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

Representation: Cleaning Data

ıted Time: 10 minutes

Apple trees produce some mixture of great fruit and wormy messes. Yet the apples in high-end grocery stores display 100% perfect fruit. Between orchard and grocery, someone spends significant time removing the bad apples or throwing a little wax on the salvageable ones. As an ML engineer, you'll spend enormous amounts of your time tossing out bad examples and cleaning up the salvageable ones. Even a few "bad apples" can spoil a large data set.

Scaling feature values

Scaling means converting floating-point feature values from their natural range (for example, 100 to 900) into a standard range (for example, 0 to 1 or -1 to +1). If a feature set consists of only a single feature, then scaling provides little to no practical benefit. If, however, a feature set consists of multiple features, then feature scaling provides the following benefits:

- Helps gradient descent converge more quickly.
- Helps avoid the "NaN trap," in which one number in the model becomes a NaN (https://wikipedia.org/wiki/NaN) (e.g., when a value exceeds the floating-point precision limit during training), and—due to math operations—every other number in the model also eventually becomes a NaN.
- Helps the model learn appropriate weights for each feature. Without feature scaling, the model will pay too much attention to the features having a wider range.

You don't have to give every floating-point feature exactly the same scale. Nothing terrible will happen if Feature A is scaled from -1 to +1 while Feature B is scaled from -3 to +3. However, your model will react poorly if Feature B is scaled from 5000 to 100000.

+ **Click the plus icon to learn more about scaling.**

One obvious way to scale numerical data is to linearly map [min value, max value] to a small scale, such as [-1, +1].

Another popular scaling tactic is to calculate the Z score of each value. The Z score relates the number of standard deviations away from the mean. In other words:

$$scaledvalue = (value - mean)/stddev.$$

For example, given:

- mean = 100
- standard deviation = 20
- original value = 130

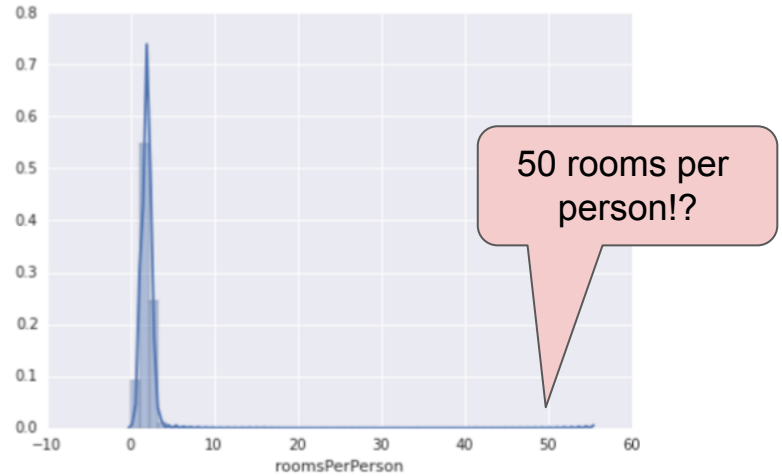
then:

```
scaled_value = (130 - 100) / 20
scaled_value = 1.5
```

Scaling with Z scores means that most scaled values will be between -3 and +3, but a few values will be a little higher or lower than that range.

Handling extreme outliers

The following plot represents a feature called `roomsPerPerson` from the California Housing data set (https://developers.google.com/machine-learning/crash-course/california-housing-data-description?authuser=0). The value of `roomsPerPerson` was calculated by dividing the total number of rooms for an area by the population for that area. The plot shows that the vast majority of areas in California have one or two rooms per person. But take a look along the x-axis.




```
roomsPerPerson = totalRooms / population
```

Figure 4. A verrrrry lonnnnnnng tail.

How could we minimize the influence of those extreme outliers? Well, one way would be to take the log of every value:


```
roomsPerPerson = log((totalRooms / population) + 1)
```

Figure 5. Logarithmic scaling still leaves a tail.

Log scaling does a slightly better job, but there's still a significant tail of outlier values. Let's pick yet another approach. What if we simply "cap" or "clip" the maximum value of `roomsPerPerson` at an arbitrary value, say 4.0?

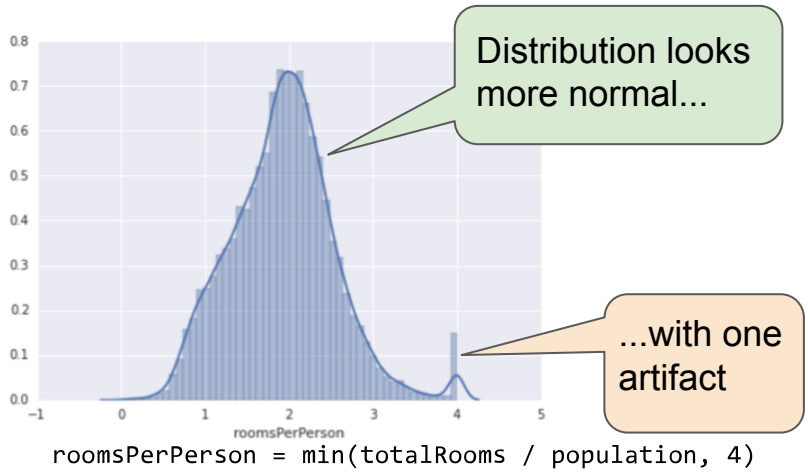


Figure 6. Clipping feature values at 4.0

Clipping the feature value at 4.0 doesn't mean that we ignore all values greater than 4.0. Rather, it means that all values that were greater than 4.0 now become 4.0. This explains the funny hill at 4.0. Despite that hill, the scaled feature set is now more useful than the original data.

Binning

The following plot shows the relative prevalence of houses at different latitudes in California. Notice the clustering—Los Angeles is about at latitude 34 and San Francisco is roughly at latitude 38.

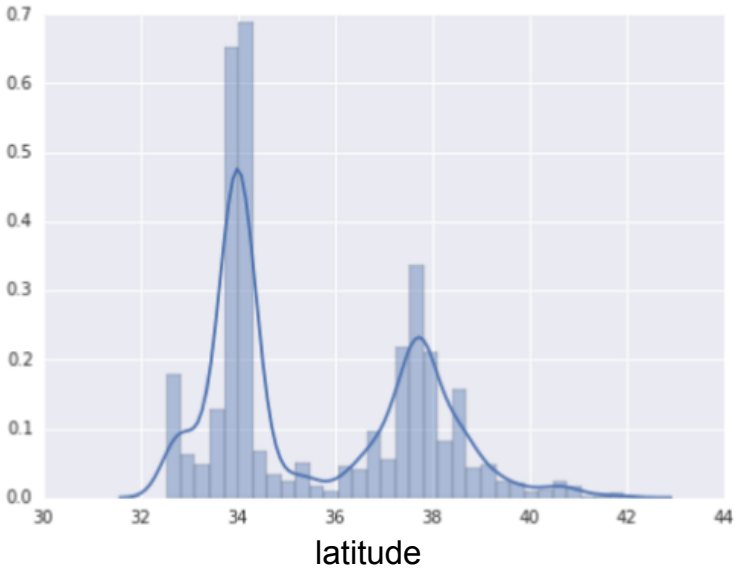


Figure 7. Houses per latitude.

In the data set, `latitude` is a floating-point value. However, it doesn't make sense to represent `latitude` as a floating-point feature in our model. That's because no linear relationship exists between latitude and housing values. For example, houses in latitude 35 are not $\frac{35}{34}$ more expensive (or less expensive) than houses at latitude 34. And yet, individual latitudes probably are a pretty good predictor of house values.

To make latitude a helpful predictor, let's divide latitudes into "bins" as suggested by the following figure:

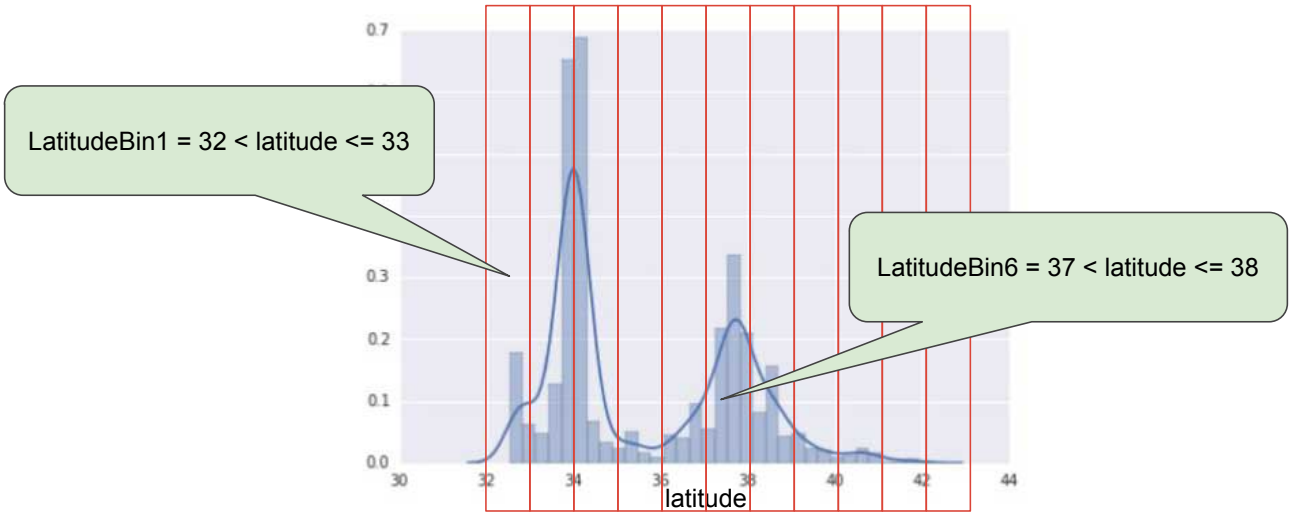


Figure 8. Binning values.

Instead of having one floating-point feature, we now have 11 distinct boolean features (`LatitudeBin1`, `LatitudeBin2`, ..., `LatitudeBin11`). Having 11 separate features is somewhat inelegant, so let's unite them into a single 11-element vector. Doing so will enable us to represent latitude 37.4 as follows:

```
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
```

Thanks to binning, our model can now learn completely different weights for each latitude.

Click the plus icon to learn more about binning boundaries.

For simplicity's sake in the latitude example, we used whole numbers as bin boundaries. Had we wanted finer-grain resolution, we could have split bin boundaries at, say, every tenth of a degree. Adding more bins enables the model to

learn different behaviors from latitude 37.4 than latitude 37.5, but only if there are sufficient examples at each tenth of a latitude.

Another approach is to bin by quantile (https://wikipedia.org/wiki/Quantile), which ensures that the number of examples in each bucket is equal. Binning by quantile completely removes the need to worry about outliers.

Scrubbing

Until now, we've assumed that all the data used for training and testing was trustworthy. In real-life, many examples in data sets are unreliable due to one or more of the following:

- **Omitted values.** For instance, a person forgot to enter a value for a house's age.
- **Duplicate examples.** For example, a server mistakenly uploaded the same logs twice.
- **Bad labels.** For instance, a person mislabeled a picture of an oak tree as a maple.
- **Bad feature values.** For example, someone typed in an extra digit, or a thermometer was left out in the sun.

Once detected, you typically "fix" bad examples by removing them from the data set. To detect omitted values or duplicated examples, you can write a simple program. Detecting bad feature values or labels can be far trickier.

In addition to detecting bad individual examples, you must also detect bad data in the aggregate. Histograms are a great mechanism for visualizing your data in the aggregate. In addition, getting statistics like the following can help:

- Maximum and minimum
- Mean and median
- Standard deviation

Consider generating lists of the most common values for discrete features. For example, do the number of examples with `country:uk` match the number you expect. Should `language:jp` really be the most common language in your data set?

Know your data

Follow these rules:

- Keep in mind what you think your data should look like.

- Verify that the data meets these expectations (or that you can explain why it doesn't).
- Double-check that the training data agrees with other sources (for example, dashboards).

Treat your data with all the care that you would treat any mission-critical code. Good ML relies on good data.

Additional Information

Rules of Machine Learning, ML Phase II: Feature Engineering (https://developers.google.com/machine-learning/rules-of-ml/?authuser=0#ml_phase_ii_feature_engineering)

erms

- ıning ://developers.google.com/machine-ıg/glossary?authuser=0#binning)
- ıN trap ://developers.google.com/machine-ıg/glossary?authuser=0#NaN_trap)
- feature set (https://developers.google.com/machine-learning/glossary?authuser=0#feature_set)
- outliers (https://developers.google.com/machine-learning/glossary?authuser=0#outliers)

aling (https://developers.google.com/machine-learning/glossary?authuser=0#scaling)

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

Previous
← Qualities of Good Features (https://developers.google.com/machine-learning/crash-course/representation/qualities-of-good-features?authuser=0)

Next
Video Lecture → (https://developers.google.com/machine-learning/crash-course/feature-crosses/video-lecture?authuser=0)