Features

Alex Sverdlov
alex@theparticle.com

1 Measurement Tuple

Pattern recognition starts with observations. For example, we may observe a patient's vitals by measuring the patient's temperature, blood pressure, pulse, weight, blood sugar, etc. These individual measurements result in a single measurement tuple, with each component being a measurement feature.

Note that the measurement tuple consists of what we decide to measure. For example, we could have also measured patient's height, drug history, outside temperature, the kind of shoes the patient if wearing, phase of the moon, and last week's lottery numbers. Some of these measurements may be relevant for patient care—while others create noise. Similarly, sometimes we may take the patient's vitals, but don't record everything: we might not perform the blood sugar test, etc.

2 Feature Vector

Of everything that was measured, we need to decide what items to select into our feature vector that we will use to train our classifier on (or cluster data). This is the step where business knowledge comes in useful. We need to throw away features that are seemingly irrelevant, and keep features that may potentially be useful.

We cannot keep everything: with many irrelevant variables it is possible some of them will correlate well with the target label—for example, we don't want the phase of the moon to influence our patient treatment decisions.

Getting this feature vector right is what most data scientists spend their entire day doing (well, that and data engineering; need to get data to a usable state before working on features). The other parts of pattern recognition can be done relatively mechanically.

3 Similarity

Once we have a vector of features we think are relevant to the task at hand, we need to define how we will measure similarity (distance) between such tuples. Refer to class notes on distance measures.

Keep in mind that feature vectors include items that are often not directly comparable to each other—and that most algorithms will co-mingle anyway.

For example, a linear classifier essentially takes a weighted sum of all inputs, irrelevant of whether it makes sense to add pounds to glucose levels to heart beats per minute to phase of the moon. What are the units of such an output from the classifier?

4 Normalization

It is often necessary to normalize the data, as it often has different scales and units. For example, is temperature in Celsius? Is height in inches? feet? meters?

Sometimes it may be useful to scale numerical data into some sort of range. For example, 0 to 1. For that, we can take:

$$x_{out} = \frac{x_{in} - min(X)}{max(X) - min(X)}$$

One problem with the above normalization is that outliers can skew the results (e.g. one large value makes the rest of the numbers tiny).

We may bucketize the range of a numeric field. Perhaps store value 1 through 10. Refer to notes on quantiazation.

Another common technique is Z-scores:

$$x_{out} = \frac{x_{in} - avg(X)}{stddev(X)}$$

These will generally be in -4 to 4 range, depending on how wide spread the X is.

Sometimes it is enough to take the log of the input:

$$x_{out} = \log(x_{in})$$

Perhaps subsequently followed by the z-score logic.

5 Dates & Timestamps

Dates and timestamps can often be treated as numeric data. Sometimes it is required to discretize them a bit, to convert timestamps to seconds since midnight, or some other point. To work with month-by-month data instead of day-by-day data, etc.

6 Non-Numerical Data

One way to deal with non-numerical (string) data is to record the string as-is. This allows for equality comparisons.

Sometimes that is not appropriate. If there are a few different string values indicating different things, then creating 'one-hot-vector' may be appropriate. A one-hot vector is a sequence of bits, one bit for each distinct value of the string feature. Exactly one bit is on for any given tuple—indicating the value of string.

Such bit sequences may be used to store numeric values too. An integer feature with 10 distinct values can be represented by 10 bits.

7 Ratios

Sometimes the raw measurement items are not directly comparable, even when normalized. But perhaps ratios may be directly comparable. For example, a stock priced at \$50 earned \$5 last year, while a stock priced \$200 earned \$10.

Each of these are not directly comparable (as stock price is not relevant to performance of the company), and the amount the company earns could just be indicative of the size of the company.

But we may compare Price/Earnings (P/E) ratio. First company has P/E of 10, while the other one has P/E of 20. These numbers, 10, and 20 might make good features for someone evaluating performance.

Note that most ratios also make sense when inverted. e.g. E/P ratio is earnings yield, 5/50, or 10% annual yield (the \$50 of stock gets you \$5 in earnings).

8 Missing Values

Some tuples will have missing values. It is important to treat these as "we do not know" instead of having a well defined value of "null".

Sometimes we may be able to fill in the values: If measurements are happening in sequence, perhaps we can pull the value from the previous populated record.

If values do change, perhaps doing linear interpolation between previously known value and next known value is appropriate.

Another approach is k-NN: find the closest tuple that has the attribute populated and use that value.

In some scenarios, it may be appropriate to use neighboring features to interpolate (or fill in with average of neighbors). For example, a dead pixel in a camera sensor may be fixed with a median filter.

For non-numeric attributes, we may stick in the most common non-numeric values. Another alternative is to replicate the measurement tuple with every single possible value that may show up.

All that said, if there is plenty of data, it is often easier to just drop the records with missing fields.

9 Noise

Sometimes observations will have noise/errors. For example, temperature measurement returns impossible values due to faulty sensor. Or corrupt message indicates unlikely prices for index components.

If there is danger of such noise causing damage, we need to detect such outliers and either remove them or smooth them out.

If data is very spiky, we may decide to make do with a moving average (or moving median) instead of the raw values.

Of course there is danger of a spike in the data being an actual signal for something critical.

10 Variable Length Measurement Tuple

So far, we've assumed that the measurement tuple is fixed size. It certainly makes comparing them simpler. But lots of data sources create a stream of items, not measurement tuples of a fixed size.

For example, text data, voice data, video data, etc. Text gives us one word at a time. Each word builds up on words that came before it.

One way to handle such situations is to move a fixed window over the measurement vector, and always look at n items back.

When we are sliding such a window across the sequence, we may be incrementing by 1, or any other number.

Some learning algorithms do not require this explicit window, but maintain internal memory of previous instances—allowing for inputs one at a time (instead of a window at a time).

11 Deep Learning

A note on the "future".

A lot of deep learning approaches do close to zero feature engineering (except to define the measurement tuple). It seems, if there is plenty of data, the deep neural network can learn what features are important, and extract relevant information from them. So long term, feature engineering may not be as critical to the success as it is right now.