Bayesian target encoding Documentation

Akshay Gupta

Nov 24, 2022

QUICKSTART

1 Installation **3**

2 Contents **5**

2.1 When should you use this package? 5

2.2 Encode your categorical variables 9

2.3 Build a model through repeated encoding 10

2.4 Encoding algorithm 11

`bayte` offers a lightweight, `scikit-learn`-compliant[1] implementation of Bayesian Target Encoding. The algorithm was introduced in 2019 by Slakey *et al.*[2], with ensemble modeling methodology from Larionov[3]. Our explanation of the algorithm is available *here*.

[1] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 108–122. 2013.

[2] Austin Slakey, Daniel Salas, and Yoni Schamroth. Encoding categorical variables with conjugate bayesian models for wework lead scoring engine. *CoRR*, 2019. URL: http://arxiv.org/abs/1904.13001, arXiv:1904.13001.

[3] Michael Larionov. Sampling techniques in bayesian target encoding. *CoRR*, 2020. URL: https://arxiv.org/abs/2006.01317, arXiv:2006.01317.

CHAPTER

ONE

INSTALLATION

To install `bayte` from PyPI, run

```
$ python -m pip install bayte
```

This is the preferred method to install `bayte`.

CHAPTER

TWO

CONTENTS

2.1 When should you use this package?

We have leveraged the experimental framework discussed by Pargent *et al.*[1] to analyze bayesian target encoding (BTE) and answer the following questions:

- **Marginal BTE**: Is there lift from a staged approach:
  1. Fit a submodel*[0] that uses all non-categorical columns to predict the target.
  2. Fit the encoder using the submodel output as the target.
  3. Use the encoding and the raw input non-categorical data to fit the final model.

2.1.1 Encoder comparison

In this experiment, we wanted to compare the standard bayesian target encoding to other popular encoding methodologies. We also wanted to test a "staged approach", where we

1. fit a submodel that uses all non-categorical columns as features,
2. fit the encoder using either the submodel output (the "marginal" approach) or the residuals as the target, and
3. use the encoding and raw numeric features to fit the final model.

The idea here is that the categorical encoding can try to use the information not captured by numeric variables and produce a more useful encoding.

The aggregated visualization doesn't show this well, but we have three takeaways from this experiment:

1. Non-sampled bayesian target encoding does not outperform other encoding methods,
2. Sample bayesian target encoding performs the best, and
3. marginal/residual encoding provides very incremental benefit at best.

[1] Florian Pargent, Florian Pfisterer, Janek Thomas, and Bernd Bischl. Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. 2021. arXiv:2104.00629.

[0] What if the encoder is fitted using the residuals from the submodel as the target?

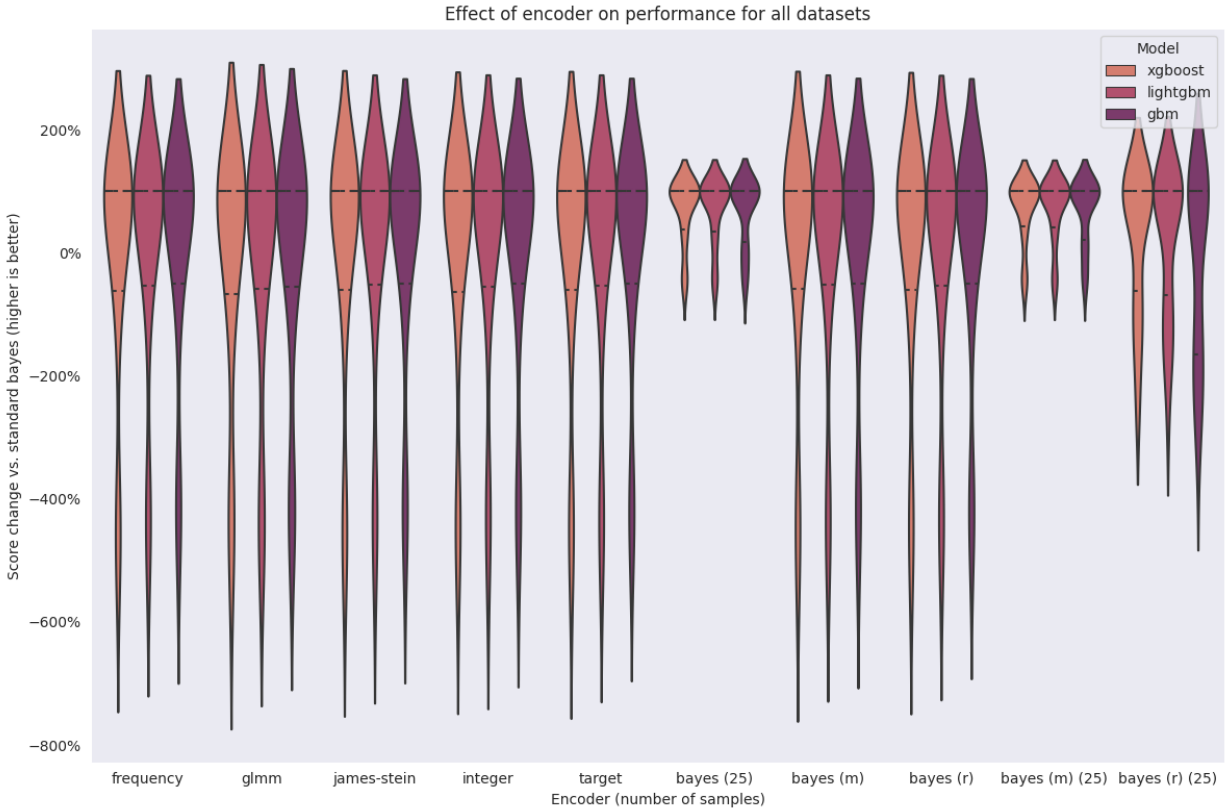
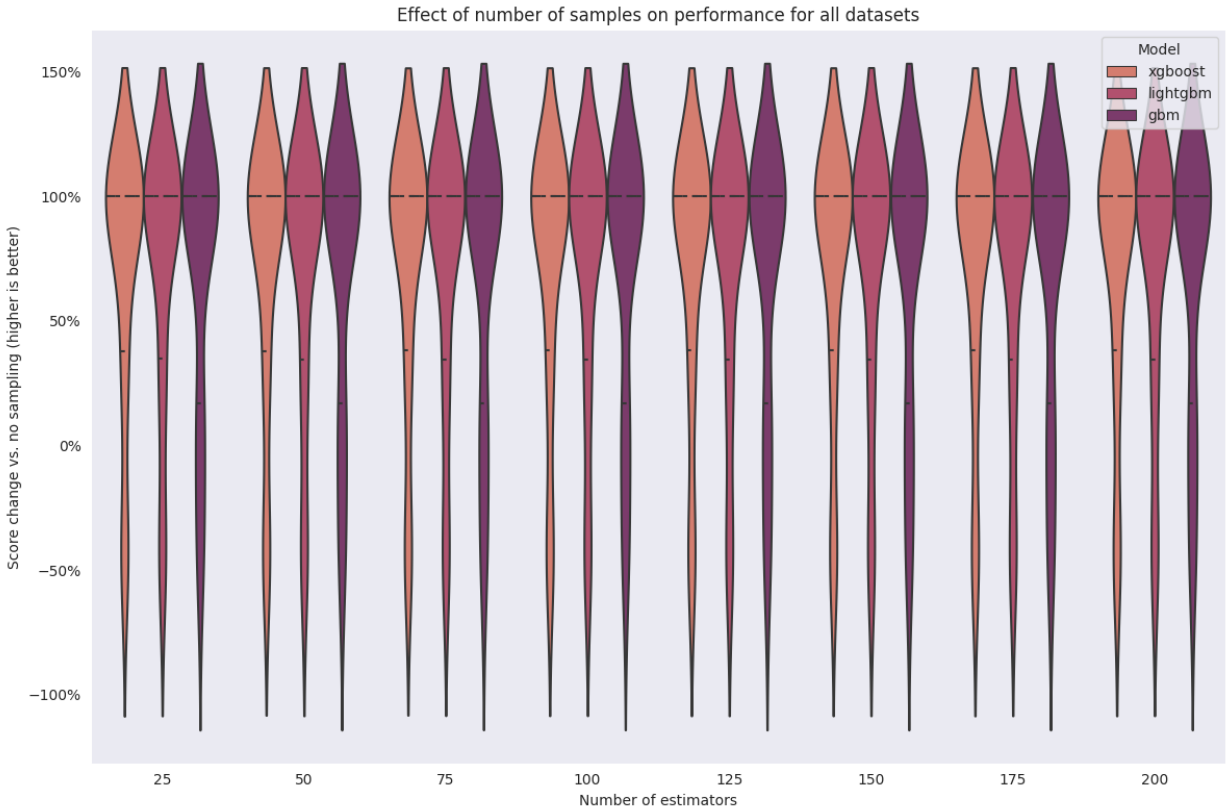


Fig. 1: `(m)` indicates that the encoder used the "marginal" approach. `(r)` indicates that the encoder uses the "residual" approach.

2.1.2 Ensemble methodology[2]



We wanted to answer the following questions:

- How much does repeated sampling help?
- How many samples do you need?

The short answer is that repeated sampling will *almost definitely help* with test performance. Only two datasets, churn and flight-delay-usa-dec-2017, saw decreases in test performance.

Surprisingly, only 25 samples are required to see an increase in performance.

2.1.3 Comparative encoding methodology

When conducting these experiments, we'll compare BTE to the following encoding methodologies. Suppose you have n training observations, with $Y = (y_1, ..., y_n)$ representing the target and categorical variable $X_1 = (x_1, ..., x_n)$ with distinct values $V = (v_1, ..., v_l)$.

Pargent *et al.*[Page 5, 1] provide a description for each encoding methodology listed below.

Encoding	Supervised?	Implementation
Frequency	N	`category_encoders.CountEncoder`
Generalized Linear Mixed Model	Y	`category_encoders.GLMMEncoder`
James-Stein	Y	`category_encoders.JamesSteinEncoder`
Integer	N	`sklearn.preprocessing.OrdinalEncoder`
Target	Y	`category_encoders.TargetEncoder`

[2] Michael Larionov. Sampling techniques in bayesian target encoding. *CoRR*, 2020. URL: https://arxiv.org/abs/2006.01317, arXiv:2006.01317.

2.1.4 Modeling algorithms

The following modelling implementations will be tested:

Package	Class
LightGBM[3]	`LGBMClassifier`
	`LGBMRegressor`
Scikit-Learn[4]	`GradientBoostingRegressor`
	`GradientBoostingClassifier`
XGBoost[5]	`XGBClassifier`
	`XGBRegressor`

2.1.5 Datasets

Regression

Below is a list of the regression datasets used for experimentation[Page 5, 1].

OpenML ID	Dataset name
41211	ames-housing
41445	employee_salaries
41210	avocado-sales
41267	particulate-matter-ukair-2017
41251	flight-delay-usa-dec-2017

Classification

OpenML ID	Dataset name
40701	churn
41434	click_prediction_small

2.1.6 Performance evaluation

Pargent *et al.*[Page 5, 1] discussed a three-phase approach for creating a baseline assessment of model performance. We'll adapt that here and use something slightly different. **Baseline performance** will be the average test score for a model fitted using the *standard* bayesian target encoder. We will repeat each experiment with 5 different random seeds for the train-test split.

Similar to Pargent *et al.*[Page 5, 1], we will use root mean squared error (RMSE) for evaluating the performance of regression models and the area under the receiver operating characteristic (AUROC) for classification problems. Both metrics are available in `scikit-learn`[4] under the strings `neg_root_mean_squared_error` and `roc_auc`, respectively.

[3] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: a highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] Tianqi Chen and Carlos Guestrin. XGBoost: a scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, 785–794. New York, NY, USA, 2016. ACM. URL: http://doi.acm.org/10.1145/2939672.2939785, doi:10.1145/2939672.2939785.

Since we will *not* be doing any hyperparameter optimization, we will express the change in performance using a percentage increase in the stated metric.

2.2 Encode your categorical variables

To encode your variables, you have to first choose a likelihood for your target.

Model type	Description	Likelihood
Classification	Binary	`bernoulli`
	Multi-class	`multinomial`
Regression		`normal`
		`exponential`
		`gamma`
		`invgamma`

Important: The normal likelihood assumes a *known* variance that is estimated from the training data. Similarly, the gamma and inverse gamma likelihoods assume a known shape parameter. Both of these assumptions were made to help make implementing the algorithm easier.

2.2.1 Basic usage

Once you've chosen your likelihood, import and fit the encoder on your data. Suppose you have `X` and `y`, with three categorical columns: 1, 2, and 5.

```
import bayte as bt

encoder = bt.BayesianTargetEncoder(dist=...)
encoder.fit(X[:, [1, 2, 5]], y)
```

By default, when you transform the data

```
X_encoded = encoder.transform(X[:, [1, 2, 5]])
```

the encoding level will be the mean of the posterior distribution for the level. To sample, set `sample=True` on encoder initialization.

Important: The encoder has support for joblib. Since the encoding procedure involves generating posterior parameters for every categorical level in every supplied variable, it can be computationally inefficient if executed serially.

2.2.2 Changing hyperparameter initialization

If you want to change how the hyperparameters are initialized for a given likelihood, supply a callable for the `initializer` argument. This callable must take the `dist` and the target values `y` and return a tuple of the parameters.

Important: Although you can change the initializer, your code will break if you try to implement a new likelihood.

2.3 Build a model through repeated encoding

In this guide, we will build an ensemble estimator using bayesian target encoding. First, initialize the `bayte` class for your modelling problem your base estimator as well as an initialized encoder and the number of samples you want to draw.

Classification

Regression

In this classification example, we will fit a logistic regression.

```
from sklearn.linear_model import LogisticRegression

import bayte as bt

estimator = bt.BayesianTargetClassifier(
    base_estimator=LogisticRegression(),
    encoder=bt.BayesianTargetEncoder(dist="bernoulli"),
    n_estimators=10,
)
```

In this regression example, we will fit a simple linear model.

```
from sklearn.linear_model import LinearRegression

import bayte as bt

estimator = bt.BayesianTargetRegressor(
    base_estimator=LinearRegression(),
    encoder=bt.BayesianTargetEncoder(dist="gamma"),
    n_estimators=10,
)
```

Next, call `fit`. The `fit` call accepts a `categorical_feature` parameter for specifying which features in the training dataset should be encoded. If you are using a `numpy.ndarray`, specify the indices of the categorical columns. If you are using a `pandas.DataFrame`, specify the column names. If not set, any pandas columns with a categorical data type will be encoded; for numpy users, no supplied list indicates that *all* features are categorical.

```
estimator.fit(X, y, categorical_feature=[1, 2, 5])
```

For regression problems, `predict` will produce an average of each estimator prediction. For classification, `predict` depends on an estimator initialization parameter called `voting`[1]:

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas,

`voting` value	Description
`hard` (default)	The predicted class label is a majority vote of the predicted labels from each estimator.
`soft`	The predicted class label is based on the sums of predicted probabilities for each class.

For well-calibrated classifiers, `soft` voting is preferred.

2.4 Encoding algorithm

2.4.1 Refresher on bayesian statistics

In bayesian statstics, we have

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}$$

where $p(\theta)$ is the *prior distribution* for parameter θ, $p(y|\theta)$ is the *likelihood* of y given θ, and $p(\theta|y)$ is the *posterior distribution* of parameter θ using y. In particular, we will focus on *conjugate Bayesian models*, where the prior distribution and posterior distribution of θ are from the same family.

Example

Consider a situation where the target variable in our dataset is binary. This means that $y_1, ..., y_n$ are independent and identically distributed from a Bernoulli process where θ, the probability of a 1, is **unknown**.

Using Fink's Compendium of conjugate priors, the prior distribution of θ is a Beta distribution with **hyperparameters** α and β. i.e., $\theta \sim Beta(\alpha, \beta)$

Since we are using a conjugate Bayesian model, the posterior distribution $p(\theta|y)$ follows a $Beta(\alpha', \beta')$. Fink stipulates that

$$\alpha' = \alpha + \sum_{i=1}^{n} y_i$$

and

$$\beta' = \beta + n - \sum_{i=1}^{n} y_i$$

A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

2.4.2 Procedure

Ok, let's lay out the procedure for bayesian target encoding. Suppose you have n training observations, with $Y = (y_1, ..., y_n)$ representing the target and categorical variable $X_1 = (x_1, ..., x_n)$ with distinct values $V = (v_1, ..., v_l)$.

1. Choose a likelihood for the target variable (e.g. Bernoulli for binary classification),
2. Derive the conjugate prior for the likelihood (e.g. Beta),
3. Use the training data to initialize the hyperparameters for the prior distribution (e.g. α and β)[1],
4. Derive the methodology for generating the posterior distribution parameters,
5. For each level $v_i \in V$,
    1. Generate the posterior distribution using $y_1, ..., y_m | x_j = v_i, \forall j \in (1, m)$,
    2. Set the encoding value to a sample from the posterior distribution[2]

[1] Initializing the hyperparameters is generally reliant on common interpretations.
[2] If a new level has appeared in the dataset, the encoding will be sampled from the prior distribution.