(https://google.com/racialequity?authuser=0)

Representation: Feature Engineering

ıted Time: 9 minutes

In traditional programming, the focus is on code. In machine learning projects, the focus shifts to representation. That is, one way developers hone a model is by adding and improving its features.

Mapping Raw Data to Features

The left side of Figure 1 illustrates raw data from an input data source; the right side illustrates a **feature vector**, which is the set of floating-point values comprising the examples in your data set. **Feature engineering** means transforming raw data into a feature vector. Expect to spend significant time doing feature engineering.

Many machine learning models must represent the features as real-numbered vectors since the feature values must be multiplied by the model weights.

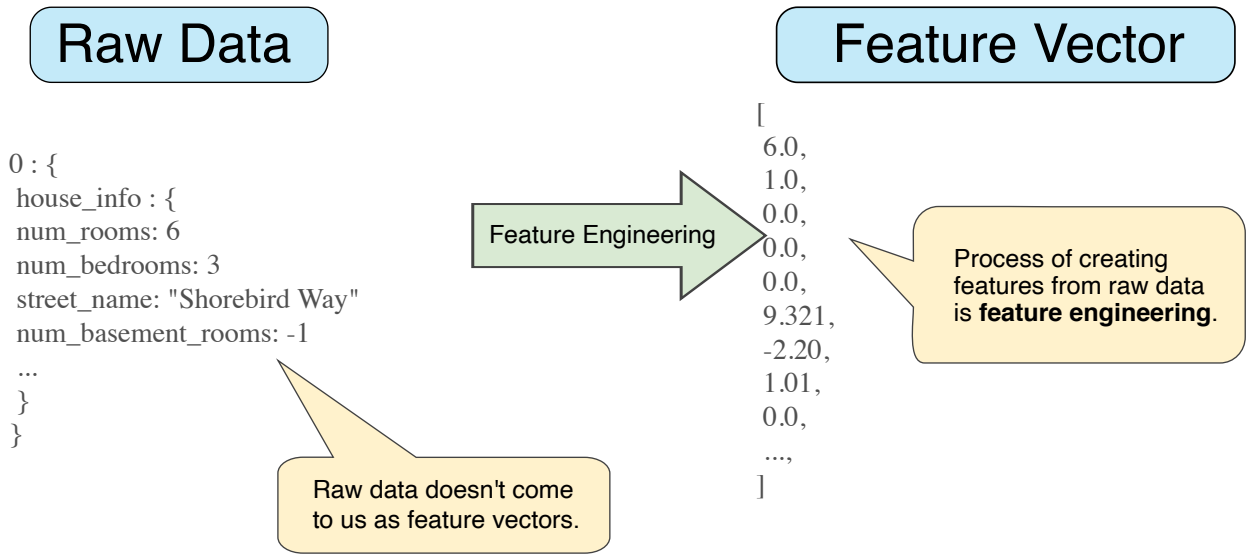

Figure 1. Feature engineering maps raw data to ML features.

Mapping numeric values

Integer and floating-point data don't need a special encoding because they can be multiplied by a numeric weight. As suggested in Figure 2, converting the raw integer value 6 to the feature value 6.0 is trivial:

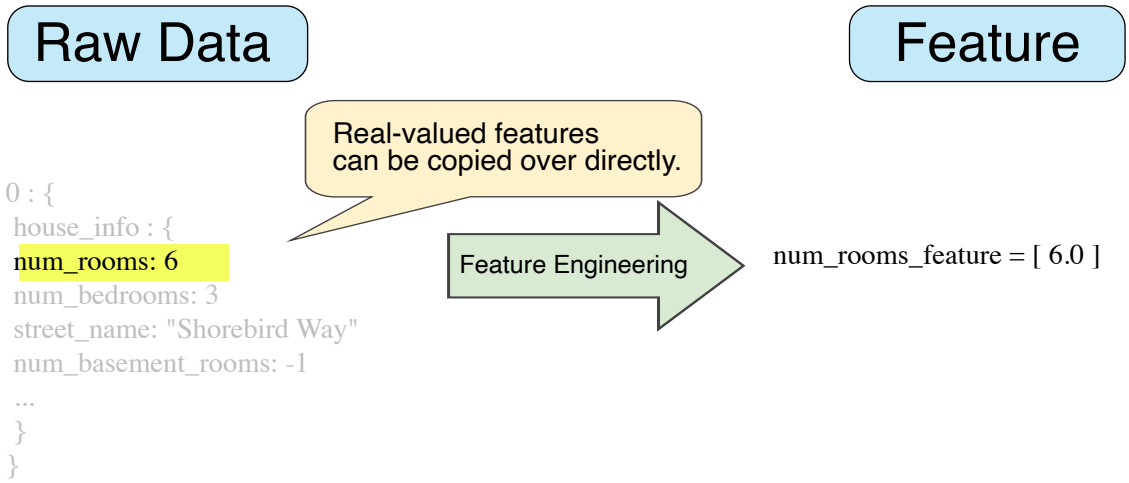

Figure 2. Mapping integer values to floating-point values.

Mapping categorical values

Categorical features (https://developers.google.com/machine-learning/glossary?authuser=0#categorical_data) have a discrete set of possible values. For example, there might be a feature called `street_name` with options that include:

```
{'Charleston Road', 'North Shoreline Boulevard', 'Shorebird Way', 'Rengstorff
```

Since models cannot multiply strings by the learned weights, we use feature engineering to convert strings to numeric values.

We can accomplish this by defining a mapping from the feature values, which we'll refer to as the **vocabulary** of possible values, to integers. Since not every street in the world will appear in our dataset, we can group all other streets into a catch-all "other" category, known as an **OOV (out-of-vocabulary) bucket**.

Using this approach, here's how we can map our street names to numbers:

- map Charleston Road to 0

- map North Shoreline Boulevard to 1
- map Shorebird Way to 2
- map Rengstorff Avenue to 3
- map everything else (OOV) to 4

However, if we incorporate these index numbers directly into our model, it will impose some constraints that might be problematic:

- We'll be learning a single weight that applies to all streets. For example, if we learn a weight of 6 for `street_name`, then we will multiply it by 0 for Charleston Road, by 1 for North Shoreline Boulevard, 2 for Shorebird Way and so on. Consider a model that predicts house prices using `street_name` as a feature. It is unlikely that there is a linear adjustment of price based on the street name, and furthermore this would assume you have ordered the streets based on their average house price. Our model needs the flexibility of learning different weights for each street that will be added to the price estimated using the other features.
- We aren't accounting for cases where `street_name` may take multiple values. For example, many houses are located at the corner of two streets, and there's no way to encode that information in the `street_name` value if it contains a single index.

To remove both these constraints, we can instead create a binary vector for each categorical feature in our model that represents values as follows:

- For values that apply to the example, set corresponding vector elements to `1`.
- Set all other elements to `0`.

The length of this vector is equal to the number of elements in the vocabulary. This representation is called a **one-hot encoding** when a single value is 1, and a **multi-hot encoding** when multiple values are 1.

Figure 3 illustrates a one-hot encoding of a particular street: Shorebird Way. The element in the binary vector for Shorebird Way has a value of `1`, while the elements for all other streets have values of `0`.

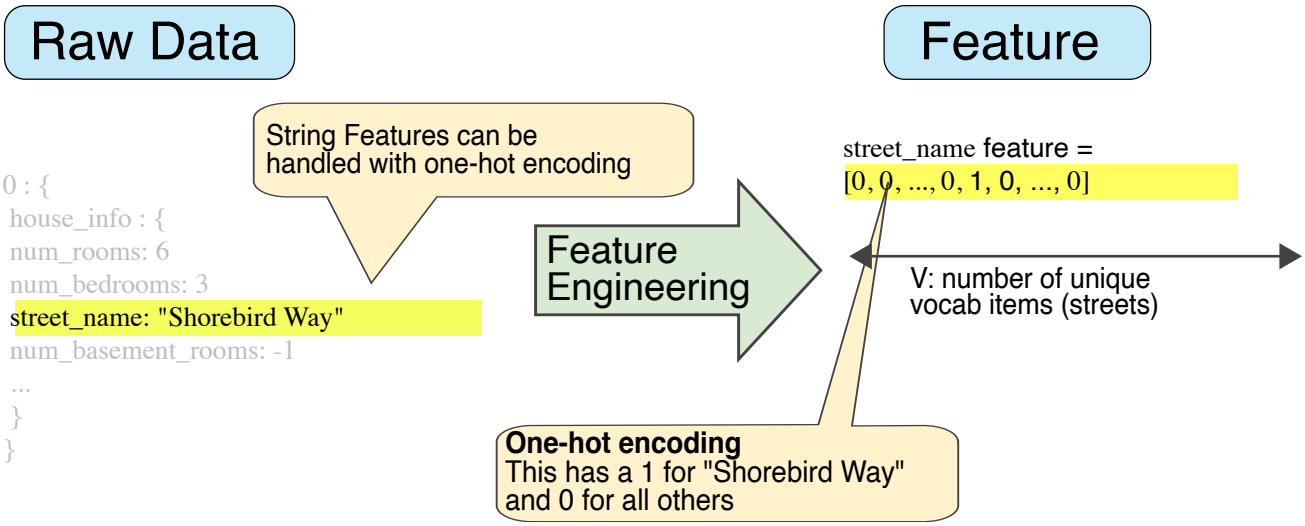

Figure 3. Mapping street address via one-hot encoding.

This approach effectively creates a Boolean variable for every feature value (e.g., street name). Here, if a house is on Shorebird Way then the binary value is 1 only for Shorebird Way. Thus, the model uses only the weight for Shorebird Way.

Similarly, if a house is at the corner of two streets, then two binary values are set to 1, and the model uses both their respective weights.

ɔt encoding extends to numeric data that you do not want to directly multiply by a weight, such as a posta

Sparse Representation

Suppose that you had 1,000,000 different street names in your data set that you wanted to include as values for `street_name`. Explicitly creating a binary vector of 1,000,000 elements where only 1 or 2 elements are true is a very inefficient representation in terms of both storage and computation time when processing these vectors. In this situation, a common approach is to use a sparse representation (/machine-learning/glossary#sparse_representation) in which only nonzero values are stored. In sparse representations, an independent model weight is still learned for each feature value, as described above.

ɪrms

ɜcrete feature
://developers.google.com/machine-
ıg/glossary?authuser=0#discrete_feature)
e-hot encoding

- feature engineering
(https://developers.google.com/machine-learning/glossary?authuser=0#feature_engineering)
- representation