

This member-only story is on us. [Upgrade](#) to access all of Medium.

★ Member-only story

# How CatBoost encodes categorical variables?

One of the key ingredients of CatBoost explained from the ground up



Adrien Biarnes · [Follow](#)

Published in Towards Data Science · 17 min read · Feb 10, 2021



206



2



Photo by [Chandler Cruttenden](#) on [Unsplash](#)

CatBoost is a “relatively” new package developed by Yandex researchers. It is pretty popular right now, especially in Kaggle competitions where it generally outperforms other gradient tree boosting libraries.

Among other ingredients, one of the very cool feature of CatBoost is that it handles categorical variables out of the box (hence the name of the algorithm).

When using an implementation, it is important to really understand how it works under the hood. This is the goal of this article. We are going to take an in-depth look at this technique called **Ordered Target Statistics**. It is presumed that you have a good understanding of gradient tree boosting. If not, check out [my article](#) on the subject.

## Introduction

Gradient Tree Boosting is one of the best, of the shelf, family of algorithms to deal with structured data. This family of algorithms originated from the boosting method which is one of the most powerful learning ideas introduced in the last 25 years. AdaBoost introduced by Yoav Freund and Robert Schapire in 1997 was the first instance of this method. Then later that year Leo Breiman realized that boosting could be reformulated as an optimization problem with a proper cost function which gave birth to gradient boosting by Jerome Friedman in 2001. Since then, boosting methods with trees have seen a lot of newcomers, namely and in order of appearance, XGBoost — eXtreme Gradient Boosting (2014), lightGBM (2016), and CatBoost (2017).

*Gradient Boosting can also be used in combination with neural networks these days, either to integrate structured additional knowledge or just to get a boost in performance. In this regard, a technique introduced by the authors of GrowNet where the weak learners are replaced by shallow neural networks seems very promising.*

## Categorical variable encoding

*A word a caution though, I use the terms category, level, or factor interchangeably (in this article they mean the exact same thing)*

As you probably already know, machine learning algorithms only handle numerical features. So let's say that you are trying to predict the sailing price of a second-hand car. Among the available features, you have the brand of the car (Toyota, Honda, Ford...). Now in order to use this feature (which will definitely have an impact on the sailing price) in your favorite gradient

boosting algorithm, you need to convert the brand to a numerical value because the algorithm needs to sort the brands in order to enumerate the split possibilities and find the best one.

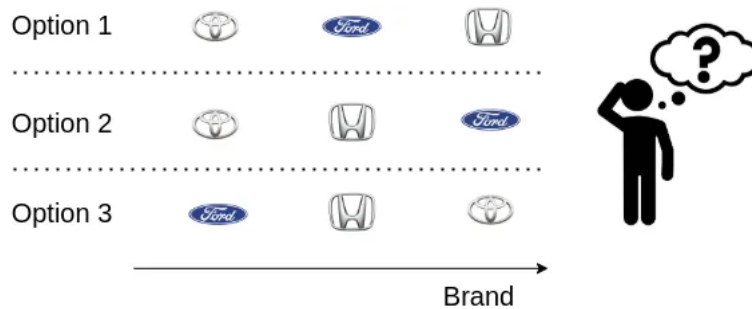


Image by Author

Ok... but this sounds more like a technical justification of the internal constraints of the decision tree mechanism. If we really think about it, what is it that we are trying to do when transforming the categorical variable into a numeric one? Well, we can try to confront, on the same scale, the categorical value with the target, to visualize the problem:

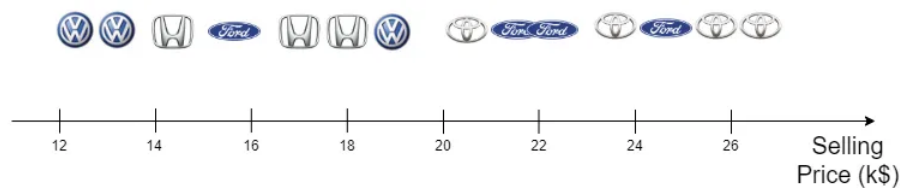


Image by author

When visualizing things like this, it becomes more apparent that our objective is to **reveal the ordering strength that appears among categories when predicting the target**. Ideally, we would like to reflect the ordering but also our uncertainty in the measure.

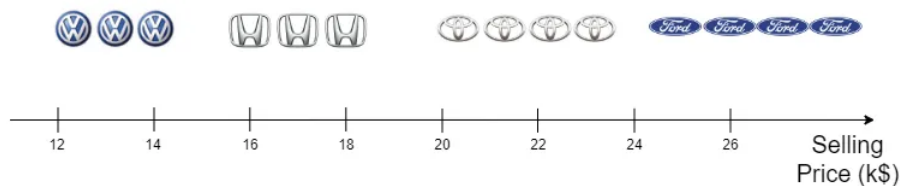
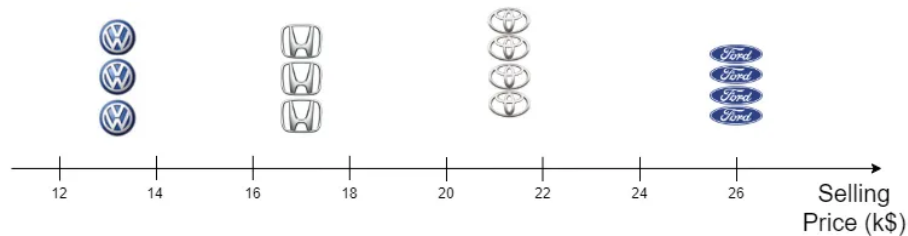


Image by author

In this second example, the brands order with respect to the selling price appears very clearly. But this ordering can be even stronger like in the below third example:



In order to introduce the technique used in CatBoost, we will first go through a couple of categorical encoding methods that are frequently used in conjunction with decision trees.

### One-hot encoding

Brand	
Toyota	
Ford	
Toyota	
Honda	

→

Brand_Toyota	Brand_Ford
1	0
0	1
1	0
0	0

Image by author

The idea is pretty simple. You replace the categorical feature with dummy variables (one for every distinct category). A dummy variable is a variable that indicates if a specific factor is present (value 1) or absent (value 0). You can also decide not to include a dummy variable for one of the factors.

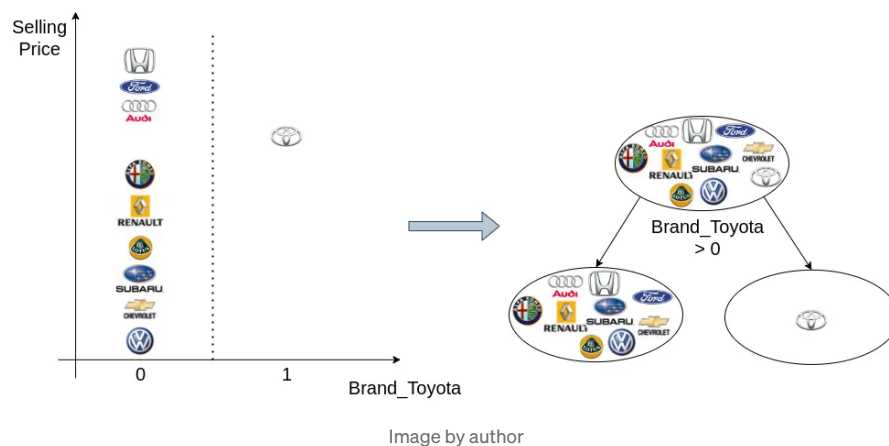
*This last point is very much recommended to avoid multicollinearity issues and mandatory in the case of linear regression where the OLS estimator will not exist otherwise (because of the perfect collinearity, the input matrix will not be of full rank and the Gram matrix won't be invertible).*

When used in conjunction with a linear model, the good thing about this technique is that you get one coefficient for every level. Every coefficient can then be interpreted as the effect of the associated category in comparison to the reference level (the one you did not include a dummy variable for).

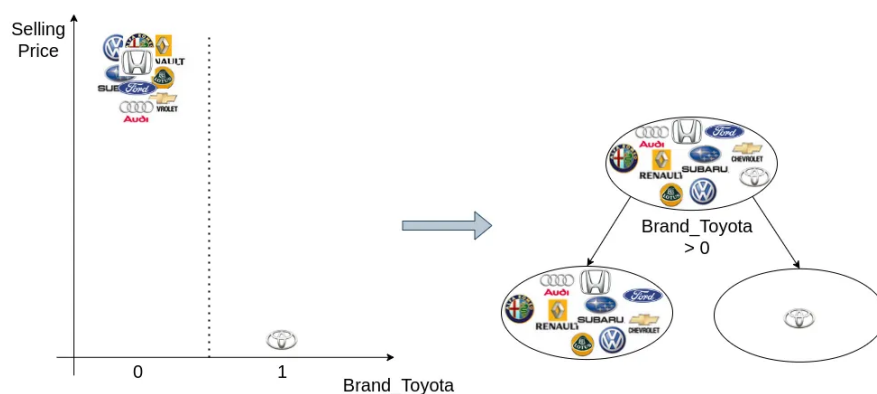
But the major problem is that you get one extra variable for every factor. If you have a lot of different categories you will suffer from high dimensionality issues. In such a case, when used in conjunction with a decision tree, or an ensemble of decision trees, because your dummy variables will mostly contain zeros, they will be highly unbalanced. When selecting a feature for a split, this highers the chance that the dummy variables will be ignored in favor of the other predictors. This happens because the associated information gain (entropy reduction in case of

classification and target variance reduction in case of regression), will probably be smaller when selecting this feature for a split. Of course, this statement is not always true because the performance of the split will highly depend on the distribution of the target with respect to the category of interest.

Let's visualize this idea with a simplified example. We have 10 observations (10 different cars) for which we are trying to predict selling price (in k€). In this first case, each car has a different brand and the selling prices are homogeneously distributed. What happens if we select one of the dummy variables for a split:



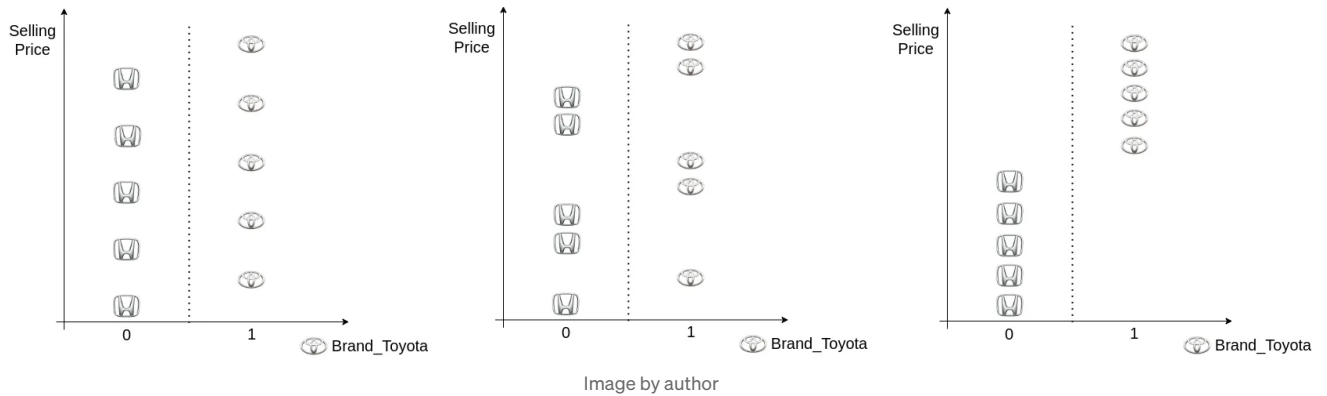
In this example, the target variance in the left leaf is very high so the reduction in variance that this split exhibit is pretty lame (hence this is a bad split). Of course, the distribution of the target with respect to the Toyota brand can be totally different like so:



In this case, the split is better **but** if it was not for the Toyota observation, the target variance would **already be very low**. This means that apart from predicting correctly that one outlier, the predictive power of the categorical feature is pretty lame. All in all, **we are much more interested in an encoding scheme that exhibits a strong predictive power when the feature**

is meaningfully correlated with the target.

Now let's see what happens when we go down to only 2 brands (sticking with the case where the selling prices are homogeneously distributed). We display 3 different scenarios from one extreme to the other:



Top highlight

All three scenarios lead to the same balanced tree, but they are not all equivalent in terms of target variance reduction. The scenario on the left proposes the lowest variance reduction and the one on the right the highest. So we see from that example, that by reducing the number of possible categories, we can now imagine situations where the target variance reduction is important.

In the end, what is to remember is that in the case of a high number of levels, using one-hot encoding might not be the best transformation to use because it highers the chance of diminishing the importance of the categorical variable in the resulting model. On the contrary, if you have a small number of levels, this technique can still be pretty efficient. This is why in CatBoost, there is a hyperparameter ([one\\_hot\\_max\\_size](#)) to configure the number of categories under which one-hot encoding is used instead of the main technique we describe later in this post.

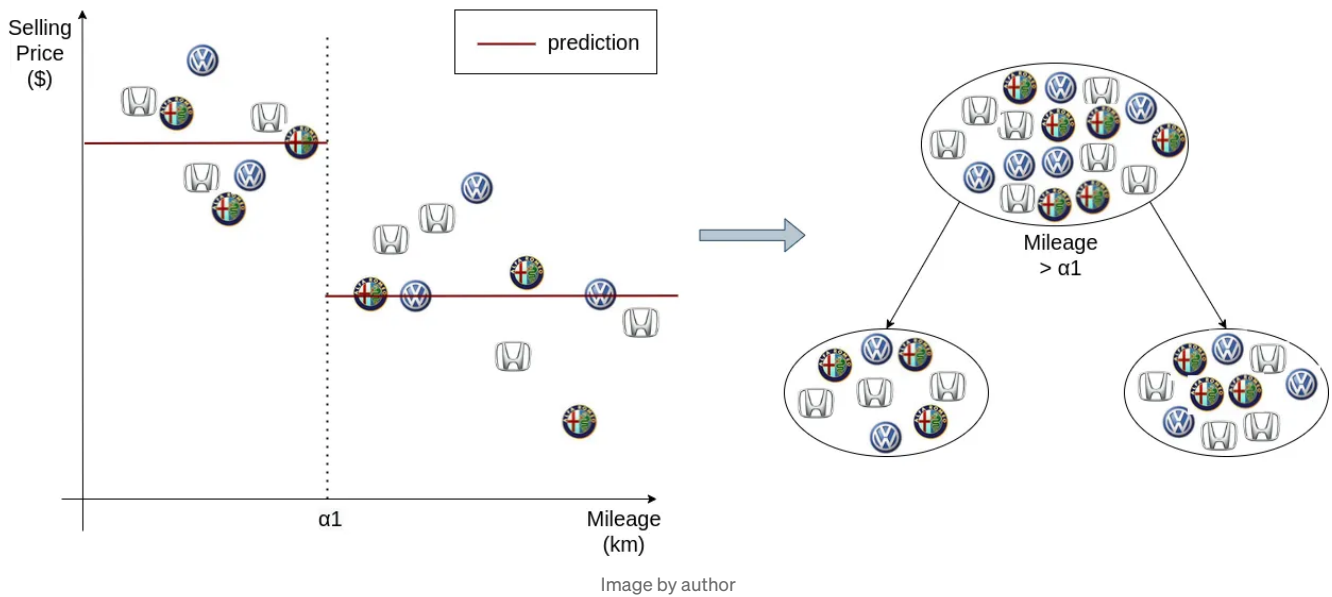
### Gradient statistics

This technique was introduced by lightGBM. The basic idea is to sort the categories according to the training objective at each split relying on the gradients of the cost function. To properly understand this technique, one needs to understand the mechanisms of GradientBoosting. If you feel like you need a refresher you can take a look at [my article](#) on the matter.

The first thing is that instead of encoding the categories once and for all prior to training, they are encoded during training. At each split decision for a node, categories are ordered by the amount of error (using the squared gradients as a proxy) they weigh in this node (in regard to the target).

*Please not that we consider the residuals ( $y - \hat{y}$ ) as being equal to the gradients. This is only the case when performing regression with the squared error loss. If another loss is used, we can't make the same assumption (we rely on the pseudo-residuals) but the idea is exactly the same.*

So let's have a visual explanation of what is going on. Let's say that we are building a weak learner (a regression tree) and have already chosen the numerical feature "Mileage" for the first split:

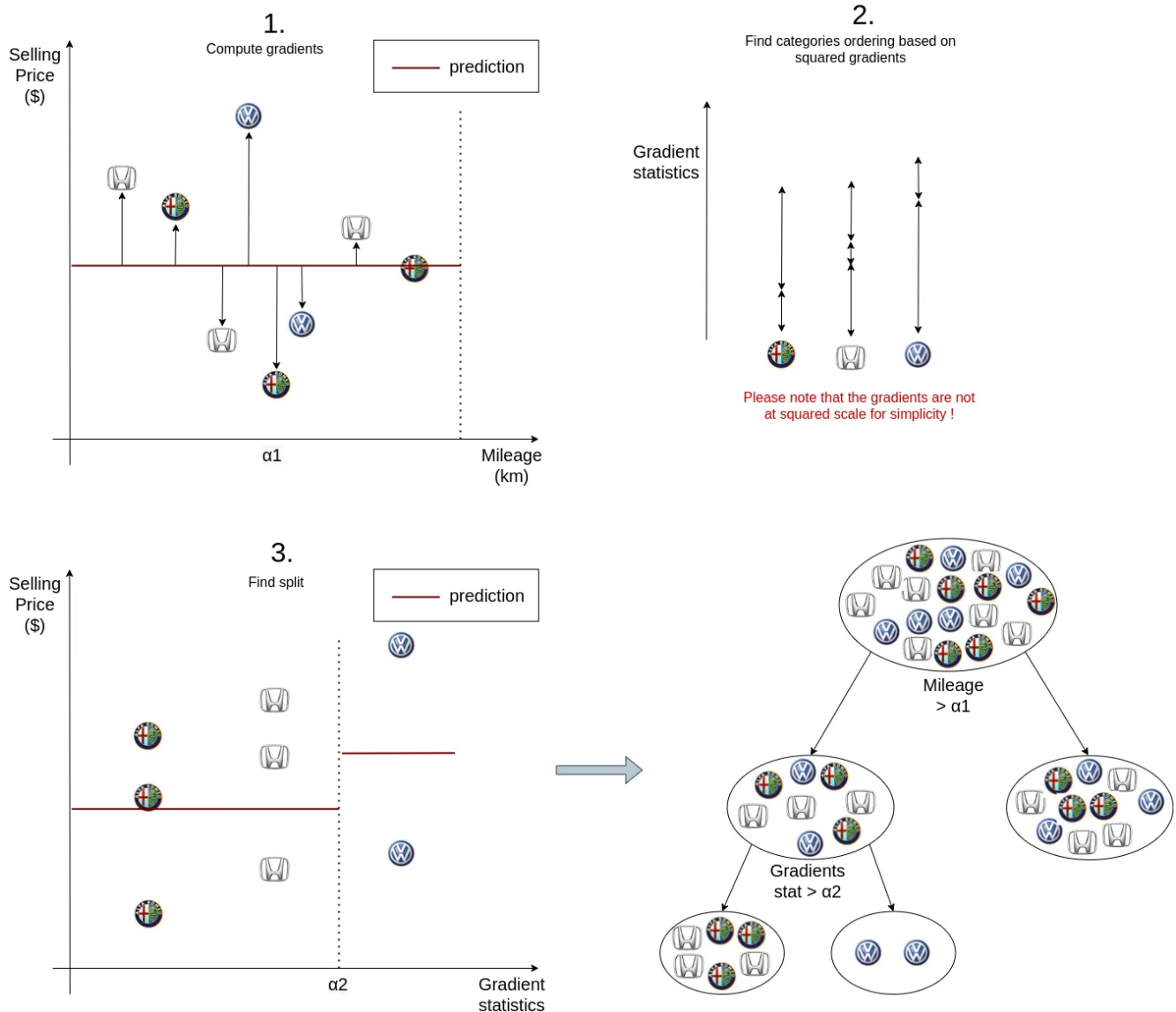


Now we want to build the second level starting with the left node of the tree. It appears that, for the sake of this example, the brand categorical feature proposes the best split. Let's see how we came up with that conclusion. The following operations took place:

1. We computed the current gradients for every observation of this node
2. We summed up the squared gradients by category and ordered the categories using this statistic
3. Evaluated the split possibilities in regard to gradient statistics. One of the split claimed the best target variance reduction overall so it was chosen

Let's zoom in on the left part to see what happened:





So overall this technique is pretty nice because it orders the categories in regard to their prediction objective at each split. But it suffers from a few flaws, quoting the authors of CatBoost from [their paper](#), it increases:

- (i) computation time, since it calculates statistics for each categorical value at each step
- (ii) memory consumption to store which category belongs to which node for each split based on a categorical feature

And this problem gets worse with the number of different categories. To try to overcome this, in lightGBM, they group tail categories into one cluster but therefore lose part of the information. Besides, the authors claim that it is still better to convert categorical features with high cardinality to numerical features prior to modeling.

*Please note that this explanation was simplified in comparison to what is really happening in lightGBM. Indeed they use a technique called second-order loss approximation (introduced by XGBoost) to compute the loss and the best split. Therefore they don't exactly use the gradients (first-order) to order categorical values but rather a ratio between first-order and second-order derivatives of the loss. If you want more information on this particular point, take a look at [this nice article](#) on the matter.*

### Target statistics

As I previously said, one of the most important traits that we want out of the encoding method is that it should reflect the ordering of the categories with respect to the target. So one simple idea is to substitute the values of a categorical feature  $x$  with either:

- the expected value of the target conditioned on the categories (in a regression setting)
- the expected probability of the positive class conditioned on the categories (in a binary classification setting)
- $m-1$  new variables for an  $m$ -classes classification task. Each new variable will be constructed as if it was a binary classification task.

In our example (regression), this would give:

$$\begin{aligned} \text{Alfa Romeo} &\Rightarrow \mathbb{E}[\text{Selling Price} \mid x = \text{Alfa Romeo}] \\ \text{Honda} &\Rightarrow \mathbb{E}[\text{Selling Price} \mid x = \text{Honda}] \\ \text{VW} &\Rightarrow \mathbb{E}[\text{Selling Price} \mid x = \text{VW}] \end{aligned}$$

Image by author

So let's say we have the following observations:

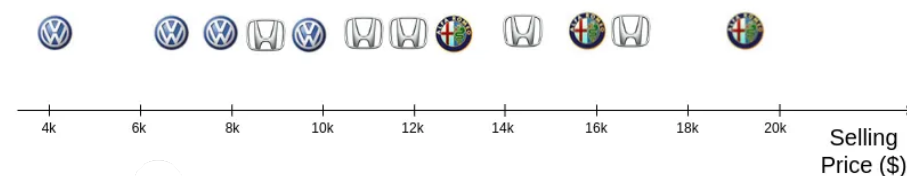


Image by author

The encoding will produce the following computations:

$$\text{VW} \Rightarrow \frac{1}{4}(4.1 + 6.8 + 7.8 + 9.8) \approx 7.125$$

$$\text{Honda} \Rightarrow \frac{1}{5}(8.8 + 10.9 + 11.9 + 14.4 + 16.8) \approx 12.56$$

$$\text{Other} \Rightarrow \frac{1}{3}(12.9 + 15.9 + 19.4) \approx 16.06$$

Image by author

Which in turn will give the following result:

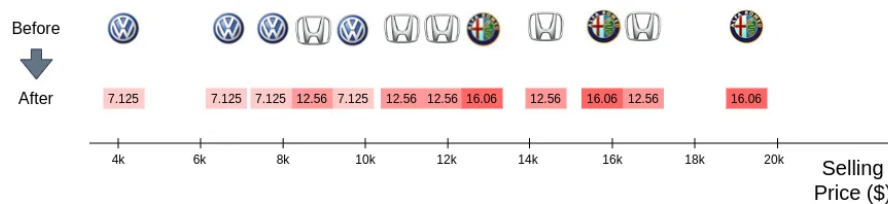


Image by author

Also, instead of using the empirical expected value (the mean), one could use other statistics like the median, the mode, the variance, or higher moments of the underlying distribution which would give different properties to the encoding.

This method is statistically sound and preserves most of the predictive power of the categorical variable. But it is not perfect. One common issue is that the estimate is not sufficiently robust for categories with low cardinality. To remedy this issue we generally add a weighted prior. The category  $k$  of a specific categorical variable  $x$  is then replaced with the following value:

$p$  is the prior for which we can use the overall target average for example.  $\alpha$  is a hyperparameter that tunes the strength of the prior. We talk about a prior because this way of smoothing the result could be considered as the frequentist equivalent of the bayesian approach to target encoding. We introduce a prior belief about the statistic of interest and we inject a specific amount of that prior.  $\alpha$  can also be a function parameterized by the cardinality of the category and the dataset size so that you use more or less of the prior depending on whether that category is more or less represented.

Please note that we compute the statistics on the training data and use them equally on the test set. If a category from the test set was absent in the training set we use the statistic (mean, median, mode, whatever) computed over the full training set (without conditioning).

But this technique suffers from a condition known as **target leakage**. Solely using the target to encode the variable necessarily leads to a situation where the learned relationship overfitted on the idiosyncrasies of the training set. The distribution of the categorical variable conditioned on the target value ( $x|y$ ) computed using the training set will not necessarily be the same if computed using the test set. This leads to a **prediction shift**. To illustrate this issue, we are going to use an extreme example borrowed from the [CatBoost paper](#).

So let's say that we are trying to solve a binary classification problem (is this car going to be sold or not in the next month?). We only have one predictor which is the brand of the car. To ensure that there is absolutely **no relationship** between the target and the feature, we assign the label (0 or 1) **at random** using Bernoulli trials with probability  $p=0.5$ . Also, we ensure that there is one unique brand for every observation. Clearly, there is absolutely no way that one can come up with a prediction model better than random guessing. Let's see what happens with say 10 observations:

Image by author

To simplify our example, we did not use the prior in the encoding scheme. The encoded feature either gets the value 0 or 1 depending on the associated target. But what happened? Well, we got a perfect split. The reduction in impurity (entropy) was maximal because all observations in the leaves have the same target label. So prediction accuracy is 1.0 (we always predicted the right label) on the training set. But as soon as we switch to the test set, given the fact that target labels are assigned randomly, the error rate will be much higher and probably equal to random guessing.

Of course, this example is extreme but it highlights the fact that by using the target of the observation for which we are performing the encoding we leak

prediction information into the feature. And that information might not be relevant for other similar examples. And the problem is worse for low cardinality categories.

### **K-fold target encoding**

This technique introduced to mitigate the feature leakage problem find its roots in the well-known cross-validation procedure (that was invented to fight against overfitting). So we divide the dataset into K folds. We iterate through every fold and compute the target statistics for the current fold using the data from the K-1 remaining fold. A special case of this technique is called Holdout target encoding when K=2. The drawbacks of this technique are that it can significantly increase training time and we are not using the full dataset to compute the statistic which increases bias.

### **Leave-one-out target encoding**

This time we push the cross-validation scheme one step further trying to use as much data as possible. The technique could also be called N-fold target encoding (with N the number of observations). For every observation, we compute the target statistic by only removing the current observation from the training data. This technique is better than K-fold target encoding because it almost uses the full dataset and therefore leads to more accurate statistics at the price of an increased computational budget.

One could think that this technique prevents target leakage. But in fact, it does not. We will demonstrate it with a simple example. So let's say that we are back to our binary classification problem with 10 observations. This time we have 7 positive observations and 3 negatives but most importantly the categorical variable only has one unique brand. We know that such a feature is useless because it doesn't have any predictive power. The problem here is that when we compute the statistics we only have two different cases, either we are leaving a 1 out or a 0 out. So the only two possible values for the encoded features are:  $(7-1)/10$  when we leave a 1 out and  $(7-0)/10$  when we leave a 0 out. We get the following result:

Image by author

Once again we got a perfect split meaning that we perfectly classified the training examples. But we know that this feature is useless because exactly the same for every observation. Using this tree on the test data will be a failure. So with this simple example, we show that the target leakage problem is an issue with leave-one-out target encoding.

### Leakage analysis

As a general rule of thumb, a target leakage happens when one can get a better idea of the target value by exploiting the training data in a way that can not be reproduced using the test data. It can be anything like the specificities of the training data distributions, a timestamp peculiarity associated with specific target values, or the way we perform feature engineering like in the case of leave-one-out target encoding. In the example above because we are solving a binary classification problem, the encoding produces only two distinct values and therefore the encoded value **automatically reveals** the target. Also, as the attentive reader might guess, in K-fold target encoding, the more we remove observations from the training fold, the less the target leakage is important because the less we are able to deduce the target from the encoded feature but at the same time, the less accurate we are in our expectation estimates.

Now is there a way to get rid of the target leakage and at the same time use all of the training data? The answer is yes, read on.

### Ordered Target Statistics

Finally, we arrive at the procedure introduced by CatBoost to encode categorical variables. Up to that point we have shown the drivers that guided Ordered Target Statistics inception.

- Using the target to encode the category allows to create a powerful predictor, but...
- ... it also leads to target leakage and should therefore be used with caution. One way to avoid the problem is to learn the statistic on a separate hold out dataset but at the same time...
- ...we want the encoded statistic to be as accurate as possible which means that we should use as much relevant data as possible

If we had unlimited labeled data, we could, for every observation, draw a big new sample from the population and learn the statistic of interest on it. But we don't have that luxury so we need something else. The underlying principle is pretty simple and inspired by permutation testing and online learning in the way the observations arrive sequentially in time.

So we start by drawing a random permutation order. Then we iterate sequentially throughout the observations respecting that new order. And for

every observation, we compute the statistic of interest using only the observations that we have already seen in the past. The random permutation acts as an artificial time.

Image by author

Now, once the permutation is done, how is the categorical statistic computed?

First, the target is converted to a discrete integer and the way it is done depends on the problem we are solving:

- If we are in a regression setting, a quantization operation is performed on the continuous target. It means that the range of possible values for the target is divided into a specific number of buckets and all the targets that are in the same bucket are replaced with the same value. The way the buckets are determined depends on the selected strategy. More on this in the [CatBoost documentation](#).
- If we are in a classification setting, every distinct level is simply replaced with an increasing integer starting from 0.

Then, we compute the statistic. The calculation depends on the selected strategy but the idea remains approximately the same. For example, with the strategy Counter, the following computation takes place:

To understand the constituents of this formula, the best thing is to quote the documentation:

- ***curCount*** is the total number of objects in the training dataset with the current categorical feature value.
- ***maxCount*** the number of objects in the training dataset with the most frequent feature value.
- ***prior*** is a number (constant) defined by the starting parameters.

The `curCount` definition from the documentation is, to my mind, not very clear so we will take an example to understand it (we set the prior to 0.05).

Image by author

When computing the statistic for the fifth observation, we have:

- `countInClass` = 1 (The number of times, in the past training data, the target was equal to 1 when the categorical feature was equal to the Toyota)
- `maxCount` = 2 (The total number of Toyota brands in the past training data)

The statistic for that fifth observation is then equal to  $(1+0.05)/(2+1)=0.35$

In the end, the computed statistic reveals, the presence of each category and label combination in the training dataset when compared to the other categories. This has the effect of revealing the ordering of the categories when confronted to the target (which is what we want).

As you might notice, the first observations will not contain enough training data to produce robust estimates. The variance is very high. In order to remedy this issue, the authors of CatBoost proposed to start by generated several random permutations and produce an encoding for every permutation. The final result is simply the average of the different encodings.

With this procedure, we use all of the training data and we get rid of target leakage for good. The authors also studied the effect of the number of permutations that they apply during training. The result from the paper is exposed below:



Image from [CatBoost paper](#)

The authors concluded that the number of permutations has an effect on the encoding efficacy but the improvement that we get when increasing the number of permutations is minor.

### Conclusion

In this article we have reviewed common categorical variable encoding methodologies and have shown their strengths and weaknesses. Ordered Target Statistics is to my mind the best categorical variable encoding scheme that one can use of the shelf.

As a final remark, I briefly mentioned in this article that there is a method called Bayesian Target Encoding. This article was already too long to cover this procedure but I strongly encouraged you to read more about it (for example [here](#)) because it has a lot of very nice properties.

Finale note, you can find implementations of the encoding procedures described above (including CatBoost) with [this scikit-learn contrib package](#).

That is all for me, so I wish you the best in your data preparation adventures. Don't hesitate to leave a comment if something is unclear.

[Statistics](#)[Machine Learning](#)[Gradient Boosting](#)[Editors Pick](#)



## Written by Adrien Biarnes

703 Followers · Writer for Towards Data Science

Follow

I am a senior ML engineer working on recommender systems —  
<https://www.linkedin.com/in/adrien-biarnes-81975717>

### More from Adrien Biarnes and Towards Data Science

Adrien Biarnes in Towards Data Science

#### Mixture-of-Softmaxes for Deep Session-based Recommender

Modern session-based deep recommender systems can be somehow limited by the

🌟 · 15 min read · Jul 19

👏 59



Dominik Polzer in Towards Data Science

#### All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt

🌟 · 26 min read · Jun 22

👏 3.5K



31



Miriam Santos in Towards Data Science

#### Pandas 2.0: A Game-Changer for Data Scientists?

The Top 5 Features for Efficient Data Manipulation

7 min read · Jun 27

👏 2K



25



Adrien Biarnes in MLearning.ai

#### Building a Multi-Stage Recommendation System (Part 1.1)

Understanding candidate generation and the two-tower model

🌟 · 15 min read · Aug 13, 2022

👏 318



8



See all from Adrien Biarnes

See all from Towards Data Science

## Recommended from Medium

 Aicha Bokb... in Artificial Intelligence in Plain Engl...

### 4 ways to encode categorical features with high cardinality

We explore 4 methods to encode categorical variables with high cardinality: target

9 min read · Jun 26



 Kevin Menear

### Comparing Label Encoding, One-Hot Encoding, and Binary Encoding

# This article is a bit different. I wanted to understand binary encoding, so I began a

7 min read · Feb 8



## Lists

### Predictive Modeling w/ Python

18 stories · 183 saves

### Natural Language Processing

459 stories · 85 saves

### Practical Guides to Machine Learning

10 stories · 198 saves

### The New Chatbots: ChatGPT, Bard, and Beyond

13 stories · 63 saves

 Diogo Leitão in Towards Data Science


## Gradient Boosting: To Early Stop or Not To Early Stop

How early stopping halves training time for models like LightGBM, XGBoost and

7 min read · Mar 23

 93  5

 Subrat Sekhar Sahu in Walmart Global Tech Blog

## Efficient One-hot encoding for categorical features with high

Tags: Data Science, Python, Machine Learning, Big Data, Artificial Intelligence

4 min read · Apr 27

 52 

 Krishnakanth Naik Jarapala in AI Skunks

## Categorical Data Encoding Techniques

Introduction:

7 min read · Mar 14

 12 

 Connie Zhou

## Understanding Important Machine Learning Model Metrics—Gini/KS

Machine learning models are powerful tools for making predictions and gaining insights

4 min read · Jun 14

See more recommendations