26 Case Study 4: Clustering

Objectives	26-1
Theory and Examples	26-2
Description of the Forest Cover Problem	26-2
Data Collection and Preprocessing	26-4
Selecting the Architecture	26-5
Training the Network	26-6
Validation	26-7
Data Sets	26-10
Epilogue	26-11
Further Reading	26-12

Objectives

This chapter presents a case study in using neural networks for clustering. In clustering problems, you want a neural network to group data by similarity. For example, market segmentation can be done by grouping people according to their buying patterns, data mining can be done by partitioning data into related subsets, and bioinformatic analysis can be done by grouping genes with related expression patterns.

In this chapter, we will apply clustering to a problem in forestry, in which we would like to analyze forest cover types. We will use the self-organizing feature map network of Chapter 16 to perform the clustering, and we will demonstrate a variety of visualization tools that can be used in conjunction with the SOFM.

Theory and Examples

This chapter presents a case study in using neural networks for clustering. In clustering problems, we generally don't have a set of network targets available, so clustering networks are trained by unsupervised training algorithms. Instead of training a network to produce a desired response, we want to analyze a data set to look for hidden patterns. There are many application areas for clustering. It is widely used in data mining, in which we analyze large data sets to identify similarities within subsets of the data. It is used in city planning, when town councils apportion regions of the city into areas of similar home type and land usage. It is used in image compression, in which a small set of prototype sub-images are identified and combined to represent a large collection of images. It is used in speech recognition systems, in which speakers are clustered into categories in order to simplify the problem of speaker-independent recognition. Clustering is used by marketers to identify distinct groups in their customer bases. It has also been used to organize large bibliographic data bases so that related material can be quickly accessed.

The neural network that we will use in this application is the self-organizing feature map (SOFM), which we introduced in Chapter 16. This clustering network has a unique attribute that enables us to visualize large data sets in many dimensions. We will focus on that visualization capability in this case study.

Description of the Forest Cover Problem

An important job of the forest service is to maintain accurate natural resource inventory information. One key characteristic that is recorded is the type of forest cover found in wilderness areas. This type of data can be expensive to collect, since it generally requires on-site inspection or estimation from remotely sensed data. [BlDe99] describes how forest cover type can be predicted from independent variables that can be more easily obtained. In this chapter, we will use the data described in that paper to perform a clustering analysis. We will demonstrate how an analysis of the data using an SOFM can allow us to visualize the high-dimensional space of independent variables and identify relationships between the forest cover types.

Ten independent variables that can indicate forest cover type are shown in Table 26.1. (There were 12 variables used in [BlDe99], but for ease of presentation, we selected only the first 10 for this case study.) These variables can be measured or estimated much more easily than forest cover type. We want to use the SOFM to find out whether these variables can be used to cluster the data in such a way as to separate regions with different forest cover types.

Variable Number	Description	Units
1	Elevation in meters	meters
2	Aspect in degrees azimuth	azimuth
3	Slope in degrees	degrees
4	Horz Dist to nearest surface water	meters
5	Vert Dist to nearest surface water	meters
6	Horz Dist to nearest roadway	meters
7	Hillshade index at 9am, summer solstice	0 to 255 index
8	Hillshade index at noon, summer solstice	0 to 255 index
9	Hillshade index at 3pm, summer solstice	0 to 255 index
10	Horz Dist to nearest wildfire ignition points	meters

Table 26.1 Description of Independent Variables

The forest cover types of interest in [BlDe99] are shown in Table 26.1. The data set we will use for this case study contains information on cover type, but we will not use this as part of the training process. We will use it to test the clustering ability of the SOM.

Label	Name
0	Krummholz
1	Spruce/Fir
2	Lodgepole Pine
3	Ponderosa Pine
4	Cottonwood/Willow
5	Aspen
6	Douglas-fir

Table 26.2 Forest Cover Types

Data Collection and Preprocessing

The data used in this study came originally from [HeBa99]. It contains the forest cover type for 30 x 30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. The original data set contained 581,012 observations of 12 independent variables and the forest cover type. We have used the first 20,000 observations, and we have used only the first 10 independent variables, which are described in Table 26.1. The forest cover types are given in Table 26.2. As mentioned previously, we did not use these for training.

For supervised learning, as demonstrated in the previous three chapters, after the data is collected, the next step is to divide the data into training, validation and test sets. For unsupervised learning, we don't generally divide the data in this way, because there is no need for a validation set to stop the training. Competitive training is typically performed for a fixed number of iterations. We use the entire data set for training.

The next step is to normalize the data. The data were scaled using Eq. (22.1), so that the inputs were in the range [-1,1]. (For the SOFM, data can also be scaled using Eq. (22.2), so that the input variables have a mean of 0 and a variance of 1.)

Before proceeding to train the network, it is often useful to view the input data. One convenient format for this is the scatter plot. Figure 26.1 illustrates a set of scatter plots among the input variables 7, 8 and 9. The diagonal plots in this figure are histograms for these three input variables, and the off-diagonal plots are the scatter plots. (We only show three of the variables in this figure because of the limits of the page size.)

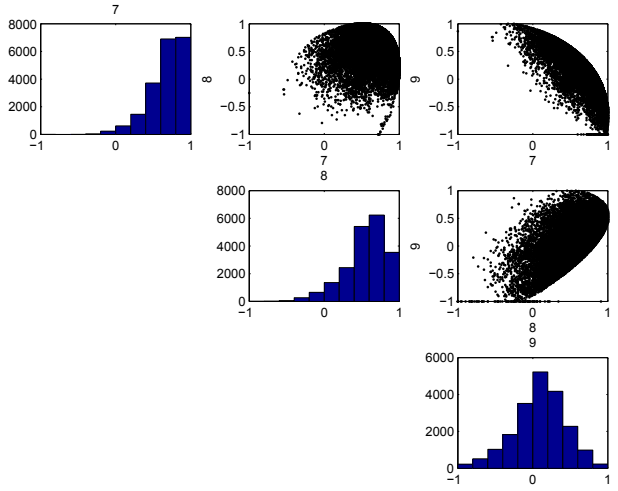

Figure 26.1 Scatter Plots for Input Variables 7, 8, 9.

There are several things we can look for in Figure 26.1. First, we want to see how well the data is scattered throughout the range. If some variables show little or no variation, then we would remove them from the analysis. We also look for correlation between the variables. For example, if the points in the scatter plot fell exactly along a line, then we would know that the two variables were linearly related. There would be no need to use both variables in the analysis. From Figure 26.1 we can see that there is some correlation between the variables, but they are not linearly dependent.

Selecting the Architecture

We will use the SOFM network, described in Chapter 16, to perform the clustering for this case study. The specific architecture is often selected based, in part, on the number of data points, so that there will be a reasonable amount of data associated with each prototype vector. (Recall that each row of the weight matrix represents a prototype vector. An input is associated with the prototype vector to which it is closest.) As the data set size increases, the number of neurons should increase as well, although not as rapidly. A rule of thumb is to have the number of neurons increase as the square root of the number of data points.

Figure 26.2 shows the architecture of the network we selected. We have 10 input variables (defined in Table 26.1), and we are using 150 neurons. The feature map is 15x10, and it uses an hexagonal arrangement of neurons. This means that each internal neuron will have six neighbors.

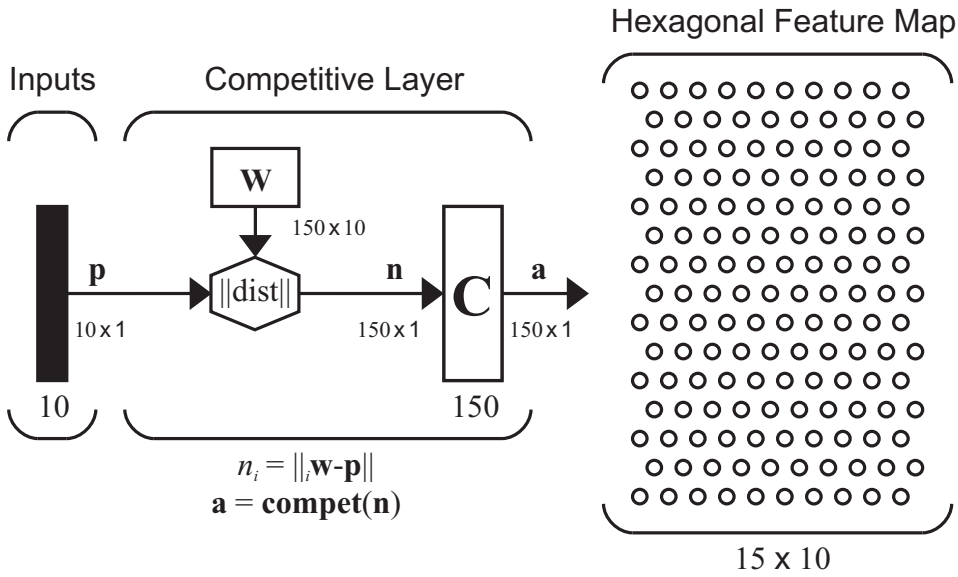

Figure 26.2 SOM Network Architecture

After the network has been trained, we will analyze the results to determine if the network architecture is satisfactory. In practical cases, we often try several different architectures. Unlike with supervised training, in which we have a clear performance measure (normally sum square error), there is no firm criterion for best performance in SOFM networks. Often

what we are looking for is insight into the data set. It is somewhat of an art to selecting the best architecture and training regime for SOFM networks. This will become more clear when we analyze the results of the trained network in a later section.

Training the Network

Before beginning the training, the weight vectors (rows of **W**) were initialized using what is called *linear initialization* [Koho95]. First, a covariance matrix of the input vectors was computed. Then, the two eigenvectors of this matrix having the two largest eigenvalues were found. The rows of **W** were then assigned by taking the average of the input vectors and adding linear combinations of the two eigenvectors. This places all of the initial weight vectors in the space spanned by the two eigenvectors. This initialization process produces quicker training convergence than when using a purely random weight initialization. (It is also possible to randomly select input vectors from the training set to be the initial weight vectors.)

Recall from Eq. (16.21) the SOFM learning rule, which we repeat here:

$$\begin{aligned} {}_i\mathbf{w}(q) &= {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)) \\ &= (1-\alpha){}_i\mathbf{w}(q-1) + \alpha\mathbf{p}(q) \end{aligned} \qquad i \in N_{i^*}(d) \tag{26.1}$$

where i^* is the index of the winning neuron, and

$$N_i(d) = \{j, d_{ij} \le d\} \tag{26.2}$$

defines the neuron neighborhood. For this case study, we have used a batch form of the algorithm, in which all of the inputs in the training set are applied to the network before the weights are updated. To develop the batch form, we can first modify the sequential form of Eq. (26.1) to

$${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + h_{i^*,i}(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)), \tag{26.3}$$

where $h_{i^*,i}$ is the neighborhood function. The neighborhood function that would produce Eq. (26.1) is

$$h_{i^*,i} = \begin{cases} \alpha & i \in N_{i^*}(d) \\ 0 & i \notin N_{i^*}(d) \end{cases} \tag{26.4}$$

Using this definition of neighborhood function, we can define a batch version of Eq. (26.1):

$${}_i\mathbf{w}(k) = \frac{\sum_{q=1}^{Q} h_{i^*(q),i}\mathbf{p}(q)}{\sum_{q=1}^{Q} h_{i^*(q),i}}, \tag{26.5}$$

where k is the iteration number and $i^*(q)$ is the winning neuron for input $\mathbf{p}(q)$. Note that for the batch algorithm we have to distinguish between the iteration number and the input number, since all inputs are applied to the network at each iteration. This is in contrast to the sequential algorithm of Eq. (26.1), where there is one iteration for each input. Also, notice that the learning rate does not affect the batch algorithm, since it would appear in both the numerator and the denominator of Eq. (26.5).

For the neighborhood function of Eq. (26.4), this batch algorithm has the effect of assigning each weight to the average of the input vectors for which it is in the neighborhood of the winner. As with the sequential algorithm, the neighborhood size is decreased during training. The neighborhood size is set large at the beginning of training until all weights move into the region of the input space where the data lies. Then the neighborhood size is reduced, to fine-tune the position of the weights.

The batch algorithm requires many fewer iterations than the sequential algorithm, although each iteration requires much more computation. For this case study, we used two iterations of the batch algorithm. During the first iteration the neighborhood size was 4, and during the second iteration the neighborhood size was reduced to 1.

Validation

We will consider two numerical measures of the quality of a trained SOM: resolution and topology preservation (see page 22-23). One measure of SOM resolution is the *quantization error*, which is the average distance between each data vector and its winning neuron. If the average distance is too large, then there are many input vectors that are not adequately represented by any of the prototypes.

A measure of SOM topology preservation is the *topographic error*. This is the proportion of all input vectors for which the closest (winning) neuron and the next closest neuron are not adjacent to each other in the feature map topology. When this number is small, it means that the neurons that are neighbors in the topology are also neighbors in the input space. It is important that this topology be preserved, so that the visualization tools we will discuss later can provide valid insight into the data set.

For our trained SOM, the final quantization error was 0.535, and the final topographic error was 0.037. This means that for less than 4% of all input vectors, the winning neuron and the next closest neuron were not adjacent to each other. It appears that the SOM has achieved the correct topology by the completion of the training.

There are a number of visualization methods that can be used to assess the trained SOM network. One of the key tools is called the unified distance matrix, or *u-matrix*. This is a figure that shows the distance between neighboring neurons in the feature map. The u-matrix has a cell for each neuron

in the feature map and an additional cell between each pair of neurons. The cells between neurons are color-coded with the distance between the corresponding weight vectors. The cells that represent the neurons are coded with the mean of the surrounding values. Figure 26.3 shows the u-matrix for our trained SOM.

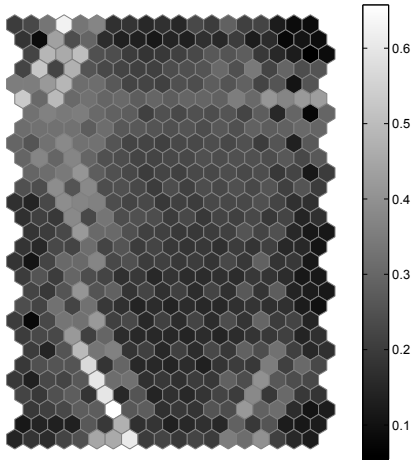

Figure 26.3 U-Matrix for Trained SOM

In Figure 26.3, the light-colored cells represent large distances between neurons. We can see that there is a string of light colored cells on the left side of the feature map. This indicates that the clusters associated with the neurons on the left side of the map are significantly different than those in the middle and right sides of the map. For this data set we actually know the forest cover types for each data point. We can label the feature map cells with the cover type that is associated with the closest input vector to that cluster center. The resulting labeled map is shown in Figure 26.4.

By comparing Figure 26.4 with Figure 26.3, we can see that the forest cover type 2 (see Table 26.2) is associated with the left edge of the feature map. As we move from left to right across the map, we see type 0 and 1 coded into the center section, followed by types 5, 3 and 4, with type 6 located mainly in the upper right section of the map. It is clear that the SOM has learned to cluster the data according to forest cover type.

To get more insight into how the SOM has clustered the data, we can produce a “hit histogram.” For this graph, we count how many times each neuron was the winning neuron for the entire data set. Since our data is labeled with forest cover type, we can also see where each type falls on the feature map. Such a graph is displayed in Figure 26.5. In each cell you can see a hexagram with a certain gray-scale. The size of the hexagrams indicate how many times the corresponding neuron was the winning neuron. The gray level of the hexagram indicates the forest cover type. The darkest hexagons correspond to type 0 forest covers, and the lightest hexagons correspond to type 6 forest covers. We can see that the various regions of the

map have consistent colors. The left side has medium gray levels, corresponding to type 2 cover. The darkest levels are in the center-left region of the map, which corresponds to cover types 0 and 1. The lighter levels, which correspond to types 5 and 6, are in the center-right region, and the median levels of gray, corresponding to types 3 and 4, are on the right edge.

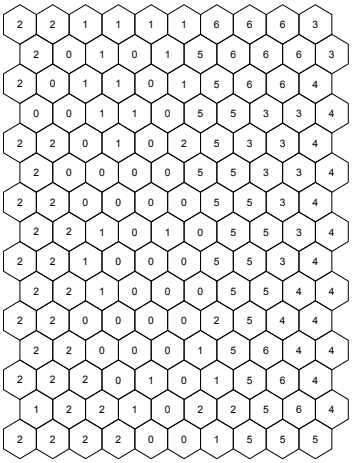

Figure 26.4 Labeled SOM

For many problems, we would not be able to label each input vector. The point here is that the SOM has been able to cluster the data into similar cover types, without knowing what the actual cover types were. This means that the 10 variables making up the input vectors have enough correlation with cover type to allow the SOM to make a useful clustering of the data.

Another tool that is useful in analyzing the trained SOM is the component plane. A component plane is a figure that represents a column of the weight matrix of the SOM. Each column corresponds to one element of the input vector; the jth element of column i represents the connection from input i to neuron j. In a component plane, each element of the weight is represented by a cell in the feature map topology at the location of the neuron to which it is connected. The gray level of the cell represents the magnitude of that element of the weight vector.

The ten component planes (one for each column of the weight matrix - each element of the input vector) for the trained SOM are shown in Figure 26.6. The first thing that we notice is that each of the columns is distinct. There are no two columns that have the same pattern. We can also see that input variables 1, 4, 5, 6 and 10 seem to be important in separating type 2 cover types from the rest of the data. They show patterns in which a boundary appears on the left edge of the feature map, where the type 2 cover types are clustered. By going back to Table 26.1, we can then locate the appropriate variables to see if we can deduce their connection to type 2 cover.

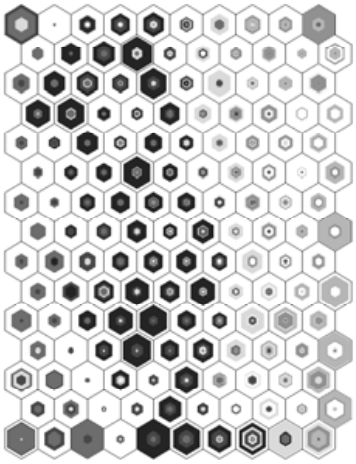

Figure 26.5 Hit Histogram for Trained SOM

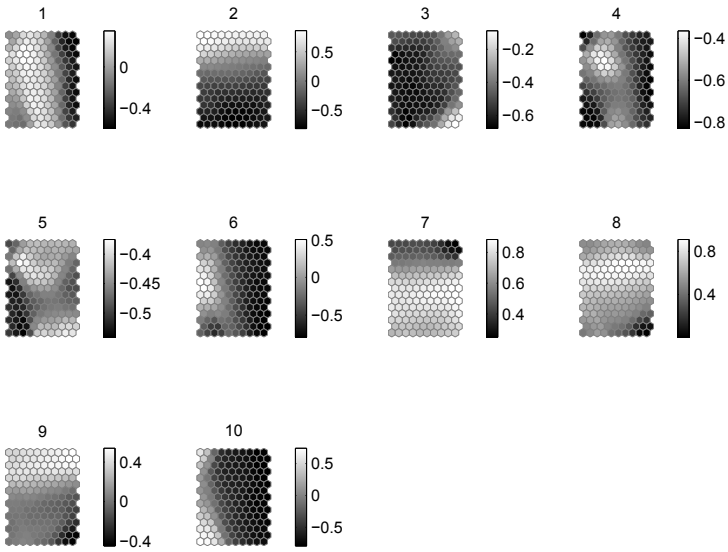

Figure 26.6 Component Planes for the Trained SOM

Data Sets

There are two data files associated with this case study:

- cover_p.txt — contains the input vectors in the data set
- cover_t.txt — contains the targets (labels) in the data set

They can be found with the demonstration software, which is described in Appendix C.

Epilogue

This chapter has demonstrated the use of SOM networks for clustering, in which input vectors in a data set are arranged so that similar vectors are placed in the same cluster. In this case study, the SOM was used to cluster forestry data. The idea was to cluster land into similar forest cover types.

One of the principal advantages of the SOM network, in addition to its ability to efficiently cluster a data set, is its ability to enable visualization of high dimensional data sets.

In the next chapter, we apply neural networks to a prediction problem. We will use a Nonlinear Autoregressive model with eXogenous inputs (NARX) network for that application.

Further Reading

[BlDe99] J. A. Blackard and D. J. Dean, “Comparative Accuracies of Artificial Neural Networks and Discriminant Analysis in Predicting Forest Cover Types from Cartographic Variables,” Computers and Electronics in Agriculture, vol. 24, pp. 131-151, 1999.

This study compared neural networks and discriminant analysis for predicting forest cover types from cartographic variables. The study evaluated four wilderness areas in the Roosevelt National Forest, located in the Front Range of northern Colorado.

[HeBa99] S. Hettich and S. D. Bay, *The UCI KDD Archive* [http://kdd.ics.uci.edu], Irvine, CA: University of California, Department of Information and Computer Science, 1999.

The UCI Knowledge Discovery in Databases Archive. This is an online repository of large data sets which encompasses a wide variety of data types, analysis tasks, and application areas. It is maintained by the University of California, Irvine.

[Koho93] T. Kohonen, “Things you haven't heard about the Self-Organizing Map,” *Proceedings of the International Conference on Neural Networks* (ICNN), San Francisco, pp. 1147-1156, 1993.

This paper describes the batch form of the SOM learning rule, as well as other variations on the SOM.

[Koho95] T. Kohonen, Self-Organizing Map, 2nd ed., Springer-Verlag, Berlin, 1995.

This text describes the theory and practical operation of the Self-Organizing Map in detail. It also has a chapter on the Learning Vector Quantization algorithms.