Pre-processing and training data

Transforming your data and training a model for best performance is a holistic process.

Guy Maskall Follow
Jul 15 · 5 min read

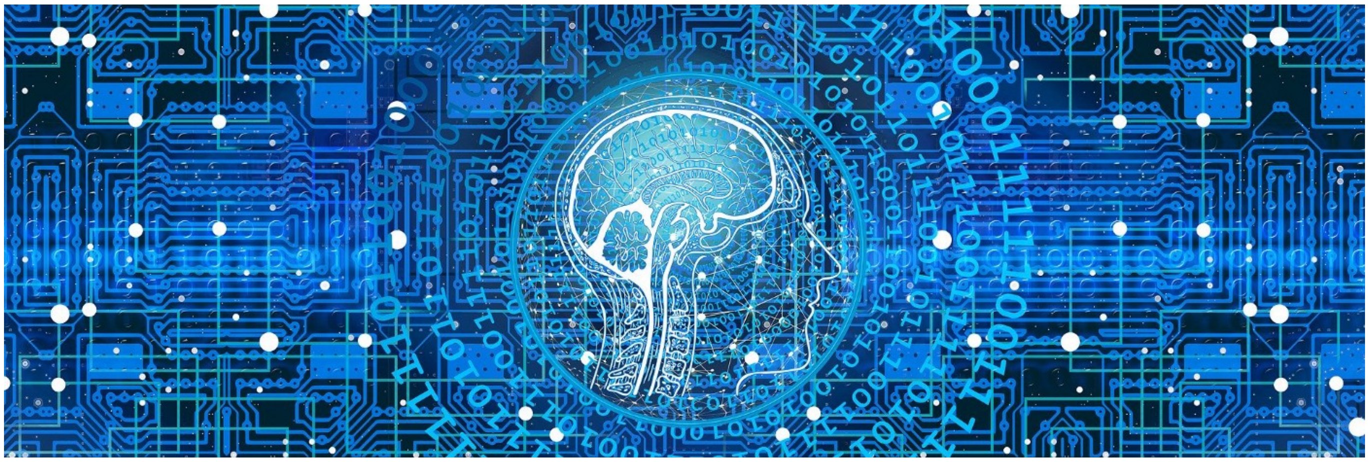

Image by Gerd Altmann from Pixabay

Pre-processing

So, what is pre-processing? Common examples include

- imputing missing values
- transforming the values (linearly or nonlinearly)
- Encoding categorical variables

Imputing missing values

There are often holes in your data. Someone didn't respond to a survey question. A sensor glitched. The data got lost. There are all kinds of missing value and causes thereof. Missing values can be denoted by a special 'value' of NaN (in Python, from NumPy) or NA (in R). But missingness isn't always denoted by these special markers. Negative values can denote missing data, for example -999 as someone's age. This can depend on the originating system or software as well as how the data was handled up to the point you received it. An empty string may or may not be classed as "missing" in your context.

An important question that you should have tackled by now is whether there seems to be any pattern in the missing data. You did explore this in your preceding EDA stage, right? Were those river level sensor readings missing because the station was flooded? Or are the missing values scattered randomly throughout the data, suggesting some more innocuous reason such as intermittent radio interference perhaps? If the former, then I hope you've added a new feature capturing the fact that the values were missing. Regardless, most machine learning algorithms don't like missing values so you need to take steps such as, well frankly, guessing what the value might have been. This is imputing.

There are any number of schemes for imputing missing values. You could use the mean of the remaining values. Or the median. Or take the last known value. Yes, picking the best strategy can involve trial and error.

Transforming the values

There are a few reasons to want to transform the values of your features. You may have highly skewed distributions, and a nonlinear transformation can tend to normalize these, which can help many algorithms. You might have a collection of features measured on very different scales, for example land elevation in feet and area in acres. Or people's weight in pounds and body temperature in degrees Fahrenheit, vs weight in kilograms and body temperature in degrees Celsius. Indeed, these units aren't so wildly different, but this leads us to another reason to want to transform feature values; algorithms that work off distances between points, and algorithms that regularize (penalize) their coefficients can produce very different results arbitrarily because of such differences in unit. Scaling features puts them all on an equal footing.

Encoding categorical variables

Algorithms like numbers. They're much less keen on strings (text). Much of data science is, arguably, the quest to get your data into an appropriate numerical representation that a machine learning algorithm can get to work on it. Rather than have a column named 'Animal' with the value of 'Cat' or 'Dog', you'd probably create a column 'Cat' with the value 0 or 1 denoting 'not a cat' or 'is a cat' and a column 'Dog' with value 0 or 1 denoting 'not a dog' or 'is a dog'. Even this relatively simple scheme can have some subtleties such as the need to drop one of those columns for models such as linear regression that include an intercept. This is to avoid the dummy variable trap.

Train–test split

If there are so many possible decisions to make before even training a machine learning model, how do we decide which to use? We could make our pre-processing, and our model, ever more and more complex to fit more and more arbitrary structure in our data. How do we know our pre-processing decisions were the right ones and we weren't just fitting to noise in our data? This is known as overfitting. The cornerstone of assessing whether a model is overfitting to the data it knows about is to not give it all the data when training it, and hold some back to then see how it performs.

A cunning variation on this is to further partition the training data into multiple splits and train on all but one, and test on the remaining one. By rotating the partition so held back, we end up with a number of estimates of performance on a validation set (where the validation set is different each time). This is cross-validation and is particularly useful in performing model selection. Of course, we can still keep a set of data held back from all of this, to ensure we have some data that our model has never seen. After multiple cross-validation rounds where multiple models were trialled, such a final test set is a useful final check of expected future performance.

Metrics

But how do we measure performance? There are a number of possible measures, or metrics, and the most appropriate one varies depending on the use case. If we're performing classification, for example "cat" or "dog", then a metric that captures the number of times we got the right answer vs the number of times we got it wrong is appropriate. But if we're doing regression, where we're trying to predict a continuous variable such as someone's salary, or a house price, then a measure that reflects how close we tended to be, on average to the true value is best. A common metric here is the mean absolute error, which, as the name suggests, is the average of the magnitudes of the errors.

Pulling it all together

Now we actually have all the ingredients we need. For our specific type of machine learning problem, we can pick a metric. Having a metric allows us to measure our degree of success (or failure). Having a train-test split gives us a means to apply that metric to a partition of data the algorithm hasn't been privy to. These together allow us to trial many different types of model and pre-processing steps and assess which one performed the best.

Hey Presto! If you've trialled a variety of pre-processing steps and algorithms, and applied an appropriate metric and assessed your entire process against a test set (e.g. via cross-validation), you've got a trained model. Specifically, you've established the combination of pre-processing steps and the nature of the algorithm.

About this article

This is the fifth article of a linked series written to provide a straightforward introduction to getting started with the data science process. You can find the introduction here, the previous article here, and the next article in the series here.