Transforming Categorical Data

Some of your features may be discrete values that aren't in an ordered relationship. Examples include breeds of dogs, words, or postal codes. These features are known as **categorical** (/machine-learning/glossary#categorical_data) and each value is called a category. You can represent categorical values as strings or even numbers, but you won't be able to compare these numbers or subtract them from each other.

Oftentimes, you should represent features that contain integer values as categorical data instead of as numerical data. For example, consider a postal code feature in which the values are integers. If you mistakenly represent this feature numerically, then you're asking the model to find a numeric relationship between different postal codes; for example, you're expecting the model to determine that postal code 20004 is twice (or half) the signal as postal code 10002. By representing postal codes as categorical data, you enable the model to find separate signals for each individual postal code.

If the number of categories of a data field is small, such as the day of the week or a limited palette of colors, you can make a unique feature for each category. For example:

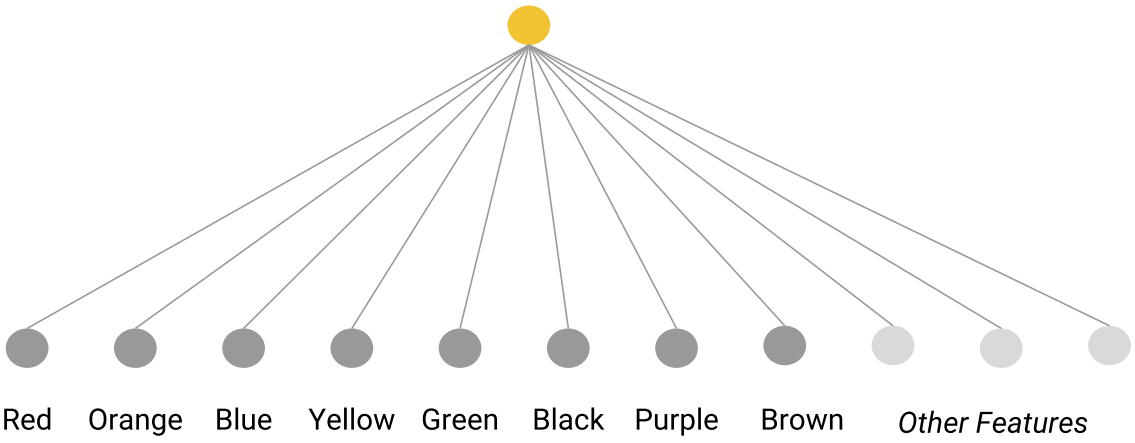

Figure 1: A unique feature for each category.

A model can then learn a separate weight for each color. For example, perhaps the model could learn that red cars are more expensive than green cars.

The features can then be indexed.

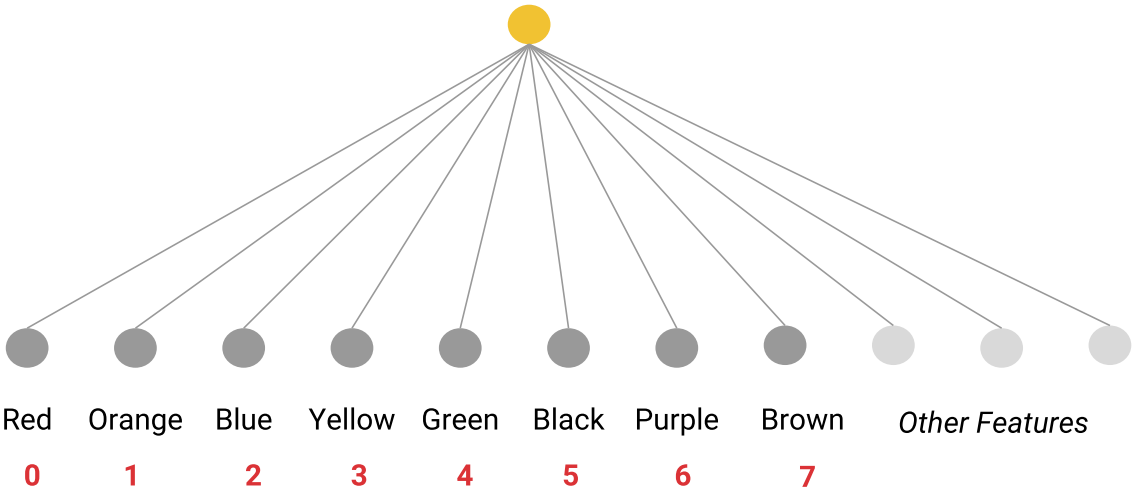

Figure 2: Indexed features.

This sort of mapping is called a **vocabulary**.

Vocabulary

In a vocabulary, each value represents a unique feature.

Index Number	Category
0	Red
1	Orange
2	Blue
...	...

The model looks up the index from the string, assigning 1.0 to the corresponding slot in the feature vector and 0.0 to all the other slots in the feature vector.

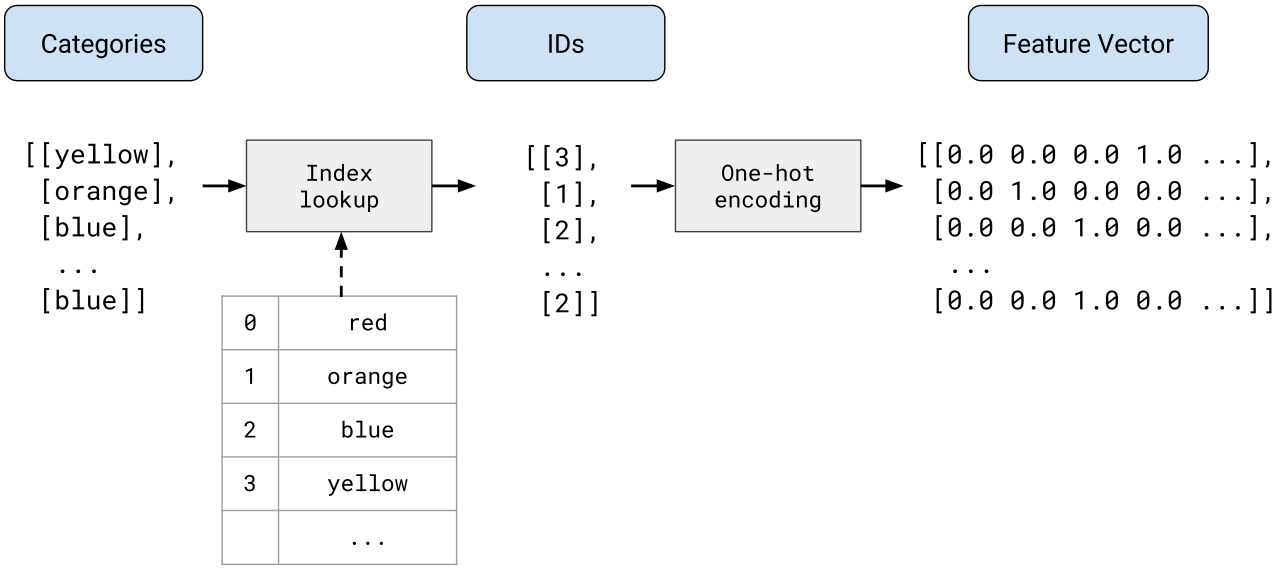

Figure 3: The end-to-end process to map categories to feature vectors.

Note about sparse representation

If your categories are the days of the week, you might, for example, end up representing Friday with the feature vector [0, 0, 0, 0, 1, 0, 0]. However, most implementations of ML systems will represent this vector in memory with a sparse representation. A common representation is a list of non-empty values and their corresponding indices—for example, 1.0 for the value and [4] for the index. This allows you to spend less memory storing a huge amount of 0s and allows more efficient matrix multiplication. In terms of the underlying math, the [4] is equivalent to [0, 0, 0, 0, 1, 0, 0].

Out of Vocab (OOV)

Just as numerical data contains outliers, categorical data does, as well. For example, consider a data set containing descriptions of cars. One of the features of this data set could be the car's color. Suppose the common car colors (black, white, gray, and so on) are well represented in this data set and you make each one of them into a category so you can learn how these different colors affect value. However, suppose this data set contains a small number of cars with eccentric colors (mauve, puce, avocado). Rather than giving each of these colors a separate category, you could lump them into a catch-all category called **Out of Vocab** (**OOV**). By using OOV, the system won't waste time training on each of those rare colors.

Hashing

Another option is to hash every string (category) into your available index space. Hashing often causes collisions, but you rely on the model learning some shared representation of

the categories in the same index that works well for the given problem.

For important terms, hashing can be worse than selecting a vocabulary, because of collisions. On the other hand, hashing doesn't require you to assemble a vocabulary, which is advantageous if the feature distribution changes heavily over time.

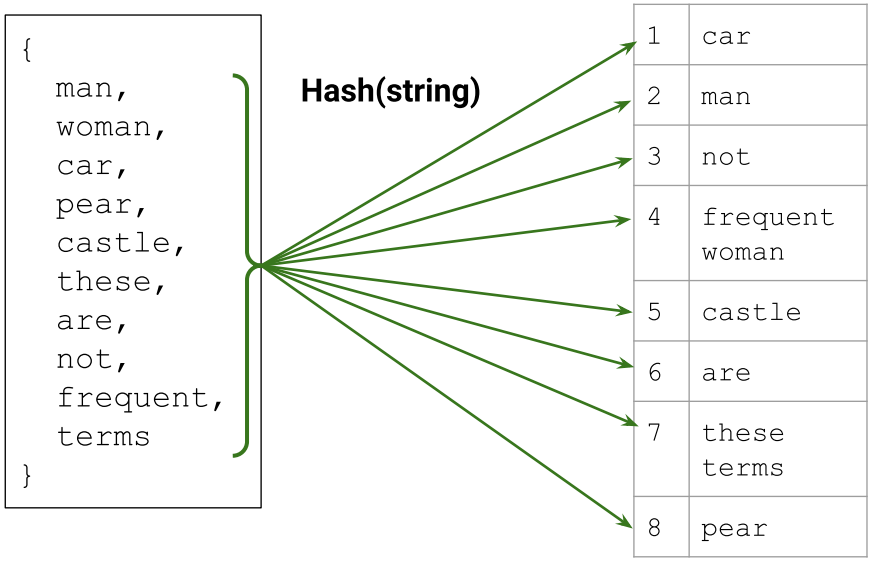

Figure 4: Mapping items to a vocabulary.

Hybrid of Hashing and Vocabulary

You can take a hybrid approach and combine hashing with a vocabulary. Use a vocabulary for the most important categories in your data, but replace the OOV bucket with multiple OOV buckets, and use hashing to assign categories to buckets.

The categories in the hash buckets must share an index, and the model likely won't make good predictions, but we have allocated some amount of memory to attempt to learn the categories outside of our vocabulary.

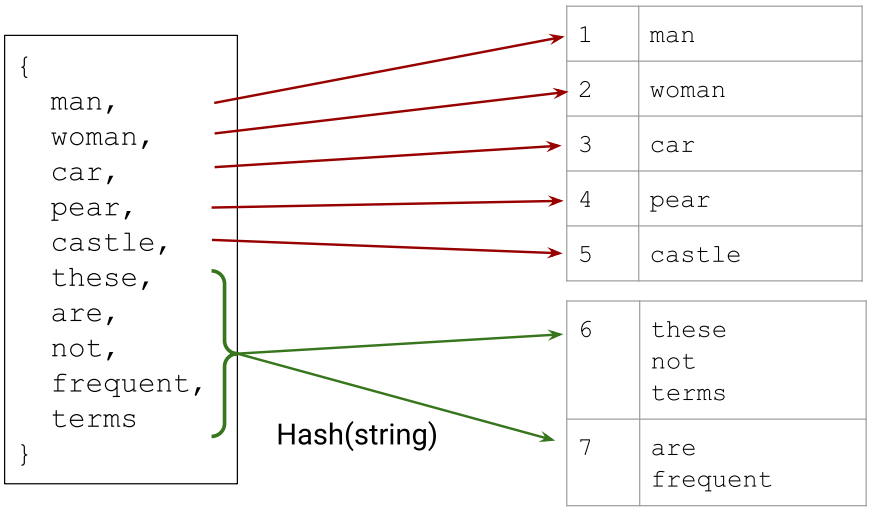

Figure 5: Hybrid approach combining vocabulary and hashing.

Note about Embeddings

Recall from the Machine Learning Crash Course (/machine-learning/crash-course/embeddings/video-lecture) that an **embedding** (/machine-learning/crash-course/glossary#embeddings) is a categorical feature represented as a continuous-valued feature. Deep models frequently convert the indices from an index to an embedding.

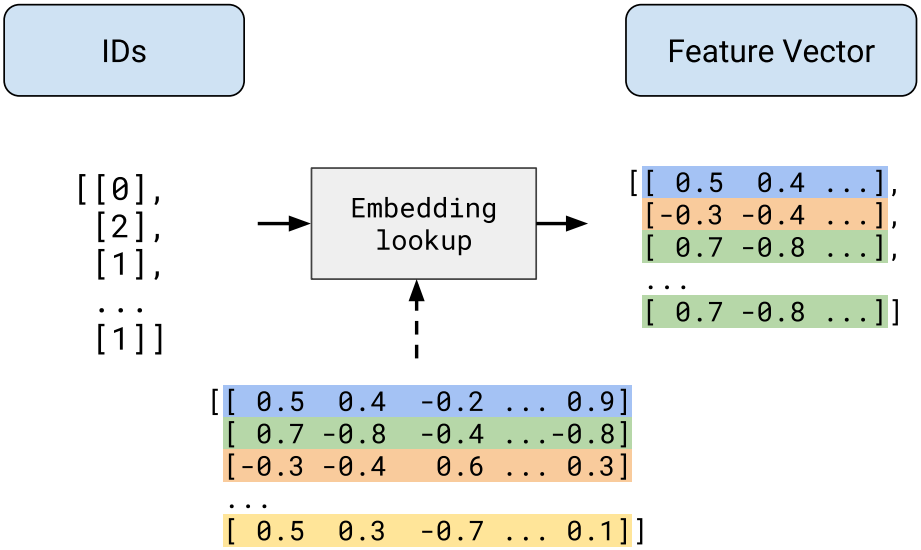

Figure 6: Sparse feature vectors via embedding

The other transformations we've discussed could be stored on disk, but embeddings are different. Since embeddings are trained, they're not a typical data transformation—they are part of the model. They're trained with other model weights, and functionally are equivalent to a layer of weights.

What about pretrained embeddings? Pretrained embeddings are still typically modifiable during training, so they're still conceptually part of the model.

erms:

- tegorical data nachine-rning/glossary#categorical_data)
- iture cross nachine-rning/glossary#feature_cross)
- discrete feature (/machine-learning/glossary#discrete_feature)
- one-hot encoding (/machine-learning/glossary#one-hot_encoding)
- embeddings (/machine-learning/glossary#embed

Previous
← Bucketing (/machine-learning/data-prep/transform/bucketing)

Next
Check Your Understanding → (/machine-learning/data-prep/transform/check-your-understanding)