Categorical Encoding Methods

Alexander Wang
Feature Labs, Inc.

Abstract

Here, we will be investigating the most common approaches to categorical encoding and looking at possible integration into Featuretools software.

1 Introduction

During feature engineering, we most commonly deal with two types of structured data: numerical data and categorical data. It is often easier to deal with numeric data compared to categorical data because Machine Learning models typically deal with mathematical vectors–numeric data can thus be used without as many additional complexities. However, machine learning algorithms cannot work directly with categorical data and therefore we must do some amount of transformations and engineering on our data before continuing. We will look at some of the most common approaches.

2 Categorical Encoding Approaches

There are two major types of categorical data, both of which have different optimal encoding approaches. First, there's nominal data, which has no concept of ordering (i.e. music genres or cuisine types). Ordinal categorical attributes instead do have some ordering, such as shirt size ranging from XS to XXL.

2.1 Classic Encoders

These are the most straightforward and easy to understand encoders, so they're often very useful for ML practitioners.

Ordinal/Label

This is the most straightforward approach to categorical encoding. Ordinal Encoders convert each string value into a whole number. The first unique value in the column is assigned to 1, the second becomes 2, etc. However, typically just keeping the data like this is not recommended, especially if the data is nominal. Machine Learning algorithms will assume this variable is continuous, and will assume an ordering that is either incorrect or does not exist.

If the value is ordinal and increments by the same amount each time, then an ordinal encoder is useful assuming one assigns the correct numbers to the correct values. For example, we'd want S, M, L sizes to map to 1, 2, 3 not 1, 3, 2. If the variable is nominal or does not increase evenly, then one should use another encoder after in order to appropriately handle the data.

OneHot/Dummy

One-hot encoding creates a new column for each value. For each new column, a 1 is assigned if the row contains that column's value and a 0 otherwise.

So, if we had colors blue, green, red, then a row with the value blue would look like 1,0,0 while a green row would look like 0,1,0.

This typically performs very well, and Featuretools encode_features currently utilizes this. However, the number of new features is equal to the number of unique values, which leads to severe memory issues with high cardinality datasets. Furthermore, one-hot encoded data can degrade the performance of decision-tree based algorithms.

Binary

The categories, if not already in numeric form, are Ordinal Encoded. The resulting integers are converted to binary, and then the digits are split into columns. For example, 5 would become 3 columns (1,0,1).

This creates fewer columns than one-hot, is more memory efficient, and reduces chances of dimensionality problems overall. Furthermore, similar values may tend to overlap with each other across multiple columns, allowing for machine learning algorithms to learn similarities. However, on the other hand, binary encoding does still imply ordering/similarity where it may not exist, so it really only excels for ordinal data with high cardinality.

Typically for nominal data, another approach such as hashing with more control would make more sense.

BaseN

For N=2, it's the same as BinaryEncoder (simply converting the integer to base N). The only real benefit is that you can use it with gridsearchCV (makes it easy to tune), but this isn't an essential step nor difficult to do otherwise.

Hashing

Similar to one-hot encoding but with fewer new dimensions but comes with the cost of info loss due to collisions. This is normally not a problem unless there's a lot of overlap. With high cardinality, it's worth trying because you do have a lot more control versus something like a binary encoder.

2.2 Contrast Encoders

All output one column for each column value. This is typically not recommended for nominal variables, but these encoders do give you a control for very specific patterns.

Helmert

Compares the mean of the dependent variable for a level to the mean of the dependent variable over all the previous values. This can actually be rather easily calculated through matrix multiplication.

Sum

Similar to Helmert except it compares the mean of the dependent variable to the overall mean over all the levels instead.

Backward Difference

Mean of the dependent variable is compared with the mean for the prior level.

Polynomial

Form of encoding that looks for linear, quadratic, and cubic (or any degree) trends. This can be used when the intervals between variables are not even but still follow a defined pattern.

2.3 Bayesian Encoders

These use information from the dependent variable in their encodings. It outputs only one column and works well with high cardinality data.

Target

Uses the mean of the DV but one must take steps to avoid overfitting/response leakage. This is typically used for classification tasks.

LeaveOneOut

Very similar to Target encoding, except the row in question leaves its own value out when calculating the mean to avoid contamination. This is used by Owen Zhang a lot. It can help split up larger categories logically and fit/group certain categories together better.

WeightsOfEvidence

The Weight of Evidence tells the predictive power of an independent variable in relation to the dependent variable. It's calculated through:

$$\text{WOE} = \ln \frac{\text{Distribution of non-events}}{\text{Distribution of events}}$$

Then, categories with similar WOE's are typically also similar, which could help with the accuracy of a machine learning algorithm.

James-Stein

For a feature value, the James-Stein estimator returns a weighted average of the mean target value of the observed feature value and the overall mean target value. If the weight if the overall mean is given by B, Stein sets $B = \frac{\text{var}y_i}{\text{var}y_i + \text{var}y}$ and then provides ways to estimate these values.

M-estimator

Essentially a simpler version of Target encoding. This only has one tunable parameter (n) versus target encoder, which has two tunable parameters (min_samples_leaf and smoothing).

3 Featuretools Applications

3.1 Current Status

Currently, Featuretools handles categorical encoding through the encode_features function, which takes in the output of dfs. For the categorical encoding approach itself, Featuretools currently codes it own method of one-hot encoding.

3.2 Future Directions

Potentially an argument within encode_features that can detail what kind of categorical encoder is used (while keeping one-hot encoding as the default). Adding an option for label encoding and binary encoding should be relatively simple. However, for more complicated encoders such as James-Stein or Hashing, there may need to more keyword arguments that need to be passed in for these encoders since they rely on values that aren't universal to datasets.

In most cases, one should be able to implement the more complicated categorical encoders relatively easily after getting the feature matrix from DFS. However, for the simpler encoding methods (namely Binary and Label), we should implement them to provide an alternative if a user is faced with high cardinality data.

After those two, the next possible encoders to implement are Hashing (we can have default hashing functions with options to change it to a more customized one) and some variation of Target Encoding. The problem with Target Encoding is that the test/train data must be separated beforehand, which makes direct integration into encode_features more challenging.