Chapter 17

Introduction to Clustering

Introduction to clustering. K-means and k-medoids. Expectation-maximization.

17.1 Clustering

The goal of clustering is to group similar examples and separate different examples in different clusters. In other words, to create groups (clusters) of examples in a way that maximizes some measure of similarity between examples within the same cluster and minimizes the similarity between examples in different clusters. The term *clustering* can refer both to the process of computing the clusters and to the resulting set of clusters.

Clustering can help us understand the structure of the data and the relations between different examples and features. For one thing, grouping similar things together in categories that are distinct from other groups is an important part of how we understand the world around us. That is why we have words like “chair”, “stool” or “sofa”, that refer to clusters of objects. This is especially important when we have large datasets, as happens in fields like biology, astronomy or climatology, or when studying social networks online or credit card transactions. Clustering may also show us some important properties of the data. For example, clustering living organisms gives us an insight into their evolutionary relations. Figure 17.1 shows two examples of clustering used to understand the data. On the left panel, Darwin’s “tree of life”, an example of hierarchical clustering that gave an important insight into the mechanisms through which species originate. The right panel is an image from a study using a large set of positional data from Baidu, the most used search engine in China, to characterize empty neighbourhoods in chinese cities [6]. The authors used DBSCAN [10], a density-based clustering algorithm, to group the locations of Baidu users and estimate the home location of each user based on the density clusters of positional information. From this data, the authors then estimated the occupancy of residential neighbourhoods in chinese cities.

Clustering can also be used to *summarise* the data, replacing a large data set with a smaller number of data points that still retain the same overall structure. Figure 17.2 shows the result of using the *k-means* algorithm [16] to compute a simplified dataset, with a smaller number of points, but with the same “shape” as the original set.

There are several choices to make when deciding how to cluster some data. One problem is defining the number of clusters. Figure 17.3 illustrates this, showing the same set of points clustered into two, three or five clusters. Some clustering algorithms determine the number of clusters while others require

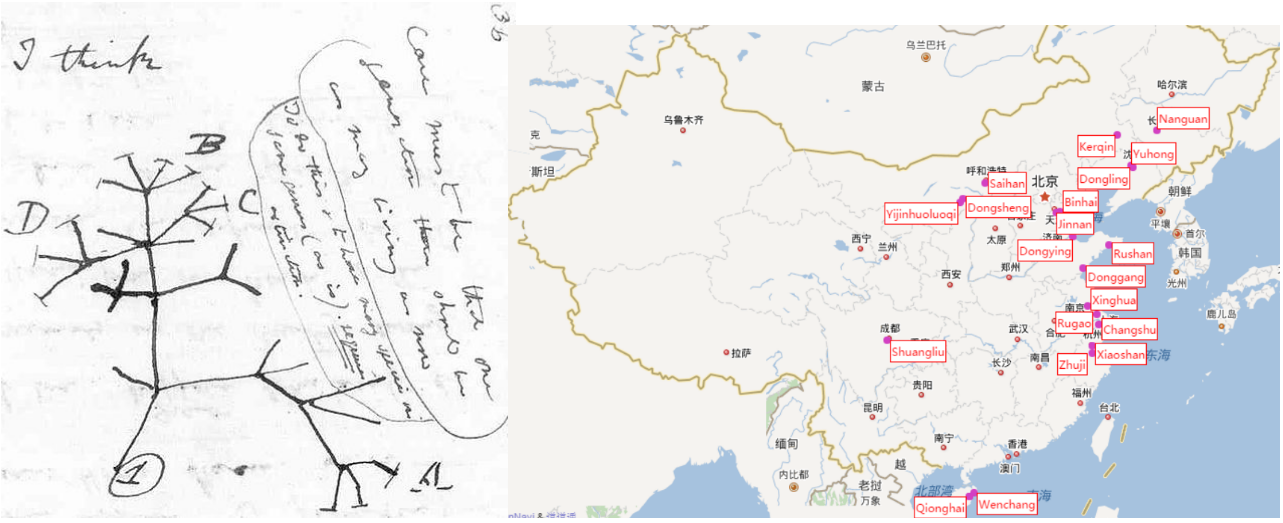

Figure 17.1: The left panel shows Darwin's depiction of the evolutionary relations between different organisms, an example of hierarchical clustering. The right panel is an image from [6], a characterization of "ghost cities" in China by density clustering.

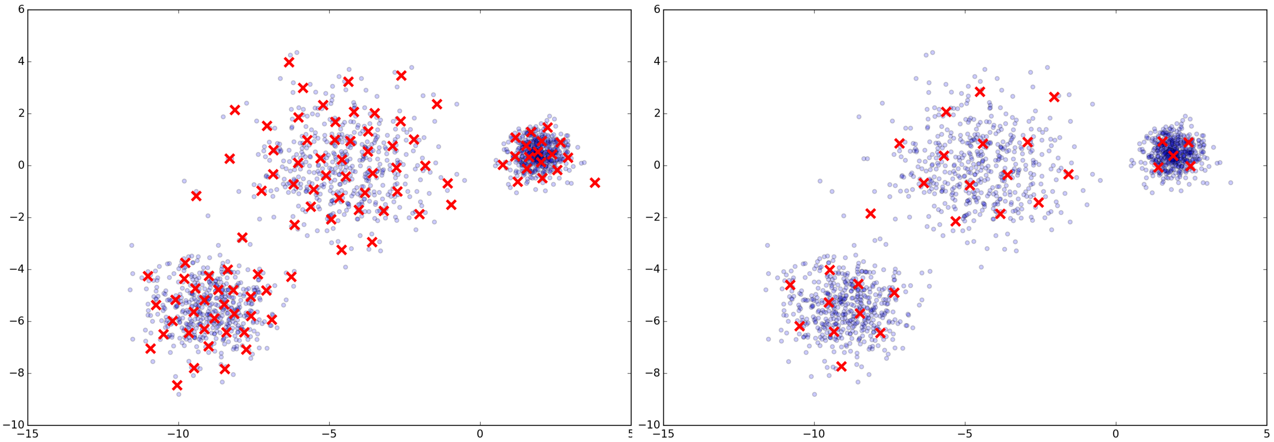

Figure 17.2: Summarizing the larger original dataset (blue points) into 100 or 30 points (red crosses), while still retaining the overall structure of the data.

that the number of clusters be specified in advance. We also need to decide if we want a *partitional* clustering, where the data set is divided into clusters at the same level, or a *hierarchical* clustering, where clusters are also clustered in higher-level clusters. Figure 17.4 shows the difference between partitional and hierarchical clustering.

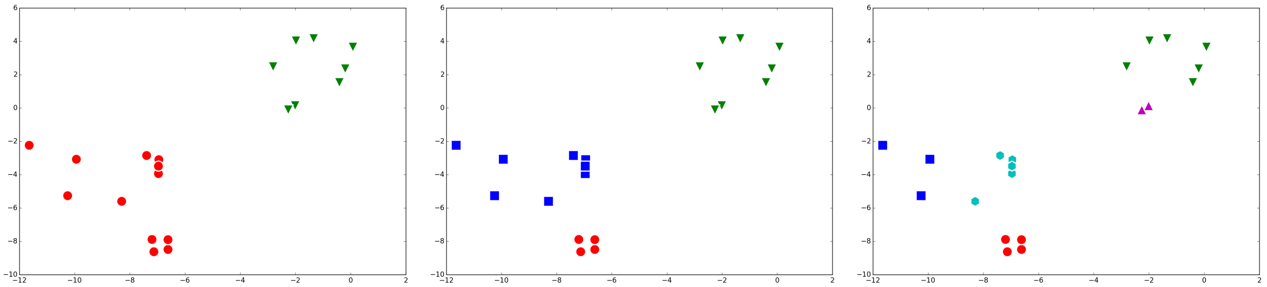

Figure 17.3: Choosing the number of clusters: two, three or five.

Regarding the membership of each example, the clustering can be *exclusive*, if each example belongs only to one cluster; *overlapping* if an example can belong to two or more clusters; *fuzzy* clustering

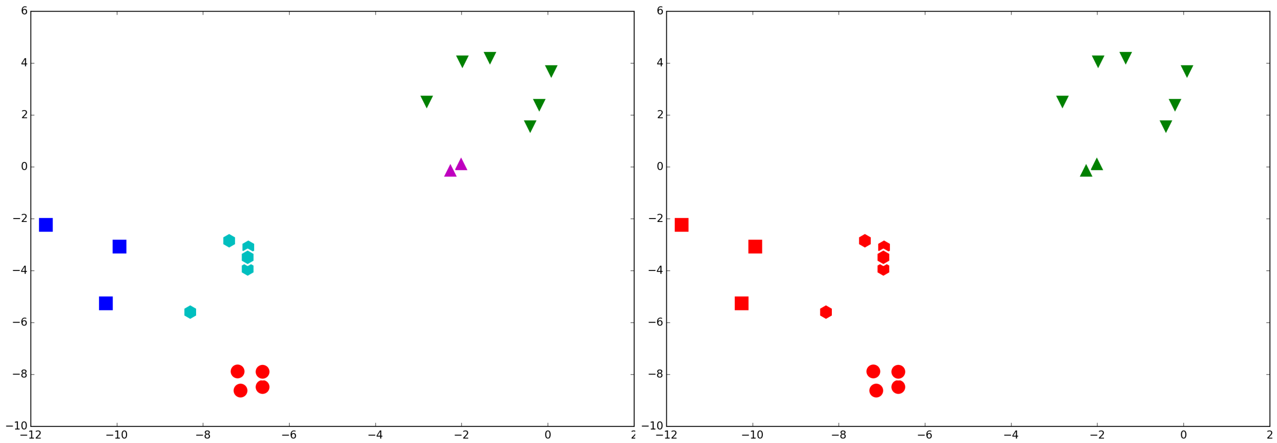

Figure 17.4: The left panel shows an example of partitional clustering. The right panel shows the same data set clustered hierarchically, with the colour representing the top-level clustering and the shapes the bottom level clustering. The lower level clusters are part of the higher level clusters.

if all examples belong to all clusters with some continuous membership value between 0 and 1; or *probabilistic* clustering if the membership value of each example to each cluster represents a probability. The clustering itself can be *partial* if not all examples belong to clusters or *complete* if all examples are assigned to clusters. Depending on the structure of the data and the clustering, the clusters can be *well-separated* if no example is more similar to any example outside its cluster than to any example within its cluster.

Clustering criteria can be based on different aspects of the structure of the data. Figure 17.5 shows some examples. Clustering based on *prototypes* assigns each example to the cluster represented by the closest prototype. With an euclidean distance measure, this results in a Voronoy partition of the feature space. *Contiguity-based* clustering creates clusters according to networks of contiguous examples, and *density-based* clustering assigns examples to clusters defined by high-density regions, allowing for some examples to be left unassigned and discarded as noise. Clustering can also be hierarchical, with groups clustered in larger groups, or defined by probability distributions, such as Gaussian Mixture Models, which we will cover later on.

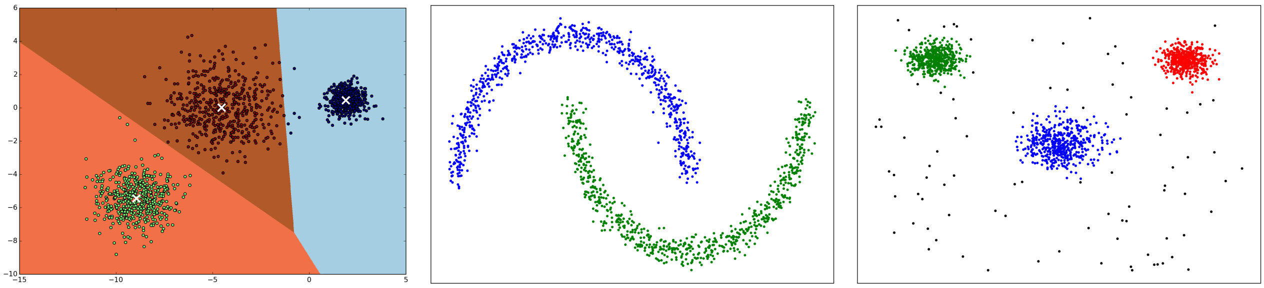

Figure 17.5: Examples of prototype-based clustering, contiguity-based clustering and density-based clustering.

17.2 K-means clustering

Lloyd's algorithm for k-means clustering algorithm[15] is conceptually very simple. It consists in dividing the data set into k clusters, each defined by the mean vector of the members of the cluster, which is the *prototype* of that cluster. Each example belongs to the cluster represented by the closest prototype. Thus, k-means is an exclusive, partitional and prototype-based clustering method. The algorithm for computing the k-means clustering is:

1. Start with a random set of k prototypes.

2. Assign each example to the closest prototype.

3. Recompute each prototype as the mean point of all the examples assigned to that cluster.

4. Repeat steps 2 and 3 until convergence or some stopping criterion.

There are several possible initialization methods for k-means. For example, the *Forgy* method consists of assigning to each of the k prototypes the feature vector of a randomly selected examples as the starting point for the algorithm. The *Random Partition* method starts by randomly assigning each example to one of k clusters, and then computing the starting point of the prototypes as the mean point of each cluster. Figure 17.6 illustrates these two initialization methods, showing also how the *Random Partition* method tends to group the initial positions of the prototypes in the centre of the data space.

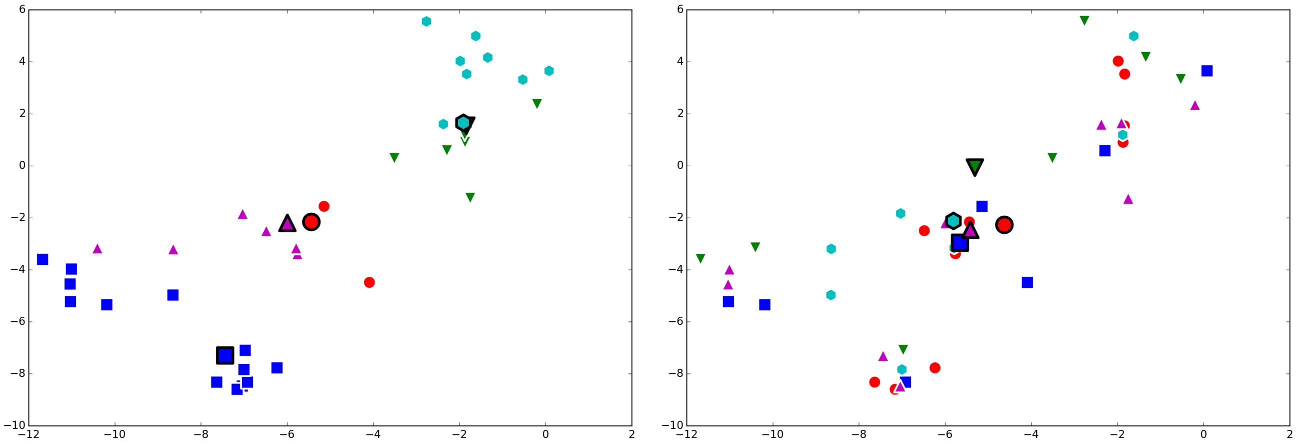

Figure 17.6: Initializing the k-means prototypes by the Forgy method (left panel) or the Random Partition method (right panel). The prototypes are represented with larger symbols and the smaller symbols represent the data points.

Figure 17.7 shows the first iterations of computing the clusters by the k-means algorithm, starting with the random (Forgy) assignment of the initial prototype positions and assignment of the data points to the clusters (left panel), recomputing the position of the cluster prototypes as the mean point of each cluster (middle panel) and then recomputing the cluster assignment by assigning each example to the cluster of the closest prototype (right panel).

To illustrate in more detail, we can see how to implement the k-means algorithm in Python. We start with a function that determines the cluster of each data point in matrix `data` given a matrix of centroid coordinates for the positions of the prototypes in `centroids`:

```python
def closest_centroids(data,centroids):
    ys = np.zeros(data.shape[0])
    for ix in range(data.shape[0]):
        dists = np.sum((centroids-data[ix,:])**2,axis=1)
        ys[ix] = np.argmin(dists)
    return ys
```

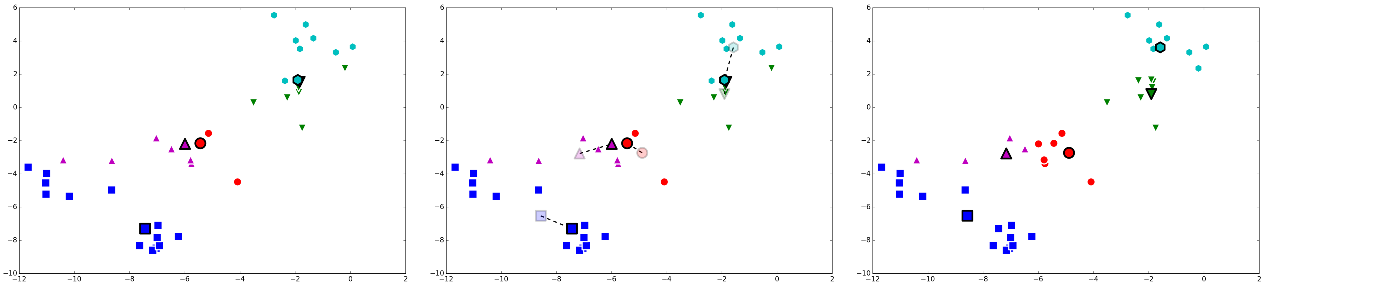

Figure 17.7: First iterations of the k-means algorithm. See text for more details.

Next, a function to recompute the centroids based on the assigned clusters. This function iterates through the lines of the `centroids` matrix computing, for each line, the means of the data points assigned to that cluster.

```
def adjust_centroids(data,centroids):
    ys = closest_centroids(data,centroids)
    for ix in range(centroids.shape[0]):
        centroids[ix,:] = np.mean(data[ys==ix,:],axis=0)
```

Now we need a function for initializing the prototype positions (the centroids of the clusters). Using the Forgy method, we can do this assigning the coordinates of randomly selected examples. Note that the matrix returned is a copy of the selected lines of the data matrix, otherwise we could be returning pointers to the data points and altering the centroids would alter the data.

```
def forgy(data,k):
    ixs = np.arange(data.shape[0])
    np.random.shuffle(ixs)
    return data[:k,:].copy()
```

Alternatively, we can initialize the centroids using the random partition algorithm.

```
def rand_part(data,k):
    ys = np.random.randint(0,k,data.shape[0])
    centroids = np.zeros((k,data.shape[1]))
    for ix in range(k):
        centroids[ix,:] = np.mean(data[ys==ix,:],axis=0)
    return centroids,ys
```

To find the best prototypes for clusters we need a distance measure. Very often, the measure is the Euclidean distance. However, we can generalize this, as we saw before, with the Minkowsky distance, which depends on a parameter p:

$$D_{x,x'} = \sqrt[p]{\sum_d |x_d - x'_d|^p}$$

For $p = 2$ this is the Euclidean distance, and for $p = 1$ the Manhattan distance.

17.3 K-medoids

The *k-medoids* algorithm is a variant of the k-means algorithm with the difference that the prototypes always coincide with points in the data set. This makes it unnecessary to have a true distance measure, since all we need is to measure similarities between points in the data set, and makes the algorithm more robust to noise and outliers. The k-medoids clustering can be computed using the *Partitioning Around Medoids* (PAM) algorithm, as follows:

1. Initialize the k prototypes (*e.g.* with the Forgy method).
2. Assign the examples to clusters.
3. For each medoid and each data point, test if swapping the medoid for that data point reduces the sum of pairwise dissimilarities between data points and respective medoids. If so, then update the medoid and reassign examples to clusters.
4. repeat step 3 until no improvement possible.

17.4 Expectation-Maximization

The *Expectation-Maximization* (EM) method is an important part of many unsupervised learning algorithms, and we shall revisit it in more detail in future chapters. But, for now, we can introduce it in a simplified overview and see how it relates to the k-means algorithm. Let us assume we have a set X of observed data — for example, the known data points — and a set Z of variables we do not observe, which are called *latent variables*. For example, the assignments of each data point to each cluster, which we initially do not know. We also have a set of parameters θ that we wish to adjust in order to maximize the likelihood of the parameters, which is the probability of all the data, including both the known and unknown variables, given the set of parameters θ. For example, the centroids representing the clusters.

$$L(\theta; X, Z) = p(X, Z|\theta)$$

We cannot compute this likelihood directly because we do not know the values Z. But we can estimate the posterior, or conditional, distribution of Z given the known X and some previous assumption about θ:

$$p(Z|X, \theta^{old})$$

This allows us to compute an expected value for Z and thus estimate the necessary parameters for the likelihood function $\mathcal{Q}(\theta, \theta^{old})$ for θ given some previous estimated θ^{old}. From the expected values of Z given X and θ^{old} and the likelihood of θ for the known X and the expected Z, we can write:

$$\mathcal{Q}(\theta, \theta^{old}) = \mathrm{E}_{Z|X,\theta^{old}} \ln p(X, Z|\theta)$$

We can now find the new values of $theta$ that maximize the likelihood function:

$$\theta^{new} = \arg\max_{\theta} \mathrm{E}_{Z|X,\theta^{old}} \ln p(X, Z|\theta)$$

Broadly speaking, this is what the k-means algorithm does[1]. The latent variables Z correspond to the assignment of examples to clusters and the known variables X correspond to the coordinates (features) of the examples in the data set. Given a set of prototype centroids, the θ^{old}, we can estimate the best values for Z by assigning each data point to the closest centroid. With this, we can obtain a

[1] K-means is not exactly Expectation-Maximization because k-means assigns each point to each cluster instead of estimating a probability of the point belonging to the cluster. However, we can see k-means as a limit case of EM.

maximum likelihood estimate of the centroid positions, the θ^{new}, by computing the mean point of each cluster.

Another way of understanding the EM algorithm is as an alternating sequence of optimizations with respect to different variables. This can be seen if we define the *distortion measure*:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - \mu_k\|^2$$

to evaluate the clusters, where r_{nk} is 1 if x_n belongs to μ_k, else 0. Thus, the r_{nk} variables are the latent variables assigning each point n to each cluster k. This distortion measure J, which is the sum of the squared distances between points and their respective cluster centroids, is what we want to minimize by optimizing the μ_k parameters, which are the coordinates of the centroids. However, we cannot do this without the r_{nk} values. Thus we first estimate the best r_{nk} values by minimizing J with respect to the r_{nk}, which consists simply of assigning each data point to the closest centroid, which results in the smallest distance added. Now we optimize the function J with respect to the μ_k centroids. Since J is a quadratic function on μ_k, the minimum can be found where the derivative is zero, which corresponds to the value of μ_k that is the mean point of the data points in cluster k.

As a limiting case of the more general Gaussian mixture models we will see in Chapter 20, the complete-data likelihood (including both the observed X and latent Z) for k-means tends towards:

$$\mathrm{E}_Z\left(\ln p(X, Z|\boldsymbol{\mu})\right) \rightarrow -\frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - \mu_k\|^2 + C$$

Thus, in these conditions, EM corresponds to the alternating minimization of J we saw above. However, we will postpone a more detailed explanation of the EM method to Chapter 20, after looking at the Gaussian mixture models.

17.5 Application example for k-means

The k-means algorithm is often used for vector quantization, which is a procedure for reducing vectors in a range of values to a smaller set of representative prototypes. In this example, we'll use k-means to quantize the colour space of an image[2]. Figure 17.8 shows the starting image and the results of colour quantization with 64 and 8 centroids.

We start by loading the image and converting it into a set of two-dimensional vectors using the red and green colour components, since this image has no blue components. This is easy to do by using the `imread` function from the Scikit-image library:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from skimage.io import imsave,imread

rosa = imread("Rosa.png")
w,h,d = rosa.shape
cols = np.reshape(rosa/255.0, (w * h, d))
image_array = cols[:,:2]
```

[2] This example is based on the Scikit-learn demo on colour quantization, http://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html, using an example of vector quantization in Wikipedia, https://en.wikipedia.org/wiki/K-means_clustering

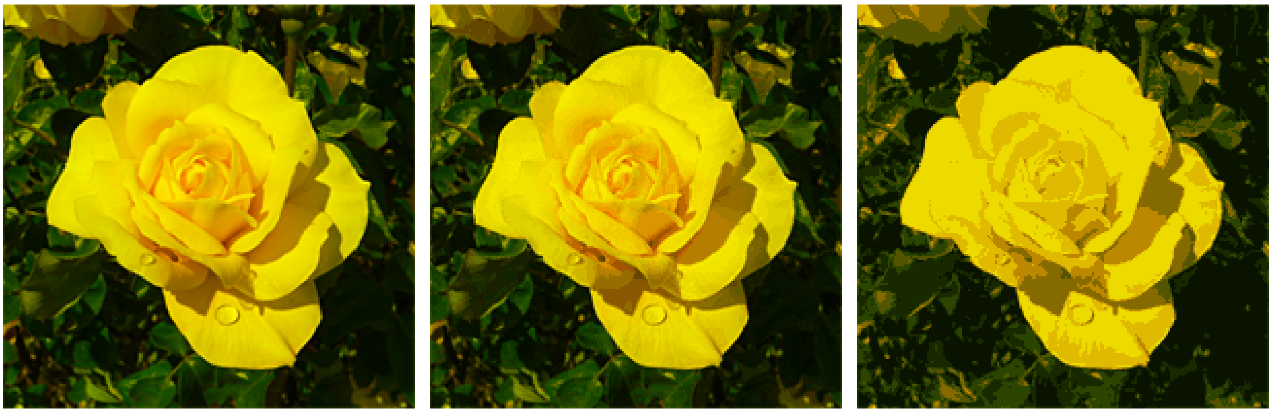

Figure 17.8: The original image (left panel) compared with two different colour quantizations, reducing the colour space to 6 bits (64 colours) and 3 bits (8 colours).

We can now plot all the pixels in the red-green colour space

```
plt.figure(figsize=(12,8))
plt.xlabel('Red')
plt.ylabel('Green')
plt.scatter(image_array[:, 0], image_array[:, 1], color=cols.tolist(), s=10)
plt.axis([0,1,0,1])
plt.savefig('L17-rosa-plot.png', dpi=200,bbox_inches='tight')
```

and use the k-means algorithm to find the best set of k prototypes to represent the colours. For example, for 64 colours, we can use the `KMeans` class from the `cluster` module of the Scikit-learn library. Computing the centroids and plotting over the plot of the set of points:

```
n_colors = 64
kmeans = KMeans(n_clusters=n_colors).fit(image_array)
labels = kmeans.predict(image_array)
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', color='k',s=200, linewidths=5)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', color='w',s=150, linewidths=2)
plt.savefig('L17-rosa-plot-cs-'+str(n_colors)+'.png', dpi=200,bbox_inches='tight')
```

The result is shown in Figure 17.9

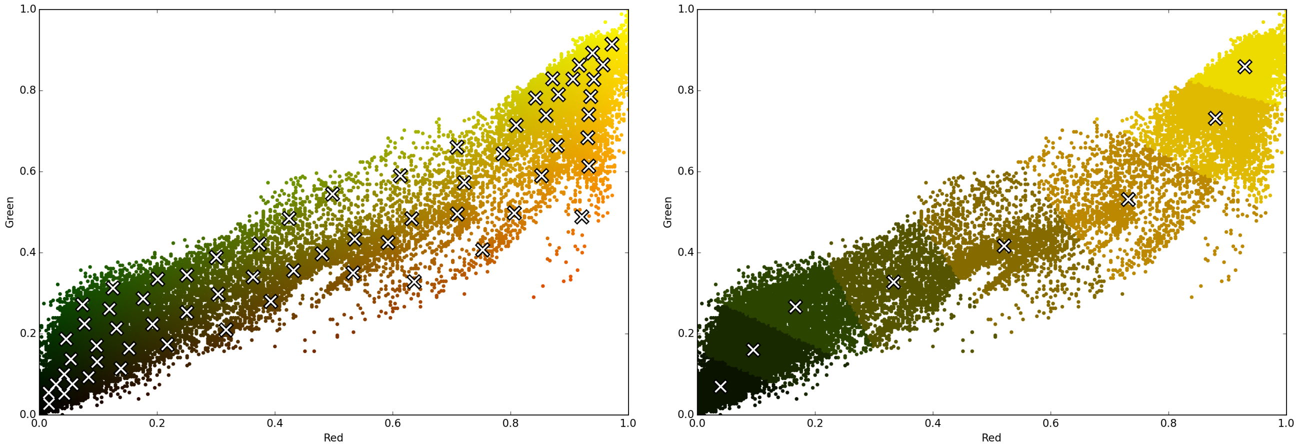

Figure 17.9: Quantizing the colour-space into 64 or 8 colours.

17.6 Further Reading

1. Alpaydin [2], Section 7.3
2. Marsland [17], Chapter 9
3. Bishop [4], Section 9.1

Bibliography

[1] Uri Alon, Naama Barkai, Daniel A Notterman, Kurt Gish, Suzanne Ybarra, Daniel Mack, and Arnold J Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences*, 96(12):6745–6750, 1999.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[3] David F Andrews. Plots of high-dimensional data. *Biometrics*, pages 125–136, 1972.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, New York, 1st ed. edition, oct 2006.

[5] Deng Cai, Xiaofei He, Zhiwei Li, Wei-Ying Ma, and Ji-Rong Wen. Hierarchical clustering of www image search results using visual. Association for Computing Machinery, Inc., October 2004.

[6] Guanghua Chi, Yu Liu, and Haishandbscan Wu. Ghost cities analysis based on positioning data in china. *arXiv preprint arXiv:1510.08505*, 2015.

[7] Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, pages 396–404. Morgan Kaufmann, 1990.

[8] Pedro Domingos. A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning. Stanford CA Morgan Kaufmann*, pages 231–238, 2000.

[9] Hakan Erdogan, Ruhi Sarikaya, Stanley F Chen, Yuqing Gao, and Michael Picheny. Using semantic analysis to improve speech recognition performance. *Computer Speech & Language*, 19(3):321–343, 2005.

[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[11] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[12] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[13] Patrick Hoffman, Georges Grinstein, Kenneth Marx, Ivo Grosse, and Eugene Stanley. Dna visual and analytic data mining. In *Visualization'97., Proceedings*, pages 437–441. IEEE, 1997.

[14] Chang-Hwan Lee, Fernando Gutierrez, and Dejing Dou. Calculating feature weights in naive bayes with kullback-leibler measure. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 1146–1151. IEEE, 2011.

[15] Stuart Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[16] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[17] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st edition, 2009.

[18] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[19] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[20] Roberto Valenti, Nicu Sebe, Theo Gevers, and Ira Cohen. Machine learning techniques for face analysis. In Matthieu Cord and Pádraig Cunningham, editors, *Machine Learning Techniques for Multimedia*, Cognitive Technologies, pages 159–187. Springer Berlin Heidelberg, 2008.

[21] Giorgio Valentini and Thomas G Dietterich. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. *The Journal of Machine Learning Research*, 5:725–775, 2004.

[22] Jake VanderPlas. Frequentism and bayesianism: a python-driven primer. *arXiv preprint arXiv:1411.5018*, 2014.