as described in the introduction) to reduce the computational load. To tremendously speed up the computation for multiclass classification, our implementation uses the ordering approach introduced by the ranger package (Wright & König, 2019). This converts a categorical feature into numerics before handing it to rparts multiclass routine. New levels are encoded with the number of the terminal node with most observations during training. As the leaf numbering is arbitrary, encoded values are generally treated as a new categorical feature which has to be further encoded to numerical values. In our study, this is done by one-hot encoding.

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
```

Cluster Encoding

Another strategy to reduce the number of feature levels is cluster encoding (Algorithm 2). As briefly mentioned in Micci-Barreca (2001), hierarchical cluster analysis can be used to combine levels. The dendrogram is cut at a suitable height to achieve the desired number of levels, provided as a hyperparameter. Here, we use two kinds of information to determine the similarity between levels. First, levels should be combined if they show similar values on the target. For regression, this is expressed by the mean target value for each level in the training set. For classification, the frequency of each target class is computed for each feature level. Second, we want to combine levels based on their frequency in the training set. We compute the absolute difference between the frequency of the level and the mean frequency for all levels, to also encourage collapsing very frequent with very rare levels. The fastcluster R package (Müllner, 2013) is used to compute clusters. We always use euclidean distances with complete linkage, but all traditional metrics and agglomeration methods are possible. The cluster algorithm has no reasonable strategy to handle new levels. We encode them as missing values and use mode imputation on the training set to return the most frequent cluster. One-hot encoding is used to encode the resulting cluster labels.

Algorithm 2 Cluster Encoding

```
Training: require number of desired levels $V \in \mathbb{N}$
if task is regression then
    for all $l \in \mathcal{L}^{train}$ do $\bar{y}_l^{train} = \frac{\sum_{i:x_i^{train}=l} y_i^{train}}{N_l}$
        define $\boldsymbol{v}_l = (\bar{y}_l^{train}, \delta_l)^T$ with $\delta_l = \left| N_l - \frac{\sum_{k=1}^{L} N_k}{L} \right|$
if task is classification then
    for all $l \in \mathcal{L}^{train}$ do
        for all $c \in \mathcal{C}$ do $s_{lc} = \sum_{i:x_i^{train}=l} I(y_i^{train} = c)$
        define $\boldsymbol{v}_l = (s_{l1}, \dots, s_{lC}, \delta_l)^T$ with $\delta_l = \left| N_l - \frac{\sum_{k=1}^{L} N_k}{L} \right|$
1. compute distance matrix $\boldsymbol{D}^{L \times L}$ with $d_{jk} = ||\boldsymbol{v}_j - \boldsymbol{v}_k||$
2. fit hierarchical cluster analysis on $\boldsymbol{D}$
3. prune dendrogram to obtain $V$ combined levels

for all $x_i^{train} \in \boldsymbol{x}^{train}$ do $\tilde{x}_i^{train} = v$ with $x_i^{train}$ in leaf $v$
Prediction:
for $x^{new}$ do
    if $x^{new} \in \mathcal{L}^{train}$ then $\tilde{x}^{new} = v$ with $x^{new}$ in leaf $v$ else $\tilde{x}^{new} = NA$
```

Impact Encoding

The first formal description of an impact or target encoder was provided by Micci-Barreca (2001). The basic idea is to encode each feature level by the conditional target mean (regression) or the conditional relative frequency of one or more target classes (classification). The implementation from the mlrCPO package (Appendix A) is a version of smoothed target encoding, which is centered at the unconditional target mean. A logit transformation is performed in the classification setting. The impact encoder for classification transforms the original feature into C numeric feature, each representing one target class. A smoothing parameter ϵ is necessary for computational reasons (to avoid division by zero) and can be used to regularize towards the unconditional mean. However, we explicitly construct two new encoding strategies which add more efficient regularization. To compare those methods with simple target encoding, we always choose a small $\epsilon = 0.0001$ for the impact encoder.

GLMM Encoding

As mentioned in the introduction, smoothed target encoding 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 for example in (Kuhn & Johnson, 2019) who also provide some implementations in the embed R package (Kuhn, 2018). We constructed our own glmm encoders based on the mlrCPO package for regression, binary, and multiclass classification which are described in Algorithms 3, 4, and 5. Similar to one variant from the embed package, we use the lmer (regression) and glmer (classification) functions from the lme4 package in R (Bates, Mächler, Bolker, & Walker, 2015) as an efficient way to fit glmms. To further speed up the computation, we followed the performance tips from the lme4 vignette (https://cran.r-project.org/web/packages/lme4/vignettes/lmerperf.html): we did not compute derivations (`calc.derivs = FALSE`) and used the `NLOPT_LN_BOBYQA` optimizer from the nloptr package with liberal stopping criteria (`maxeval = 1000`, `xtol_abs = 1e-6`, `ftol_abs = 1e-6`). The encoded value for each level is based on the spherical conditional mode estimates computed by the lme4 package. In regression, the conditional modes are very similar to the mean target value for each level, weighted by the relative observed frequency of that level in the training set (Gelman & 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. In theory, an important advantage of using a glmm over target encoding with a smoothing parameter is that a reasonable amount of regularization is determined automatically. Thus, we do not have to tune the complete machine learning pipeline repeatedly with different smoothing values.

The glmm encoder is already equipped with an internal strategy to prevent overfitting to the target values in the training set. As mentioned earlier, another strategy to avoid overfitting in target encoding is to combine it with resampling. Cross-validation can be used to train the encoder on independent observations without limiting the data to train the machine learning model. To our knowledge, we provide the first implementation which combines target encoding based on glmms with cross-validation. During the training phase, we partition the data using a standard cross-validation scheme with n.folds and fit a glmm on each resulting training set. For each observation, there is exactly one glmm which did not use that observation for model fitting and can be safely used for encoding. Note that the n.folds cross-validation 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.

Algorithm 3 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
        $\hat{x}_i^{train} = cond\hat{M}ode_{\mathcal{D}^{train}}(\beta_{0l})$ 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
        $\hat{x}_i^{train} = cond\hat{M}ode_{\mathcal{D}_m^{train}}(\beta_{0l})$ 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
        $\hat{x}^{new} = cond\hat{M}ode_{\mathcal{D}^{train}}(\beta_{0l})$ 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 $\mathcal{D}^{train}$
```

Algorithm 4 GLMM Encoding Binary Classification

```
Training: require $n.folds \in \mathbb{N}$
fit simple glmm: $E(y_i^{train}) = h(\eta_i) = \frac{\exp(\eta_i)}{1+\exp(\eta_i)}$, $\eta_i = \beta_{0l} = \gamma_{00} + u_l$ on $\mathcal{D}^{train}$
with $u_l \overset{iid}{\sim} N(0, \sigma^2)$, $y_i^{train} \overset{ind}{\sim} Be(h(\eta_i))$ 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
        $\hat{x}_i^{train} = cond\hat{M}ode_{\mathcal{D}^{train}}(\beta_{0l})$ 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 glmm on each $\mathcal{D}_m^{train}$
    for all $x_i^{train} \in \boldsymbol{x}^{train}$ do
        $\hat{x}_i^{train} = cond\hat{M}ode_{\mathcal{D}_m^{train}}(\beta_{0l})$ 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
        $\hat{x}^{new} = cond\hat{M}ode_{\mathcal{D}^{train}}(\beta_{0l})$ 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 $\mathcal{D}^{train}$
```

Algorithm 5 GLMM Encoding Multiclass Classification

```
Training: require $n.folds \in \mathbb{N}$
define response matrix $\boldsymbol{Y}^{N \times C}$ with $y_{ic} = 1$ if $y_i^{train} = c$ and $y_{ic} = 0$ if $y_i^{train} \neq c$:
$\tilde{\mathcal{D}}^{train} = \{(x_i^{train}, y_{i1}^{train}, \ldots, y_{iC}^{train}), \ldots, (x_N^{train}, y_{N1}^{train}, \ldots, y_{NC}^{train})\}$
for all $C$ classes do
    fit simple glmm: $E(y_{ic}^{train}) = h(\eta_i) = \frac{\exp(\eta_i)}{1+\exp(\eta_i)}$, $\eta_i = \beta_{0l} = \gamma_{00} + u_l$ on $\boldsymbol{y}_c^{train}$ from $\tilde{\mathcal{D}}^{train}$
    with $u_l \overset{iid}{\sim} N(0, \sigma^2)$, $y_{ic}^{train} \overset{ind}{\sim} Be(h(\eta_i))$ and $x_i^{train} = l$, $l \in \mathcal{L}^{train}$
if $n.folds = 1$ then
    for all $C$ models do
        for all $x_i^{train} \in \boldsymbol{x}^{train}$ do
            $\hat{x}_{ic}^{train} = cond\hat{M}ode_{\tilde{\mathcal{D}}^{train}}(\beta_{0l})$ with $x_i^{train} = l$
else use $n.folds$ cross-validation scheme to make training sets $\tilde{\mathcal{D}}_1^{train}, \ldots, \tilde{\mathcal{D}}_{n.folds}^{train}$
    for all $C$ classes do
        fit simple glmm on $\boldsymbol{y}_c^{train}$ from each $\tilde{\mathcal{D}}_m^{train}$
        for all $x_i^{train} \in \boldsymbol{x}^{train}$ do
            $\hat{x}_{ic}^{train} = cond\hat{M}ode_{\tilde{\mathcal{D}}_m^{train}}(\beta_{0l})$ with $x_i^{train} = l$ based on the model $m$ for class $c$
            with $(x_i^{train}, y_i^{train}) \notin \tilde{\mathcal{D}}_m^{train}$
Prediction:
for $x^{new}$ do
    for all $C$ class models do
        if $x^{new} \in \mathcal{L}^{train}$ then
            $\hat{x}_c^{new} = cond\hat{M}ode_{\tilde{\mathcal{D}}^{train}}(\beta_{0l})$ with $x^{new} = l$ based on full model for class $c$
        else $\hat{x}_c^{new} = \hat{\gamma}_{00}$ based on full model for class $c$
```

Random Forest Encoding

Discussions on the Kaggle forum indicate that one way to improve target encoding is to ensure that a feature level is not always mapped to the same number. Thus, the consecutive machine learning model cannot easily overfit to the training data by identifying each feature level with its deterministic value. When using the glmm encoder with cross-validation, the n.folds models already provide slightly different numbers for the same feature level. If a large number of different values is related to good performance in target encoding, the cross-validation scheme cannot be expected to scale well for a high number of folds. Predictions become highly similar due to overlapping training sets and fitting many glmms (although quite fast) increases the computational load. We developed random forest (rf) encoding (Algorithms 6 and 7) as a new strategy which combines regularization with an efficient way to provide a large number of different numerical values for each feature level. A random forest of size num.trees is trained to predict the target based on the single categorical feature (so bagging encoder might be a more precise name). For each observation in the training set, we use only trees in which the observation was out-of-bag (OOB: not part of the sample used to fit the tree) to compute the random forest prediction. During the prediction phase, the complete forest is always used for the encoding. For new levels, we predict the average prediction across all leafs for each tree before averaging across all trees. Relative class frequencies are used as tree predictions in classification. The rf encoder for multiclass classification transforms the original feature into C numeric features, each representing one target class. We use the ranger function from the ranger R package (Wright & Ziegler, 2017) to compute the random forest (for classification, `probability = TRUE`) . Ranger offers a fast implementation, provides categorical feature support (`respect.unordered.factors = "order"`) and can handle new levels (which can happen when computing predictions for some trees during the training phase). Note that the regularization induced by the random forest might not be optimal for very rare levels. As the forest includes only the categorical feature as predictor, observations with the same level will always result in the same terminal node. For the extreme case of a level which was only observed once during training in combination with an extreme target value, the level will likely be encoded with the extreme target value of that single observation without any regularization. Unfortunately, ranger does not provide a hyperparameter which directly limits the minimum number of observations in each node (which would prevent that from happening). We increased the required minimum number of observations in a node to apply further splitting to `min.node.size = 20`, which should discourage very small terminal nodes. In addition, we use subsampling instead of bootstrapping (`replace = FALSE`, `sample.fraction = 0.632`) when growing the forest. This should further decrease the likelihood that rare levels get their own terminal node by avoiding that observations with rare levels (and possibly extreme target values) can be present more than once in each tree.

Algorithm 6 RF Encoding Regression and Binary Classification

```
Training: require number of trees $B \in \mathbb{N}$
fit random forest with trees $T_1, ..., T_B$
for all $x_i^{train} \in \boldsymbol{x}^{train}$ do
    if $x_i^{train}$ inbag for all $B$ trees then $\hat{x}_i^{train} = \frac{1}{B}\sum_{b=1}^{B} T_b(x_i^{train})$
    else $\hat{x}_i^{train} = \frac{1}{B_i^{OOB}} \sum_{b:\ x_i^{train}\ OOB\ T_b} T_b(x_i^{train})$
Prediction:
for $x^{new}$ do
    if $x^{new} \in \mathcal{L}^{train}$ then $\hat{x}^{new} = \frac{1}{B}\sum_{b=1}^{B} T_b(x^{new})$
    else $\hat{x}^{new} = \frac{1}{B}\sum_{b=1}^{B} \frac{1}{L} \sum_{l \in \mathcal{L}^{train}} T_b(l)$
```

To summarize, integer and frequency encoding are two simplistic strategies which transform a categorical feature into a numerical representation by making assumptions which are unreasonable or at least unlikely. The numerical mapping induced by indicator encoding does not loose information, but becomes increasingly inefficient as the number of levels increases. Thus, the strategy is to reduce the number of feature levels before using indicator encoding. Level reduction is achieved by collapsing small levels in our implementation of indicator encoding, by randomly collapsing levels with the hash encoder, or by using target information in the leaf and cluster encoders. Target encoders use actual predictions of the target variable as numerical feature values and only differ with respect to the origin of those predictions. In contrast to the basic impact encoder which directly uses the (centered) target mean for each level, glmm encoding gives stronger weights to more frequent levels. The newly developed rf encoder fits a different function to model the relationship between the categorical feature and the target, provides a different strategy to reduce overfitting, and ensures that different

Algorithm 7 RF Encoding Multiclass Classification

```
Training: require number of trees $B \in \mathbb{N}$
fit random forest with trees $T_1, ..., T_B$
for all $C$ classes do
    for all $x_i^{train} \in \boldsymbol{x}^{train}$ do
        if $x_i^{train}$ inbag for all $B$ trees then $\hat{x}_{ic}^{train} = \frac{1}{B}\sum_{b=1}^{B} T_b^c(x_i^{train})$
        else $\hat{x}_{ic}^{train} = \frac{1}{B_i^{OOB}} \sum_{b:\ x_i^{train}\ OOB\ T_b} T_b^c(x_i^{train})$
Prediction:
for $x^{new}$ do
    for all $C$ classes do
        if $x^{new} \in \mathcal{L}^{train}$ then $\hat{x}_c^{new} = \frac{1}{B}\sum_{b=1}^{B} T_b^c(x^{new})$
        else $\hat{x}_c^{new} = \frac{1}{B}\sum_{b=1}^{B} \frac{1}{L} \sum_{l \in \mathcal{L}^{train}} T_b^c(l)$
```

numbers are used to encode the same feature level.

Control Conditions

We include two control conditions to better understand the effectiveness of the investigated encoders: In the no encoding (none) condition, the performance of a featureless (FL) learner 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 observation in the test set. For each dataset, we also performed cross-validation for a random forest model without encoding, to compare the use of encoding methods in random forest with the natural categorical splitting approach. The random forest was again based on the ranger package (Wright & Ziegler, 2017). As mentioned, ranger (with `respect.unordered.factors = "order"`) provides efficient categorical feature support for an unlimited number of levels by ordering levels once before starting the tree growing algorithm.

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 small levels) for the remaining features. This condition is necessary to ensure that including high cardinality features does indeed improve predictive performance in all of our datasets.

Otherwise the best encoding might just provide the least impairment compared to not including any high cardinality features.

Research Questions

We investigate which encoding technique leads to the best performance in predictive modeling when predictor sets include high cardinality features. As optimal encoding might differ depending on the employed machine learning algorithm, we consider one representative of the following algorithmic classes: regularized linear models, random forests, gradient tree boosting, k-nearest neighbors, and support vector machines. A stronger focus will be placed on the random forest and gradient boosting. For the random forest, we compare encodings with native categorical feature support of the algorithm. The effectiveness of gradient boosting in combination with target encoding is of great interest, as this seems to be the method of choice in many machine learning competitions. Also, there seem to be conflicting reports about the performance of indicator and integer encoding in tree based models, which might be dependent on the number of feature levels.

To find possible default encodings for high cardinality features, we analyse a variety of datasets with different characteristics including regression, binary classification and multiclass classification problems. If reasonable, we compare encoding rankings across datasets and try to relate encoding rankings with dataset characteristics like the number of observations, the proportion of missing values or the maximum number of levels.

As the use of target encoding has increased in recent years, we put a special focus on impact, glmm, and rf encoding. It is important to know, under which circumstances those might be preferred over more simple approaches. We expect that target encoding becomes more effective as the number of levels increase. To investigate this notion, we look at datasets with different numbers of levels per feature. We also vary the minimum number of levels above which features are transformed by target encoding. The glmm and rf encodings should outperform impact encoding due to data dependent regularization. We constructed those encoders in the hope of finding a version of target encoding which does not require extensive hyperparameter tuning. Thus, we investigate performance differences under different parameter settings.

Finally, we will take a closer look at the performance of the more traditional methods: integer, frequency, indicator (after collapsing rare levels), hash, leaf, and cluster encoding. An interesting question will be whether we can give advice on when those might be preferred over target encoding. Here, we will also consider running times of the encoders. On a smaller note, we compare one-hot with dummy encoding to determine whether it is arbitrary which variant is used in practice. Most machine learning practitioners tend to prefer one-hot encoding, while indicators of the dummy type are the standard in the statistics community.

Table 1
Benchmark Datasets

OmlId	Name	Cl	N	NA%	Num	Bin	Cat	HighCardLevels
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,...,2016,4291,4291,4291,5073,5713,13990,15415,15415
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
41435	KDD98	2	191260	6.12	341	29	107	10,...,132,159,159,198,207,208,210,306,961,993,25847
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
41436	sf-police-incidents	2	2215023	0.00	3	0	5	10,12,16,25147
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
41443	Traffic__violations	3	1406824	0.13	1	10	13	19,26,33,68,70,70,1048,3636,7549,10087,16722

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 = levels for each categorical feature with at least 10 levels (some levels are not displayed for KDDCup09__upselling and KDD98).

Benchmark Setup

Datasets

Table 1 displays a summary of the datasets used in our benchmark study. All datasets can be downloaded from the OpenML platform (Vanschoren, N. van Rijn, Bischl, & Torgo, 2013) based on the name or the displayed OmlId. The dataset collection includes 8 regression problems, 12 binary classification and 7 multiclass classification problems (between 3 and 12 classes).

Sample sizes range between 736 and 2215023 observations (Md = 50789). The total number of features ranges between 5 and 477 (Md = 14). Datasets contain between 1 and 65 categorical features with more than 10 levels (Md = 4), with the maximum number of levels for one feature in a dataset ranging between 14 and 30114 (Md = 790). When only considering categorical features with more than 10 levels, the median for the number of levels within each dataset is between 10 and 19803 (Md = 54). Missing values are present in 0.56 percent of the datasets.

Several datasets have been used in earlier studies on predictive modeling with high cardinality features. Cerda et al. (2018) used variants of the datasets *open_payments*, *road-safety-drivers-sex*, *Traffic_violations*, *Midwest-survey*, *employee_salaries*, and *medical_charges*. Datasets *kick*, *Click_prediction_small*, and *kdd_internet_usage* (with slight variations) were featured in the CatBoost paper by Prokhorenkova et al. (2018). Datasets *eucalyptus*, *Amazon_employee_access*, *adult*, *KDDCup09_upselling*, *KDD98* were used in the development process of autoxgboost (Coors, 2018; Thomas et al., 2018).

Datasets *ames-housing*, *hpc-job-scheduling*, *churn*, and a variant of *okcupid-stem* were used in (Kuhn & Johnson, 2019). The dataset *nyc-taxi-green-dec-2016* was inspired by a blogpost by Jie Liu (https://blogs.oracle.com/r/computing-weight-of-evidence-woe-and-information-value-iv), *Diabetes130US* was used in a blogpost by Manuel Amunategui (http://amunategui.github.io/feature-hashing/), and *sf-police-incidents* was inspired by a blogpost by Nina Zumel (http://www.win-vector.com/blog/2012/07/modeling-trick-impact-coding-of-categorical - variables-with-many-levels/).

Machine Learning Algorithms and Hyperparameter Tuning

Being interested in running experiments on large datasets, we had to reduce the required amount of computational resources by keeping the tuning of hyperparameters at a minimum. This decision is reasonable, as we do not compare the predictive performance of different machine learning algorithms, which would only make sense if all algorithms were tuned optimally. Instead, we rank encoding strategies for each machine learning algorithm separately. Thus, we make the assumption that for each algorithm, observed performance differences between encoders are more or less independent of an expected performance increase resulting from more elaborate tuning.

Regularized linear models (LASSO) based on the least absolute shrinkage and selection operator (`alpha = 1`) were trained with the cv.glmnet function from the glmnet R package (Friedman, Hastie, & Tibshirani, 2010). The regularization parameter `lambda` was internally tuned with 5-fold cross-validation on a regular grid of size 100. Optimal `lambda` was determined by the minimal expected prediction error (regression: mean squared error, binary classification: binomial loss, multiclass classification: multinomial loss). An intercept was included and all features were standardized prior to model fitting.

Random forests (RF) were trained with the ranger function from the ranger R package (Wright & Ziegler, 2017). Trees (regression: `splitrule = "variance"`, classification: `splitrule = "gini"`) were fitted on 500 bootstrap samples with `mtry` equal to the square root of the number of features. The random forest algorithm can be expected to give reasonable results without tuning (Probst, Wright, & Boulesteix, 2019).

Gradient boosting models (GB) were trained with the tree booster (regression: `objective = "reg:linear"`, binary classification: `objective = "binary:logistic"`, multiclass classification: `objective = "multi:softprob"`) of the xgb.train function from the xgboost R package (T. Chen et al., 2018). The learning rate `eta` was set to 0.01 with package defaults for the remaining hyperparameters (`gamma = 0`, `min_child_weight = 1`, `max_depth = 6`, no L1/L2 regularization, no subsampling of features or observations). Early stopping was used to determine the optimal number of boosting iterations. Models were trained on only 80% of the original training set. Boosting was stopped when the estimated performance error (regression: root mean squared error, classification: misclassification error) on the remaining 20% did not improve for 5 consecutive iterations.

Unweighted k-nearest neighbors (KNN) were trained with the kknn function (`distance = 2`, `kernel = "rectangular"`) from the kknn R package (Schliep & Hechenbichler, 2016). As this algorithm was only of minor interest, a constant number of 15 neighbors was used in all tasks and a simple information gain filter was applied by mlrCPO (`method = "FSelectorRcpp_information.gain"`) to limit the number of features to 25. All features were standardized prior to model fitting.

Support vector machines (SVM) with Gaussian kernel were trained with the lsSVM (regression) and mcSVM (classification) functions from the liquidSVM R package (Steinwart & Thomann, 2017). The bandwidth parameter of the kernel `gamma` and the regularization parameter `lambda` were internally tuned on a regular grid of size 10×10 with 5-fold cross-validation. All features were standardized prior to model fitting. For multiclass problems, one vs. all models were trained. To obtain probability predictions in classification settings, the mcSVM function performs least-squares regression as in the regression case. For training sets with more than 50000 observations, local SVMs were trained on spatial cells (`cells = TRUE`, `partition_choice = 6`, `RETRAIN_METHOD = "SELECT_ON_EACH_FOLD"`) to avoid memory issues during computation.

Experimental Conditions

Benchmark conditions are summarized in Table 2. Resampling estimates of each machine learning algorithm (LASSO, RF, GB, KNN, SVM) applied to all 27 datasets were computed for different hyperparameter configurations of each encoding (integer, frequency, indicator, hash, leaf, cluster, impact, glmm, rf). In addition, resampling estimates of each machine learning algorithm applied to all datasets were computed for different configurations of the remove control condition. Finally, resampling estimates for the no encoding control condition on each dataset were computed for FL and RF.

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, cluster, impact, glmm, and rf 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 cluster encoding, the dendrogram is pruned to return at most HCT collapsed levels. 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 had to be 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.

For the indicator, glmm, and rf encoders, a second hyperparameter was varied in addition to HCT. For indicator encoding, we used either the one-hot or the dummy strategy to construct indicator variables. For glmm encoding, we performed cross-validation based on 1 (no cross-validation), 5, or 10 folds. For rf encoding, either 10, 50, or 100 trees were grown.

Machine Learning Pipeline

The following machine learning pipeline was used for all experimental conditions:

- Imputation I: Separate factor levels are created for missing values in categorical features with more than two categories. Missing values in binary features are imputed by the

Table 2
Benchmark Design

Encoder	HCT	Learner	indicator.type	n.folds	num.trees
integer	10,25,125	LASSO,RF,GB,KNN,SVM			
frequency	10,25,125	LASSO,RF,GB,KNN,SVM			
indicator	10,25,125	LASSO,RF,GB,KNN,SVM	one-hot,dummy		
hash	10,25,125	LASSO,RF,GB,KNN,SVM			
leaf	10,25,125	LASSO,RF,GB,KNN,SVM			
cluster	10,25,125	LASSO,RF,GB,KNN,SVM			
impact	10,25,125	LASSO,RF,GB,KNN,SVM			
glmm	10,25,125	LASSO,RF,GB,KNN,SVM		1,5,10	
rf	10,25,125	LASSO,RF,GB,KNN,SVM			10,50,100
none		FL,RF			
remove	10,25,125	LASSO,RF,GB,KNN,SVM			

Note. Each condition is applied to all 27 datasets. HCT = high cardinality threshold.

feature mode, missing values in numerical features are imputed by the mean feature value in the training data.

- Encoding: The now complete categorical features are transformed by the respective encoder. In the no encoding condition, the encoder does no transformation. The leaf, cluster, and remove conditions still return categorical features, while the remaining encoders return only numerical features.
- Imputation II: Another round of imputation is placed after the integer (mean imputation), cluster (mode imputation), and remove (mode imputation) conditions, to handle new levels which are transformed to missing values by those conditions during the prediction phase.
- Drop constants: We drop features which are constant in the training phase. As none of the problems did include constant columns before splitting the complete dataset into training and test sets, this step only removes constant features that are produced by the encoders. As mentioned earlier, leaf encoding is the prime candidate here.
- Final one-hot encoding: All remaining categorical features are transformed by one-hot encoding. This step is skipped in the no encoding condition, where all categorical

features remain untouched.

- Learner: Except for the no encoding condition, datasets exclusively contain numerical features at this point. In the training phase, the transformed data from each training set is used to fit the respective machine learning 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.

Resampling and Performance Measures

Expected predictive performance for all experimental conditions was estimated with 5-fold cross-validation (5-CV). Due to limited computational resources (long runtimes for the larger datasets in combination with high memory requirements) repeated cross-validation was not feasible. For classification problems, stratification was used to ensure a similar distribution of the target variable in each fold. Depending on the problem type, a different performance measure was used to define predictive performance. All performance measures were aggregated with the mean across test sets from each cross-validation run.

For regression we use the root mean squared error $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$ (n: number of observations) as the standard measure. For binary classification we use the area under the curve $AUC = \frac{U}{n_{pos} \cdot n_{neg}}$ (U: test statistic of the Mann-Whitney-U test; n_{pos} and n_{neg}: number of true labels in both classes), which can be interpreted as the probability that a randomly drawn observation of the positive class is scored higher than a randomly drawn observation from the negative class. For multiclass classification, we look at a simple extension of the AUC (Ferri, Hernández-Orallo, & Modroiu, 2009), $AUNU = \frac{\sum_{c=1}^{C} AUC_c}{C}$ (AUC_c: one vs. rest AUC of class c). As $AUNU$ does not take the class ratios into account, it punishes classifiers that achieve low performance for small classes.

Software and Computational Resources

The open source statistical software R (R Core Team, 2018) was used for all analyses. Datasets were pulled from the OpenML (Vanschoren et al., 2013) platform with the respective R package (Casalicchio et al., 2017). Encoding methods were implemented or taken from the mlrCPO package (Binder, 2018). The complete pipeline consisting of preprocessing, encoding, and the final machine learning algorithm was trained and resampled in the mlr framework (Bischl et al., 2016). The batchtools package (Lang, Bischl, & Surmann, 2017) was used to run the benchmark analysis on the Linux Cluster of the Leibniz Supercomputing Centre in Garching. Plots were created with ggplot2 (Wickham, 2016) and some extension packages. This manuscript was written with the packages knitr (Y. Xie, 2015) and papaja (Aust & Barth, 2018). All materials for this study, including reproducible code for this manuscript and the necessary result objects can be downloaded from https://osf.io/6fstx/.

Benchmark Results

A substantive amount of conditions could not be computed for *KDD98*, *Traffic_violations*, and *sf-police-incidents* due to memory problems. We will not report any results for those datasets. Some conditions with the SVM failed for the datasets *road-safety-drivers-sex, Amazon_employee_access, Click_prediction_small, open_payments, wine-reviews, medical_charges.* We completely remove those datasets for the SVM when computing ranks or other statistics to compare encodings across datasets.

Encoder Performance on the Dataset Level and Rankings Across Datasets

Mean performance estimates along with minimum and maximum performance from the 5 cross-validation folds are reported for all datasets in Figures 1 (regression), 2 (binary classification), and 3 (multiclass classification). To reduce the complexity induced by the hyperparameters HCT, indicator.type, n.folds, and num.trees we only display the parameter condition with the best performance for each combination of dataset × encoding × machine

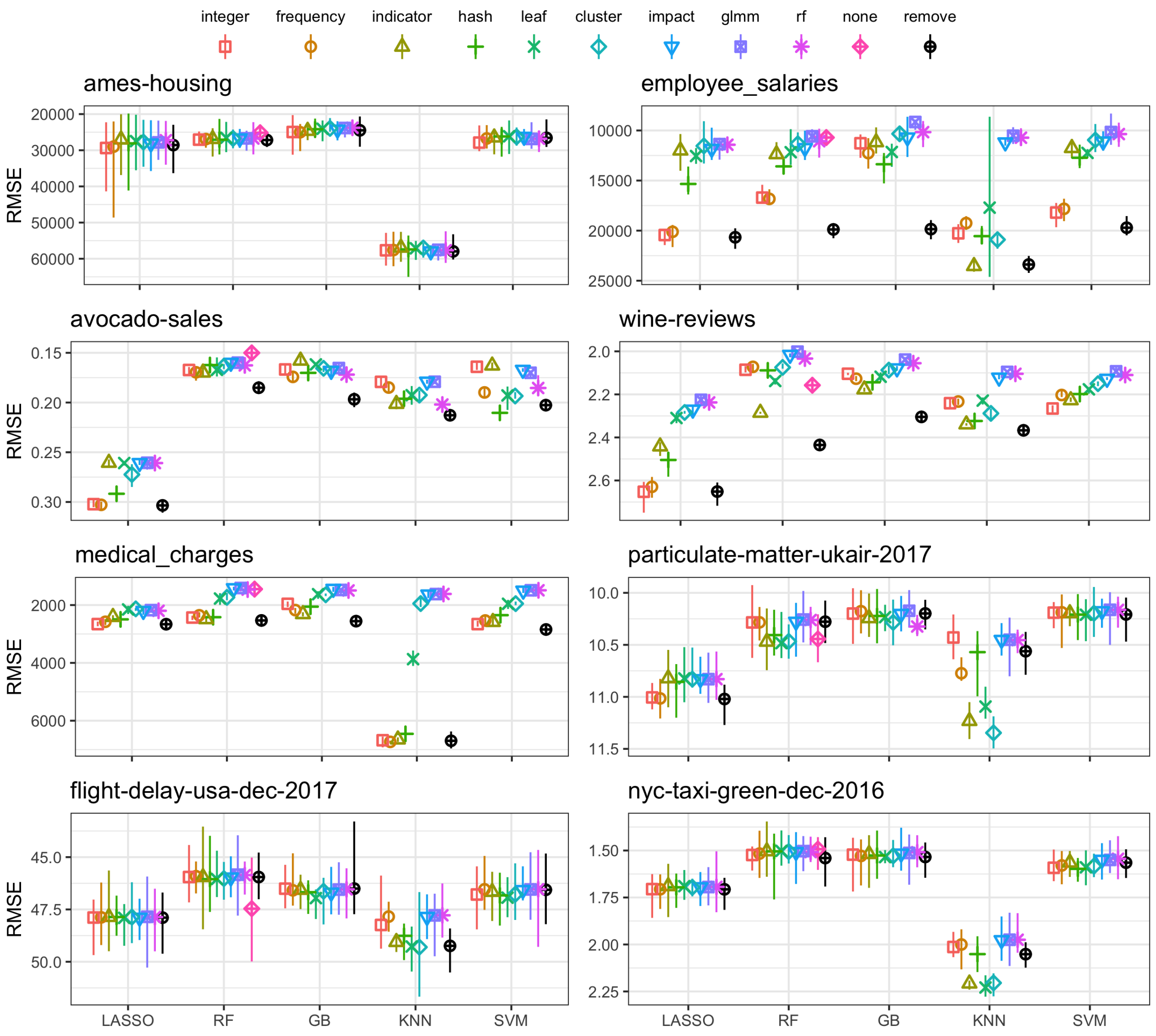

Figure 1. Performance estimates from 5-CV for regression (mean, min, max). For each condition, only the best hyperparameter combination is displayed. Note the reversed y-axis to ease visual interpretation.

learning algorithm. The y-axis differs for all datasets and is reversed for the RMSE to ease visual comparison.

Above chance predictions were possible for all datasets. Performance in some folds was below the FL learner for *flight-delay-usa-dec-2017* ($RMSE_{FL} = 48.81$) and *nyc-taxi-green-dec-2016* ($RMSE_{FL} = 2.22$). Some datasets turned out not to be very useful, as the best remove condition performed very similar to the other encodings (e.g. *ames-housing*, *porto-seguro*).

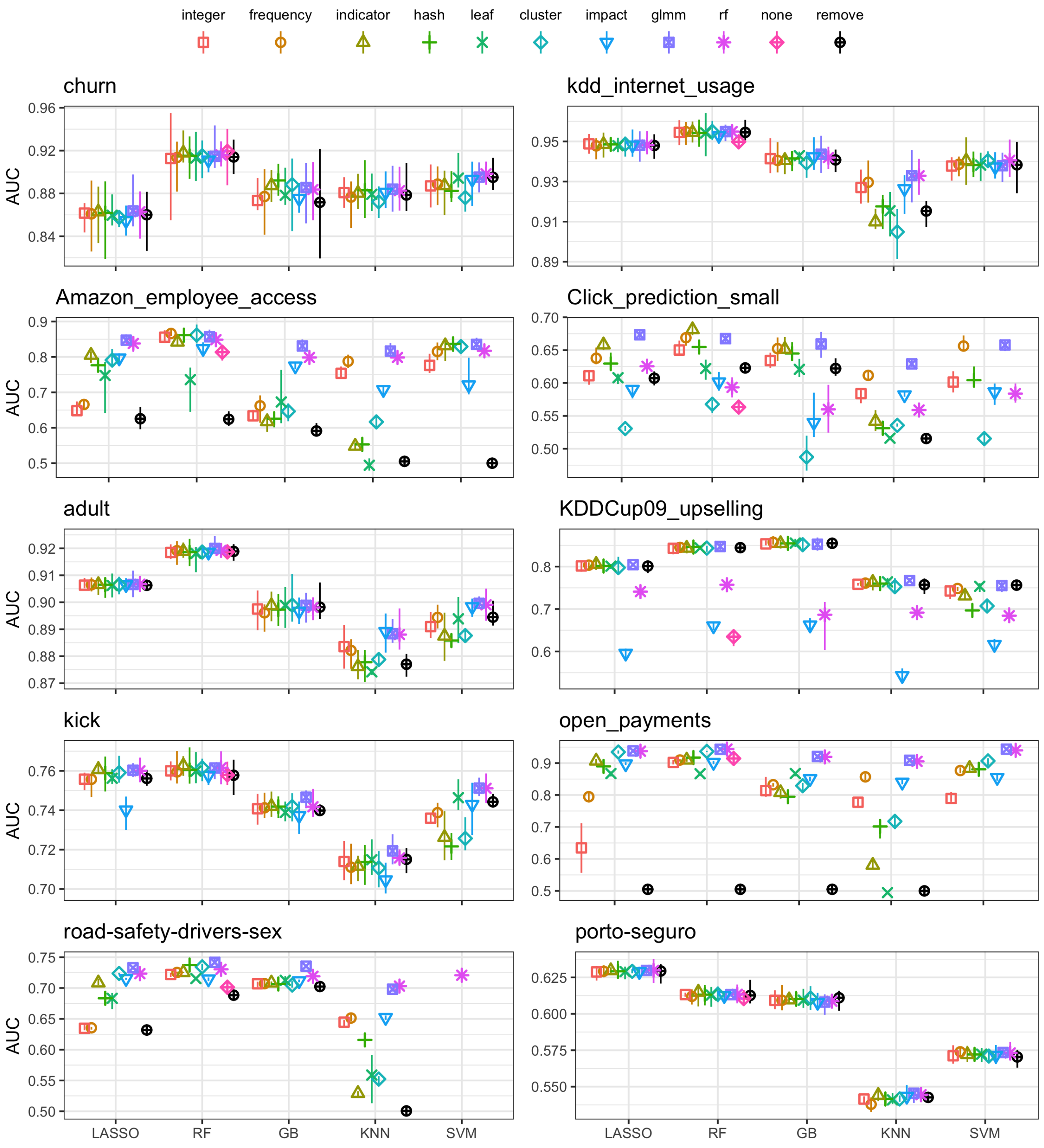

Figure 2. Performance estimates from 5-CV for binary classification (mean, min and max). For each condition, only the best hyperparameter combination is displayed.

On datasets where substantive performance differences could be observed, regularized target encoding was generally most effective. The best performance was mostly achieved by the glmm encoder, closely followed by rf encoding. The worst performance was often displayed by integer encoding, although there were exceptions of some encoders showing extremely bad

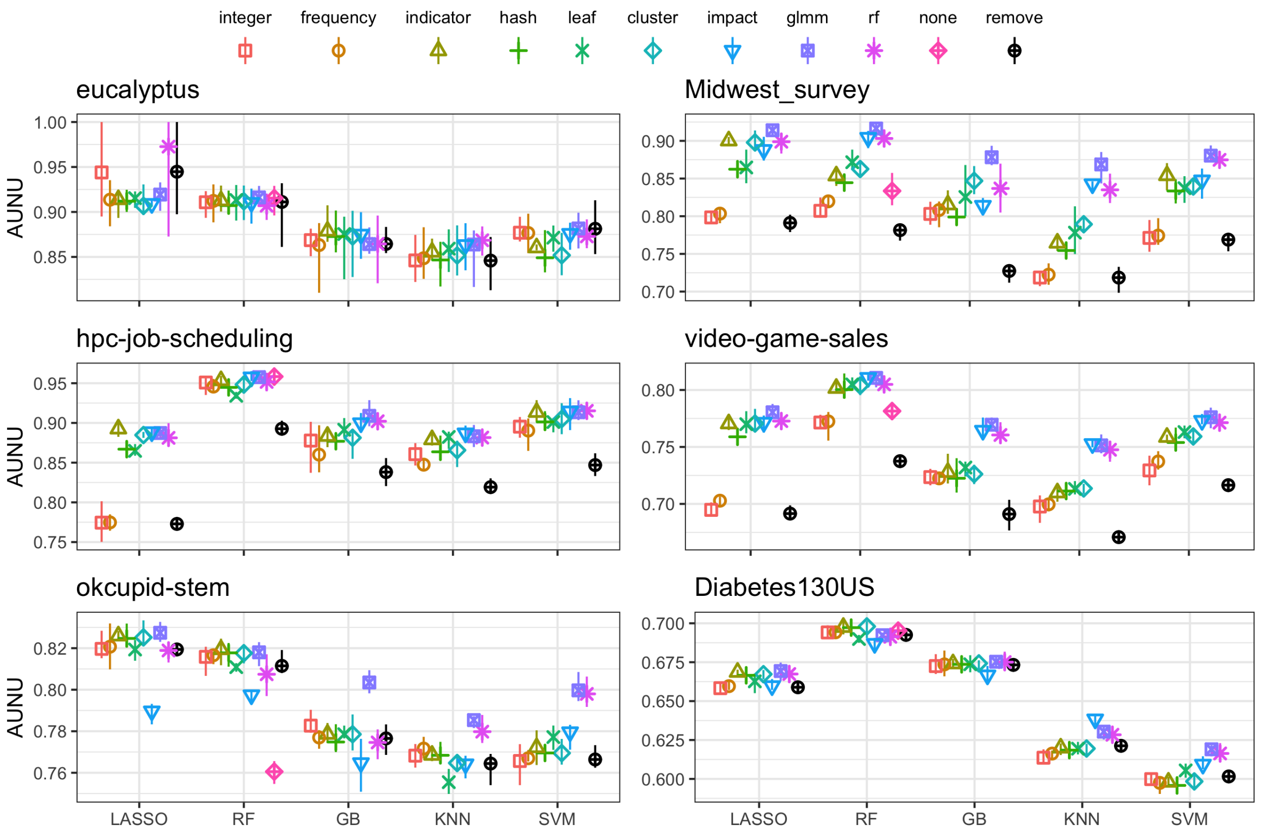

Figure 3. Performance estimates from 5-CV for multiclass classification (mean, min and max). For each condition, only the best hyperparameter combination is displayed.

performance in some conditions. For datasets *Click_prediction_small*, *KDDCup09_upselling*, *kick*, and *okcupid-stem* some combinations of encoder × machine learning algorithm performed clearly worse than removing high cardinality features.

To help summarize the results across datasets, we ranked the encodings for each combination of dataset × machine learning algorithm based on each cross-validation mean (again considering only the best hyperparameter conditions). Lower ranks indicate better performance. Then we computed mean ranks (across all datasets and separately for each problem type), which can be interpreted as computing a consensus relation based on the Borda approach (Borda, 1781; Hornik & Meyer, 2007). Mean encoder ranks for each machine learning algorithm and problem type are displayed in detail in Appendix A1. In line with our observations on the dataset level, winner and loser ranks were quite stable for each machine learning algorithm and different problem types, justifying a rough generalization across algorithms. Further averaging mean ranks across machine learning algorithms (initial ranks were recomputed

without the no encoding condition for the RF) produces the following meta ranking:
glmm > rf > indicator > impact > cluster > leaf > frequency > hash > integer > remove
However, note that rankings in the midfield including indicator, impact, cluster, leaf, frequency, hash are not stable and tend to switch places for different datasets and machine learning algorithms.

Correlations with Dataset Characteristics

For each machine learning algorithm, we computed Spearman correlations of all encoder ranks with the following dataset characteristics: total number of observations, total ratio of missing values, highest number of levels per feature. Due to the small number of datasets, those correlations are very noisy and should be interpreted with care (see Appendix A2). We will only focus on patterns which showed some stability across algorithms. The ranking of glmm encoding was better in datasets with a higher number of maximum levels per variable or with a higher ratio of missing values. The opposite pattern was found for the impact encoder, which was ranked worse in those cases. The ranking of integer encoding was better for datasets with more observations (except for the LASSO). A similar pattern was found for the frequency encoder, which additionally was also ranked better for higher ratios of missing values and higher numbers of maximum levels. The ranking of leaf encoding was worse in datasets with more observations. The no encoding condition with RF was ranked worse for datasets with a higher number of maximum levels.

Comparison of Target Encoders

All results indicate that (regularized) target encoding was superior or at least competitive on all datasets. While the impact encoding used no regularization, the amount of regularization was varied for the glmm and rf encoders. In Figures 4 (regression), 5 (binary classification), and 6 (multiclass classification) we reconsider performance of the target encoders for different hyperparameter settings. For each combination of dataset × encoding × parameter value (n.folds or num.trees) we only display the best HCT condition. When performance differences

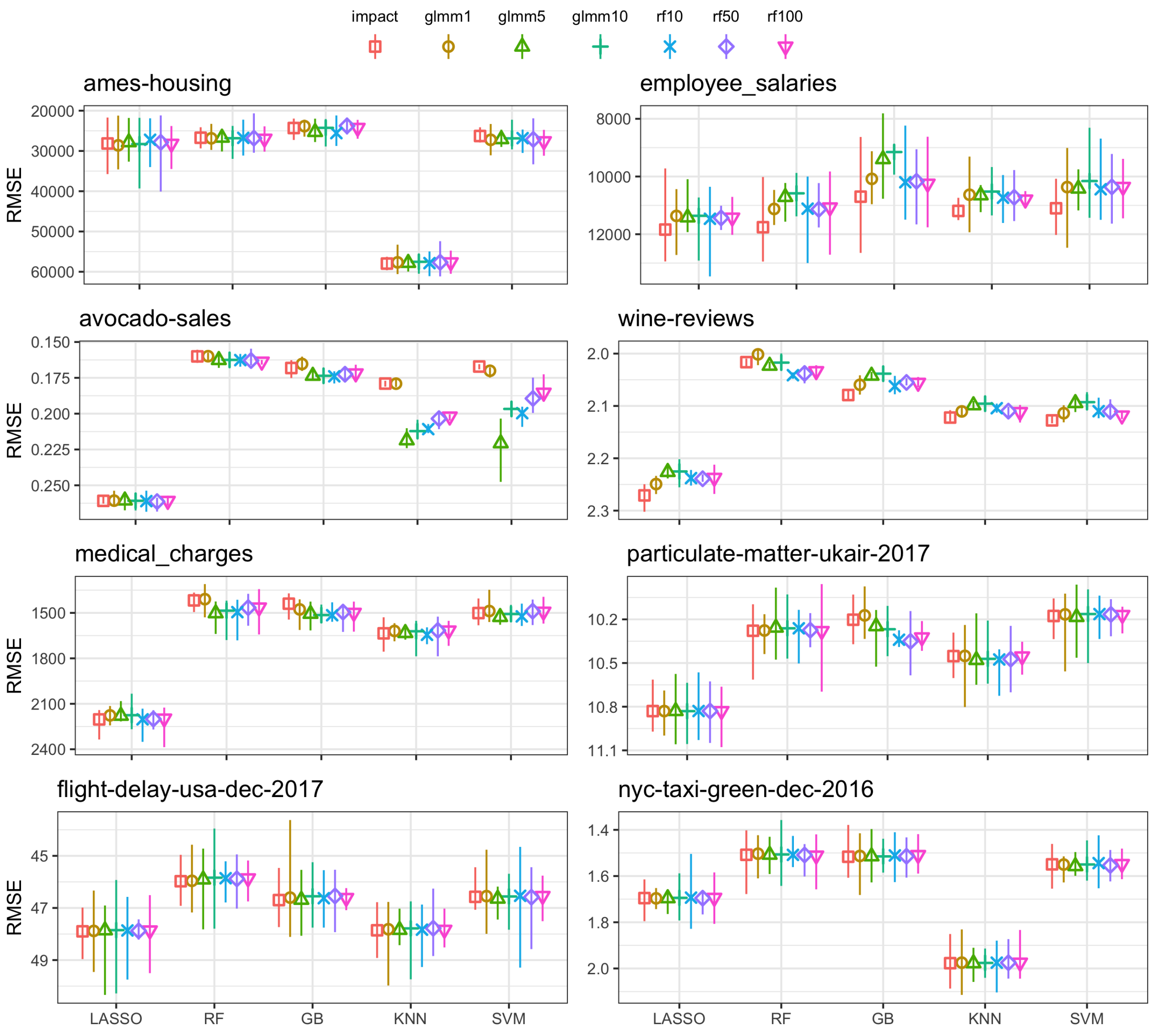

Figure 4. Performance estimates from 5-CV of target encoders with different hyperparameter settings (glmm: n.folds, rf: num.trees) for regression (mean, min, max). For each hyperparameter value of the encoder, the best HCT condition is displayed.

were observed, more regularization seemed beneficial. In those datasets, impact encoding not only performed worse than rf or glmm with cross-validation but was sometimes also inferior compared to other encoders. Interestingly, performance was always similar for glmm with 5 and 10 folds. Without cross-validation, the glmm performance was sometimes more similar to impact encoding and sometimes more similar to cross-validated glmm. The rf encoder mostly performed better than unregularized impact encoding but worse than glmm with cross-validation. Performance of rf encoding was very similar for different num.trees.

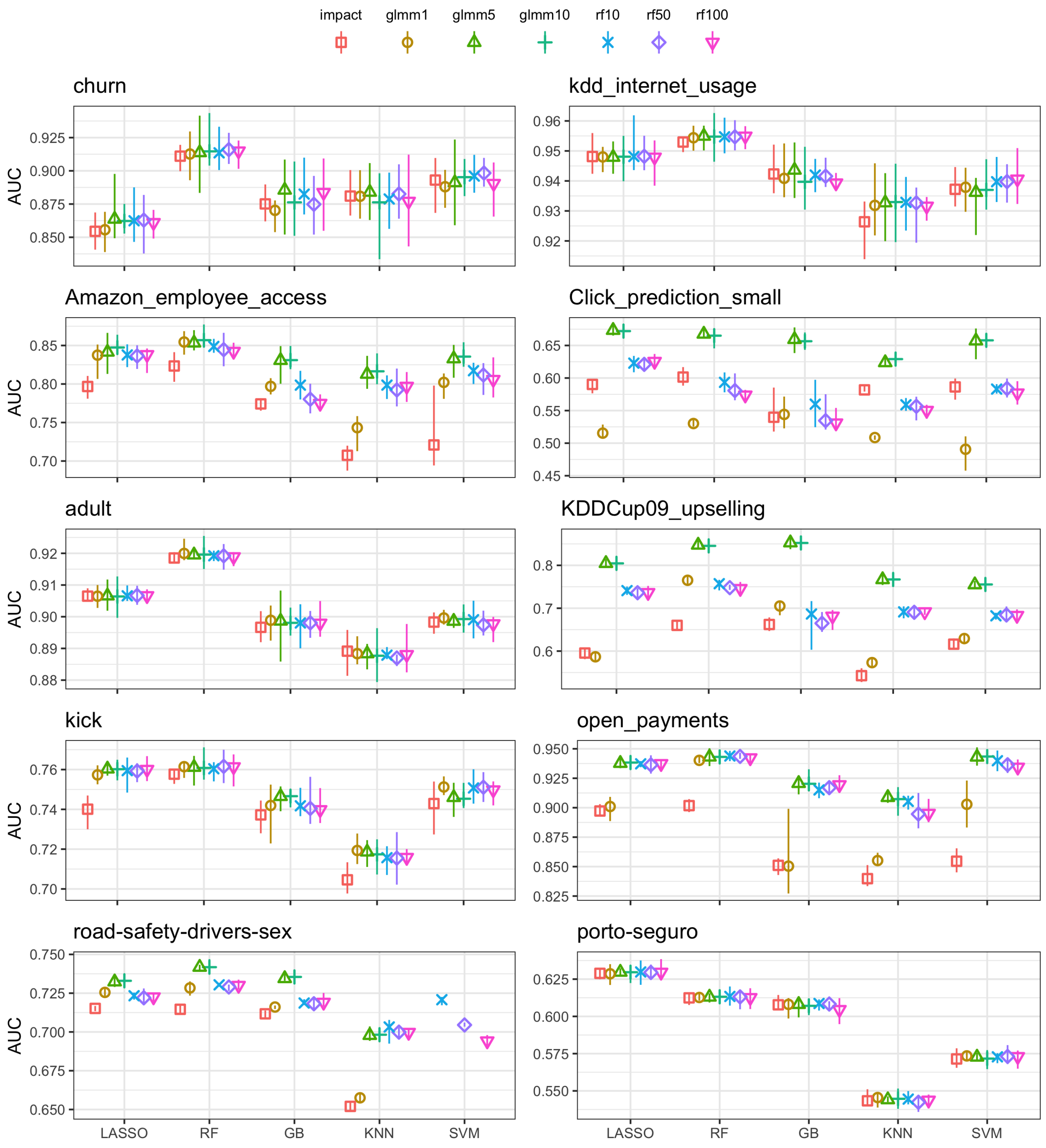

Figure 5. Performance estimates from 5-CV of target encoders with different hyperparameter settings (glmm: n.folds, rf: num.trees) for binary classification (mean, min, max). For each hyperparameter value of the encoder, the best HCT condition is displayed.

Comparison of Traditional Encodings

We did not observe a single combination of dataset × machine learning algorithm in which (regularized) target encoding was convincingly beaten by one of the more traditional strate-

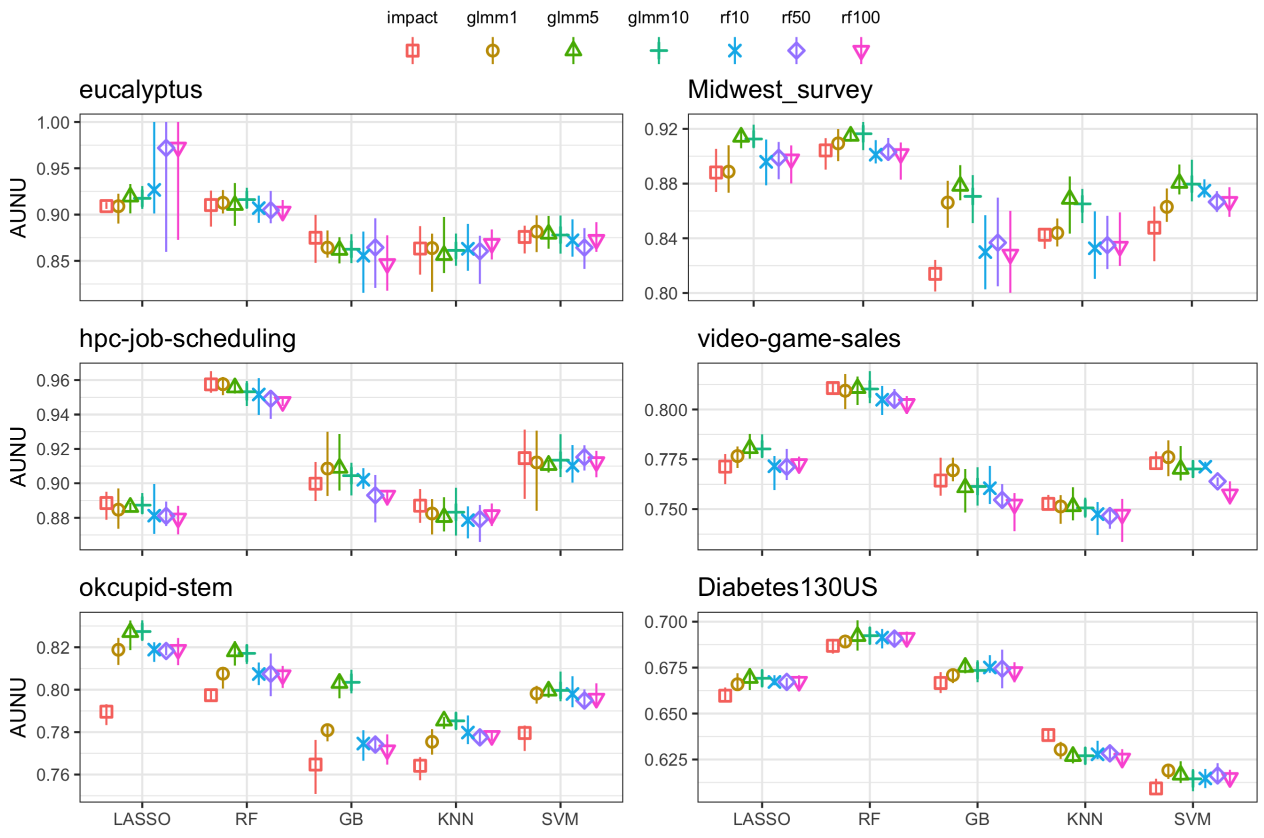

Figure 6. Performance estimates from 5-CV of target encoders with different hyperparameter settings (glmm: n.folds, rf: num.trees) for multiclass classification (mean, min, max). For each hyperparameter value of the encoder, the best HCT condition is displayed.

gies (integer, frequency, indicator, hash, leaf, cluster). Although some non-target encoder performed OK for many datasets, none performed well in all cases. When looking for a simple default method, indicator encoding in combination with collapsing small levels still seems to be one of the more robust strategies across algorithms and datasets. The leaf encoder sometimes produced constant features as indicated by similar performance to the remove condition (e.g. *Click_prediction_small*). The cluster encoding also showed extremely bad performance here, but was inconspicuous on most other datasets. Integer and frequency encoding performed bad, especially when combined with the LASSO. Hash encoding was better than expected as it did not rank last on any dataset.

To determine whether traditional encodings might be worth when facing limited computational resources, we also analyzed runtimes. Mean runtimes for the complete resampling of the machine learning pipeline with different encoders and machine learning algorithms (averaged

Table 3
Mean Ranks of Runtimes

Encoder	LASSO	RF	GB	KNN	SVM
integer	3.21	4.83	3.21	3.75	4.94
frequency	3.33	4.92	3.17	2.71	5.78
indicator	8.75	8.50	6.96	6.12	5.11
hash	7.92	9.29	6.79	5.04	5.00
leaf	4.42	5.21	5.42	5.42	5.17
cluster	7.38	7.75	6.29	6.33	5.72
impact	4.92	5.54	5.17	6.29	5.06
glmm	7.54	7.29	7.50	7.92	6.61
rf	5.33	6.67	6.96	7.92	6.17
none		2.29			
remove	2.21	3.71	3.54	3.50	5.44

Note. Runtime for 5-CV of the encoder is ranked for each combination of dataset x machine learning algorithm and than averaged across datasets. Ranks are based on the best performing hyperparameter settings. Low ranks indicate shorter runtimes. Datasets producing errors were removed for the SVM.

across datasets) are presented in Appendix A4. Unfortunately, absolute runtimes are hard to interpret. First, the distributions are heavily skewed as proportionally larger runtimes are observed for the big datasets. Second, while runtimes can be dominated by the encoder for small datasets, the training of the consecutive machine learning algorithm is clearly dominating for large datasets. Third, dependent on how many features are created by an encoder, this can tremendously increase training times for the consecutive model, again interacting with the size of the dataset. An in-depth analysis taking all of these factors into account is out of the scope of this manuscript. A simplified picture is drawn by Table 3, which shows encoder rankings of runtimes from 5-CV for each machine learning algorithm. Target encoding does not consistently have slower runtimes compared to simple strategies like indicator encoding. This indicates that a possible runtime vs. predictive performance trade-off might also be in favor of target encoding, especially for large datasets where a high number of indicator variables tremendously increases computational load.

Table 4
Percentages of different HCT values

Encoder	LASSO	RF	GB	KNN	SVM
integer	38,42,91	50,42,64	62,58,18	67,17,55	56,27,62
frequency	38,42,91	50,42,64	58,33,55	54,58,36	61,27,50
indicator	4,17,83	0,25,78	38,17,48	33,38,30	22,22,59
hash	4,35,94	12,35,81	25,35,62	50,30,31	33,29,70
leaf	43,42,82	50,50,55	61,42,45	70,42,27	61,27,50
cluster	17,35,75	29,35,56	38,48,25	46,43,19	50,24,50
impact	46,33,82	58,25,64	75,33,18	67,33,36	83,18,12
glmm	38,50,82	62,42,36	67,42,27	71,42,18	72,27,25
rf	46,42,73	54,42,55	54,50,45	67,25,45	78,18,25
remove	41,42,91	42,50,73	42,50,73	50,33,73	61,18,62

Note. For each encoder, the percentage of datasets in which the HCT values were 10, 25, 125 in the best performing hyperparameter setting. The total number is divided by the number of datasets in which the respective HCT value is available. Datasets producing errors were removed for the SVM.

For indicator encoding, our benchmark included both the one-hot and the dummy strategy. Although performance differences were generally small, one-hot outperformed dummy encoding in 69 percent of all direct comparisons. Win percentages for each combination of machine learning algorithm × problem type are presented in Appendix A3. Although win percentages are in favor of one-hot encoding in all cells, they seem most pronounced for the LASSO. Differences between problem types should not be overinterpreted as they strongly depend on single datasets.

Analysing High Cardinality Thresholds

Until now, we ignored the high cardinality threshold parameter by reporting only the condition with the best performance. Table 4 shows the percentage of datasets in which the best performing condition for each combination of encoding × machine learning algorithm was based on HCT equal to 10, 25, or 125 (frequencies are divided by the number of datasets in which each HCT setting was available). Note that this can only give a very rough description of the underlying factors. For the SVM, we completely removed all datasets where some conditions could not be computed, as failures seemed to be more frequent for HCT = 10.

In general, the optimal threshold seemed to be strongly data dependent, but a few weak patterns emerged: The LASSO showed a stronger preference for high HCT values with all encoders. This further suggests that the LASSO deals well with many indicator variables and might only profit from different encodings for features with a very high number of levels. Compared to the LASSO, target encoding with HCT = 10 seemed to be more effective on a larger proportion of datasets for the remaining algorithms (especially for KNN and SVM). The RF generally seemed to prefer higher HCT values compared to GB, KNN, and SVM. This might suggest that random forests are better in dealing with a high number of indicator variables than expected. The HCT pattern showed similarities between the three target encoders within each machine learning algorithm. Thus, the different regularizations might not require strongly different encoding thresholds.

Discussion

Use Target Encoding for High Cardinality Features

In our benchmark we compared encoding strategies for high cardinality features on a variety of regression, binary and multiclass classification datasets with different machine learning algorithms. Regularized target encoding (glmm, rf) was superior for most datasets and machine learning algorithms. Although the performance of other encoding strategies was comparable for some combinations of dataset $\times$ machine learning algorithm, target encoding was never outperformed. In general, our results suggest that regularized target encoding works well for all kinds of algorithms and should definitely be considered when working with high cardinality features.

Categorical feature support in the random forest based on the order once strategy of the ranger package often performed poorly on our datasets. For categorical features with a small number of levels, the ordering approach has been reported superior to indicator and integer encoding (Wright & König, 2019). In contrast, we observed bad performance that was clearly inferior to both standard encodings for datasets with a very high number of levels (*Click_prediction_small*, *KDDCup09_upselling*, and *okcupid-stem*). This was also reflected in the observed correlation between the no encoding rank for RF with the maximum number of levels per feature.

In line with discussions on the Kaggle platform, target encoding showed good performance with our boosting algorithm. Note that integer encoding did not perform well with XGBoost in our benchmark, although this has been recommended as a simple but effective strategy in the documentation of other boosting libraries (https://lightgbm.readthedocs.io/en/latest/Advanced-Topics.html#categorical-feature-support). Future research could compare glmm and rf encodings with the categorical feature support of CatBoost, as our encoders should provide better regularization than the simpler versions used as baseline in Prokhorenkova et al. (2018).

Target encoding was also very effective for algorithms which are not based on trees. For LASSO, earlier studies have suggested that indicator encoding should work quite well, even with a very high number of levels (Cerda et al., 2018). Although the glmm encoder ranked first in our benchmark for the LASSO, it was the only algorithm where indicator encoding ranked second. 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 high number of datasets. Thus, it cannot be ruled out that the performance of indicator encoding might further increase when allowing a higher number of indicator variables per categorical feature.

Both KNN and SVM rely on numerical distances in feature space and tend to perform poorly in the presence of a very large feature set. Thus, we expected that target encoding should work well here as it transforms categories into a single, smooth numerical feature. This notion was backed by our benchmark results. The performance increase with target encoding was especially impressive for KNN. For datasets like *medical_charges*, *road-safety-drivers-sex*, and *Midwest_survey* KNN (without tuning of the optimal number of nearest neighbors) could compete with other machine learning algorithms when combined with target encoding, but performed horribly with other encoders. Although consistent with the big picture, our results for the SVM have to be considered with care, as some experimental conditions resulted in unexpected computational errors.

In line with earlier research (Micci-Barreca, 2001; Prokhorenkova et al., 2018), target encoding with regularization (glmm and rf) 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 clearly outperformed impact and rf encoding on some datasets. Our tuning results show that combining target encoding based on glmms with 5-fold cross-validation led to improved predictive performance on many datasets. Performance did not improve further when using 10 folds, suggesting that 5 folds might be a good default in practice. This suggests that the glmm encoder has a clear advantage over target encoding with a smoothing hyperparameter (Micci-Barreca, 2001), as costly repeated training of the whole machine learning pipeline with different smoothing values is not required.
The newly developed rf encoder generally performed well but was slightly inferior to glmm encoding. The rf encoder was beaten by the glmm encoding on *Click_prediction_small*, *KDDCup09_upselling*, and *okcupid-stem* which all include high cardinality features with several thousand levels. This suggests that the rf encoder has a problem with many rare levels. As mentioned earlier, the current implementation of rf encoding cannot prevent that small levels with extreme target values form their own terminal node which can lead to a bad performance for those levels due to a lack of regularization. However, an even better explanation for the inferior performance might be what has been termed the "absent level problem" (Au, 2018). With categorical features, trees in a random forest sometimes need to make predictions for new levels. Even if a level was observed in the complete training set, the level might not have been present in the data used to train some of the trees due to the internal subsampling (or bagging). To deal with this situation, random forest implementations frequently use a default direction for unobserved levels. For example, the ranger package currently sends unobserved levels to the left child node of each split in which the categorical feature would be required. As presented by Au (2018), this is problematic in combination

with a splitting algorithm based on ordering levels. As explained in our introduction, ranger treats a categorical feature as a numeric variable constructed by ordering all levels based on target statistics. The left child node always contains levels with on average smaller target statistics in the training set, thus, predictions for new observations are downwardly biased. In the rf encoder, the categorical features is the only predictor variable in each tree which further amplifies that problem: if its feature level was not present for the training of that tree, an observation will be deterministically sent to the leftmost terminal node which makes the smallest prediction of all terminal nodes. In contrast, we would prefer an average prediction for new levels which is achieved in the glmm encoder. A simple solution would be to determine the default direction of new levels for each split at random (Au, 2018). Unfortunately, a random option is not available in ranger at the moment. Without changes in ranger, rf encoding could use only those trees which contained the respective level during training. With the present implementation, tuning results suggest that the rf encoder is quite insensitive to the used number of trees. Interestingly, performance did not increase when using more than 10 trees. This is in line with research on using random forests to imputate missing values (Shah, Bartlett, Carpenter, Nicholas, & Hemingway, 2014), where 10 trees seem to be sufficient as well. If trees would be omitted for encoding when a level is absent, more trees might lead to better performance. After fixing the absent level problem, future benchmarks should investigate whether the rf encoder can reach or even surpass the performance of glmm encoding.

None of the more traditional encoders could reliably compete with target encoding and we could not identify datasets or data characteristics of high cardinality features for which any of those methods might be superior. Our analyses did not show a tremendous increase in runtime with the glmm or rf encoders, which further strengthens our recommendation to use target encoding as the standard strategy to handle high cardinality features. On a side note, we found that one-hot encoding usually gave slightly better performance than dummy encoding. This phenomenon has also been observed by Chiquet, Grandvalet, and Rigaill (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 predictive modeling applications. Our results suggest that one-hot encoding might also be the better standard compared to dummy encoding when applying nonlinear regularized machine learning models like RF, GB or SVM with (high cardinal) categorical features.

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. We were surprised that 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 clearly 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 strategies for all affected features alongside one-hot encoding for the remaining ones. Although this strategy appears reasonable, it might be beneficial to make the decision whether to use target encoding on a feature by feature basis. Machine learning 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 might help to decide which encoders could be omitted from consideration.

Limitations

In our study, we made the following decisions to reduce the computational burden:
We only used minimal tuning for our machine learning algorithms. 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 relatively high stability of encoder rankings between

algorithms.

We only used 5-CV without repetitions to estimate predictive performance. Fortunately, the variance between folds was small enough to visually detect performance differences of encoders for many datasets. We decided against using hypothesis tests in comparing encoder performance and in computing rankings. With more computational resources, subsampling should be used as resampling strategy to provide a robust basis to compare encoders, as suggested in the benchmark analysis frameworks by Hothorn, Leisch, Zeileis, and Hornik (2005) and Eugster, Leisch, and Strobl (2014).

Our investigated encoders are model agnostic and can be combined with any supervised machine learning algorithm. In neural networks, it is possible to include a linear entity embedding layer that transforms a one-hot or integer encoded feature prior to any dense layers (Guo & Berkhahn, 2016). Original feature levels are coerced to a smaller number of embedded levels represented by the embedding neurons, so both the encoding and the machine learning model are trained simultaneously. In theory, a simple neural network consisting only of the embedding layer and a single dense layer can be trained as a feature encoder and the output of the embedding neurons can be fed to any consecutive machine learning model. This has been implemented in the embed package in R (Kuhn, 2018). Although the original publication suggested that embeddings extracted from a large network can improve the performance of other machine learning algorithms (Guo & Berkhahn, 2016), entity embeddings should be most efficient when naturally integrating them into a neural network. As we did not have the computational resources to include multilayered neural networks as one of our machine learning algorithms, we decided an entity embedding encoder.

By conditioning on the best hyperparameter setting when describing our benchmark results, we slightly favor the glmm and rf encoders which have a higher number of available parameter settings. However, our analyses revealed a stable pattern for different amounts of regularization in target encoding. Thus, it seems unlikely that glmm and rf simply profitted from noise introduced by more hyperparameter values. For a clean comparison, hyperparameter tuning for the encoders would have to be included into the resampling process. Comparing encoders with different hyperparameter settings would not have been possible in that case and we did not have the computational resources to compute both regimes.

Our encoding strategies work for all types of high categorical features as all encoders are invariant to the original labeling of the levels. If levels are labeled with text strings and similarity between levels is related to similarity between labels, more specialized encodings can effectively use this additional information (Cerda et al., 2018).
We only investigated univariate encodings which treat each categorical feature separately. An extension would be to encode multiple features at the same time. Note that in univariate target encoding, levels with comparable main effects cannot be distinguished based on the transformed feature, which prevents the consecutive machine learning algorithm from learning interactions. In theory, it would be easy to include multiple categorical features and binary interaction terms into the target encoding model. This is possible in CatBoost (Prokhorenkova et al., 2018) and has also been considered on Kaggle (https://www.kaggle.com/vprokopev/mean-likelihood-encodings-a-comprehensive-study). Our newly developed rf encoder might show great potential in this more complex setting, due to the natural handling of interactions in tree models.

Machine learning repositories like UCI (Dua & Graff, 2017) or OpenML (Vanschoren et al., 2013) currently do not contain many datasets with categorical features, and even less with a high number of levels. Although we used more datasets than earlier studies and uploaded some new ones, the pool is still very limited and cannot be considered a representative random sample from the kind of high cardinality problems encountered in practical machine learning applications. Thus, it is unclear how well our results generalize.

Note that in the current implementation of cluster encoding, we forgot to standardize the artificial variables representing frequencies and target characteristics before computing similarities between levels. Although this operation will probably improve the cluster encoder, we do not expect the new performance to be competitive with glmm or rf encoding.

Conclusion

This benchmark study compared 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 was a target encoder which combines simple generalized linear mixed models with cross-validation and does not require excessive tuning of hyperparameters. A newly developed target encoder based on predictions from a random forest also showed promising results but the current implementation seems to have problems with a very high number of rare feature levels.

References

Au, T. C. (2018). Random forests, decision trees, and categorical predictors: The "absent levels" problem. *J. Mach. Learn. Res.*, *19*(1), 1737–1766. Retrieved from http://dl.acm.org/citation.cfm?id=3291125.3309607

Aust, F., & Barth, M. (2018). *papaja: Create APA manuscripts with R Markdown.* Retrieved from https://github.com/crsh/papaja

Bates, D. (2018). Computational methods for mixed models. *Vignette for Lme4.* Retrieved from https://cran.r-project.org/web/packages/lme4/vignettes/Theory.pdf

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

Binder, M. (2018). *MlrCPO: Composable preprocessing operators and pipelines for machine learning.* Retrieved from https://github.com/mlr-org/mlrCPO

Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., … Jones, Z. M. (2016). mlr: Machine learning in r. *Journal of Machine Learning Research*, *17*(170), 1–5. Retrieved from http://jmlr.org/papers/v17/15-066.html

Borda, J. C. de. (1781). Mémoire sur les élections au scrutin.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. doi:10.1023/A:1010933404324

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees.* CRC press.

Casalicchio, G., Bossek, J., Lang, M., Kirchhoff, D., Kerschke, P., Hofner, B., … Bischl, B. (2017). OpenML: An R package to connect to the machine learning platform OpenML. *Computational Statistics*, 1–15. doi:10.1007/s00180-017-0742-2

Cerda, P., Varoquaux, G., & Kégl, B. (2018). Similarity encoding for learning with dirty categorical variables. *Machine Learning*, *107*(8), 1477–1494. doi:10.1007/s10994-018-5724-2

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22Nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794). New York, NY, USA: ACM. doi:10.1145/2939672.2939785

Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., … Li, Y. (2018). *Xgboost: Extreme gradient boosting*. Retrieved from https://CRAN.R-project.org/package=xgboost

Chiquet, J., Grandvalet, Y., & Rigaill, G. (2016). On coding effects in regularized categorical regression. *Statistical Modelling*, *16*(3), 228–237. doi:10.1177/1471082X16644998

Coors, S. (2018). *Automatic gradient boosting* (Master's thesis). LMU Munich. Retrieved from https://epub.ub.uni-muenchen.de/59108/1/MA_Coors.pdf

Coppersmith, D., Hong, S. J., & Hosking, J. R. (1999). Partitioning nominal attributes in decision trees. *Data Mining and Knowledge Discovery*, *3*(2), 197–217. doi:10.1023/A:1009869804967

Dua, D., & Graff, C. (2017). UCI machine learning repository. University of California, Irvine, School of Information; Computer Sciences. Retrieved from http://archive.ics.uci.edu/ml

Eugster, M. J., Leisch, F., & Strobl, C. (2014). (Psycho-) analysis of benchmark experiments: A formal framework for investigating the relationship between data sets and learning algorithms. *Computational Statistics & Data Analysis*, *71*, 986–1000. doi:10.1016/j.csda.2013.08.007

Ferri, C., Hernández-Orallo, J., & Modroiu, R. (2009). An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, *30*(1), 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 C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems 28* (pp. 2962–2970). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf

Fisher, W. D. (1958). On grouping for maximum homogeneity. *Journal of the American Statistical Association*, *53*(284), 789–798. doi:10.1080/01621459.1958.10501479

Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, *29*(5), 1189–1232. Retrieved from http://www.jstor.org/stable/2699986

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, *33*(1), 1–22. Retrieved from http://www.jstatsoft.org/v33/i01/

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

Guo, C., & Berkhahn, F. (2016). Entity embeddings of categorical variables. *arXiv Preprint arXiv:1604.06737*.

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*(3), 523–541. doi:10.1111/j.1467-985X.1997.00078.x

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning*. Springer.

Hornik, K., & Meyer, D. (2007). Deriving consensus rankings from benchmarking experiments. In *Advances in data analysis* (pp. 163–170). Springer. doi:10.1007/978-3-540-70981-7_19

Hothorn, T., Leisch, F., Zeileis, A., & Hornik, K. (2005). The design and analysis of benchmark experiments. *Journal of Computational and Graphical Statistics*, *14*(3), 675–699. doi:10.1198/106186005X59630

Ishwaran, H., & Kogalur, U. (2019). *Random forests for survival, regression, and classification (rf-src).* manual. Retrieved from https://cran.r-project.org/package=randomForestSRC

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., … Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems 30* (pp. 3146–3154). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf

Kuhn, M. (2018). *Embed: Extra recipes for encoding categorical predictors.* Retrieved from https://CRAN.R-project.org/package=embed

Kuhn, M., & Johnson, K. (2019). *Feature engineering and selection: A practical approach for predictive models.* Retrieved from http://www.feat.engineering/index.html

Lang, M., Bischl, B., & Surmann, D. (2017). Batchtools: Tools for r to work on batch systems. *The Journal of Open Source Software*, *2*(10). doi:10.21105/joss.00135

Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. *R News*, *2*(3), 18–22. Retrieved from https://CRAN.R-project.org/doc/Rnews/

Micci-Barreca, D. (2001). A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *SIGKDD Explor. Newsl.*, *3*(1), 27–32. doi:10.1145/507533.507538

Mount, J., & Zumel, N. (2019). *Vtreat: A statistically sound 'data.frame' processor/conditioner.* Retrieved from https://CRAN.R-project.org/package=vtreat

Müllner, D. (2013). fastcluster: Fast hierarchical, agglomerative clustering routines for

R and Python. *Journal of Statistical Software*, *53*(9), 1–18. Retrieved from http://www.jstatsoft.org/v53/i09/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830. Retrieved from http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

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*(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 S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems 31* (pp. 6638–6648). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/7898-catboost-unbiased-boosting-with-categorical-features.pdf

R Core Team. (2018). *R: A language and environment for statistical computing.* Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Schliep, K., & Hechenbichler, K. (2016). *Kknn: Weighted k-nearest neighbors.* Retrieved from https://CRAN.R-project.org/package=kknn

Shah, A. D., Bartlett, J. W., Carpenter, J., Nicholas, O., & Hemingway, H. (2014). Comparison of Random Forest and Parametric Imputation Models for Imputing Missing Data Using MICE: A CALIBER Study. *American Journal of Epidemiology*, *179*(6), 764–774. doi:10.1093/aje/kwt312

Steinwart, I., & Thomann, P. (2017). liquidSVM: A fast and versatile svm package. *ArXiv* *e-prints 1702.06899*. Retrieved from http://www.isa.uni-stuttgart.de/software

Strobl, C., Boulesteix, A.-L., Zeileis, A., & Hothorn, T. (2007). Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, *8*(1), 25. doi:10.1186/1471-2105-8-25

Therneau, T., & Atkinson, B. (2018). *Rpart: Recursive partitioning and regression trees*. Retrieved from https://CRAN.R-project.org/package=rpart

Thomas, J., Coors, S., & Bischl, B. (2018). *Automatic gradient boosting*. Retrieved from http://arxiv.org/abs/1807.03873v2

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* (pp. 847–855). New York, NY, USA: ACM. doi:10.1145/2487575.2487629

Tutz, G., & Gertheiss, J. (2016). Rejoinder: Regularized regression for categorical data. *Statistical Modelling*, *16*(3), 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*.

Wickham, H. (2016). *Ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. Retrieved from http://ggplot2.org

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), 1–17.

doi:10.18637/jss.v077.i01

Wu, W., & Benesty, M. (2018). *FeatureHashing: Creates a model matrix via feature hashing with a formula interface.* Retrieved from https://CRAN.R-project.org/package=FeatureHashing

Xie, Y. (2015). *Dynamic documents with R and knitr* (2nd ed.). Boca Raton, Florida: Chapman; Hall/CRC. Retrieved from https://yihui.name/knitr/

Appendix

Pseudocode for Encoders

Algorithm 8 Integer Encoding

```
Training:
compute random permutation $\boldsymbol{int} = (int_1, \ldots, int_k, \ldots, int_L)^T$ of $(1, \ldots, L)^T$
**for all** $x_i^{train} \in \boldsymbol{x}^{train}$ **do** $\hat{x}_i^{train} = int_k$ with $x_i^{train} = l_k$, $k = 1, \ldots, L$
Prediction:
**for** $x^{new}$ **do**
    **if** $x^{new} \in \mathcal{L}^{train}$ **then** $\hat{x}^{new} = int_k$ with $x^{new} = l_k$, $k = 1, \ldots, L$ **else** $\hat{x}^{new} = NA$
```

Algorithm 9 Frequency Encoding

```
  Training:
  for all $x_i^{train} \in \boldsymbol{x}^{train}$ do $\hat{x}_i^{train} = \frac{N_l}{N}$ with $x_i^{train} = l$
  Prediction:
  for $x^{new}$ do
      if $x^{new} \in \mathcal{L}^{train}$ then $\hat{x}^{new} = \frac{N_l}{N}$ with $x^{new} = l$ else $\hat{x}^{new} = 1$
```

Algorithm 10 One-Hot Encoding

```
  Training:
  for all $x_i^{train} \in \boldsymbol{x}^{train}$ do
      for all $l \in \mathcal{L}^{train}$ do $\hat{x}_{il}^{train} = I(x_i^{train} = l)$
  Prediction:
  for $x^{new}$ do
      for all $l \in \mathcal{L}^{train}$ do
          if $x^{new} \in \mathcal{L}^{train}$ then $\hat{x}_l^{new} = I(x^{new} = l)$ else $\hat{x}_l^{new} = 0$
```

Algorithm 11 Dummy Encoding

```
  Training:
  for all $x_i^{train} \in \boldsymbol{x}^{train}$ do
      for all $l \in \mathcal{L}^{train} \setminus l_{ref}$ do $\hat{x}_{il}^{train} = I(x_i^{train} = l)$
  Prediction:
  for $x^{new}$ do
      for all $l \in \mathcal{L}^{train} \setminus l_{ref}$ do
          if $x^{new} \in \mathcal{L}^{train}$ then $\hat{x}_l^{new} = I(x^{new} = l)$ else $\hat{x}_l^{new} = NA$
```

Algorithm 12 Hash Encoding

```
  Training: require $hash.size \in \mathbb{N}$
  for all $l \in \mathcal{L}^{train}$ do $ind_l = (hash(l) \mod hash.size) + 1$, $ind_l \in \mathbb{N}$, $hash(l) \in \mathbb{N}$
  1. define matrix $\boldsymbol{D}^{N \times hash.size}$ with $d_{ih} = 1$ if $ind_l = h$ and $d_{ih} = 0$ if $ind_l \neq h$, $x_i^{train} = l$
  2. $\boldsymbol{D} \to \tilde{\boldsymbol{D}}^{N \times V}$ with $V \leq hash.size$, by dropping constant columns in $\boldsymbol{D}$
  for all $x_i^{train} \in \boldsymbol{x}^{train}$ do $\hat{x}_{iv}^{train} = \tilde{d}_{iv}$
  Prediction:
  for $x^{new}$ do
      $ind^{new} = (hash(x^{new}) \mod hash.size) + 1$
      $\boldsymbol{d}^{new}$ of length $hash.size$ with $d_h^{new} = 1$ if $ind^{new} = h$ and $d_h^{new} = 1$ if $ind^{new} \neq h$
      $\boldsymbol{d}^{new} \to \tilde{\boldsymbol{d}}^{new}$ of length $V$, by dropping columns which were constant in $\boldsymbol{D}$
      for all $V$ columns in $\tilde{\boldsymbol{D}}$ do $\hat{x}_v^{new} = \tilde{d}_v^{new}$
```

Algorithm 13 Impact Encoding Regression

```
Training: require smoothing parameter $\epsilon \in \mathbb{R}$
for all $l \in \mathcal{L}^{train}$ do $\delta_l = \frac{\sum_{i:x_i^{train}=l} y_i^{train} + \epsilon \cdot \bar{y}^{train}}{N_l + \epsilon} - \bar{y}^{train}$ with $\bar{y}^{train} = \frac{\sum_{i=1}^{N} y_i^{train}}{N}$
for all $x_i^{train} \in \boldsymbol{x}^{train}$ do $\hat{x}_i^{train} = \delta_l$ with $x_i^{train} = l$
Prediction:
for $x^{new}$ do
    if $x^{new} \in \mathcal{L}^{train}$ then $\hat{x}^{new} = \delta_l$ with $x^{new} = l$ else $\hat{x}^{new} = 0$
```

Algorithm 14 Impact Encoding Classification

```
Training: require smoothing parameter $\epsilon \in \mathbb{R}$
for all $c \in \mathcal{C}$ do
    $p_c = \frac{N_c}{N}$, $p_c^{new} = \frac{N_c + \epsilon}{N + 2\epsilon}$, $logit_c = \log(\frac{p_c}{1 - p_c})$, $logit_c^{new} = \log(\frac{p_c^{new}}{1 - p_c^{new}})$
    $\delta_c^{new} = logit_c^{new} - logit_c$
    for all $l \in \mathcal{L}^{train}$ do
        $p_{lc} = \frac{\sum_{i:x_i^{train}=l} I(y_i^{train} = c) + \epsilon}{N_l + 2\epsilon}$, $logit_{lc} = \log(\frac{p_{lc}}{1 - p_{lc}})$
        $\delta_{lc} = logit_{lc} - logit_c$
for all $x_i^{train} \in \boldsymbol{x}^{train}$ do $\hat{x}_{ic}^{train} = \delta_{lc}$ with $x_i^{train} = l$
Prediction:
for $x^{new}$ do
    for all $c$ in $\mathcal{C}$ do
        if $x^{new} \in \mathcal{L}^{train}$ then $\hat{x}_c^{new} = \delta_{lc}$ with $x^{new} = l$ else $\hat{x}_c^{new} = \delta_c^{new}$
```

Additional Results

Table A1
Mean Encoding Ranks

Learner	Encoder	MRk	Pr25	Pr75	RMSE	AUC	AUNU
LASSO	glmm	2.17	1.00	3.00	2.62	2.00	1.83
LASSO	indicator	2.92	1.00	4.00	3.00	2.80	3.00
LASSO	rf	3.38	2.00	4.00	3.12	3.30	3.83
LASSO	cluster	4.96	3.00	6.00	4.25	5.60	4.83
LASSO	hash	5.71	4.00	7.00	6.25	5.00	6.17
LASSO	leaf	6.17	5.00	7.00	4.25	7.60	6.33
LASSO	impact	6.25	4.00	9.00	5.25	7.20	6.00
LASSO	frequency	7.38	6.00	8.00	8.25	6.70	7.33
LASSO	integer	7.67	6.00	9.00	8.62	6.90	7.67
LASSO	remove	8.42	7.00	10.00	9.38	7.90	8.00
RF	glmm	2.42	1.00	3.00	1.62	2.90	2.67
RF	rf	4.79	3.00	6.00	4.12	4.20	6.67
RF	cluster	4.96	3.00	6.00	6.00	4.30	4.67
RF	indicator	5.25	2.00	7.00	8.00	3.90	3.83
RF	hash	6.00	4.00	8.00	7.38	4.60	6.50
RF	frequency	6.12	4.00	8.00	6.88	5.10	6.83
RF	none	6.42	2.00	10.00	4.50	8.40	5.67
RF	impact	6.83	3.00	9.00	4.25	9.40	6.00
RF	leaf	7.21	5.00	9.00	6.38	8.30	6.50
RF	integer	7.38	6.00	9.00	7.62	7.00	7.67
RF	remove	8.62	6.00	11.00	9.25	7.90	9.00
GB	glmm	2.42	1.00	2.00	2.00	2.80	2.33
GB	rf	4.46	2.00	7.00	3.75	5.10	4.33
GB	leaf	4.88	3.00	6.00	6.25	4.40	3.83

GB	indicator	5.12	3.00	8.00	6.12	5.10	3.83
GB	cluster	5.25	3.00	7.00	5.25	5.80	4.33
GB	impact	5.88	3.00	9.00	4.62	7.00	5.67
GB	integer	6.21	5.00	7.00	5.25	6.80	6.50
GB	hash	6.38	5.00	8.00	7.00	5.30	7.33
GB	frequency	6.79	6.00	9.00	7.25	5.70	8.00
GB	remove	7.62	6.00	10.00	7.50	7.00	8.83
KNN	glmm	1.58	1.00	2.00	2.00	1.20	1.67
KNN	rf	3.12	2.00	3.00	3.50	3.10	2.67
KNN	impact	4.04	2.00	4.00	3.75	5.00	2.83
KNN	frequency	5.79	3.00	8.00	5.50	5.10	7.33
KNN	integer	5.92	4.00	8.00	5.25	4.90	8.50
KNN	hash	6.33	6.00	7.00	6.25	6.10	6.83
KNN	leaf	6.58	4.00	9.00	6.12	7.50	5.67
KNN	indicator	6.67	5.00	9.00	7.50	6.80	5.33
KNN	cluster	6.79	6.00	8.00	6.75	7.60	5.50
KNN	remove	8.17	7.00	10.00	8.38	7.70	8.67
SVM	glmm	2.56	1.00	3.00	3.50	2.67	1.50
SVM	rf	2.94	1.00	3.00	3.17	3.00	2.67
SVM	impact	4.39	3.00	5.00	3.33	6.67	3.17
SVM	indicator	5.50	3.00	7.00	4.50	6.50	5.50
SVM	leaf	5.61	4.00	7.00	7.00	4.50	5.33
SVM	frequency	5.83	4.00	8.00	5.50	4.33	7.67
SVM	remove	6.39	4.00	10.00	7.00	4.50	7.67
SVM	cluster	6.61	5.00	8.00	6.00	7.50	6.33
SVM	integer	7.33	5.00	9.00	7.17	7.33	7.50
SVM	hash	7.83	6.00	10.00	7.83	8.00	7.67

Note. Ranks are based on the best performing hyperparameter settings. Low ranks indicate better performance. Datasets producing errors were removed for the SVM. MRk = mean ranks across all datasets based on the primary performance measure, Pr25 = 0.25 quantile of ranks, Pr75 = 0.75 quantile of ranks, RMSE = mean ranks for regression, AUC = mean ranks for binary classification, AUNU = mean ranks for multiclass classification.

Table A2
Correlations of Encoder Ranks with Dataset Characteristics

Encoder	Learner	Obs	NA%	CorMaxLvl
glmm	LASSO	-0.02	-0.12	-0.54
glmm	KNN	-0.07	-0.38	-0.48
glmm	SVM	-0.05	-0.44	-0.39
glmm	RF	0.00	-0.30	-0.08
glmm	GB	0.05	-0.07	-0.43
rf	LASSO	0.10	-0.09	0.30
rf	KNN	-0.35	-0.26	-0.21
rf	SVM	-0.19	0.40	0.07
rf	RF	-0.22	-0.13	0.12
rf	GB	0.02	0.22	-0.05
integer	LASSO	0.17	-0.09	0.10
integer	KNN	-0.24	0.17	-0.10
integer	SVM	-0.14	-0.01	0.18
integer	RF	-0.39	0.01	-0.28
integer	GB	-0.39	0.00	-0.19
frequency	LASSO	-0.03	-0.25	-0.24
frequency	KNN	-0.09	-0.15	-0.42

frequency	SVM	-0.35	-0.21	0.16
frequency	RF	-0.14	-0.06	-0.37
frequency	GB	-0.50	-0.23	-0.66
leaf	LASSO	-0.02	0.26	0.06
leaf	KNN	0.47	-0.09	0.15
leaf	SVM	0.14	-0.42	-0.35
leaf	RF	0.23	-0.10	-0.03
leaf	GB	0.41	-0.39	0.08
impact	LASSO	0.09	0.19	0.02
impact	KNN	0.06	0.27	0.29
impact	SVM	-0.02	0.49	0.23
impact	RF	0.10	0.34	0.08
impact	GB	0.18	0.21	0.09
remove	LASSO	0.07	-0.32	0.10
remove	KNN	-0.49	-0.01	0.10
remove	SVM	-0.08	-0.15	0.02
remove	RF	-0.15	-0.05	0.12
remove	GB	-0.32	-0.05	0.16
indicator	LASSO	-0.05	0.25	0.20
indicator	KNN	0.25	-0.05	0.13
indicator	SVM	0.20	0.24	-0.01
indicator	RF	0.00	-0.19	0.06
indicator	GB	0.14	-0.06	0.23
hash	LASSO	-0.17	0.00	-0.11
hash	KNN	-0.07	-0.02	0.09
hash	SVM	0.25	-0.11	0.02
hash	RF	-0.30	-0.27	-0.63
hash	GB	-0.05	0.13	0.21

cluster	LASSO	-0.10	-0.19	-0.25
cluster	KNN	0.33	0.08	0.11
cluster	SVM	0.28	0.18	-0.14
cluster	RF	-0.04	-0.29	-0.25
cluster	GB	0.29	-0.03	0.36
none	RF	0.36	0.46	0.58

Note. Spearman correlations between encoder rankings with dataset characteristics for each machine learning algorithm. Datasets producing errors were removed for the SVM. Obs = number of observations, NA% = percentage of missing values, MaxLvl = maximum number of levels per categorical variable.

Table A3
Win Percentages of One-hot vs. Dummy Encoding

Learner	BinCl	MultCl	Regr
LASSO	80	94	75
RF	83	71	71
GB	63	71	54
KNN	67	59	50
SVM	68	82	52

Note. Percentage of indicator encoding conditions in which one-hot performs better than dummy per task setting.

Table A4
Mean Runtimes

Encoder	LASSO	RF	GB	KNN	SVM
integer	10	39	30	26	41
frequency	9	35	28	24	42
indicator	27	69	77	49	45
hash	32	100	89	37	43
leaf	14	89	85	36	38
cluster	25	89	85	50	46
impact	12	34	24	29	41
glmm	25	50	29	34	50
rf	14	58	29	43	41
none		18			
remove	8	32	20	25	42

Note. Mean runtime in minutes of 5-CV for each combination of encoder x machine learning algorithm, averaged across datasets. Datasets producing errors were removed for the SVM.