7. Exploratory data analysis II

7.1 Exploratory analysis of coordinates

In certain cases, a sequential binary partition can be quite informative in itself. Sometimes it is possible to form a meaningful partition based on knowledge of the data, while sometimes such partitions arise from the type of data exploration presented in the previous chapter. In either case, analyzing the coordinates may provide further insights.

7.1.1 Correlation analysis

Because the coordinates of the basis are orthogonal and real vectors, we can apply normal real multivariate analysis to the ILR transformed data as discussed in Chapter 3. For instance, given a composition $\mathbf{x}$ and an orthonormal basis $\mathbf{\Psi}$,

$$\text{ilr}(\mathbf{x}) = \text{clr}(\mathbf{x}) \cdot \mathbf{\Psi}^T, \tag{7.1}$$

gives the logratio-coordinates in the basis $\mathbf{\Psi}$. Recall from Chapter 1 how the standard Pearson correlation coefficients,

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sqrt{\text{var}(X)\text{var}(Y)}}, \tag{7.2}$$

gives a negative correlation bias when applied to compositional data. This bias can be expressed by the relation,

$$\text{cov}(x_1,x_2) + \text{cov}(x_1,x_3) + \ldots + \text{cov}(x_1,x_D) = -\text{var}(x_1), \tag{7.3}$$

where $\text{cov}(\cdot,\cdot)$ is the covariance and $\text{var}(\cdot)$ is the variance. The meaning of Eq. 7.3 is that when the covariance between a part and another increases, the variance within the part itself must decrease by the same amount.

We can obtain proper correlations by either using ILR-coordinates directly in Eq. 7.2 or from the logratio variation matrix **T**, defined in def. 6.1.2 as $t_{ij} = \mathrm{var}(\ln[x_i/x_j])$. If we already have **T**, we can obtain the covariance matrix **S** *in the basis* $\mathbf{\Psi}$ from,

$$\mathbf{S} = -\frac{1}{2}\mathbf{\Psi} \times \mathbf{T} \times \mathbf{\Psi}^T. \tag{7.4}$$

From this, we can calculate the correlation coefficient matrix **R** using Eq. 7.2,

$$r_{ij} = \frac{s_{ij}}{\sqrt{s_i s_j}}. \tag{7.5}$$

The interpretation of **R** is straight forward: r_{ij} close to 0 means little or no correlation, while r_{ij} close to ± 1 means a linear association between the i'th and the j'th balance coordinate.

It should be noted here that even though it isn't possible to straight up calculate correlations between parts in compositional data, there are algorithms that attempt to estimate correlated parts. SparCC (Friedmann & Alm, PLOS Computational Biology, 2012) is a reasonably successful method that works on large compositions with relatively few correlations.

7.1.2 Balance dendrogram

A useful way to visualize a sequential binary partition basis is by using a balance dendrogram. A balance dendrogram shows the basis in a familiar tree structure. In addition to showing the partitioning, it also shows the sample variance and the geometric mean of each balance, represented by branch length and branch split inception point.

As an example, we can construct a sequential binary partition basis for the protein data set from Chapter 6. One such partition could be,

	Red meat	White meat	Eggs	Milk	Fish	Cereals	Starch	Nuts	Vegetables
1	1	1	1	1	1	-1	-1	-1	-1
2	1	1	-1	-1	1	0	0	0	0
3	1	1	0	0	-1	0	0	0	0
4	1	-1	0	0	0	0	0	0	0
5	0	0	1	-1	0	0	0	0	0
6	0	0	0	0	0	1	-1	1	-1
7	0	0	0	0	0	1	0	-1	0
8	0	0	0	0	0	0	1	0	-1

The choice here is to separate vegan from non-vegan in the first row, vegetarian from non-vegetarian in the second row, fish from meat in the third row, and finally meat in the fourth row. In the sixth row, we rather arbitrarily split cereals and nuts from starch and vegetables, both of which are split in rows seven and eight. Using this basis, we can calculate ILR coordinates, from which we can calculate the coordinate means and variances. The result is displayed in a balance dendrogram in Fig. 7.1. Long vertical lines (blue) mean large variance, whereas short lines mean low variance, i.e., that the coordinate is close to constant across the data set.

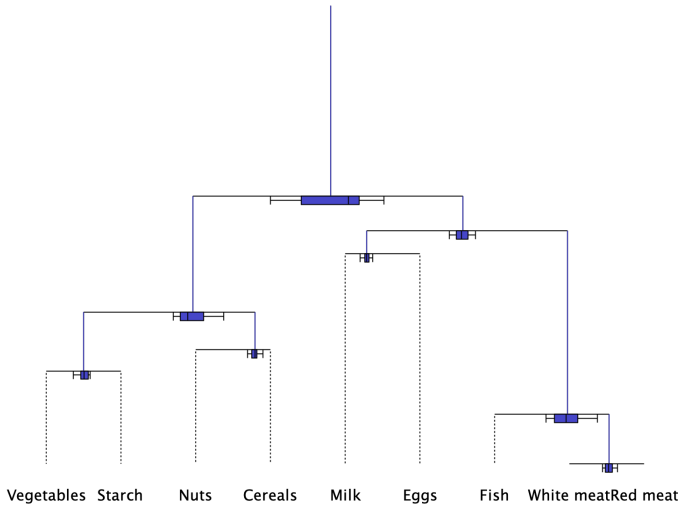

Figure 7.1: A balance dendrogram for a specific sequential binary partition of the protein data set.

7.2 Principal component analysis revisited

Principal Component Analysis, which was introduced in the last chapter, is an example of linear dimensionality reduction. Let us briefly consider the mathematics behind PCA. The PCA is a visualization of the eigenvectors belonging to a matrix that represents a set of compositions. Eigenvectors and the corresponding eigenvalues are defined as

$$A\mathbf{x} = \lambda\mathbf{x}, \tag{7.6}$$

where A is the matrix representation of the data set, $\mathbf{x}$ are the eigenvectors, and λ are the eigenvalues. The interpretation of this equation is that an eigenvector is a vector that, when multiplied by a matrix, does not change direction – only length, and the change in length, the scaling, is determined by the corresponding eigenvalue. The larger the eigenvalue, the more the corresponding eigenvector will be scaled, and we call the two (or sometimes three) eigenvectors with the largest eigenvalues the principal components. A PCA bi-plot is simply the samples and the parts projected onto a plane (or 3-space) spanned by the principal components.

In general, eigenvectors are a linear combination of the parts and therefore can not uniquely be associated with a single part, but we can construct an example where the eigenvectors exactly line up with the basis vectors of the data. Consider a 2D data set of 100 points sampled from the function $f(x) = 5\sin(3x),\ x \in \{-10, 10\}$. The samples are shown in Fig. 7.2. The resulting data set is not compositional (the sum of x- and y coordinates has no meaning), but it nonetheless serves our purpose. By construction, the eigenvectors of this data set coincide with the x- and y-axes, which can easily be seen from the fact that if the frequency (x-axis) or amplitude (y-axis) is changed, points will move only along those two directions. A PCA plot should therefore have the feature vectors along the principal components, which indeed it has, as can be seen in Fig. 7.2. Because the data is two-dimensional, we can describe 100% of the variation using just two principal components. However, if we allowed a non-linear principal component, particularly a

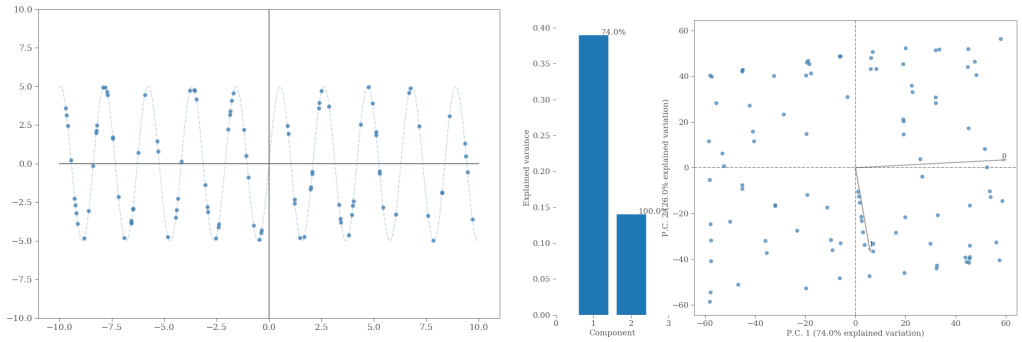

Figure 7.2: Left panel shows 100 random points drawn from a sine function. Right panel shows the PCA of the x and y coordinates of those points.

principal component along the direction of $5\sin(3x)$, the we could describe 100% of the variation using only 1 principal component. In other words, if we use a nonlinear dimensional reduction method, we can describe more variation using the same number of components or the same variation using fewer components. A number of non-linear dimension reduction methods exist, many of which can be seen as generalizations of PCA. In this chapter, we will learn about one of these methods, namely the self-organizing map.

7.3 Self-organizing maps

A self-organizing map (SOM) is a type of artificial neural network (ANN), trained using unsupervised learning, to produce a two-dimensional discrete approximation of the input space from which the training samples are drawn. The purpose is to create views of high-dimensional data that preserve as much of the high-dimensional structure as possible, and it is primarily a visualization technique. The number of neurons in the ANN determines the type of map that is produced. If the number n of neurons is much smaller than the number M of samples (n$\ll$m), the result is akin to k-means clustering (see Sect. 7.4) with k=n, whereas if n$\gg$m, the result is more akin to a topological map. In the special case of a one-dimensional SOM, the output can be interpreted as a non-linear PCA, as illustrated in Fig. 7.3. In this figure, the first (linear) principal component is shown in blue, and it falls along the direction of the largest variance in the data (grey points). The red squares are a one-dimensional neuron grid, which approximates the data much better than the linear component.

The neurons of the ANN are arranged in a two-dimensional grid, either as a rectangular grid or a hexagonal grid. The grid can either be finite or it can have periodic boundaries so that it describes the surface of a torus. Each cell in the grid represents a neuron, which again is described by a vector with the same length as the input vectors (the samples). These neuron vectors (sometimes called weights) are initialized with random values.

The ANN needs to be trained, which is done by feeding it a set of training samples – usually the entire data set that we wish to visualize. In an iterative manner, going through each sample, the ANN identifies the neuron that is most similar to the sample and moves it as well as the adjacent neurons towards the sample. The distance we allow the neurons to move decreases at each iteration, and the training continues until either after a predefined

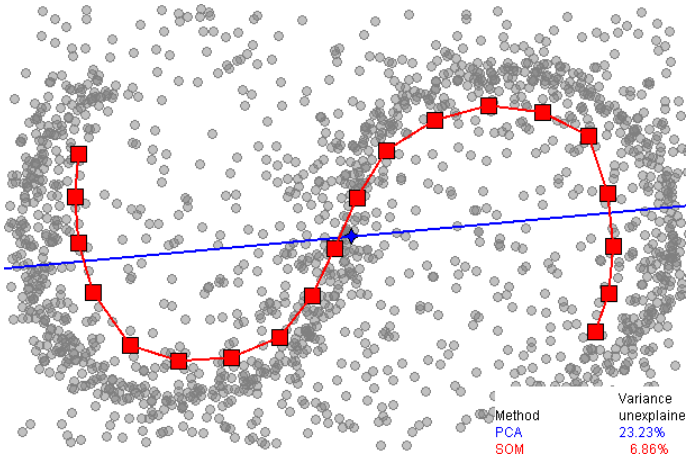

Figure 7.3: Comparison of PCA to SOM: blue shows the first principal component of the data, which is a linear approximation along the direction of largest varaince. Red is the 1D SOM, which can be interpreted as a non-linear principal component. Source: wikipedia

set of iterations or until the map has converged, i.e., the neurons don't move anymore. The result is that the network of neurons gets stretched and bends into the shape of the data, but its topology is preserved, so there is no twisting or rearranging of the neurons. Because the ANN determines the distance between neurons and samples using the Euclidean metric, we need to provide CLR-transformed values as input when mapping compositional data.

Once the training has converged and the network is molded into a shape that approximates the data, we visualize the network by coloring the map according to the distance from the neurons to their neighbors. We can then map the samples onto this map by plotting a sample label next to the neuron that is closest to the sample. If the number of neurons is small, many samples will map to the same neurons, and we will have a clustering algorithm, while a large number of neurons will provide a topological map that shows the samples locations relative to each other.

Figure 7.4 shows the European protein consumption data on a SOM. In this case, the SOM was made using a 100×100 rectangular map with finite boundaries. The dark ridges indicate greater neuronal distances, while the light patches describe neurons that are closer together. The SOM clearly separates the southern European countries from the northern, and it separates the western Mediterranean countries from the southern East Block countries. France, which we by random definition label as southern, is placed among the northern countries, while the opposite is true for Hungary. The four Scandinavian countries are grouped together inside their own light patch while still being close to Germany and the UK. The SOM tends to place countries closer that are physically close to each other (France and Belgium, Denmark and Germany, Portugal and Spain, etc.), almost preserving physical topology. Albania is clearly seen to be an outlier but is still placed among the other south-eastern countries. We can also see from the SOM that the central European countries are quite similar, regardless of whether they belong to the eastern or western block.

We can use such a map to predict the country of origin for a person if we know the protein consumption composition of that person. We simply map the CLR-transformed

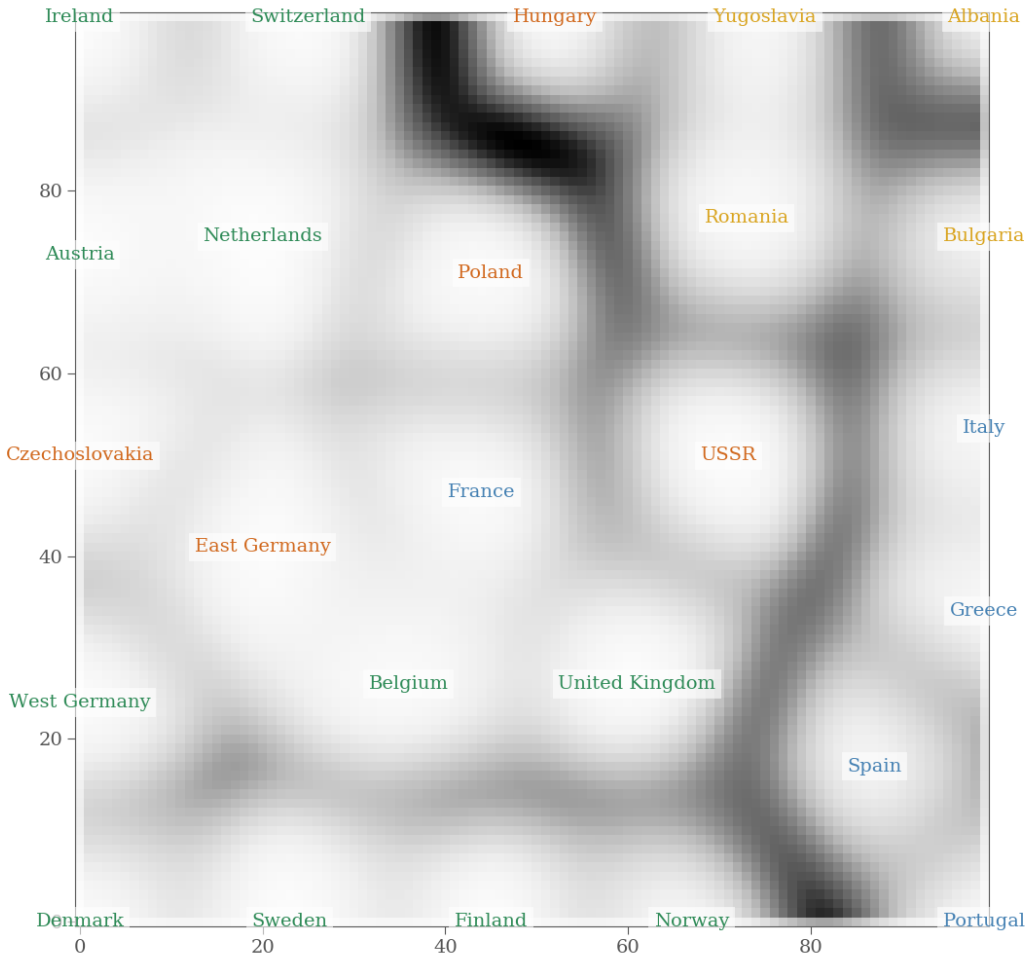

Figure 7.4: Self-organizing map of the protein consumption data. Black ridges correspond to large neuron distances.

composition to the SOM, which gives us the neuron that most resembles the person. By identifying that neuron on the map, we can predict which part of Europe and possibly which country the person lives in.

There are a few downsides to SOMs, though. Due to the fact that they need to be trained and converge, they can be quite computationally expensive to make. But a more problematic aspect is that the algorithm is stochastic by nature, and SOMs are therefore difficult to reproduce. This can, to some extent, be remedied by initializing the weights, not with random numbers, but by sampling from the subspace spanned by the first two principal components. This provides a more robust starting point, which already traces the data, and the resulting map will be more consistent with consecutive runs, but there is still some randomness involved. Moreover, a large number of parameters can be adjusted, for instance the learning rate, which makes the resulting maps a bit ambiguous.

We can try and produce a SOM in the regime where the number of neurons is small compared to the number of samples, so that the SOM resembles a k-means clustering of the data. If we chose 4 neurons, corresponding to 4 clusters, we find that the SOM divides

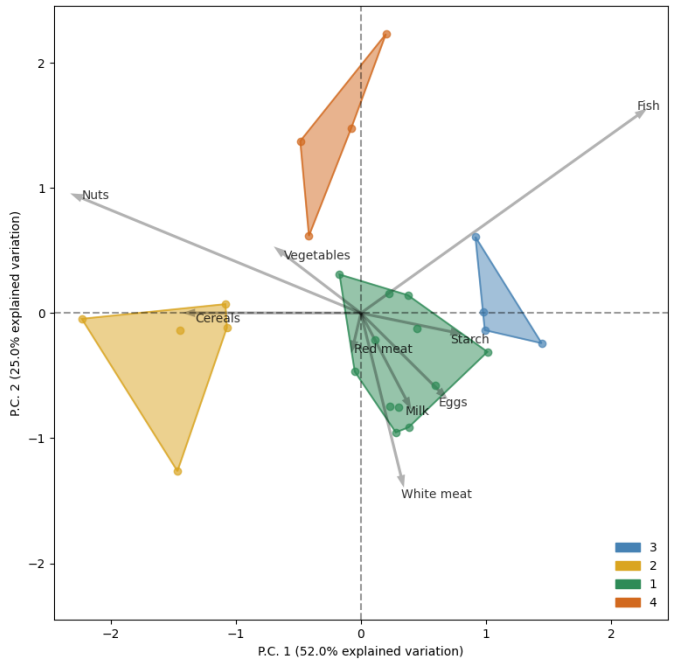

Figure 7.5: A PCA biplot of the protein data with samples colored according to the 4 clusters defined by the SOM. The convex hull has been drawn for each group to show that there is no overlap.

the data set into the following partition:

1	2	3	4
Austria	Albania	Denmark	Greece
Belgium	Bulgaria	Finland	Italy
Czechoslovakia	Hungary	Norway	Portugal
East Germany	Romania	Sweeden	Spain
France	Yugoslavia		
Ireland			
Netherlands			
Poland			
Switzerland			
USSR			
United Kingdom			
West Germany			

Here, cluster 2 consists of the south-eastern countries, cluster 3 is Scandinavia, cluster 4 is the Mediterranean countries, and cluster 1 is all the rest. If we chose five clusters, we would split the USSR off of cluster 1 into its own cluster with Romania. If we recreate the PCA biplot from Chapter 6 and color our samples according to the 4 clusters, we see that the SOM splits the samples nicely into 4 distinct groups, which is shown in Fig. 7.5.

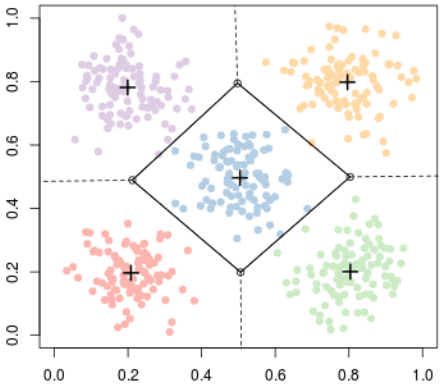

Figure 7.6: The result of k-means clustering is a Voronoi tessellation.

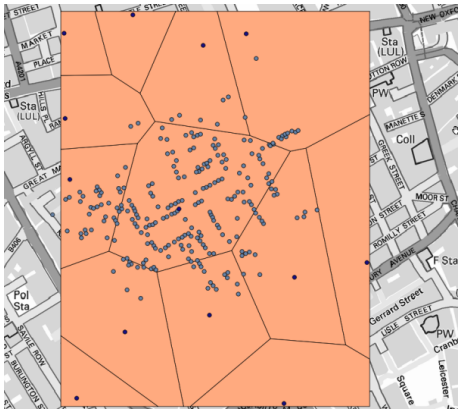

Figure 7.7: John Snow's map of 1854 cholera cases in London, with a Voronoi diagram superposed.

7.4 k-means clustering

One commonly used technique to identify clusters and particularly to define the membership of clusters is k-means clustering. This technique partitions n samples into k clusters so that each sample belongs to the cluster with the nearest mean. The result is a so-called Voronoi tessellation with the cluster means as generators.

One of the earliest and most famous, while also coincidental, uses of k-means clustering and Voronoi diagrams was when the British doctor John Snow discovered that the source of a London cholera outbreak was the water pump in Broad Street. He realized that the vast majority of cases lived closer to that pump than to any other water pump in London, and therefore that the one thing that all cases had in common was that they all collected their water in the same place. A modern rendering of the Voronoi diagram with the location of the water pumps as generators is shown in Fig. 7.7, where it is easily seen that almost all cases lived within the same Voronoi cell.

We can use the same approach when trying to identify samples with similar properties in a data set. Sometimes we know how many clusters we expect (like John Snow knew the number of water pumps), but most often we do not know how many clusters we have. In that case, if we cannot visually determine the appropriate number of clusters, we must repeat the clustering algorithm for an increasing number of clusters and determine when the explained variance no longer increases, at which point we have the most probable number of clusters. How to determine the variance between clusters is the topic of the next chapter.