Feature Scaling, Normalization, and Standardization in Machine Learning

Written by: *Mohammad Amin Alemohammad, Mahdi Rafati*

- Features with Different Scales

Nowadays, Machine Learning is one of the most popular fields in technology industry. In the process of building an ML projects, we need to perform many important tasks. One of these tasks which is also impossible not to happen, is called feature engineering.

Sometimes the dataset which you are working on might have some attributes that some of them have much bigger or much smaller range of values than other attributes. This can cause problems during fitting your model to the training data.

One key aspect of feature engineering is **scaling**, **normalization**, and **standardization**, which involves transforming the data to make it more suitable for modeling. These techniques can help to improve model performance, reduce the impact of outliers, and ensure that the data is on the same scale.

- Feature Scaling

Feature scaling is a data preprocessing technique used to transform the values of features or variables in a dataset to a similar scale.

There are several common techniques for feature scaling, including **standardization**, **normalization**, and **min-max** scaling. These methods adjust the feature values while preserving their relative relationships and distributions.

By applying feature scaling, the dataset's features can be transformed to a more consistent scale, making it easier to build accurate and effective machine learning models. As an example, having features on a similar scale can help the gradient descent converge more quickly towards the global minimal.

- Normalization

Normalization, a vital aspect of Feature Scaling, is a data preprocessing technique employed to standardize the values of features in a dataset, bringing them to a common scale.

Four common normalization techniques may be useful:

1. **Min-Max Scaling:**
   Means converting floating-point feature values from their natural range (for example, 100 to 900) into a standard range—usually 0 and 1 (or sometimes -1 to +1). Use the following simple formula to scale to a range:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

   Scaling to a range is a good choice when both of the following conditions are met:

   - You know the approximate upper and lower bounds on your data with few or no outliers.
   - Your data is approximately uniformly distributed across that range.

A good example is age. Most age values falls between 0 and 90, and every part of the range has a substantial number of people.

2. **Feature Clipping:**
   If your data set contains extreme outliers, you might try feature clipping, which caps all feature values above (or below) a certain value to fixed value. For example, you could clip all temperature values above 40 to be exactly 40.
   You may apply feature clipping before or after other normalizations.

   Formula: Set min/max values to avoid outliers.

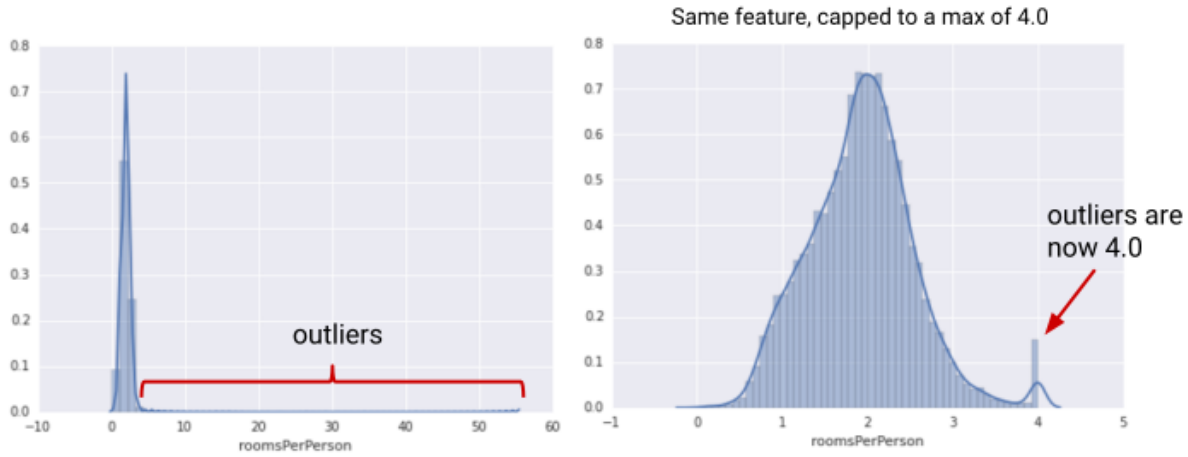

Figure 1. Comparing a raw distribution and its clipped version.

3. **Log Scaling:**
   Log scaling computes the log of your values to compress a wide range to a narrow range.

$$x' = \log(x)$$

Log scaling is helpful when a handful of your values have many points, while most other values have few points. This data distribution is known as the *power law* distribution. Movie ratings are a good example. In the chart below, most movies have very few ratings (the data in the tail), while a few have lots of ratings (the data in the head). Log scaling changes the

distribution, helping to improve linear model performance.

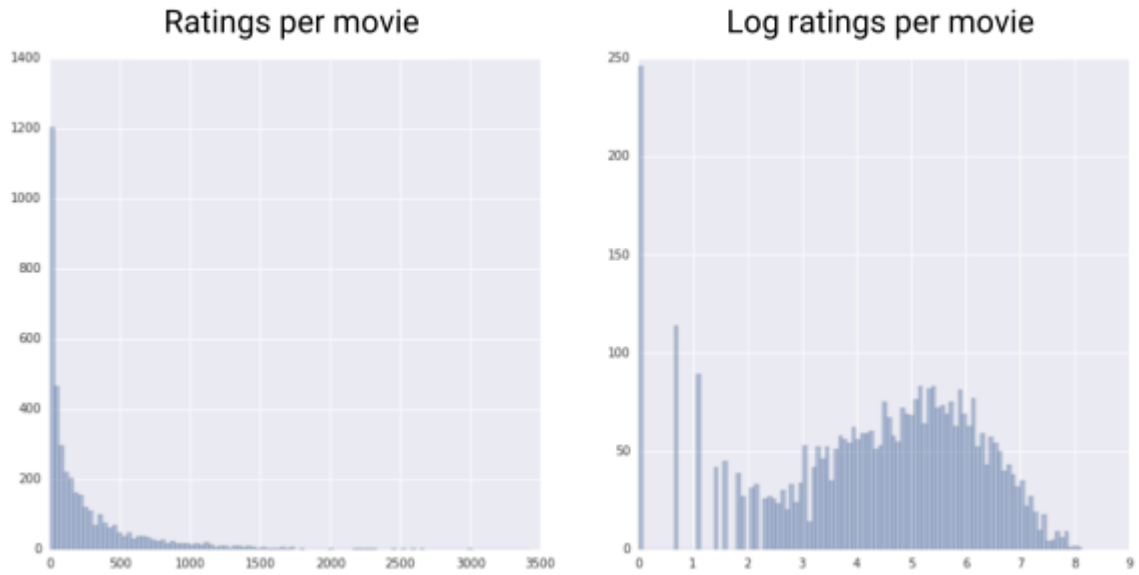

Figure 2. Comparing a raw distribution to its log.

4. **Standadization (Z-score normalization):**
   Standardization is another Feature scaling method where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero, and the resultant distribution has a unit standard deviation.
   The formula for calculating the z-score of a point, x, is as follows:

$$x' = (x - \mu)/\sigma$$

   - **Note:** μ is the mean and σ is the standard deviation.

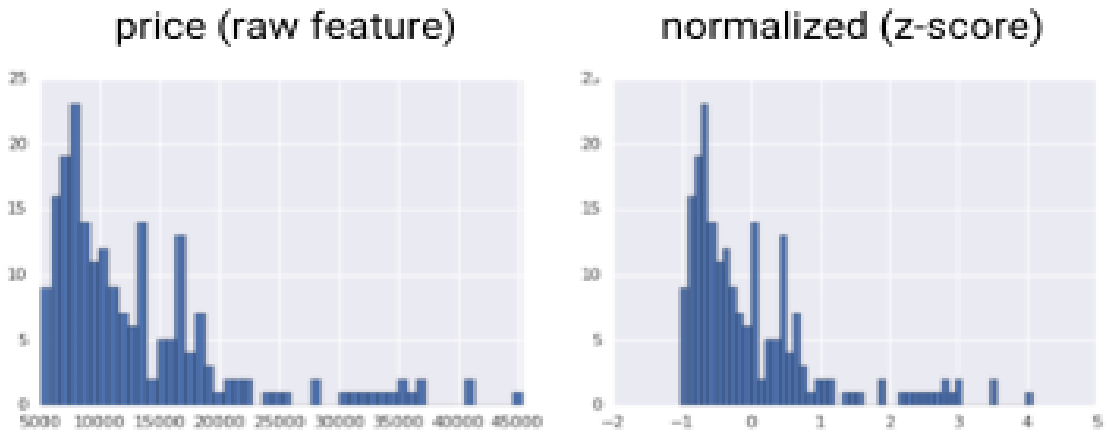

Figure 3. Comparing a raw distribution to its z-score distribution.

Notice that z-score squeezes raw values that have a range of ~40000 down into a range from roughly -1 to +4.

- ## Let's see how to standardize our data using Scikit-Learn library

Python sklearn library offers us with StandardScaler() function to standardize the data values into a standard format.

Syntax:

```
object = StandardScaler()
object.fit_transform(data)
```

According to the above syntax, we initially create an object of the **StandardScaler()** function. Further, we use **fit_transform()** along with the assigned object to transform the data and standardize it.

- **Note:** Standardization is only applicable on the data values that follows **Normal Distribution**.

Have a look at the below example:

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

dataset = load_iris()
object= StandardScaler()

# Splitting the independent and dependent variables
i_data = dataset.data
response = dataset.target

# standardization
scale = object.fit_transform(i_data)
print(scale)
```

Explanation:

1. Import the necessary libraries required. We have imported **sklearn** library to use the **StandardScaler** function.
2. Load the dataset. Here we have used the IRIS dataset from **sklearn.datasets** library. You can find the dataset here.
3. Set an object to the **StandardScaler**() function.
4. Segregate the independent and the target variables as shown above.
5. Apply the function onto the dataset using the **fit_transform()** function.

- Main question is when do we normalize our data?

When you don’t know the distribution of your data or when you know it’s not Gaussian, normalization is a smart approach to apply. This can be known when you make a figure out of your data too. When the data is right shifted or left shifted you can use a normalization.

- Conclusion

If you are working on ML projects, you probably have others using these methods to make their models' performance better. The best normalization technique is one that empirically works well, so try new ideas if you think they'll work well on your feature distribution.