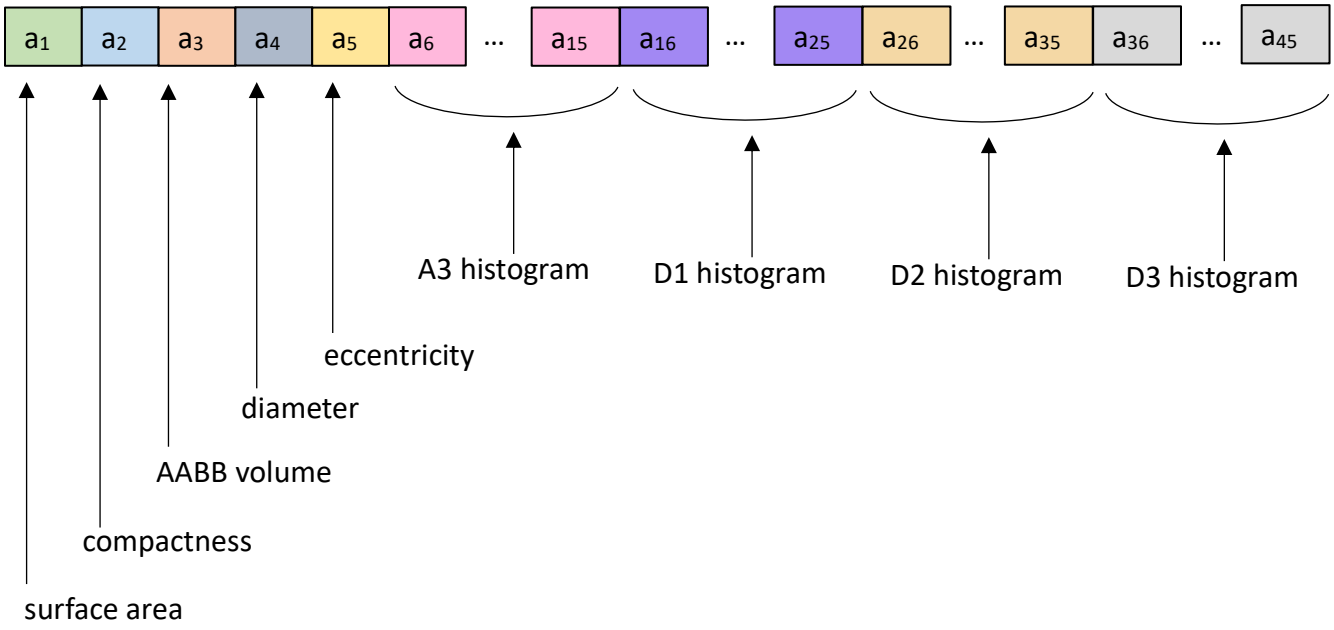
Technical tips: Step 4

1. Computing distances between two feature vectors

Let A = (a_1,….a_n) be a feature vector. Typically A contains elements that describe different features, each having its own nature. An example is below:



Here, we have a feature vector of 45 elements. The first five elements (a_1…a_5) indicate elementary features, e.g., surface area, compactness, AABB volume, diameter, and elongation. Next, we have 10-bin histograms for the descriptors A3 (a_6..a_{15}), D1 (a_{16}..a_{25}), D2 (a_{26}..a_{35}), and D3 (a_{36}..a_{45}).

Now consider two such vectors A and B. How to compute the **distance** between them? There are several problems of, and solutions for, that.

(1) Different feature ranges

The first problem relates to the fact that the involved features may have different ranges. As such, simply using the values a_1..a_{45} as above could make features which have *high ranges* count more than features which have *low ranges*.

A solution for this is to normalize the features so they have the same range. This should be done differently for single-value features and for multiple-value (e.g. histogram) features.

- **Single-value features:** Normalize by standardization, which is less sensitive to outliers than min-max normalization.
- **Histogram features:** Normalize by dividing by the area (element count).

The above two normalizations bring, statistically speaking, both single-value and histogram features in the (absolute-value) range [0,1]. By statistically speaking, we mean that single-value features *could* still be outside of the [0,1] absolute-value range for a *few* samples which have really outlier values. We arguably like to keep these as such, since they describe truly different data values.

(2) Different distance ranges

You may have noticed that there is an asymmetry in how we normalize single-value and multiple-value features: Single value features are normalized considering the *entire set of feature-values* in the process (when computing the standard deviation). Histogram features are normalized *independently per histogram*: We don't compute anything like the 'standard deviation histogram'.

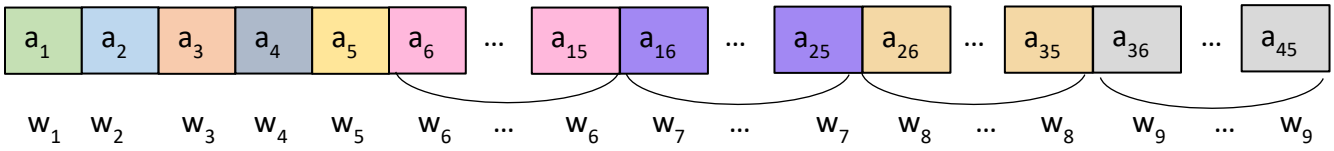
As such, there is a problem: After the normalization (1), the *range of the distances* between feature-values for the different features a_1, a_2, … can be *very different*. For example

- The range of eccentricity (elongation) values can span the whole spectrum [0,1] after normalization. Indeed, our shape database can have all shapes ranging from close to a ball to close to a pencil. Consequently, the range of **distances** between elongations can also span the whole spectrum [0,1].
- The D4 histogram values for a set of shapes can be very similar. Hence, the range of **distances** between D4 histograms (computed e.g. with the EMD metric or any similar histogram-distance) can be much smaller than 1. For area-normalized histograms, note that a distance of 1 between two histograms is *huge* and would likely never be attained in practice.

If we do not address this problem, then features having a *small range of distances* will count little in the overall distance function.

There are multiple ways to address this problem:

- **Feature weighting:** For every set of feature-values which, together, form a feature, add a weight w_i (so that the sum of all w_i equals one). The image below shows this on the example feature-vector at the beginning:



  Then, adjust the weights w_i, relative to each other, so as to boost the variations of the features having small ranges. In practice, this would typically mean leaving $w_1 \ldots w_5$ equal to each other and making them smaller than the weights $w_7 \ldots w_9$ used for the narrow-range histograms. The advantage of this method is that it is very simple to implement. However, playing around with weights can cost quite some time until the desired results are achieved.

- **Distance weighting:** Rather than weighting the feature values, we can go to the 'root' of the problem and weigh the distance values themselves. This is exactly what the standardization does for the single-value features: It actually considers the *spread* of values (by measuring their standard deviation, which is a distance) and normalizes them by this spread. We can generalize this also for multiple-value features such as histograms.

  Consider e.g. the feature-vector elements $a_6 \ldots a_{15}$ which, together, create the D1 descriptor. We can them compute all distances $d(a_6 \ldots a_{15}, b_6 \ldots b_{15})$ between the elements 6..15 of two feature vectors A and B over an entire shape database. These give all possible values for the *distances* between D1 descriptors over our database, computed by any desired distance function (Euclidean, EMD, cross-bin matching, etc). Then, we can standardize these distances, just before combining them with the other feature distances (that is, for a1..a5 and a16…a45) to yield the final distance. This way, even small variations in D1 will count similarly to e.g. large variations in a_1.

Note that this histogram standardization is strongly dependent on the quality of the extracted descriptors: If, for instance, we have a poor computation of D1, which yields more or less the same values for all shapes, then the standardization above will artificially *amplify* tiny differences in D1 which likely mean nothing, leading to poor matching.