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

Representation: Qualities of Good Features

ıted Time: 10 minutes

We've explored ways to map raw data into suitable feature vectors, but that's only part of the work. We must now explore what kinds of values actually make good features within those feature vectors.

Avoid rarely used discrete feature values

Good feature values should appear more than 5 or so times in a data set. Doing so enables a model to learn how this feature value relates to the label. That is, having many examples with the same discrete value gives the model a chance to see the feature in different settings, and in turn, determine when it's a good predictor for the label. For example, a `house_type` feature would likely contain many examples in which its value was `victorian`:

```
house_type: victorian
```
✔

Conversely, if a feature's value appears only once or very rarely, the model can't make predictions based on that feature. For example, `unique_house_id` is a bad feature because each value would be used only once, so the model couldn't learn anything from it:

```
unique_house_id: 8SK982ZZ1242Z
```
✘

Prefer clear and obvious meanings

Each feature should have a clear and obvious meaning to anyone on the project. For example, the following good feature is clearly named and the value makes sense with respect to the name:

```
house_age_years: 27
```
✔

Conversely, the meaning of the following feature value is pretty much indecipherable to anyone but the engineer who created it:

```
house_age: 851472000
```
✘

In some cases, noisy data (rather than bad engineering choices) causes unclear values. For example, the following user_age_years came from a source that didn't check for appropriate values:

```
user_age_years: 277
```
✘

Don't mix "magic" values with actual data

Good floating-point features don't contain peculiar out-of-range discontinuities or "magic" values. For example, suppose a feature holds a floating-point value between 0 and 1. So, values like the following are fine:

```
quality_rating: 0.82
quality_rating: 0.37
```
✔

However, if a user didn't enter a `quality_rating`, perhaps the data set represented its absence with a magic value like the following:

```
quality_rating: -1
```
✘

To explicitly mark magic values, create a Boolean feature that indicates whether or not a `quality_rating` was supplied. Give this Boolean feature a name like `is_quality_rating_defined`.

In the original feature, replace the magic values as follows:

- For variables that take a finite set of values (discrete variables), add a new value to the set and use it to signify that the feature value is missing.
- For continuous variables, ensure missing values do not affect the model by using the mean value of the feature's data.

Account for upstream instability

The definition of a feature shouldn't change over time. For example, the following value is useful because the city name *probably* won't change. (Note that we'll still need to convert a string like "br/sao_paulo" to a one-hot vector.)

```
city_id: "br/sao_paulo"
```
✔

But gathering a value inferred by another model carries additional costs. Perhaps the value "219" currently represents Sao Paulo, but that representation could easily change on a future run of the other model:

```
inferred_city_cluster: "219"
```
✘

Help Center (https://support.google.com/machinelearningeducation?authuser=0)

Previous
← Feature Engineering
(https://developers.google.com/machine-learning/crash-course/representation/feature-engineering?authuser=0)

Next
Cleaning Data →
(https://developers.google.com/machine-learning/crash-course/representation/cleaning-data?authuser=0)