Introduction to Transforming Data

ng Objectives

Identify types of data transformation, including why and where to transform.

Transform numerical data (normalization and bucketization).

Transform categorical data.

Feature engineering (/machine-learning/glossary#feature_engineering) is the process of determining which features might be useful in training a model, and then creating those features by transforming raw data found in log files and other sources. In this section, we focus on when and how to transform numeric and categorical data, and the tradeoffs of different approaches.

Reasons for Data Transformation

We transform features primarily for the following reasons:

1. **Mandatory transformations** for data compatibility. Examples include:
    - Converting non-numeric features into numeric. You can't do matrix multiplication on a string, so we must convert the string to some numeric representation.
    - Resizing inputs to a fixed size. Linear models and feed-forward neural networks have a fixed number of input nodes, so your input data must always have the same size. For example, image models need to reshape the images in their dataset to a fixed size.
2. **Optional quality transformations** that may help the model perform better. Examples include:
    - Tokenization or lower-casing of text features.
    - Normalized numeric features (most models perform better afterwards).
    - Allowing linear models to introduce non-linearities into the feature space.

Strictly speaking, quality transformations are not necessary--your model could still run without them. But using these techniques may enable the model to give better results.

Where to Transform?

You can apply transformations either while generating the data on disk, or within the model.

Transforming prior to training

In this approach, we perform the transformation before training. This code lives separate from your machine learning model.

👍 Pros

- Computation is performed only once.
- Computation can look at entire dataset to determine the transformation.

👎 Cons

- Transformations need to be reproduced at prediction time. Beware of skew!
- Any transformation changes require rerunning data generation, leading to slower iterations.

Skew is more dangerous for cases involving online serving. In offline serving, you might be able to reuse the code that generates your training data. In online serving, the code that creates your dataset and the code used to handle live traffic are almost necessarily different, which makes it easy to introduce skew.

Transforming within the model

For this approach, the transformation is part of the model code. The model takes in untransformed data as input and will transform it within the model.

👍 Pros

- Easy iterations. If you change the transformations, you can still use the same data files.
- You're guaranteed the same transformations at training and prediction time.

👎 Cons

- Expensive transforms can increase model latency.
- Transformations are per batch.

There are many considerations for transforming per batch. Suppose you want to **normalize** (/machine-learning/glossary#normalization) a feature by its average value--that is, you want to

change the feature values to have mean `0` and standard deviation `1`. When transforming inside the model, this normalization will have access to only one batch of data, not the full dataset. You can either normalize by the average value within a batch (dangerous if batches are highly variant), or precompute the average and fix it as a constant in the model. We'll explore normalization in the next section.

Explore, Clean, and Visualize Your Data

Explore and clean up your data before performing any transformations on it. You may have done some of the following tasks as you collected and constructed your dataset:

- Examine several rows of data.
- Check basic statistics.
- Fix missing numerical entries.

Visualize your data frequently. Consider Anscombe's Quartet (https://wikipedia.org/wiki/Anscombe's_quartet): your data can look one way in the basic statistics, and another when graphed. Before you get too far into analysis, look at your data graphically, either via scatter plots or histograms. View graphs not only at the beginning of the pipeline, but also throughout transformation. Visualizations will help you continually check your assumptions and see the effects of any major changes.

- Intro to Pandas (Machine Learning Crash Course Pre-req CoLab) (https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/prework/intro_to_pandas.ipynb?utm_source=ss-data-prep&utm_campaign=colab-external&utm_medium=referral&utm_content=pandas-colab)
- Working with Missing Data (Pandas Documentation) (http://pandas.pydata.org/pandas-docs/stable/missing_data.html)
- Visualizations (Pandas Documentation) (http://pandas.pydata.org/pandas-docs/stable/visualization.html)

erms:

ature engineering (/machine-learning/glossary#feature_engineering)

Previous

← Check Your Understanding (/machine-learning/data-prep/construct/sampling-splitting/check-your-understanding)

Next
Transforming Numeric Data (/machine-learning/data-prep/transform/transform-numeric) →

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 4.0 License (https://creativecommons.org/licenses/by/4.0/), and code samples are licensed under the Apache 2.0 License (https://www.apache.org/licenses/LICENSE-2.0). For details, see the Google Developers Site Policies (https://developers.google.com/site-policies). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2020-12-09 UTC.