Chapter 15
Machine Learning (Unsupervised)

Shailesh Kumar

We live in the age of data. This data is emanating from a variety of natural phenomena, captured by different types of sensors, generated by different business processes, or resulting from individual or collective behavior of people or systems. This observed sample data (e.g., the falling of the apple) contains a view of reality (e.g., the laws of gravity) that generates it. *In a way, reality does not know any other way to reveal itself but through the data we can perceive about it.*

The goal of unsupervised learning is essentially to "reverse engineer" as much of this reality from the data we can sample from it. In this chapter, we will explore unsupervised learning—an important paradigm in machine learning—that helps uncover the proverbial needle in the haystack, discover the grammar of the process that generated the data, and exaggerate the "signal" while ignoring the "noise" in it. In particular, we will explore methods of projection, clustering, density estimation, itemset mining, and network analysis—some of the core unsupervised learning frameworks that help us perceive the data in different ways and hear the stories it is telling about the reality it is sampled from. The examples, corresponding code, and exercises for the chapter are given in the online appendices.

1 Introduction

> The most elementary and valuable statement in Science, the beginning of Wisdom is—'I do not know'
>
> —Star Trek.

S. Kumar (✉)
Reliance Jio, Navi Mumbai, Maharashtra, India
e-mail: skumar.0127@gmail.com

B. Pochiraju, S. Seshadri (eds.), *Essentials of Business Analytics*, International Series in Operations Research & Management Science 264,
https://doi.org/10.1007/978-3-319-68837-4_15

Any scientific process begins with observation (data), formulating a hypothesis about the observation, testing the hypothesis through experimentation, and validating and evolving the hypothesis until it "fits the observation." Most scientific discoveries start with the ability to "observe" the data objectively followed by a pursuit to discover the "why" behind "what" we observe. The broad field of data *science* follows a similar scientific process by first trying to understand the nuances in the data, formulating a variety of hypotheses about, for example, what cause (e.g., a bad customer experience) might lead to what effect (e.g., customer churn), or which variables (e.g., education and age) might be correlated with others (e.g., income). It then provides algorithms to validate these hypotheses by building and interpreting models both descriptive and predictive, and finally, it enables us to take decisions to make the businesses, processes, applications, infrastructures, cities, traffic, economies, etc. more efficient.

The broad field of machine learning has evolved over the last several decades to generate a very large collection of modeling paradigms—including the supervised learning paradigm covered in the next chapter and the unsupervised learning paradigm, the subject of this chapter. Apart from these, there are a number of other paradigms such as the semi-supervised learning, active learning, and reinforcement learning. We will first understand the core differences between the supervised and unsupervised learning paradigms and then go into the various frameworks available within the unsupervised learning paradigm.

Supervised vs. Unsupervised Learning

Any intelligent system—including our own brain—does a variety of things with the data it observes:

- *Summarizes and organizes the data* (e.g., a business (retail or finance) might want to segment all its customers into a coherent group of similar customers based on their demographics and behavior).
- *Infers the grammar of the data* (e.g., typically what products in a retail market basket "go together" or what word will follow a sequence of words, say, "as soon as").
- *Interprets the data semantically* (e.g., a speech-enabled interface tries to first interpret the speech command of a user to text and from text to user intent).
- *Finds significant patterns in data* (e.g., which words typically occur before or after others, which sets of products are purchased together, what genes get activated together, or which neurons fire together).
- *Predicts what is about to happen* (e.g., in a bank or telecom businesses can predict that a certain customer is about to churn or an IoT system can predict that a certain part is about to fail).
- *Optimizes the best action given the prediction* (e.g., give a certain offer to the customer to prevent churn or preorder the part before it fails, to avert the unfavorable predicted future).

Some of these tasks require us to just observe the data in various ways and find structures and patterns in it. Here there is no "mapping" from some input to some output. Here we are just given a lot of data and asked to find something "interesting" in it, to reveal from data insights that we might not be aware of. For example, one might find product bundles that "go together" in a retail point of sale data or the fact that age, income, and education are correlated in a census data. The art and science of finding such structures in data without any particular end use-case in mind falls under unsupervised learning. Here we are just "reading the book" of the data and not "trying to answer a specific question" about the data. It is believed that in early childhood, most of what our brain does is unsupervised learning. For example:

- *Repeated Patterns*: when a baby hears the same set of sounds over and over again (e.g., "no"), it learns that this sound seems important and creates and stores a pattern in the brain to recognize that sound whenever it comes. It may not "understand" what the sound means but registers it as important because of repetition. The interpretation of this pattern might be learnt later as it grows.
- *Sequential patterns*: a child might register the fact that a certain event (e.g., ringing of a doorbell) is typically followed by another event (e.g., someone opens the door). This sequential pattern learning is key to how we pick up music, art, and language (mother tongue) even without understanding its grammar but by simply observing these sequential patterns over and over.
- *Co-occurrence patterns*: a child might recognize that two things always seem to co-occur together (e.g., whenever she sees eyes, she also sees nose, ear, and mouth). A repeated co-occurrence of same objects in the same juxtaposition leads to the recognition of a higher order object (e.g., the face).

In all these patterns, the grammar of the data is being learnt for no specific purpose except that it is there.

Supervised learning, on the other hand, is a mapping from a set of observed features to either a class label (classification paradigm) or a real value (regression paradigm) or a list of items (recommendation or retrieval paradigm), etc. Here we deliberately learn a mapping between one set of inputs (e.g., a visual pattern on a paper) and an output (e.g., this is letter "A"). This mapping is used both in interpreting and assigning names (or classes) to the patterns we have learnt (e.g., the sound for "dad" and "mom") as a baby in early childhood, which now are interpreted to mean certain people, or to the visual patterns one has picked up in childhood which are now given names (e.g., "this is a ball," "chair," "cat"), etc. This mapping is also used for learning cause (a disease) and effect (symptoms) relationships or observation (e.g., customer is not using my services as much as before) and prediction (e.g., customer is about to churn) relationships. A whole suite of supervised learning paradigms is discussed in the next chapter. In this chapter we will focus only on unsupervised learning paradigms.

Unsupervised Learning Paradigms

I don't know what I don't know—the Second Order of Ignorance

One of the most important frameworks in machine learning is unsupervised learning that lets us "observe" the data systematically, holistically, objectively, and often creatively to discover the nuances of the underlying process that generated the data, the grammar in the data, and insights that we didn't know existed in the data in the first place. In this chapter, we will cover five unsupervised learning paradigms:

- **Projections**—which is about taking a high dimensional data and finding lower dimensional projections that will help us both visualize the data and see if the data really belongs to a lower dimensional "manifolds" or is it inherently high dimensional. In particular, we will study various broad types of projection algorithms such as (a) principal components analysis (PCA) that try to minimize loss of variance, (b) self-organizing maps that try to smear the data on a predefined grid, and (c) multidimensional scaling (MDS) that try to preserve pairwise distances between data points after projection.
- **Clustering**—which is about taking the entire set of entities (customers, movies, stars, gene sequences, LinkedIn profiles, etc.) and finding "groups of similar entities" or hierarchies of entities. In a way our brain is a compression engine and it tries to map what we are observing into groups or quantization. Clustering ignores what might be noise or unimportant (e.g., accent when trying to recognize the word in a speech might not be important). It is also useful in organizing a very large amount of data into meaningful clusters that can then be interpreted and acted upon (e.g., segment-based marketing). In particular, we will study (a) partitional clustering, (b) hierarchical clustering, and (c) spectral clustering.
- **Density Estimation**—which is about quantifying whether a certain observation is even possible or not given the entire data. Density estimation is used in fraud detection scenarios where certain patterns in the data are considered normal (high probability) while certain other patterns might be considered outlier or abnormal (low probability). In particular, we will study both parametric and nonparametric approaches to learning how to compute the probability density of a record.
- **Pattern Recognition**—which is about finding the most frequent or significant repetitive patterns in the data (e.g., "people who buy milk also buy bread," or what words typically follow a given sequence of words). These patterns reveal the grammar of the data simply be relative frequency of patterns. High frequency patterns are deemed important or signal, while low frequency patterns are deemed noise. In particular, we will study (a) market-basket analysis, where patterns from sets are discovered, and (b) n-grams, where patterns from sequences are discovered.
- **Network Analysis**—which is about finding structures in what we call a network or graph data, for example, communities in social networks (e.g., terrorist cells, fraud syndicates), importance of certain nodes over others given the link structure of the graph (e.g., PageRank), and finding structures of interests (e.g., gene pathways, money laundering schemes, bridge structures). Graph theory and

network analysis algorithms when applied to real-word networks can generate tremendous insights that are otherwise hard to perceive.

Modeling and Optimization

Before we dive into the five paradigms, we will make another horizontal observation that will help us become a better "formulator" of a business problem into a machine learning problem—the key quality of a data scientist. Most machine learning algorithms—whether supervised or unsupervised—boil down to some form of an optimization problem. In this section, we will develop this *way of thinking* that what we really do in machine learning is a four stage optimization process:

- **Intuition**: We develop an intuition about how to approach the problem as an optimization problem.
- **Formulation**: We write the precise mathematical objective function in terms of data using intuition.
- **Modification**: We modify the objective function into something simpler or "more solvable."
- **Optimization**: We solve the modified objective function using traditional optimization approaches.

As we go through the various algorithms, we will see this common theme. Let us take two examples to highlight this process—as one of the goals of becoming a data scientist is to develop this systematic process of thinking about a business problem.

The Mean of a Set of Numbers

First we start with a very simple problem and formulate this as an objective function and apply the remaining steps. Consider a set of N numbers $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}$. Now let us say we want to find the mean of these numbers. We know the answer already but that answer is not a formula we memorize, it is actually a result of an optimization problem. Let us first make an assumption (like in Algebra) that let m be the mean we are looking for. This is called the *parameter* we are trying to find.

Intuition: What makes m the *mean* of the set $\mathbf{X}$? The intuition says that mean is a point that is "closest to all the points." We now need to formulate this intuition into a mathematical objective function.

Formulation: Typically, in an objective function there are three parts: The unknown parameter we are optimizing (in this case m), the data (in this case $\mathbf{X}$), and the constraints (in this case there are no constraints). We can write the objective function as the sum of absolute distance between the point m and each data point that we must minimize to find m as a function of $\mathbf{X}$.

$$J\left(m\,|\mathbf{X}\right) = \sum_{n=1}^{N} |m - x_n|$$

Modification: Now the above objective function makes sense intuitive, but it is not easy to optimize it from a mathematical perspective. Hence, we come up with

a more "solvable" or "cleaner" version of the same function. In this case, we want to make it "differentiable" and "convex." The following objective function is also known as sum of squared error (SSE).

$$J\left(m\,|\mathbf{X}\right) = \sum_{n=1}^{N} (m - x_n)^2$$

Now we have derived an objective function that matches our intuition as well as is mathematically easy to optimize using traditional approaches—in this case, simple calculus.

Optimization: The most basic optimization method is to set the derivative of the objective w.r.t. the parameter to zero:

$$\frac{\partial J\left(m\,|\mathbf{X}\right)}{\partial m} = \sum_{n=1}^{N} \frac{\partial (m - x_n)^2}{\partial m} = 2\sum_{n=1}^{N} (m - x_n) = 0 \Rightarrow \widehat{m} = \frac{1}{N}\sum_{n}^{N} x_n$$

So we see that there is no *formula* for mean of a set of numbers. That formula is a *result* of an optimization problem. Let us see one more example of this process.

Probability of Heads

Let us say we have a two-sided (possibly biased) coin that we tossed a number of times and we know how many times we got heads (say H) and how many times we got tails (say T). We want to find the probability p of heads in the next coin toss. Again, we know the answer, but let us again go through the optimization process to find the answer. Here the data is *(H, T)* and parameter is p.

Intuition: The intuition says that we want to find that parameter value p that explains the data the most. In other words, if we knew p, what would be the likelihood of seeing the data (H, T)?

Formulation: Now we formulate this as an optimization problem by assuming that all the coin tosses are independent (i.e., outcome of previous coin tosses does not affect the outcome of the next coin toss) and the process (the probability p) is constant throughout the exercise. Now if p is the probability of seeing a head, then the joint probability of seeing H heads is p^H. Also since heads and tails are the only two options, probability of seeing a tail is $(1 - p)$ and seeing T tails is $(1 - p)^T$. The final *Likelihood* of seeing the data *(H, T)* is given by the product of the two:

$$J\left(p|H, T\right) = p^H (1 - p)^T$$

Modification: The above objective function captures the intuition well but is not mathematically easy to solve. We modify this objective function by taking the log of it. This is typically called the *Log Likelihood* and is used commonly in both supervised and unsupervised learning.

$$J\left(p|H, T\right) = H \ln p + T \ln (1 - p)$$

Optimization: Again we will use calculus—setting the derivative w.r.t. the parameter to zero.

$$\frac{\partial J\left(p|H,T\right)}{\partial p} = \frac{H}{p} - \frac{T}{1-p} = 0 \Rightarrow \widehat{p} = \frac{H}{H+T}$$

So we see how the intuitive answer that we remember for this problem is actually not a *formula* but a *solution* to an optimization problem.

Machine learning is full of these processes. In the above examples, we saw two types of objective functions (sum of squared error and log likelihood) which cover a wide variety of machine learning algorithms. Also, here we were lucky. The solutions to the objective functions we formulated were simple *closed-form* solutions where we could just write the parameters in terms of the data (or some statistics on the data). But, in general, the objective functions might become more complex and the solution might become more iterative or nonlinear. In any case, the process remains the same and we will follow pretty much the same process in developing machine learning (unsupervised in this chapter and supervised in next) models.

Visualizations

Machine learning is really the art of marrying our understanding of the data, our appreciation for domain knowledge, and our command on the algorithms that are available to us. Here we will explore the power of simple visualizations that reveal a lot about the nuances in the data. This is the essential first step in any data science journey. Through this process, we will also understand how to read the data (and not just ask questions), learn the art of listening to data (and not just testing our own hypotheses), and then apply the right transformations and prepare the data for subsequent stages of insight generation and modeling.

Histograms

One of the simplest and most powerful visualizations are the histograms of each dimension of the data. Figure 15.1 shows histograms of the Higgs-Boson data.[1] This reveals some interesting facts. None of the dimensions is actually

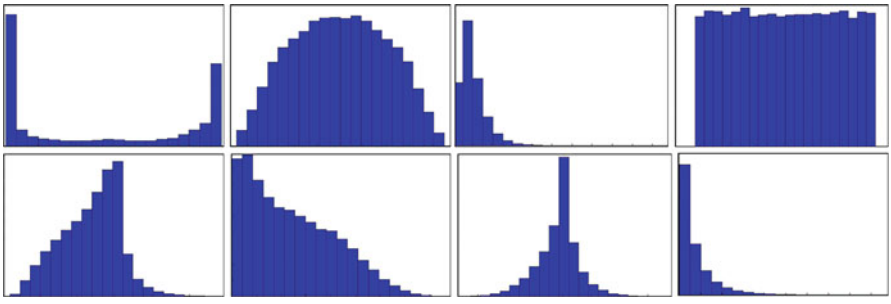

Fig. 15.1 Histograms of eight different dimensions of Higgs boson dataset (none is normally distributed)

[1] https://www.kaggle.com/c/Higgs-boson (Retrieved September 5, 2018).

normally distributed. The distributions we see here are exponential, log-normal, doubly exponential, a combination of linear and exponential, parabolic, and reverse normal distributions. Any transformation such as min-max or z-scoring on these dimensions that assumes normality will not yield the desired results. This insight is very important—when we choose certain algorithms (e.g., K-means later), we make certain implicit assumptions about the nature of the distance functions and feature distributions. Histograms help us validate or invalidate those assumptions and can therefore force us to transform some of the features (e.g., by taking their logs) to make them closer to the assumptions that are used by those techniques.

Log Transforms

One of the most common distributions in real-world data is not really a normal distribution but a log-normal, exponential, or Zip's law distribution. For example, the income distribution of any reasonably large population will be exponentially distributed. Frequency of words and their rank order have a Zip's law distribution and so on. One of the common and practical things is to try the log of a feature with such distributions instead of using the feature "as is." For example, PCA or K-means clustering that depends on normal distribution assumption and uses Euclidean distances performs better when we undo the effect of this exponential distribution by taking the log of the features. Figure 15.2 shows an example of the Higgs boson features with log-normal or exponential distribution (left) and their distribution after taking the log (right). In both cases, new insights emerge from this process. Hence, it is important to explore the histogram of each feature and determine whether log will help or not.

Scatter Plots

Another simple yet very effective tool for understanding data is to visualize the scatter plot between all pairs (or triples) of dimensions and even color code

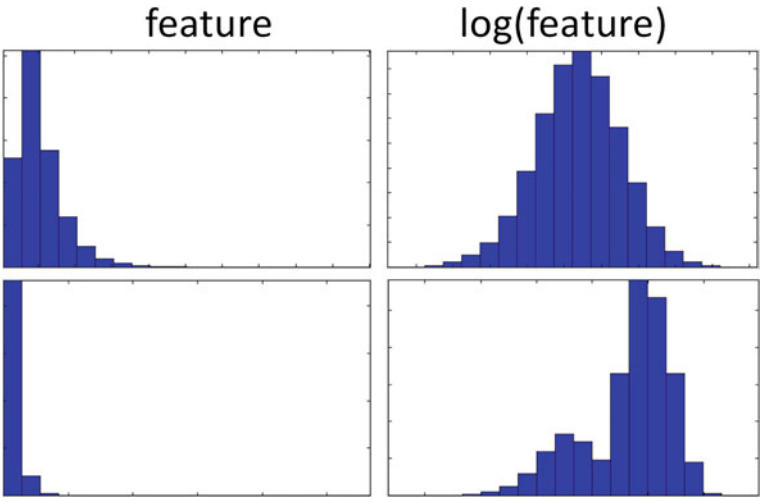

Fig. 15.2 Histogram of a feature (left) and its log (right). Taking log is a useful transformation

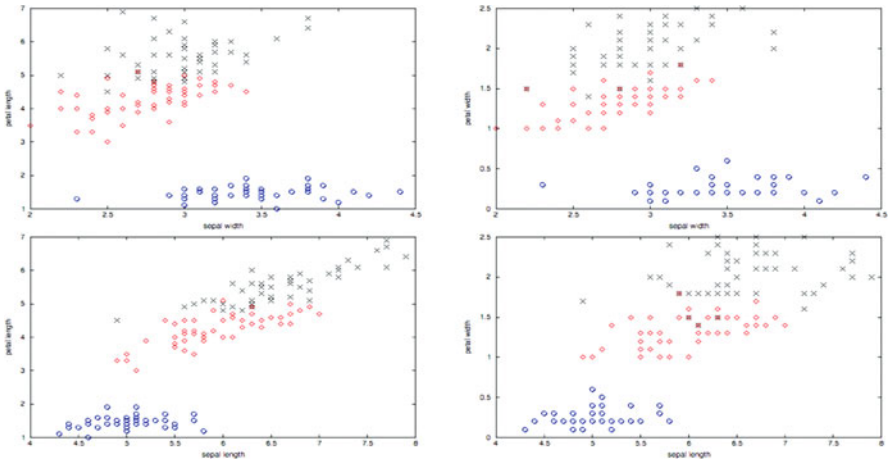

Fig. 15.3 A few scatter plots of IRIS data show that two classes are closer to each other than the third

each point by another property. Scatter plots reveal the structure of the data in the projected spaces and develop our intuition about what techniques might be best suited for this data, what kind of features we might want to extract, which features are more correlated to each other, which features are able to discriminate the classes better, etc.

Figure 15.3 shows the scatter plot of the IRIS dataset[2] between a few pairs of dimensions. The color/shape coding of a point is the class (type of Iris flower) the point represents. This immediately shows that two of the three classes of flowers are more similar to each other than the third class.

While histograms give us a one-dimensional view of the data and scatter plots give us two- or three-dimensional view of the data, they are limited in what they can do. We need more sophisticated methods to both visualize the data in lower dimensions and extract features for next stages. This is where we resort to a variety of projection methods discussed next.

2 Projections

One of the first things we do when we are faced with a lot of data is to get a grasp of it from both domain perspective and statistical perspective. More often than not, any real-world data is comprised of large number of features either because each record inherently is comprised of large number of input features (i.e., there are lots of sensors or the logs contain many aspects of each entry) or because we have engineered a large number of features on top of the input data. High dimensionality has its own problems.

[2] https://en.wikipedia.org/wiki/Iris_flower_data_set (Retrieved September 5, 2018).

- First, it becomes difficult to visualize the data and understand its structure. A number of methods help optimally project the data into two or three dimensions to exaggerate the signal and suppress the noise and make data visualization possible.
- Second, just because there are a large number of features does not mean that the data is inherently high dimensional. Many of the features might be correlated with other features (e.g., age, income, and education levels in census data). In other words, the data might lie in a lower dimensional "manifold" within the higher dimensional space. Projection methods uncorrelate these dimensions and discover the lower linear and nonlinear manifolds in the data.
- Finally, the *curse of dimensionality* starts to kick in with high dimensional data. The amount of data needed to build a model grows exponentially with the number of dimensions. For all these reasons a number of projection techniques have evolved in the past. The unsupervised projection techniques are discussed in this chapter. Some of the supervised projection techniques (e.g., Fisher Discriminant Analysis) are discussed in the Supervised Learning chapter.

In this section, we will introduce three different types of projection methods: principal components analysis, self-organizing maps, and multidimensional scaling.

2.1 Principal Components Analysis

Principal components analysis (PCA) is one of the oldest and most commonly used projection algorithms in machine learning. It linearly projects a high dimensional multivariate numeric data (with possibly correlated features) into a set of lower orthogonal (uncorrelated) dimensions where the first dimension captures most of the variance, next dimension—while being orthogonal to the first—captures the remaining variance, and so on. Before we go into the mathematical formulation of PCA, let us take a few examples to convey the basic intuition behind orthogonality and principalness of dimensions in PCA.

- **The Number System**: Let us take a number (e.g., 1974) in our base ten number system. It is represented as a weighted sum of powers of 10 (e.g., $1974 = 1 \times 10^3 + 9 \times 10^2 + 7 \times 10^1 + 4 \times 10^0$). Each place in the number is independent of another (hence orthogonal), and the digit at the ones place is least important, while the digit at the thousands' place is the most important. If we were forced to mask one of the digits to zero by *minimizing the loss of information*, it would be the ones place. So here thousands' place is the first principal component, hundreds' is the second, and so on.
- **Our Sensory System**: Another example of PCA concept is our sensory system. We have five senses—vision, hearing, smell, taste, and touch. There are two

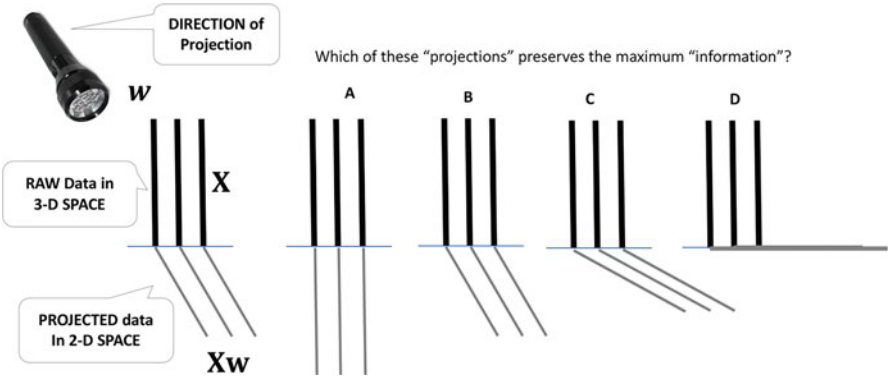

Fig. 15.4 The idea of a projection and loss of information as a result of projection

properties we want to highlight about our sensory system: First, all the five senses are orthogonal to each other, that is, they capture a very different perspective of reality. Second, the amount of information they capture about reality is not the same. The vision sense perhaps captures most of the information, followed by auditory, then taste, and so on. We might say that vision is the first principal component, auditory is the second, and so on. So PCA captures the notion of orthogonality and different amount of information in each of the dimensions.

Let us first understand the idea of "projection" and "loss of information." Figure 15.4 shows the idea of a projection in another way. Consider the stumps in a cricket game—this is the raw data. Now imagine we hold a torch light (or use the sun) to "project" this three-dimensional data on to the two-dimensional field. The shadow is the projection. The nature of the shadow depends on the angle of the light. In Fig. 15.4 we show four options. Among these options, projection A is "closest to reality," that is, loses minimal amount of information, while option D is farthest from reality, that is, loses all the information. Thus, there is a notion of the "optimal" projection w.r.t a certain objective called "loss of information."

We will use the above intuition to develop an understanding of principal components analysis. We first need to define the notion of "loss of information due to projection." Let $\mathbf{X} = \left[\boldsymbol{x}_n^T\right]_{n=1}^N$ be the $N \times D$ data matrix with N rows where each row is a D-dimensional data point. Let $\boldsymbol{w}^{(k)}$ be the k^{th} principal component, constrained to be a unit vector, such that $k \leq \min\{D, N-1\}$. We will use the same four-stage process to develop PCA:

Intuition: The real information in the data from a statistical perspective is the *variability* in it. So any projection that maximizes the variance in the projected space is considered the first principal component (direction of projection), $\boldsymbol{w}^{(1)}$. Note that since the input data $\mathbf{X}$ is (or is transformed to have) zero mean, its linear projection will also be zero mean (Fig. 15.5).

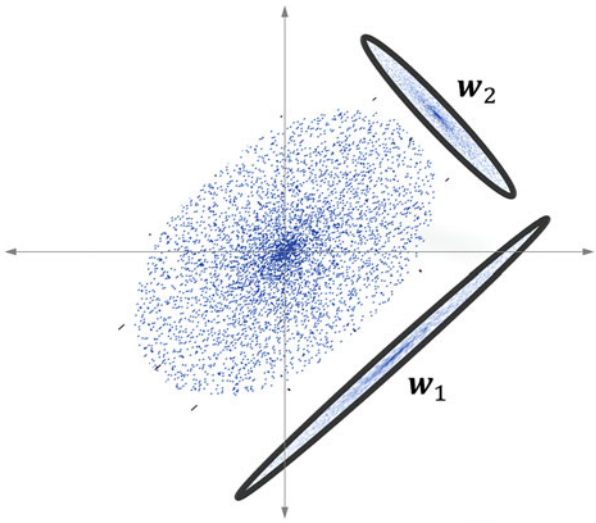

Fig. 15.5 The first and second principal components of a zero-mean data cloud

Formulation: Let $y_n = \boldsymbol{x}_n^T \boldsymbol{w}$ be the projection of the data point on $\boldsymbol{w}$. The variance of the entire projected data is given by: $\sum_{n=1}^{N} (y_n)^2 = \sum_{n=1}^{N} \left(\boldsymbol{x}_n^T \boldsymbol{w}\right)^2$. Thus, the first principal component is found by solving:

$$\boldsymbol{w}_1 = \arg \max_{\|\boldsymbol{w}\|=1} \sum_{n=1}^{N} \left(\boldsymbol{x}_n^T \boldsymbol{w}\right)^2$$

Modification: The above is a "constrained optimization problem" where the constraint is that the projection is a unit vector, that is, $\|\boldsymbol{w}\| = 1$. We can rewrite this constrained objective function as an unconstrained objective function as follows:

$$\boldsymbol{w}_1 = \arg \max_{\|\boldsymbol{w}\|=1} \|\mathbf{X}\boldsymbol{w}\|^2 = \arg \max \left\{ \frac{\boldsymbol{w}^T \mathbf{X}^T \mathbf{X} \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{w}} \right\}$$

Optimization: The standard solution to this problem is the first ***eigenvector*** of the positive semi-definite matrix $\mathbf{X}^T\mathbf{X}$, that is, $\boldsymbol{w}_1 = eig_1(\mathbf{X}^T\mathbf{X})$, with the maximum value being the first **eigenvalue**, λ_1.

The $\boldsymbol{k^{th}}$ principal component is derived by first removing the first $\boldsymbol{k-1}$ principal components from $\mathbf{X}$ and then finding the first principal component in the residual ($\mathbf{X_0} = \mathbf{X}$):

$$\mathbf{X}_k = \mathbf{X}_{k-1} - \mathbf{X}\boldsymbol{w}_{k-1}(\boldsymbol{w}_{k-1})^T$$

And from this, we iteratively find:

$$w_{k-1} = \arg \max_{\|\boldsymbol{w}\|=1} \|\mathbf{X}_k \boldsymbol{w}\|^2 = \arg \max \left\{ \frac{\boldsymbol{w}^T \mathbf{X}_k^T \mathbf{X}_k \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{w}} \right\}$$

In general, the first k principal components of the data correspond to the first k eigenvectors (that are both orthogonal and decreasing in the order of variance captured) of the covariance matrix of the data. The percent variance captured by the first d principal components out of D is given by the sum of squares of the first d eigenvalues of $\mathbf{X}^T\mathbf{X}$.

$$S(d|D) = \frac{\lambda_1^2 + \lambda_2^2 + \cdots + \lambda_d^2}{\lambda_1^2 + \lambda_2^2 + \cdots + \lambda_D^2}$$

Figure 15.6 shows the eigenvalues (above) and the fraction of variance captured (below) as a function of the number of principal components for MNIST data which is 28 × 28 images of handwritten data. Top 30 principal components capture more than 95% of variance in the data. The same can be seen in Figure 15.7 that shows

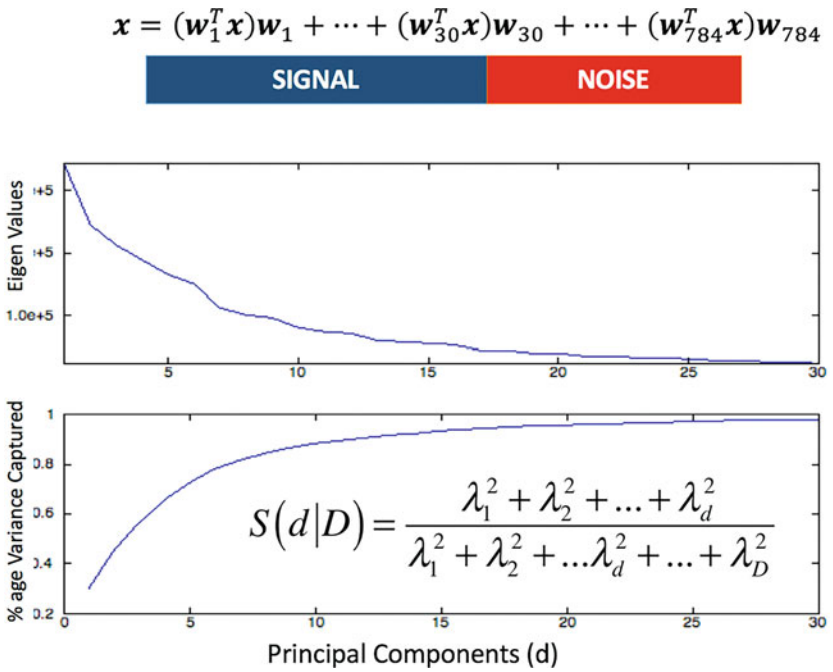

Fig. 15.6 Eigenvalues of the first 30 principal components for MNIST and fraction of variance captured

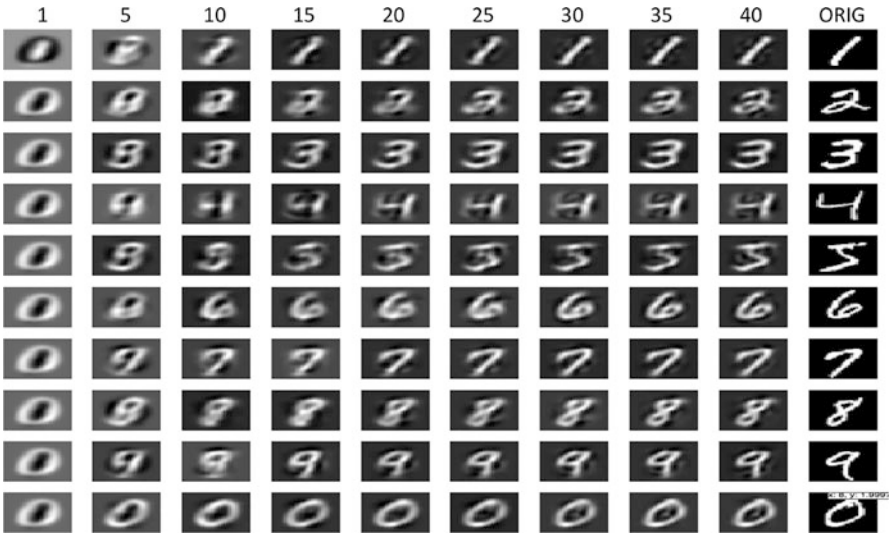

Fig. 15.7 Reconstruction of the digits by projecting them into k dimensions and back

the reconstruction of the ten digits when the data is projected to different number of principal components and reconstructed back. Again, we can see that although the original data is 784-dimensional (28×28), the top 30–40 dimensions capture the essence of the data (signal).

A number of other such linear (e.g., independent components analysis) and nonlinear (e.g., principal surfaces) projection methods with different variants of the loss of information objective function have been proposed. PCA is a "type" of projection method. Next we study another "type" of projection method which is very different in nature compared to the PCA-like methods.

To learn more about principal components analysis, refer to Chap. 3 (Sect. 3.4.3) in Han et al. (2011), Chap. 12 (Sect. 12.2) in Murphy (2012), and Chap. 14 (Sect. 14.5.1) in Friedman et al. (2001).

2.2 Self-Organizing Maps

Another classical approach to project data into 2–3 dimensions is self-organizing map (SOM) approach that uses competitive self-organization to smear the input data on a predefined grid structure.

Intuition: Figure 15.8 shows the basic intuition behind SOM. The left part shows the original data in a high dimensional space. Two points close to each other in this original space map to either the same or nearby grid points on the right grid also known as the "map."

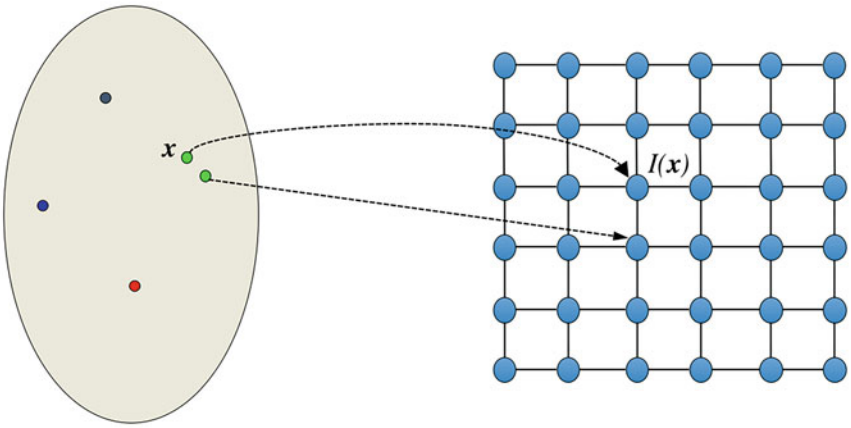

Fig. 15.8 Nearby points in the original space (left) map to nearby or same point in the SOM grid (right)

Formulation: A SOM is defined say M grid points organized typically in a rectangular (each grid point has four neighbors) or hexagonal (each grid point has three neighbors) grid. Each grid point is associated with a weight vector (the parameters) and a neighborhood structure.

Let $\mathbf{X} = \left\{\boldsymbol{x}_n \in \boldsymbol{R}^D\right\}_{n=1}^N$ be the set of N data points each in a D dimensional space. Let $\mathbf{W} = \{\boldsymbol{w}_m\}_{m=1}^M$ be the weights associated with the M grid points. The goal is to learn these weights so they "quantize" the input space in such a way that the weights associated with nearby grid points are similar to each other, that is, there is a smooth transition between weights on the grid.

Initially (in iteration $t = 0$) the weights are set to random. Then with each iteration the weights are updated through competitive learning, that is, (a) each data point is first associated with that grid point whose weights are closest to the data point itself, (b) then this grid point weights are updated to move closer to the data point, and (c) not only that, the weights of the "nearby grid points" are also moved toward this data point, albeit to a slightly lesser degree.

Optimization: SOM is learnt through an iterative algorithm where (a) each data point is first associated with the nearest grid point and (b) weights of all the grid points are updated depending on how far they are from the grid point associated with the data point. SOM starts with random initial weights: $\mathbf{W}(0) = \{\boldsymbol{w}_m(0)\}_{m=1}^M$ and updates these weights iteratively as follows:

- Associate each data point with its nearest grid point (image of the data point) in iteration t

$$I_t(n) = \arg \min_{m=1\ldots M} \mid |\boldsymbol{x}_n - \boldsymbol{w}_m(t)| \mid$$

- Compute degree of association $\theta_t(n, m)$ between the n^{th} data point and the m^{th} grid point, such that it decreases with the distance between m and $I_t(n)$, $\delta(I_t(n), m)$:

$$\theta_t(n, m) = \exp\left(-\frac{\delta(I_t(n), m)}{\sigma(t)^2}\right), \forall m = 1 \ldots M$$

- Now each of the grid point weights $\boldsymbol{w}_m$ is updated in the direction of the input $\boldsymbol{x}_n$ with different degrees that depends on $\theta_t(n, m)$:

$$\boldsymbol{w}_m(t+1) = \boldsymbol{w}_m(t) + \eta(t)\theta_t(n, m)(\boldsymbol{x}_n - \boldsymbol{w}_m(t))$$

- Decrease the learning rate $\eta(t)$ and the variance $\sigma(t)$ as iterations progress.

Figure 15.9 shows a semantic space of a news corpus comprising of millions of news articles from the last ten years of a country. Here word embeddings (300-dimensional semantic representation of words such that two words with similar meaning are nearby in the embedding space) of the top 30K most frequent words (minus the stop words) are smeared over a 30 × 30 SOM grid. Two words close to each other in the embedding space are mapped to either the same or nearby grid

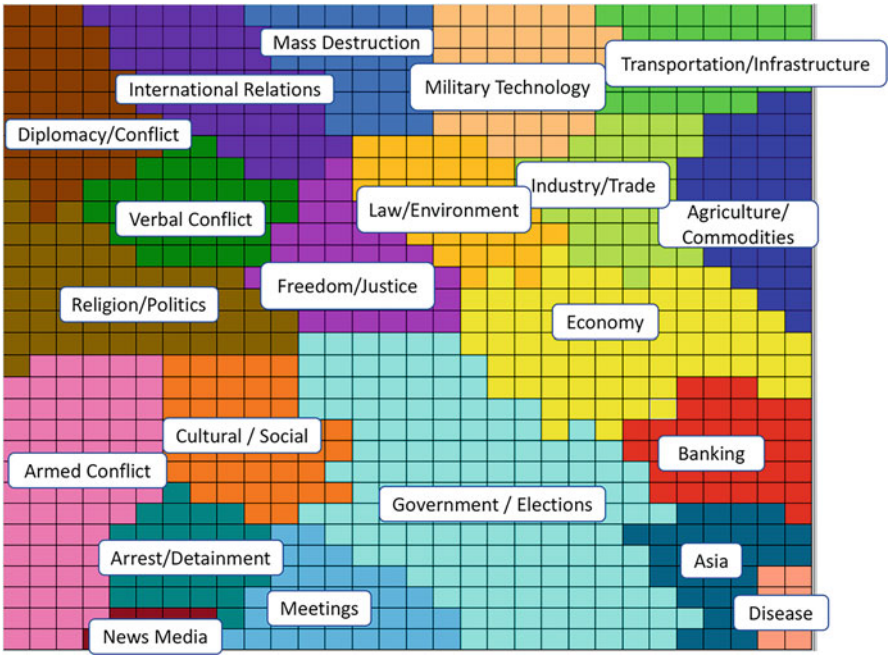

Fig. 15.9 Word embeddings of a large news corpus visualized on a 2D SOM

points. The grid vectors quantize different parts of the semantic embedding space representing different meanings. These grid point vectors are further clustered into macro concepts shown on the map.

SOMs smear the entire data into a 2D grid. Sometimes, however, we do not want to put the projected data on a grid. Additionally, we are not given a natural representation of data in a Euclidean space. In such cases, we use another class of projection method called multidimensional scaling.

The reader can refer to Chap. 14 (Sect. 14.4) in Friedman et al. (2001) to learn more about self-organizing maps.

2.3 *Multidimensional Scaling*

PCA and SOM are two different kinds of projection/visualization methods. In PCA, we project the data linearly to minimize loss of variance, while in SOM we quantize each data point into a grid point via competitive learning. Another way to map a high dimensional data into a low dimensional data is to find each data point's representatives in the lower dimensions such that the distance between every pair of points in the high dimension matches the distance between their representatives in the projected space. This is known as multidimensional scaling (MDS).

Intuition: The idea of "structure" in data manifests in many ways—correlation, variance, or pairwise distances between data points. In MDS, we find a representative (not a quantization as in SOM) for each data point in the original space such that the distance between two points in the original space is preserved in the MDS projected space. Figure 15.10 shows the basic idea behind MDS.

Formulation: Let $\mathbf{D} = [\delta_{ij}]$ be the $N \times N$ distance matrix between all pairs of N points in the original space. Note that techniques like MDS do not require

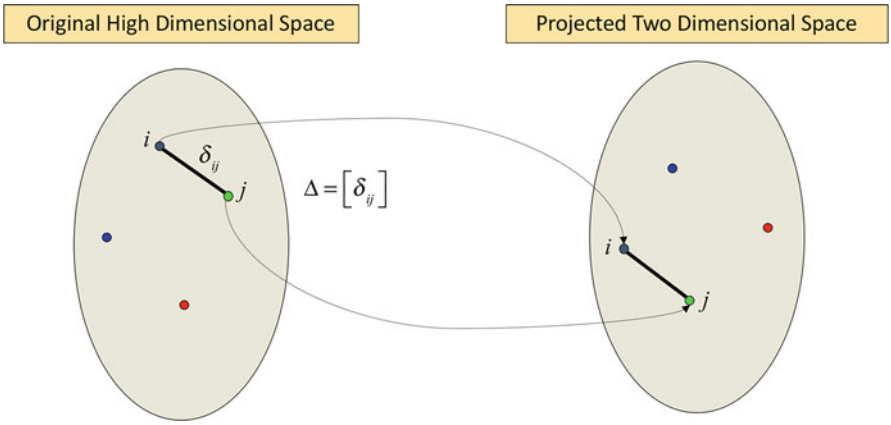

Fig. 15.10 Multidimensional scaling preserves pairwise distances between all pairs of points

the original data to be a multivariate data. As long as we can compute distance between pairs of points (e.g., Euclidian distance between multivariate real-valued vectors, cosine similarity between word or paragraph or document embeddings, TFIDF cosine similarity between two documents, Jaccard coefficient between two market baskets, even a subjective score by "smell expert" on how similar two smells are, or any engineered distance or similarity function) MDS can be applied. In MDS, each data point is associated with a low dimensional representative vector, that is, let $\mathbf{X} = \{\boldsymbol{x}_n\}_{n=1}^{N}$ be the N representative vectors (parameters to be learnt), one for each data point. The goal is to find $\mathbf{X}$ such that the distance between every pair of points $\Delta(\boldsymbol{x}_i, \boldsymbol{x}_j)$ in the MDS space matches as much as possible the actual distance δ_{ij} between the corresponding points, that is,

$$J(\mathbf{X}) = \min_{\mathbf{X}} \sum_{1 \le i < j \le N} \left(\Delta\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) - \delta_{ij}\right)^2$$

Modifications: Different variants of proximity preserving embeddings have been developed over time.

- **Multidimensional Scaling**: The original objective is modified by dividing with the sum squared of all the distances. This is done to make sure that the overall distances between points do not grow.

$$J_{MDS}(\mathbf{X}) = \frac{\sum_{1 \le i < j \le N} \left(\Delta\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) - \delta_{ij}\right)^2}{\sum_{1 \le i < j \le N} \Delta\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)^2}$$

- **Sammon Map**: The intuition behind this is that when two points are very far from each other in the original space, then the error between their distances in the projected space and the original space matters less. Only when points are close to each other in the original space that the error matters.

$$J_{SPE}(\mathbf{X}) = \sum_{1 \le i < j \le N} \frac{\left(\Delta\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) - \delta_{ij}\right)^2}{\delta_{ij}}$$

Figure 15.11 shows a 2D map of the various product categories of a grocery store. This map was created by first learning the strength of co-occurrence consistency between all pairs of categories from the point-of-sale data. Consistency measures the degree with which a pair of product categories is purchased ***together more often than random***. Two products that are consistently closer to each other (e.g., meat and seafood) land up in close proximity in the 2D space as well. This visualization reveals not only the structure in the purchase co-occurrence grammar of the customers but can also be used to change the store layout to match customer buying patterns or create rules for recommending products, etc.

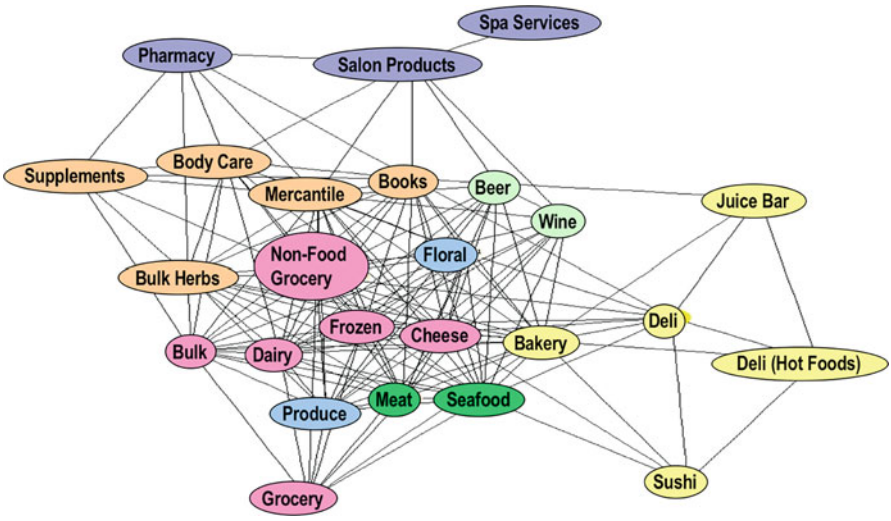

Fig. 15.11 Store layout based on co-occurrence of products from various categories

In this section, we have studied a variety of algorithms that help understand the data better by projecting it to a two- or three-dimensional space and creating different kinds of visualizations around them—histograms, scatter plots, self-organizing-maps, and multidimensional scaling. Next, we explore one of the most popular paradigms called clustering in the unsupervised learning suite of algorithms.

The reader can refer to Chap. 14 (Sects. 14.8 and 14.9) from Friedman et al. (2001) to learn more about multidimensional scaling.

3 Clustering

The fundamental hypothesis in finding structure in data is that while a dataset can be very large, the underlying processes that generated the data has only finite degrees of freedom. There are only a small number of actual latent sources of variations from which the data actually emerged. For example,

- In retail point-of-sale data, we might see a lot of variation from customer to customer but inherently there are only a finite types of customer behaviors based on their lifestyle (e.g., brand savvy, frugal), life-stage (e.g., bachelor, married, has kids, old age), purchase behavior (e.g., when, where, how much, which channel) and purchase intents (grocery, birthday, vacation related, home improvement, etc.). We might not know all these variations or combinations in advance, but we know we are not dealing with an infinite number of such variations and we can discover such quantization if we let the data speak for itself.

- Similarly, while it seems that there are billions of videos on YouTube or billions of pages on the web, or millions of people on LinkedIn and Facebook, the different types of videos (music albums, home videos, vacation videos, talent videos, cat videos, etc.), pages (news, spam, blogs, entertainment, etc.), or people (software engineers, managers, data scientists, artists, musicians, politicians, etc.) is a reasonably finite set. Whether we know all the types or not is another question, but what we definitely know is that the number of such types is not as many as the number of entities.
- Similarly, consider all words in a language. It appears that there are many words in the dictionary but they can again be grouped by, say, parts of speech, root, tense, and meaning, into only a small number of types.
- Finally, consider telematics data while someone is driving the car. Again, the data variation might be very large across all cars but the number of things people do while driving (soft or hard brake, soft or hard acceleration, sharp or comfortable left or right turns, etc.) combined with the number of driving scenarios (pot-holes uphill, downhill, highway, inner-roads, etc.) is still finite.

When these finite "sources" of variation in the data are already known in advance and/or when we have to map the data variations into a specific set of *known* types, this becomes a **classification problem**. We deal with classification in the supervised learning chapter at length. On the other hand, when these variations are *not known* in advance and need to be *discovered*, by grouping similar data points together (whatever "similar" means for that type of data), then it becomes a **clustering problem**.

In many systems of intelligence including our own, we transition from clustering to classification. For example, in early childhood, babies do not know all the variations of what they see or hear so they internally do clustering to quantize these variations. If they see more data of a certain type, the resolution of quantization on those parts of the data becomes fine-grained. At this stage, we do know that this is similar to what I have seen before (quantized symbol number 48), but we do not know yet what to call it. As we grow and language develops, we learn that those quantization have been given names (vertical line, sleeping line or "nose," "eyes," "square," "triangle," etc.). Now we have some *known* quantization that we call **classes**, and when a new experience comes, we first try to map it to a known class (e.g., if a child has never seen a goat before, she might "classify" it as a "dog"), but if this quantization is not "close enough" to any of the known classes, then she might ask the mother—it looks like a dog, is it a dog? And when she gets a new label (no it is called a "goat"), she creates another class in the brain. In this stage, we rely not only on the known set of named classes but are also open to discovering beyond the known. This is where we are in the *hybrid* stage of learning—exploit the known and explore the unknown simultaneously. As we grow older and we have "seen enough," the number of new quantization reduces as we have a sufficiently large number of classes to represent all inputs and nothing seems to surprise us anymore.

Another example of this process of transition from clustering to classification happens in customer feedback when we move from an early stage product to a maturity stage product, say, when we build our first product (an app, a service, a physical product, etc.) and we start to get customer feedback. As we start to go through this feedback, we realize that there are only so many variations that we are seeing. For example, in an online retail business, customers might complain about delivery time, delivery charges, wrong product delivered, login problems, etc. In restaurant business, customers might complain about quality of service, ambience, quality of food, price, etc. In fleet management (Ola, Uber), customers might complain about quality of car, delay in pickup, cancellation by driver, driver rudeness, driving safety, etc. Now initially when we do not know what these categories might be, we just cluster similar text feedback together based on keywords and assign name to these clusters. Once we know these clusters, we can then create a menu system based on the most common types of complaints and can transition to a more structured feedback than unstructured text feedback in the early days of the product. Thus, while clustering and classification are two very different paradigms, they are related to each other. This, in fact, is one of the most important use cases of clustering—to *discover* the quantized states of the system, that is, the sources of variations in the data. In this section, we will explore three broad clustering approaches: partitional, agglomerative, and spectral.

3.1 Partitional Clustering

If we assume a certain number of clusters and try to partition the data into those many clusters, then it is called partitional clustering. Different algorithms, most notably *K-means clustering*—that partitions the data into K clusters—are examples of partitional clustering. Consider a multivariate dataset where we can define Euclidean distance between two points meaningfully, that is, we have already transformed all the features and z-scored them.

Let $\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ be the N data points that need to be clustered into K clusters ($1 \leq K \leq N$). If $K = 1$, that means the entire data is clustered into one cluster. In that case, the mean of the entire dataset is the cluster center we are looking for. In case $K = N$, then each data point is by itself a cluster center. Both these are valid but not useful extreme cases. Typically, the value of K is somewhere in between. We will first formulate this as an optimization problem using the same process as above—intuition, formulation, modification, and optimization.

Intuition: Clustering is about "grouping similar things (feature vectors representing them) together." There are two equivalent ways to represent a "clustering": ***Enumeration*** vs ***Representation***. In ***enumeration***, we can explicitly label each data point with the cluster id it belongs to. Let $\delta_{n,k} \in \{0, 1\}$ be a set of binary labels such that it is ones if n^{th} data point is associated with the k^{th} cluster and zero otherwise. In ***representation***, each cluster is represented by a cluster mean of all data points it represents.

Formulation: Let $\mathbf{M} = \{\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_K\}$ be the K cluster means—the representatives of the K clusters—that we are looking for. In a way we are quantizing the raw data by these cluster centers that act as representatives of the data. The objective is to find such representatives that approximate the data the best, that is, the error of approximation is minimum. Now if data point $\boldsymbol{x}_n$ is represented by the cluster center $\boldsymbol{m}_k$, that is, $\delta_{n,k}$ is 1 and $\delta_{n,\ell} = 0$ for all other $\ell \neq k$, then the objective we are trying to minimize is the sum (squared) of the distance between each data point and ***its*** representative:

$$J(\mathbf{M}, \boldsymbol{\Delta}) = \sum_{n=1}^{N} \sum_{k=1}^{K} \delta_{n,k} \|\boldsymbol{x}_n - \boldsymbol{m}_k\|^2$$

Optimization: In the above equation, there are two kinds of parameters—the enumeration parameters, $\boldsymbol{\Delta} = [\delta_{n,k}]$, that associate a data point with a cluster, and representation parameters, $\mathbf{M}$, the mean of each cluster. Note that, both classes of parameters are interdependent on each other, that is, if $\mathbf{M}$ is known, then $\boldsymbol{\Delta}$ can be computed, and if $\boldsymbol{\Delta}$ is known, then $\mathbf{M}$ can be computed. This is an example of a class of optimization problems that can be solved using **Expectation–Maximization (EM)** algorithms that alternate between two steps in each iteration: expectation step and maximization step.

(a) **Expectation step** (the E-step) updates $\boldsymbol{\Delta}_t$ given the current value of $\mathbf{M}_t$ by *associating* a data point with its ***nearest*** representative:

$$\delta_{n,k}^{(t)} \leftarrow 1 \text{ if } \left(k = \arg \min_{j=1\ldots K} \left\| \boldsymbol{x}_n - \boldsymbol{m}_j^{(t)} \right\|^2\right), 0 \text{ otherwise.}$$

(b) **Maximization step** (the M-step) updates $\mathbf{M}_{t+1}$ given the current value of $\boldsymbol{\Delta}_t$ by *maximizing* the above objective function resulting in:

$$\frac{\partial J(\mathbf{M}, \boldsymbol{\Delta})}{\partial \boldsymbol{m}_k} = 2 \sum_{n=1}^{N} \delta_{n,k} (\boldsymbol{x}_n - \boldsymbol{m}_k) = 0 \Rightarrow \boldsymbol{m}_k^{(t+1)} \leftarrow \frac{\sum_{n=1}^{N} \delta_{n,k}^t \boldsymbol{x}_n}{\sum_{n=1}^{N} \delta_{n,k}^t}$$

Figure 15.12 shows these two steps pictorially on how a complete EM iteration works. Figure 15.12a shows two randomly initialized cluster centers. Figure 15.12b shows how given those two initial cluster centers, the data points are enumerated by the cluster they are closest to (orange vs. green). Figure 15.12c shows how with the new associations the cluster centers are updated using the M-step. Figure 15.12d shows the final association update leading to the desirable clustering of the data.

There are several properties of K-means clustering that are likeable and some that are not:

Sensitivity to Initialization—In case of PCA, the solution to the objective function was what we call a "closed-form solution" because there is only one optimal answer there. But clustering does not have such an objective function that

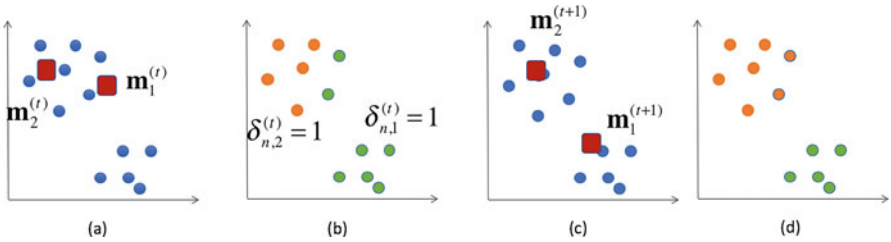

Fig. 15.12 (**a**) Initial cluster centers, (**b**) E-step associating data points with one cluster or the other, (**c**) M-step updating the cluster centers, (**d**) next E-step shows convergence

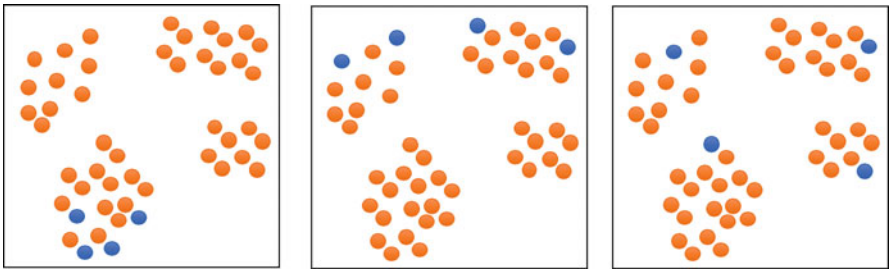

Fig. 15.13 Clustering is sensitive to initialization. Three different possible random initializations (blue) that might either result in different final clusters or more iterations to convergence

gives us one final answer. Here, the final clusters learnt depend on the way we have initialized the clustering. Figure 15.13 shows three different initializations of the same data. Depending on the initialization, the final cluster might either be suboptimal or take longer to converge to the optimal even if it is possible. But random initialization could give any of these or other combinations as the initial clusters, and hence K-means is not always guaranteed to give the same clusters. As a general rule, we do not like "non-determinism" in our algorithms—no guarantee that we will get the same results for the same data and the same hyper parameters (number of clusters).

Smart Initialization: There are a number of algorithms that have been proposed to make K-means clustering more "optimal" and "deterministic" from an initialization perspective. One such initialization method is the farthest first point (FFP) initialization where we choose the first cluster center deterministically, that is, pick the data point farthest from the mean of the entire data. Then we choose the second cluster center that is farthest from the first. The third is picked such that it is farthest from the first two, and so on. Figure 15.14 shows one such initialization where first figure shows the first two clusters picked. Middle figure shows how the next cluster is chosen such that it is farthest from the first two, and third is chosen such that it is farthest from the first three. This guarantees good coverage of the space and leads to a decent initialization, resulting in a closer to optimal clustering.

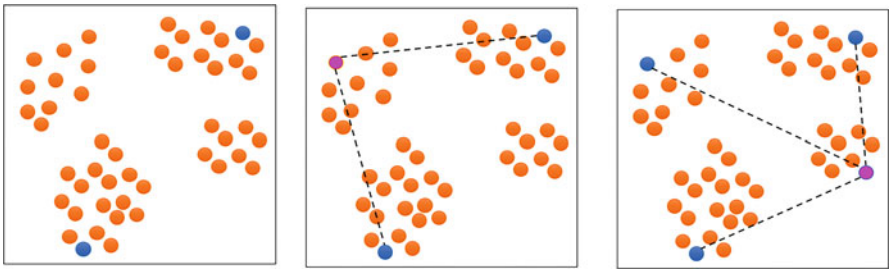

Fig. 15.14 Farthest first point initialization. The first figure shows two cluster centers initialized. Middle figure shows how the third is picked such that it is farthest from both the first two and the fourth is farthest from all three

Scale and Speed—K-means clustering and such partitional algorithms are highly scalable. The overall complexity of K-means is O(NKDT) where N is the number of data points, K is the number of clusters, D is the dimensionality of the data (quantifying the time it takes to compute the distance between a data point and a cluster center), and T is the number of iterations it takes for a K-means to converge. Their linearity in all these dimensions makes such partitional algorithms so popular. Not only that, K-means clustering is also highly parallelizable at two levels. In the map-reduce sense, if (a) we make previous iteration cluster centers available to all mappers, (b) if a map job—while processing one record at a time—generates its nearest cluster index as the key and the record itself as value, and (c) reducer takes the average of all the data points of the same key, then we can achieve K-fold parallelism for K-means clustering. Alternately, the distance computation using Euclidean distances between a data point and all the cluster centers can be parallelized using GPUs.

The Distance Function—The above formulation works well for Euclidean distances, but when the data is not represented as a point in space (only distances between pairs of points are given), we cannot use K-means or its variants. Then, we have to rely on other algorithms. When the data point is not a point in a Euclidean space but an L1 normalized (e.g., clustering probability distributions) or L2 normalized (e.g., TFIDF representation of documents) space, then we need to tweak K-means algorithm slightly. In an L2 normed TFIDF vector space, all documents are in a hyper-sphere with their L2 norms being 1. In **spherical K-means clustering**, the cluster centers are also forced to be in the same "spherical" L2 normed space as the original data. Here instead of computing distance we assign a document to that cluster whose cosine similarity is maximum among all clusters. Second, after computing the mean, we renormalize the mean vectors into L2 normed vectors. These two modifications to K-means clustering make it amenable to L2 normed data.

Number of Clusters: Another problem with K-means clustering is that it requires the number of clusters K as a hyper-parameter. Now without any prior knowledge or understanding of the data, we cannot say what is the right number

of clusters. Often this number is decided through a rigorous statistical analysis by trying different values of K and measuring a quantity such as the "gap" between random clustering and actual clustering with a certain value of K (this is also known as the "gap statistic"). There are other such mechanisms that can be deployed to find the right number of clusters for the data. Another option is to let the business limitations decide the number K. For example, if we can only create five unique campaigns for all our customers, then we may want to segment our customers only into five clusters and create one campaign for each.

Heterogeneous Clusters: K-means clustering only discovers homogeneous and spherical clusters. For example, when clusters are of different sizes from each other, of different densities, or of different shapes, then K-means clustering does not do a good job of discovering them. This is because of the Euclidean distance used in K-means which makes clustering look for hyper-spherical clouds in the data space. Other methods such as agglomerative clustering are typically used to discover elongated clusters, and mixture of Gaussians is able to model arbitrary shaped clusters. These will be covered later.

Hard vs. Soft Clustering: Most machine learning algorithms have two variations—the "hard" or brittle version and the "soft" or robust version. In K-means clustering, when it is decided that a certain data point is closest to one of the clusters, it is "hard-assigned" to that cluster only and to none other, that is, $\delta_{n,k} \in \{0, 1\}$. This has two problems: First, it ignores the actual distance between the data point and the cluster center. If the distance between the data point and the cluster center is small, then the "degree-of-belongingness" of this point to this cluster should be higher. Second, if a data point is just at the boundary—only barely closer to one cluster mean than the other—then this assigns that point to the cluster it is closer to—so it does not take the second nearest into account. To alleviate these problems, we do a *softer* version of K-means clustering known as the ***soft K-means clustering*** where instead of doing a hard assignment, we can define a soft or probabilistic assignment $\delta_{n,k} \in [0, 1]$ between each data point and each cluster center. In soft clustering, we define the degree of association of a data point with a cluster as inversely proportional to its distance from the cluster center going from soft to hard as iterations progress.

$$\delta_{n,k}^{(t)} \leftarrow \frac{\exp\left(-\frac{\|\boldsymbol{x}_n - \boldsymbol{m}_k\|^2}{\sigma^2(t)}\right)}{\sum_{j=1}^{K} \exp\left(-\frac{\|\boldsymbol{x}_n - \boldsymbol{m}_j\|^2}{\sigma^2(t)}\right)}$$

To learn more about K-means clustering and expectation maximization, one can read Chap. 10 (Sect. 10.2.1) and Chap. 11 (Sect. 11.1.3) in Han et al. (2011), Chap. 11 (Sect. 11.4) in Murphy (2012), and Chap. 6.12 (Sect. 6.12) in Michalski et al. (2013).

3.2 Hierarchical Clustering

Partitional clustering assumes that there is only one "level" in clustering. But in general, the world is made up of a "hierarchy of objects." For example, the biological classification of species has several levels—domain, kingdom, phylum, class, order, family, genus, and species. All the documents on the web can be clustered into coarse (sports, news, entertainment, science, academic, etc.) to fine grained (hockey, football, ..., or political news, financial news, etc.). To discover such a "hierarchical organization" from data, we do hierarchical clustering in two ways: top-down and bottom-up.

Top-down hierarchical clustering also known as ***divisive clustering*** where we apply partitional clustering recursively first to, say, find K1 clusters at the first level of the hierarchy, then within each find K2 clusters and so on. With the right number of levels and number of clusters at each level (which may be different in different parts of the hierarchy), we can now discover the overall structure in the data in a top-down fashion. This, however, still suffers—at each level in the hierarchy—the problems that a partitional clustering algorithm suffers from—initialization issues, number of clusters at each level, etc. So it can still give a variety of different answers and the problem of non-determinism remains.

Bottom-up hierarchical clustering also known as ***agglomerative clustering*** is the other approach to building the hierarchy of clusters. Here we start with the raw data points themselves at the bottom of the hierarchy, and we find the distances between all pairs of points and merge the two points that are nearest to each other since they make the most sense to "merge." Now the merged point replaces the two points that were merged and we are left with $N - 1$ data points when we started with N data points. The process continues as we keep merging two data points or clusters together until the entire data is merged into a single root node. Figure 15.15 shows the result of a clustering of ten digits (images) in a bottom-up fashion. The structure

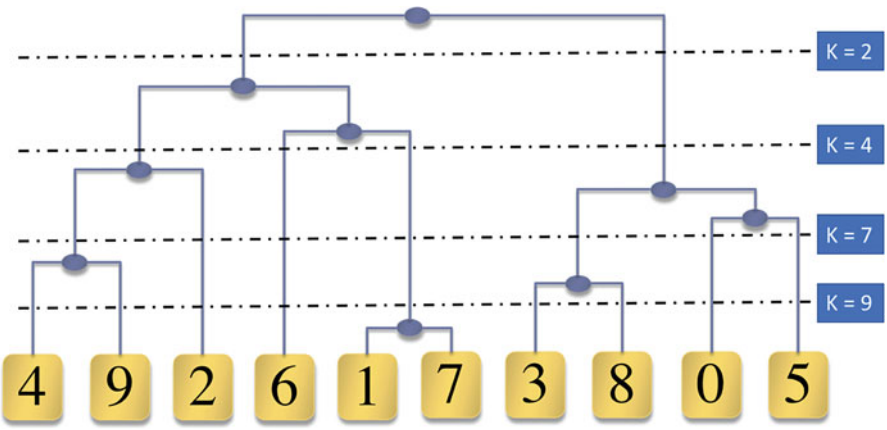

Fig. 15.15 Bottom-up agglomerative clustering of digits—tree structure

is called a **dendrogram** that shows how at each stage two points or clusters are merged together. First, digits 1 and 7 got merged. Then 3 and 8 got merged, then 4 and 9 got merged, then 0 and 5 got merged, then the cluster {3,8} and {0,5} got merged, and so on. The process leads eventually to a binary tree that we can cut at any stage to get any number of clusters we want.

The key to agglomerative clustering is the definition of distance between two "clusters" in general (e.g., clusters {3,8} and {0,5}). Different ways of doing this define different kinds of agglomerative clustering, resulting in different forms of clustering shapes. In the following, let $\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_P\}$ be the set of P points in cluster $\mathbf{X}$ and let $\mathbf{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_Q\}$ be the set of Q points in cluster $\mathbf{Y}$. Note that either P or Q or both can be 1. The distance between the set $\mathbf{X}$ and set $\mathbf{Y}$ can be defined in many ways:

- **Single linkage**—distance between two nearest points across $\mathbf{X}$ and $\mathbf{Y}$ is used as distance between the two clusters. This gives elongated clusters as two clusters with even one point close to one point of another cluster will be merged.

$$\Delta(\mathbf{X}, \mathbf{Y}) = \min_{p=1\ldots P} \left\{ \min_{q=1\ldots Q} \left\{ \Delta\left(\boldsymbol{x}_p, \boldsymbol{y}_q\right) \right\} \right\}$$

- **Complete linkage**—the other extreme of single linkage is where distance between two farthest points across $\mathbf{X}$ and $\mathbf{Y}$ is used as distance between clusters. Here the clusters discovered are more rounded as every point of one cluster must be close to every other point of the other cluster.

$$\Delta(\mathbf{X}, \mathbf{Y}) = \max_{p=1\ldots P} \left\{ \max_{q=1\ldots Q} \left\{ \Delta\left(\boldsymbol{x}_p, \boldsymbol{y}_q\right) \right\} \right\}$$

- **Average linkage**—is between the single and complete linkage clustering where distance between the two clusters is computed as the average distance between all pair of points among them. This makes clustering robust to noise.

$$\Delta(\mathbf{X}, \mathbf{Y}) = \frac{1}{\mathrm{PQ}} \sum_{p=1}^{P} \sum_{q=1}^{Q} \Delta\left(\boldsymbol{x}_p - \boldsymbol{y}_q\right)$$

There are several pros and cons of hierarchical agglomerative clustering.

- **Deterministic clusters**—Unlike partitional clustering (e.g., K-means) where the final cluster depends on initialization, agglomerative clustering always gives the same clustering for the same dataset and definition of distance. It does not depend on any initialization since there is no initialization.
- **Feature representation vs. distance function**—Partitional clustering works only on multivariate data where each data point must be a point in a Euclidean

space. Agglomerative clustering can work on datasets where only pairwise distances are given and data has no feature representation.

- **Scale**: One of the drawbacks of agglomerative clustering is that it is quadratic in the number of data points because, to begin with, we have to compute pairwise distances between all pairs of points. This makes it highly impractical as the number of data points increases. For very large datasets, it is possible to first do a large number (e.g., $K = \sqrt{N}$) of clusters to remove fine grained noise and then do hierarchical clustering on these K cluster centers as data points. Thus mixing partitional and hierarchical merges the best of both worlds.
- **Number of clusters**: Finally, agglomerative clustering gives us all the number of cluster we need. We can cut the dendrogram at any level to get that many clusters. This does not require us to start with prior knowledge about the number of clusters as a parameter.

The reader can refer to Chap. 25 (Sect. 25.5) from Murphy (2012), Chap. 14 (Sect. 14.3.12) from Friedman et al. (2001), and Chap. 3 (Sect. 3.4.3) from Han et al. (2011) for additional material on hierarchical clustering.

3.3 Spectral Clustering

Partitional clustering works on data with Euclidean feature spaces. Hierarchical clustering works on pairwise distance functions in a bottom-up fashion and recursive partitional clustering in a top-down fashion. There is another class of clustering algorithm that works on similarity graphs where each node represents an entity and weight on the edge; connecting the nodes quantifies similarity between the two edges. Spectral clustering is a very useful clustering algorithm in domains where it is easier to quantify such similarity measures between entities rather than representing them as a feature vector, for example, two LinkedIn profiles, two songs, two movies, and two stock market returns time series. Again, we will follow the four stages to develop a proper objective function for spectral clustering.

Intuition: Consider a graph with six nodes {a, . . . ,f} shown in Fig. 15.16. Edge weights indicate similarity between pairs of entities. In order to partition this graph into two parts, we must remove a subset of edges. The edges removed constitute "loss of information" which we want to minimize. Clearly removing the smallest weight edges makes sense as shown in Figure 15.16. Removing the three edges between nodes {a,d}, {c,d}, and {c,f} will result in two partitions {a,b,c} and {d,e,f} that by themselves are highly connected to each other.

Formulation: We translate the above intuition into an objective function. Let $\mathbf{W} = [w_{ij}]$ be the symmetric similarity matrix of size $N \times N$ where N is the number or nodes in the graph (i.e., number of LinkedIn profiles or number of movies among which we know similarity). One way to formulate this would be to introduce variables $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}$ where $x_n \in \{1, -1\}$ depending on whether after partitioning this graph into two parts, the node n belongs to the first partition

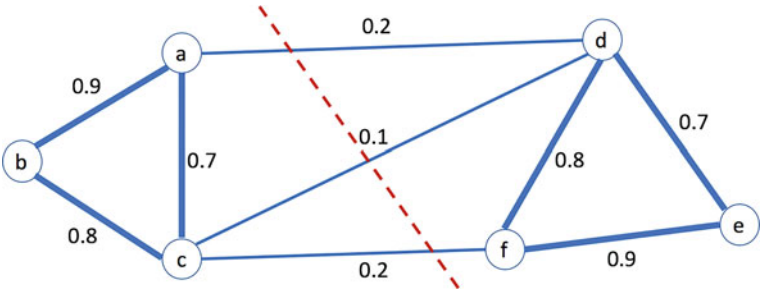

Fig. 15.16 A similarity graph with six nodes. Edge weights are similarity between corresponding pairs of entities. The graph is partitioned into two parts such that total weight of removed edges is minimum

(1) or the second partition (0). Now the intuition suggests that two nodes (i, j) should be in the same partition (i.e., $(x_i - x_j)^2$ is 0) if they are very similar (i.e., w_{ij} is high) and in different partitions (i.e., $(x_i - x_j)^2$ is 1) if they are dissimilar, that is, (w_{ij} is low). We can therefore capture the intuition by maximizing the following objective function.

$$J(\mathbf{X}|\mathbf{W}) = \frac{1}{2} \sum_{1 \le i,j \le N}^{N} w_{ij} \left(x_i - x_j\right)^2$$

Modification: As we solve the above objective, we get the following:

$$J(\mathbf{X}|\mathbf{W}) = \frac{1}{2} \sum_{1 \le i,j \le N}^{N} w_{ij} \left(x_i^2 + x_j^2 - 2x_i x_j\right) = \sum_{i=1}^{N} x_i^2 \left(\sum_{j=1}^{N} w_{ij}\right)$$
$$- \sum_{1 \le i,j \le N}^{N} x_i w_{ij} x_j = \boldsymbol{x}^{\boldsymbol{T}} (\mathbf{D} - \mathbf{W}) \boldsymbol{x}$$

where $\boldsymbol{D}$ is the diagonal matrix whose diagonal elements are sum of the rows of $\boldsymbol{W}$

$$\boldsymbol{D} = \begin{bmatrix} d_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & d_N \end{bmatrix}$$

where $d_n = \left(\sum_{j=1}^{N} w_{nj}\right)$. The matrix $\boldsymbol{L} = (\boldsymbol{D} - \boldsymbol{W})$ is called the unnormalized graph Laplacian of the similarity matrix $\boldsymbol{W}$. It is a positive semi-definite matrix with smallest eigenvalue 0 and the corresponding eigenvector as all 1's. If the graph has k connected components, that is, each connected component has no link across, then there will be k smallest eigenvalues equal to 0. Assuming the graph has only one

connected component, the second smallest eigenvector is used to partition the graph into two parts. We can take the median value of the second smallest eigenvector and partition the graph such that the nodes whose second eigenvector components are above the median are in one partition and the remaining nodes in the other partition. This partitioning can be applied recursively now to break the two components further into two partitions in a top-down fashion.

In this section, we have studied a number of clustering algorithms depending on the nature of the data. One of the open problems in clustering is how to systematically define distance functions when data is not a straightforward multivariate real-valued vector. This is where the critical domain knowledge is required. The next paradigm—density estimation—extends the idea of clustering by allowing us to describe each cluster with a "shape" called its density function.

For further reading, the reader can refer to Chap. 25 (Sect. 25.4) from Murphy (2012) and Chap. 14 (Sect. 14.5.3) from Friedman et al. (2001).

4 Density Estimation

The fundamental hypothesis that data has structure implies that it is not uniformly distributed across the entire space. If it were, it would not have any structure. In other words, all parts of the feature space are not equally ***probable***. Consider a space with two features "age" and "education." Let us say age takes a value from 0 to 100 years and "education" from, say, 1 to 20. Now probability P(age = 3, education = PhD) is zero and P(age = 26, education = PhD) is high. Similarly, P(age = 20, education = grade-1) is low and P(age = 5, education = grade-1) is high. Estimating this joint probability, given the data, gives us a sense of which combination of feature values are more likely than others. This is the essence of *structure* in the data, and density estimation captures such joint probability distributions in the data. Density estimation has many applications, for example:

- **Imputation**: If one or more of the feature values is missing, given the others we can estimate the missing value as the value that gives the highest joint probability after substituting it.
- **Bayesian classifiers**: Another application of density estimation is to build a "descriptive" classifier for each class where the descriptor is essentially a class conditional density function P($\mathbf{x}$|c).
- **Outlier detection**: Another important application of density estimation is outlier detection used in many domains such as fraud, cyber security, and when dealing with noisy data. A data point with low probability after we have learnt the density function is considered an outlier point.

There are two broad density estimation frameworks. First, is the ***nonparametric density estimation*** where we do not learn a model but use the "memory" of all the known data points to determine the density of the next data point. Parzen Window is an example of a nonparametric density estimation algorithm. Second

is the ***parametric density estimation*** where we first make an assumption about the distribution of the data itself and then fit the parameters of this function using maximum log likelihood optimization. If individual parametric density functions are not enough to represent the complexity of the data (e.g., data is multimodal), then we apply mixture of parametric density models (e.g., mixture of Gaussians).

4.1 Nonparametric Density Estimation

Let us first develop an intuition behind density functions from an example. Imagine that in a room floor we scatter a large number of magnets at specific locations. Each of these magnets has the same "magnetic field of influence" that diminishes as we go away from the magnet. Now imagine if there is a piece of iron at a certain location in the room, it will experience a total magnetic field that is the sum of all the magnets. The magnets that are closer to this piece of iron will have a higher influence than the farther ones.

Let $\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ be the set of N magnets (data points) scattered in some high dimensional space. Let $\boldsymbol{x}$ be a new data point (iron) whose density (influence by all the magnets), $P(\boldsymbol{x})$, has to be estimated. In a nonparametric kernel density estimation, we represent this total field of influence as follows:

$$P(\boldsymbol{x}) = \frac{1}{n}\sum_{n=1}^{N} K_\sigma(\boldsymbol{x}, \boldsymbol{x}_n) = \frac{1}{n\sigma\sqrt{2\pi}}\sum_{n=1}^{N} \exp\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}_n\|^2}{2\sigma^2}\right)$$

Here $K_\sigma(\boldsymbol{x}, \boldsymbol{x}_n)$ is the kernel function (chosen to be Gaussian in the above example) that measures the influence of the training data point (magnet) $\boldsymbol{x}_n$ on the test data point (iron) $\boldsymbol{x}$ and σ is the decay with which the field of influence drops (and therefore how wide the field spreads). If σ is too small, then each training point has a very sharp and narrow range field of influence. If σ is too large, then each training point has a very broad field of influence. Like in K-means clustering, K controls complexity, and σ controls the complexity of the density function here. Nonparametric density estimation has the following pros and cons.

- **No prior knowledge**: Nonparametric density estimation does not require that we know the functional form of the density function. This is very handy when there is no domain knowledge about the phenomenon that generates the data. However, we still have to play with σ, the spread of each density function around each training data point. Choosing a small value of σ will model noise in the data and choosing a large value will not capture the signal. There is, like in all hyper-parameter spaces, a sweet spot that we must find through experimentation.
- **Scoring time**: Nonparametric methods are also known as "lazy-learners" since they spend no time "training" a model but at the scoring time their complexity is O(N)—linear in the number of training data points (magnets). This makes them unsuitable for real-time tasks (e.g., if we were to make a real-time decision

about whether a credit card transaction is fraud or not and we are using outlier detection based on density estimations, we cannot use such nonparametric density estimators).

- **Robustness to noise**: Since each training data point has an influence on density estimation of each point, even noisy points get to have their say. It is therefore important to identify and remove the noisy points from the training set or use parametric techniques for highly noisy datasets.

4.2 Parametric Density Estimation

In nonparametric density functions, the data is stored "as is" and is used to compute density using kernel functions. The parametric density estimation functions, on the other hand, first define a parametric form and then find the parameters by optimizing an objective function. We will follow the same four-stage process of intuition, formulation, modification, and optimization to learn parameters.

Intuition: Let $P(\boldsymbol{x}|\,\boldsymbol{\theta})$ be a parametric density function where $\boldsymbol{\theta}$ is the set of parameters to be learnt. For N data points $\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$, we have to find the set of parameters that "fits" the data best. In other words, we need to find the parameters $\boldsymbol{\theta}$ such that the probability of seeing the entire data is maximum.

Formulation: Parametric density function problems are all modeled as optimization problems where we try to find the set of parameters that maximizes the likelihood of seeing the data. Since each data point is identical and independently distributed, the likelihood of seeing the entire data is the product of the likelihood of seeing each data point independently.

$$\theta^* = \arg\max_{\boldsymbol{\theta}} J\,(\boldsymbol{\theta}\,|\mathbf{X}) = \arg\max_{\boldsymbol{\theta}} \prod_{n=1}^{N} P\,(x_n\,|\boldsymbol{\theta}\,)$$

Modification: Typically, when any density functional form (e.g., Gaussian or Poisson or exponential) is substituted for $P(\boldsymbol{x}_n|\,\boldsymbol{\theta})$, the product term becomes too complex to solve. We therefore modify this to the **log likelihood function** which is monotonic and equivalent to maximizing likelihood function, $\theta^* = \arg\max_{\boldsymbol{\theta}} \ln J\,(\boldsymbol{\theta}\,|\mathbf{X}) = \arg\max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \ln P\,(x_n\,|\boldsymbol{\theta}\,)$.

Optimization: Finally, we will optimize this for a few density functions in one-dimensional spaces.

- **Exponential distribution**: where $P(x|\,\theta) = \theta e^{-\theta x}$, for $x > 0$ and 0 otherwise. So

$$J\,(\theta\,|\mathbf{X}) = \sum_{n=1}^{N} \ln\ P\,(x_n\,|\theta\,) = \sum_{n=1}^{N} [\ln\theta - \theta x_n] = N\ln\theta - \theta \sum_{n=1}^{N} x_n$$

$$\frac{\partial J(\theta|\boldsymbol{X})}{\partial \theta} = \frac{N}{\theta} - \sum_{n=1}^{N} x_n = 1 \therefore \widehat{\theta} = \frac{1}{\frac{1}{N}\sum_{n=1}^{N} x_n}$$

- **Bernoulli distribution**: where $P(x|\,\theta) = \theta^x(1-\theta)^{1-x}$, for $x \in \{0,1\}, 0 < \theta < 1$. So

$$J\,(\theta\,|\mathbf{X}) = \sum_{n=1}^{N} \ln P\,(x_n\,|\theta\,) = \sum_{n=1}^{N} [x_n \ln\theta + (1-x_n)\ln(1-\theta)]$$

$$\frac{\partial J\,(\theta\,|\boldsymbol{X})}{\partial\theta} = \frac{1}{\theta} - \sum_{n=1}^{N} x_n - \frac{1}{1-\theta}\left[N - \sum_{n=1}^{N} x_n\right] = 0 \therefore \widehat{\theta} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

- **Poisson distribution**: where $P\,(x|\theta) = \frac{\theta^x}{x!}e^{-\theta}$, $x = 0,1,2,\ldots$ *and* $\theta > 0$. So

$$J\,(\theta\,|\mathbf{X}) = \sum_{n=1}^{N} \ln P\,(x_n\,|\theta\,) = \sum_{n=1}^{N} \Big[x_n \ln\theta - \theta - \ln x_n!$$

$$\frac{\partial J\,(\theta\,|\boldsymbol{X})}{\partial\theta} = \frac{1}{\theta}\sum_{n=1}^{N} x_n - N = 0 \therefore \widehat{\theta} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

- **Normal distribution**: where $P\left(x|\mu,\sigma^2\right) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \theta = \left\{\mu,\sigma^2\right\}$

$$J\left(\mu,\sigma^2\,|\mathbf{X}\right) = \sum_{n=1}^{N} \ln P\,(x_n\,|\theta\,) = -\frac{1}{2}\sum_{n}^{N}\left[\ln\sigma^2 + \frac{x-\mu}{2\sigma^2} + \ln 2\pi\right]$$

$$\frac{\partial J\left(\mu,\sigma^2\,|\boldsymbol{X}\right)}{\partial\mu} = \frac{1}{2\sigma^2}\sum_{n=1}^{N}(x-\mu) = 0 \therefore \mu = \frac{1}{N}\sum_{n=1}^{N} x_n$$

$$\frac{\partial J\left(\mu,\sigma^2\,|\boldsymbol{X}\right)}{\partial\sigma^2} = \frac{1}{2}\sum_{n=1}^{N}\left[\frac{1}{\sigma^2} - \frac{(x-\mu)^2}{\sigma^4}\right] = 0 \therefore \sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x-\mu)^2$$

The reader can refer to Criminisi et al. (2012) and Robert (2014) to learn more about density estimation.

4.3 Mixture of Gaussians

Often, a single Gaussian is not enough to model the complexity of multimodal data. For example, in case of OCR, the same digit might be written in two or three different ways (e.g., a 7 with a cut in the middle or not, a 9 with a curve at the bottom or not), there could be font or other variations. In speech, there could be multiple accent variations within a language (e.g., the English language has different accents, e.g., American, British, Indian, and Australian). In such cases, it is better to learn a multimodal density function using a mixture of unimodal density functions—one for each variant. If each density function is a Gaussian, then this multimodal density function is called a mixture of Gaussians (MoG).

Insight: In MoG we assume that there are $K > 1$ Gaussians that might generate the data. Each of the Gaussians has its own mean, covariance, and prior. So we first pick one of the Gaussians from the mixture with a certain "prior" and then use that Gaussian to generate a data point with a certain probability that diminishes as we go away from the mean of that Gaussian. Another way to think about MoG is that it is a Bayesian extension of K-means clustering. In K-means clustering, one of the problems was that since we were using only Euclidean distances between a data point and its cluster center, all clusters were spherical and of different shapes and densities were not easy to handle. In MoG, we enable this extra degree of freedom that each cluster, that is, Gaussian, can now have an arbitrary covariance matrix to adjust to the "shape" of the cluster. Second, instead of forcing a data point to be in one cluster only, MoG lets a data point to be influenced by more than one Gaussian depending on its distance from the mean of the Gaussian and the shape of the Gaussian depending on the covariance matrix. This is also connected to Parzen Windows as follows. On one extreme if we model the entire data's density with a single Gaussian, we might get a very simple model that might not capture the essence of the data. If on the other extreme, we treat each data point as its own Gaussian like in Parzen window, then we might be overlearning. But a mixture of Gaussians is giving the right number of Gaussians needed to model the data between these two extremes.

Formulation: Let us say there are K Gaussians that we have to model to explain the data $\mathbf{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$. Each Gaussian has its own prior $\pi_k = P(k)$, mean $\boldsymbol{\mu}_k$, and covariance matrix $\boldsymbol{\Sigma}_k$. Let $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K\}$ be the set of parameters where $\boldsymbol{\theta}_k = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$. Like in K-means clustering, we will use a set of latent parameters: $\boldsymbol{\Delta} = [\delta_{n,k}]$ that quantifies the association of the n^{th} data point with the k^{th} Gaussian. This will morph into the softer posterior probability $P(k|\boldsymbol{x}_n)$. The overall maximum likelihood objective is

$$J(\boldsymbol{\Theta}|\mathbf{X}) = \prod_{n=1}^{N}\prod_{k=1}^{K}[P(\boldsymbol{x}_n, k)]^{\delta_{n,k}} = \ln \prod_{n=1}^{N}\prod_{k=1}^{K}[P(\boldsymbol{x}_n, k)]^{\delta_{n,k}}$$

Modification: We apply two modifications to the above data likelihood objective. First, we convert the joint probability of data and mixture into two parts:

$P(\boldsymbol{x}_n, k) = P(\boldsymbol{x}_n|\, k)P(k)$ and take the log of the likelihood to make the calculus easy for optimization. Also note that there are constraints on $\delta_{n,k}$ that for each n they must add up to 1. We put all these into the modified objective function:

$$J\left(\boldsymbol{\Theta}\,|\mathbf{X}\right) = \prod_{n=1}^{N}\left(\prod_{k=1}^{K}\delta_{n,k}\left[\ln P\left(\boldsymbol{x}_n\,|k\right)\right] + \ln P(k) + \lambda\left(\prod_{k=1}^{K}\delta_{n,k} - 1\right)\right)$$

Optimization: Similar to the K-means clustering, mixture of Gaussians also uses an EM approach to solve the two sets of parameters alternately resulting in the following iterative solution:

The expectation step becomes the Bayes theorem:

$$\delta_{n,k}^{(t)} = P^{(t)}\left(k|n\right) \leftarrow \frac{\pi_k^{(t)} P^{(t)}\left(\boldsymbol{x}_n|k\right)}{\sum_{j=1}^{K}\pi_j^{(t)} P_t\left(\boldsymbol{x}_n|j\right)} = \frac{\pi_k^{(t)} P^{(t)}\left(\boldsymbol{x}_n|\boldsymbol{\mu}_k^{(t)}, \boldsymbol{\Sigma}_k^{(t)}\right)}{\sum_{j=1}^{K}\pi_j^{(t)} P_t\left(\boldsymbol{x}_n|\boldsymbol{\mu}_j^{(t)}, \boldsymbol{\Sigma}_j^{(t)}\right)}$$

The maximization step when optimized for mean and covariance results in the following updates:

$$\pi_k^{(t+1)} \leftarrow \frac{1}{N}\sum_{n=1}^{N}\delta_{n,k}^{(t)}$$

$$\boldsymbol{\mu}_k^{(t+1)} \leftarrow \frac{\sum_{n=1}^{N}\delta_{n,k}^{(t)}\boldsymbol{x}_n}{\sum_{n=1}^{N}\delta_{n,k}^{(t)}}$$

$$\sum\nolimits_k^{(t+1)} \leftarrow \frac{\sum_{n=1}^{N}\delta_{n,k}^{(t)}\left(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{(t+1)}\right)\left(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{(t+1)}\right)^T}{\sum_{n=1}^{N}\delta_{n,k}^{(t)}}$$

Gaussian mixture models are used extensively in many domains including speech, outlier detection, and building Bayesian classifiers especially when a class has multiple latent subclasses that need to be discovered automatically. MoG still depends on initialization, and one way to do it is to first do farthest first point sampling to initialize K cluster centers. Then do K-means clustering to converge on clusters and use those cluster centers as the initial means and the covariance matrices of those clusters as the initial covariance. Using this as the seed, we learn MoG and further refine those K-means clusters.

Figure 15.17 shows increasing degrees of complexity of parametric densities for the same dataset. In (a) we use a single spherical density (i.e., variance along all dimensions is assumed to be same) -equal diagonal elements (i.e., we ignore correlation among dimensions). In (b) we still use a single Gaussian but now each dimension can have a different variance while still ignoring correlation among

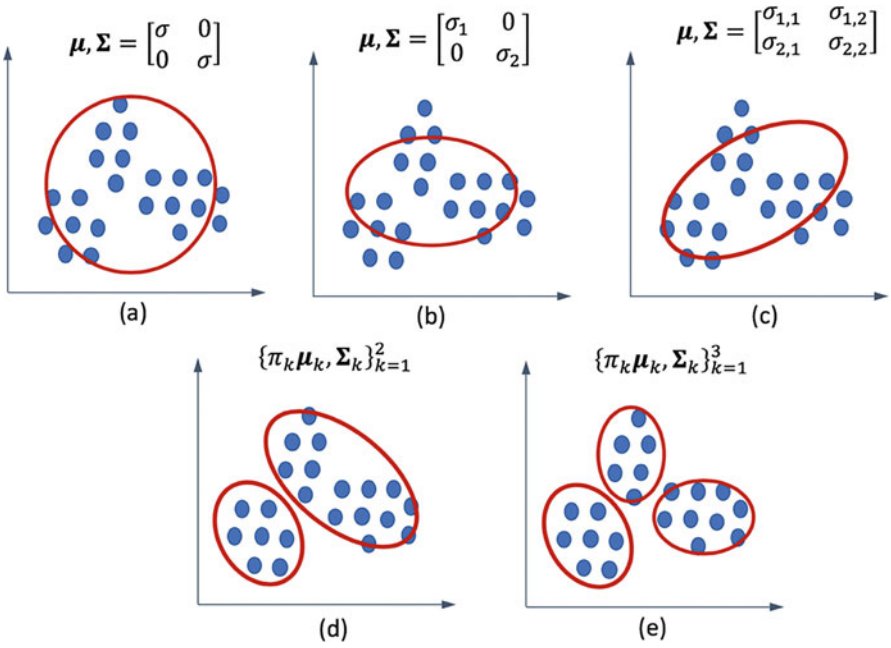

Fig. 15.17 Different complexities of a density function: (**a**) single Gaussian, spherical covariance. (**b**) Single Gaussian, diagonal covariance, (**c**) single Gaussian, full covariance, (**d**) mixture of two full covariance Gaussians, (**e**) mixture of three full covariance Gaussians

dimensions. In (c) we continue to use a single Gaussian to model the density, but we allow a full covariance. In (d) we increase the number of Gaussians to be two as one Gaussian does not seem to be sufficient to model the density of this data. In (e) we finally use three Gaussians to model the density—which seems to be sufficient. Adding more will try to memorize the data and not generalize. See Rasmussen (2004) to read more about mixture of Gaussian.

In this section, we have studied a variety of density estimation paradigms. There are other density estimation frameworks, for example, hidden Markov models for sequence of symbols. Overall, density estimations can give deep insights about "where to look" in the data, which parts of the data matter, and which parts are "surprising" or "anomalous."

5 Frequent Itemset Mining (FISM)

Insofar we have explored a variety of unsupervised learning frameworks that discover different types of structures in the data. We now explore another very common kind of data type—the **itemset data** and see what kind of structure can

be found in such data. The itemset data is best described as a dataset where each record or data point is a "set of items from a large vocabulary of possible items." Let us first consider various domains where such data occurs:

- **Market basket data**: One of the most common examples of itemset data is the market basket data or the ***point of sale (POS)*** data where each basket is comprised of a "set" of products purchased by a customer in either a single visit or multiple visits put together (e.g., all products purchased in a week or one quarter, or the whole lifetime). Here the list of all products sold by the retailer is the vocabulary from which a product in the item could come from. We are losing information by just considering the set of products and not include their quantity or the price which would make it a "weighted set" instead. The problem is that in a typical heterogeneous retail environment, the products are not comparable. For example, 1 l of milk, 1 dozen bananas, and 1 fridge are not comparable to each other either in physical or monetary units. Hence, we stick with just the unweighted sets or ***baskets*** rather than weighted sets or ***bags***.
- **Keyword sets**: Another common example of itemset data is a keyword set. Often entities such as images on Flickr, videos on YouTube, papers in conferences, or even movies in IMDB are associated with a set of keywords. These keywords are used to tag or describe the entity they are referring to. Here the vocabulary from which the keywords can come from is predefined (e.g., keyword lists for conference papers) or is taken to be the union of all the keywords associated with all the entities.
- **Token sets**: Itemset data is also present in many other contexts not as keywords or products but arbitrary tokens. For example, hashtags in tweets whether per tweet or per account is an itemset. Another example is the set of skills in each LinkedIn profile is an itemset data. In a user session on YouTube, all videos watched by a user in one session constitute an itemset as well. All WhatsApp groups are itemsets of phone numbers. In a payment app or a credit card account, the set of merchants where a customer shopped in the last n days is also an itemset. It is up to us how we convert any transaction data into an itemset data as it makes sense.
- **Derived itemsets**: Itemset data can also be derived from other datasets. In neuroscience experiments, for example, we might want to discover which neurons fire together in response to different experiences or memories. In such cases, we can consider a moving (overlapping) window of a certain size and all *neurons* that fire in the same window could be considered an itemset. Similarly, in gene expression experiments, all genes that express themselves from the same stimuli could also be considered an itemset.

In all these itemset datasets we are interested in finding patterns of "co-occurrence"—that is, which subsets of items co-occur in the same itemsets. There are many ways to define co-occurrence. In frequent itemset mining (FISM), we are interested in finding "large and frequent" itemsets. While the dataset is simple and the definition of what is a pattern is also very straightforward, what makes this a complex problem is the combinatorial explosion when the vocabulary of possible

items is very large. One of the key algorithms that we will develop here called the ***apriori algorithm*** solves this problem using a very basic insight from set theory.

Intuition: Consider an itemset data shown in Fig. 15.18i, it has a total of 10 data points over a vocabulary of 6 items {a,b,c,d,e,f}. We will first consider itemsets of size 1. There are six itemsets of size 1 (Fig. 15.18ii). For each we can compute the frequency which is the number of itemsets (out of 10) in which that item was present. Now that we have itemsets of size 1 and their frequencies (also known as *support*), we can compute itemsets of size 2 and so on. Now since we only care about the "frequent" itemsets and not all itemsets, we can define a frequency threshold θ_f (also known as support threshold) such that only those itemsets of size k (= 1 for now) will be kept whose frequency is above this threshold and others will be deemed not "supported" (i.e., noisy). This goes with the underlying philosophy of pattern recognition that anything that is high frequency is a pattern worth remembering. Now from itemsets of size 1 we can find itemsets of size 2 and their support, again pruning off those whose support is less, etc.

Formulation: The only problem with this brute-force counting is the following. In order for us to count the frequency of an itemset of size k, we need to maintain a counter with the itemset as the key and its count as value. As we go through a dataset, we check whether this itemset is a subset of the data itemset or not. If so, we increment its counter. Now as the vocabulary size grows and the value of

Item-set Data

Data set X
{a,b}
{b,c,e}
{a,c,e,f}
{c,d,f}
{a,e,f}
{b,e,f}
{a,b,c,d}
{c,e,f}
{a,c,f}
{a,b,e,f}

(i)

Item-sets of Size 1

F(1)	Freq
{a}	6
{b}	5
{c}	6
{d}	2
{e}	6
{f}	7

(ii)

Candidates of size 2

C(2)
{a,b}
{a,c}
{a,e}
{a,f}
{b,c}
{b,e}
{b,f}
{c,e}
{c,f}
{e,f}

(iii)

Item-sets of size 2

F(2)	Freq
{a,b}	3
{a,c}	3
{a,e}	3
{a,f}	4
{b,c}	2
{b,e}	3
{b,f}	2
{c,e}	3
{c,f}	4
{e,f}	5

(iv)

Candidates of size 3

C(3)
{a,b,e}
{a,c,e}
{a,c,f}
{a,e,f}
{c,e,f}

(v)

Item-sets of size 3

F(3)	Freq
{a,b,e}	1
{a,c,e}	1
{a,c,f}	2
{a,e,f}	3
{c,e,f}	2

(vi)

Fig. 15.18 The apriori algorithm at work. (i) The dataset where each data point is a set of items from a dictionary of six possible items {a,b,c,d,e,f}. (ii) Frequent itemsets of size 1. If support threshold is 3, then all itemsets of size less than 3 are ignored (i.e., {d}). (iii) Using the apriori trick, all candidate itemsets of size 2 created from frequent itemsets of size 1. (iv) A pass over the data gives frequency of each of the candidates. Note that we did not have to worry about any pair of itemsets involving item d because its frequency count is less than threshold (3). (v) Again applying apriori trick to create candidates of size 3. (vi) Final frequent itemset of size 3 or more is {a,e,f} of size 3 and others of size 2

k grows, the potential number of combinations that we might have to keep in the counter memory grows to $\mathrm{O}\binom{N}{k}$. So we apply the famous "apriori trick" here which tames the combinatorial explosion in an intelligent fashion.

Modification: The ***Apriori Trick*** is based on a simple observation that if $f(\mathbf{s}|\,\mathbf{X})$ is the frequency of the itemset $\mathbf{s}$ of size k in a dataset $\mathbf{X}$, then its frequency cannot be greater than the frequency of the ***least*** frequent subset of size $k-1$ of $\mathbf{s}$. In other words, let us say if $\mathbf{s} = \{a,b,c\}$ and let us say its frequency is *3*, then it must be true that the frequency of all of its subsets, that is, {a,b}, {b,c}, and {a,c}, is at least 3. Otherwise, it will not be possible for {a,b,c} to have a frequency of 3. More formally:

$$f\,(\mathbf{s}|\mathbf{X}) \leq \min_{i \in \mathbf{s}} \{f\,(\mathbf{s} \sim i\,|\mathbf{X})\}$$

Where $\mathbf{s} \sim i$ is the set obtained by removing item i from set $\mathbf{s}$. Using this "apriori trick," the frequent itemset is able to ignore many *itemsets from counting as it knows that they will not be frequent anyway.*

Optimization*: The frequent itemset mining algorithm essentially grows itemsets from size k to size k + 1 as follows using a three-step process.*

- ***Candidate Generation Step****:* The input to this step is the frequent itemsets (whose support is above a threshold) of size *k,* $\mathbf{F}_k$. From this frequent itemset we first generate a candidate set of size $k+1$, $\mathbf{C}_{k+1}$ that satisfy the apriori property, that is, we add all itemsets of size $k+1$ to $\mathbf{C}_{k+1}$ whose subsets of size k are present in $\mathbf{F}_k$ (Fig. 15.18iii, v).
- ***Frequency Counting Step***: The $k+1$ size itemsets in the candidate set $\mathbf{C}_{k+1}$ are the only itemsets that have a chance to have a frequency above the support threshold, θ_f. All other combination of itemsets of size $k+1$ are not counted at all. This really reduces the combination of itemsets on which the counter has to run in the next iteration:

$$f\,(\mathbf{s}\,|\mathbf{X}) = \sum_{n=1}^{N} \delta\,(\mathbf{s} \subseteq \boldsymbol{x}_n)\,, \forall \mathbf{s} \in \mathbf{C}_{\mathrm{k}+1}$$

- ***Frequency Pruning Step***: Finally, when a pass through the data has been made and all frequencies of candidate itemsets are counted, the itemsets whose frequency is below the support threshold are removed to obtain $\mathbf{F}_{k+1}$, the final frequent itemsets of size $k+1$.

Figure 15.18 shows the entire process of generating frequent itemsets of size up to 3 from an itemset data with support threshold of 3. Each iteration alternates between the above three steps.

The purpose of creating frequent itemsets is to find rules of the sort: (If *condition* then *trigger*) with some confidence. For example, once we have discovered through

the above process that {a,e,f} is a frequent itemset, we can now create rules of the form: {a,e} ➔{f}, {a,f} ➔ {e}, {e,f} ➔ {a}, {a} ➔ {e,f}, {e} ➔ {a,f}, {f} ➔ {a,e}. Each rule comes with a confidence score computed based on the frequency of the entire set {a,e,f} and the frequency of the condition set, that is,

$$Confidence\left(\{a,e\}\rightarrow\{f\}\right) = Support\left(\{a,e,f\}\right)/Support\left(\{a,e\}\right)$$

In other words, this says that if someone bought both a and e, then the probability that they will also buy f is 1 and can therefore be recommended with a very high confidence. In frequent itemset mining, all such rules are created and a confidence threshold θ_c is used to prune out rules with lower confidence. The output of the frequent itemset algorithm is the set of such rules with high support and confidence.

Frequent itemset mining has been one of the early algorithms that almost gave birth to the field of "data mining." It was the first breakthrough of its kind in mining such itemset data and since then, there have been a number of improvements in smart data structures to store the candidate and frequent itemsets to make it faster and more scalable. It has also been applied to areas beyond retail data mining for which it was originally invented. It has been used to discover "higher order features" of type "sets of items" in various domains including computer vision where each image region could be thought of as a collection of symbols from a vocabulary (HoG or SIFT). If many regions across many images show the same set of items (e.g., face images all show eye, nose, mouth, etc.), then a new object (face) can be created from a set of lower order features. Wherever we have a "set of items" dataset, we can use FISM.

See Chap. 6 (Sect. 6.2) in Han et al. (2011) for additional material on frequent itemset mining.

6 Network Analysis

Now that we understand how to find patterns in sets and multivariate data, we turn our attention to an even more complex yet commonly available data type—a ***graph*** or ***network*** data. These graphs may be weighted (i.e., edges have weights) or unweighted (i.e., edges are binary—either present or not), directed (i.e., edges either go from a node to another) or undirected (i.e., there is no direction on edges), or homogeneous (i.e., all nodes and edges are of same types) or heterogeneous (i.e., nodes or edges are of different types). Analyzing graphs for patterns presents very interesting challenges and a lot of opportunities in a wide variety of applications. There are a number of different kinds of patterns that can be discovered in graphs. In this section, we will focus on two kinds of network analyses problems: (1) PageRank, one of the most important algorithms in graph theory that led to the birth of companies like Google, (2) Detecting Cliques in graphs—another commonly used algorithm with many applications.

Graphs or networks are present in many domains. Internet, for example, is a collection of a very large number of web pages (generated at the rate of more than

1000 pages per minute) with links going from one page to another (directed graph). This is perhaps one of the largest graph out there. Social networks are another class of large graphs—LinkedIn, Facebook, telecom networks (e.g., people calling each other above a threshold), financial networks (e.g., based on money transfers), etc.

Weighted graphs can also be created from transaction or co-occurrence data. For example, consider a market basket data where we can quantify the consistency with which two products (a, b) are co-purchased together $P(a, b)$ more often than random $P(a)P(b)$ using, for example, pointwise mutual information.

$$\phi\,(a, b) = \log \frac{P\,(a, b)}{P(a)\,P(b)}$$

Any co-occurrence data can be converted to such weighted graphs where the edges can be removed if these weights are below a threshold. Many measures such as Jaccard coefficient, normalized pointwise mutual information, and cosine similarity can be used to create these weighted graphs. Next we develop two algorithms for network analysis.

6.1 Random Walks (PageRank)

Given a directed graph like the Internet, we are interested in finding out which is the most important node in the graph. The key motivation behind this problem came from Google where they wanted to sort all pages that contained a keyword in an order that "made sense." They posed this as a "random surfer" problem—if a surfer randomly picks a page on the Internet and starts following the links, what would be the probability that he will be at a certain page, and if we average over all such random surfers, which page on the Internet would have the most number of people, the second most number of people, and so on. This distribution over pages in the steady state gives the PageRank of each page on the Internet.

Intuition: Every page on the Internet has a set of incoming edges (shown for node j in Fig. 15.19) and a set of outgoing edges (shown for node i in

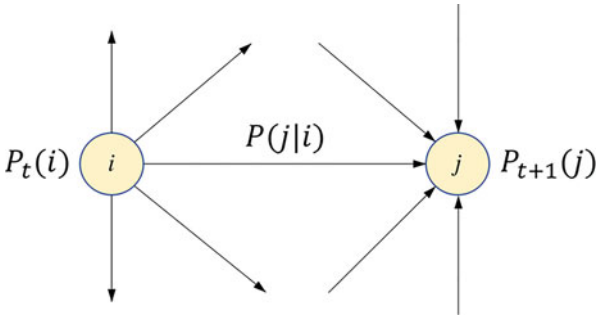

Fig. 15.19 Outgoing edges of node i and **Incoming** edges of node j. The probability of being on node j at time t + 1 depends on the probability of being on node i at time t and making a transition from i to j

Fig. 15.19). When on any page (node i), a random surfer might have some (e.g., equal) probability of going to one of the outgoing edges from this page. Thus the probability of the random surfer to "reach" a page (node j) would be to first "be" at one of the incoming pages (e.g., node i) of this page with a certain probability and then reach this page (node j) with a certain transition probability from that page (node i) in the next iteration as shown in Fig. 15.19.

Formulation: We now formulate this PageRank problem. Let us assume that there are N pages on the Internet $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}$. Let $\mathbf{I}(x_n)$ be the set of in-neighbors of x_n and let $\mathbf{O}(x_n)$ be the set of out-neighbors of x_n. The Link structure is characterized by the transition probabilities: $\mathbf{P} = [P(x_j|\, x_i)]$, $\forall\, x_j \in \mathbf{O}(x_i)$. This could be either an equal probability or a weighted probability depending on the nature of the links going from x_i to x_j. For example, this transition probability will depend on whether there are lots of prominent links or a few footnote links going from x_i to x_j vs. x_i to x_k, some other page going out of x_i. Let us assume that there is a prior probability that a user might "start" or "randomly go to" a particular page. This prior depends on, for example, how many people have this page as their home page or how often is this page typed directly in the browser compared to the other pages. Let $Q(x_i)$ be this initial probability of going to this page. Let us say this random jumping to this page happens with a probability $(1 - \lambda)$ and with a probability λ the surfer actually systematically follows the links (browser behavior). Now at any given iteration t, we can compute the probability that a random surfer will be at a certain page:

$$P_{t+1}\left(x_j\right) \leftarrow (1-\lambda)\, Q\left(x_j\right) + \lambda \sum_{i=1}^{N} P_t\left(x_i\right) P\left(x_j\,|x_i\right) = \frac{1-\lambda}{N} + \lambda \sum_{x_i \in \mathbf{I}(x_j)} \frac{P_t(x_i)}{|\mathbf{O}(x_i)|}$$

In the above we made an assumption that $Q(x_j)$ are all equal to 1/N and outgoing probabilities $P(x_j|\, x_i)$ are all equal to $1/|\mathbf{O}(x_i)|$. Once converged, this gives the most "central" or important pages based on the link structure of the graph. Such an analysis can be done not just on the Internet graph but any directed graph. For example, if we have a gene expression graph that suggests which gene affects which other genes, we can find the most important genes in the network. Similarly, if we have an influencer–follower graph on a social network, we can find the most influential people in the social network and so on.

6.2 Maximal Cliques

PageRank was an example of a global network analysis algorithm. Cliques are an important class of patterns that are sought in graphs in many domains. Consider for example, in retail, a product graph of which products "go with" which other products. A clique in such a graph would indicate a "product bundle" that characterizes the latent intent of a user. Similarly a set of keywords that are all connected to each other might indicate a coherent concept as shown in Fig. 15.20.

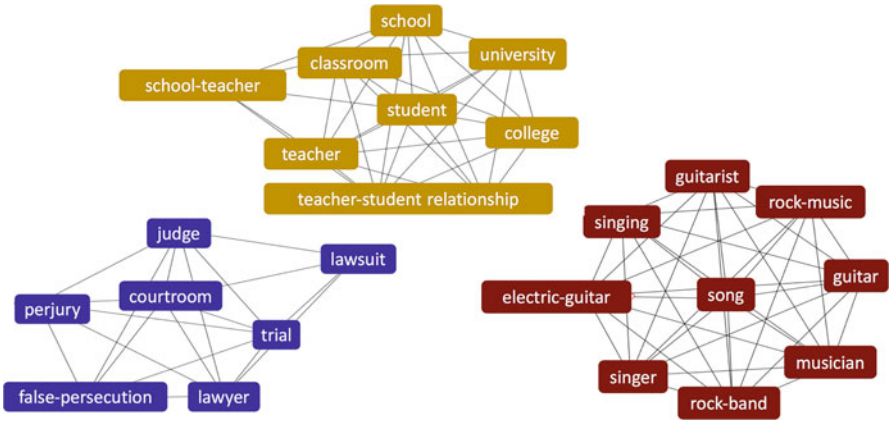

Fig. 15.20 A set of cliques found in keyword–keyword co-occurrence graph created from IMDB dataset

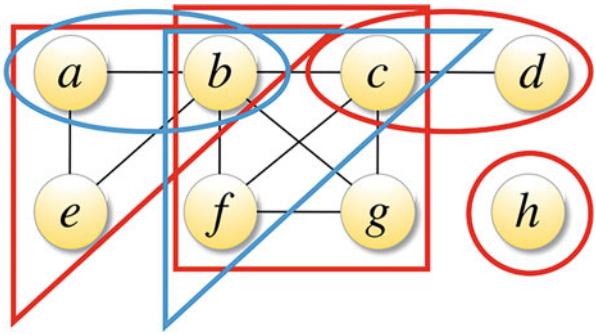

Fig. 15.21 A graph with eight nodes and ten edges. Sub-graphs marked in blue ({a,b}, {b,c,f}) are cliques but not maximal cliques. This graph has four maximal cliques: {h}, {c,d}, {b,c,f,g}, and {a,b,e} marked in red

Here we first create a graph between all pairs of keywords based on how often they co-occur more than random. This graph is then binarized by applying a threshold and then cliques are sought in this graph.

A "**Clique**" is a fully connected subgraph of a binary graph. A "**Maximal Clique**" is a clique that is not a sub-graph of any other clique. Figure 15.21 shows a graph with eight nodes and ten edges. It has four maximal cliques. Finding all maximal cliques in a graph is an NP-hard problem with a known complexity of $O\left(3^{\frac{n}{3}}\right)$ for a graph with n nodes. In this section, we will present a MapReduce algorithm for finding all maximal cliques of a binary graph. Finding such maximal cliques in graphs could help improve our understanding of the graph, find actionable insights in the graph, and even discover higher order structures beyond nodes and edges (e.g., product bundles or communities).

In order to develop the MapReduce algorithm for finding all maximal cliques, we will first introduce a few concepts:

- ***Neighborhood of a clique***: For any known clique in the graph (e.g., {b,c}) we define its neighbor as the set of nodes that are connected to *all* nodes in the clique. Here since node f and node g are connected to ***both*** b and c, they form the neighborhood of the clique {b,c}. In other words:

$$N(\{b, c\}) = \{f, g\}$$

$$N(\{b, c, f\}) = \{g\}$$

$$N(\{a\}) = \{b, e\}$$

$$N(\{a, b\}) = \{e\}$$

- ***Neighborhood of a maximal clique***: Note that neighborhood of a maximal clique by this definition is an empty set:

$$N(\{a, b, e\}) = \varnothing$$

$$N(\{b, c, f, g\}) = \varnothing$$

$$N(\{c, d\}) = \varnothing$$

$$N(\{h\}) = \varnothing$$

- ***Clique map***: We define a map between a clique (key) and its neighborhood (value) as a Clique Map. This is the main data structure that will be used by the MapReduce algorithm to find all maximal cliques iteratively.

Iterative MapReduce for Finding Maximal Cliques

The key to a MapReduce algorithm is the way we represent each record, what we do with it during the Map step, and how we define the Reduce step. Finding maximal cliques in a graph is accomplished by running a MapReduce algorithm that

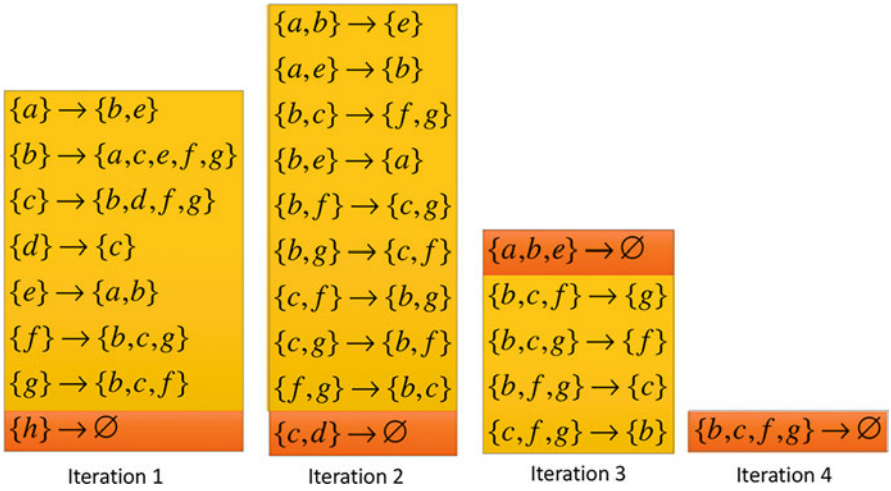

Fig. 15.22 Four MapReduce iterations needed to find all maximal cliques of different sizes for the graph shown in Fig. 15.20. Iteration 1 is the input to the algorithm—it comprises of all cliques of size 1 and their clique neighbors. Iteration 2 is the set of all cliques of size 2 (edges) and their neighbors, iteration 3 is the set of all cliques of size 3 and their neighbors, and so on. In each iteration, a clique whose clique neighbor is empty is deemed a maximal clique and stored

starts with cliques of size 1, that is, each node is a clique. This is stored along with its adjacency list or clique neighbor (forming a clique map shown in Fig. 15.21, iteration 1). Figure 15.22 shows the four iterations of the algorithm where each iteration is the same MapReduce step where we go from clique maps of size k cliques to clique maps of size $k + 1$ cliques. The crux of this algorithm is now the Map and the Reduce steps that will take us from one iteration to the next.

The Map Step

In each iteration of the algorithm, we are given a clique map with a clique and its neighborhood. We want to grow the clique by adding one neighbor at a time to the original clique. We make the following observation about a clique map (e.g., $N(\{b, c\}) = \{f, g\}$): If one element (say f) is removed from the clique neighbor and added to the clique itself ($\{b, c\}$), the resulting set ($\{b, c, f\}$) will also be a clique. This is true because we know that by definition f is connected to both b and c and $\{b, c\}$ is already a clique so $\{b, c, f\}$ will also be a clique. However, also note that we cannot guarantee that what remains on the neighborhood side (i.e., $\{g\}$) is still a neighbor of the new clique ($\{b, c, f\}$) because for that we need to guarantee that g is connected to f, information that is not availale to this mapper.

The Reduce Steps

The output of the mapper is an intermediate key value obtained from clique maps. While the keys of these maps are guaranteed to be cliques of size $K + 1$, the values are not guaranteed to be the neighbors of the corresponding cliques. In order to obtain the clique map of size $K + 1$, the reducer must take an intersection of all the

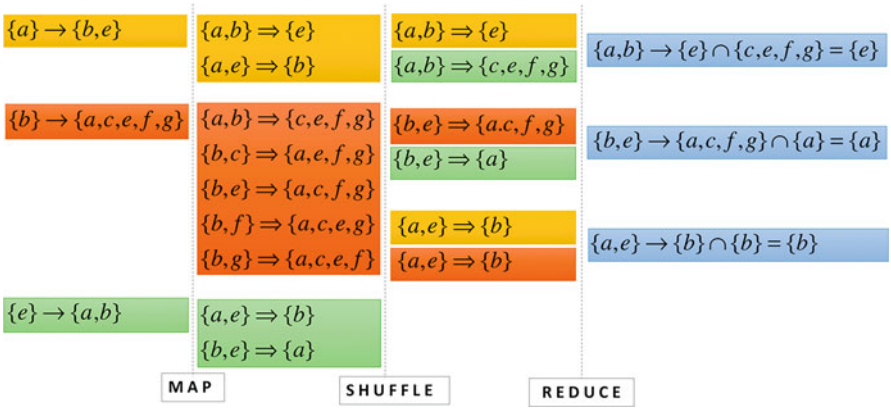

Fig. 15.23 Each MapReduce iteration for finding all maximal cliques in an unweighted graph. The Map step takes clique maps of size K, generates all possible cliques of size $K + 1$ by moving one element at a time to the clique side from the neighborhood side. The Reduce step then takes the intersection of the remaining neighbors for the same clique of size K + 1 resulting in clique maps of size K + 1

sets of the same clique. Figure 15.23 shows the entire process from Map to Shuffle to Reduce that takes us from clique maps of size 1 to clique maps of size 2. Repeating this process in each iteration results in cliques of various sizes.

While we explored only two broad ideas—one macros and one micro—in network analysis, there are a large number of algorithms especially around community detection where softer variants of cliques—communities are discovered within the networks. Analysis of networks can find interesting structures like fraud syndicates in telecommunication or service networks and financial networks. Link Prediction, another important area in network analysis, is used by LinkedIn and other social networks to suggest more connections to any individual based on their neighborhood structure and so on. Handcock et al. (2007) can be a helpful resource to learn further.

7 Conclusion

In this chapter, we explored a variety of unsupervised learning paradigms—projection, clustering, density estimation, frequent itemset mining, and network analysis. These paradigms are typically used to understand different types of data (multivariate, sets, similarity matrices, graphs). This is an essential first step before we start to build supervised learning models from such data. These algorithms help us visualize the data, remove redundancy in the form of feature correlations, find groups of similar items to quantize the data into "representative" clusters, find objects at higher order of abstractions, and in general help reverse engineer the process that might have generated the data in the first place. In general, unsupervised learning is like "reading the book of data" to get a general lay of the land, a broad understanding of the data, without a particular question being asked of this data.

Supervised learning, on the other hand, starts with a question and forces us to read the book but only with respect to the question. In general, it is always better to explore the data using these unsupervised learning approaches before building supervised learning models on it. The insights derived from these algorithms can be used as is to draw conclusions about the data, make decisions, or serve as features for the next stages of modeling.

Electronic Supplementary Material

More examples, corresponding code, and exercises for the chapter are given in the online appendices to the chapter. All the datasets, code, and other material referred in this section are available in www.allaboutanalytics.net.

References

Criminisi, A., Shotton, J., & Konukoglu, E. (2012). Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends® in Computer Graphics and Vision, 7*(2–3), 81–227.

Friedman, J., Hastie, T., & Tibshirani, R. (2001). *The elements of statistical learning (Vol. 1, No. 10). Springer series in statistics*. New York, NY: Springer.

Han, J., Pei, J., & Kamber, M. (2011). *Data mining: Concepts and techniques*. Amsterdam: Elsevier.

Handcock, M. S., Raftery, A. E., & Tantrum, J. M. (2007). Model-based clustering for social networks. *Journal of the Royal Statistical Society: Series A (Statistics in Society), 170*(2), 301–354.

Murphy, K. (2012). *Machine learning – A probabilistic perspective*. Cambridge, MA: The MIT Press.

Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.). (2013). *Machine learning: An artificial intelligence approach*. Berlin: Springer Science & Business Media.

Rasmussen, C. E. (2004). Gaussian processes in machine learning. In O. Bousquet, U. von Luxburg, & G. Rätsch (Eds.), *Advanced lectures on machine learning. ML 2003. Lecture notes in computer science* (Vol. 3176). Berlin: Springer.

Robert, C. (2014). Machine learning, a probabilistic perspective. *Chance, 27*(2), 62–63.