

It should be noted, however, that even when one has an apparently massive data set, the effective number of data points for certain cases of interest might be quite small. In fact, data across a variety of domains exhibits a property known as the **long tail**, which means that a few things (e.g., words) are very common, but most things are quite rare (see Section 2.4.7 for details). For example, 20% of Google searches each day have never been seen before⁴. This means that the core statistical issues that we discuss in this book, concerning generalizing from relatively small samples sizes, are still very relevant even in the big data era.

1.1.1 Types of machine learning

Machine learning is usually divided into two main types. In the **predictive** or **supervised learning** approach, the goal is to learn a mapping from inputs \mathbf{x} to outputs y , given a labeled set of input-output pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$. Here \mathcal{D} is called the **training set**, and N is the number of training examples.

In the simplest setting, each training input \mathbf{x}_i is a D -dimensional vector of numbers, representing, say, the height and weight of a person. These are called **features**, **attributes** or **covariates**. In general, however, \mathbf{x}_i could be a complex structured object, such as an image, a sentence, an email message, a time series, a molecular shape, a graph, etc.

Similarly the form of the output or **response variable** can in principle be anything, but most methods assume that y_i is a **categorical** or **nominal** variable from some finite set, $y_i \in \{1, \dots, C\}$ (such as male or female), or that y_i is a real-valued scalar (such as income level). When y_i is categorical, the problem is known as **classification** or **pattern recognition**, and when y_i is real-valued, the problem is known as **regression**. Another variant, known as **ordinal regression**, occurs where label space \mathcal{Y} has some natural ordering, such as grades A–F.

The second main type of machine learning is the **descriptive** or **unsupervised learning** approach. Here we are only given inputs, $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^N$, and the goal is to find “interesting patterns” in the data. This is sometimes called **knowledge discovery**. This is a much less well-defined problem, since we are not told what kinds of patterns to look for, and there is no obvious error metric to use (unlike supervised learning, where we can compare our prediction of y for a given \mathbf{x} to the observed value).

There is a third type of machine learning, known as **reinforcement learning**, which is somewhat less commonly used. This is useful for learning how to act or behave when given occasional reward or punishment signals. (For example, consider how a baby learns to walk.) Unfortunately, RL is beyond the scope of this book, although we do discuss decision theory in Section 5.7, which is the basis of RL. See e.g., (Kaelbling et al. 1996; Sutton and Barto 1998; Russell and Norvig 2010; Szepesvari 2010; Wiering and van Otterlo 2012) for more information on RL.

4.

<http://certifiedknowledge.org/blog/are-search-queries-becoming-even-more-unique-statistics-from-google>.

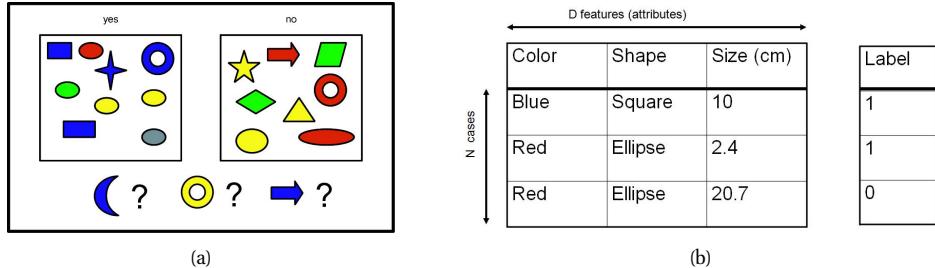


Figure 1.1 Left: Some labeled training examples of colored shapes, along with 3 unlabeled test cases. Right: Representing the training data as an $N \times D$ design matrix. Row i represents the feature vector \mathbf{x}_i . The last column is the label, $y_i \in \{0, 1\}$. Based on a figure by Leslie Kaelbling.

1.2 Supervised learning

We begin our investigation of machine learning by discussing supervised learning, which is the form of ML most widely used in practice.

1.2.1 Classification

In this section, we discuss classification. Here the goal is to learn a mapping from inputs \mathbf{x} to outputs y , where $y \in \{1, \dots, C\}$, with C being the number of classes. If $C = 2$, this is called **binary classification** (in which case we often assume $y \in \{0, 1\}$); if $C > 2$, this is called **multiclass classification**. If the class labels are not mutually exclusive (e.g., somebody may be classified as tall and strong), we call it **multi-label classification**, but this is best viewed as predicting multiple related binary class labels (a so-called **multiple output model**). When we use the term “classification”, we will mean multiclass classification with a single output, unless we state otherwise.

One way to formalize the problem is as **function approximation**. We assume $y = f(\mathbf{x})$ for some unknown function f , and the goal of learning is to estimate the function f given a labeled training set, and then to make predictions using $\hat{y} = \hat{f}(\mathbf{x})$. (We use the hat symbol to denote an estimate.) Our main goal is to make predictions on novel inputs, meaning ones that we have not seen before (this is called **generalization**), since predicting the response on the training set is easy (we can just look up the answer).

1.2.1.1 Example

As a simple toy example of classification, consider the problem illustrated in Figure 1.1(a). We have two classes of object which correspond to labels 0 and 1. The inputs are colored shapes. These have been described by a set of D features or attributes, which are stored in an $N \times D$ design matrix \mathbf{X} , shown in Figure 1.1(b). The input features \mathbf{x} can be discrete, continuous or a combination of the two. In addition to the inputs, we have a vector of training labels \mathbf{y} .

In Figure 1.1, the test cases are a blue crescent, a yellow circle and a blue arrow. None of these have been seen before. Thus we are required to **generalize** beyond the training set. A

reasonable guess is that blue crescent should be $y = 1$, since all blue shapes are labeled 1 in the training set. The yellow circle is harder to classify, since some yellow things are labeled $y = 1$ and some are labeled $y = 0$, and some circles are labeled $y = 1$ and some $y = 0$. Consequently it is not clear what the right label should be in the case of the yellow circle. Similarly, the correct label for the blue arrow is unclear.

1.2.1.2 The need for probabilistic predictions

To handle ambiguous cases, such as the yellow circle above, it is desirable to return a probability. The reader is assumed to already have some familiarity with basic concepts in probability. If not, please consult Chapter 2 for a refresher, if necessary.

We will denote the probability distribution over possible labels, given the input vector \mathbf{x} and training set \mathcal{D} by $p(y|\mathbf{x}, \mathcal{D})$. In general, this represents a vector of length C . (If there are just two classes, it is sufficient to return the single number $p(y = 1|\mathbf{x}, \mathcal{D})$, since $p(y = 1|\mathbf{x}, \mathcal{D}) + p(y = 0|\mathbf{x}, \mathcal{D}) = 1$.) In our notation, we make explicit that the probability is conditional on the test input \mathbf{x} , as well as the training set \mathcal{D} , by putting these terms on the right hand side of the conditioning bar $|$. We are also implicitly conditioning on the form of model that we use to make predictions. When choosing between different models, we will make this assumption explicit by writing $p(y|\mathbf{x}, \mathcal{D}, M)$, where M denotes the model. However, if the model is clear from context, we will drop M from our notation for brevity.

Given a probabilistic output, we can always compute our “best guess” as to the “true label” using

$$\hat{y} = \hat{f}(\mathbf{x}) = \operatorname{argmax}_{c=1}^C p(y = c|\mathbf{x}, \mathcal{D}) \quad (1.1)$$

This corresponds to the most probable class label, and is called the **mode** of the distribution $p(y|\mathbf{x}, \mathcal{D})$; it is also known as a **MAP estimate** (MAP stands for **maximum a posteriori**). Using the most probable label makes intuitive sense, but we will give a more formal justification for this procedure in Section 5.7.

Now consider a case such as the yellow circle, where $p(\hat{y}|\mathbf{x}, \mathcal{D})$ is far from 1.0. In such a case we are not very confident of our answer, so it might be better to say “I don’t know” instead of returning an answer that we don’t really trust. This is particularly important in domains such as medicine and finance where we may be risk averse, as we explain in Section 5.7. Another application where it is important to assess risk is when playing TV game shows, such as Jeopardy. In this game, contestants have to solve various word puzzles and answer a variety of trivia questions, but if they answer incorrectly, they lose money. In 2011, IBM unveiled a computer system called Watson which beat the top human Jeopardy champion. Watson uses a variety of interesting techniques (Ferrucci et al. 2010), but the most pertinent one for our present purposes is that it contains a module that estimates how confident it is of its answer. The system only chooses to “buzz in” its answer if sufficiently confident it is correct. Similarly, Google has a system known as SmartASS (ad selection system) that predicts the probability you will click on an ad based on your search history and other user and ad-specific features (Metz 2010). This probability is known as the **click-through rate** or **CTR**, and can be used to maximize expected profit. We will discuss some of the basic principles behind systems such as SmartASS later in this book.

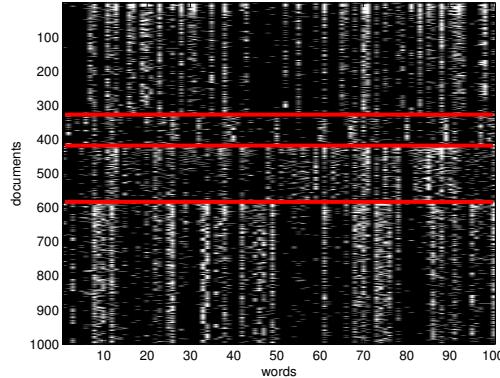


Figure 1.2 Subset of size 16242×100 of the 20-newsgroups data. We only show 1000 rows, for clarity. Each row is a document (represented as a bag-of-words bit vector), each column is a word. The red lines separate the 4 classes, which are (in descending order) comp, rec, sci, talk (these are the titles of USENET groups). We can see that there are subsets of words whose presence or absence is indicative of the class. The data is available from <http://cs.nyu.edu/~roweis/data.html>. Figure generated by `newsgroupsVisualize`.

1.2.1.3 Real-world applications

Classification is probably the most widely used form of machine learning, and has been used to solve many interesting and often difficult real-world problems. We have already mentioned some important applications. We give a few more examples below.

Document classification and email spam filtering

In **document classification**, the goal is to classify a document, such as a web page or email message, into one of C classes, that is, to compute $p(y = c|\mathbf{x}, \mathcal{D})$, where \mathbf{x} is some representation of the text. A special case of this is **email spam filtering**, where the classes are spam $y = 1$ or ham $y = 0$.

Most classifiers assume that the input vector \mathbf{x} has a fixed size. A common way to represent variable-length documents in feature-vector format is to use a **bag of words** representation. This is explained in detail in Section 3.4.4.1, but the basic idea is to define $x_{ij} = 1$ iff word j occurs in document i . If we apply this transformation to every document in our data set, we get a binary document \times word co-occurrence matrix: see Figure 1.2 for an example. Essentially the document classification problem has been reduced to one that looks for subtle changes in the pattern of bits. For example, we may notice that most spam messages have a high probability of containing the words “buy”, “cheap”, “viagra”, etc. In Exercise 8.1 and Exercise 8.2, you will get hands-on experience applying various classification techniques to the spam filtering problem.

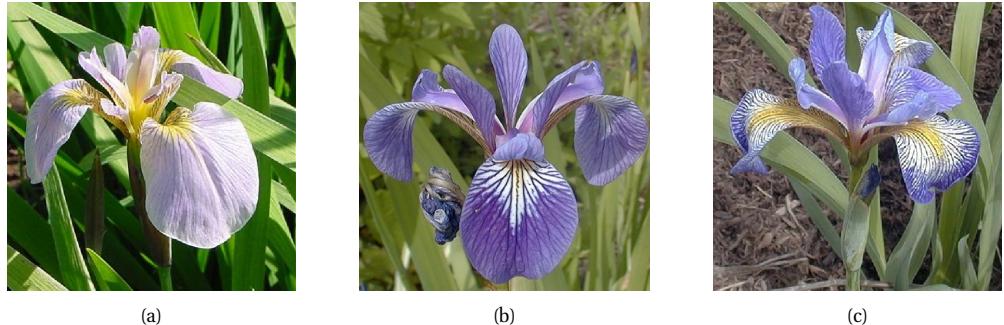


Figure 1.3 Three types of iris flowers: setosa, versicolor and virginica. Source: <http://www.statlab.uni-heidelberg.de/data/iris/>. Used with kind permission of Dennis Kramb and SIGNA.

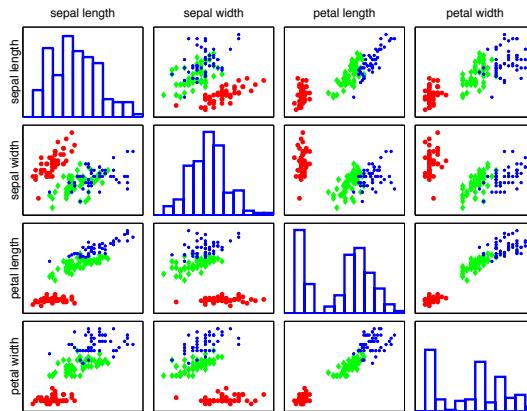


Figure 1.4 Visualization of the Iris data as a pairwise scatter plot. The diagonal plots the marginal histograms of the 4 features. The off diagonals contain scatterplots of all possible pairs of features. Red circle = setosa, green diamond = versicolor, blue star = virginica. Figure generated by `fisheririsDemo`.

Classifying flowers

Figure 1.3 gives another example of classification, due to the statistician Ronald Fisher. The goal is to learn to distinguish three different kinds of iris flower, called setosa, versicolor and virginica. Fortunately, rather than working directly with images, a botanist has already extracted 4 useful features or characteristics: sepal length and width, and petal length and width. (Such **feature extraction** is an important, but difficult, task. Most machine learning methods use features chosen by some human. Later we will discuss some methods that can learn good features from the data.) If we make a **scatter plot** of the iris data, as in Figure 1.4, we see that it is easy to distinguish setosas (red circles) from the other two classes by just checking if their petal length

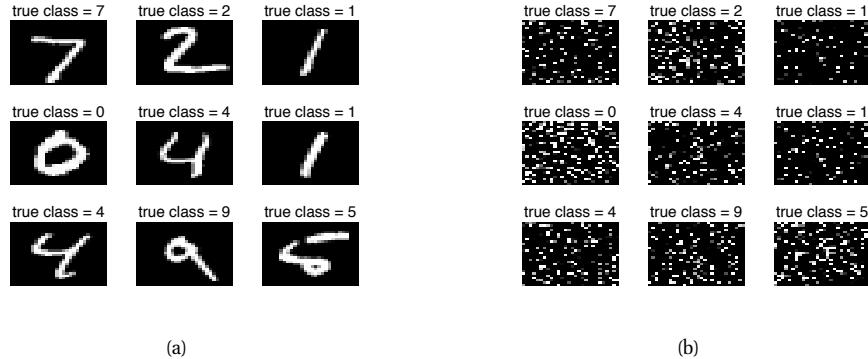


Figure 1.5 (a) First 9 test MNIST gray-scale images. (b) Same as (a), but with the features permuted randomly. Classification performance is identical on both versions of the data (assuming the training data is permuted in an identical way). Figure generated by `shuffledDigitsDemo`.

or width is below some threshold. However, distinguishing *versicolor* from *virginica* is slightly harder; any decision will need to be based on at least two features. (It is always a good idea to perform **exploratory data analysis**, such as plotting the data, before applying a machine learning method.)

Image classification and handwriting recognition

Now consider the harder problem of classifying images directly, where a human has not pre-processed the data. We might want to classify the image as a whole, e.g., is it an indoors or outdoors scene? is it a horizontal or vertical photo? does it contain a dog or not? This is called **image classification**.

In the special case that the images consist of isolated handwritten letters and digits, for example, in a postal or ZIP code on a letter, we can use classification to perform **handwriting recognition**. A standard dataset used in this area is known as **MNIST**, which stands for “Modified National Institute of Standards”⁵. (The term “modified” is used because the images have been preprocessed to ensure the digits are mostly in the center of the image.) This dataset contains 60,000 training images and 10,000 test images of the digits 0 to 9, as written by various people. The images are size 28×28 and have grayscale values in the range 0 : 255. See Figure 1.5(a) for some example images.

Many generic classification methods ignore any structure in the input features, such as spatial layout. Consequently, they can also just as easily handle data that looks like Figure 1.5(b), which is the same data except we have randomly permuted the order of all the features. (You will verify this in Exercise 1.1.) This flexibility is both a blessing (since the methods are general purpose) and a curse (since the methods ignore an obviously useful source of information). We will discuss methods for exploiting structure in the input features later in the book.

⁵. Available from <http://yann.lecun.com/exdb/mnist/>.

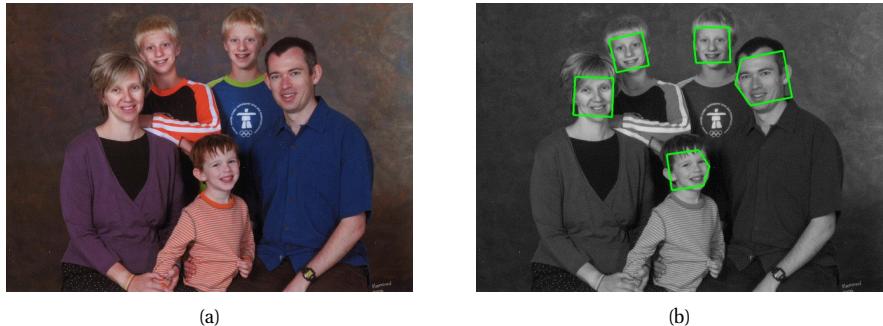


Figure 1.6 Example of face detection. (a) Input image (Murphy family, photo taken 5 August 2010). Used with kind permission of Bernard Diedrich of Sherwood Studios. (b) Output of classifier, which detected 5 faces at different poses. This was produced using the online demo at <http://demo.pittpatt.com/>. The classifier was trained on 1000s of manually labeled images of faces and non-faces, and then was applied to a dense set of overlapping patches in the test image. Only the patches whose probability of containing a face was sufficiently high were returned. Used with kind permission of Pittpatt.com

Face detection and recognition

A harder problem is to find objects within an image; this is called **object detection** or **object localization**. An important special case of this is **face detection**. One approach to this problem is to divide the image into many small overlapping patches at different locations, scales and orientations, and to classify each such patch based on whether it contains face-like texture or not. This is called a **sliding window detector**. The system then returns those locations where the probability of face is sufficiently high. See Figure 1.6 for an example. Such face detection systems are built-in to most modern digital cameras; the locations of the detected faces are used to determine the center of the auto-focus. Another application is automatically blurring out faces in Google's StreetView system.

Having found the faces, one can then proceed to perform **face recognition**, which means estimating the identity of the person (see Figure 1.10(a)). In this case, the number of class labels might be very large. Also, the features one should use are likely to be different than in the face detection problem: for recognition, subtle differences between faces such as hairstyle may be important for determining identity, but for detection, it is important to be **invariant** to such details, and to just focus on the differences between faces and non-faces. For more information about visual object detection, see e.g., (Szeliski 2010).

1.2.2 Regression

Regression is just like classification except the response variable is continuous. Figure 1.7 shows a simple example: we have a single real-valued input $x_i \in \mathbb{R}$, and a single real-valued response $y_i \in \mathbb{R}$. We consider fitting two models to the data: a straight line and a quadratic function. (We explain how to fit such models below.) Various extensions of this basic problem can arise, such as having high-dimensional inputs, outliers, non-smooth responses, etc. We will discuss ways to handle such problems later in the book.

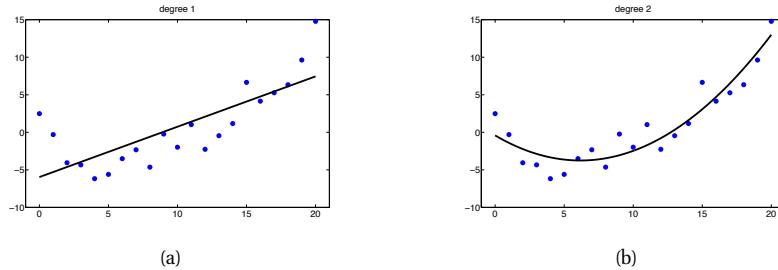


Figure 1.7 (a) Linear regression on some 1d data. (b) Same data with polynomial regression (degree 2). Figure generated by `linregPolyVsDegree`.

Here are some examples of real-world regression problems.

- Predict tomorrow's stock market price given current market conditions and other possible side information.
- Predict the age of a viewer watching a given video on YouTube.
- Predict the location in 3d space of a robot arm end effector, given control signals (torques) sent to its various motors.
- Predict the amount of prostate specific antigen (PSA) in the body as a function of a number of different clinical measurements.
- Predict the temperature at any location inside a building using weather data, time, door sensors, etc.

1.3 Unsupervised learning

We now consider **unsupervised learning**, where we are just given output data, without any inputs. The goal is to discover “interesting structure” in the data; this is sometimes called **knowledge discovery**. Unlike supervised learning, we are not told what the desired output is for each input. Instead, we will formalize our task as one of **density estimation**, that is, we want to build models of the form $p(\mathbf{x}_i|\boldsymbol{\theta})$. There are two differences from the supervised case. First, we have written $p(\mathbf{x}_i|\boldsymbol{\theta})$ instead of $p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$; that is, supervised learning is conditional density estimation, whereas unsupervised learning is unconditional density estimation. Second, \mathbf{x}_i is a vector of features, so we need to create multivariate probability models. By contrast, in supervised learning, y_i is usually just a single variable that we are trying to predict. This means that for most supervised learning problems, we can use univariate probability models (with input-dependent parameters), which significantly simplifies the problem. (We will discuss multi-output classification in Chapter 19, where we will see that it also involves multivariate probability models.)

Unsupervised learning is arguably more typical of human and animal learning. It is also more widely applicable than supervised learning, since it does not require a human expert to

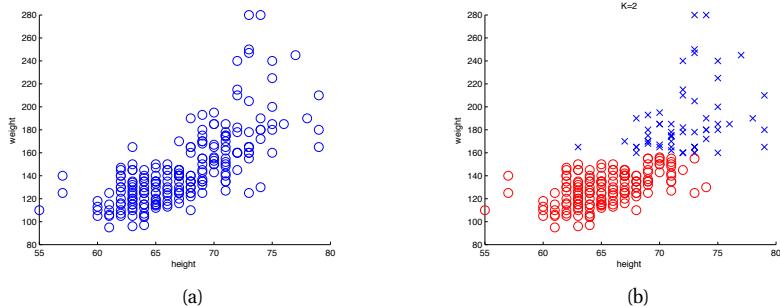


Figure 1.8 (a) The height and weight of some people. (b) A possible clustering using $K = 2$ clusters. Figure generated by `kmeansHeightWeight`.

manually label the data. Labeled data is not only expensive to acquire⁶, but it also contains relatively little information, certainly not enough to reliably estimate the parameters of complex models. Geoff Hinton, who is a famous professor of ML at the University of Toronto, has said:

When we're learning to see, nobody's telling us what the right answers are — we just look. Every so often, your mother says "that's a dog", but that's very little information. You'd be lucky if you got a few bits of information — even one bit per second — that way. The brain's visual system has 10^{14} neural connections. And you only live for 10^9 seconds. So it's no use learning one bit per second. You need more like 10^5 bits per second. And there's only one place you can get that much information: from the input itself. — Geoffrey Hinton, 1996 (quoted in (Gorder 2006)).

Below we describe some canonical examples of unsupervised learning.

1.3.1 Discovering clusters

As a canonical example of unsupervised learning, consider the problem of **clustering** data into groups. For example, Figure 1.8(a) plots some 2d data, representing the height and weight of a group of 210 people. It seems that there might be various clusters, or subgroups, although it is not clear how many. Let K denote the number of clusters. Our first goal is to estimate the distribution over the number of clusters, $p(K|\mathcal{D})$; this tells us if there are subpopulations within the data. For simplicity, we often approximate the distribution $p(K|\mathcal{D})$ by its mode, $K^* = \arg \max_K p(K|\mathcal{D})$. In the supervised case, we were told that there are two classes (male and female), but in the unsupervised case, we are free to choose as many or few clusters as we like. Picking a model of the “right” complexity is called model selection, and will be discussed in detail below.

Our second goal is to estimate which cluster each point belongs to. Let $z_i \in \{1, \dots, K\}$ represent the cluster to which data point i is assigned. (z_i is an example of a **hidden** or

6. The advent of **crowd sourcing** web sites such as Mechanical Turk, (<https://www.mturk.com/mturk/welcome>), which outsource data processing tasks to humans all over the world, has reduced the cost of labeling data. Nevertheless, the amount of unlabeled data is still orders of magnitude larger than the amount of labeled data.

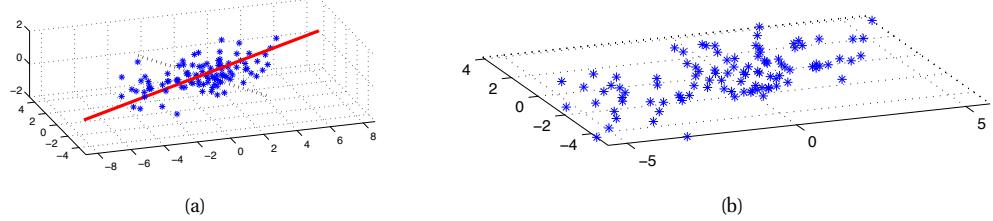


Figure 1.9 (a) A set of points that live on a 2d linear subspace embedded in 3d. The solid red line is the first principal component direction. The dotted black line is the second PC direction. (b) 2D representation of the data. Figure generated by `pcaDemo3d`.

latent variable, since it is never observed in the training set.) We can infer which cluster each data point belongs to by computing $z_i^* = \operatorname{argmax}_k p(z_i = k | \mathbf{x}_i, \mathcal{D})$. This is illustrated in Figure 1.8(b), where we use different colors to indicate the assignments, assuming $K = 2$.

In this book, we focus on **model based clustering**, which means we fit a probabilistic model to the data, rather than running some ad hoc algorithm. The advantages of the model-based approach are that one can compare different kinds of models in an objective way (in terms of the likelihood they assign to the data), we can combine them together into larger systems, etc.

Here are some real world applications of clustering.

- In astronomy, the **autoclass** system (Cheeseman et al. 1988) discovered a new type of star, based on clustering astrophysical measurements.
- In e-commerce, it is common to cluster users into groups, based on their purchasing or web-surfing behavior, and then to send customized targeted advertising to each group (see e.g., (Berkhin 2006)).
- In biology, it is common to cluster flow-cytometry data into groups, to discover different sub-populations of cells (see e.g., (Lo et al. 2009)).

1.3.2 Discovering latent factors

When dealing with high dimensional data, it is often useful to reduce the dimensionality by projecting the data to a lower dimensional subspace which captures the “essence” of the data. This is called **dimensionality reduction**. A simple example is shown in Figure 1.9, where we project some 3d data down to a 2d plane. The 2d approximation is quite good, since most points lie close to this subspace. Reducing to 1d would involve projecting points onto the red line in Figure 1.9(a); this would be a rather poor approximation. (We will make this notion precise in Chapter 12.)

The motivation behind this technique is that although the data may appear high dimensional, there may only be a small number of degrees of variability, corresponding to **latent factors**. For example, when modeling the appearance of face images, there may only be a few underlying latent factors which describe most of the variability, such as lighting, pose, identity, etc, as illustrated in Figure 1.10.

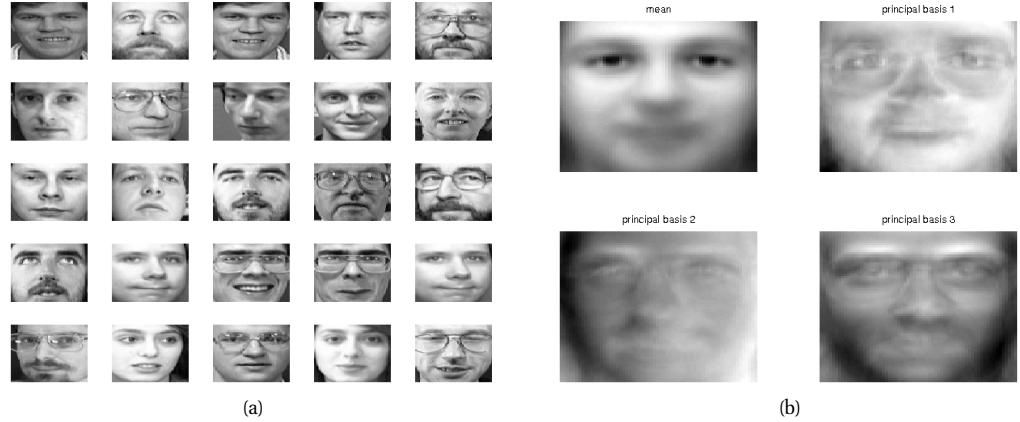


Figure 1.10 a) 25 randomly chosen 64×64 pixel images from the Olivetti face database. (b) The mean and the first three principal component basis vectors (eigenfaces). Figure generated by `pcaImageDemo`.

When used as input to other statistical models, such low dimensional representations often result in better predictive accuracy, because they focus on the “essence” of the object, filtering out inessential features. Also, low dimensional representations are useful for enabling fast nearest neighbor searches and two dimensional projections are very useful for **visualizing** high dimensional data.

The most common approach to dimensionality reduction is called **principal components analysis** or **PCA**. This can be thought of as an unsupervised version of (multi-output) linear regression, where we observe the high-dimensional response \mathbf{y} , but not the low-dimensional “cause” \mathbf{z} . Thus the model has the form $\mathbf{z} \rightarrow \mathbf{y}$; we have to “invert the arrow”, and infer the latent low-dimensional \mathbf{z} from the observed high-dimensional \mathbf{y} . See Section 12.1 for details.

Dimensionality reduction, and PCA in particular, has been applied in many different areas. Some examples include the following:

- In biology, it is common to use PCA to interpret gene microarray data, to account for the fact that each measurement is usually the result of many genes which are correlated in their behavior by the fact that they belong to different biological pathways.
- In natural language processing, it is common to use a variant of PCA called latent semantic analysis for document retrieval (see Section 27.2.2).
- In signal processing (e.g., of acoustic or neural signals), it is common to use ICA (which is a variant of PCA) to separate signals into their different sources (see Section 12.6).
- In computer graphics, it is common to project motion capture data to a low dimensional space, and use it to create animations. See Section 15.5 for one way to tackle such problems.

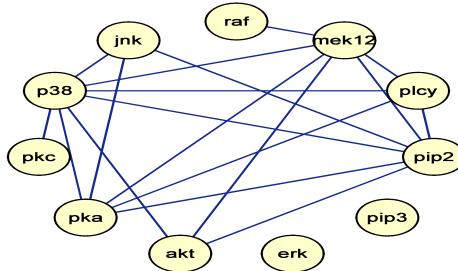


Figure 1.11 A sparse undirected Gaussian graphical model learned using graphical lasso (Section 26.7.2) applied to some flow cytometry data (from (Sachs et al. 2005)), which measures the phosphorylation status of 11 proteins. Figure generated by `ggmLassoDemo`.

1.3.3 Discovering graph structure

Sometimes we measure a set of correlated variables, and we would like to discover which ones are most correlated with which others. This can be represented by a graph G , in which nodes represent variables, and edges represent direct dependence between variables (we will make this precise in Chapter 10, when we discuss graphical models). We can then learn this graph structure from data, i.e., we compute $\hat{G} = \text{argmax } p(G|\mathcal{D})$.

As with unsupervised learning in general, there are two main applications for learning sparse graphs: to discover new knowledge, and to get better joint probability density estimators. We now give some examples of each.

- Much of the motivation for learning sparse graphical models comes from the systems biology community. For example, suppose we measure the phosphorylation status of some proteins in a cell (Sachs et al. 2005). Figure 1.11 gives an example of a graph structure that was learned from this data (using methods discussed in Section 26.7.2). As another example, Smith et al. (2006) showed that one can recover the neural “wiring diagram” of a certain kind of bird from time-series EEG data. The recovered structure closely matched the known functional connectivity of this part of the bird brain.
- In some cases, we are not interested in interpreting the graph structure, we just want to use it to model correlations and to make predictions. One example of this is in financial portfolio management, where accurate models of the covariance between large numbers of different stocks is important. Carvalho and West (2007) show that by learning a sparse graph, and then using this as the basis of a trading strategy, it is possible to outperform (i.e., make more money than) methods that do not exploit sparse graphs. Another example is predicting traffic jams on the freeway. Horvitz et al. (2005) describe a deployed system called JamBayes for predicting traffic flow in the Seattle area; predictions are made using a graphical model whose structure was learned from data.