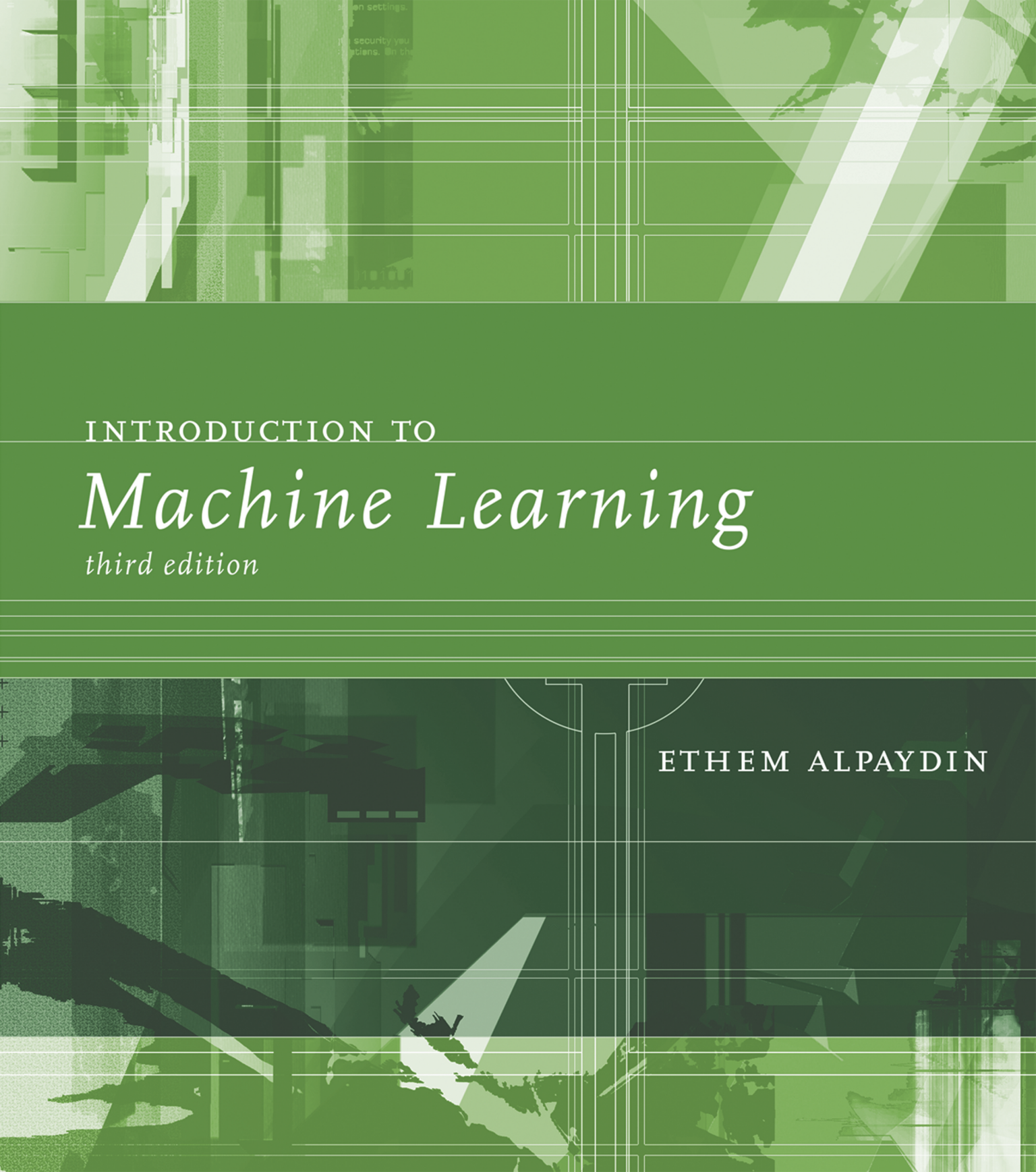

INTRODUCTION TO

# *Machine Learning*

*third edition*

ETHEM ALPAYDIN

# Introduction to Machine Learning

Third Edition

**Adaptive Computation and Machine Learning**

Thomas Dietterich, Editor
Christopher Bishop, David Heckerman, Michael Jordan, and Michael Kearns, Associate Editors

A complete list of books published in The Adaptive Computation and Machine Learning series appears at the back of this book.

# Introduction to Machine Learning

Third Edition

Ethem Alpaydın

The MIT Press
Cambridge, Massachusetts
London, England

Typeset in 10/13 Lucida Bright by the author using LaTeX 2$_\varepsilon$.
Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Information

Alpaydin, Ethem.
Introduction to machine learning / Ethem Alpaydin—3rd ed.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-262-02818-9 (hardcover : alk. paper)
1. Machine learning. I. Title
Q325.5.A46 2014
006.3'1—dc23 2014007214
CIP

10 9 8 7 6 5 4 3 2 1

# *Brief Contents*

# *Contents*

who send me words of appreciation, criticism, or errata, or who provide feedback in any other way. Please keep them coming. My email address is `alpaydin@boun.edu.tr`. The book's web site is `http://www.cmpe.boun.edu.tr/~ethem/i2ml3e`.

It has been a pleasure to work with the MIT Press again on this third edition, and I thank Marie Lufkin Lee, Marc Lowenthal, and Kathleen Caruso for all their help and support.

# *Preface*

Machine learning must be one of the fastest growing fields in computer science. It is not only that the data is continuously getting "bigger," but also the theory to process it and turn it into knowledge. In various fields of science, from astronomy to biology, but also in everyday life, as digital technology increasingly infiltrates our daily existence, as our digital footprint deepens, more data is continuously generated and collected. Whether scientific or personal, data that just lies dormant passively is not of any use, and smart people have been finding ever new ways to make use of that data and turn it into a useful product or service. In this transformation, machine learning plays a larger and larger role.

This data evolution has been continuing even stronger since the second edition appeared in 2010. Every year, datasets are getting larger. Not only has the number of observations grown, but the number of observed attributes has also increased significantly. There is more structure to the data: It is not just numbers and character strings any more but images, video, audio, documents, web pages, click logs, graphs, and so on. More and more, the data moves away from the parametric assumptions we used to make—for example, normality. Frequently, the data is dynamic and so there is a time dimension. Sometimes, our observations are multi-view—for the same object or event, we have multiple sources of information from different sensors and modalities.

Our belief is that behind all this seemingly complex and voluminous data, there lies a simple explanation. That although the data is big, it can be explained in terms of a relatively simple model with a small number of hidden factors and their interaction. Think about millions of customers who each day buy thousands of products online or from their local supermarket. This implies a very large database of transactions, but there is a

pattern to this data. People do not shop at random. A person throwing a party buys a certain subset of products, and a person who has a baby at home buys a different subset; there are hidden factors that explain customer behavior.

This is one of the areas where significant research has been done in recent years—namely, to infer this hidden model from observed data. Most of the revisions in this new edition are related to these advances. Chapter 6 contains new sections on feature embedding, singular value decomposition and matrix factorization, canonical correlation analysis, and Laplacian eigenmaps.

There are new sections on distance estimation in chapter 8 and on kernel machines in chapter 13: Dimensionality reduction, feature extraction, and distance estimation are three names for the same devil—the ideal distance measure is defined in the space of the ideal hidden features, and they are fewer in number than the values we observe.

Chapter 16 is rewritten and significantly extended to cover such generative models. We discuss the Bayesian approach for all major machine learning models, namely, classification, regression, mixture models, and dimensionality reduction. Nonparametric Bayesian modeling, which has become increasingly popular during these last few years, is especially interesting because it allows us to adjust the complexity of the model to the complexity of data.

New sections have been added here and there, mostly to highlight different recent applications of the same or very similar methods. There is a new section on outlier detection in chapter 8. Two new sections in chapters 10 and 13 discuss ranking for linear models and kernel machines, respectively. Having added Laplacian eigenmaps to chapter 6, I also include a new section on spectral clustering in chapter 7. Given the recent resurgence of deep neural networks, it became necessary to include a new section on deep learning in chapter 11. Chapter 19 contains a new section on multivariate tests for comparison of methods.

Since the first edition, I have received many requests for the solutions to exercises from readers who use the book for self-study. In this new edition, I have included the solutions to some of the more didactic exercises. Sometimes they are complete solutions, and sometimes they give just a hint or offer only one of several possible solutions.

I would like to thank all the instructors and students who have used the previous two editions, as well as their translations into German, Chinese, and Turkish, and their reprints in India. I am always grateful to those

# *Notations*

| | |
|---|---|
| $x$ | Scalar value |
| $\boldsymbol{x}$ | Vector |
| $\mathbf{X}$ | Matrix |
| $\boldsymbol{x}^T$ | Transpose |
| $\mathbf{X}^{-1}$ | Inverse |
| | |
| $X$ | Random variable |
| $P(X)$ | Probability mass function when $X$ is discrete |
| $p(X)$ | Probability density function when $X$ is continuous |
| $P(X\|Y)$ | Conditional probability of $X$ given $Y$ |
| $E[X]$ | Expected value of the random variable $X$ |
| $\text{Var}(X)$ | Variance of $X$ |
| $\text{Cov}(X, Y)$ | Covariance of $X$ and $Y$ |
| $\text{Corr}(X, Y)$ | Correlation of $X$ and $Y$ |
| | |
| $\mu$ | Mean |
| $\sigma^2$ | Variance |
| $\boldsymbol{\Sigma}$ | Covariance matrix |
| $m$ | Estimator to the mean |
| $s^2$ | Estimator to the variance |
| $\mathbf{S}$ | Estimator to the covariance matrix |

| | |
|---|---|
| $\mathcal{N}(\mu, \sigma^2)$ | Univariate normal distribution with mean $\mu$ and variance $\sigma^2$ |
| $\mathcal{Z}$ | Unit normal distribution: $\mathcal{N}(0, 1)$ |
| $\mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ | $d$-variate normal distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ |
| | |
| $x$ | Input |
| $d$ | Number of inputs (input dimensionality) |
| $y$ | Output |
| $r$ | Required output |
| $K$ | Number of outputs (classes) |
| $N$ | Number of training instances |
| $z$ | Hidden value, intrinsic dimension, latent factor |
| $k$ | Number of hidden dimensions, latent factors |
| $C_i$ | Class $i$ |
| $\mathcal{X}$ | Training sample |
| $\{x^t\}_{t=1}^N$ | Set of $x$ with index $t$ ranging from 1 to $N$ |
| $\{x^t, r^t\}_t$ | Set of ordered pairs of input and desired output with index $t$ |
| | |
| $g(x\|\theta)$ | Function of $x$ defined up to a set of parameters $\theta$ |
| $\arg\max_\theta g(x\|\theta)$ | The argument $\theta$ for which $g$ has its maximum value |
| $\arg\min_\theta g(x\|\theta)$ | The argument $\theta$ for which $g$ has its minimum value |
| $E(\theta\|\mathcal{X})$ | Error function with parameters $\theta$ on the sample $\mathcal{X}$ |
| $l(\theta\|\mathcal{X})$ | Likelihood of parameters $\theta$ on the sample $\mathcal{X}$ |
| $\mathcal{L}(\theta\|\mathcal{X})$ | Log likelihood of parameters $\theta$ on the sample $\mathcal{X}$ |
| | |
| $1(c)$ | 1 if $c$ is true, 0 otherwise |
| $\#\{c\}$ | Number of elements for which $c$ is true |
| $\delta_{ij}$ | Kronecker delta: 1 if $i = j$, 0 otherwise |

# 1 *Introduction*

## 1.1 What Is Machine Learning?

THIS IS the age of "big data." Once upon a time, only companies had data. There used to be computer centers where that data was stored and processed. First with the arrival of personal computers and later with the widespread use of wireless communications, we all became producers of data. Every time we buy a product, every time we rent a movie, visit a web page, write a blog, or post on the social media, even when we just walk or drive around, we are generating data.

Each of us is not only a generator but also a consumer of data. We want to have products and services specialized for us. We want our needs to be understood and interests to be predicted.

Think, for example, of a supermarket chain that is selling thousands of goods to millions of customers either at hundreds of brick-and-mortar stores all over a country or through a virtual store over the web. The details of each transaction are stored: date, customer id, goods bought and their amount, total money spent, and so forth. This typically amounts to a lot of data every day. What the supermarket chain wants is to be able to predict which customer is likely to buy which product, to maximize sales and profit. Similarly each customer wants to find the set of products best matching his/her needs.

This task is not evident. We do not know exactly which people are likely to buy this ice cream flavor or the next book of this author, see this new movie, visit this city, or click this link. Customer behavior changes in time and by geographic location. But we know that it is not completely random. People do not go to supermarkets and buy things at random. When they buy beer, they buy chips; they buy ice cream in summer and

spices for Glühwein in winter. There are certain patterns in the data.

To solve a problem on a computer, we need an algorithm. An algorithm is a sequence of instructions that should be carried out to transform the input to output. For example, one can devise an algorithm for sorting. The input is a set of numbers and the output is their ordered list. For the same task, there may be various algorithms and we may be interested in finding the most efficient one, requiring the least number of instructions or memory or both.

For some tasks, however, we do not have an algorithm. Predicting customer behavior is one; another is to tell spam emails from legitimate ones. We know what the input is: an email document that in the simplest case is a file of characters. We know what the output should be: a yes/no output indicating whether the message is spam or not. But we do not know how to transform the input to the output. What is considered spam changes in time and from individual to individual.

What we lack in knowledge, we make up for in data. We can easily compile thousands of example messages, some of which we know to be spam and some of which are not, and what we want is to "learn" what constitutes spam from them. In other words, we would like the computer (machine) to extract automatically the algorithm for this task. There is no need to learn to sort numbers since we already have algorithms for that, but there are many applications for which we do not have an algorithm but have lots of data.

We may not be able to identify the process completely, but we believe we can construct *a good and useful approximation*. That approximation may not explain everything, but may still be able to account for some part of the data. We believe that though identifying the complete process may not be possible, we can still detect certain patterns or regularities. This is the niche of machine learning. Such patterns may help us understand the process, or we can use those patterns to make predictions: Assuming that the future, at least the near future, will not be much different from the past when the sample data was collected, the future predictions can also be expected to be right.

Application of machine learning methods to large databases is called *data mining*. The analogy is that a large volume of earth and raw material is extracted from a mine, which when processed leads to a small amount of very precious material; similarly, in data mining, a large volume of data is processed to construct a simple model with valuable use, for example, having high predictive accuracy. Its application areas are

abundant: In addition to retail, in finance banks analyze their past data to build models to use in credit applications, fraud detection, and the stock market. In manufacturing, learning models are used for optimization, control, and troubleshooting. In medicine, learning programs are used for medical diagnosis. In telecommunications, call patterns are analyzed for network optimization and maximizing the quality of service. In science, large amounts of data in physics, astronomy, and biology can only be analyzed fast enough by computers. The World Wide Web is huge; it is constantly growing, and searching for relevant information cannot be done manually.

But machine learning is not just a database problem; it is also a part of artificial intelligence. To be intelligent, a system that is in a changing environment should have the ability to learn. If the system can learn and adapt to such changes, the system designer need not foresee and provide solutions for all possible situations.

Machine learning also helps us find solutions to many problems in vision, speech recognition, and robotics. Let us take the example of recognizing faces: This is a task we do effortlessly; every day we recognize family members and friends by looking at their faces or from their photographs, despite differences in pose, lighting, hair style, and so forth. But we do it unconsciously and are unable to explain how we do it. Because we are not able to explain our expertise, we cannot write the computer program. At the same time, we know that a face image is not just a random collection of pixels; a face has structure. It is symmetric. There are the eyes, the nose, the mouth, located in certain places on the face. Each person's face is a pattern composed of a particular combination of these. By analyzing sample face images of a person, a learning program captures the pattern specific to that person and then recognizes by checking for this pattern in a given image. This is one example of *pattern recognition*.

Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be *predictive* to make predictions in the future, or *descriptive* to gain knowledge from data, or both.

Machine learning uses the theory of statistics in building mathematical models, because the core task is making inference from a sample. The role of computer science is twofold: First, in training, we need efficient

algorithms to solve the optimization problem, as well as to store and process the massive amount of data we generally have. Second, once a model is learned, its representation and algorithmic solution for inference needs to be efficient as well. In certain applications, the efficiency of the learning or inference algorithm, namely, its space and time complexity, may be as important as its predictive accuracy.

Let us now discuss some example applications in more detail to gain more insight into the types and uses of machine learning.

## 1.2 Examples of Machine Learning Applications

### 1.2.1 Learning Associations

In the case of retail—for example, a supermarket chain—one application of machine learning is *basket analysis*, which is finding associations between products bought by customers: If people who buy $X$ typically also buy $Y$, and if there is a customer who buys $X$ and does not buy $Y$, he or she is a potential $Y$ customer. Once we find such customers, we can target them for cross-selling.

ASSOCIATION RULE

In finding an *association rule*, we are interested in learning a conditional probability of the form $P(Y|X)$ where $Y$ is the product we would like to condition on $X$, which is the product or the set of products which we know that the customer has already purchased.

Let us say, going over our data, we calculate that $P(\text{chips}|\text{beer}) = 0.7$. Then, we can define the rule:

70 percent of customers who buy beer also buy chips.

We may want to make a distinction among customers and toward this, estimate $P(Y|X, D)$ where $D$ is the set of customer attributes, for example, gender, age, marital status, and so on, assuming that we have access to this information. If this is a bookseller instead of a supermarket, products can be books or authors. In the case of a web portal, items correspond to links to web pages, and we can estimate the links a user is likely to click and use this information to download such pages in advance for faster access.

### 1.2.2 Classification

A credit is an amount of money loaned by a financial institution, for example, a bank, to be paid back with interest, generally in installments. It is important for the bank to be able to predict in advance the risk associated with a loan, which is the probability that the customer will default and not pay the whole amount back. This is both to make sure that the bank will make a profit and also to not inconvenience a customer with a loan over his or her financial capacity.

In *credit scoring* (Hand 1998), the bank calculates the risk given the amount of credit and the information about the customer. The information about the customer includes data we have access to and is relevant in calculating his or her financial capacity—namely, income, savings, collaterals, profession, age, past financial history, and so forth. The bank has a record of past loans containing such customer data and whether the loan was paid back or not. From this data of particular applications, the aim is to infer a general rule coding the association between a customer's attributes and his risk. That is, the machine learning system fits a model to the past data to be able to calculate the risk for a new application and then decides to accept or refuse it accordingly.

CLASSIFICATION

This is an example of a *classification* problem where there are two classes: low-risk and high-risk customers. The information about a customer makes up the *input* to the classifier whose task is to assign the input to one of the two classes.

After training with the past data, a classification rule learned may be of the form

IF income$> \theta_1$ AND savings$> \theta_2$ THEN low-risk ELSE high-risk

for suitable values of $\theta_1$ and $\theta_2$ (see figure 1.1). This is an example of a *discriminant*; it is a function that separates the examples of different classes.

DISCRIMINANT

PREDICTION

Having a rule like this, the main application is *prediction*: Once we have a rule that fits the past data, if the future is similar to the past, then we can make correct predictions for novel instances. Given a new application with a certain income and savings, we can easily decide whether it is low-risk or high-risk.

In some cases, instead of making a 0/1 (low-risk/high-risk) type decision, we may want to calculate a probability, namely, $P(Y|X)$, where $X$ are the customer attributes and $Y$ is 0 or 1 respectively for low-risk

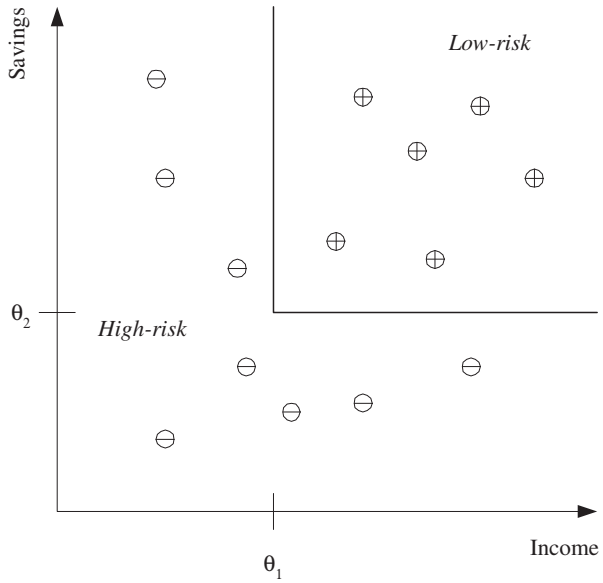


**Figure 1.1** Example of a training dataset where each circle corresponds to one data instance with input values in the corresponding axes and its sign indicates the class. For simplicity, only two customer attributes, income and savings, are taken as input and the two classes are low-risk ('+') and high-risk ('−'). An example discriminant that separates the two types of examples is also shown.

and high-risk. From this perspective, we can see classification as learning an association from $X$ to $Y$. Then for a given $X = x$, if we have $P(Y = 1|X = x) = 0.8$, we say that the customer has an 80 percent probability of being high-risk, or equivalently a 20 percent probability of being low-risk. We then decide whether to accept or refuse the loan depending on the possible gain and loss.

PATTERN RECOGNITION

There are many applications of machine learning in *pattern recognition*. One is *optical character recognition*, which is recognizing character codes from their images. This is an example where there are multiple classes, as many as there are characters we would like to recognize. Especially interesting is the case when the characters are handwritten—for example, to read zip codes on envelopes or amounts on checks. People have different handwriting styles; characters may be written small or large, slanted, with a pen or pencil, and there are many possible images corresponding

to the same character. Though writing is a human invention, we do not have any system that is as accurate as a human reader. We do not have a formal description of 'A' that covers all 'A's and none of the non-'A's. Not having it, we take samples from writers and learn a definition of A-ness from these examples. But though we do not know what it is that makes an image an 'A', we are certain that all those distinct 'A's have something in common, which is what we want to extract from the examples. We know that a character image is not just a collection of random dots; it is a collection of strokes and has a regularity that we can capture by a learning program.

If we are reading a text, one factor we can make use of is the redundancy in human languages. A word is a *sequence* of characters and successive characters are not independent but are constrained by the words of the language. This has the advantage that even if we cannot recognize a character, we can still read t?e word. Such contextual dependencies may also occur in higher levels, between words and sentences, through the syntax and semantics of the language. There are machine learning algorithms to learn sequences and model such dependencies.

In the case of *face recognition*, the input is an image, the classes are people to be recognized, and the learning program should learn to associate the face images to identities. This problem is more difficult than optical character recognition because there are more classes, input image is larger, and a face is three-dimensional and differences in pose and lighting cause significant changes in the image. There may also be occlusion of certain inputs; for example, glasses may hide the eyes and eyebrows, and a beard may hide the chin.

In *medical diagnosis*, the inputs are the relevant information we have about the patient and the classes are the illnesses. The inputs contain the patient's age, gender, past medical history, and current symptoms. Some tests may not have been applied to the patient, and thus these inputs would be missing. Tests take time, may be costly, and may inconvenience the patient so we do not want to apply them unless we believe that they will give us valuable information. In the case of a medical diagnosis, a wrong decision may lead to a wrong or no treatment, and in cases of doubt it is preferable that the classifier reject and defer decision to a human expert.

In *speech recognition*, the input is acoustic and the classes are words that can be uttered. This time the association to be learned is from an acoustic signal to a word of some language. Different people, because

of differences in age, gender, or accent, pronounce the same word differently, which makes this task rather difficult. Another difference of speech is that the input is *temporal*; words are uttered in time as a sequence of speech phonemes and some words are longer than others.

Acoustic information only helps up to a certain point, and as in optical character recognition, the integration of a "language model" is critical in speech recognition, and the best way to come up with a language model is again by learning it from some large corpus of example data. The applications of machine learning to *natural language processing* is constantly increasing. Spam filtering is one where spam generators on one side and filters on the other side keep finding more and more ingenious ways to outdo each other. Summarizing large documents is another interesting example, yet another is analyzing blogs or posts on social networking sites to extract "trending" topics or to determine what to advertise. Perhaps the most impressive would be *machine translation*. After decades of research on hand-coded translation rules, it has become apparent that the most promising way is to provide a very large number of example pairs of texts in both languages and have a program figure out automatically the rules to map one to the other.

*Biometrics* is recognition or authentication of people using their physiological and/or behavioral characteristics that requires an integration of inputs from different modalities. Examples of physiological characteristics are images of the face, fingerprint, iris, and palm; examples of behavioral characteristics are dynamics of signature, voice, gait, and key stroke. As opposed to the usual identification procedures—photo, printed signature, or password—when there are many different (uncorrelated) inputs, forgeries (spoofing) would be more difficult and the system would be more accurate, hopefully without too much inconvenience to the users. Machine learning is used both in the separate recognizers for these different modalities and in the combination of their decisions to get an overall accept/reject decision, taking into account how reliable these different sources are.

KNOWLEDGE EXTRACTION

Learning a rule from data also allows *knowledge extraction*. The rule is a simple model that explains the data, and looking at this model we have an explanation about the process underlying the data. For example, once we learn the discriminant separating low-risk and high-risk customers, we have the knowledge of the properties of low-risk customers. We can then use this information to target potential low-risk customers more efficiently, for example, through advertising. Learning also performs *com-*

COMPRESSION

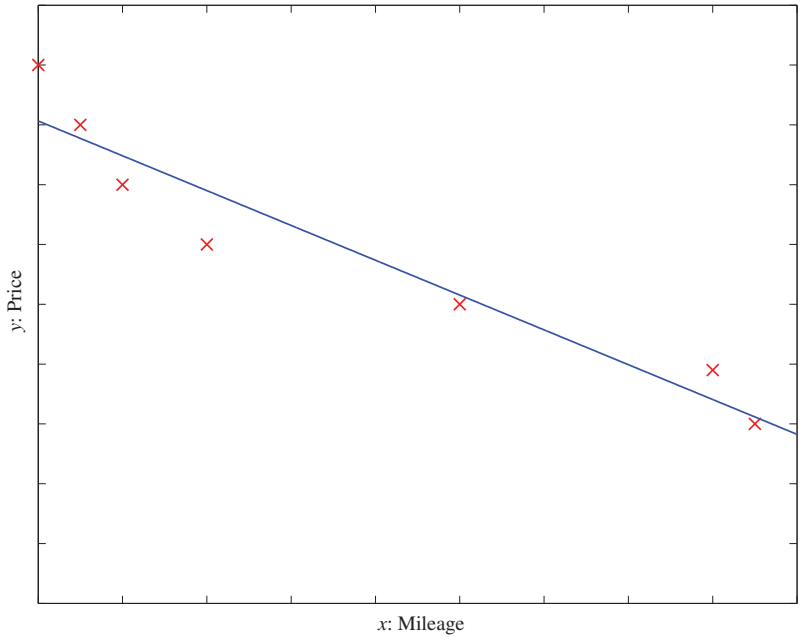


**Figure 1.2** A training dataset of used cars and the function fitted. For simplicity, mileage is taken as the only input attribute and a linear model is used.

program optimizes the parameters, $\theta$, such that the approximation error is minimized, that is, our estimates are as close as possible to the correct values given in the training set. For example in figure 1.2, the model is linear, and $w$ and $w_0$ are the parameters optimized for best fit to the training data. In cases where the linear model is too restrictive, we can use, for example, a quadratic

$$y = w_2 x^2 + w_1 x + w_0$$

or a higher-order polynomial, or any other nonlinear function of the input, this time optimizing its parameters for best fit.

Another example of regression is navigation of a mobile robot, for example, an autonomous car, where the output is the angle by which the steering wheel should be turned at each time, to advance without hitting obstacles and deviating from the route. Inputs in such a case are provided by sensors on the car—for example, a video camera, GPS, and so

forth. Training data can be collected by monitoring and recording the actions of a human driver.

We can envisage other applications of regression where we are trying to optimize a function.[1] Let us say we want to build a machine that roasts coffee. The machine has many inputs that affect the quality: various settings of temperatures, times, coffee bean type, and so forth. We make a number of experiments and for different settings of these inputs, we measure the quality of the coffee, for example, as consumer satisfaction. To find the optimal setting, we fit a regression model linking these inputs to coffee quality and choose new points to sample near the optimum of the current model to look for a better configuration. We sample these points, check quality, and add these to the data and fit a new model. This is generally called *response surface design*.

Sometimes instead of estimating an absolute numeric value, we want to be able to learn relative positions. For example, in a *recommendation system* for movies, we want to generate a list ordered by how much we believe the user is likely to enjoy each. Depending on the movie attributes such as genre, actors, and so on, and using the ratings of the user he/she has already seen, we would like to be able to learn a *ranking* function that we can then use to choose among new movies.

RANKING

### 1.2.4 Unsupervised Learning

In supervised learning, the aim is to learn a mapping from the input to an output whose correct values are provided by a supervisor. In unsupervised learning, there is no such supervisor and we only have input data. The aim is to find the regularities in the input. There is a structure to the input space such that certain patterns occur more often than others, and we want to see what generally happens and what does not. In statistics, this is called *density estimation*.

DENSITY ESTIMATION

CLUSTERING

One method for density estimation is *clustering* where the aim is to find clusters or groupings of input. In the case of a company with a data of past customers, the customer data contains the demographic information as well as the past transactions with the company, and the company may want to see the distribution of the profile of its customers, to see what type of customers frequently occur. In such a case, a clustering model allocates customers similar in their attributes to the same group,

1. I would like to thank Michael Jordan for this example.

providing the company with natural groupings of its customers; this is called *customer segmentation*. Once such groups are found, the company may decide strategies, for example, services and products, specific to different groups; this is known as *customer relationship management*. Such a grouping also allows identifying those who are outliers, namely, those who are different from other customers, which may imply a niche in the market that can be further exploited by the company.

An interesting application of clustering is in *image compression*. In this case, the input instances are image pixels represented as RGB values. A clustering program groups pixels with similar colors in the same group, and such groups correspond to the colors occurring frequently in the image. If in an image, there are only shades of a small number of colors, and if we code those belonging to the same group with one color, for example, their average, then the image is quantized. Let us say the pixels are 24 bits to represent 16 million colors, but if there are shades of only 64 main colors, for each pixel we need 6 bits instead of 24. For example, if the scene has various shades of blue in different parts of the image, and if we use the same average blue for all of them, we lose the details in the image but gain space in storage and transmission. Ideally, we would like to identify higher-level regularities by analyzing repeated image patterns, for example, texture, objects, and so forth. This allows a higher-level, simpler, and more useful description of the scene, and for example, achieves better compression than compressing at the pixel level. If we have scanned document pages, we do not have random on/off pixels but bitmap images of characters. There is structure in the data, and we make use of this redundancy by finding a shorter description of the data: $16 \times 16$ bitmap of 'A' takes 32 bytes; its ASCII code is only 1 byte.

In *document clustering*, the aim is to group similar documents. For example, news reports can be subdivided as those related to politics, sports, fashion, arts, and so on. Commonly, a document is represented as a *bag of words*—that is, we predefine a lexicon of $N$ words, and each document is an $N$-dimensional binary vector whose element $i$ is 1 if word $i$ appears in the document; suffixes "–s" and "–ing" are removed to avoid duplicates and words such as "of," "and," and so forth, which are not informative, are not used. Documents are then grouped depending on the number of shared words. It is of course critical how the lexicon is chosen.

Machine learning methods are also used in *bioinformatics*. DNA in our genome is the "blueprint of life" and is a sequence of bases, namely, A, G,

C, and T. RNA is transcribed from DNA, and proteins are translated from the RNA. Proteins are what the living body is and does. Just as a DNA is a sequence of bases, a protein is a sequence of amino acids (as defined by bases). One application area of computer science in molecular biology is *alignment*, which is matching one sequence to another. This is a difficult string matching problem because strings may be quite long, there are many template strings to match against, and there may be deletions, insertions, and substitutions. Clustering is used in learning *motifs*, which are sequences of amino acids that occur repeatedly in proteins. Motifs are of interest because they may correspond to structural or functional elements within the sequences they characterize. The analogy is that if the amino acids are letters and proteins are sentences, motifs are like words, namely, a string of letters with a particular meaning occurring frequently in different sentences.

### 1.2.5 Reinforcement Learning

In some applications, the output of the system is a sequence of *actions*. In such a case, a single action is not important; what is important is the *policy* that is the sequence of correct actions to reach the goal. There is no such thing as the best action in any intermediate state; an action is good if it is part of a good policy. In such a case, the machine learning program should be able to assess the goodness of policies and learn from past good action sequences to be able to generate a policy. Such learning methods are called *reinforcement learning* algorithms.

REINFORCEMENT LEARNING

A good example is *game playing* where a single move by itself is not that important; it is the sequence of right moves that is good. A move is good if it is part of a good game playing policy. Game playing is an important research area in both artificial intelligence and machine learning. This is because games are easy to describe and at the same time, they are quite difficult to play well. A game like chess has a small number of rules but it is very complex because of the large number of possible moves at each state and the large number of moves that a game contains. Once we have good algorithms that can learn to play games well, we can also apply them to applications with more evident economic utility.

A robot navigating in an environment in search of a goal location is another application area of reinforcement learning. At any time, the robot can move in one of a number of directions. After a number of trial runs, it should learn the correct sequence of actions to reach to the goal state

from an initial state, doing this as quickly as possible and without hitting any of the obstacles.

One factor that makes reinforcement learning harder is when the system has unreliable and partial sensory information. For example, a robot equipped with a video camera has incomplete information and thus at any time is in a *partially observable state* and should decide on its action taking into account this uncertainty; for example, it may not know its exact location in a room but only that there is a wall to its left. A task may also require a concurrent operation of *multiple agents* that should interact and cooperate to accomplish a common goal. An example is a team of robots playing soccer.

## 1.3 Notes

Evolution is the major force that defines our bodily shape as well as our built-in instincts and reflexes. We also learn to change our behavior during our lifetime. This helps us cope with changes in the environment that cannot be predicted by evolution. Organisms that have a short life in a well-defined environment may have all their behavior built-in, but instead of hardwiring into us all sorts of behavior for any circumstance that we could encounter in our life, evolution gave us a large brain and a mechanism to learn, such that we could update ourselves with experience and adapt to different environments. When we learn the best strategy in a certain situation, that knowledge is stored in our brain, and when the situation arises again, when we re-cognize ("cognize" means to know) the situation, we can recall the suitable strategy and act accordingly.

Learning has its limits though; there may be things that we can never learn with the limited capacity of our brains, just like we can never "learn" to grow a third arm, or an eye on the back of our head, even if either would be useful. Note that unlike in psychology, cognitive science, or neuroscience, our aim in machine learning is not to understand the processes underlying learning in humans and animals, but to build useful systems, as in any domain of engineering.

Almost all of science is fitting models to data. Scientists design experiments and make observations and collect data. They then try to extract knowledge by finding out simple models that explain the data they observed. This is called *induction* and is the process of extracting general rules from a set of particular cases.

We are now at a point that such analysis of data can no longer be done by people, both because the amount of data is huge and because people who can do such analysis are rare and manual analysis is costly. There is thus a growing interest in computer models that can analyze data and extract information automatically from them, that is, learn.

The methods we discuss in the coming chapters have their origins in different scientific domains. Sometimes the same algorithm was independently invented in more than one field, following a different historical path.

In statistics, going from particular observations to general descriptions is called *inference* and learning is called *estimation*. Classification is called *discriminant analysis* in statistics (McLachlan 1992; Hastie, Tibshirani, and Friedman 2011). Before computers were cheap and abundant, statisticians could only work with small samples. Statisticians, being mathematicians, worked mostly with simple parametric models that could be analyzed mathematically. In engineering, classification is called *pattern recognition* and the approach is nonparametric and much more empirical (Duda, Hart, and Stork 2001; Webb and Copey 2011).

Machine learning is also related to *artificial intelligence* (Russell and Norvig 2009) because an intelligent system should be able to adapt to changes in its environment. Application areas like vision, speech, and robotics are also tasks that are best learned from sample data. In electrical engineering, research in *signal processing* resulted in adaptive computer vision and speech programs. Among these, the development of *hidden Markov models* for speech recognition is especially important.

In the late 1980s with advances in VLSI technology and the possibility of building parallel hardware containing thousands of processors, the field of *artificial neural networks* was reinvented as a possible theory to distribute computation over a large number of processing units (Bishop 1995). Over time, it has been realized in the neural network community that most neural network learning algorithms have their basis in statistics—for example, the multilayer perceptron is another class of nonparametric estimator—and claims of brain-like computation have started to fade.

In recent years, kernel-based algorithms, such as support vector machines, have become popular, which, through the use of kernel functions, can be adapted to various applications, especially in bioinformatics and language processing. It is common knowledge nowadays that a good representation of data is critical for learning and kernel functions turn out

to be a very good way to introduce such expert knowledge.

Another recent approach is the use of *generative models* that explain the observed data through the interaction of a set of hidden factors. Generally, *graphical models* are used to visualize the interaction of the factors and the data, and *Bayesian formalism* allows us to define our prior information on the hidden factors and the model, as well as to infer the model parameters.

Recently, with the reduced cost of storage and connectivity, it has become possible to have very large datasets available over the Internet, and this, coupled with cheaper computation, have made it possible to run learning algorithms on a lot of data. In the past few decades, it was generally believed that for artificial intelligence to be possible, we needed a new paradigm, a new type of thinking, a new model of computation, or a whole new set of algorithms.

Taking into account the recent successes in machine learning in various domains, it may be claimed that what we needed was not new algorithms but a lot of example data and sufficient computing power to run the algorithms on that much data. For example, the roots of support vector machines go to potential functions, linear classifiers, and neighbor-based methods, proposed in the 1950s or the 1960s; it is just that we did not have fast computers or large storage then for these algorithms to show their full potential. It may be conjectured that tasks such as machine translation, and even planning, can be solved with such relatively simple learning algorithms but trained on large amounts of example data, or through long runs of trial and error. Recent successes with "deep learning" algorithm supports this claim. Intelligence seems not to originate from some outlandish formula, but rather from the patient, almost brute-force use of a simple, straightforward algorithm.

*Data mining* is the name coined in the business world for the application of machine learning algorithms to large amounts of data (Witten and Frank 2011; Han and Kamber 2011). In computer science, it used to be called *knowledge discovery in databases*.

Research in these different communities (statistics, pattern recognition, neural networks, signal processing, control, artificial intelligence, and data mining) followed different paths in the past with different emphases. In this book, the aim is to incorporate these emphases together to give a unified treatment of the problems and the proposed solutions.

## 1.4 Relevant Resources

The latest research on machine learning is distributed over journals and conferences from different fields. Dedicated journals are *Machine Learning* and the *Journal of Machine Learning Research*. Journals such as *Neural Computation*, *Neural Networks*, and *IEEE Transactions on Neural Networks and Learning Systems* publish also heavily machine learning papers. Statistics journals like *Annals of Statistics* and the *Journal of the American Statistical Association* publish papers interesting from the point of view of machine learning, and many of the *IEEE Transactions* such as *Pattern Analysis and Machine Intelligence*, *Systems, Man, and Cybernetics*, *Image Processing*, and *Signal Processing* contain interesting papers related to either the theory of machine learning or one of its numerous applications.

Journals on artificial intelligence, pattern recognition, and signal processing also contain machine learning papers. Journals with an emphasis on data mining are *Data Mining and Knowledge Discovery*, *IEEE Transactions on Knowledge and Data Engineering*, and *ACM Special Interest Group on Knowledge Discovery and Data Mining Explorations Journal*.

The major conferences on machine learning are *Neural Information Processing Systems* (NIPS), *Uncertainty in Artificial Intelligence* (UAI), *International Conference on Machine Learning* (ICML), *European Conference on Machine Learning* (ECML), *Artificial Intelligence and Statistics* (AISTATS), and *Computational Learning Theory* (COLT). Conferences on pattern recognition, neural networks, artificial intelligence, fuzzy logic, and genetic algorithms, along with conferences on application areas like computer vision, speech technology, robotics, and data mining, have sessions on machine learning.

*UCI Repository*, at http://archive.ics.uci.edu/ml, contains a large number of datasets frequently used by machine learning researchers for benchmarking purposes. Another resource is the *Statlib Repository*, which is at http://lib.stat.cmu.edu. In addition to these, there are also repositories for particular applications, for example, computational biology, face recognition, speech recognition, and so forth.

New and larger datasets are constantly being added to these repositories. Still, some researchers believe that such repositories do not reflect the full characteristics of real data and are of limited scope, and therefore accuracies on datasets from such repositories are not indicative of anything. When some datasets from a fixed repository are used repeat-

edly while tailoring a new algorithm, we are generating a new set of "UCI algorithms" specialized for those datasets. It is like students who are studying for a course by solving a set of example questions only. As we see in later chapters, different algorithms are better on different tasks anyway, and therefore it is best to keep one application in mind, to have one or a number of large datasets drawn for that and compare algorithms on those, for that specific task.

Most recent papers by machine learning researchers are accessible over the Internet. Most authors also make their codes and data available over the web. Videos of tutorial lectures of machine learning conferences and summer schools are mostly available too. There are also free software toolboxes and packages implementing various machine learning algorithms, and among these, Weka at http://www.cs.waikato.ac.nz/ml/weka/, is especially noteworthy.

## 1.5 Exercises

1. Imagine we have two possibilities: We can scan and email the image, or we can use an optical character reader (OCR) and send the text file. Discuss the advantage and disadvantages of the two approaches in a comparative manner. When would one be preferable over the other?
2. Let us say we are building an OCR and for each character, we store the bitmap of that character as a template that we match with the read character pixel by pixel. Explain when such a system would fail. Why are barcode readers still used?

   SOLUTION: Such a system allows only one template per character and cannot distinguish characters from multiple fonts, for example. There are standardized fonts such as OCR-A and OCR-B—the fonts we typically see on the packaging of stuff we buy—which are used with OCR software (the characters in these fonts have been slightly changed to minimize the similarities between them). Barcode readers are still used because reading barcodes is still a better (cheaper, more reliable, more available) technology than reading characters in arbitrary font, size, and styles.
3. Assume we are given the task of building a system to distinguish junk email. What is in a junk email that lets us know that it is junk? How can the computer detect junk through a syntactic analysis? What would we like the computer to do if it detects a junk email—delete it automatically, move it to a different file, or just highlight it on the screen?

   SOLUTION: Typically, text-based spam filters check for the existence/absence of words and symbols. Words such as "opportunity," "viagra," "dollars," and

characters such as '$' and '!' increase the probability that the email is spam. These probabilities are learned from a training set of example past emails that the user has previously marked as spam. We see many algorithms for this in later chapters.

The spam filters do not work with 100 percent reliability and may make errors in classification. If a junk mail is not filtered, this is not good, but it is not as bad as filtering a good mail as spam. We discuss how we can take into account the relative costs of such false positives and false negatives later on.

Therefore, mail messages that the system considers as spam should not be automatically deleted but kept aside so that the user can see them if he/she wants to, especially in the early stages of using the spam filter when the system has not yet been trained sufficiently. Spam filtering is probably one of the best application areas of machine learning where learning systems can adapt to changes in the ways spam messages are generated.

4. Let us say we are given the task of building an automated taxi. Define the constraints. What are the inputs? What is the output? How can we communicate with the passenger? Do we need to communicate with the other automated taxis, that is, do we need a "language"?
5. In basket analysis, we want to find the dependence between two items $X$ and $Y$. Given a database of customer transactions, how can we find these dependencies? How would we generalize this to more than two items?
6. In a daily newspaper, find five sample news reports for each category of politics, sports, and the arts. Go over these reports and find words that are used frequently for each category, which may help you discriminate between different categories. For example, a news report on politics is likely to include words such as "government," "recession," "congress," and so forth, whereas a news report on the arts may include "album," "canvas," or "theater." There are also words such as "goal" that are ambiguous.
7. If a face image is a $100 \times 100$ image, written in row-major, this is a 10,000-dimensional vector. If we shift the image one pixel to the right, this will be a very different vector in the 10,000-dimensional space. How can we build face recognizers robust to such distortions?

   SOLUTION: Face recognition systems typically have a preprocessing stage for normalization where the input is centered and possibly resized before recognition. This is generally done by first finding the eyes and then translating the image accordingly. There are also recognizers that do not use the face image as pixels but rather extract structural features from the image, for example, the ratio of the distance between the two eyes to the size of the whole face. Such features would be invariant to translations and size changes.
8. Take, for example, the word "machine." Write it ten times. Also ask a friend to write it ten times. Analyzing these twenty images, try to find features,

types of strokes, curvatures, loops, how you make the dots, and so on, that discriminate your handwriting from that of your friend's.

9. In estimating the price of a used car, it makes more sense to estimate the percent depreciation over the original price than to estimate the absolute price. Why?

## 1.6 References

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition.* Oxford: Oxford University Press.

Duda, R. O., P. E. Hart, and D. G. Stork. 2001. *Pattern Classification,* 2nd ed. New York: Wiley.

Han, J., and M. Kamber. 2011. *Data Mining: Concepts and Techniques,* 3rd ed. San Francisco: Morgan Kaufmann.

Hand, D. J. 1998. "Consumer Credit and Statistics." In *Statistics in Finance,* ed. D. J. Hand and S. D. Jacka, 69–81. London: Arnold.

Hastie, T., R. Tibshirani, and J. Friedman. 2011. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction,* 2nd ed. New York: Springer.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition.* New York: Wiley.

Russell, S., and P. Norvig. 2009. *Artificial Intelligence: A Modern Approach,* 3rd ed. New York: Prentice Hall.

Webb, A., and K. D. Copsey. 2011. *Statistical Pattern Recognition,* 3rd ed. New York: Wiley.

Witten, I. H., and E. Frank. 2005. *Data Mining: Practical Machine Learning Tools and Techniques,* 2nd ed. San Francisco: Morgan Kaufmann.

# 2 *Supervised Learning*

*We discuss supervised learning starting from the simplest case, which is learning a class from its positive and negative examples. We generalize and discuss the case of multiple classes, then regression, where the outputs are continuous.*

## 2.1 Learning a Class from Examples

LET US say we want to learn the *class*, $C$, of a "family car." We have a set of examples of cars, and we have a group of people that we survey to whom we show these cars. The people look at the cars and label them; the cars that they believe are family cars are *positive examples*, and the other cars are *negative examples*. Class learning is finding a description that is shared by all the positive examples and none of the negative examples. Doing this, we can make a prediction: Given a car that we have not seen before, by checking with the description learned, we will be able to say whether it is a family car or not. Or we can do knowledge extraction: This study may be sponsored by a car company, and the aim may be to understand what people expect from a family car.

POSITIVE EXAMPLES
NEGATIVE EXAMPLES

After some discussions with experts in the field, let us say that we reach the conclusion that among all features a car may have, the features that separate a family car from other type of cars are the price and engine power. These two attributes are the *inputs* to the class recognizer. Note that when we decide on this particular *input representation*, we are ignoring various other attributes as irrelevant. Though one may think of other attributes such as seating capacity and color that might be important for distinguishing among car types, we will consider only price and engine power to keep this example simple.

INPUT REPRESENTATION

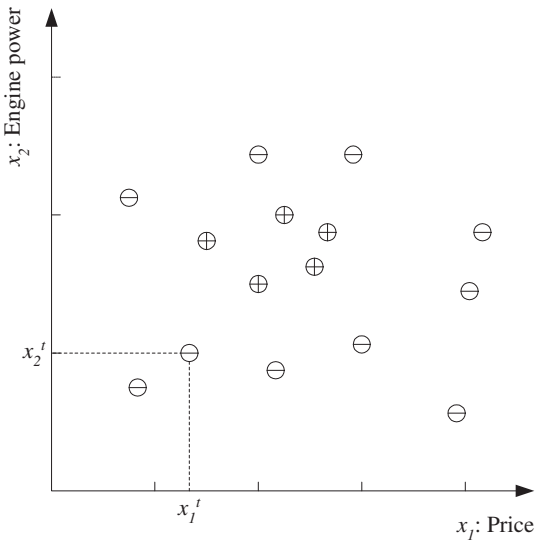


**Figure 2.1** Training set for the class of a "family car." Each data point corresponds to one example car, and the coordinates of the point indicate the price and engine power of that car. '+' denotes a positive example of the class (a family car), and '−' denotes a negative example (not a family car); it is another type of car.

Let us denote price as the first input attribute $x_1$ (e.g., in U.S. dollars) and engine power as the second attribute $x_2$ (e.g., engine volume in cubic centimeters). Thus we represent each car using two numeric values

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{2.1}$$

and its label denotes its type

$$r = \begin{cases} 1 & \text{if } \mathbf{x} \text{ is a positive example} \\ 0 & \text{if } \mathbf{x} \text{ is a negative example} \end{cases} \tag{2.2}$$

Each car is represented by such an ordered pair $(\mathbf{x}, r)$ and the training set contains $N$ such examples

$$\mathcal{X} = \{\mathbf{x}^t, r^t\}_{t=1}^N \tag{2.3}$$

where $t$ indexes different examples in the set; it does not represent time or any such order.

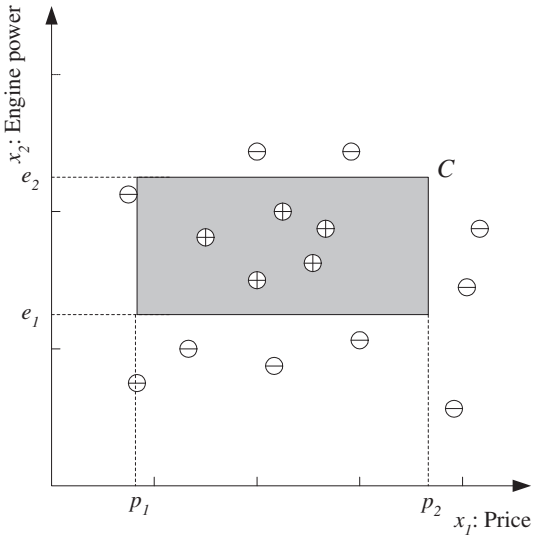


**Figure 2.2** Example of a hypothesis class. The class of family car is a rectangle in the price-engine power space.

Our training data can now be plotted in the two-dimensional $(x_1, x_2)$ space where each instance $t$ is a data point at coordinates $(x_1^t, x_2^t)$ and its type, namely, positive versus negative, is given by $r^t$ (see figure 2.1).

After further discussions with the expert and the analysis of the data, we may have reason to believe that for a car to be a family car, its price and engine power should be in a certain range

$$(p_1 \leq \text{price} \leq p_2) \text{ AND } (e_1 \leq \text{engine power} \leq e_2) \tag{2.4}$$

for suitable values of $p_1, p_2, e_1$, and $e_2$. Equation 2.4 thus assumes $C$ to be a rectangle in the price-engine power space (see figure 2.2).

HYPOTHESIS CLASS

Equation 2.4 fixes $\mathcal{H}$, the *hypothesis class* from which we believe $C$ is drawn, namely, the set of rectangles. The learning algorithm then finds the particular *hypothesis*, $h \in \mathcal{H}$, specified by a particular quadruple of $(p_1^h, p_2^h, e_1^h, e_2^h)$, to approximate $C$ as closely as possible.

HYPOTHESIS

Though the expert defines this hypothesis class, the values of the parameters are not known; that is, though we choose $\mathcal{H}$, we do not know

which particular $h \in \mathcal{H}$ is equal, or closest, to $C$. But once we restrict our attention to this hypothesis class, learning the class reduces to the easier problem of finding the four parameters that define $h$.

The aim is to find $h \in \mathcal{H}$ that is as similar as possible to $C$. Let us say the hypothesis $h$ makes a prediction for an instance $\boldsymbol{x}$ such that

$$h(\boldsymbol{x}) = \begin{cases} 1 & \text{if } h \text{ classifies } \boldsymbol{x} \text{ as a positive example} \\ 0 & \text{if } h \text{ classifies } \boldsymbol{x} \text{ as a negative example} \end{cases} \tag{2.5}$$

In real life we do not know $C(\boldsymbol{x})$, so we cannot evaluate how well $h(\boldsymbol{x})$ matches $C(\boldsymbol{x})$. What we have is the training set $\mathcal{X}$, which is a small subset of the set of all possible $\boldsymbol{x}$. The *empirical error* is the proportion of training instances where *predictions* of $h$ do not match the *required values* given in $\mathcal{X}$. The error of hypothesis $h$ given the training set $\mathcal{X}$ is

EMPIRICAL ERROR

$$E(h|\mathcal{X}) = \sum_{t=1}^{N} 1(h(\boldsymbol{x}^t) \neq r^t) \tag{2.6}$$

where $1(a \neq b)$ is 1 if $a \neq b$ and is 0 if $a = b$ (see figure 2.3).

In our example, the hypothesis class $\mathcal{H}$ is the set of all possible rectangles. Each quadruple $(p_1^h, p_2^h, e_1^h, e_2^h)$ defines one hypothesis, $h$, from $\mathcal{H}$, and we need to choose the best one, or in other words, we need to find the values of these four parameters given the training set, to include all the positive examples and none of the negative examples. Note that if $x_1$ and $x_2$ are real-valued, there are infinitely many such $h$ for which this is satisfied, namely, for which the error, $E$, is 0, but given a future example somewhere close to the boundary between positive and negative examples, different candidate hypotheses may make different predictions. This is the problem of *generalization*—that is, how well our hypothesis will correctly classify future examples that are not part of the training set.

GENERALIZATION

MOST SPECIFIC HYPOTHESIS

One possibility is to find the *most specific hypothesis*, $S$, that is the tightest rectangle that includes all the positive examples and none of the negative examples (see figure 2.4). This gives us one hypothesis, $h = S$, as our induced class. Note that the actual class $C$ may be larger than $S$ but is never smaller. The *most general hypothesis*, $G$, is the largest rectangle we can draw that includes all the positive examples and none of the negative examples (figure 2.4). Any $h \in \mathcal{H}$ between $S$ and $G$ is a valid hypothesis with no error, said to be *consistent* with the training set, and such $h$ make up the *version space*. Given another training set, $S$, $G$, version space, the parameters and thus the learned hypothesis, $h$, can be different.

MOST GENERAL HYPOTHESIS

VERSION SPACE

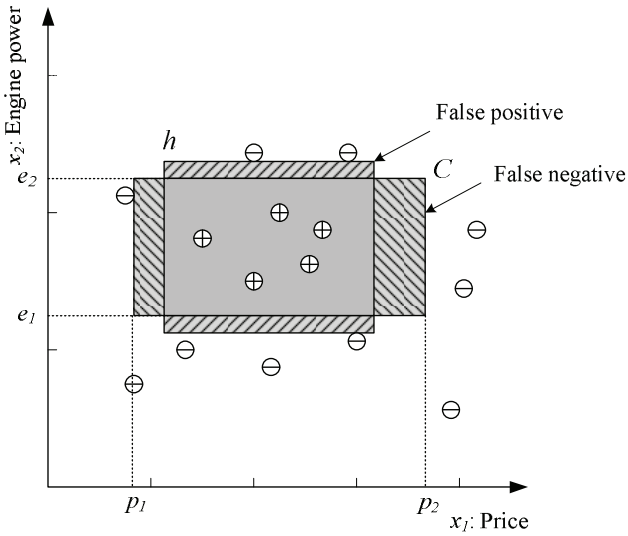


**Figure 2.3** $C$ is the actual class and $h$ is our induced hypothesis. The point where $C$ is 1 but $h$ is 0 is a false negative, and the point where $C$ is 0 but $h$ is 1 is a false positive. Other points—namely, true positives and true negatives—are correctly classified.

Actually, depending on $\mathcal{X}$ and $\mathcal{H}$, there may be several $S_i$ and $G_j$ which respectively make up the $S$-set and the $G$-set. Every member of the $S$-set is consistent with all the instances, and there are no consistent hypotheses that are more specific. Similarly, every member of the $G$-set is consistent with all the instances, and there are no consistent hypotheses that are more general. These two make up the boundary sets and any hypothesis between them is consistent and is part of the version space. There is an algorithm called candidate elimination that incrementally updates the $S$- and $G$-sets as it sees training instances one by one; see Mitchell 1997. We assume $\mathcal{X}$ is large enough that there is a unique $S$ and $G$.

Given $\mathcal{X}$, we can find $S$, or $G$, or any $h$ from the version space and use it as our hypothesis, $h$. It seems intuitive to choose $h$ halfway between $S$ and $G$; this is to increase the *margin*, which is the distance between the boundary and the instances closest to it (see figure 2.5). For our error function to have a minimum at $h$ with the maximum margin, we should

MARGIN

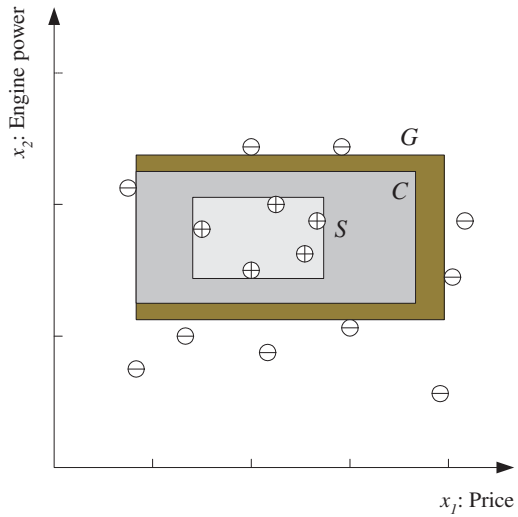


**Figure 2.4** $S$ is the most specific and $G$ is the most general hypothesis.

use an error (loss) function which not only checks whether an instance is on the correct side of the boundary but also how far away it is. That is, instead of $h(\boldsymbol{x})$ that returns 0/1, we need to have a hypothesis that returns a value which carries a measure of the distance to the boundary and we need to have a loss function which uses it, different from $1(\cdot)$ that checks for equality.

In some applications, a wrong decision may be very costly and in such a case, we can say that any instance that falls in between $S$ and $G$ is a case of *doubt*, which we cannot label with certainty due to lack of data. In such a case, the system *rejects* the instance and defers the decision to a human expert.

DOUBT

Here, we assume that $\mathcal{H}$ includes $C$; that is, there exists $h \in \mathcal{H}$, such that $E(h|\mathcal{X})$ is 0. Given a hypothesis class $\mathcal{H}$, it may be the case that we cannot learn $C$; that is, there exists no $h \in \mathcal{H}$ for which the error is 0. Thus, in any application, we need to make sure that $\mathcal{H}$ is flexible enough, or has enough "capacity," to learn $C$.

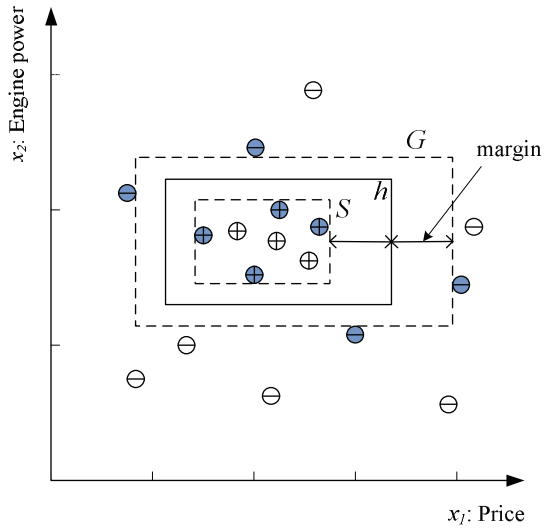


**Figure 2.5** We choose the hypothesis with the largest margin, for best separation. The shaded instances are those that define (or support) the margin; other instances can be removed without affecting $h$.

## 2.2 Vapnik-Chervonenkis Dimension

Let us say we have a dataset containing $N$ points. These $N$ points can be labeled in $2^N$ ways as positive and negative. Therefore, $2^N$ different learning problems can be defined by $N$ data points. If for any of these problems, we can find a hypothesis $h \in \mathcal{H}$ that separates the positive examples from the negative, then we say $\mathcal{H}$ *shatters* $N$ points. That is, any learning problem definable by $N$ examples can be learned with no error by a hypothesis drawn from $\mathcal{H}$. The maximum number of points that can be shattered by $\mathcal{H}$ is called the *Vapnik-Chervonenkis* (VC) *dimension* of $\mathcal{H}$, is denoted as $VC(\mathcal{H})$, and measures the *capacity* of $\mathcal{H}$.

VC DIMENSION

In figure 2.6, we see that an axis-aligned rectangle can shatter four points in two dimensions. Then $VC(\mathcal{H})$, when $\mathcal{H}$ is the hypothesis class of axis-aligned rectangles in two dimensions, is four. In calculating the VC dimension, it is enough that we find four points that can be shattered; it is not necessary that we be able to shatter *any* four points in two di-

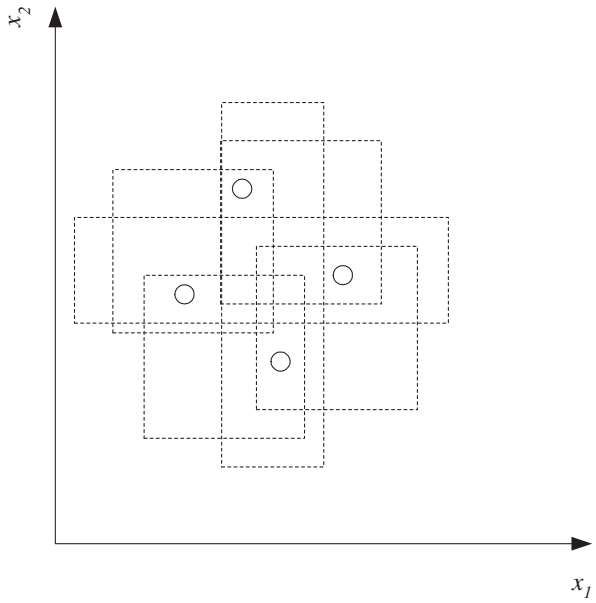


**Figure 2.6** An axis-aligned rectangle can shatter four points. Only rectangles covering two points are shown.

mensions. For example, four points placed on a line cannot be shattered by rectangles. However, we cannot place five points in two dimensions *anywhere* such that a rectangle can separate the positive and negative examples for all possible labelings.

VC dimension may seem pessimistic. It tells us that using a rectangle as our hypothesis class, we can learn only datasets containing four points and not more. A learning algorithm that can learn datasets of four points is not very useful. However, this is because the VC dimension is independent of the probability distribution from which instances are drawn. In real life, the world is smoothly changing, instances close by most of the time have the same labels, and we need not worry about *all possible labelings*. There are a lot of datasets containing many more data points than four that are learnable by our hypothesis class (figure 2.1). So even hypothesis classes with small VC dimensions are applicable and are preferred over those with large VC dimensions, for example, a lookup table that has infinite VC dimension.

## 2.3 Probably Approximately Correct Learning

Using the tightest rectangle, $S$, as our hypothesis, we would like to find how many examples we need. We would like our hypothesis to be approximately correct, namely, that the error probability be bounded by some value. We also would like to be confident in our hypothesis in that we want to know that our hypothesis will be correct most of the time (if not always); so we want to be probably correct as well (by a probability we can specify).

PAC LEARNING

In *probably approximately correct* (PAC) *learning*, given a class, $C$, and examples drawn from some unknown but fixed probability distribution, $p(x)$, we want to find the number of examples, $N$, such that with probability at least $1-\delta$, the hypothesis $h$ has error at most $\epsilon$, for arbitrary $\delta \leq 1/2$ and $\epsilon > 0$

$$P\{C\Delta h \leq \epsilon\} \geq 1-\delta$$

where $C\Delta h$ is the region of difference between $C$ and $h$.

In our case, because $S$ is the tightest possible rectangle, the error region between $C$ and $h = S$ is the sum of four rectangular strips (see figure 2.7). We would like to make sure that the probability of a positive example falling in here (and causing an error) is at most $\epsilon$. For any of these strips, if we can guarantee that the probability is upper bounded by $\epsilon/4$, the error is at most $4(\epsilon/4) = \epsilon$. Note that we count the overlaps in the corners twice, and the total actual error in this case is less than $4(\epsilon/4)$. The probability that a randomly drawn example misses this strip is $1-\epsilon/4$. The probability that all $N$ independent draws miss the strip is $(1-\epsilon/4)^N$, and the probability that all $N$ independent draws miss any of the four strips is at most $4(1-\epsilon/4)^N$, which we would like to be at most $\delta$. We have the inequality

$$(1-x) \leq \exp[-x]$$

So if we choose $N$ and $\delta$ such that we have

$$4\exp[-\epsilon N/4] \leq \delta$$

we can also write $4(1-\epsilon/4)^N \leq \delta$. Dividing both sides by 4, taking (natural) log and rearranging terms, we have

$$N \geq (4/\epsilon)\log(4/\delta) \tag{2.7}$$

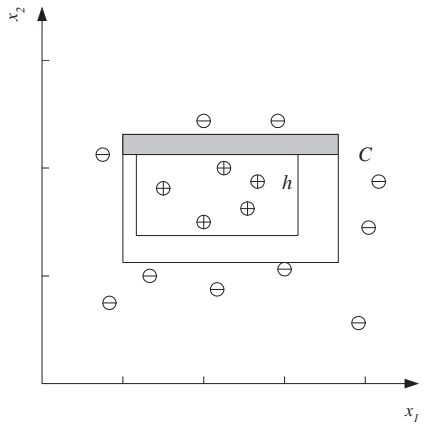


**Figure 2.7** The difference between $h$ and $C$ is the sum of four rectangular strips, one of which is shaded.

Therefore, provided that we take at least $(4/\epsilon)\log(4/\delta)$ independent examples from $C$ and use the tightest rectangle as our hypothesis $h$, with *confidence probability* at least $1-\delta$, a given point will be misclassified with *error probability* at most $\epsilon$. We can have arbitrary large confidence by decreasing $\delta$ and arbitrary small error by decreasing $\epsilon$, and we see in equation 2.7 that the number of examples is a slowly growing function of $1/\epsilon$ and $1/\delta$, linear and logarithmic, respectively.

## 2.4 Noise

NOISE *Noise* is any unwanted anomaly in the data and due to noise, the class may be more difficult to learn and zero error may be infeasible with a simple hypothesis class (see figure 2.8). There are several interpretations of noise:

- There may be imprecision in recording the input attributes, which may shift the data points in the input space.
- There may be errors in labeling the data points, which may relabel

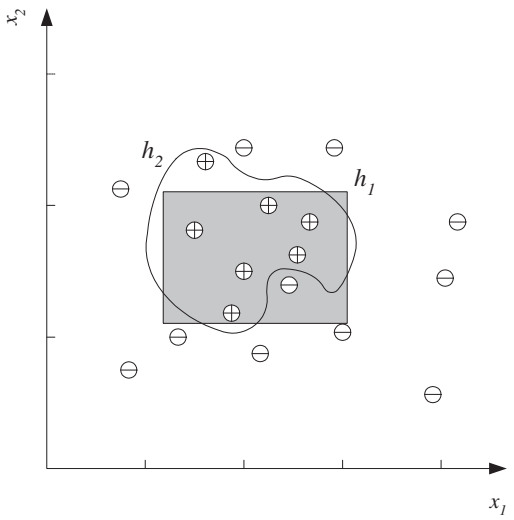


**Figure 2.8** When there is noise, there is not a simple boundary between the positive and negative instances, and zero misclassification error may not be possible with a simple hypothesis. A rectangle is a simple hypothesis with four parameters defining the corners. An arbitrary closed form can be drawn by piecewise functions with a larger number of control points.

positive instances as negative and vice versa. This is sometimes called *teacher noise*.

- There may be additional attributes, which we have not taken into account, that affect the label of an instance. Such attributes may be *hidden* or *latent* in that they may be unobservable. The effect of these neglected attributes is thus modeled as a random component and is included in "noise."

As can be seen in figure 2.8, when there is noise, there is not a simple boundary between the positive and negative instances and to separate them, one needs a complicated hypothesis that corresponds to a hypothesis class with larger capacity. A rectangle can be defined by four numbers, but to define a more complicated shape one needs a more complex model with a much larger number of parameters. With a complex model,

one can make a perfect fit to the data and attain zero error; see the wiggly shape in figure 2.8. Another possibility is to keep the model simple and allow some error; see the rectangle in figure 2.8.

Using the simple rectangle (unless its training error is much bigger) makes more sense because of the following:

1. It is a simple model to use. It is easy to check whether a point is inside or outside a rectangle and we can easily check, for a future data instance, whether it is a positive or a negative instance.

2. It is a simple model to train and has fewer parameters. It is easier to find the corner values of a rectangle than the control points of an arbitrary shape. With a small training set when the training instances differ a little bit, we expect the simpler model to change less than a complex model: A simple model is thus said to have less *variance*. On the other hand, a too simple model assumes more, is more rigid, and may fail if indeed the underlying class is not that simple: A simpler model has more *bias*. Finding the optimal model corresponds to minimizing both the bias and the variance.

3. It is a simple model to explain. A rectangle simply corresponds to defining intervals on the two attributes. By learning a simple model, we can extract information from the raw data given in the training set.

4. If indeed there is mislabeling or noise in input and the actual class is really a simple model like the rectangle, then the simple rectangle, because it has less variance and is less affected by single instances, will be a better discriminator than the wiggly shape, although the simple one may make slightly more errors on the training set. Given comparable empirical error, we say that a simple (but not too simple) model would generalize better than a complex model. This principle is known as *Occam's razor*, which states that *simpler explanations are more plausible* and any unnecessary complexity should be shaved off.

OCCAM'S RAZOR

## 2.5 Learning Multiple Classes

In our example of learning a family car, we have positive examples belonging to the class family car and the negative examples belonging to all other cars. This is a *two-class* problem. In the general case, we have $K$

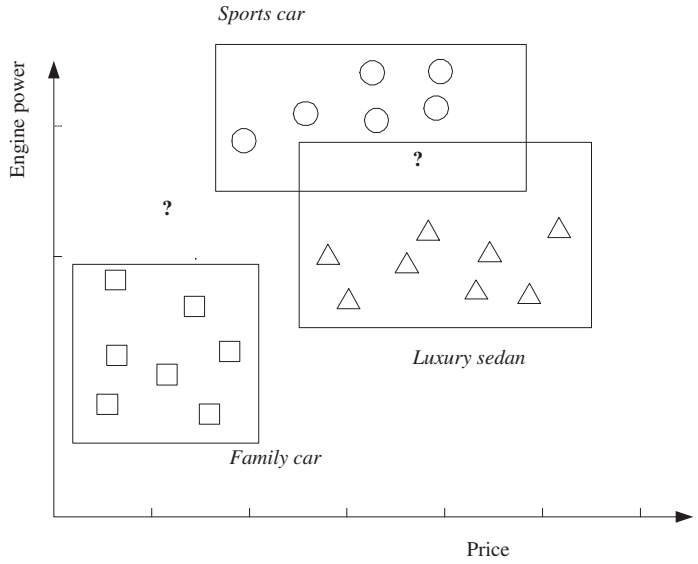


**Figure 2.9** There are three classes: family car, sports car, and luxury sedan. There are three hypotheses induced, each one covering the instances of one class and leaving outside the instances of the other two classes. '?' are reject regions where no, or more than one, class is chosen.

classes denoted as $C_i, i = 1, \ldots, K$, and an input instance belongs to one and exactly one of them. The training set is now of the form

$$\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{r}^t\}_{t=1}^N$$

where $\boldsymbol{r}$ has $K$ dimensions and

$$r_i^t = \begin{cases} 1 & \text{if } \boldsymbol{x}^t \in C_i \\ 0 & \text{if } \boldsymbol{x}^t \in C_j, j \neq i \end{cases} \tag{2.8}$$

An example is given in figure 2.9 with instances from three classes: family car, sports car, and luxury sedan.

In machine learning for classification, we would like to learn the boundary separating the instances of one class from the instances of all other classes. Thus we view a $K$-class classification problem as $K$ two-class problems. The training examples belonging to $C_i$ are the positive instances of hypothesis $h_i$ and the examples of all other classes are the

negative instances of $h_i$. Thus in a $K$-class problem, we have $K$ hypotheses to learn such that

$$h_i(\boldsymbol{x}^t) = \begin{cases} 1 & \text{if } \boldsymbol{x}^t \in C_i \\ 0 & \text{if } \boldsymbol{x}^t \in C_j, j \neq i \end{cases} \tag{2.9}$$

The total empirical error takes a sum over the predictions for all classes over all instances:

$$E(\{h_i\}_{i=1}^K | \mathcal{X}) = \sum_{t=1}^{N} \sum_{i=1}^{K} 1(h_i(\boldsymbol{x}^t) \neq r_i^t) \tag{2.10}$$

For a given $\boldsymbol{x}$, ideally only one of $h_i(\boldsymbol{x}), i = 1, \ldots, K$ is 1 and we can choose a class. But when no, or two or more, $h_i(\boldsymbol{x})$ is 1, we cannot choose a class, and this is the case of *doubt* and the classifier *rejects* such cases.

REJECT

In our example of learning a family car, we used only one hypothesis and only modeled the positive examples. Any negative example outside is not a family car. Alternatively, sometimes we may prefer to build two hypotheses, one for the positive and the other for the negative instances. This assumes a structure also for the negative instances that can be covered by another hypothesis. Separating family cars from sports cars is such a problem; each class has a structure of its own. The advantage is that if the input is a luxury sedan, we can have both hypotheses decide negative and reject the input.

If in a dataset, we expect to have all classes with similar distribution—shapes in the input space—then the same hypothesis class can be used for all classes. For example, in a handwritten digit recognition dataset, we would expect all digits to have similar distributions. But in a medical diagnosis dataset, for example, where we have two classes for sick and healthy people, we may have completely different distributions for the two classes; there may be multiple ways for a person to be sick, reflected differently in the inputs: All healthy people are alike; each sick person is sick in his or her own way.

## 2.6 Regression

In classification, given an input, the output that is generated is Boolean; it is a yes/no answer. When the output is a numeric value, what we would like to learn is not a class, $C(\boldsymbol{x}) \in \{0, 1\}$, but is a numeric function. In

machine learning, the function is not known but we have a training set of examples drawn from it

$$\mathcal{X} = \{\boldsymbol{x}^t, r^t\}_{t=1}^N$$

INTERPOLATION

where $r^t \in \Re$. If there is no noise, the task is *interpolation*. We would like to find the function $f(x)$ that passes through these points such that we have

$$r^t = f(\boldsymbol{x}^t)$$

In *polynomial interpolation*, given $N$ points, we find the $(N-1)$st degree polynomial that we can use to predict the output for any $\boldsymbol{x}$. This is called *extrapolation* if $\boldsymbol{x}$ is outside of the range of $\boldsymbol{x}^t$ in the training set. In time-series prediction, for example, we have data up to the present and we want to predict the value for the future. In *regression*, there is noise added to the output of the unknown function

EXTRAPOLATION

REGRESSION

$$r^t = f(\boldsymbol{x}^t) + \epsilon \tag{2.11}$$

where $f(\boldsymbol{x}) \in \Re$ is the unknown function and $\epsilon$ is random noise. The explanation for noise is that there are extra *hidden* variables that we cannot observe

$$r^t = f^*(\boldsymbol{x}^t, \boldsymbol{z}^t) \tag{2.12}$$

where $\boldsymbol{z}^t$ denote those hidden variables. We would like to approximate the output by our model $g(\boldsymbol{x})$. The empirical error on the training set $\mathcal{X}$ is

$$E(g|\mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N}[r^t - g(\boldsymbol{x}^t)]^2 \tag{2.13}$$

Because $r$ and $g(\boldsymbol{x})$ are numeric quantities, for example, $\in \Re$, there is an ordering defined on their values and we can define a *distance* between values, as the square of the difference, which gives us more information than equal/not equal, as used in classification. The square of the difference is one error (loss) function that can be used; another is the absolute value of the difference. We will see other examples in the coming chapters.

Our aim is to find $g(\cdot)$ that minimizes the empirical error. Again our approach is the same; we assume a hypothesis class for $g(\cdot)$ with a small set of parameters. If we assume that $g(\boldsymbol{x})$ is linear, we have

$$g(\boldsymbol{x}) = w_1 x_1 + \cdots + w_d x_d + w_0 = \sum_{j=1}^{d} w_j x_j + w_0 \tag{2.14}$$

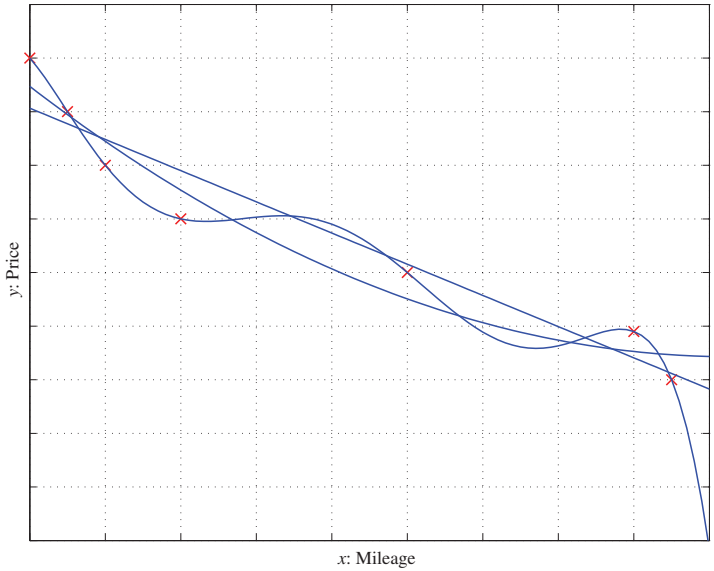


**Figure 2.10** Linear, second-order, and sixth-order polynomials are fitted to the same set of points. The highest order gives a perfect fit, but given this much data it is very unlikely that the real curve is so shaped. The second order seems better than the linear fit in capturing the trend in the training data.

Let us now go back to our example in section 1.2.3 where we estimated the price of a used car. There we used a single input linear model

$$g(x) = w_1 x + w_0 \tag{2.15}$$

where $w_1$ and $w_0$ are the parameters to learn from data. The $w_1$ and $w_0$ values should minimize

$$E(w_1, w_0|\mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N}[r^t - (w_1 x^t + w_0)]^2 \tag{2.16}$$

Its minimum point can be calculated by taking the partial derivatives of $E$ with respect to $w_1$ and $w_0$, setting them equal to 0, and solving for the two unknowns:

$$\begin{aligned} w_1 &= \frac{\sum_t x^t r^t - \overline{x}\overline{r}N}{\sum_t (x^t)^2 - N\overline{x}^2} \\ w_0 &= \overline{r} - w_1\overline{x} \end{aligned} \tag{2.17}$$

**Table 2.1** With two inputs, there are four possible cases and sixteen possible Boolean functions

| $x_1$ | $x_2$ | $h_1$ | $h_2$ | $h_3$ | $h_4$ | $h_5$ | $h_6$ | $h_7$ | $h_8$ | $h_9$ | $h_{10}$ | $h_{11}$ | $h_{12}$ | $h_{13}$ | $h_{14}$ | $h_{15}$ | $h_{16}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

where $\overline{x} = \sum_t x^t / N$ and $\overline{r} = \sum_t r^t / N$. The line found is shown in figure 1.2.

If the linear model is too simple, it is too constrained and incurs a large approximation error, and in such a case, the output may be taken as a higher-order function of the input—for example, quadratic

$$g(x) = w_2 x^2 + w_1 x + w_0 \tag{2.18}$$

where similarly we have an analytical solution for the parameters. When the order of the polynomial is increased, the error on the training data decreases. But a high-order polynomial follows individual examples closely, instead of capturing the general trend; see the sixth-order polynomial in figure 2.10. This implies that Occam's razor also applies in the case of regression and we should be careful when fine-tuning the model complexity to match it with the complexity of the function underlying the data.

## 2.7 Model Selection and Generalization

Let us start with the case of learning a Boolean function from examples. In a Boolean function, all inputs and the output are binary. There are $2^d$ possible ways to write $d$ binary values and therefore, with $d$ inputs, the training set has at most $2^d$ examples. As shown in table 2.1, each of these can be labeled as 0 or 1, and therefore, there are $2^{2^d}$ possible Boolean functions of $d$ inputs.

Each distinct training example removes half the hypotheses, namely, those whose guesses are wrong. For example, let us say we have $x_1 = 0$, $x_2 = 1$ and the output is 0; this removes $h_5, h_6, h_7, h_8, h_{13}, h_{14}, h_{15}, h_{16}$. This is one way to interpret learning: We start with all possible hypotheses and as we see more training examples, we remove those hypotheses

that are not consistent with the training data. In the case of a Boolean function, to end up with a single hypothesis we need to see *all* $2^d$ training examples. If the training set we are given contains only a small subset of all possible instances, as it generally does—that is, if we know what the output should be for only a small percentage of the cases—the solution is not unique. After seeing $N$ example cases, there remain $2^{2^d-N}$ possible functions. This is an example of an *ill-posed problem* where the data by itself is not sufficient to find a unique solution.

ILL-POSED PROBLEM

The same problem also exists in other learning applications, in classification, and in regression. As we see more training examples, we know more about the underlying function, and we carve out more hypotheses that are inconsistent from the hypothesis class, but we still are left with many consistent hypotheses.

So because learning is ill-posed, and data by itself is not sufficient to find the solution, we should make some extra assumptions to have a unique solution with the data we have. The set of assumptions we make to have learning possible is called the *inductive bias* of the learning algorithm. One way we introduce inductive bias is when we assume a hypothesis class $\mathcal{H}$. In learning the class of family cars, there are infinitely many ways of separating the positive examples from the negative examples. Assuming the shape of a rectangle is one inductive bias, and then the rectangle with the largest margin for example, is another inductive bias. In linear regression, assuming a linear function is an inductive bias, and among all lines, choosing the one that minimizes squared error is another inductive bias.

INDUCTIVE BIAS

But we know that each hypothesis class has a certain capacity and can learn only certain functions. The class of functions that can be learned can be extended by using a hypothesis class with larger capacity, containing more complex hypotheses. For example, the hypothesis class that is a union of two rectangles has higher capacity, but its hypotheses are more complex. Similarly in regression, as we increase the order of the polynomial, the capacity and complexity increase. The question now is to decide where to stop.

Thus learning is not possible without inductive bias, and now the question is how to choose the right bias. This is called *model selection*, which is choosing between possible $\mathcal{H}$. In answering this question, we should remember that the aim of machine learning is rarely to replicate the training data but the prediction for new cases. That is we would like to be able to generate the right output for an input instance outside the training set,

MODEL SELECTION

one for which the correct output is not given in the training set. How well a model trained on the training set predicts the right output for new instances is called *generalization*.

GENERALIZATION

For best generalization, we should match the complexity of the hypothesis class $\mathcal{H}$ with the complexity of the function underlying the data. If $\mathcal{H}$ is less complex than the function, we have *underfitting*, for example, when trying to fit a line to data sampled from a third-order polynomial. In such a case, as we increase the complexity, the training error decreases. But if we have $\mathcal{H}$ that is too complex, the data is not enough to constrain it and we may end up with a bad hypothesis, $h \in \mathcal{H}$, for example, when fitting two rectangles to data sampled from one rectangle. Or if there is noise, an overcomplex hypothesis may learn not only the underlying function but also the noise in the data and may make a bad fit, for example, when fitting a sixth-order polynomial to noisy data sampled from a third-order polynomial. This is called *overfitting*. In such a case, having more training data helps but only up to a certain point. Given a training set and $\mathcal{H}$, we can find $h \in \mathcal{H}$ that has the minimum training error but if $\mathcal{H}$ is not chosen well, no matter which $h \in \mathcal{H}$ we pick, we will not have good generalization.

UNDERFITTING

OVERFITTING

TRIPLE TRADE-OFF

We can summarize our discussion citing the *triple trade-off* (Dietterich 2003). In all learning algorithms that are trained from example data, there is a trade-off between three factors:

- the complexity of the hypothesis we fit to data, namely, the capacity of the hypothesis class,
- the amount of training data, and
- the generalization error on new examples.

As the amount of training data increases, the generalization error decreases. As the complexity of the model class $\mathcal{H}$ increases, the generalization error decreases first and then starts to increase. The generalization error of an overcomplex $\mathcal{H}$ can be kept in check by increasing the amount of training data but only up to a point. If the data is sampled from a line and if we are fitting a higher-order polynomial, the fit will be constrained to lie close to the line if there is training data in the vicinity; where it has not been trained, a high-order polynomial may behave erratically.

We can measure the generalization ability of a hypothesis, namely, the quality of its inductive bias, if we have access to data outside the training

set. We simulate this by dividing the dataset we have into two parts. We use one part for training (i.e., to fit a hypothesis), and the remaining part is called the *validation set* and is used to test the generalization ability. That is, given a set of possible hypothesis classes $\mathcal{H}_i$, for each we fit the best $h_i \in \mathcal{H}_i$ on the training set. Then, assuming large enough training and validation sets, the hypothesis that is the most accurate on the validation set is the best one (the one that has the best inductive bias). This process is called *cross-validation*. So, for example, to find the right order in polynomial regression, given a number of candidate polynomials of different orders where polynomials of different orders correspond to $\mathcal{H}_i$, for each order, we find the coefficients on the training set, calculate their errors on the validation set, and take the one that has the least validation error as the best polynomial.

VALIDATION SET

CROSS-VALIDATION

Note that if we then need to report the error to give an idea about the expected error of our best model, we should not use the validation error. We have used the validation set to choose the best model, and it has effectively become a part of the training set. We need a third set, a *test set*, sometimes also called the *publication set*, containing examples not used in training or validation. An analogy from our lives is when we are taking a course: the example problems that the instructor solves in class while teaching a subject form the training set; exam questions are the validation set; and the problems we solve in our later, professional life are the test set.

TEST SET

We cannot keep on using the same training/validation split either, because after having been used once, the validation set effectively becomes part of training data. This will be like an instructor who uses the same exam questions every year; a smart student will figure out not to bother with the lectures and will only memorize the answers to those questions.

We should always remember that the training data we use is a random sample, that is, for the same application, if we collect data once more, we will get a slightly different dataset, the fitted $h$ will be slightly different and will have a slightly different validation error. Or if we have a fixed set which we divide for training, validation, and test, we will have different errors depending on how we do the division. These slight differences in error will allow us to estimate how large differences should be to be considered *significant* and not due to chance. That is, in choosing between two hypothesis classes $\mathcal{H}_i$ and $\mathcal{H}_j$, we will use them both multiple times on a number of training and validation sets and check if the difference between average errors of $h_i$ and $h_j$ is larger than the average difference

between multiple $h_i$. In chapter 19, we discuss how to design machine learning experiments using limited data to best answer our questions—for example, which is the best hypothesis class?—and how to analyze the results of these experiments so that we can achieve statistically significant conclusions minimally affected by random chance.

## 2.8 Dimensions of a Supervised Machine Learning Algorithm

Let us now recapitulate and generalize. We have a sample

$$\mathcal{X} = \{x^t, r^t\}_{t=1}^N \tag{2.19}$$

INDEPENDENT AND IDENTICALLY DISTRIBUTED (IID)

The sample is *independent and identically distributed* (*iid*); the ordering is not important and all instances are drawn from the same joint distribution $p(x, r)$. $t$ indexes one of the $N$ instances, $x^t$ is the arbitrary dimensional input, and $r^t$ is the associated desired output. $r^t$ is 0/1 for two-class learning, is a $K$-dimensional binary vector (where exactly one of the dimensions is 1 and all others 0) for $(K > 2)$-class classification, and is a real value in regression.

The aim is to build a good and useful approximation to $r^t$ using the model $g(x^t|\theta)$. In doing this, there are three decisions we must make:

1. *Model* we use in learning, denoted as

   $$g(x|\theta)$$

   where $g(\cdot)$ is the model, $x$ is the input, and $\theta$ are the parameters.

   $g(\cdot)$ defines the hypothesis class $\mathcal{H}$, and a particular value of $\theta$ instantiates one hypothesis $h \in \mathcal{H}$. For example, in class learning, we have taken a rectangle as our model whose four coordinates make up $\theta$; in linear regression, the model is the linear function of the input whose slope and intercept are the parameters learned from the data. The model (inductive bias), or $\mathcal{H}$, is fixed by the machine learning system designer based on his or her knowledge of the application and the hypothesis $h$ is chosen (parameters are tuned) by a learning algorithm using the training set, sampled from $p(x, r)$.

2. *Loss function*, $L(\cdot)$, to compute the difference between the desired output, $r^t$, and our approximation to it, $g(x^t|\theta)$, given the current value

of the parameters, $\theta$. The *approximation error*, or *loss*, is the sum of losses over the individual instances

$$E(\theta|\mathcal{X}) = \sum_t L(r^t, g(x^t|\theta)) \tag{2.20}$$

In class learning where outputs are 0/1, $L(\cdot)$ checks for equality or not; in regression, because the output is a numeric value, we have ordering information for distance and one possibility is to use the square of the difference.

3. *Optimization procedure* to find $\theta^*$ that minimizes the total error

$$\theta^* = \arg\min_\theta E(\theta|\mathcal{X}) \tag{2.21}$$

where arg min returns the argument that minimizes. In polynomial regression, we can solve analytically for the optimum, but this is not always the case. With other models and error functions, the complexity of the optimization problem becomes important. We are especially interested in whether it has a single minimum corresponding to a globally optimal solution, or whether there are multiple minima corresponding to locally optimal solutions.

For this setting to work well, the following conditions should be satisfied: First, the hypothesis class of $g(\cdot)$ should be large enough, that is, have enough capacity, to include the unknown function that generated the data that is represented in $\mathcal{X}$ in a noisy form. Second, there should be enough training data to allow us to pinpoint the correct (or a good enough) hypothesis from the hypothesis class. Third, we should have a good optimization method that finds the correct hypothesis given the training data.

Different machine learning algorithms differ either in the models they assume (their hypothesis class/inductive bias), the loss measures they employ, or the optimization procedure they use. We will see many examples in the coming chapters.

## 2.9 Notes

Mitchell proposed version spaces and the candidate elimination algorithm to incrementally build $S$ and $G$ as instances are given one by one;

see Mitchell 1997 for a recent review. The rectangle-learning is from exercise 2.4 of Mitchell 1997. Hirsh (1990) discusses how version spaces can handle the case when instances are perturbed by small amount of noise.

In one of the earliest works on machine learning, Winston (1975) proposed the idea of a "near miss." A near miss is a negative example that is very much like a positive example. In our terminology, we see that a near miss would be an instance that falls in the gray area between $S$ and $G$, an instance which would affect the margin, and would hence be more useful for learning, than an ordinary positive or negative example. The instances that are close to the boundary are the ones that define it (or support it); those which are inside and are surrounded by many instances with the same label can be removed without affecting the boundary.

Related to this idea is *active learning* where the learning algorithm can generate instances itself and ask for them to be labeled, instead of passively being given them (Angluin 1988) (see exercise 5).

VC dimension was proposed by Vapnik and Chervonenkis in the early 1970s. A recent source is Vapnik 1995 where he writes, "Nothing is more practical than a good theory" (p. x), which is as true in machine learning as in any other branch of science. You should not rush to the computer; you can save yourself from hours of useless programming by some thinking, a notebook, and a pencil—you may also need an eraser.

The PAC model was proposed by Valiant (1984). The PAC analysis of learning a rectangle is from Blumer et al. 1989. A good textbook on computational learning theory covering PAC learning and VC dimension is Kearns and Vazirani 1994.

The definition of the optimization problem solved for model fitting has been getting very important in recent years. Once quite content with local descent methods that converge to the nearest good solution starting from some random initial state, nowadays we are, for example, interested in showing that the problem is convex—there is a single, global solution (Boyd and Vandenberghe 2004). As dataset sizes grow and models get more complex, we are also, for example, interested in how fast the optimization procedure converges to a solution.

## 2.10 Exercises

1. Let us say our hypothesis class is a circle instead of a rectangle. What are the parameters? How can the parameters of a circle hypothesis be calculated in

such a case? What if it is an ellipse? Why does it make more sense to use an ellipse instead of a circle?

SOLUTION: In the case of a circle, the parameters are the center and the radius (see figure 2.11). We then need to find $S$ and $G$ where $S$ is the tightest circle that includes all the positive examples and $G$ is the largest circle that includes all the positive examples and no negative example; any circle between them is a consistent hypothesis.

It makes more sense to use an ellipse because the two axes need not have the same scale and an ellipse has two separate parameters for the widths in the two axes rather than a single radius. Actually, price and engine power are positively correlated; the price of a car tends to increase as its engine power increases, and hence it makes more sense to use an oblique ellipse—we will see such models in chapter 5.

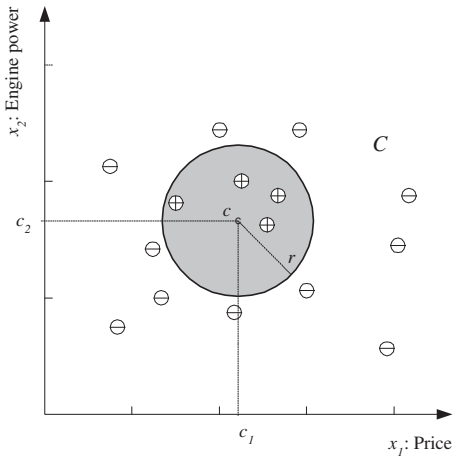


**Figure 2.11** Hypothesis class is a circle with two parameters, the coordinates of its center and its radius.

2. Imagine our hypothesis is not one rectangle but a union of two (or $m > 1$) rectangles. What is the advantage of such a hypothesis class? Show that any class can be represented by such a hypothesis class with large enough $m$.

   SOLUTION: In the case when there is a single rectangle, all the positive instances should form one single group; with two rectangles, for example (see figure 2.12), the positive instances can form two, possibly disjoint clusters in the input space. Note that each rectangle corresponds to a conjunction on the two input attributes, and having multiple rectangles corresponds to a disjunction. Any logical formula can be written as a disjunction of conjunctions.

In the worst case ($m = N$), we have a separate rectangle for each positive instance.

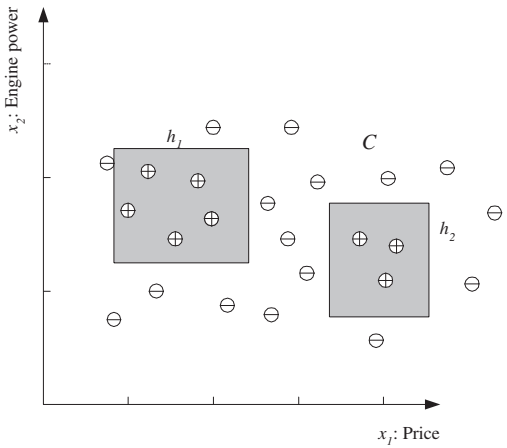


**Figure 2.12** Hypothesis class is a union of two rectangles.

3. In many applications, wrong decisions—namely, false positives and false negatives—have a monetary cost, and these two costs may be different. What is the relationship between the positioning of $h$ between $S$ and $G$ and the relative costs of these?

   SOLUTION: We can see that $S$ makes no false positives, but only false negatives; similarly, $G$ makes no false negatives, only false positives. So if false positives and false negatives are equally bad, we want our $h$ to be halfway; if false positives are costlier, we want $h$ to be closer to $S$; if false negatives are costlier, $h$ should be closer to $G$.

4. The complexity of most learning algorithms is a function of the training set. Can you propose a filtering algorithm that finds redundant instances?

   SOLUTION: The instances that affect the hypothesis are those that are in the vicinity of instances with a different label. A positive instance that is surrounded on all sides by many positive instances is not needed, nor is a negative instance surrounded by many negative instances. We discuss such *neighbor-based* methods in chapter 8.

5. If we have a supervisor who can provide us with the label for any $\boldsymbol{x}$, where should we choose $\boldsymbol{x}$ to learn with fewer queries?

   SOLUTION: The region of ambiguity is between $S$ and $G$. It would be best to be given queries there, so that we can make this region of doubt smaller. If a

given instance there turns out to be positive, this means we can make $S$ larger up to that instance; if it is negative, this means we can shrink $G$ down until there.

6. In equation 2.13, we summed up the squares of the differences between the actual value and the estimated value. This error function is the one most frequently used, but it is one of several possible error functions. Because it sums up the squares of the differences, it is not robust to outliers. What would be a better error function to implement *robust regression*?

7. Derive equation 2.17.

8. Assume our hypothesis class is the set of lines, and we use a line to separate the positive and negative examples, instead of bounding the positive examples as in a rectangle, leaving the negatives outside (see figure 2.13). Show that the VC dimension of a line is 3.

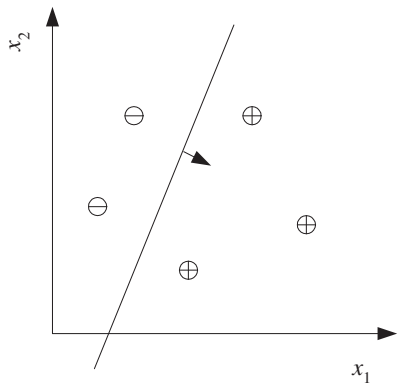


**Figure 2.13** A line separating positive and negative instances.

9. Show that the VC dimension of the triangle hypothesis class is 7 in two dimensions. (Hint: For best separation, it is best to place the seven points equidistant on a circle.)

10. Assume as in exercise 8 that our hypothesis class is the set of lines. Write down an error function that not only minimizes the number of misclassifications but also maximizes the margin.

11. One source of noise is error in the labels. Can you propose a method to find data points that are highly likely to be mislabeled?

## 2.11 References

Angluin, D. 1988. "Queries and Concept Learning." *Machine Learning* 2:319–342.

Blumer, A., A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. 1989. "Learnability and the Vapnik-Chervonenkis Dimension." *Journal of the ACM* 36:929–965.

Boyd, S., and L. Vandenberge. 2004. *Convex Optimization*. Cambridge, UK: Cambridge University Press.

Dietterich, T. G. 2003. "Machine Learning." In *Nature Encyclopedia of Cognitive Science*. London: Macmillan.

Hirsh, H. 1990. *Incremental Version Space Merging: A General Framework for Concept Learning*. Boston: Kluwer.

Kearns, M. J., and U. V. Vazirani. 1994. *An Introduction to Computational Learning Theory*. Cambridge, MA: MIT Press.

Mitchell, T. 1997. *Machine Learning*. New York: McGraw-Hill.

Valiant, L. 1984. "A Theory of the Learnable." *Communications of the ACM* 27:1134–1142.

Vapnik, V. N. 1995. *The Nature of Statistical Learning Theory*. New York: Springer.

Winston, P. H. 1975. "Learning Structural Descriptions from Examples." In *The Psychology of Computer Vision*, ed. P. H. Winston, 157–209. New York: McGraw-Hill.

# 3 *Bayesian Decision Theory*

*We discuss probability theory as the framework for making decisions under uncertainty. In classification, Bayes' rule is used to calculate the probabilities of the classes. We generalize to discuss how we can make rational decisions among multiple actions to minimize expected risk. We also discuss learning association rules from data.*

## 3.1 Introduction

PROGRAMMING COMPUTERS to make inference from data is a cross between statistics and computer science, where statisticians provide the mathematical framework of making inference from data and computer scientists work on the efficient implementation of the inference methods.

Data comes from a process that is not completely known. This lack of knowledge is indicated by modeling the process as a random process. Maybe the process is actually deterministic, but because we do not have access to complete knowledge about it, we model it as random and use probability theory to analyze it. At this point, it may be a good idea to jump to the appendix and review basic probability theory before continuing with this chapter.

Tossing a coin is a random process because we cannot predict at any toss whether the outcome will be heads or tails—that is why we toss coins, or buy lottery tickets, or get insurance. We can only talk about the probability that the outcome of the next toss will be heads or tails. It may be argued that if we have access to extra knowledge such as the exact composition of the coin, its initial position, the force and its direction that is applied to the coin when tossing it, where and how it is caught, and so forth, the exact outcome of the toss can be predicted.

UNOBSERVABLE VARIABLES
OBSERVABLE VARIABLE

The extra pieces of knowledge that we do not have access to are named the *unobservable variables*. In the coin tossing example, the only *observable variable* is the outcome of the toss. Denoting the unobservables by $\mathbf{z}$ and the observable as $x$, in reality we have

$$x = f(\mathbf{z})$$

where $f(\cdot)$ is the deterministic function that defines the outcome from the unobservable pieces of knowledge. Because we cannot model the process this way, we define the outcome $X$ as a random variable drawn from a probability distribution $P(X = x)$ that specifies the process.

The outcome of tossing a coin is heads or tails, and we define a random variable that takes one of two values. Let us say $X = 1$ denotes that the outcome of a toss is heads and $X = 0$ denotes tails. Such $X$ are Bernoulli-distributed where the parameter of the distribution $p_o$ is the probability that the outcome is heads:

$$P(X = 1) = p_o \text{ and } P(X = 0) = 1 - P(X = 1) = 1 - p_o$$

Assume that we are asked to predict the outcome of the next toss. If we know $p_o$, our prediction will be heads if $p_o > 0.5$ and tails otherwise. This is because if we choose the more probable case, the probability of error, which is 1 minus the probability of our choice, will be minimum. If this is a fair coin with $p_o = 0.5$, we have no better means of prediction than choosing heads all the time or tossing a fair coin ourselves!

SAMPLE

If we do not know $P(X)$ and want to estimate this from a given sample, then we are in the realm of statistics. We have a *sample*, $\mathcal{X}$, containing examples drawn from the probability distribution of the observables $x^t$, denoted as $p(x)$. The aim is to build an approximator to it, $\hat{p}(x)$, using the sample $\mathcal{X}$.

In the coin tossing example, the sample contains the outcomes of the past $N$ tosses. Then using $\mathcal{X}$, we can estimate $p_o$, which is the parameter that uniquely specifies the distribution. Our estimate of $p_o$ is

$$\hat{p}_o = \frac{\#\{\text{tosses with outcome heads}\}}{\#\{\text{tosses}\}}$$

Numerically using the random variables, $x^t$ is 1 if the outcome of toss $t$ is heads and 0 otherwise. Given the sample {heads, heads, heads, tails, heads, tails, tails, heads, heads}, we have $\mathcal{X} = \{1, 1, 1, 0, 1, 0, 0, 1, 1\}$ and the estimate is

$$\hat{p}_o = \frac{\sum_{t=1}^{N} x^t}{N} = \frac{6}{9}$$

## 3.2 Classification

We discussed credit scoring in section 1.2.2, where we saw that in a bank, according to their past transactions, some customers are low-risk in that they paid back their loans and the bank profited from them and other customers are high-risk in that they defaulted. Analyzing this data, we would like to learn the class "high-risk customer" so that in the future, when there is a new application for a loan, we can check whether that person obeys the class description or not and thus accept or reject the application. Using our knowledge of the application, let us say that we decide that there are two pieces of information that are observable. We observe them because we have reason to believe that they give us an idea about the credibility of a customer. Let us say, for example, we observe customer's yearly income and savings, which we represent by two random variables $X_1$ and $X_2$.

It may again be claimed that if we had access to other pieces of knowledge such as the state of economy in full detail and full knowledge about the customer, his or her intention, moral codes, and so forth, whether someone is a low-risk or high-risk customer could have been deterministically calculated. But these are nonobservables and with what we can observe, the credibility of a customer is denoted by a Bernoulli random variable $C$ conditioned on the observables $\mathbf{X} = [X_1, X_2]^T$ where $C = 1$ indicates a high-risk customer and $C = 0$ indicates a low-risk customer. Thus if we know $P(C|X_1, X_2)$, when a new application arrives with $X_1 = x_1$ and $X_2 = x_2$, we can

$$\text{choose} \begin{cases} C = 1 & \text{if } P(C = 1|x_1, x_2) > 0.5 \\ C = 0 & \text{otherwise} \end{cases}$$

or equivalently

$$\text{choose} \begin{cases} C = 1 & \text{if } P(C = 1|x_1, x_2) > P(C = 0|x_1, x_2) \\ C = 0 & \text{otherwise} \end{cases} \tag{3.1}$$

The probability of error is $1 - \max(P(C = 1|x_1, x_2), P(C = 0|x_1, x_2))$. This example is similar to the coin tossing example except that here, the Bernoulli random variable $C$ is conditioned on two other observable variables. Let us denote by $\mathbf{x}$ the vector of observed variables, $\mathbf{x} = [x_1, x_2]^T$.

BAYES' RULE

The problem then is to be able to calculate $P(C|\mathbf{x})$. Using *Bayes' rule*, it can be written as

$$P(C|\mathbf{x}) = \frac{P(C)p(\mathbf{x}|C)}{p(\mathbf{x})} \tag{3.2}$$

PRIOR PROBABILITY

$P(C = 1)$ is called the *prior probability* that $C$ takes the value 1, which in our example corresponds to the probability that a customer is high-risk, regardless of the $\boldsymbol{x}$ value—It is the proportion of high-risk customers in our customer base. It is called the prior probability because it is the knowledge we have as to the value of $C$ *before* looking at the observables $\boldsymbol{x}$, satisfying

$$P(C = 0) + P(C = 1) = 1$$

CLASS LIKELIHOOD

$p(\boldsymbol{x}|C)$ is called the *class likelihood* and is the conditional probability that an event belonging to $C$ has the associated observation value $\boldsymbol{x}$. In our case, $p(x_1, x_2|C = 1)$ is the probability that a high-risk customer has his or her $X_1 = x_1$ and $X_2 = x_2$. It is what the data tells us regarding the class.

EVIDENCE

$p(\boldsymbol{x})$, the *evidence*, is the marginal probability that an observation $\boldsymbol{x}$ is seen, regardless of whether it is a positive or negative example.

$$p(\boldsymbol{x}) = \sum_{C} p(\boldsymbol{x}, C) = p(\boldsymbol{x}|C = 1)P(C = 1) + p(\boldsymbol{x}|C = 0)P(C = 0) \tag{3.3}$$

POSTERIOR PROBABILITY

Combining the prior and what the data tells us using Bayes' rule, we calculate the *posterior probability* of the concept, $P(C|\boldsymbol{x})$, *after* having seen the observation, $\boldsymbol{x}$.

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

Because of normalization by the evidence, the posteriors sum up to 1:

$$P(C = 0|\boldsymbol{x}) + P(C = 1|\boldsymbol{x}) = 1$$

Once we have the posteriors, we decide by using equation 3.1. For now, we assume that we know the prior and likelihoods; in later chapters, we discuss how to estimate $P(C)$ and $p(\boldsymbol{x}|C)$ from a given training sample.

In the general case, we have $K$ mutually exclusive and exhaustive classes; $C_i, i = 1, \ldots, K$; for example, in optical digit recognition, the input is a bitmap image and there are ten classes. We have the prior probabilities satisfying

$$P(C_i) \geq 0 \text{ and } \sum_{i=1}^{K} P(C_i) = 1 \tag{3.4}$$

$p(\boldsymbol{x}|C_i)$ is the probability of seeing $\boldsymbol{x}$ as the input when it is known to

belong to class $C_i$. The posterior probability of class $C_i$ can be calculated as

$$P(C_i|\mathbf{x}) = \frac{p(\mathbf{x}|C_i)P(C_i)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|C_i)P(C_i)}{\sum_{k=1}^{K} p(\mathbf{x}|C_k)P(C_k)} \tag{3.5}$$

BAYES' CLASSIFIER

and for minimum error, the *Bayes' classifier* chooses the class with the highest posterior probability; that is, we

$$\text{choose } C_i \text{ if } P(C_i|\mathbf{x}) = \max_k P(C_k|\mathbf{x}) \tag{3.6}$$

## 3.3 Losses and Risks

It may be the case that decisions are not equally good or costly. A financial institution when making a decision for a loan applicant should take into account the potential gain and loss as well. An accepted low-risk applicant increases profit, while a rejected high-risk applicant decreases loss. The loss for a high-risk applicant erroneously accepted may be different from the potential gain for an erroneously rejected low-risk applicant. The situation is much more critical and far from symmetry in other domains like medical diagnosis or earthquake prediction.

LOSS FUNCTION
EXPECTED RISK

Let us define action $\alpha_i$ as the decision to assign the input to class $C_i$ and $\lambda_{ik}$ as the *loss* incurred for taking action $\alpha_i$ when the input actually belongs to $C_k$. Then the *expected risk* for taking action $\alpha_i$ is

$$R(\alpha_i|\mathbf{x}) = \sum_{k=1}^{K} \lambda_{ik} P(C_k|\mathbf{x}) \tag{3.7}$$

and we choose the action with minimum risk:

$$\text{Choose } \alpha_i \text{ if } R(\alpha_i|\mathbf{x}) = \min_k R(\alpha_k|\mathbf{x}) \tag{3.8}$$

0/1 LOSS

Let us define $K$ actions $\alpha_i, i = 1, \ldots, K$, where $\alpha_i$ is the action of assigning $\mathbf{x}$ to $C_i$. In the special case of the *0/1 loss* where

$$\lambda_{ik} = \begin{cases} 0 & \text{if } i = k \\ 1 & \text{if } i \neq k \end{cases} \tag{3.9}$$

all correct decisions have no loss and all errors are equally costly. The risk of taking action $\alpha_i$ is

$$R(\alpha_i|\mathbf{x}) = \sum_{k=1}^{K} \lambda_{ik} P(C_k|\mathbf{x})$$

$$\begin{aligned} &= \sum_{k \neq i} P(C_k|\boldsymbol{x}) \\ &= 1 - P(C_i|\boldsymbol{x}) \end{aligned}$$

because $\sum_k P(C_k|\boldsymbol{x}) = 1$. Thus to minimize risk, we choose the most probable class. In later chapters, for simplicity, we will always assume this case and choose the class with the highest posterior, but note that this is indeed a special case and rarely do applications have a symmetric, 0/1 loss. In the general case, it is a simple postprocessing to go from posteriors to risks and to take the action to minimize the risk.

In some applications, wrong decisions—namely, misclassifications—may have very high cost, and it is generally required that a more complex—for example, manual—decision is made if the automatic system has low certainty of its decision. For example, if we are using an optical digit recognizer to read postal codes on envelopes, wrongly recognizing the code causes the envelope to be sent to a wrong destination.

REJECT In such a case, we define an additional action of *reject* or *doubt*, $\alpha_{K+1}$, with $\alpha_i, i = 1, \ldots, K$, being the usual actions of deciding on classes $C_i, i = 1, \ldots, K$ (Duda, Hart, and Stork 2001).

A possible loss function is

$$\lambda_{ik} = \begin{cases} 0 & \text{if } i = k \\ \lambda & \text{if } i = K+1 \\ 1 & \text{otherwise} \end{cases} \tag{3.10}$$

where $0 < \lambda < 1$ is the loss incurred for choosing the $(K+1)$st action of reject. Then the risk of reject is

$$R(\alpha_{K+1}|\boldsymbol{x}) = \sum_{k=1}^{K} \lambda P(C_k|\boldsymbol{x}) = \lambda \tag{3.11}$$

and the risk of choosing class $C_i$ is

$$R(\alpha_i|\boldsymbol{x}) = \sum_{k \neq i} P(C_k|\boldsymbol{x}) = 1 - P(C_i|\boldsymbol{x}) \tag{3.12}$$

The optimal decision rule is to

$$\begin{aligned} &\text{choose } C_i && \text{if } R(\alpha_i|\boldsymbol{x}) < R(\alpha_k|\boldsymbol{x}) \text{ for all } k \neq i \text{ and} \\ &&& R(\alpha_i|\boldsymbol{x}) < R(\alpha_{K+1}|\boldsymbol{x}) \\ &\text{reject} && \text{if } R(\alpha_{K+1}|\boldsymbol{x}) < R(\alpha_i|\boldsymbol{x}), i = 1, \ldots, K \end{aligned} \tag{3.13}$$

Given the loss function of equation 3.10, this simplifies to

$$\begin{array}{lll} \text{choose } C_i & \text{if } P(C_i|\boldsymbol{x}) > P(C_k|\boldsymbol{x}) \text{ for all } k \neq i \text{ and} \\ & P(C_i|\boldsymbol{x}) > 1-\lambda \\ \text{reject} & \text{otherwise} \end{array} \tag{3.14}$$

This whole approach is meaningful if $0 < \lambda < 1$: If $\lambda = 0$, we always reject; a reject is as good as a correct classification. If $\lambda \geq 1$, we never reject; a reject is as costly as, or costlier than, an error.

In the case of reject, we are choosing between the automatic decision made by the computer program and human decision that is costlier but assumed to have a higher probability of being correct. Similarly, we can imagine a cascade of multiple automatic decision makers, which as we proceed are costlier but have a higher chance of being correct; we discuss such cascades in chapter 17 where we talk about combining multiple learners.

## 3.4 Discriminant Functions

DISCRIMINANT FUNCTIONS

Classification can also be seen as implementing a set of *discriminant functions*, $g_i(\boldsymbol{x}), i = 1, \ldots, K$, such that we

$$\text{choose } C_i \text{ if } g_i(\boldsymbol{x}) = \max_k g_k(\boldsymbol{x}) \tag{3.15}$$

We can represent the Bayes' classifier in this way by setting

$$g_i(\boldsymbol{x}) = -R(\alpha_i|\boldsymbol{x})$$

and the maximum discriminant function corresponds to minimum conditional risk. When we use the 0/1 loss function, we have

$$g_i(\boldsymbol{x}) = P(C_i|\boldsymbol{x})$$

or ignoring the common normalizing term, $p(\boldsymbol{x})$, we can write

$$g_i(\boldsymbol{x}) = p(\boldsymbol{x}|C_i)P(C_i)$$

DECISION REGIONS

This divides the feature space into $K$ *decision regions* $\mathcal{R}_1, \ldots, \mathcal{R}_K$, where $\mathcal{R}_i = \{\boldsymbol{x}|g_i(\boldsymbol{x}) = \max_k g_k(\boldsymbol{x})\}$. The regions are separated by *decision boundaries*, surfaces in feature space where ties occur among the largest discriminant functions (see figure 3.1).

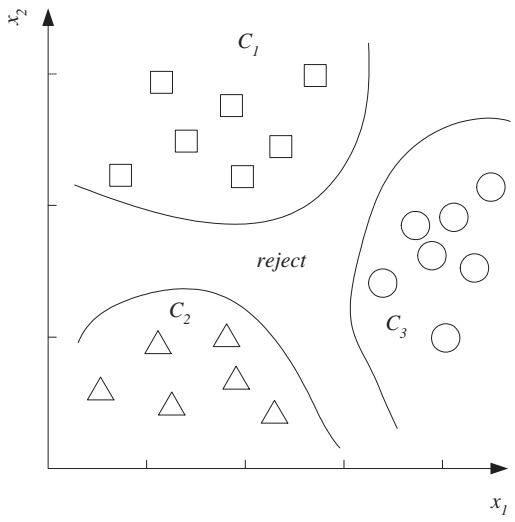


**Figure 3.1** Example of decision regions and decision boundaries.

When there are two classes, we can define a single discriminant

$$g(\boldsymbol{x}) = g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$$

and we

$$\text{choose} \begin{cases} C_1 & \text{if } g(\boldsymbol{x}) > 0 \\ C_2 & \text{otherwise} \end{cases}$$

An example is a two-class learning problem where the positive examples can be taken as $C_1$ and the negative examples as $C_2$. When $K = 2$, the classification system is a *dichotomizer* and for $K \geq 3$, it is a *polychotomizer*.

DICHOTOMIZER
POLYCHOTOMIZER

## 3.5 Association Rules

ASSOCIATION RULE

An *association rule* is an implication of the form $X \rightarrow Y$ where $X$ is the *antecedent* and $Y$ is the *consequent* of the rule. One example of association rules is in *basket analysis* where we want to find the dependency between two items $X$ and $Y$. The typical application is in retail where $X$ and $Y$ are items sold, as we discussed in section 1.2.1.

BASKET ANALYSIS

In learning association rules, there are three measures that are frequently calculated:

SUPPORT

- *Support* of the association rule $X \rightarrow Y$:

$$\text{Support}(X, Y) \equiv P(X, Y) = \frac{\#\{\text{customers who bought } X \text{ and } Y\}}{\#\{\text{customers}\}} \tag{3.16}$$

CONFIDENCE

- *Confidence* of the association rule $X \rightarrow Y$:

$$\begin{aligned} \text{Confidence}(X \rightarrow Y) \equiv P(Y|X) &= \frac{P(X, Y)}{P(X)} \\ &= \frac{\#\{\text{customers who bought } X \text{ and } Y\}}{\#\{\text{customers who bought } X\}} \end{aligned} \tag{3.17}$$

LIFT
INTEREST

- *Lift*, also known as *interest* of the association rule $X \rightarrow Y$:

$$\text{Lift}(X \rightarrow Y) = \frac{P(X, Y)}{P(X)P(Y)} = \frac{P(Y|X)}{P(Y)} \tag{3.18}$$

There are other measures as well (Omiecinski 2003), but these three, especially the first two, are the most widely known and used. *Confidence* is the conditional probability, $P(Y|X)$, which is what we normally calculate. To be able to say that the rule holds with enough confidence, this value should be close to 1 and significantly larger than $P(Y)$, the overall probability of people buying $Y$. We are also interested in maximizing the *support* of the rule, because even if there is a dependency with a strong confidence value, if the number of such customers is small, the rule is worthless. Support shows the statistical significance of the rule, whereas confidence shows the strength of the rule. The minimum support and confidence values are set by the company, and all rules with higher support and confidence are searched for in the database.

If $X$ and $Y$ are independent, then we expect lift to be close to 1; if the ratio differs—if $P(Y|X)$ and $P(Y)$ are different—we expect there to be a dependency between the two items: If the lift is more than 1, we can say that $X$ makes $Y$ more likely, and if the lift is less than 1, having $X$ makes $Y$ less likely.

These formulas can easily be generalized to more than two items. For example, $\{X, Y, Z\}$ is a three-item set, and we may look for a rule, such as $X, Z \rightarrow Y$, that is, $P(Y|X, Z)$. We are interested in finding all such rules having high enough support and confidence and because a sales database

Apriori algorithm

is generally very large, we want to find them by doing a small number of passes over the database. There is an efficient algorithm, called *Apriori* (Agrawal et al. 1996) that does this, which has two steps: (1) finding frequent itemsets, that is, those which have enough support, and (2) converting them to rules with enough confidence, by splitting the items into two, as items in the antecedent and items in the consequent:

1. To find frequent itemsets quickly (without complete enumeration of all subsets of items), the Apriori algorithm uses the fact that for $\{X, Y, Z\}$ to be frequent (have enough support), all its subsets $\{X, Y\}$, $\{X, Z\}$, and $\{Y, Z\}$ should be frequent as well—adding another item can never increase support. That is, we only need to check for three-item sets all of whose two-item subsets are frequent; or, in other words, if a two-item set is known not to be frequent, all its supersets can be pruned and need not be checked.

   We start by finding the frequent one-item sets and at each step, inductively, from frequent $k$-item sets, we generate candidate $k+1$-item sets and then do a pass over the data to check if they have enough support. The Apriori algorithm stores the frequent itemsets in a hash table for easy access. Note that the number of candidate itemsets will decrease very rapidly as $k$ increases. If the largest itemset has $n$ items, we need a total of $n + 1$ passes over the data.

2. Once we find the frequent $k$-item sets, we need to convert them to rules by splitting the $k$ items into two as antecedent and consequent. Just like we do for generating the itemsets, we start by putting a single consequent and $k - 1$ items in the antecedent. Then, for all possible single consequents, we check if the rule has enough confidence and remove it if it does not.

   Note that for the same itemset, there may be multiple rules with different subsets as antecedent and consequent. Then, inductively, we check whether we can move another item from the antecedent to the consequent. Rules with more items in the consequent are more specific and more useful. Here, as in itemset generation, we use the fact that to be able to have rules with two items in the consequent with enough confidence, each of the two rules with single consequent by itself should have enough confidence; that is, we go from one consequent rules to two consequent rules and need not check for all possible two-term consequents (exercise 9).

HIDDEN VARIABLES

Keep in mind that a rule $X \rightarrow Y$ need not imply causality but just an association. In a problem, there may also be *hidden variables* whose values are never known through evidence. The advantage of using hidden variables is that the dependency structure can be more easily defined. For example, in basket analysis when we want to find the dependencies among items sold, let us say we know that there is a dependency among "baby food," "diapers," and "milk" in that a customer buying one of these is very much likely to buy the other two. Instead of representing dependencies among these three, we may designate a hidden node, "baby at home," as the hidden cause of the consumption of these three items. Graphical models that we discuss in chapter 14 allow us to represent such hidden variables. When there are hidden nodes, their values are estimated given the values of observed nodes and filled in.

## 3.6 Notes

Making decisions under uncertainty has a long history, and over time humanity has looked at all sorts of strange places for evidence to remove the uncertainty: stars, crystal balls, and coffee cups. Reasoning from meaningful evidence using probability theory is only a few hundred years old; see Newman 1988 for the history of probability and statistics and some very early articles by Laplace, Bernoulli, and others who have founded the theory.

Russell and Norvig (2009) give an excellent discussion of utility theory and the value of information, also discussing the assignment of utilities in monetary terms. Shafer and Pearl 1990 is an early collection of articles on reasoning under uncertainty.

Association rules are successfully used in many data mining applications, and we see such rules on many web sites that recommend books, movies, music, and so on. The algorithm is very simple and its efficient implementation on very large databases is critical (Zhang and Zhang 2002; Li 2006). Later, we see in chapter 14 how to generalize from association rules to concepts that need not be binary and where associations can be of different types, also allowing hidden variables.

RECOMMENDATION SYSTEMS

*Recommendation systems* are fast becoming one of the major application areas of machine learning. Many retail industries are interested in predicting future customer behavior using past sales data. We can visualize the data as a matrix where rows are the customers, columns

are the items, and entries are the amounts purchased or maybe customer ratings; typically this matrix is very big and also very sparse—most customers have purchased only a very small percentage of the possible items. Though this matrix is very large, it has small rank. This is because there is lot of dependency in the data. People do not shop randomly. People with babies, for example, buy similar things. Certain products are always bought together, or never at the same time. It is these types of regularities, a small number of hidden factors, that makes the matrix low rank. When we talk about dimensionality reduction in chapter 6, we see how we can extract such hidden factors, or dependencies, from data.

## 3.7 Exercises

1. Assume a disease so rare that it is seen in only one person out of every million. Assume also that we have a test that is effective in that if a person has the disease, there is a 99 percent chance that the test result will be positive; however, the test is not perfect, and there is a one in a thousand chance that the test result will be positive on a healthy person. Assume that a new patient arrives and the test result is positive. What is the probability that the patient has the disease ?

   SOLUTION: Let us represent disease by $d$ and test result by $t$. We are given the following: $P(d=1)=10^{-6}, P(t=1|d=1)=0.99, P(t=1|d=0)=10^{-3}$. We are asked $P(d=1|t=1)$.

   We use Bayes' rule:

$$\begin{aligned} P(d=1|t=1) &= \frac{P(t=1|d=1)P(d=1)}{P(t=1)} \\ &= \frac{P(t=1|d=1)P(d=1)}{P(t=1|d=1)P(d=1)+P(t=1|d=0)P(d=0)} \\ &= \frac{0.99\cdot 10^{-6}}{0.99\cdot 10^{-6}+10^{-3}\cdot(1-10^{-6})} = 0.00098902 \end{aligned}$$

   That is, knowing that the test result is positive increased the probability of disease from one in a million to one in a thousand.

LIKELIHOOD RATIO

2. In a two-class problem, the *likelihood ratio* is

$$\frac{p(\boldsymbol{x}|C_1)}{p(\boldsymbol{x}|C_2)}$$

   Write the discriminant function in terms of the likelihood ratio.

   SOLUTION: We can define a discriminant function as

$$g(x)=\frac{P(C_1|x)}{P(C_2|x)} \text{ and choose } \begin{cases} C_1 & \text{if } g(x)>1 \\ C_2 & \text{otherwise} \end{cases}$$

We can write the discriminant as the product of the likelihood ratio and the ratio of priors:

$$g(x) = \frac{p(x|C_1)}{p(x|C_2)} \frac{P(C_1)}{P(C_2)}$$

If the priors are equal, the discriminant is the likelihood ratio.

LOG ODDS

3. In a two-class problem, the *log odds* is defined as

$$\log \frac{P(C_1|\boldsymbol{x})}{P(C_2|\boldsymbol{x})}$$

Write the discriminant function in terms of the log odds.

SOLUTION: We define a discriminant function as

$$g(x) = \log \frac{P(C_1|x)}{P(C_2|x)} \text{ and choose } \begin{cases} C_1 & \text{if } g(x) > 0 \\ C_2 & \text{otherwise} \end{cases}$$

Log odds is the sum of log likelihood ratio and log of prior ratio:

$$g(x) = \log \frac{p(x|C_1)}{p(x|C_2)} + \log \frac{P(C_1)}{P(C_2)}$$

If the priors are equal, the discriminant is the log likelihood ratio.

4. In a two-class, two-action problem, if the loss function is $\lambda_{11} = \lambda_{22} = 0$, $\lambda_{12} = 10$, and $\lambda_{21} = 5$, write the optimal decision rule. How does the rule change if we add a third action of reject with $\lambda = 1$?

SOLUTION: The loss table is as follows:

| | Truth | |
|---|---|---|
| Action | $C_1$ | $C_2$ |
| $\alpha_1$: Choose $C_1$ | 0 | 10 |
| $\alpha_2$: Choose $C_2$ | 5 | 0 |

Let us calculate the expected risks of the two actions:

$$\begin{aligned} R(\alpha_1|x) &= 0 \cdot P(C_1|x) + 10 \cdot P(C_2|x) = 10 \cdot (1 - P(C_1|x)) \\ R(\alpha_2|x) &= 5 \cdot P(C_1|x) + 0 \cdot P(C_2|x) = 5 \cdot P(C_1|x) \end{aligned}$$

We choose $\alpha_1$ if

$$\begin{aligned} R(\alpha_1|x) &< R(\alpha_2|x) \\ 10 \cdot (1 - P(C_1|x)) &< 5 \cdot P(C_1|x) \\ P(C_1|x) &> 2/3 \end{aligned}$$

If the two misclassifications were equally costly, the decision threshold would be at $1/2$ but because the cost of wrongly choosing $C_1$ is higher, we want to choose $C_1$ only when we are really certain; see figure 3.2a and b.

If we add a reject option with a cost of 1, the loss table now becomes

| Action | Truth $C_1$ | $C_2$ |
|---|---|---|
| $\alpha_1$: Choose $C_1$ | 0 | 10 |
| $\alpha_2$: Choose $C_2$ | 5 | 0 |
| $\alpha_r$: Reject | 1 | 1 |

Let us calculate the expected risks of the three actions:

$$R(\alpha_1|x) = 0 \cdot P(C_1|x) + 10 \cdot P(C_2|x) = 10 \cdot (1 - P(C_1|x))$$
$$R(\alpha_2|x) = 5 \cdot P(C_1|x) + 0 \cdot P(C_2|x) = 5 \cdot P(C_1|x)$$
$$R(\alpha_r|x) = 1$$

We choose $\alpha_1$ if

$$R(\alpha_1|x) < 1 \Rightarrow P(C_1|x) > 9/10$$

We choose $\alpha_2$ if

$$R(\alpha_2|x) < 1 \Rightarrow P(C_1|x) < 1/5, \text{ or equivalently if } P(C_1|x) > 4/5$$

We reject otherwise, that is, if $1/5 < P(C_1|x) < 9/10$; see figure 3.2c.

5. Propose a three-level cascade where when one level rejects, the next one is used as in equation 3.10. How can we fix the $\lambda$ on different levels?
6. Somebody tosses a fair coin and if the result is heads, you get nothing; otherwise, you get \$5. How much would you pay to play this game? What if the win is \$500 instead of \$5?
7. Given the following data of transactions at a shop, calculate the support and confidence values of milk → bananas, bananas → milk, milk → chocolate, and chocolate → milk.

| Transaction | Items in basket |
|---|---|
| 1 | milk, bananas, chocolate |
| 2 | milk, chocolate |
| 3 | milk, bananas |
| 4 | chocolate |
| 5 | chocolate |
| 6 | milk, chocolate |

SOLUTION:

$$\text{milk} \rightarrow \text{bananas} : \text{Support} = 2/6, \text{Confidence} = 2/4$$
$$\text{bananas} \rightarrow \text{milk} : \text{Support} = 2/6, \text{Confidence} = 2/2$$
$$\text{milk} \rightarrow \text{chocolate} : \text{Support} = 3/6, \text{Confidence} = 3/4$$
$$\text{chocolate} \rightarrow \text{milk} : \text{Support} = 3/6, \text{Confidence} = 3/5$$

Though only half of the people who buy milk buy bananas too, anyone who buys bananas also buys milk.

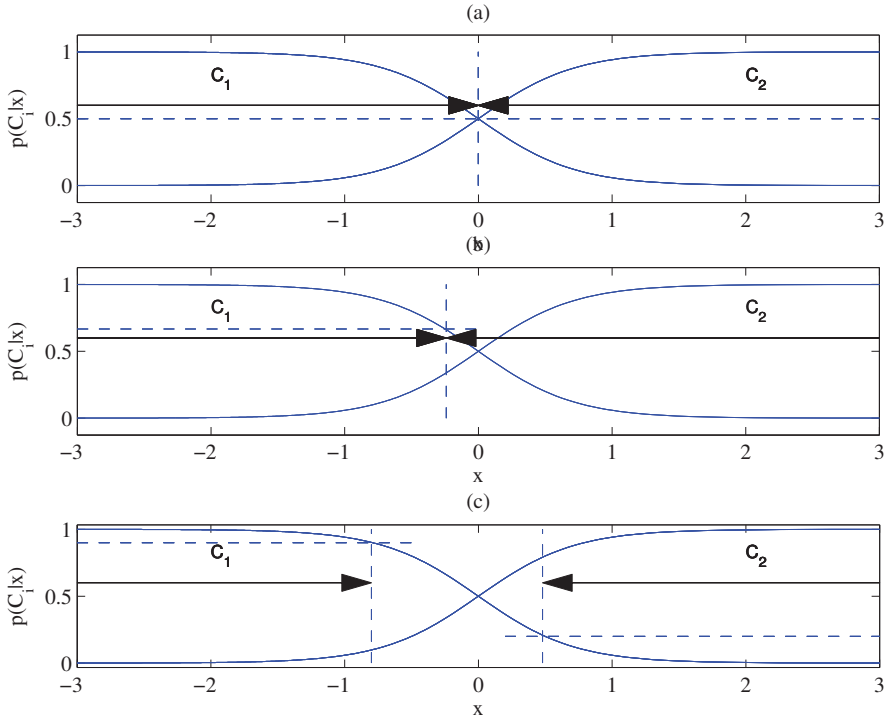


**Figure 3.2** The boundary changes as the misclassification losses change. (a) The boundary is where the two posteriors are equal when both misclassifications are equally costly. (b) When the losses are not symmetric, the boundary shifts toward the class that incurs higher risk when misclassified. (c) When there is the option of reject, a region around the boundary is the region of reject.

8. Generalize the confidence and support formulas for basket analysis to calculate $k$-dependencies, namely, $P(Y|X_1, \ldots, X_k)$.
9. Show that as we move an item from the consequent to the antecedent, confidence can never increase: confidence$(ABC \rightarrow D) \geq$ confidence$(AB \rightarrow CD)$.
10. Associated with each item sold in basket analysis, if we also have a number indicating how much the customer enjoyed the product, for example, on a scale of 0 to 10, how can you use this extra information to calculate which item to propose to a customer?
11. Show example transaction data where for the rule $X \rightarrow Y$:

(a) Both support and confidence are high.

(b) Support is high and confidence is low.

(c) Support is low and confidence is high.

(d) Both support and confidence are low.

## 3.8 References

Agrawal, R., H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. 1996. "Fast Discovery of Association Rules." In *Advances in Knowledge Discovery and Data Mining*, ed. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, 307–328. Cambridge, MA: MIT Press.

Duda, R. O., P. E. Hart, and D. G. Stork. 2001. *Pattern Classification*, 2nd ed. New York: Wiley.

Li, J. 2006. "On Optimal Rule Discovery." *IEEE Transactions on Knowledge and Data Discovery* 18:460–471.

Newman, J. R., ed. 1988. *The World of Mathematics*. Redmond, WA: Tempus.

Omiecinski, E. R. 2003. "Alternative Interest Measures for Mining Associations in Databases." *IEEE Transactions on Knowledge and Data Discovery* 15:57–69.

Russell, S., and P. Norvig. 2009. *Artificial Intelligence: A Modern Approach,* 3rd ed. New York: Prentice Hall.

Shafer, G., and J. Pearl, eds. 1990. *Readings in Uncertain Reasoning*. San Mateo, CA: Morgan Kaufmann.

Zhang, C., and S. Zhang. 2002. *Association Rule Mining: Models and Algorithms*. New York: Springer.

# 4 *Parametric Methods*

*Having discussed how to make optimal decisions when the uncertainty is modeled using probabilities, we now see how we can estimate these probabilities from a given training set. We start with the parametric approach for classification and regression; we discuss the semiparametric and nonparametric approaches in later chapters. We introduce bias/variance dilemma and model selection methods for trading off model complexity and empirical error.*

## 4.1 Introduction

A STATISTIC is any value that is calculated from a given sample. In statistical inference, we make a decision using the information provided by a sample. Our first approach is parametric where we assume that the sample is drawn from some distribution that obeys a known model, for example, Gaussian. The advantage of the parametric approach is that the model is defined up to a small number of parameters—for example, mean, variance—the *sufficient statistics* of the distribution. Once those parameters are estimated from the sample, the whole distribution is known. We estimate the parameters of the distribution from the given sample, plug in these estimates to the assumed model, and get an estimated distribution, which we then use to make a decision. The method we use to estimate the parameters of a distribution is maximum likelihood estimation. We also introduce Bayesian estimation, which we continue to discuss in chapter 16.

We start with *density estimation*, which is the general case of estimating $p(x)$. We use this for *classification* where the estimated densities are the class densities, $p(x|C_i)$, and priors, $P(C_i)$, to be able to calculate the pos-

teriors, $P(C_i|x)$, and make our decision. We then discuss *regression* where the estimated density is $p(y|x)$. In this chapter, $x$ is one-dimensional and thus the densities are univariate. We generalize to the multivariate case in chapter 5.

## 4.2 Maximum Likelihood Estimation

Let us say we have an independent and identically distributed (iid) sample $\mathcal{X} = \{x^t\}_{t=1}^N$. We assume that $x^t$ are instances drawn from some known probability density family, $p(x|\theta)$, defined up to parameters, $\theta$:

$$x^t \sim p(x|\theta)$$

LIKELIHOOD

We want to find $\theta$ that makes sampling $x^t$ from $p(x|\theta)$ as likely as possible. Because $x^t$ are independent, the *likelihood* of parameter $\theta$ given sample $\mathcal{X}$ is the product of the likelihoods of the individual points:

$$l(\theta|\mathcal{X}) \equiv p(\mathcal{X}|\theta) = \prod_{t=1}^{N} p(x^t|\theta) \tag{4.1}$$

MAXIMUM LIKELIHOOD ESTIMATION

In *maximum likelihood estimation*, we are interested in finding $\theta$ that makes $\mathcal{X}$ the most likely to be drawn. We thus search for $\theta$ that maximizes the likelihood, which we denote by $l(\theta|\mathcal{X})$. We can maximize the log of the likelihood without changing the value where it takes its maximum. $\log(\cdot)$ converts the product into a sum and leads to further computational simplification when certain densities are assumed, for example, containing exponents. The *log likelihood* is defined as

LOG LIKELIHOOD

$$\mathcal{L}(\theta|\mathcal{X}) \equiv \log l(\theta|\mathcal{X}) = \sum_{t=1}^{N} \log p(x^t|\theta) \tag{4.2}$$

Let us now see some distributions that arise in the applications we are interested in. If we have a two-class problem, the distribution we use is *Bernoulli*. When there are $K > 2$ classes, its generalization is the *multinomial*. *Gaussian* (*normal*) density is the one most frequently used for modeling class-conditional input densities with numeric input. For these three distributions, we discuss the maximum likelihood estimators (MLE) of their parameters.

### 4.2.1 Bernoulli Density

In a Bernoulli distribution, there are two outcomes: An event occurs or it does not; for example, an instance is a positive example of the class, or it is not. The event occurs and the Bernoulli random variable $X$ takes the value 1 with probability $p$, and the nonoccurrence of the event has probability $1 - p$ and this is denoted by $X$ taking the value 0. This is written as

$$P(x) = p^x(1-p)^{1-x}, x \in \{0,1\} \tag{4.3}$$

The expected value and variance can be calculated as

$$\begin{aligned} E[X] &= \sum_x xp(x) = 1 \cdot p + 0 \cdot (1-p) = p \\ \text{Var}(X) &= \sum_x (x - E[X])^2 p(x) = p(1-p) \end{aligned}$$

$p$ is the only parameter and given an iid sample $\mathcal{X} = \{x^t\}_{t=1}^N$, where $x^t \in \{0,1\}$, we want to calculate its estimator, $\hat{p}$. The log likelihood is

$$\begin{aligned} \mathcal{L}(p|\mathcal{X}) &= \log \prod_{t=1}^N p^{(x^t)}(1-p)^{(1-x^t)} \\ &= \sum_t x^t \log p + \left(N - \sum_t x^t\right) \log(1-p) \end{aligned}$$

$\hat{p}$ that maximizes the log likelihood can be found by solving for $d\mathcal{L}/dp = 0$. The hat (circumflex) denotes that it is an estimate.

$$\hat{p} = \frac{\sum_t x^t}{N} \tag{4.4}$$

The estimate for $p$ is the ratio of the number of occurrences of the event to the number of experiments. Remembering that if $X$ is Bernoulli with $p$, $E[X] = p$, and, as expected, the maximum likelihood estimator of the mean is the sample average.

Note that the estimate is a function of the sample and is another random variable; we can talk about the distribution of $\hat{p}_i$ given different $\mathcal{X}_i$ sampled from the same $p(x)$. For example, the variance of the distribution of $\hat{p}_i$ is expected to decrease as $N$ increases; as the samples get bigger, they (and hence their averages) get more similar.

### 4.2.2 Multinomial Density

Consider the generalization of Bernoulli where instead of two states, the outcome of a random event is one of $K$ mutually exclusive and exhaustive states, for example, classes, each of which has a probability of occurring $p_i$ with $\sum_{i=1}^{K} p_i = 1$. Let $x_1, x_2, \ldots, x_K$ are the indicator variables where $x_i$ is 1 if the outcome is state $i$ and 0 otherwise.

$$P(x_1, x_2, \ldots, x_K) = \prod_{i=1}^{K} p_i^{x_i} \tag{4.5}$$

Let us say we do $N$ such independent experiments with outcomes $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^{N}$ where

$$x_i^t = \begin{cases} 1 & \text{if experiment } t \text{ chooses state } i \\ 0 & \text{otherwise} \end{cases}$$

with $\sum_i x_i^t = 1$. The MLE of $p_i$ is

$$\hat{p}_i = \frac{\sum_t x_i^t}{N} \tag{4.6}$$

The estimate for the probability of state $i$ is the ratio of experiments with outcome of state $i$ to the total number of experiments. There are two ways one can get this: If $x_i$ are 0/1, then they can be thought of as $K$ separate Bernoulli experiments. Or, one can explicitly write the log likelihood and find $p_i$ that maximize it (subject to the condition that $\sum_i p_i = 1$).

### 4.2.3 Gaussian (Normal) Density

$X$ is Gaussian (normal) distributed with mean $E[X] \equiv \mu$ and variance $\text{Var}(X) \equiv \sigma^2$, denoted as $\mathcal{N}(\mu, \sigma^2)$, if its density function is

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right], -\infty < x < \infty \tag{4.7}$$

Given a sample $\mathcal{X} = \{x^t\}_{t=1}^{N}$ with $x^t \sim \mathcal{N}(\mu, \sigma^2)$, the log likelihood is

$$\mathcal{L}(\mu, \sigma | \mathcal{X}) = -\frac{N}{2}\log(2\pi) - N\log\sigma - \frac{\sum_t (x^t - \mu)^2}{2\sigma^2}$$

The MLE that we find by taking the partial derivatives of the log likelihood and setting them equal to 0 are

$$\begin{aligned} m &= \frac{\sum_t x^t}{N} \\ s^2 &= \frac{\sum_t (x^t - m)^2}{N} \end{aligned} \tag{4.8}$$

We follow the usual convention and use Greek letters for the population parameters and Roman letters for their estimates from the sample. Sometimes, the hat is also used to denote the estimator, for example, $\hat{\mu}$.

## 4.3 Evaluating an Estimator: Bias and Variance

Let $\mathcal{X}$ be a sample from a population specified up to a parameter $\theta$, and let $d = d(\mathcal{X})$ be an estimator of $\theta$. To evaluate the quality of this estimator, we can measure how much it is different from $\theta$, that is, $(d(\mathcal{X}) - \theta)^2$. But since it is a random variable (it depends on the sample), we need to average this over possible $\mathcal{X}$ and consider $r(d, \theta)$, the *mean square error* of the estimator $d$ defined as

MEAN SQUARE ERROR

$$r(d, \theta) = E[(d(\mathcal{X}) - \theta)^2] \tag{4.9}$$

BIAS

The *bias* of an estimator is given as

$$b_\theta(d) = E[d(\mathcal{X})] - \theta \tag{4.10}$$

UNBIASED ESTIMATOR

If $b_\theta(d) = 0$ for all $\theta$ values, then we say that $d$ is an *unbiased estimator* of $\theta$. For example, with $x^t$ drawn from some density with mean $\mu$, the sample average, $m$, is an unbiased estimator of the mean, $\mu$, because

$$E[m] = E\left[\frac{\sum_t x^t}{N}\right] = \frac{1}{N}\sum_t E[x^t] = \frac{N\mu}{N} = \mu$$

This means that though on a particular sample, $m$ may be different from $\mu$, if we take many such samples, $\mathcal{X}_i$, and estimate many $m_i = m(\mathcal{X}_i)$, *their* average will get close to $\mu$ as the number of such samples increases. $m$ is also a *consistent* estimator, that is, $\text{Var}(m) \to 0$ as $N \to \infty$.

$$\text{Var}(m) = \text{Var}\left(\frac{\sum_t x^t}{N}\right) = \frac{1}{N^2}\sum_t \text{Var}(x^t) = \frac{N\sigma^2}{N^2} = \frac{\sigma^2}{N}$$

As $N$, the number of points in the sample, gets larger, $m$ deviates less from $\mu$. Let us now check, $s^2$, the MLE of $\sigma^2$:

$$\begin{aligned} s^2 &= \frac{\sum_t (x^t - m)^2}{N} = \frac{\sum_t (x^t)^2 - Nm^2}{N} \\ E[s^2] &= \frac{\sum_t E[(x^t)^2] - N \cdot E[m^2]}{N} \end{aligned}$$

Given that $\text{Var}(X) = E[X^2] - E[X]^2$, we get $E[X^2] = \text{Var}(X) + E[X]^2$, and we can write

$$E[(x^t)^2] = \sigma^2 + \mu^2 \text{ and } E[m^2] = \sigma^2/N + \mu^2$$

Then, plugging these in, we get

$$E[s^2] = \frac{N(\sigma^2 + \mu^2) - N(\sigma^2/N + \mu^2)}{N} = \left(\frac{N-1}{N}\right)\sigma^2 \neq \sigma^2$$

which shows that $s^2$ is a biased estimator of $\sigma^2$. $(N/(N-1))s^2$ is an unbiased estimator. However when $N$ is large, the difference is negligible. This is an example of an *asymptotically unbiased estimator* whose bias goes to 0 as $N$ goes to infinity.

The mean square error can be rewritten as follows—$d$ is short for $d(\mathcal{X})$:

$$\begin{aligned} r(d,\theta) &= E\left[(d-\theta)^2\right] \\ &= E\left[(d - E[d] + E[d] - \theta)^2\right] \\ &= E\left[(d-E[d])^2 + (E[d]-\theta)^2 + 2(E[d]-\theta)(d-E[d])\right] \\ &= E\left[(d-E[d])^2\right] + E\left[(E[d]-\theta)^2\right] + 2E\left[(E[d]-\theta)(d-E[d])\right] \\ &= E\left[(d-E[d])^2\right] + (E[d]-\theta)^2 + 2(E[d]-\theta)E[d-E[d]] \\ &= \underbrace{E\left[(d-E[d])^2\right]}_{variance} + \underbrace{(E[d]-\theta)^2}_{bias^2} \end{aligned} \tag{4.11}$$

The two equalities follow because $E[d]$ is a constant and therefore $E[d] - \theta$ also is a constant, and because $E[d - E[d]] = E[d] - E[d] = 0$. In equation 4.11, the first term is the *variance* that measures how much, on average, $d_i$ vary around the expected value (going from one dataset to another), and the second term is the *bias* that measures how much the expected value varies from the correct value $\theta$ (figure 4.1). We then write error as the sum of these two terms, the variance and the square of the bias:

VARIANCE

$$r(d,\theta) = \mathrm{Var}(d) + (b_\theta(d))^2 \tag{4.12}$$

## 4.4 The Bayes' Estimator

Sometimes, before looking at a sample, we (or experts of the application) may have some *prior* information on the possible value range that a parameter, $\theta$, may take. This information is quite useful and should be used, especially when the sample is small. The prior information does not tell us exactly what the parameter value is (otherwise we would not

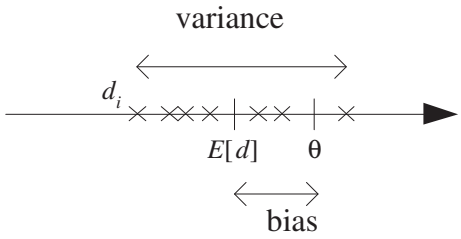


**Figure 4.1** $\theta$ is the parameter to be estimated. $d_i$ are several estimates (denoted by '$\times$') over different samples $\mathcal{X}_i$. Bias is the difference between the expected value of $d$ and $\theta$. Variance is how much $d_i$ are scattered around the expected value. We would like both to be small.

need the sample), and we model this uncertainty by viewing $\theta$ as a random variable and by defining a prior density for it, $p(\theta)$. For example, let us say we are told that $\theta$ is approximately normal and with 90 percent confidence, $\theta$ lies between 5 and 9, symmetrically around 7. Then we can write $p(\theta)$ to be normal with mean 7 and because

$$\begin{aligned} P\{-1.64 < \frac{\theta - \mu}{\sigma} < 1.64\} &= 0.9 \\ P\{\mu - 1.64\sigma < \theta < \mu + 1.64\sigma\} &= 0.9 \end{aligned}$$

we take $1.64\sigma = 2$ and use $\sigma = 2/1.64$. We can thus assume $p(\theta) \sim \mathcal{N}(7, (2/1.64)^2)$.

PRIOR DENSITY

The *prior density*, $p(\theta)$, tells us the likely values that $\theta$ may take *before* looking at the sample. We combine this with what the sample data tells us, namely, the likelihood density, $p(\mathcal{X}|\theta)$, using Bayes' rule, and get the

POSTERIOR DENSITY

*posterior density* of $\theta$, which tells us the likely $\theta$ values *after* looking at the sample:

$$p(\theta|\mathcal{X}) = \frac{p(\mathcal{X}|\theta)p(\theta)}{p(\mathcal{X})} = \frac{p(\mathcal{X}|\theta)p(\theta)}{\int p(\mathcal{X}|\theta')p(\theta')d\theta'} \tag{4.13}$$

For estimating the density at $x$, we have

$$\begin{aligned} p(x|\mathcal{X}) &= \int p(x, \theta|\mathcal{X})d\theta \\ &= \int p(x|\theta, \mathcal{X})p(\theta|\mathcal{X})d\theta \\ &= \int p(x|\theta)p(\theta|\mathcal{X})d\theta \end{aligned}$$

$p(x|\theta, \mathcal{X}) = p(x|\theta)$ because once we know $\theta$, the sufficient statistics, we know everything about the distribution. Thus we are taking an average over predictions using all values of $\theta$, weighted by their probabilities. If we are doing a prediction in the form, $y = g(x|\theta)$, as in regression, then we have

$$y = \int g(x|\theta)p(\theta|\mathcal{X})d\theta$$

Evaluating the integrals may be quite difficult, except in cases where the posterior has a nice form. When the full integration is not feasible, we reduce it to a single point. If we can assume that $p(\theta|\mathcal{X})$ has a narrow peak around its mode, then using the *maximum a posteriori* (MAP) *estimate* will make the calculation easier:

MAXIMUM A POSTERIORI ESTIMATE

$$\theta_{MAP} = \arg\max_{\theta} p(\theta|\mathcal{X}) \tag{4.14}$$

thus replacing a whole density with a single point, getting rid of the integral and using as

$$\begin{aligned} p(x|\mathcal{X}) &= p(x|\theta_{MAP}) \\ y_{MAP} &= g(x|\theta_{MAP}) \end{aligned}$$

If we have no prior reason to favor some values of $\theta$, then the prior density is flat and the posterior will have the same form as the likelihood, $p(\mathcal{X}|\theta)$, and the MAP estimate will be equivalent to the maximum likelihood estimate (section 4.2) where we have

$$\theta_{ML} = \arg\max_{\theta} p(\mathcal{X}|\theta) \tag{4.15}$$

BAYES' ESTIMATOR

Another possibility is the *Bayes' estimator*, which is defined as the expected value of the posterior density

$$\theta_{Bayes} = E[\theta|\mathcal{X}] = \int \theta p(\theta|\mathcal{X})d\theta \tag{4.16}$$

The reason for taking the expected value is that the best estimate of a random variable is its mean. Let us say $\theta$ is the variable we want to predict with $E[\theta] = \mu$. It can be shown that if $c$, a constant value, is our estimate of $\theta$, then

$$\begin{aligned} E[(\theta - c)^2] &= E[(\theta - \mu + \mu - c)^2] \\ &= E[(\theta - \mu)^2] + (\mu - c)^2 \end{aligned} \tag{4.17}$$

which is minimum if $c$ is taken as $\mu$. In the case of a normal density, the mode is the expected value and if $p(\theta|\mathcal{X})$ is normal, then $\theta_{Bayes} = \theta_{MAP}$.

As an example, let us suppose $x^t \sim \mathcal{N}(\theta, \sigma^2)$ and $\theta \sim \mathcal{N}(\mu_0, \sigma_0^2)$, where $\mu_0$, $\sigma_0^2$, and $\sigma^2$ are known:

$$\begin{aligned} p(\mathcal{X}|\theta) &= \frac{1}{(2\pi)^{N/2}\sigma^N} \exp\left[-\frac{\sum_t (x^t-\theta)^2}{2\sigma^2}\right] \\ p(\theta) &= \frac{1}{\sqrt{2\pi}\sigma_0} \exp\left[-\frac{(\theta-\mu_0)^2}{2\sigma_0^2}\right] \end{aligned}$$

It can be shown that $p(\theta|\mathcal{X})$ is normal with

$$E[\theta|\mathcal{X}] = \frac{N/\sigma^2}{N/\sigma^2 + 1/\sigma_0^2} m + \frac{1/\sigma_0^2}{N/\sigma^2 + 1/\sigma_0^2} \mu_0 \tag{4.18}$$

Thus the Bayes' estimator is a weighted average of the prior mean $\mu_0$ and the sample mean $m$, with weights being inversely proportional to their variances. As the sample size $N$ increases, the Bayes' estimator gets closer to the sample average, using more the information provided by the sample. When $\sigma_0^2$ is small, that is, when we have little prior uncertainty regarding the correct value of $\theta$, or when $N$ is small, our prior guess $\mu_0$ has a higher effect.

Note that both MAP and Bayes' estimators reduce the whole posterior density to a single point and lose information unless the posterior is unimodal and makes a narrow peak around these points. With computation getting cheaper, we can use a Monte Carlo approach that generates samples from the posterior density (Andrieu et al. 2003). There also are approximation methods one can use to evaluate the full integral. We are going to discuss Bayesian estimation in more detail in chapter 16.

## 4.5 Parametric Classification

We saw in chapter 3 that using the Bayes' rule, we can write the posterior probability of class $C_i$ as

$$P(C_i|x) = \frac{p(x|C_i)P(C_i)}{p(x)} = \frac{p(x|C_i)P(C_i)}{\sum_{k=1}^K p(x|C_k)P(C_k)} \tag{4.19}$$

and use the discriminant function

$$g_i(x) = p(x|C_i)P(C_i)$$

or equivalently

$$g_i(x) = \log p(x|C_i) + \log P(C_i) \tag{4.20}$$

If we can assume that $p(x|C_i)$ are Gaussian

$$p(x|C_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left[-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right] \tag{4.21}$$

equation 4.20 becomes

$$g_i(x) = -\frac{1}{2}\log 2\pi - \log \sigma_i - \frac{(x-\mu_i)^2}{2\sigma_i^2} + \log P(C_i) \tag{4.22}$$

Let us see an example: Assume we are a car company selling $K$ different cars, and for simplicity, let us say that the sole factor that affects a customer's choice is his or her yearly income, which we denote by $x$. Then $P(C_i)$ is the proportion of customers who buy car type $i$. If the yearly income distributions of such customers can be approximated with a Gaussian, then $p(x|C_i)$, the probability that a customer who bought car type $i$ has income $x$, can be taken $\mathcal{N}(\mu_i, \sigma_i^2)$, where $\mu_i$ is the mean income of such customers and $\sigma_i^2$ is their income variance.

When we do not know $P(C_i)$ and $p(x|C_i)$, we estimate them from a sample and plug in their estimates to get the estimate for the discriminant function. We are given a sample

$$\mathcal{X} = \{x^t, \boldsymbol{r}^t\}_{t=1}^N \tag{4.23}$$

where $x \in \Re$ is one-dimensional and $\boldsymbol{r} \in \{0,1\}^K$ such that

$$r_i^t = \begin{cases} 1 & \text{if } \boldsymbol{x}^t \in C_i \\ 0 & \text{if } \boldsymbol{x}^t \in C_k, k \neq i \end{cases} \tag{4.24}$$

For each class separately, the estimates for the means and variances are (relying on equation 4.8)

$$m_i = \frac{\sum_t x^t r_i^t}{\sum_t r_i^t} \tag{4.25}$$

$$s_i^2 = \frac{\sum_t (x^t - m_i)^2 r_i^t}{\sum_t r_i^t} \tag{4.26}$$

and the estimates for the priors are (relying on equation 4.6)

$$\hat{P}(C_i) = \frac{\sum_t r_i^t}{N} \tag{4.27}$$

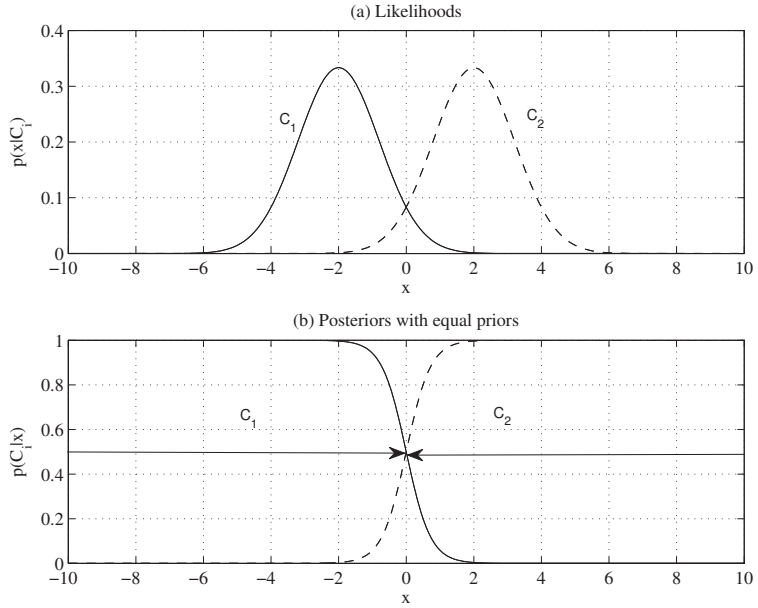


**Figure 4.2** (a) Likelihood functions and (b) posteriors with equal priors for two classes when the input is one-dimensional. Variances are equal and the posteriors intersect at one point, which is the threshold of decision.

Plugging these estimates into equation 4.22, we get

$$g_i(x) = -\frac{1}{2}\log 2\pi - \log s_i - \frac{(x-m_i)^2}{2s_i^2} + \log \hat{P}(C_i) \tag{4.28}$$

The first term is a constant and can be dropped because it is common in all $g_i(x)$. If the priors are equal, the last term can also be dropped. If we can further assume that variances are equal, we can write

$$g_i(x) = -(x-m_i)^2 \tag{4.29}$$

and thus we assign $x$ to the class with the nearest mean:

Choose $C_i$ if $|x-m_i| = \min_k |x-m_k|$

With two adjacent classes, the midpoint between the two means is the threshold of decision (see figure 4.2).

$$g_1(x) = g_2(x)$$

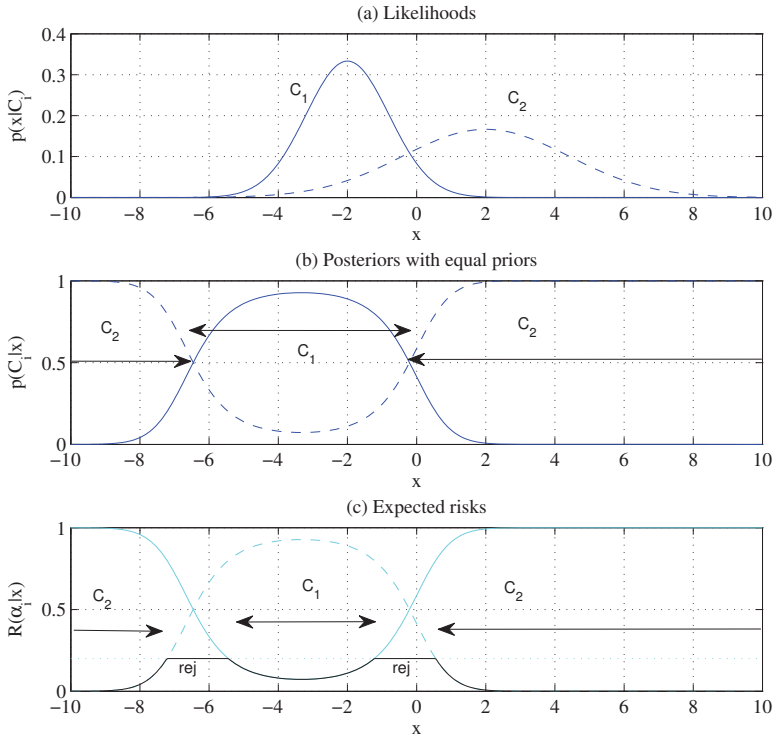


**Figure 4.3** (a) Likelihood functions and (b) posteriors with equal priors for two classes when the input is one-dimensional. Variances are unequal and the posteriors intersect at two points. In (c), the expected risks are shown for the two classes and for reject with $\lambda = 0.2$ (section 3.3).

$$\begin{aligned}(x - m_1)^2 &= (x - m_2)^2 \\ x &= \frac{m_1 + m_2}{2}\end{aligned}$$

When the variances are different, there are two thresholds (see figure 4.3), which can be calculated easily (exercise 4). If the priors are different, this has the effect of moving the threshold of decision toward the mean of the less likely class.

Here we use the maximum likelihood estimators for the parameters but if we have some prior information about them, for example, for the means, we can use a Bayesian estimate of $p(x|C_i)$ with prior on $\mu_i$.

One note of caution is necessary here: When $x$ is continuous, we should not immediately rush to use Gaussian densities for $p(x|C_i)$. The classification algorithm—that is, the threshold points—will be wrong if the densities are not Gaussian. In statistical literature, tests exist to check for normality, and such a test should be used before assuming normality. In the case of one-dimensional data, the easiest test is to plot the histogram and to check visually whether the density is bell-shaped, namely, unimodal and symmetric around the center.

This is the *likelihood-based approach* to classification where we use data to estimate the densities separately, calculate posterior densities using Bayes' rule, and then get the discriminant. In later chapters, we discuss the *discriminant-based approach* where we bypass the estimation of densities and directly estimate the discriminants.

## 4.6 Regression

In regression, we would like to write the numeric output, called the *dependent variable*, as a function of the input, called the *independent variable*. We assume that the numeric output is the sum of a deterministic function of the input and random noise:

$$r = f(x) + \epsilon$$

where $f(x)$ is the unknown function, which we would like to approximate by our estimator, $g(x|\theta)$, defined up to a set of parameters $\theta$. If we assume that $\epsilon$ is zero mean Gaussian with constant variance $\sigma^2$, namely, $\epsilon \sim \mathcal{N}(0, \sigma^2)$, and placing our estimator $g(\cdot)$ in place of the unknown function $f(\cdot)$, we have (figure 4.4)

$$p(r|x) \sim \mathcal{N}(g(x|\theta), \sigma^2) \tag{4.30}$$

We again use maximum likelihood to learn the parameters $\theta$. The pairs $(x^t, r^t)$ in the training set are drawn from an unknown joint probability density $p(x, r)$, which we can write as

$$p(x, r) = p(r|x)p(x)$$

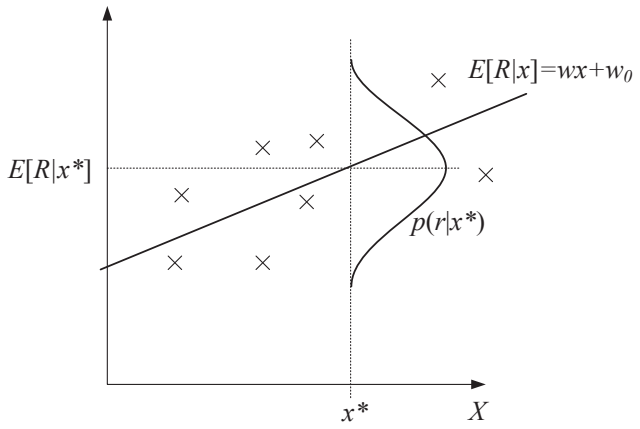


**Figure 4.4** Regression assumes 0 mean Gaussian noise added to the model; here, the model is linear.

$p(r|x)$ is the probability of the output given the input, and $p(x)$ is the input density. Given an iid sample $\mathcal{X} = \{x^t, r^t\}_{t=1}^N$, the log likelihood is

$$\begin{aligned}\mathcal{L}(\theta|\mathcal{X}) &= \log \prod_{t=1}^{N} p(x^t, r^t) \\ &= \log \prod_{t=1}^{N} p(r^t|x^t) + \log \prod_{t=1}^{N} p(x^t)\end{aligned}$$

We can ignore the second term since it does not depend on our estimator, and we have

$$\begin{aligned}\mathcal{L}(\theta|\mathcal{X}) &= \log \prod_{t=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{[r^t - g(x^t|\theta)]^2}{2\sigma^2}\right] \\ &= \log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)^N \exp\left[-\frac{1}{2\sigma^2}\sum_{t=1}^{N}[r^t - g(x^t|\theta)]^2\right] \\ &= -N\log(\sqrt{2\pi}\sigma) - \frac{1}{2\sigma^2}\sum_{t=1}^{N}[r^t - g(x^t|\theta)]^2\end{aligned} \tag{4.31}$$

The first term is independent of the parameters $\theta$ and can be dropped, as can the factor $1/\sigma^2$. Maximizing this is equivalent to minimizing

$$E(\theta|\mathcal{X}) = \frac{1}{2}\sum_{t=1}^{N}[r^t - g(x^t|\theta)]^2 \tag{4.32}$$

LEAST SQUARES ESTIMATE

which is the most frequently used error function, and $\theta$ that minimize it are called the *least squares estimates*. This is a transformation frequently done in statistics: When the likelihood $l$ contains exponents, instead of maximizing $l$, we define an *error function*, $E = -\log l$, and minimize it.

LINEAR REGRESSION

In *linear regression*, we have a linear model

$$g(x^t|w_1, w_0) = w_1 x^t + w_0$$

and taking the derivative of the sum of squared errors (equation 4.32) with respect to $w_1$ and $w_0$, we have two equations in two unknowns

$$\begin{aligned} \sum_t r^t &= N w_0 + w_1 \sum_t x^t \\ \sum_t r^t x^t &= w_0 \sum_t x_t + w_1 \sum_t (x^t)^2 \end{aligned}$$

which can be written in vector-matrix form as $\mathbf{A}\boldsymbol{w} = \boldsymbol{y}$ where

$$\mathbf{A} = \begin{bmatrix} N & \sum_t x^t \\ \sum_t x^t & \sum_t (x^t)^2 \end{bmatrix}, \quad \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} \sum_t r^t \\ \sum_t r^t x^t \end{bmatrix}$$

and can be solved as $\boldsymbol{w} = \mathbf{A}^{-1}\boldsymbol{y}$.

POLYNOMIAL REGRESSION

In the general case of *polynomial regression*, the model is a polynomial in $x$ of order $k$

$$g(x^t|w_k, \ldots, w_2, w_1, w_0) = w_k (x^t)^k + \cdots + w_2 (x^t)^2 + w_1 x^t + w_0$$

The model is still linear with respect to the parameters and taking the derivatives, we get $k+1$ equations in $k+1$ unknowns, which can be written in vector matrix form $\mathbf{A}\boldsymbol{w} = \boldsymbol{y}$ where we have

$$\mathbf{A} = \begin{bmatrix} N & \sum_t x^t & \sum_t (x^t)^2 & \cdots & \sum_t (x^t)^k \\ \sum_t x^t & \sum_t (x^t)^2 & \sum_t (x^t)^3 & \cdots & \sum_t (x^t)^{k+1} \\ \vdots & & & & \\ \sum_t (x^t)^k & \sum_t (x^t)^{k+1} & \sum_t (x^t)^{k+2} & \cdots & \sum_t (x^t)^{2k} \end{bmatrix}$$

$$\boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} \sum_t r^t \\ \sum_t r^t x^t \\ \sum_t r^t (x^t)^2 \\ \vdots \\ \sum_t r^t (x^t)^k \end{bmatrix}$$

We can write $\mathbf{A} = \mathbf{D}^T\mathbf{D}$ and $\boldsymbol{y} = \mathbf{D}^T\boldsymbol{r}$ where

$$\mathbf{D} = \begin{bmatrix} 1 & x^1 & (x^1)^2 & \cdots & (x^1)^k \\ 1 & x^2 & (x^2)^2 & \cdots & (x^2)^k \\ \vdots & & & & \\ 1 & x^N & (x^N)^2 & \cdots & (x^N)^k \end{bmatrix}, \boldsymbol{r} = \begin{bmatrix} r^1 \\ r^2 \\ \vdots \\ r^N \end{bmatrix}$$

and we can then solve for the parameters as

$$\boldsymbol{w} = (\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T\boldsymbol{r} \tag{4.33}$$

Assuming Gaussian distributed error and maximizing likelihood corresponds to minimizing the sum of squared errors. Another measure is the *relative square error* (RSE):

RELATIVE SQUARE ERROR

$$E_{RSE} = \frac{\sum_t [r^t - g(x^t|\theta)]^2}{\sum_t (r^t - \overline{r})^2} \tag{4.34}$$

If $E_{RSE}$ is close to 1, then our prediction is as good as predicting by the average; as it gets closer to 0, we have better fit. If $E_{RSE}$ is close to 1, this means that using a model based on input $x$ does not work better than using the average which would be our estimator if there were no $x$; if $E_{RSE}$ is close to 0, input $x$ helps.

COEFFICIENT OF DETERMINATION

A measure to check the goodness of fit by regression is the *coefficient of determination* that is

$$R^2 = 1 - E_{RSE}$$

and for regression to be considered useful, we require $R^2$ to be close to 1.

Remember that for best generalization, we should adjust the complexity of our learner model to the complexity of the data. In polynomial regression, the complexity parameter is the order of the fitted polynomial, and therefore we need to find a way to choose the best order that minimizes the generalization error, that is, tune the complexity of the model to best fit the complexity of the function inherent in the data.

## 4.7 Tuning Model Complexity: Bias/Variance Dilemma

Let us say that a sample $\mathcal{X} = \{x^t, r^t\}$ is drawn from some unknown joint probability density $p(x, r)$. Using this sample, we construct our estimate

$g(\cdot)$. The expected square error (over the joint density) at $x$ can be written as (using equation 4.17)

$$E[(r-g(x))^2|x] = \underbrace{E[(r-E[r|x])^2|x]}_{noise} + \underbrace{(E[r|x]-g(x))^2}_{squared\ error} \tag{4.35}$$

The first term on the right is the variance of $r$ given $x$; it does not depend on $g(\cdot)$ or $\mathcal{X}$. It is the variance of noise added, $\sigma^2$. This is the part of error that can never be removed, no matter what estimator we use. The second term quantifies how much $g(x)$ deviates from the regression function, $E[r|x]$. This does depend on the estimator and the training set. It may be the case that for one sample, $g(x)$ may be a very good fit; and for some other sample, it may make a bad fit. To quantify how well an estimator $g(\cdot)$ is, we average over possible datasets.

The expected value (average over samples $\mathcal{X}$, all of size $N$ and drawn from the same joint density $p(r,x)$) is (using equation 4.11)

$$E_{\mathcal{X}}[(E[r|x]-g(x))^2|x] = \underbrace{(E[r|x]-E_{\mathcal{X}}[g(x)])^2}_{bias} + \underbrace{E_{\mathcal{X}}[(g(x)-E_{\mathcal{X}}[g(x)])^2]}_{variance} \tag{4.36}$$

As we discussed earlier, bias measures how much $g(x)$ is wrong disregarding the effect of varying samples, and variance measures how much $g(x)$ fluctuate around the expected value, $E[g(x)]$, as the sample varies. We want both to be small.

Let us see a didactic example: To estimate the bias and the variance, we generate a number of datasets $\mathcal{X}_i = \{x_i^t, r_i^t\}, i = 1,\ldots,M$, from some known $f(\cdot)$ with added noise, use each dataset to form an estimator $g_i(\cdot)$, and calculate bias and variance. Note that in real life, we cannot do this because we do not know $f(\cdot)$ or the parameters of the added noise. Then $E[g(x)]$ is estimated by the average over $g_i(\cdot)$:

$$\bar{g}(x) = \frac{1}{M}\sum_{i=1}^{M} g_i(x)$$

Estimated bias and variance are

$$\begin{aligned}\text{bias}^2(g) &= \frac{1}{N}\sum_t [\bar{g}(x^t) - f(x^t)]^2\\ \text{variance}(g) &= \frac{1}{NM}\sum_t\sum_i [g_i(x^t) - \bar{g}(x^t)]^2\end{aligned}$$

Let us see some models of different complexity: The simplest is a constant fit

$$g_i(x) = 2$$

This has no variance because we do not use the data and all $g_i(x)$ are the same. But the bias is high, unless of course $f(x)$ is close to 2 for all $x$. If we take the average of $r^t$ in the sample

$$g_i(x) = \sum_t r_i^t / N$$

instead of the constant 2, this decreases the bias because we would expect the average in general to be a better estimate. But this increases the variance because the different samples $\mathcal{X}_i$ would have different average values. Normally in this case the decrease in bias would be larger than the increase in variance, and error would decrease.

In the context of polynomial regression, an example is given in figure 4.5. As the order of the polynomial increases, small changes in the dataset cause a greater change in the fitted polynomials; thus variance increases. But a complex model on the average allows a better fit to the underlying function; thus bias decreases (see figure 4.6). This is called the *bias/variance dilemma* and is true for any machine learning system and not only for polynomial regression (Geman, Bienenstock, and Doursat 1992). To decrease bias, the model should be flexible, at the risk of having high variance. If the variance is kept low, we may not be able to make a good fit to data and have high bias. The optimal model is the one that has the best trade-off between the bias and the variance.

BIAS/VARIANCE DILEMMA

If there is bias, this indicates that our model class does not contain the solution; this is *underfitting*. If there is variance, the model class is too general and also learns the noise; this is *overfitting*. If $g(\cdot)$ is of the same hypothesis class with $f(\cdot)$, for example, a polynomial of the same order, we have an unbiased estimator, and estimated bias decreases as the number of models increases. This shows the error-reducing effect of choosing the right model (which we called *inductive bias* in chapter 2—the two uses of "bias" are different but not unrelated). As for variance, it also depends on the size of the training set; the variability due to sample decreases as the sample size increases. To sum up, to get a small value of error, we should have the proper inductive bias (to get small bias in the statistical sense) and have a large enough dataset so that the variability of the model can be constrained with the data.

UNDERFITTING
OVERFITTING

Note that when the variance is large, bias is low: This indicates that $\overline{g}(x)$ is a good estimator. So to get a small value of error, we can take a large number of high-variance models and use their average as our estimator. We discuss such approaches for model combination in chapter 17.

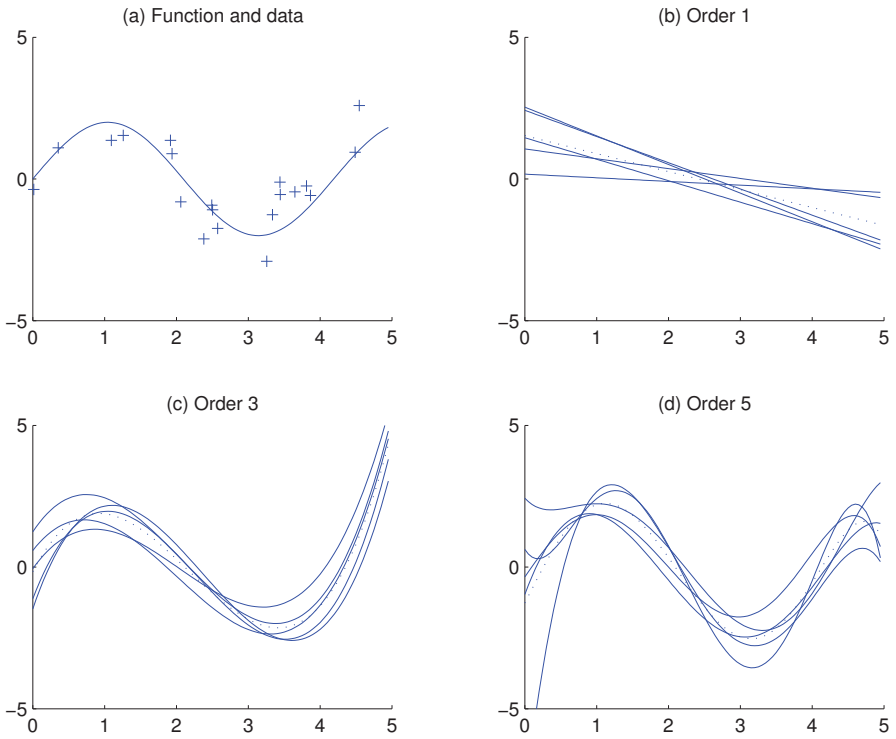


**Figure 4.5** (a) Function, $f(x) = 2\sin(1.5x)$, and one noisy ($\mathcal{N}(0,1)$) dataset sampled from the function. Five samples are taken, each containing twenty instances. (b), (c), (d) are five polynomial fits, namely, $g_i(\cdot)$, of order 1, 3, and 5. For each case, dotted line is the average of the five fits, namely, $\overline{g}(\cdot)$.

## 4.8 Model Selection Procedures

There are a number of procedures we can use to fine-tune model complexity.

CROSS-VALIDATION

In practice, the method we use to find the optimal complexity is *cross-validation*. We cannot calculate the bias and variance for a model, but we can calculate the total error. Given a dataset, we divide it into two parts as training and validation sets, train candidate models of different complexities on the training set and test their error on the validation set left out during training. As the model complexity increases, training error keeps decreasing. The error on the validation set however decreases up to

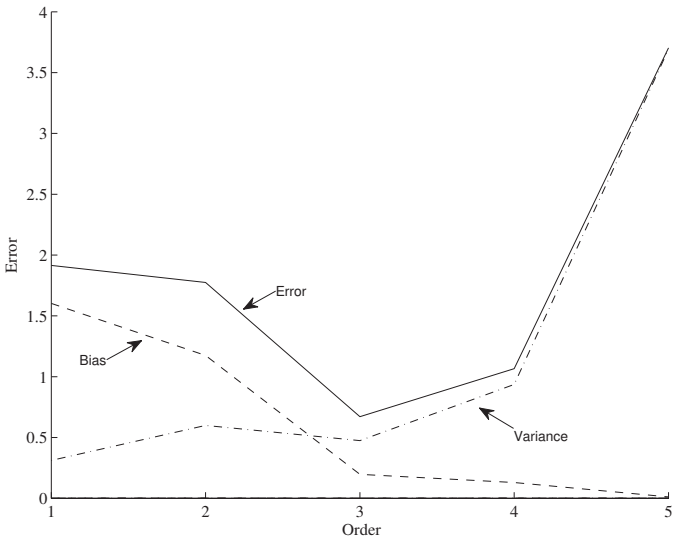


**Figure 4.6** In the same setting as that of figure 4.5, using one hundred models instead of five, bias, variance, and error for polynomials of order 1 to 5. Order 1 has the smallest variance. Order 5 has the smallest bias. As the order is increased, bias decreases but variance increases. Order 3 has the minimum error.

a certain level of complexity, then stops decreasing or does not decrease further significantly, or even increases if there is noise in the data. This "elbow" corresponds to the optimal complexity level (see figure 4.7).

In real life, we cannot calculate the bias and hence the error as we do in figure 4.6; the validation error in figure 4.7 is an estimate of that except that it also contains the variance of the noise: Even if we have the right model where there is no bias and large enough data that variance is negligible, there may still be nonzero validation error. Note that the validation error of figure 4.7 is not as V-shaped as the error of figure 4.6 because the former uses more training data and we know that we can constrain variance with more data. Indeed we see in figure 4.5d that even the fifth-order polynomial behaves like a third-order where there is data—note that at the two extremes where there are fewer data points, it is not as accurate.

REGULARIZATION Another approach that is used frequently is *regularization* (Breiman

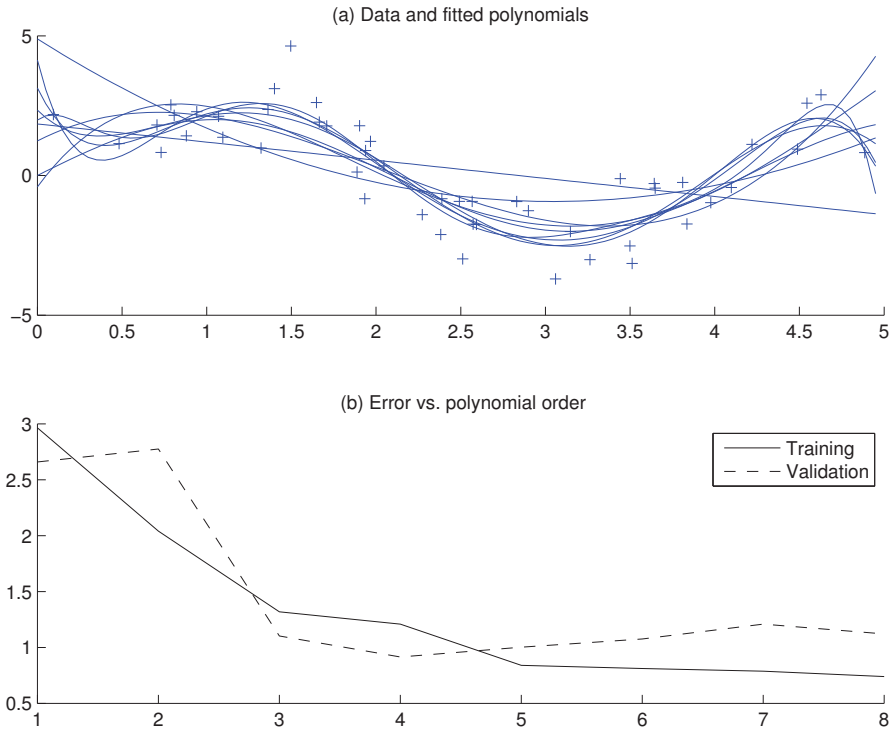


**Figure 4.7** In the same setting as that of figure 4.5, training and validation sets (each containing 50 instances) are generated. (a) Training data and fitted polynomials of order from 1 to 8. (b) Training and validation errors as a function of the polynomial order. The "elbow" is at 3.

1998). In this approach, we write an *augmented error function*

$$E' = \text{error on data} + \lambda \cdot \text{model complexity} \tag{4.37}$$

This has a second term that penalizes complex models with large variance, where $\lambda$ gives the weight of this penalty. When we minimize the augmented error function instead of the error on data only, we penalize complex models and thus decrease variance. If $\lambda$ is taken too large, only very simple models are allowed and we risk introducing bias. $\lambda$ is optimized using cross-validation.

Another way we can view equation 4.37 is by regarding $E'$ as the error on new test data. The first term on the right is the training error and the

second is an *optimism* term estimating the discrepancy between training and test error (Hastie, Tibshirani, and Friedman 2011). Methods such as *Akaike's information criterion* (AIC) and *Bayesian information criterion* (BIC) work by estimating this optimism and adding it to the training error to estimate test error, without any need for validation. The magnitude of this optimism term increases linearly with $d$, the number of inputs (here, it is $k+1$), and decreases as $N$, training set size, increases; it also increases with $\sigma^2$, the variance of the noise added (which we can estimate from the error of a low-bias model). For models that are not linear, $d$ should be replaced with the "effective" number of parameters.

AIC
BIC

STRUCTURAL RISK MINIMIZATION

*Structural risk minimization* (SRM) (Vapnik 1995) uses a set of models ordered in terms of their complexities. An example is polynomials of increasing order. The complexity is generally given by the number of free parameters. VC dimension is another measure of model complexity. In equation 4.37, we can have a set of decreasing $\lambda_i$ to get a set of models ordered in increasing complexity. Model selection by SRM then corresponds to finding the model simplest in terms of order and best in terms of empirical error on the data.

MINIMUM DESCRIPTION LENGTH

*Minimum description length* (MDL) (Rissanen 1978; Grünwald 2007) is based on an information theoretic measure. *Kolmogorov complexity* of a dataset is defined as the shortest description of the data. If the data is simple, it has a short complexity; for example, if it is a sequence of '0's, we can just write '0' and the length of the sequence. If the data is completely random, we cannot have any description of the data shorter than the data itself. If a model is appropriate for the data, then it has a good fit to the data, and instead of the data, we can send/store the model description. Out of all the models that describe the data, we want to have the simplest model so that it lends itself to the shortest description. So we again have a trade-off between how simple the model is and how well it explains the data.

BAYESIAN MODEL SELECTION

*Bayesian model selection* is used when we have some prior knowledge about the appropriate class of approximating functions. This prior knowledge is defined as a prior distribution over models, $p(\text{model})$. Given the data and assuming a model, we can calculate $p(\text{model}|\text{data})$ using Bayes' rule:

$$p(\text{model}|\text{data}) = \frac{p(\text{data}|\text{model})p(\text{model})}{p(\text{data})} \tag{4.38}$$

$p(\text{model}|\text{data})$ is the posterior probability of the model given our prior

subjective knowledge about models, namely, $p(\text{model})$, and the objective support provided by the data, namely, $p(\text{data}|\text{model})$. We can then choose the model with the highest posterior probability, or take an average over all models weighted by their posterior probabilities. We will talk about the Bayesian approach in detail in chapter 16.

If we take the log of equation 4.38, we get

$$\log p(\text{model}|\text{data}) = \log p(\text{data}|\text{model}) + \log p(\text{model}) - c \tag{4.39}$$

which has the form of equation 4.37; the log likelihood of the data is the training error and the log of the prior is the penalty term. For example, if we have a regression model and use the prior $p(\boldsymbol{w}) \sim \mathcal{N}(0, 1/\lambda)$, we minimize

$$E = \sum_t [r^t - g(x^t|\boldsymbol{w})]^2 + \lambda \sum_i w_i^2 \tag{4.40}$$

That is, we look for $w_i$ that both decrease error and are also as close as possible to 0, and the reason we want them close to 0 is because the fitted polynomial will be smoother. As the polynomial order increases, to get a better fit to the data, the function will go up and down, which will mean coefficients moving away from 0 (see figure 4.8); when we add this penalty, we force a flatter, smoother fit. How much we penalize depends on $\lambda$, which is the inverse of the variance of the prior—that is, how much we expect the weights a priori to be away from 0. In other words, having such a prior is equivalent to forcing parameters to be close to 0. We discuss this in greater detail in chapter 16.

That is, when the prior is chosen such that we give higher probabilities to simpler models (following Occam's razor), the Bayesian approach, regularization, SRM, and MDL are equivalent. Cross-validation is different from all other methods for model selection in that it makes no prior assumption about the model or parameters. If there is a large enough validation dataset, it is the best approach. The other models become useful when the data sample is small.

## 4.9 Notes

A good source on the basics of maximum likelihood and Bayesian estimation is Ross 1987. Many pattern recognition textbooks discuss classification with parametric models (e.g., MacLachlan 1992; Devroye, Györfi, and

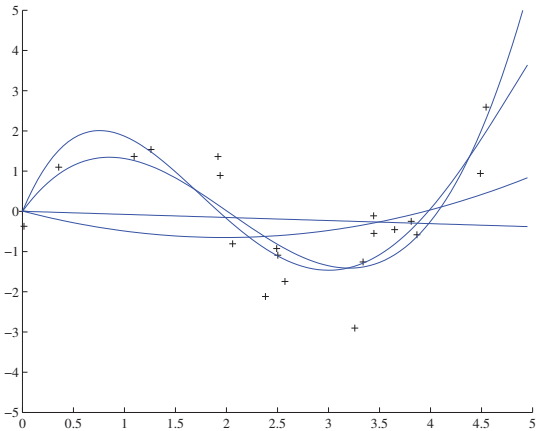


**Figure 4.8** In the same setting as that of figure 4.5, polynomials of order 1 to 4 are fitted. The magnitude of coefficients increase as the order of the polynomial increases. They are as follows: $1 : [-0.0769, 0.0016]^T$, $2 : [0.1682, -0.6657, 0.0080]^T$, $3 : [0.4238, -2.5778, 3.4675, -0.0002]^T$, $4 : [-0.1093, 1.4356, -5.5007, 6.0454, -0.0019]^T$.

Lugosi 1996; Webb and Copsey 2011; Duda, Hart, and Stork 2001). Tests for checking univariate normality can be found in Rencher 1995.

Geman, Bienenstock, and Doursat (1992) discuss bias and variance decomposition for several learning models, which we discuss in later chapters. Bias/variance decomposition is for sum of squared loss and is for regression; such a nice additive splitting of error into bias, variance and noise is not possible for 0/1 loss, because in classification, there is error only if we accidentally move to the other side of the boundary. For a two-class problem, if the correct posterior is 0.7 and if our estimate is 0.8, there is no error; we have error only if our estimate is less than 0.5. Various researchers proposed different definitions of bias and variance for classification; see Friedman 1997 for a review.

## 4.10 Exercises

1. Write the code that generates a Bernoulli sample with given parameter $p$, and the code that calculates $\hat{p}$ from the sample.
2. Write the log likelihood for a multinomial sample and show equation 4.6.

3. Write the code that generates a normal sample with given $\mu$ and $\sigma$, and the code that calculates $m$ and $s$ from the sample. Do the same using the Bayes' estimator assuming a prior distribution for $\mu$.
4. Given two normal distributions $p(x|C_1) \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $p(x|C_2) \sim \mathcal{N}(\mu_2, \sigma_2^2)$ and $P(C_1)$ and $P(C_2)$, calculate the Bayes' discriminant points analytically.
   SOLUTION: Given that

$$\begin{aligned} p(x|C_1) &\sim \mathcal{N}(\mu_1, \sigma_1^2) = \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left[-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right] \\ p(x|C_2) &\sim \mathcal{N}(\mu_2, \sigma_2^2) \end{aligned}$$

   we would like to find $x$ that satisfy $P(C_1|x) = P(C_2|x)$, or

$$\begin{aligned} p(x|C_1)P(C_1) &= p(x|C_2)P(C_2) \\ \log p(x|C_1) + \log P(C_1) &= \log p(x|C_2) + \log P(C_2) \\ -\frac{1}{2}\log 2\pi - \log\sigma_1 - \frac{(x-\mu_1)^2}{2\sigma_1^2} + \log P(C_1) &= \cdots \\ -\log\sigma_1 - \frac{1}{2\sigma_1^2}\left(x^2 - 2x\mu_1 + \mu_1^2\right) + \log P(C_1) &= \cdots \\ \left(\frac{1}{2\sigma_2^2} - \frac{1}{2\sigma_1^2}\right)x^2 + \left(\frac{\mu_1}{\sigma_1^2} - \frac{\mu_2}{\sigma_2^2}\right)x + & \\ \left(\frac{\mu_2^2}{2\sigma_2^2} - \frac{\mu_1^2}{2\sigma_1^2}\right) + \log\frac{\sigma_2}{\sigma_1} + \log\frac{P(C_1)}{P(C_2)} &= 0 \end{aligned}$$

   This is of the form $ax^2 + bx + c = 0$ and the two roots are

$$x_1, x_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

   Note that if the variances are equal, the quadratic terms vanishes and there is one root, that is, the two posteriors intersect at a single $x$ value.
5. What is the likelihood ratio

$$\frac{p(x|C_1)}{p(x|C_2)}$$

   in the case of Gaussian densities?
   SOLUTION:

$$\frac{p(x|C_1)}{p(x|C_2)} = \frac{\frac{1}{\sqrt{2\pi}\sigma_1}\exp\left[-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right]}{\frac{1}{\sqrt{2\pi}\sigma_2}\exp\left[-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right]}$$

   If we have $\sigma_1^2 = \sigma_2^2 = \sigma^2$, we can simplify

$$\frac{p(x|C_1)}{p(x|C_2)} = \exp\left[-\frac{(x-\mu_1)^2}{2\sigma^2} + \frac{(x-\mu_2)^2}{2\sigma^2}\right]$$

$$= \exp\left[\frac{(\mu_1-\mu_2)}{\sigma^2}x + \frac{(\mu_2^2-\mu_1^2)}{2\sigma^2}\right]$$
$$= \exp(wx + w_0)$$

for $w = (\mu_1 - \mu_2)/\sigma^2$ and $w_0 = (\mu_2^2 - \mu_1^2)/2\sigma^2$.

6. For a two-class problem, generate normal samples for two classes with different variances, then use parametric classification to estimate the discriminant points. Compare these with the theoretical values.
7. Assume a linear model and then add 0-mean Gaussian noise to generate a sample. Divide your sample into two as training and validation sets. Use linear regression using the training half. Compute error on the validation set. Do the same for polynomials of degrees 2 and 3 as well.
8. When the training set is small, the contribution of variance to error may be more than that of bias and in such a case, we may prefer a simple model even though we know that it is too simple for the task. Can you give an example?
9. Let us say, given the samples $\mathcal{X}_i = \{x_i^t, r_i^t\}$, we define $g_i(x) = r_i^1$, namely, our estimate for any $x$ is the $r$ value of the first instance in the (unordered) dataset $\mathcal{X}_i$. What can you say about its bias and variance, as compared with $g_i(x) = 2$ and $g_i(x) = \sum_t r_i^t/N$? What if the sample is ordered, so that $g_i(x) = \min_t r_i^t$?
10. In equation 4.40, what is the effect of changing $\lambda$ on bias and variance?

    SOLUTION: $\lambda$ controls smoothness: If it is large, we may smooth too much and decrease variance at the expense of an increase in bias; if it is small, bias may be small but variance will be high.

## 4.11 References


Andrieu, C., N. de Freitas, A. Doucet, and M. I. Jordan. 2003. "An Introduction to MCMC for Machine Learning." *Machine Learning* 50:5–43.

Breiman, L. 1998. "Bias-Variance, Regularization, Instability and Stabilization." In *Neural Networks and Machine Learning*, ed. C. M. Bishop, 27–56. Berlin: Springer.

Devroye, L., L. Györfi, and G. Lugosi. 1996. *A Probabilistic Theory of Pattern Recognition*. New York: Springer.

Duda, R. O., P. E. Hart, and D. G. Stork. 2001. *Pattern Classification*, 2nd ed. New York: Wiley.

Friedman, J. H. 1997. "On Bias, Variance, 0/1-Loss and the Curse of Dimensionality." *Data Mining and Knowledge Discovery* 1:55–77.

Geman, S., E. Bienenstock, and R. Doursat. 1992. "Neural Networks and the Bias/Variance Dilemma." *Neural Computation* 4:1–58.

Grünwald, P. D. 2007. *The Minimum Description Length Principle.* Cambridge, MA: MIT Press.

Hastie, T., R. Tibshirani, and J. Friedman. 2011. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction,* 2nd ed. New York: Springer.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition.* New York: Wiley.

Rencher, A. C. 1995. *Methods of Multivariate Analysis.* New York: Wiley.

Rissanen, J. 1978. "Modeling by Shortest Data Description." *Automatica* 14:465–471.

Ross, S. M. 1987. *Introduction to Probability and Statistics for Engineers and Scientists.* New York: Wiley.

Vapnik, V. 1995. *The Nature of Statistical Learning Theory.* New York: Springer.

Webb, A., and K. D. Copsey. 2011. *Statistical Pattern Recognition,* 3rd ed. New York: Wiley.

# 5 *Multivariate Methods*

*In chapter 4, we discussed the parametric approach to classification and regression. Now, we generalize this to the multivariate case where we have multiple inputs and where the output, which is class code or continuous output, is a function of these multiple inputs. These inputs may be discrete or numeric. We will see how such functions can be learned from a labeled multivariate sample and also how the complexity of the learner can be fine-tuned to the data at hand.*

## 5.1 Multivariate Data

IN MANY APPLICATIONS, several measurements are made on each individual or event generating an observation vector. The sample may be viewed as a *data matrix*

$$\mathbf{X} = \begin{bmatrix} X_1^1 & X_2^1 & \cdots & X_d^1 \\ X_1^2 & X_2^2 & \cdots & X_d^2 \\ \vdots & & & \\ X_1^N & X_2^N & \cdots & X_d^N \end{bmatrix}$$

where the $d$ columns correspond to $d$ *variables* denoting the result of measurements made on an individual or event. These are also called *inputs*, *features*, or *attributes*. The $N$ rows correspond to independent and identically distributed *observations*, *examples*, or *instances* on $N$ individuals or events.

INPUT
FEATURE
ATTRIBUTE
OBSERVATION
EXAMPLE
INSTANCE

For example, in deciding on a loan application, an observation vector is the information associated with a customer and is composed of age, marital status, yearly income, and so forth, and we have $N$ such past

customers. These measurements may be of different scales, for example, age in years and yearly income in monetary units. Some like age may be numeric, and some like marital status may be discrete.

Typically these variables are correlated. If they are not, there is no need for a multivariate analysis. Our aim may be *simplification*, that is, summarizing this large body of data by means of relatively few parameters. Or our aim may be *exploratory*, and we may be interested in generating hypotheses about data. In some applications, we are interested in predicting the value of one variable from the values of other variables. If the predicted variable is discrete, this is multivariate classification, and if it is numeric, this is a multivariate regression problem.

## 5.2 Parameter Estimation

MEAN VECTOR

The *mean vector* $\boldsymbol{\mu}$ is defined such that each of its elements is the mean of one column of $\mathbf{X}$:

$$E[\boldsymbol{x}] = \boldsymbol{\mu} = [\mu_1, \ldots, \mu_d]^T \tag{5.1}$$

The variance of $X_i$ is denoted as $\sigma_i^2$, and the covariance of two variables $X_i$ and $X_j$ is defined as

$$\sigma_{ij} \equiv \text{Cov}(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)] = E[X_i X_j] - \mu_i \mu_j \tag{5.2}$$

with $\sigma_{ij} = \sigma_{ji}$, and when $i = j$, $\sigma_{ii} = \sigma_i^2$. With $d$ variables, there are $d$ variances and $d(d-1)/2$ covariances, which are generally represented as

COVARIANCE MATRIX

a $d \times d$ matrix, named the *covariance matrix*, denoted as $\boldsymbol{\Sigma}$, whose $(i, j)$th element is $\sigma_{ij}$:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1d} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2d} \\ \vdots & & & \\ \sigma_{d1} & \sigma_{d2} & \cdots & \sigma_d^2 \end{bmatrix}$$

The diagonal terms are the variances, the off-diagonal terms are the covariances, and the matrix is symmetric. In vector-matrix notation

$$\boldsymbol{\Sigma} \equiv \text{Cov}(\boldsymbol{X}) = E[(\boldsymbol{X} - \boldsymbol{\mu})(\boldsymbol{X} - \boldsymbol{\mu})^T] = E[\boldsymbol{X}\boldsymbol{X}^T] - \boldsymbol{\mu}\boldsymbol{\mu}^T \tag{5.3}$$

If two variables are related in a linear way, then the covariance will be positive or negative depending on whether the relationship has a positive

or negative slope. But the size of the relationship is difficult to interpret because it depends on the units in which the two variables are measured.

CORRELATION

The *correlation* between variables $X_i$ and $X_j$ is a statistic normalized between $-1$ and $+1$, defined as

$$\text{Corr}(X_i, X_j) \equiv \rho_{ij} = \frac{\sigma_{ij}}{\sigma_i \sigma_j} \tag{5.4}$$

If two variables are independent, then their covariance, and hence their correlation, is 0. However, the converse is not true: The variables may be dependent (in a nonlinear way), and their correlation may be 0.

SAMPLE MEAN

Given a multivariate sample, estimates for these parameters can be calculated: The maximum likelihood estimator for the mean is the *sample mean*, $\boldsymbol{m}$. Its $i$th dimension is the average of the $i$th column of $\mathbf{X}$:

$$\boldsymbol{m} = \frac{\sum_{t=1}^{N} \boldsymbol{x}^t}{N} \text{ with } m_i = \frac{\sum_{t=1}^{N} x_i^t}{N}, i = 1, \ldots, d \tag{5.5}$$

SAMPLE COVARIANCE

The estimator of $\boldsymbol{\Sigma}$ is $\mathbf{S}$, the *sample covariance* matrix, with entries

$$s_i^2 = \frac{\sum_{t=1}^{N} (x_i^t - m_i)^2}{N} \tag{5.6}$$

$$s_{ij} = \frac{\sum_{t=1}^{N} (x_i^t - m_i)(x_j^t - m_j)}{N} \tag{5.7}$$

These are biased estimates, but if in an application the estimates vary significantly depending on whether we divide by $N$ or $N - 1$, we are in serious trouble anyway.

SAMPLE CORRELATION

The *sample correlation* coefficients are

$$r_{ij} = \frac{s_{ij}}{s_i s_j} \tag{5.8}$$

and the sample correlation matrix $\mathbf{R}$ contains $r_{ij}$.

## 5.3 Estimation of Missing Values

Frequently, values of certain variables may be missing in observations. The best strategy is to discard those observations all together, but generally we do not have large enough samples to be able to afford this and we do not want to lose data as the non-missing entries do contain information. We try to fill in the missing entries by estimating them. This is called *imputation*.

IMPUTATION

In *mean imputation*, for a numeric variable, we substitute the mean (average) of the available data for that variable in the sample. For a discrete variable, we fill in with the most likely value, that is, the value most often seen in the data.

In *imputation by regression*, we try to predict the value of a missing variable from other variables whose values are known for that case. Depending on the type of the missing variable, we define a separate regression or classification problem that we train by the data points for which such values are known. If many different variables are missing, we take the means as the initial estimates and the procedure is iterated until predicted values stabilize. If the variables are not highly correlated, the regression approach is equivalent to mean imputation.

Depending on the context, however, sometimes the fact that a certain attribute value is missing may be important. For example, in a credit card application, if the applicant does not declare his or her telephone number, that may be a critical piece of information. In such cases, this is represented as a separate value to indicate that the value is missing and is used as such.

## 5.4 Multivariate Normal Distribution

In the multivariate case where $\boldsymbol{x}$ is $d$-dimensional and normal distributed, we have

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right] \tag{5.9}$$

and we write $\boldsymbol{x} \sim \mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where $\boldsymbol{\mu}$ is the mean vector and $\boldsymbol{\Sigma}$ is the covariance matrix (see figure 5.1). Just as

$$\frac{(x-\mu)^2}{\sigma^2} = (x-\mu)(\sigma^2)^{-1}(x-\mu)$$

is the squared distance from $x$ to $\mu$ in standard deviation units, normalizing for different variances, in the multivariate case the *Mahalanobis distance* is used:

MAHALANOBIS DISTANCE

$$(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) \tag{5.10}$$

$(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) = c^2$ is the $d$-dimensional hyperellipsoid centered at $\boldsymbol{\mu}$, and its shape and orientation are defined by $\boldsymbol{\Sigma}$. Because of the use of the inverse of $\boldsymbol{\Sigma}$, if a variable has a larger variance than another, it receives

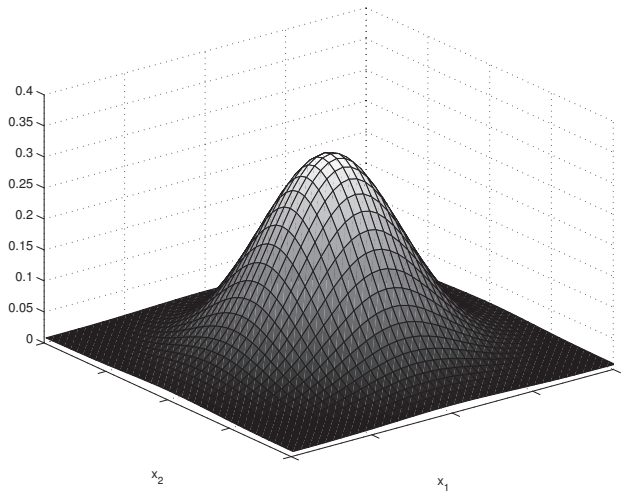


**Figure 5.1** Bivariate normal distribution.

less weight in the Mahalanobis distance. Similarly, two highly correlated variables do not contribute as much as two less correlated variables. The use of the inverse of the covariance matrix thus has the effect of standardizing all variables to unit variance and eliminating correlations.

Let us consider the bivariate case where $d = 2$ for visualization purposes (see figure 5.2). When the variables are independent, the major axes of the density are parallel to the input axes. The density becomes an ellipse if the variances are different. The density rotates depending on the sign of the covariance (correlation). The mean vector is $\boldsymbol{\mu}^T = [\mu_1, \mu_2]$, and the covariance matrix is usually expressed as

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$$

The joint bivariate density can be expressed in the form (see exercise 1)

$$p(x_1, x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[-\frac{1}{2(1-\rho^2)}\left(z_1^2 - 2\rho z_1 z_2 + z_2^2\right)\right] \tag{5.11}$$

where $z_i = (x_i - \mu_i)/\sigma_i, i = 1, 2$, are standardized variables; this is called *z-normalization*. Remember that

Z-NORMALIZATION

$$z_1^2 + 2\rho z_1 z_2 + z_2^2 = \text{constant}$$

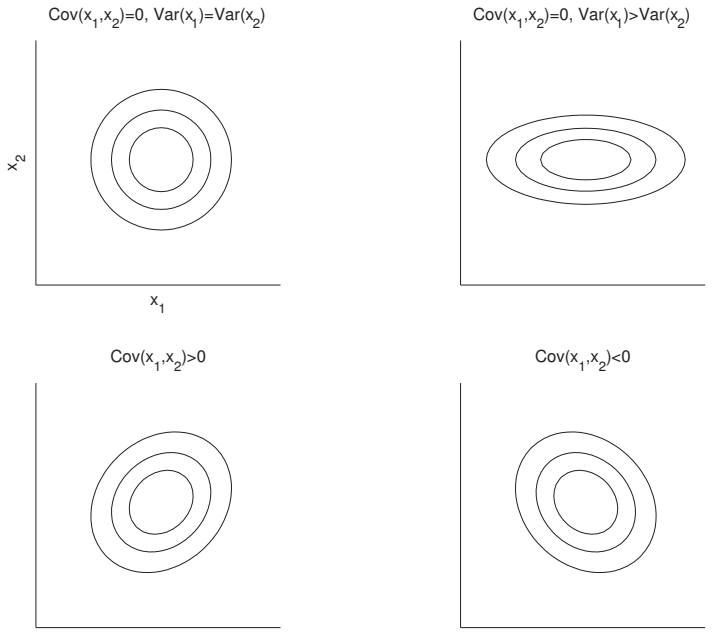


**Figure 5.2** Isoprobability contour plot of the bivariate normal distribution. Its center is given by the mean, and its shape and orientation depend on the covariance matrix.

for $|\rho| < 1$, is the equation of an ellipse. When $\rho > 0$, the major axis of the ellipse has a positive slope and if $\rho < 0$, the major axis has a negative slope.

In the expanded Mahalanobis distance of equation 5.11, each variable is normalized to have unit variance, and there is the cross-term that corrects for the correlation between the two variables.

The density depends on five parameters: the two means, the two variances, and the correlation. $\boldsymbol{\Sigma}$ is nonsingular, and hence positive definite, provided that variances are nonzero and $|\rho| < 1$. If $\rho$ is $+1$ or $-1$, the two variables are linearly related, the observations are effectively one-dimensional, and one of the two variables can be disposed of. If $\rho = 0$, then the two variables are independent, the cross-term disappears, and we get a product of two univariate densities.

In the multivariate case, a small value of $|\boldsymbol{\Sigma}|$ indicates samples are close to $\boldsymbol{\mu}$, just as in the univariate case where a small value of $\sigma^2$ indicates

samples are close to $\mu$. Small $|\Sigma|$ may also indicate that there is high correlation between variables. $\Sigma$ is a symmetric positive definite matrix; this is the multivariate way of saying that $\text{Var}(X) > 0$. If not so, $\Sigma$ is singular and its determinant is 0. This is either due to linear dependence between the dimensions or because one of the dimensions has variance 0. In such a case, dimensionality should be reduced to a get a positive definite matrix; we discuss methods for this in chapter 6.

If $\boldsymbol{x} \sim \mathcal{N}_d(\boldsymbol{\mu}, \Sigma)$, then each dimension of $\boldsymbol{x}$ is univariate normal. (The converse is not true: Each $X_i$ may be univariate normal and $\boldsymbol{X}$ may not be multivariate normal.) Actually any $k < d$ subset of the variables is $k$-variate normal.

A special, *naive* case is where the components of $\boldsymbol{x}$ are independent and $\text{Cov}(X_i, X_j) = 0$, for $i \neq j$, and $\text{Var}(X_i) = \sigma_i^2$, $\forall i$. Then the covariance matrix is diagonal and the joint density is the product of the individual univariate densities:

$$p(\boldsymbol{x}) = \prod_{i=1}^{d} p_i(x_i) = \frac{1}{(2\pi)^{d/2} \prod_{i=1}^{d} \sigma_i} \exp\left[-\frac{1}{2}\sum_{i=1}^{d}\left(\frac{x_i - \mu_i}{\sigma_i}\right)^2\right] \tag{5.12}$$

Now let us see another property we make use of in later chapters. Let us say $\boldsymbol{x} \sim \mathcal{N}_d(\boldsymbol{\mu}, \Sigma)$ and $\boldsymbol{w} \in \Re^d$, then

$$\boldsymbol{w}^T\boldsymbol{x} = w_1x_1 + w_2x_2 + \cdots + w_dx_d \sim \mathcal{N}(\boldsymbol{w}^T\boldsymbol{\mu}, \boldsymbol{w}^T\Sigma\boldsymbol{w})$$

given that

$$\begin{aligned} E[\boldsymbol{w}^T\boldsymbol{x}] &= \boldsymbol{w}^T E[\boldsymbol{x}] = \boldsymbol{w}^T\boldsymbol{\mu} && (5.13)\\ \text{Var}(\boldsymbol{w}^T\boldsymbol{x}) &= E[(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})^2] = E[(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})] \\ &= E[\boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{w}] = \boldsymbol{w}^T E[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T]\boldsymbol{w} \\ &= \boldsymbol{w}^T\Sigma\boldsymbol{w} && (5.14) \end{aligned}$$

That is, the projection of a $d$-dimensional normal on the vector $\boldsymbol{w}$ is univariate normal. In the general case, if $\mathbf{W}$ is a $d \times k$ matrix with rank $k < d$, then the $k$-dimensional $\mathbf{W}^T\boldsymbol{x}$ is $k$-variate normal:

$$\mathbf{W}^T\boldsymbol{x} \sim \mathcal{N}_k(\mathbf{W}^T\boldsymbol{\mu}, \mathbf{W}^T\Sigma\mathbf{W}) \tag{5.15}$$

That is, if we project a $d$-dimensional normal distribution to a space that is $k$-dimensional, then it projects to a $k$-dimensional normal.

## 5.5 Multivariate Classification

When $\boldsymbol{x} \in \Re^d$, if the class-conditional densities, $p(\boldsymbol{x}|C_i)$, are taken as normal density, $\mathcal{N}_d(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, we have

$$p(\boldsymbol{x}|C_i) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}_i|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right] \tag{5.16}$$

The main reason for this is its analytical simplicity (Duda, Hart, and Stork 2001). Besides, the normal density is a model for many naturally occurring phenomena in that examples of most classes can be seen as mildly changed versions of a single prototype, $\boldsymbol{\mu}_i$, and the covariance matrix, $\boldsymbol{\Sigma}_i$, denotes the amount of noise in each variable and the correlations of these noise sources. While real data may not often be exactly multivariate normal, it is a useful approximation. In addition to its mathematical tractability, the model is robust to departures from normality as is shown in many works (e.g., McLachlan 1992). However, one clear requirement is that the sample of a class should form a single group; if there are multiple groups, one should use a mixture model (chapter 7).

Let us say we want to predict the type of a car that a customer would be interested in. Different cars are the classes and $\boldsymbol{x}$ are observable data of customers, for example, age and income. $\boldsymbol{\mu}_i$ is the vector of mean age and income of customers who buy car type $i$ and $\boldsymbol{\Sigma}_i$ is their covariance matrix: $\sigma_{i1}^2$ and $\sigma_{i2}^2$ are the age and income variances, and $\sigma_{i12}$ is the covariance of age and income in the group of customers who buy car type $i$.

When we define the discriminant function as

$$g_i(\boldsymbol{x}) = \log p(\boldsymbol{x}|C_i) + \log P(C_i)$$

and assuming $p(\boldsymbol{x}|C_i) \sim \mathcal{N}_d(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, we have

$$g_i(\boldsymbol{x}) = -\frac{d}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}_i| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) + \log P(C_i) \tag{5.17}$$

Given a training sample for $K \geq 2$ classes, $\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{r}^t\}$, where $r_i^t = 1$ if $\boldsymbol{x}^t \in C_i$ and 0 otherwise, estimates for the means and covariances are found using maximum likelihood separately for each class:

$$\begin{aligned} \hat{P}(C_i) &= \frac{\sum_t r_i^t}{N} \\ \boldsymbol{m}_i &= \frac{\sum_t r_i^t \boldsymbol{x}^t}{\sum_t r_i^t} \\ \mathbf{S}_i &= \frac{\sum_t r_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T}{\sum_t r_i^t} \end{aligned} \tag{5.18}$$

These are then plugged into the discriminant function to get the estimates for the discriminants. Ignoring the first constant term, we have

$$g_i(\boldsymbol{x}) = -\frac{1}{2}\log|\mathbf{S}_i| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_i)^T\mathbf{S}_i^{-1}(\boldsymbol{x}-\boldsymbol{m}_i) + \log\hat{P}(C_i) \tag{5.19}$$

Expanding this, we get

$$g_i(\boldsymbol{x}) = -\frac{1}{2}\log|\mathbf{S}_i| - \frac{1}{2}\left(\boldsymbol{x}^T\mathbf{S}_i^{-1}\boldsymbol{x} - 2\boldsymbol{x}^T\mathbf{S}_i^{-1}\boldsymbol{m}_i + \boldsymbol{m}_i^T\mathbf{S}_i^{-1}\boldsymbol{m}_i\right) + \log\hat{P}(C_i)$$

QUADRATIC DISCRIMINANT

which defines a *quadratic discriminant* (see figure 5.3) that can also be written as

$$g_i(\boldsymbol{x}) = \boldsymbol{x}^T\mathbf{W}_i\boldsymbol{x} + \boldsymbol{w}_i^T\boldsymbol{x} + w_{i0} \tag{5.20}$$

where

$$\begin{aligned}\mathbf{W}_i &= -\frac{1}{2}\mathbf{S}_i^{-1}\\ \boldsymbol{w}_i &= \mathbf{S}_i^{-1}\boldsymbol{m}_i\\ w_{i0} &= -\frac{1}{2}\boldsymbol{m}_i^T\mathbf{S}_i^{-1}\boldsymbol{m}_i - \frac{1}{2}\log|\mathbf{S}_i| + \log\hat{P}(C_i)\end{aligned}$$

The number of parameters to be estimated are $K \cdot d$ for the means and $K \cdot d(d+1)/2$ for the covariance matrices. When $d$ is large and samples are small, $\mathbf{S}_i$ may be singular and inverses may not exist. Or, $|\mathbf{S}_i|$ may be nonzero but too small, in which case it will be unstable; small changes in $\mathbf{S}_i$ will cause large changes in $\mathbf{S}_i^{-1}$. For the estimates to be reliable on small samples, one may want to decrease dimensionality, $d$, by redesigning the feature extractor and select a subset of the features or somehow combine existing features. We discuss such methods in chapter 6.

Another possibility is to pool the data and estimate a common covariance matrix for all classes:

$$\mathbf{S} = \sum_i \hat{P}(C_i)\mathbf{S}_i \tag{5.21}$$

In this case of equal covariance matrices, equation 5.19 reduces to

$$g_i(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_i)^T\mathbf{S}^{-1}(\boldsymbol{x}-\boldsymbol{m}_i) + \log\hat{P}(C_i) \tag{5.22}$$

The number of parameters is $K \cdot d$ for the means and $d(d+1)/2$ for the shared covariance matrix. If the priors are equal, the optimal decision rule is to assign input to the class whose mean's Mahalanobis distance to the input is the smallest. As before, unequal priors shift the boundary

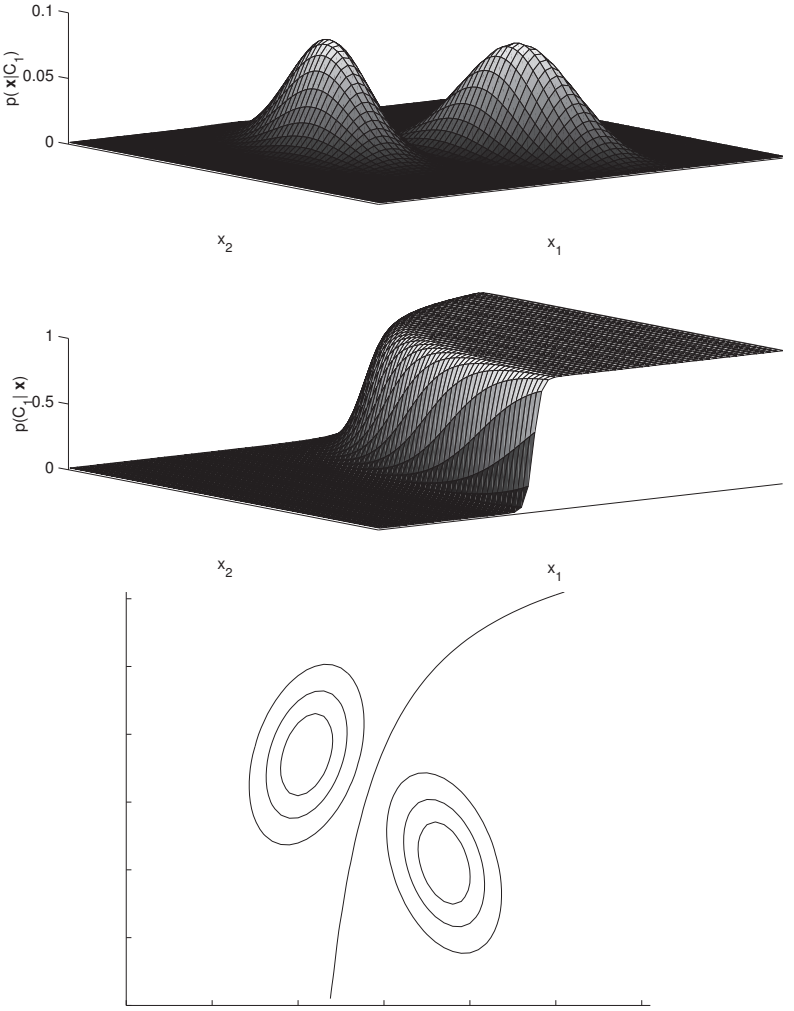


**Figure 5.3** Classes have different covariance matrices. Likelihood densities and the posterior probability for one of the classes (top). Class distributions are indicated by isoprobability contours and the discriminant is drawn (bottom).

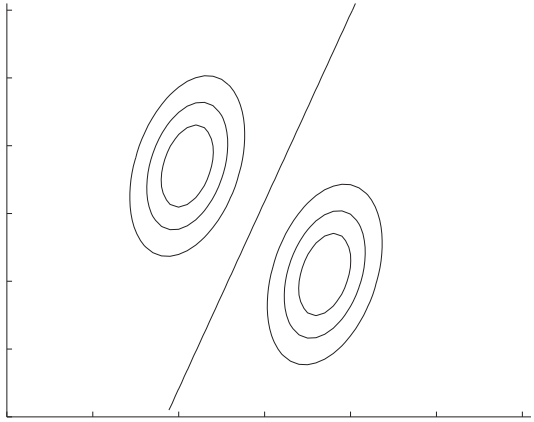

**Figure 5.4** Covariances may be arbitary but shared by both classes.

toward the less likely class. Note that in this case, the quadratic term $\boldsymbol{x}^T\mathbf{S}^{-1}\boldsymbol{x}$ cancels since it is common in all discriminants, and the decision boundaries are linear, leading to a *linear discriminant* (figure 5.4) that can be written as

LINEAR DISCRIMINANT

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T\boldsymbol{x} + w_{i0} \tag{5.23}$$

where

$$\begin{aligned} \boldsymbol{w}_i &= \mathbf{S}^{-1}\boldsymbol{m}_i \\ w_{i0} &= -\frac{1}{2}\boldsymbol{m}_i^T\mathbf{S}^{-1}\boldsymbol{m}_i + \log\hat{P}(C_i) \end{aligned}$$

Decision regions of such a linear classifier are convex; namely, when two points are chosen arbitrarily in one decision region and are connected by a straight line, all the points on the line will lie in the region.

Further simplication may be possible by assuming all off-diagonals of the covariance matrix to be 0, thus assuming independent variables. This is the *naive Bayes' classifier* where $p(x_j|C_i)$ are univariate Gaussian. $\mathbf{S}$ and its inverse are diagonal, and we get

NAIVE BAYES' CLASSIFIER

$$g_i(\boldsymbol{x}) = -\frac{1}{2}\sum_{j=1}^{d}\left(\frac{x_j^t - m_{ij}}{s_j}\right)^2 + \log\hat{P}(C_i) \tag{5.24}$$

The term $(x_j^t - m_{ij})/s_j$ has the effect of normalization and measures the distance in terms of standard deviation units. Geometrically speaking,

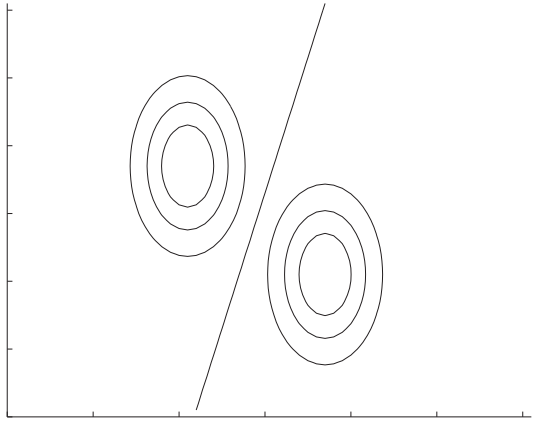

**Figure 5.5** All classes have equal, diagonal covariance matrices, but variances are not equal.

classes are hyperellipsoidal and, because the covariances are zero, are axis-aligned (see figure 5.5). The number of parameters is $K \cdot d$ for the means and $d$ for the variances. Thus the complexity of $\mathbf{S}$ is reduced from $\mathcal{O}(d^2)$ to $\mathcal{O}(d)$.

EUCLIDEAN DISTANCE

Simplifying even further, if we assume all variances to be equal, the Mahalanobis distance reduces to *Euclidean distance*. Geometrically, the distribution is shaped spherically, centered around the mean vector $\boldsymbol{m}_i$ (see figure 5.6). Then $|\mathbf{S}| = s^{2d}$ and $\mathbf{S}^{-1} = (1/s^2)\mathbf{I}$. The number of parameters in this case is $K \cdot d$ for the means and 1 for $s^2$.

$$g_i(\boldsymbol{x}) = -\frac{\|\boldsymbol{x} - \boldsymbol{m}_i\|^2}{2s^2} + \log \hat{P}(C_i) = -\frac{1}{2s^2}\sum_{j=1}^{d}(x_j^t - m_{ij})^2 + \log \hat{P}(C_i) \tag{5.25}$$

NEAREST MEAN CLASSIFIER

TEMPLATE MATCHING

If the priors are equal, we have $g_i(\boldsymbol{x}) = -\|\boldsymbol{x} - \boldsymbol{m}_i\|^2$. This is named the *nearest mean classifier* because it assigns the input to the class of the nearest mean. If each mean is thought of as the ideal prototype or template for the class, this is a *template matching* procedure. This can be expanded as

$$\begin{aligned} g_i(\boldsymbol{x}) &= -\|\boldsymbol{x} - \boldsymbol{m}_i\|^2 = -(\boldsymbol{x} - \boldsymbol{m}_i)^T(\boldsymbol{x} - \boldsymbol{m}_i) \\ &= -(\boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{m}_i^T\boldsymbol{x} + \boldsymbol{m}_i^T\boldsymbol{m}_i) \end{aligned} \tag{5.26}$$

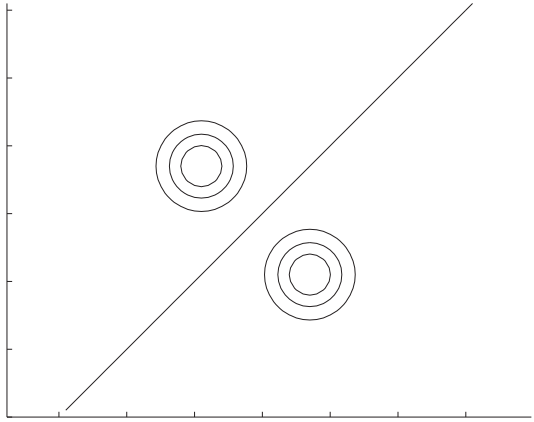

**Figure 5.6** All classes have equal, diagonal covariance matrices of equal variances on both dimensions.

The first term, $\boldsymbol{x}^T\boldsymbol{x}$, is shared in all $g_i(\boldsymbol{x})$ and can be dropped, and we can write the discriminant function as

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T\boldsymbol{x} + w_{i0} \tag{5.27}$$

where $\boldsymbol{w}_i = \boldsymbol{m}_i$ and $w_{i0} = -(1/2)\|\boldsymbol{m}_i\|^2$. If all $\boldsymbol{m}_i$ have similar norms, then this term can also be ignored and we can use

$$g_i(\boldsymbol{x}) = \boldsymbol{m}_i^T\boldsymbol{x} \tag{5.28}$$

When the norms of $\boldsymbol{m}_i$ are comparable, dot product can also be used as the similarity measure instead of the (negative) Euclidean distance.

We can actually think of finding the best discriminant function as the task of finding the best distance function. This can be seen as another approach to classification: Instead of learning the discriminant functions, $g_i(\boldsymbol{x})$, we want to learn the suitable distance function $\mathcal{D}(\boldsymbol{x}_1, \boldsymbol{x}_2)$, such that for any $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$, where $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ belong to the same class, and $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$ belong to two different classes, we would like to have

$$\mathcal{D}(\boldsymbol{x}_1, \boldsymbol{x}_2) < \mathcal{D}(\boldsymbol{x}_1, \boldsymbol{x}_3)$$

**Table 5.1** Reducing variance through simplifying assumptions

| Assumption | Covariance matrix | No. of parameters |
|---|---|---|
| Shared, Hyperspheric | $\mathbf{S}_i = \mathbf{S} = s^2\mathbf{I}$ | 1 |
| Shared, Axis-aligned | $\mathbf{S}_i = \mathbf{S}$, with $s_{ij} = 0$ | $d$ |
| Shared, Hyperellipsoidal | $\mathbf{S}_i = \mathbf{S}$ | $d(d+1)/2$ |
| Different, Hyperellipsoidal | $\mathbf{S}_i$ | $K \cdot (d(d+1)/2)$ |

## 5.6 Tuning Complexity

In table 5.1, we see how the number of parameters of the covariance matrix may be reduced, trading off the comfort of a simple model with generality. This is another example of bias/variance dilemma. When we make simplifying assumptions about the covariance matrices and decrease the number of parameters to be estimated, we risk introducing bias (see figure 5.7). On the other hand, if no such assumption is made and the matrices are arbitrary, the quadratic discriminant may have large variance on small datasets. The ideal case depends on the complexity of the problem represented by the data at hand and the amount of data we have. When we have a small dataset, even if the covariance matrices are different, it may be better to assume a shared covariance matrix; a single covariance matrix has fewer parameters and it can be estimated using more data, that is, instances of all classes. This corresponds to using *linear discriminants*, which is very frequently used in classification and which we discuss in more detail in chapter 10.

Note that when we use Euclidean distance to measure similarity, we are assuming that all variables have the same variance and that they are independent. In many cases, this does not hold; for example, age and yearly income are in different units, and are dependent in many contexts. In such a case, the inputs may be separately $z$-normalized in a preprocessing stage (to have zero mean and unit variance), and then Euclidean distance can be used. On the other hand, sometimes even if the variables are dependent, it may be better to assume that they are independent and to use the naive Bayes' classifier, if we do not have enough data to calculate the dependency accurately.

REGULARIZED DISCRIMINANT ANALYSIS

Friedman (1989) proposed a method that combines all these as special cases, named *regularized discriminant analysis* (RDA). We remember

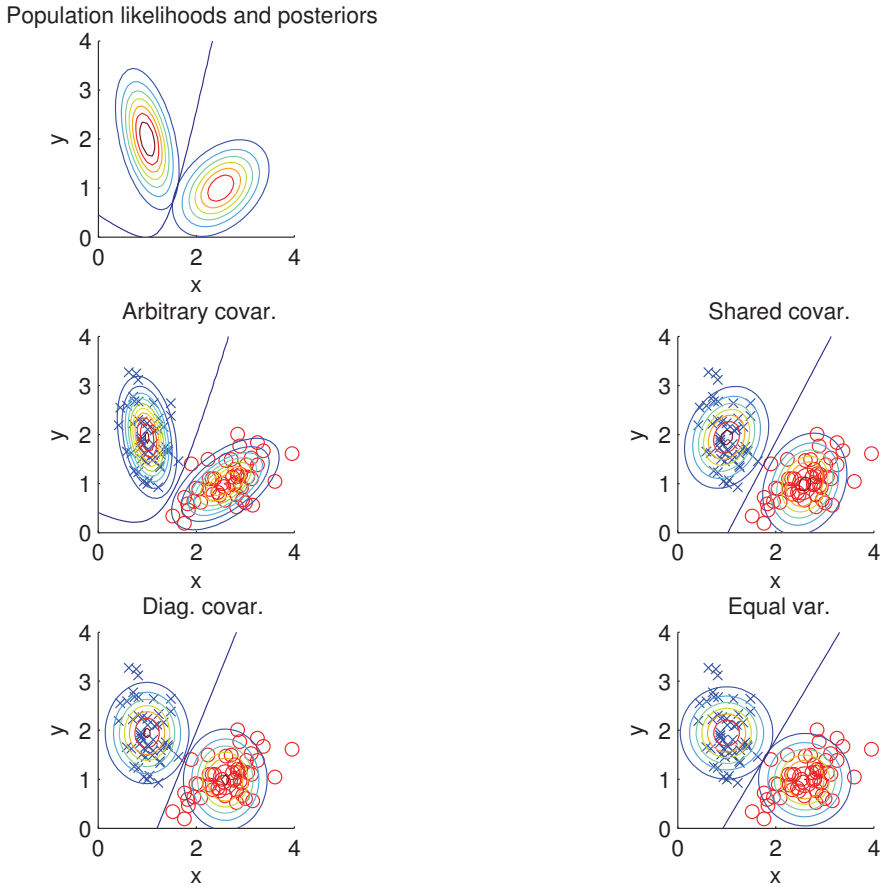


**Figure 5.7** Different cases of the covariance matrices fitted to the same data lead to different boundaries.

that regularization corresponds to approaches where one starts with high variance and constrains toward lower variance, at the risk of increasing bias. In the case of parametric classification with Gaussian densities, the covariance matrices can be written as a weighted average of the three special cases:

$$\mathbf{S}_i' = \alpha\sigma^2\mathbf{I} + \beta\mathbf{S} + (1-\alpha-\beta)\mathbf{S}_i \tag{5.29}$$

When $\alpha = \beta = 0$, this leads to a quadratic classifier. When $\alpha = 0$ and

$\beta = 1$, the covariance matrices are shared, and we get linear classifiers. When $\alpha = 1$ and $\beta = 0$, the covariance matrices are diagonal with $\sigma^2$ on the diagonals, and we get the nearest mean classifier. In between these extremes, we get a whole variety of classifiers where $\alpha, \beta$ are optimized by cross-validation.

Another approach to regularization, when the dataset is small, is one that uses a Bayesian approach by defining priors on $\boldsymbol{\mu}_i$ and $\mathbf{S}_i$ or that uses cross-validation to choose the best of the four cases given in table 5.1.

## 5.7 Discrete Features

In some applications, we have discrete attributes taking one of $n$ different values. For example, an attribute may be color $\in$ {red, blue, green, black}, or another may be pixel $\in$ {on, off}. Let us say $x_j$ are binary (Bernoulli) where

$$p_{ij} \equiv p(x_j = 1|C_i)$$

If $x_j$ are independent binary variables, we have

$$p(\boldsymbol{x}|C_i) = \prod_{j=1}^{d} p_{ij}^{x_j} (1 - p_{ij})^{(1-x_j)}$$

This is another example of the naive Bayes' classifier where $p(x_j|C_i)$ are Bernoulli. The discriminant function is

$$\begin{aligned} g_i(\boldsymbol{x}) &= \log p(\boldsymbol{x}|C_i) + \log P(C_i) \\ &= \sum_j \left[ x_j \log p_{ij} + (1 - x_j) \log(1 - p_{ij}) \right] + \log P(C_i) \end{aligned} \tag{5.30}$$

which is linear. The estimator for $p_{ij}$ is

$$\hat{p}_{ij} = \frac{\sum_t x_j^t r_i^t}{\sum_t r_i^t} \tag{5.31}$$

DOCUMENT CATEGORIZATION

BAG OF WORDS

This approach is used in *document categorization*, an example of which is classifying news reports into various categories, such as, politics, sports, fashion, and so forth. In the *bag of words* representation, we choose a priori $d$ words that we believe give information regarding the class (Manning and Schütze 1999). For example, in news classification, words such as "missile," "athlete," and "couture" are useful, rather than ambiguous words such as "model," or even "runway." In this representation, each

text is a $d$-dimensional binary vector where $x_j$ is 1 if word $j$ occurs in the document and is 0 otherwise. Note that this representation loses all ordering information of words, and hence the name *bag* of words.

After training, $\hat{p}_{ij}$ estimates the probability that word $j$ occurs in document type $i$. Words whose probabilities are similar for different classes do not convey much information; for them to be useful, we would want the probability to be high for one class (or few) and low for all others; we are going to talk about this type of *feature selection* in chapter 6. Another example application of document categorization is *spam filtering* where there are two classes of emails as spam and legitimate. In bioinformatics, too, inputs are generally sequences of discrete items, whether base-pairs or amino acids.

SPAM FILTERING

In the general case, instead of binary features, let us say we have the multinomial $x_j$ chosen from the set $\{v_1, v_2, \ldots, v_{n_j}\}$. We define new 0/1 dummy variables as

$$z_{jk}^t = \begin{cases} 1 & \text{if } x_j^t = v_k \\ 0 & \text{otherwise} \end{cases}$$

Let $p_{ijk}$ denote the probability that $x_j$ belonging to $C_i$ takes value $v_k$:

$$p_{ijk} \equiv p(z_{jk} = 1 | C_i) = p(x_j = v_k | C_i)$$

If the attributes are independent, we have

$$p(\boldsymbol{x}|C_i) = \prod_{j=1}^{d} \prod_{k=1}^{n_j} p_{ijk}^{z_{jk}} \tag{5.32}$$

The discriminant function is then

$$g_i(\boldsymbol{x}) = \sum_j \sum_k z_{jk} \log p_{ijk} + \log P(C_i) \tag{5.33}$$

The maximum likelihood estimator for $p_{ijk}$ is

$$\hat{p}_{ijk} = \frac{\sum_t z_{jk}^t r_i^t}{\sum_t r_i^t} \tag{5.34}$$

which can be plugged into equation 5.33 to give us the discriminant.

## 5.8 Multivariate Regression

MULTIVARIATE LINEAR REGRESSION

In *multivariate linear regression*, the numeric output $r$ is assumed to be

written as a linear function, that is, a weighted sum, of several input variables, $x_1, \ldots, x_d$, and noise. Actually in statistical literature, this is called *multiple* regression; statisticians use the term *multivariate* when there are multiple outputs. The multivariate linear model is

$$r^t = g(\boldsymbol{x}^t | w_0, w_1, \ldots, w_d) + \epsilon = w_0 + w_1 x_1^t + w_2 x_2^t + \cdots + w_d x_d^t + \epsilon \tag{5.35}$$

As in the univariate case, we assume $\epsilon$ to be normal with mean 0 and constant variance, and maximizing the likelihood is equivalent to minimizing the sum of squared errors:

$$E(w_0, w_1, \ldots, w_d | \mathcal{X}) = \frac{1}{2} \sum_t (r^t - w_0 - w_1 x_1^t - w_2 x_2^t - \cdots - w_d x_d^t)^2 \tag{5.36}$$

Taking the derivative with respect to the parameters, $w_j, j = 0, \ldots, d$, we get these *normal equations*:

$$\begin{aligned}
\sum_t r^t &= N w_0 + w_1 \sum_t x_1^t + w_2 \sum_t x_2^t + \cdots + w_d \sum_t x_d^t \\
\sum_t x_1^t r^t &= w_0 \sum_t x_1^t + w_1 \sum_t (x_1^t)^2 + w_2 \sum_t x_1^t x_2^t + \cdots + w_d \sum_t x_1^t x_d^t \\
\sum_t x_2^t r^t &= w_0 \sum_t x_2^t + w_1 \sum_t x_1^t x_2^t + w_2 \sum_t (x_2^t)^2 + \cdots + w_d \sum_t x_2^t x_d^t \\
&\vdots \\
\sum_t x_d^t r^t &= w_0 \sum_t x_d^t + w_1 \sum_t x_d^t x_1^t + w_2 \sum_t x_d^t x_2^t + \cdots + w_d \sum_t (x_d^t)^2
\end{aligned} \tag{5.37}$$

Let us define the following vectors and matrix:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \cdots & x_d^1 \\ 1 & x_1^2 & x_2^2 & \cdots & x_d^2 \\ \vdots & & & & \\ 1 & x_1^N & x_2^N & \cdots & x_d^N \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d, \end{bmatrix}, \boldsymbol{r} = \begin{bmatrix} r^1 \\ r^2 \\ \vdots \\ r^N \end{bmatrix}$$

Then the normal equations can be written as

$$\mathbf{X}^T \mathbf{X} \boldsymbol{w} = \mathbf{X}^T \boldsymbol{r} \tag{5.38}$$

and we can solve for the parameters as

$$\boldsymbol{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \boldsymbol{r} \tag{5.39}$$

This method is the same as we used for polynomial regression using one input. The two problems are the same if we define the variables as

MULTIVARIATE POLYNOMIAL REGRESSION

$x_1 = x, x_2 = x^2, \ldots, x_k = x^k$. This also gives us a hint as to how we can do *multivariate polynomial regression* if necessary (exercise 7), but unless $d$ is small, in multivariate regression, we rarely use polynomials of an order higher than linear.

Actually using higher-order terms of inputs as additional inputs is only one possibility; we can define any nonlinear function of the original inputs using *basis functions*. For example, we can define new inputs $x_2 = \sin(x), x_3 = \exp(x^2)$ if we believe that such a transformation is useful. Then, using a linear model in this new augmented space will correspond to a nonlinear model in the original space. The same calculation will still be valid; we need only replace $\mathbf{X}$ with the data matrix after the basis functions are applied. As we will see later under various guises (e.g., multilayer perceptrons, support vector machines, Gaussian processes), this type of generalizing the linear model is frequently used.

One advantage of linear models is that after the regression, looking at the $w_j, j = 1, \ldots, d$, values, we can extract knowledge: First, by looking at the signs of $w_j$, we can see whether $x_j$ have a positive or negative effect on the output. Second, if all $x_j$ are in the same range, by looking at the absolute values of $w_j$, we can get an idea about how important a feature is, rank the features in terms of their importances, and even remove the features whose $w_j$ are close to 0.

When there are multiple outputs, this can equivalently be defined as a set of independent single-output regression problems.

## 5.9 Notes

A good review text on linear algebra is Strang 2006. Harville 1997 is another excellent book that looks at matrix algebra from a statistical point of view.

One inconvenience with multivariate data is that when the number of dimensions is large, one cannot do a visual analysis. There are methods proposed in the statistical literature for displaying multivariate data; a review is given in Rencher 1995. One possibility is to plot variables two by two as bivariate scatter plots: If the data is multivariate normal, then the plot of any two variables should be roughly linear; this can be used as a visual test of multivariate normality. Another possibility that we discuss in chapter 6 is to project them to one or two dimensions and display there.

Most work on pattern recognition is done assuming multivariate normal densities. Sometimes such a discriminant is even called the Bayes' optimal classifier, but this is generally wrong; it is only optimal if the densities are indeed multivariate normal and if we have enough data to calculate the correct parameters from the data. Rencher 1995 discusses tests for assessing multivariate normality as well as tests for checking for equal covariance matrices. McLachlan 1992 discusses classification with multivariate normals and compares linear and quadratic discriminants.

One obvious restriction of multivariate normals is that it does not allow for data where some features are discrete. A variable with $n$ possible values can be converted into $n$ dummy 0/1 variables, but this increases dimensionality. One can do a dimensionality reduction in this $n$-dimensional space by a method explained in chapter 6 and thereby not increase dimensionality. Parametric classification for such cases of mixed features is discussed in detail in McLachlan 1992.

## 5.10 Exercises

1. Show equation 5.11.

   SOLUTION: Given that

   $$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$$

   we have

   $$\begin{aligned} |\boldsymbol{\Sigma}| &= \sigma_1^2\sigma_2^2 - \rho^2\sigma_1^2\sigma_2^2 = \sigma_1^2\sigma_2^2(1-\rho^2) \\ |\boldsymbol{\Sigma}|^{1/2} &= \sigma_1\sigma_2\sqrt{1-\rho^2} \\ \boldsymbol{\Sigma}^{-1} &= \frac{1}{\sigma_1^2\sigma_2^2(1-\rho^2)} \begin{bmatrix} \sigma_2^2 & -\rho\sigma_1\sigma_2 \\ -\rho\sigma_1\sigma_2 & \sigma_1^2 \end{bmatrix} \end{aligned}$$

   and $(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$ can be expanded as

   $$[x_1-\mu_1 \; x_2-\mu_2] \begin{bmatrix} \frac{\sigma_2^2}{\sigma_1^2\sigma_2^2(1-\rho^2)} & -\frac{\rho\sigma_1\sigma_2}{\sigma_1^2\sigma_2^2(1-\rho^2)} \\ -\frac{\rho\sigma_1\sigma_2}{\sigma_1^2\sigma_2^2(1-\rho^2)} & \frac{\sigma_1^2}{\sigma_1^2\sigma_2^2(1-\rho^2)} \end{bmatrix} \begin{bmatrix} x_1-\mu_1 \\ x_2-\mu_2 \end{bmatrix}$$

   $$= \frac{1}{1-\rho^2}\left[\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2 - 2\rho\left(\frac{x_1-\mu_1}{\sigma_1}\right)\left(\frac{x_2-\mu_2}{\sigma_2}\right) + \left(\frac{x_2-\mu_2}{\sigma_2}\right)^2\right]$$

2. Generate a sample from a multivariate normal density $\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})$, calculate $\boldsymbol{m}$ and $\mathbf{S}$, and compare them with $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Check how your estimates change as the sample size changes.

3. Generate samples from two multivariate normal densities $\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), i = 1, 2$, and calculate the Bayes' optimal discriminant for the four cases in table 5.1.
4. For a two-class problem, for the four cases of Gaussian densities in table 5.1, derive
   $$\log \frac{P(C_1|\boldsymbol{x})}{P(C_2|\boldsymbol{x})}$$
5. Another possibility using Gaussian densities is to have them all diagonal but allow them to be different. Derive the discriminant for this case.
6. Let us say in two dimensions, we have two classes with exactly the same mean. What type of boundaries can be defined?
7. Let us say we have two variables $x_1$ and $x_2$ and we want to make a quadratic fit using them, namely,
   $$f(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 (x_1)^2 + w_5 (x_2)^2$$
   How can we find $w_i, i = 0, \ldots, 5$, given a sample of $\mathcal{X} = \{x_1^t, x_2^t, r^t\}$?
   SOLUTION: We write the fit as
   $$f(x_1, x_2) = w_0 + w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + w_5 z_5$$
   where $z_1 = x_1$, $z_2 = x_2$, $z_3 = x_1 x_2$, $z_4 = (x_1)^2$, and $z_5 = (x_2)^2$. We can then use linear regression to learn $w_i, i = 0, \ldots, 5$. The linear fit in the five-dimensional $(z_1, z_2, z_3, z_4, z_5)$ space corresponds to a quadratic fit in the two-dimensional $(x_1, x_2)$ space. We discuss such generalized linear models in more detail (and other nonlinear basis functions) in chapter 10.
8. In regression we saw that fitting a quadratic is equivalent to fitting a linear model with an extra input corresponding to the square of the input. Can we also do this in classification?
   SOLUTION: Yes. We can define new, auxiliary variables corresponding to powers and cross-product terms and then use a linear model. For example, just as in exercise 7, we can define $z_1 = x_1$, $z_2 = x_2$, $z_3 = x_1 x_2$, $z_4 = (x_1)^2$, and $z_5 = (x_2)^2$ and then use a linear model to learn $w_i, i = 0, \ldots, 5$. The linear discriminant in the five-dimensional $(z_1, z_2, z_3, z_4, z_5)$ space corresponds to a quadratic discriminant in the two-dimensional $(x_1, x_2)$ space.
9. In document clustering, ambiguity of words can be decreased by taking the context into account, for example, by considering pairs of words, as in "cocktail party" vs. "party elections." Discuss how this can be implemented.

## 5.11 References

Duda, R. O., P. E. Hart, and D. G. Stork. 2001. *Pattern Classification*, 2nd ed. New York: Wiley.

Friedman, J. H. 1989. "Regularized Discriminant Analysis." *Journal of American Statistical Association* 84:165–175.

Harville, D. A. 1997. *Matrix Algebra from a Statistician's Perspective*. New York: Springer.

Manning, C. D., and H. Schütze. 1999. *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition*. New York: Wiley.

Rencher, A. C. 1995. *Methods of Multivariate Analysis*. New York: Wiley.

Strang, G. 2006. *Linear Algebra and its Applications*, 4th ed. Boston: Cengage Learning.

# 6 *Dimensionality Reduction*

*The complexity of any classifier or regressor depends on the number of inputs. This determines both the time and space complexity and the necessary number of training examples to train such a classifier or regressor. In this chapter, we discuss feature selection methods that choose a subset of important features pruning the rest and feature extraction methods that form fewer, new features from the original inputs.*

## 6.1 Introduction

IN AN APPLICATION, whether it is classification or regression, observation data that we believe contain information are taken as inputs and fed to the system for decision making. Ideally, we should not need feature selection or extraction as a separate process; the classifier (or regressor) should be able to use whichever features are necessary, discarding the irrelevant. However, there are several reasons why we are interested in reducing dimensionality as a separate preprocessing step:

- In most learning algorithms, the complexity depends on the number of input dimensions, $d$, as well as on the size of the data sample, $N$, and for reduced memory and computation, we are interested in reducing the dimensionality of the problem. Decreasing $d$ also decreases the complexity of the inference algorithm during testing.
- When an input is decided to be unnecessary, we save the cost of extracting it.
- Simpler models are more robust on small datasets. Simpler models

have less variance, that is, they vary less depending on the particulars of a sample, including noise, outliers, and so forth.

- When data can be explained with fewer features, we get a better idea about the process that underlies the data and this allows knowledge extraction. These fewer features may be interpreted as *hidden* or *latent factors* that in combination generate the observed features.
- When data can be represented in a few dimensions without loss of information, it can be plotted and analyzed visually for structure and outliers.

There are two main methods for reducing dimensionality: feature selection and feature extraction. In *feature selection*, we are interested in finding $k$ of the $d$ dimensions that give us the most information, and we discard the other $(d - k)$ dimensions. We discuss *subset selection* as a feature selection method.

FEATURE SELECTION

In *feature extraction*, we are interested in finding a new set of $k$ dimensions that are combinations of the original $d$ dimensions. These methods may be supervised or unsupervised depending on whether or not they use the output information. The best known and most widely used feature extraction methods are *principal component analysis* and *linear discriminant analysis*, which are both linear projection methods, unsupervised and supervised respectively. Principal component analysis bears much similarity to two other unsupervised linear methods, which we also discuss—namely, *factor analysis* and *multidimensional scaling*. When we have not one but two sets of observed variables, *canonical correlation analysis* can be used to find the joint features that explain the dependency between the two. Examples of *nonlinear* dimensionality reduction we cover are *isometric feature mapping*, *locally linear embedding*, and *Laplacian eigenmaps*.

FEATURE EXTRACTION

## 6.2 Subset Selection

In *subset selection*, we are interested in finding the best subset of the set of features. The best subset contains the least number of dimensions that most contribute to accuracy. We discard the remaining, unimportant dimensions. Using a suitable error function, this can be used in both regression and classification problems. There are $2^d$ possible subsets of $d$ variables, but we cannot test for all of them unless $d$ is small and

SUBSET SELECTION

we employ heuristics to get a reasonable (but not optimal) solution in reasonable (polynomial) time.

FORWARD SELECTION

There are two approaches: In *forward selection*, we start with no variables and add them one by one, at each step adding the one that decreases the error the most, until any further addition does not decrease the error (or decreases it only slightly).

BACKWARD SELECTION

In *backward selection*, we start with all variables and remove them one by one, at each step removing the one that decreases the error the most (or increases it only slightly), until any further removal increases the error significantly. In either case, checking the error should be done on a validation set distinct from the training set because we want to test the generalization accuracy. With more features, generally we have lower training error, but not necessarily lower validation error.

Let us denote by $F$, a feature set of input dimensions, $x_i, i = 1, \ldots, d$. $E(F)$ denotes the error incurred on the validation sample when only the inputs in $F$ are used. Depending on the application, the error is either the mean square error or misclassification error.

In *sequential forward selection*, we start with no features: $F = \emptyset$. At each step, for all possible $x_i$, we train our model on the training set and calculate $E(F \cup x_i)$ on the validation set. Then, we choose that input $x_j$ that causes the least error

$$j = \arg\min_i E(F \cup x_i) \tag{6.1}$$

and we

$$\text{add } x_j \text{ to } F \text{ if } E(F \cup x_j) < E(F) \tag{6.2}$$

We stop if adding any feature does not decrease $E$. We may even decide to stop earlier if the decrease in error is too small, where there is a user-defined threshold that depends on the application constraints, trading off the importance of error and complexity. Adding another feature introduces the cost of observing the feature, as well as making the classifier/regressor more complex.

WRAPPER

This algorithm is also known as the *wrapper* approach, where the process of feature extraction is thought to "wrap" around the learner it uses as a subroutine (Kohavi and John 2007).

Let us see an example on the Iris data from the UCI repository; it has four inputs and three classes. There are fifty instances per class, and we use twenty for training and the remaining thirty for validation. We

use the nearest mean as the classifier (see equation 5.26) in section 5.5. We start with a single feature; the plots of training data using single features separately are shown in figure 6.1. Using nearest mean in these one-dimensional spaces of features one to four lead to validation accuracies of 0.76, 0.57, 0.92, and 0.94, respectively. Hence, we select the fourth feature (F4) as our first feature. We then check whether adding another feature leads to improvement. The bivariate plots are shown in figure 6.2; the corresponding validation accuracies using the nearest mean classifier in these two-dimensional spaces are 0.87, 0.92, and 0.96 for (F1,F4), (F2,F4), and (F3,F4), respectively. Thus the third feature is added to as the second feature. Then we check whether adding the first feature or the second feature leads to further improvement; the validation accuracies of the nearest mean classifier in these three-dimensional spaces are both 0.94, and hence we stop with the third and fourth features as our selected features. Incidentally, using *all* four features, we get validation accuracy of 0.94—getting rid of the first two leads to an increase in accuracy.

Note that the features we select at the end depend heavily on the classifier we use. Another important point is that on small datasets, the selected features may also depend on the way data is split between training and validation data; hence on small datasets, it may be a better idea to do multiple, random training/validation splits and decide by looking at average validation performance—we will talk about such resampling methods in chapter 19.

This process of testing features one by one may be costly because to decrease the dimensions from $d$ to $k$, we need to train and test the system $d + (d-1) + (d-2) + \cdots + (d-k)$ times, which is $\mathcal{O}(d^2)$. This is a local search procedure and does not guarantee finding the optimal subset, namely, the minimal subset causing the smallest error. For example, $x_i$ and $x_j$ by themselves may not be good but together may decrease the error a lot, but because this algorithm is greedy and adds attributes one by one, it may not be able to detect this. It is possible to add multiple features at a time, instead of a single one, at the expense of more computation. We can also backtrack and check which, if any, previously added feature can be removed after a current addition, thereby increasing the search space, but this increases complexity. In *floating search* methods (Pudil, Novovičová, and Kittler 1994), the number of added features and removed features can also change at each step.

FLOATING SEARCH

In *sequential backward selection*, we start with $F$ containing all features

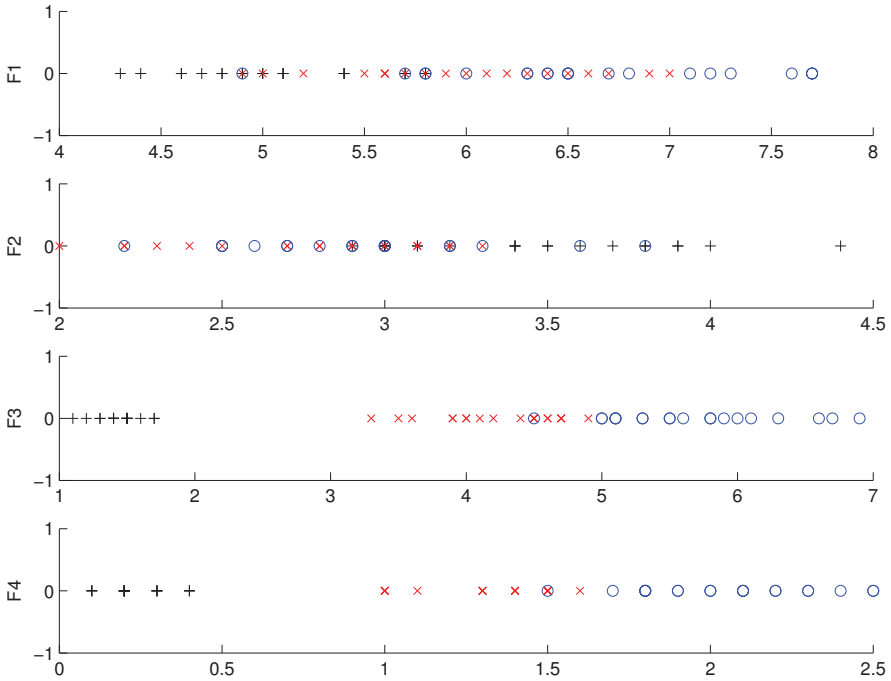


**Figure 6.1** Plot of the training data for single features on Iris dataset; the three classes are shown with different symbols. It can be seen that F4 by itself allows quite good discrimination.

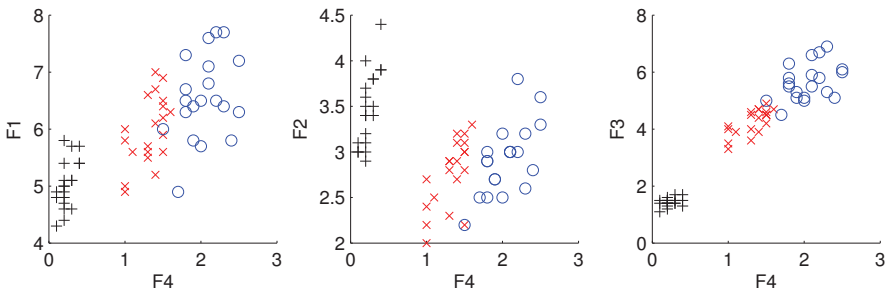


**Figure 6.2** Plot of the training data with F4 as one feature, together with one of F1, F2, and F3. Using (F3, F4) leads to best separation.

and do a similar process except that we remove one attribute from $F$ as opposed to adding to it, and we remove the one that causes the least error

$$j = \arg\min_i E(F - x_i) \tag{6.3}$$

and we

$$\text{remove } x_j \text{ from } F \text{ if } E(F - x_j) < E(F) \tag{6.4}$$

We stop if removing a feature does not decrease the error. To decrease complexity, we may decide to remove a feature if its removal causes only a slight increase in error.

All the variants possible for forward search are also possible for backward search. The complexity of backward search has the same order of complexity as forward search, except that training a system with more features is more costly than training a system with fewer features, and forward search may be preferable especially if we expect many useless features.

Subset selection is supervised in that outputs are used by the regressor or classifier to calculate the error, but it can be used with any regression or classification method. In the particular case of multivariate normals for classification, remember that if the original $d$-dimensional class densities are multivariate normal, then any subset is also multivariate normal and parametric classification can still be used with the advantage of $k \times k$ covariance matrices instead of $d \times d$.

In an application like face recognition, feature selection is not a good method for dimensionality reduction because individual pixels by themselves do not carry much discriminative information; it is the combination of values of several pixels together that carry information about the face identity. This is done by feature extraction methods that we discuss next.

## 6.3 Principal Component Analysis

In projection methods, we are interested in finding a mapping from the inputs in the original $d$-dimensional space to a new $(k < d)$-dimensional space, with minimum loss of information. The projection of $\boldsymbol{x}$ on the direction of $\boldsymbol{w}$ is

$$z = \boldsymbol{w}^T \boldsymbol{x} \tag{6.5}$$

PRINCIPAL COMPONENT ANALYSIS

*Principal component analysis* (PCA) is an unsupervised method in that it does not use the output information; the criterion to be maximized is the variance. The principal component is $\mathbf{w}_1$ such that the sample, after projection on to $\mathbf{w}_1$, is most spread out so that the difference between the sample points becomes most apparent. For a unique solution and to make the direction the important factor, we require $\|\mathbf{w}_1\| = 1$. We know from equation 5.14 that if $z_1 = \mathbf{w}_1^T\mathbf{x}$ with $\text{Cov}(\mathbf{x}) = \boldsymbol{\Sigma}$, then

$$\text{Var}(z_1) = \mathbf{w}_1^T\boldsymbol{\Sigma}\mathbf{w}_1$$

We seek $\mathbf{w}_1$ such that $\text{Var}(z_1)$ is maximized subject to the constraint that $\mathbf{w}_1^T\mathbf{w}_1 = 1$. Writing this as a Lagrange problem, we have

$$\max_{\mathbf{w}_1} \mathbf{w}_1^T\boldsymbol{\Sigma}\mathbf{w}_1 - \alpha(\mathbf{w}_1^T\mathbf{w}_1 - 1) \tag{6.6}$$

Taking the derivative with respect to $\mathbf{w}_1$ and setting it equal to 0, we have

$$2\boldsymbol{\Sigma}\mathbf{w}_1 - 2\alpha\mathbf{w}_1 = 0, \text{ and therefore } \boldsymbol{\Sigma}\mathbf{w}_1 = \alpha\mathbf{w}_1$$

which holds if $\mathbf{w}_1$ is an eigenvector of $\boldsymbol{\Sigma}$ and $\alpha$ the corresponding eigenvalue. Because we want to maximize

$$\mathbf{w}_1^T\boldsymbol{\Sigma}\mathbf{w}_1 = \alpha\mathbf{w}_1^T\mathbf{w}_1 = \alpha$$

we choose the eigenvector with the largest eigenvalue for the variance to be maximum. Therefore the principal component is the eigenvector of the covariance matrix of the input sample with the largest eigenvalue, $\lambda_1 = \alpha$.

The second principal component, $\mathbf{w}_2$, should also maximize variance, be of unit length, and be orthogonal to $\mathbf{w}_1$. This latter requirement is so that after projection $z_2 = \mathbf{w}_2^T\mathbf{x}$ is uncorrelated with $z_1$. For the second principal component, we have

$$\max_{\mathbf{w}_2} \mathbf{w}_2^T\boldsymbol{\Sigma}\mathbf{w}_2 - \alpha(\mathbf{w}_2^T\mathbf{w}_2 - 1) - \beta(\mathbf{w}_2^T\mathbf{w}_1 - 0) \tag{6.7}$$

Taking the derivative with respect to $\mathbf{w}_2$ and setting it equal to 0, we have

$$2\boldsymbol{\Sigma}\mathbf{w}_2 - 2\alpha\mathbf{w}_2 - \beta\mathbf{w}_1 = 0 \tag{6.8}$$

Premultiply by $\mathbf{w}_1^T$ and we get

$$2\mathbf{w}_1^T\boldsymbol{\Sigma}\mathbf{w}_2 - 2\alpha\mathbf{w}_1^T\mathbf{w}_2 - \beta\mathbf{w}_1^T\mathbf{w}_1 = 0$$

Note that $\boldsymbol{w}_1^T\boldsymbol{w}_2 = 0$. $\boldsymbol{w}_1^T\boldsymbol{\Sigma}\boldsymbol{w}_2$ is a scalar, equal to its transpose $\boldsymbol{w}_2^T\boldsymbol{\Sigma}\boldsymbol{w}_1$ where, because $\boldsymbol{w}_1$ is the leading eigenvector of $\boldsymbol{\Sigma}$, $\boldsymbol{\Sigma}\boldsymbol{w}_1 = \lambda_1\boldsymbol{w}_1$. Therefore

$$\boldsymbol{w}_1^T\boldsymbol{\Sigma}\boldsymbol{w}_2 = \boldsymbol{w}_2^T\boldsymbol{\Sigma}\boldsymbol{w}_1 = \lambda_1\boldsymbol{w}_2^T\boldsymbol{w}_1 = 0$$

Then $\beta = 0$ and equation 6.8 reduces to

$$\boldsymbol{\Sigma}\boldsymbol{w}_2 = \alpha\boldsymbol{w}_2$$

which implies that $\boldsymbol{w}_2$ should be the eigenvector of $\boldsymbol{\Sigma}$ with the second largest eigenvalue, $\lambda_2 = \alpha$. Similarly, we can show that the other dimensions are given by the eigenvectors with decreasing eigenvalues.

Because $\boldsymbol{\Sigma}$ is symmetric, for two different eigenvalues, the eigenvectors are orthogonal. If $\boldsymbol{\Sigma}$ is positive definite ($\boldsymbol{x}^T\boldsymbol{\Sigma}\boldsymbol{x} > 0$, for all nonnull $\boldsymbol{x}$), then all its eigenvalues are positive. If $\boldsymbol{\Sigma}$ is singular, then its rank, the effective dimensionality, is $k$ with $k < d$ and $\lambda_i, i = k+1, \ldots, d$ are 0 ($\lambda_i$ are sorted in descending order). The $k$ eigenvectors with nonzero eigenvalues are the dimensions of the reduced space. The first eigenvector (the one with the largest eigenvalue), $\boldsymbol{w}_1$, namely, the principal component, explains the largest part of the variance; the second explains the second largest; and so on.

We define

$$\boldsymbol{z} = \mathbf{W}^T(\boldsymbol{x} - \boldsymbol{m}) \tag{6.9}$$

where the $k$ columns of $\mathbf{W}$ are the $k$ leading eigenvectors of $\mathbf{S}$, the estimator to $\boldsymbol{\Sigma}$. We subtract the sample mean $\boldsymbol{m}$ from $\boldsymbol{x}$ before projection to center the data on the origin. After this linear transformation, we get to a $k$-dimensional space whose dimensions are the eigenvectors, and the variances over these new dimensions are equal to the eigenvalues (see figure 6.3). To normalize variances, we can divide by the square roots of the eigenvalues.

Let us see another derivation: We want to find a matrix $\mathbf{W}$ such that when we have $\boldsymbol{z} = \mathbf{W}^T\boldsymbol{x}$ (assume without loss of generality that $\boldsymbol{x}$ are already centered), we will get $\text{Cov}(\boldsymbol{z}) = \mathbf{D}$ where $\mathbf{D}$ is any diagonal matrix; that is, we would like to get uncorrelated $z_i$.

If we form a $(d \times d)$ matrix $\mathbf{C}$ whose $i$th column is the normalized eigenvector $\boldsymbol{c}_i$ of $\mathbf{S}$, then $\mathbf{C}^T\mathbf{C} = \mathbf{I}$ and

$$\begin{aligned}\mathbf{S} &= \mathbf{S}\mathbf{C}\mathbf{C}^T \\ &= \mathbf{S}(\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_d)\mathbf{C}^T\end{aligned}$$

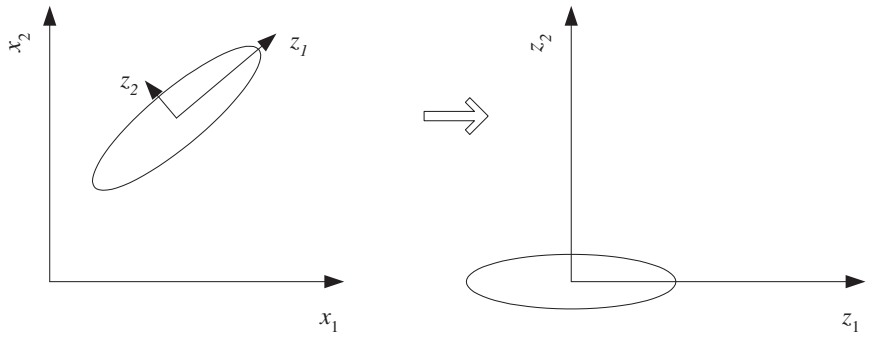


**Figure 6.3** Principal component analysis centers the sample and then rotates the axes to line up with the directions of highest variance. If the variance on $z_2$ is too small, it can be ignored and we have dimensionality reduction from two to one.

$$
\begin{aligned}
&= (\mathbf{S}\boldsymbol{c}_1, \mathbf{S}\boldsymbol{c}_2, \ldots, \mathbf{S}\boldsymbol{c}_d)C^T \\
&= (\lambda_1 \boldsymbol{c}_1, \lambda_2 \boldsymbol{c}_2, \ldots, \lambda_d \boldsymbol{c}_d)\mathbf{C}^T \\
&= \lambda_1 \boldsymbol{c}_1 \boldsymbol{c}_1^T + \cdots + \lambda_d \boldsymbol{c}_d \boldsymbol{c}_d^T \\
&= \mathbf{C}\mathbf{D}\mathbf{C}^T \qquad (6.10)
\end{aligned}
$$

SPECTRAL DECOMPOSITION

where $\mathbf{D}$ is a diagonal matrix whose diagonal elements are the eigenvalues, $\lambda_1, \ldots, \lambda_d$. This is called the *spectral decomposition* of $\mathbf{S}$. Since $\mathbf{C}$ is orthogonal and $\mathbf{C}\mathbf{C}^T = \mathbf{C}^T\mathbf{C} = \mathbf{I}$, we can multiply on the left by $\mathbf{C}^T$ and on the right by $\mathbf{C}$ to obtain

$$\mathbf{C}^T\mathbf{S}\mathbf{C} = \mathbf{D} \qquad (6.11)$$

We know that if $\boldsymbol{z} = \mathbf{W}^T\boldsymbol{x}$, then $\text{Cov}(\boldsymbol{z}) = \mathbf{W}^T\mathbf{S}\mathbf{W}$, which we would like to be equal to a diagonal matrix. Then from equation 6.11, we see that we can set $\mathbf{W} = \mathbf{C}$.

Let us see an example to get some intuition (Rencher 1995): Assume we are given a class of students with grades on five courses and we want to order these students. That is, we want to project the data onto one dimension, such that the difference between the data points become most apparent. We can use PCA. The eigenvector with the highest eigenvalue is the direction that has the highest variance, that is, the direction on which the students are most spread out. This works better than taking

the average because we take into account correlations and differences in variances.

In practice even if all eigenvalues are greater than 0, if $|\mathbf{S}|$ is small, remembering that $|\mathbf{S}| = \prod_{i=1}^{d} \lambda_i$, we understand that some eigenvalues have little contribution to variance and may be discarded. Then, we take into account the leading $k$ components that explain more than, for example, 90 percent, of the variance. When $\lambda_i$ are sorted in descending order, the *proportion of variance* explained by the $k$ principal components is

PROPORTION OF VARIANCE

$$\frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_k + \cdots + \lambda_d}$$

If the dimensions are highly correlated, there will be a small number of eigenvectors with large eigenvalues and $k$ will be much smaller than $d$ and a large reduction in dimensionality may be attained. This is typically the case in many image and speech processing tasks where nearby inputs (in space or time) are highly correlated. If the dimensions are not correlated, $k$ will be as large as $d$ and there is no gain through PCA.

SCREE GRAPH

*Scree graph* is the plot of variance explained as a function of the number of eigenvectors kept (see figure 6.4). By visually analyzing it, one can also decide on $k$. At the "elbow," adding another eigenvector does not significantly increase the variance explained.

Another possibility is to ignore the eigenvectors whose eigenvalues are less than the average input variance. Given that $\sum_i \lambda_i = \sum_i s_i^2$ (equal to the *trace* of $\mathbf{S}$, denoted as $\text{tr}(\mathbf{S})$), the average eigenvalue is equal to the average input variance. When we keep only the eigenvectors with eigenvalues greater than the average eigenvalue, we keep only those that have variance higher than the average input variance.

If the variances of the original $x_i$ dimensions vary considerably, they affect the direction of the principal components more than the correlations, so a common procedure is to preprocess the data so that each dimension has mean 0 and unit variance, before using PCA. Or, one may use the eigenvectors of the correlation matrix, $\mathbf{R}$, instead of the covariance matrix, $\mathbf{S}$, for the correlations to be effective and not the individual variances.

PCA explains variance and is sensitive to outliers: A few points distant from the center would have a large effect on the variances and thus the eigenvectors. *Robust estimation* methods allow calculating parameters in the presence of outliers. A simple method is to calculate the Mahalanobis distance of the data points, discarding the isolated data points that are

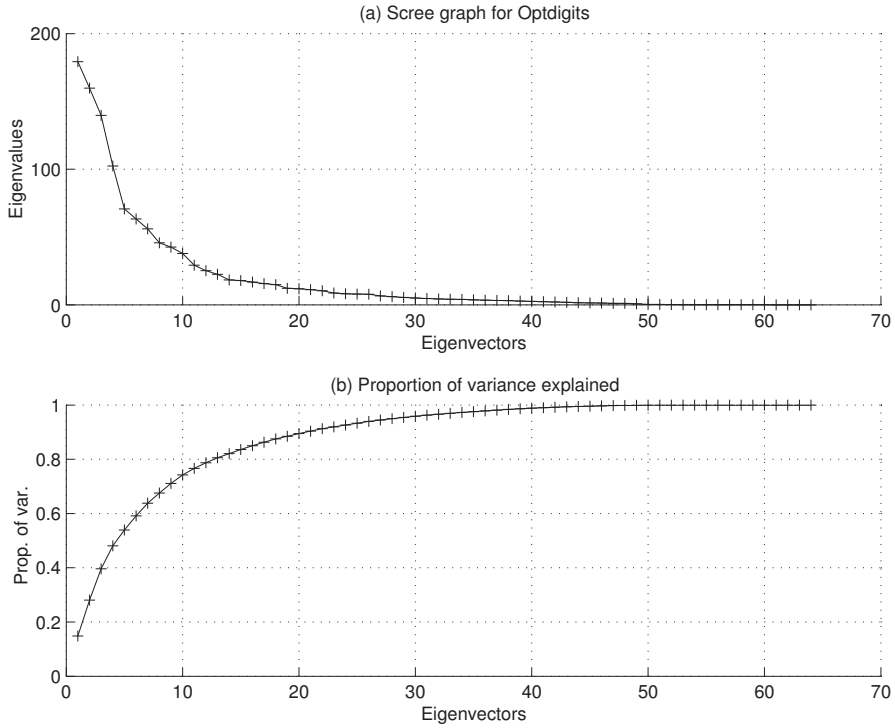


**Figure 6.4** (a) Scree graph. (b) Proportion of variance explained is given for the Optdigits dataset from the UCI Repository. This is a handwritten digit dataset with ten classes and sixty-four dimensional inputs. The first twenty eigenvectors explain 90 percent of the variance.

far away.

If the first two principal components explain a large percentage of the variance, we can do *visual analysis*: We can plot the data in this two-dimensional space (figure 6.5) and search visually for structure, groups, outliers, normality, and so forth. This plot gives a better pictorial description of the sample than a plot of any two of the original variables. By looking at the dimensions of the principal components, we can also try to recover meaningful underlying variables that describe the data. For example, in image applications where the inputs are images, the eigenvectors can also be displayed as images and can be seen as templates for important features; they are typically named "*eigenfaces*," "*eigendigits*,"

EIGENFACES
EIGENDIGITS


**Figure 6.5** Optdigits data plotted in the space of two principal components. Only the labels of a hundred data points are shown to minimize the ink-to-noise ratio.

and so forth (Turk and Pentland 1991).

We know from equation 5.15 that if $\boldsymbol{x} \sim \mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then after projection $\mathbf{W}^T\boldsymbol{x} \sim \mathcal{N}_k(\mathbf{W}^T\boldsymbol{\mu}, \mathbf{W}^T\boldsymbol{\Sigma}\mathbf{W})$. If the sample contains $d$-variate normals, then it projects to $k$-variate normals allowing us to do parametric discrimination in this lower-dimensional space. Because $z_j$ are uncorrelated, the new covariance matrices will be diagonal, and if they are normalized to have unit variance, Euclidean distance can be used in this new space, leading to a simple classifier.

Instance $\boldsymbol{x}^t$ is projected to the $z$-space as

$$\boldsymbol{z}^t = \mathbf{W}^T(\boldsymbol{x}^t - \boldsymbol{\mu})$$

When $\mathbf{W}$ is an orthogonal matrix such that $\mathbf{W}\mathbf{W}^T = \mathbf{I}$, it can be backprojected to the original space as

$$\hat{\boldsymbol{x}}^t = \mathbf{W}\boldsymbol{z}^t + \boldsymbol{\mu}$$

$\hat{\boldsymbol{x}}^t$ is the reconstruction of $\boldsymbol{x}^t$ from its representation in the $z$-space. It is known that among all orthogonal linear projections, PCA minimizes the *reconstruction error*, which is the distance between the instance and its reconstruction from the lower-dimensional space:

RECONSTRUCTION ERROR

$$\sum_t \|\boldsymbol{x}^t - \hat{\boldsymbol{x}}^t\|^2 \tag{6.12}$$

As we discussed earlier, the contribution of each eigenvector is given by its eigenvalue, and hence it makes sense to keep the eigenvectors with the highest eigenvalues; if for dimensionality reduction we discard some eigenvectors with nonzero eigenvalues, there will be a reconstruction error and its magnitude will depend on the discarded eigenvalues. In a visual recognition application—for example, face recognition—displaying $\hat{\boldsymbol{x}}^t$ allows a visual check for information loss during PCA.

PCA is unsupervised and does not use output information. It is a one-group procedure. However, in the case of classification, there are multiple groups. *Karhunen-Loève expansion* allows using class information; for example, instead of using the covariance matrix of the whole sample, we can estimate separate class covariance matrices, take their average (weighted by the priors) as the covariance matrix, and use its eigenvectors.

KARHUNEN-LOÈVE EXPANSION

COMMON PRINCIPAL COMPONENTS

In *common principal components* (Flury 1988), we assume that the principal components are the same for each class whereas the variances of these components differ for different classes:

$$\mathbf{S}_i = \mathbf{C}\mathbf{D}_i\mathbf{C}^T$$

This allows pooling data and is a regularization method whose complexity is less than that of a common covariance matrix for all classes, while still allowing differentiation of $\mathbf{S}_i$. A related approach is *flexible discriminant analysis* (Hastie, Tibshirani, and Buja 1994), which does a linear projection to a lower-dimensional space where all features are uncorrelated and then uses a minimum distance classifier.

FLEXIBLE DISCRIMINANT ANALYSIS

## 6.4 Feature Embedding

Remember that $\mathbf{X}$ is the $N \times d$ data matrix where $N$ is the number of instances and $d$ is the input dimensionality. The covariance matrix of $\boldsymbol{x}$

is $d \times d$ and is equal to $\mathbf{X}^T\mathbf{X}$ if $\mathbf{X}$ is centered to have zero mean (without loss of generality). Principal component analysis uses the eigenvectors of $\mathbf{X}^T\mathbf{X}$. Remember that the spectral decomposition is

$$\mathbf{X}^T\mathbf{X} = \mathbf{W}\mathbf{D}\mathbf{W}^T \tag{6.13}$$

where $\mathbf{W}$ is $d \times d$ and contains the eigenvectors of $\mathbf{X}^T\mathbf{X}$ in its columns and $\mathbf{D}$ is a $d \times d$ diagonal matrix with the corresponding eigenvalues. We assume that the eigenvectors are sorted according to their eigenvalues so that the first column of $\mathbf{W}$ is the eigenvector with the largest eigenvalue in $D_{11}$, and so on. If $\mathbf{X}^T\mathbf{X}$ has rank $k < d$, then $D_{ii} = 0$ for $i > k$.

Let us say we want to reduce dimensionality to $k < d$. In PCA, as we saw before, we take the first $k$ columns of $\mathbf{W}$ (with the highest eigenvalues). Let us denote them by $\boldsymbol{w}_i$ and their eigenvalues by $\lambda_i, i = 1, \ldots, k$. We map to the new $k$-dimensional space by taking a dot product of the original inputs with the eigenvectors:

$$z_i^t = \boldsymbol{w}_i^T \boldsymbol{x}^t, \; i = 1, \ldots, k, t = 1, \ldots, N \tag{6.14}$$

Given that $\lambda_i$ and $\boldsymbol{w}_i$ are the eigenvalues and eigenvectors of $\mathbf{X}^T\mathbf{X}$, for any $i \leq k$, we have

$$(\mathbf{X}^T\mathbf{X})\boldsymbol{w}_i = \lambda_i \boldsymbol{w}_i$$

Premultiplying by $\mathbf{X}$, we find

$$(\mathbf{X}\mathbf{X}^T)\mathbf{X}\boldsymbol{w}_i = \lambda_i \mathbf{X}\boldsymbol{w}_i$$

Hence, $\mathbf{X}\boldsymbol{w}_i$ must be the eigenvectors of $\mathbf{X}\mathbf{X}^T$ with the same eigenvalues (Chatfield and Collins 1980). Note that $\mathbf{X}^T\mathbf{X}$ is $d \times d$, whereas $\mathbf{X}\mathbf{X}^T$ is $N \times N$.

Let us write *its* spectral decomposition:

$$\mathbf{X}\mathbf{X}^T = \mathbf{V}\mathbf{E}\mathbf{V}^T \tag{6.15}$$

$\mathbf{V}$ is the $N \times N$ matrix containing the eigenvectors of $\mathbf{X}\mathbf{X}^T$ in its columns, and $\mathbf{E}$ is the $N \times N$ diagonal matrix with the corresponding eigenvalues. The $N$-dimensional eigenvectors of $\mathbf{X}\mathbf{X}^T$ are the coordinates in the new space. We call this *feature embedding*.

FEATURE EMBEDDING

One caution here: Eigenvectors are usually normalized to have unit length, so if the eigenvectors of $\mathbf{X}\mathbf{X}^T$ are $\boldsymbol{v}_i$ (with the same eigenvalues), we have

$$\boldsymbol{v}_i = \mathbf{X}\boldsymbol{w}_i / \lambda_i, i = 1, \ldots, k$$

because the sum of squares of $\mathbf{X}\boldsymbol{w}_i$ is $\lambda_i$. So if we have $\boldsymbol{v}_i$ (column $i$ of $\mathbf{V}$) calculated and we want to get $\mathbf{X}\boldsymbol{w}_i$, that is, do what PCA does, we should multiply with the square root of the eigenvalue:

$$z_i^t = \mathbf{V}_{ti}\sqrt{\mathbf{E}_{tt}},\ t = 1,\ldots,N, i = 1,\ldots,k \tag{6.16}$$

When $d < N$, as is generally the case, it is simpler to work with $\mathbf{X}^T\mathbf{X}$, that is, use PCA. Sometimes $d > N$ and it is easier to work with $\mathbf{X}\mathbf{X}^T$, which is $N \times N$. For example, in the eigenfaces approach (Turk and Pentland 1991), face images are $256 \times 256 = 65,536$-dimensional and there are only forty face images (four images each from ten people). Note that the rank can never exceed $\min(d, N)$; that is, in this face recognition example, even though the covariance matrix is $65,536 \times 65,536$, we know that the rank (the number of eigenvectors with eigenvalues greater than 0) can never exceed forty. Hence we can work with the $40 \times 40$ matrix instead and use the new coordinates in this forty-dimensional space; for example, do recognition using the nearest mean classifier (Turk and Pentland 1991). The same is also true in most bioinformatics applications where we may have long gene sequences but a small sample. In text clustering, the number of possible words may be much more than the number of documents, and in a movie recommendation system, the number of movies may be much more than the customers.

There is a caveat, though: In the case of PCA, we learn projection vectors, and we can map any new test $\boldsymbol{x}$ to the new space by taking dot products with the eigenvectors—we have a model for projection. We cannot do this with feature embedding, because we do not have projection vectors—we do not learn a projection model but get the coordinates directly. If we have new test data, we should add them to $\mathbf{X}$ and redo the calculation.

The element $(i, j)$ of $\mathbf{X}\mathbf{X}^T$ is equal to the dot product of instances $i$ and $j$; that is, $(\boldsymbol{x}^i)^T(\boldsymbol{x}^j)$, where $i, j = 1,\ldots,N$. If we consider dot product as measuring the similarity between vectors, we can consider $\mathbf{X}\mathbf{X}^T$ as an $N \times N$ matrix of pairwise similarities. From this perspective, we can consider feature embedding as a method of placing instances in a $k$-dimensional space such that pairwise similarities in the new space respect the original pairwise similarities. We will revisit this idea later: In section 6.7, we discuss multidimensional scaling where we use the Euclidean distance between vectors instead of the dot product, and in sections 6.10 and 6.12, we discuss Isomap and Laplacian eigenmaps respectively where we consider non-Euclidean measures of (dis)similarity.

## 6.5 Factor Analysis

In PCA, from the original dimensions $x_i, i = 1, \ldots, d$, we form a new set of variables $\boldsymbol{z}$ that are linear combinations of $x_i$:

$$\boldsymbol{z} = \mathbf{W}^T(\boldsymbol{x} - \boldsymbol{\mu})$$

FACTOR ANALYSIS
LATENT FACTORS

In *factor analysis* (FA), we assume that there is a set of unobservable, *latent factors* $z_j, j = 1, \ldots, k$, which when acting in combination *generate* $\boldsymbol{x}$. Thus the direction is opposite that of PCA (see figure 6.6). The goal is to characterize the dependency among the observed variables by means of a smaller number of factors.

Suppose there is a group of variables that have high correlation among themselves and low correlation with all the other variables. Then there may be a single underlying factor that gave rise to these variables. If the other variables can be similarly grouped into subsets, then a few factors can represent these groups of variables. Though factor analysis always partitions the variables into factor clusters, whether the factors mean anything, or really exist, is open to question.

FA, like PCA, is a one-group procedure and is unsupervised. The aim is to model the data in a smaller dimensional space without loss of information. In FA, this is measured as the correlation between variables.

As in PCA, we have a sample $\mathcal{X} = \{\boldsymbol{x}^t\}_t$ drawn from some unknown probability density with $E[\boldsymbol{x}] = \boldsymbol{\mu}$ and $\text{Cov}(\boldsymbol{x}) = \boldsymbol{\Sigma}$. We assume that the factors are unit normals, $E[z_j] = 0, \text{Var}(z_j) = 1$, and are uncorrelated, $\text{Cov}(z_i, z_j) = 0, i \neq j$. To explain what is not explained by the factors, there is an added source for each input which we denote by $\epsilon_i$. It is assumed to be zero-mean, $E[\epsilon_i] = 0$, and have some unknown variance, $\text{Var}(\epsilon_i) = \psi_i$. These specific sources are uncorrelated among themselves, $\text{Cov}(\epsilon_i, \epsilon_j) = 0, i \neq j$, and are also uncorrelated with the factors, $\text{Cov}(\epsilon_i, z_j) = 0, \forall i, j$.

FA assumes that each input dimension, $x_i, i = 1, \ldots, d$, can be written as a weighted sum of the $k < d$ factors, $z_j, j = 1, \ldots, k$, plus the residual term (see figure 6.7):

$$x_i - \mu_i = v_{i1}z_1 + v_{i2}z_2 + \cdots + v_{ik}z_k + \epsilon_i, \forall i = 1, \ldots, d$$

$$x_i - \mu_i = \sum_{j=1}^{k} v_{ij}z_j + \epsilon_i \tag{6.17}$$

This can be written in vector-matrix form as

$$\boldsymbol{x} - \boldsymbol{\mu} = \mathbf{V}\boldsymbol{z} + \boldsymbol{\epsilon} \tag{6.18}$$

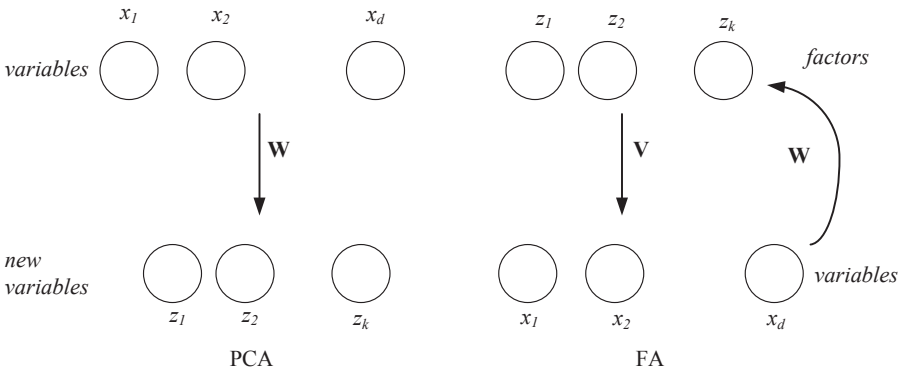


**Figure 6.6** Principal component analysis generates new variables that are linear combinations of the original input variables. In factor analysis, however, we posit that there are factors that when linearly combined generate the input variables.

where $\mathbf{V}$ is the $d \times k$ matrix of weights, called *factor loadings*. From now on, we are going to assume that $\boldsymbol{\mu} = \mathbf{0}$ without loss of generality; we can always add $\boldsymbol{\mu}$ after projection. Given that $\text{Var}(z_j) = 1$ and $\text{Var}(\epsilon_i) = \psi_i$

$$\text{Var}(x_i) = v_{i1}^2 + v_{i2}^2 + \cdots + v_{ik}^2 + \psi_i \tag{6.19}$$

$\sum_{j=1}^{k} v_{ij}^2$ is the part of the variance explained by the common factors and $\psi_i$ is the variance specific to $x_i$.

In vector-matrix form, we have

$$\begin{aligned} \boldsymbol{\Sigma} = \text{Cov}(\boldsymbol{x}) &= \text{Cov}(\mathbf{V}\boldsymbol{z} + \boldsymbol{\epsilon}) \\ &= \text{Cov}(\mathbf{V}\boldsymbol{z}) + \text{Cov}(\boldsymbol{\epsilon}) \\ &= \mathbf{V}\text{Cov}(\boldsymbol{z})\mathbf{V}^T + \boldsymbol{\Psi} \end{aligned} \tag{6.20}$$

$$= \mathbf{V}\mathbf{V}^T + \boldsymbol{\Psi} \tag{6.21}$$

where $\boldsymbol{\Psi}$ is a diagonal matrix with $\psi_i$ on the diagonals. Because the factors are uncorrelated unit normals, we have $\text{Cov}(\boldsymbol{z}) = \mathbf{I}$. With two factors, for example,

$$\text{Cov}(x_1, x_2) = v_{11}v_{21} + v_{12}v_{22}$$

If $x_1$ and $x_2$ have high covariance, then they are related through a factor. If it is the first factor, then $v_{11}$ and $v_{21}$ will both be high; if it is the second factor, then $v_{12}$ and $v_{22}$ will both be high. In either case, the sum

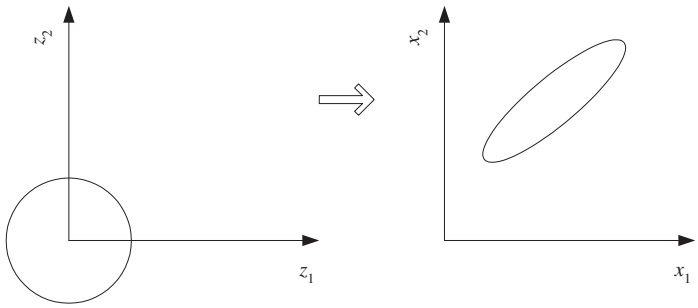


**Figure 6.7** Factors are independent unit normals that are stretched, rotated, and translated to make up the inputs.

$v_{11}v_{21} + v_{12}v_{22}$ will be high. If the covariance is low, then $x_1$ and $x_2$ depend on different factors and in the products in the sum, one term will be high and the other will be low and the sum will be low.

We see that

$$\text{Cov}(x_1, z_2) = \text{Cov}(v_{12}z_2, z_2) = v_{12}\text{Var}(z_2) = v_{12}$$

Thus $\text{Cov}(\boldsymbol{x}, \boldsymbol{z}) = \mathbf{V}$, and we see that the loadings represent the correlations of variables with the factors.

Given $\mathbf{S}$, the estimator of $\boldsymbol{\Sigma}$, we would like to find $\mathbf{V}$ and $\boldsymbol{\Psi}$ such that

$$\mathbf{S} = \mathbf{V}\mathbf{V}^T + \boldsymbol{\Psi}$$

If there are only a few factors, that is, if $\mathbf{V}$ has few columns, then we have a simplified structure for $\mathbf{S}$, as $\mathbf{V}$ is $d \times k$ and $\boldsymbol{\Psi}$ has $d$ values, thus reducing the number of parameters from $d^2$ to $d \cdot k + d$.

Since $\boldsymbol{\Psi}$ is diagonal, covariances are represented by $\mathbf{V}$. Note that PCA does not allow a separate $\boldsymbol{\Psi}$ and it tries to account for both the covariances *and* the variances. When all $\psi_i$ are equal, namely, $\boldsymbol{\Psi} = \psi\mathbf{I}$, we get *probabilistic PCA* (Tipping and Bishop 1999) and the conventional PCA is when $\psi_i$ are 0.

PROBABILISTIC PCA

Let us now see how we can find the factor loadings and the specific variances: Let us first ignore $\boldsymbol{\Psi}$. Then, from its spectral decomposition, we know that we have

$$\mathbf{S} = \mathbf{C}\mathbf{D}\mathbf{C}^T = \mathbf{C}\mathbf{D}^{1/2}\mathbf{D}^{1/2}\mathbf{C}^T = (\mathbf{C}\mathbf{D}^{1/2})(\mathbf{C}\mathbf{D}^{1/2})^T$$

where we take only $k$ of the eigenvectors by looking at the proportion of variance explained so that $\mathbf{C}$ is the $d \times k$ matrix of eigenvectors and $\mathbf{D}^{1/2}$

is the $k \times k$ diagonal matrix with the square roots of the eigenvalues on its diagonals. Thus we have

$$\mathbf{V} = \mathbf{C}\mathbf{D}^{1/2} \tag{6.22}$$

We can find $\psi_j$ from equation 6.19 as

$$\psi_i = s_i^2 - \sum_{j=1}^{k} v_{ij}^2 \tag{6.23}$$

Note that when $\mathbf{V}$ is multiplied with any orthogonal matrix—namely, having the property $\mathbf{T}\mathbf{T}^T = \mathbf{I}$—that is another valid solution and thus the solution is not unique.

$$\mathbf{S} = (\mathbf{V}\mathbf{T})(\mathbf{V}\mathbf{T})^T = \mathbf{V}\mathbf{T}\mathbf{T}^T\mathbf{V}^T = \mathbf{V}\mathbf{I}\mathbf{V}^T = \mathbf{V}\mathbf{V}^T$$

If $\mathbf{T}$ is an orthogonal matrix, the distance to the origin does not change. If $\boldsymbol{z} = \mathbf{T}\boldsymbol{x}$, then

$$\boldsymbol{z}^T\boldsymbol{z} = (\mathbf{T}\boldsymbol{x})^T(\mathbf{T}\boldsymbol{x}) = \boldsymbol{x}^T\mathbf{T}^T\mathbf{T}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{x}$$

Multiplying with an orthogonal matrix has the effect of rotating the axes and allows us to choose the set of axes most interpretable (Rencher 1995). In two dimensions,

$$\mathbf{T} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}$$

rotates the axes by $\phi$. There are two types of rotation: In orthogonal rotation the factors are still orthogonal after the rotation, and in oblique rotation the factors are allowed to become correlated. The factors are rotated to give the maximum loading on as few factors as possible for each variable, to make the factors interpretable. However, interpretability is subjective and should not be used to force one's prejudices on the data.

There are two uses of factor analysis: It can be used for knowledge extraction when we find the loadings and try to express the variables using fewer factors. It can also be used for dimensionality reduction when $k < d$. We already saw how the first one is done. Now, let us see how factor analysis can be used for dimensionality reduction.

When we are interested in dimensionality reduction, we need to be able to find the factor scores, $z_j$, from $x_i$. We want to find the loadings $w_{ji}$ such that

$$z_j = \sum_{i=1}^{d} w_{ji}x_i + \epsilon_j, j = 1, \ldots, k \tag{6.24}$$

where $x_i$ are centered to have mean 0. In vector form, for observation $t$, this can be written as

$$\boldsymbol{z}^t = \mathbf{W}^T \boldsymbol{x}^t + \boldsymbol{\epsilon}, \forall t = 1, \ldots, N$$

This is a linear model with $d$ inputs and $k$ outputs. Its transpose can be written as

$$(\boldsymbol{z}^t)^T = (\boldsymbol{x}^t)^T \mathbf{W} + \boldsymbol{\epsilon}^T, \forall t = 1, \ldots, N$$

Given that we have a sample of $N$ observations, we write

$$\mathbf{Z} = \mathbf{X}\mathbf{W} + \Xi \tag{6.25}$$

where $\mathbf{Z}$ is $N \times k$ of factors, $\mathbf{X}$ is $N \times d$ of (centered) observations, and $\Xi$ is $N \times k$ of zero-mean noise. This is multivariate linear regression with multiple outputs, and we know from section 5.8 that $\mathbf{W}$ can be found as

$$\mathbf{W} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Z}$$

but we do not know $\mathbf{Z}$; it is what we would like to calculate. We multiply and divide both sides by $N - 1$ and obtain

$$\begin{aligned} \mathbf{W} &= (N-1)(\mathbf{X}^T\mathbf{X})^{-1}\frac{\mathbf{X}^T\mathbf{Z}}{N-1} \\ &= \left(\frac{\mathbf{X}^T\mathbf{X}}{N-1}\right)^{-1}\frac{\mathbf{X}^T\mathbf{Z}}{N-1} \\ &= \mathbf{S}^{-1}\mathbf{V} \end{aligned} \tag{6.26}$$

and placing equation 6.26 in equation 6.25, we write

$$\mathbf{Z} = \mathbf{X}\mathbf{W} = \mathbf{X}\mathbf{S}^{-1}\mathbf{V} \tag{6.27}$$

assuming that $\mathbf{S}$ is nonsingular. One can use $\mathbf{R}$ instead of $\mathbf{S}$ when $x_i$ are normalized to have unit variance.

For dimensionality reduction, FA offers no advantage over PCA except the interpretability of factors allowing the identification of common causes, a simple explanation, and knowledge extraction. For example, in the context of speech recognition, $\boldsymbol{x}$ corresponds to the acoustic signal, but we know that it is the result of the (nonlinear) interaction of a small number of *articulators*, namely, jaw, tongue, velum, lips, and mouth, which are positioned appropriately to shape the air as it comes out of the lungs and generate the speech sound. If a speech signal could be

transformed to this articulatory space, then recognition would be much easier. Using such generative models is one of the current research directions for speech recognition; in chapter 14, we discuss how such models can be represented as a graphical model.

## 6.6 Singular Value Decomposition and Matrix Factorization

Given the $N \times d$ data matrix $\mathbf{X}$, we work with $\mathbf{X}^T\mathbf{X}$ if $d < N$ or work with $\mathbf{X}\mathbf{X}^T$ if $N < d$. Both are square matrices and in either case, the spectral decomposition gives us $\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$ where the eigenvector matrix $\mathbf{Q}$ is orthogonal ($\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$) and $\mathbf{\Lambda}$ contains the eigenvalues on its diagonal.

SINGULAR VALUE DECOMPOSITION

The *singular value decomposition* allows us to decompose any $N \times d$ *rectangular* matrix (Strang 2006):

$$\mathbf{X} = \mathbf{V}\mathbf{A}\mathbf{W}^T \tag{6.28}$$

where the $N \times N$ matrix $\mathbf{V}$ contains the eigenvectors of $\mathbf{X}\mathbf{X}^T$ in its columns, the $d \times d$ matrix $\mathbf{W}$ contains the eigenvectors of $\mathbf{X}^T\mathbf{X}$ in its columns, and the $N \times d$ matrix $\mathbf{A}$ contains the $k = \min(N, d)$ *singular values*, $a_i, i = 1, \ldots, k$ on its diagonal that are the square roots of the nonzero eigenvalues of both $\mathbf{X}\mathbf{X}^T$ and $\mathbf{X}^T\mathbf{X}$; the rest of $\mathbf{A}$ is zero. $\mathbf{V}$ and $\mathbf{W}^T$ are orthogonal matrices (but not necessarily transposes of each other).

$$\begin{aligned}\mathbf{X}\mathbf{X}^T &= (\mathbf{V}\mathbf{A}\mathbf{W}^T)(\mathbf{V}\mathbf{A}\mathbf{W}^T)^T = \mathbf{V}\mathbf{A}\mathbf{W}^T\mathbf{W}\mathbf{A}^T\mathbf{V}^T = \mathbf{V}\mathbf{E}\mathbf{V}^T \\ \mathbf{X}^T\mathbf{X} &= (\mathbf{V}\mathbf{A}\mathbf{W}^T)^T(\mathbf{V}\mathbf{A}\mathbf{W}^T) = \mathbf{W}\mathbf{A}^T\mathbf{V}^T\mathbf{V}\mathbf{A}\mathbf{W}^T = \mathbf{W}\mathbf{D}\mathbf{W}^T\end{aligned}$$

where $\mathbf{E} = \mathbf{A}\mathbf{A}^T$ and $\mathbf{D} = \mathbf{A}^T\mathbf{A}$. They are of different sizes but are both square and contain $a_i^2, i = 1, \ldots, k$ on their diagonal and zero elsewhere.

Just as in equation 6.10, we can write

$$\mathbf{X} = \boldsymbol{u}_1 a_1 \boldsymbol{v}_1^T + \boldsymbol{u}_2 a_2 \boldsymbol{v}_2^T + \cdots + \boldsymbol{u}_k a_k \boldsymbol{v}_k^T \tag{6.29}$$

We can ignore the corresponding $\boldsymbol{u}_i, \boldsymbol{v}_i$ of very small, though nonzero, $a_i$ and can still reconstruct $\mathbf{X}$ without too much error.

MATRIX FACTORIZATION

In *matrix factorization*, we write a large matrix as a product of (generally) two matrices:

$$\mathbf{X} = \mathbf{F}\mathbf{G} \tag{6.30}$$

where $\mathbf{X}$ is $N \times d$, $\mathbf{F}$ is $N \times k$, and $\mathbf{G}$ is $k \times d$. $k$ is the dimensionality of the factor space and is hopefully much smaller than $d$ and $N$. The idea

is that although the data may be too large, either it is sparse, or there is high correlation and it can be represented in a space of fewer dimensions.

**G** defines factors in terms of the original attributes and **F** defines data instances in terms of these factors. For example, if **X** is a sample of $N$ documents each using a bag of words representation with $d$ words, each factor may be one topic or concept written using a certain subset of words and each document is a certain combination of such factors. This is called *latent semantic indexing* (Landauer, Laham, and Derr 2004). In *nonnegative* matrix factorization, the matrices are nonnegative and this allows representing a complex object in terms of its parts (Lee and Seung 1999).

LATENT SEMANTIC INDEXING

Let us take another example from retail where **X** is the consumer data. We have $N$ customers and we sell $d$ different products. $\mathbf{X}_{ti}$ corresponds to the amount of product $i$ customer $N$ has purchased. We know that customers do not buy things at random, their purchases depend on a number of factors, for example, their their household size and composition, income level, taste, and so on—these factors are generally hidden from us. In matrix factorization of consumer data, we assume that there are $k$ such factors. **G** relates factors to products: $\mathbf{G}_j$ is a $d$-dimensional vector explaining the relationship between factor $j$ and the products; namely, $\mathbf{G}_{ji}$ is proportional to the amount of product $i$ bought due to factor $j$. Similarly, **F** relates customers to factors: $\mathbf{F}_t$ is the $k$-dimensional vector defining customer $t$ in terms of the hidden factors; namely, $\mathbf{F}_{tj}$ is the belief that behavior of customer $t$ is due to factor $j$. We can hence rewrite equation 6.30 as

$$\mathbf{X}_{ti} = \mathbf{F}_t^T \mathbf{G}_i = \sum_{j=1}^{k} \mathbf{F}_{tj} \mathbf{G}_{ji} \tag{6.31}$$

That is, to calculate the total amount, we take a sum over all such factors where for each, we multiply our belief that the customer is affected by that factor and the amount of product due to that factor—see figure 6.8.

## 6.7 Multidimensional Scaling

Let us say for $N$ points, we are given the distances between pairs of points, $d_{ij}$, for all $i, j = 1, \ldots, N$. We do not know the exact coordinates of the points, their dimensionality, or how the distances are calculated.

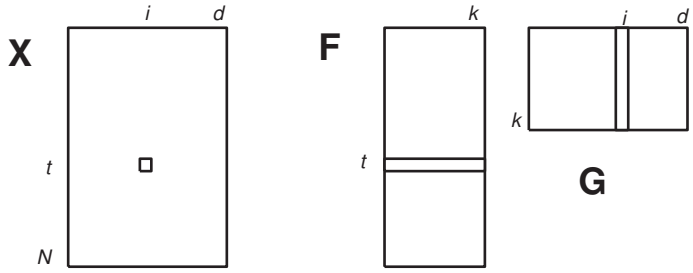


**Figure 6.8** Matrix factorization. $\mathbf{X}$ is the $N \times d$ data matrix. $\mathbf{F}$ is $N \times k$ and its row $t$ defines instance $t$ in terms of the $k$ hidden factors. $\mathbf{G}$ is $k \times d$ and explains factors in terms of the $d$ observed variables. To get $\mathbf{X}_{ti}$, we consider all $k$ factors by taking a weighted sum over them.

MULTIDIMENSIONAL SCALING

*Multidimensional scaling* (MDS) is the method for placing these points in a low—for example, two-dimensional—space such that the Euclidean distance between them there is as close as possible to $d_{ij}$, the given distances in the original space. Thus it requires a projection from some unknown dimensional space to, for example, two dimensions.

In the archetypical example of multidimensional scaling, we take the road travel distances between cities, and after applying MDS, we get an approximation to the map. The map is distorted such that in parts of the country with geographical obstacles like mountains and lakes where the road travel distance deviates much from the direct bird-flight path (Euclidean distance), the map is stretched out to accommodate longer distances (see figure 6.9). The map is centered on the origin, but the solution is still not unique. We can get any rotated or mirror image version.

MDS can be used for dimensionality reduction by calculating pairwise Euclidean distances in the $d$-dimensional $\boldsymbol{x}$ space and giving this as input to MDS, which then projects it to a lower-dimensional space so as to preserve these distances.

Let us say we have a sample $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$ as usual, where $\boldsymbol{x}^t \in \Re^d$. For two points $r$ and $s$, the squared Euclidean distance between them is

$$\begin{aligned} d_{rs}^2 &= \|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2 = \sum_{j=1}^d (x_j^r - x_j^s)^2 = \sum_{j=1}^d (x_j^r)^2 - 2\sum_{j=1}^d x_j^r x_j^s + \sum_{j=1}^d (x_j^s)^2 \\ &= b_{rr} + b_{ss} - 2b_{rs} \end{aligned} \tag{6.32}$$

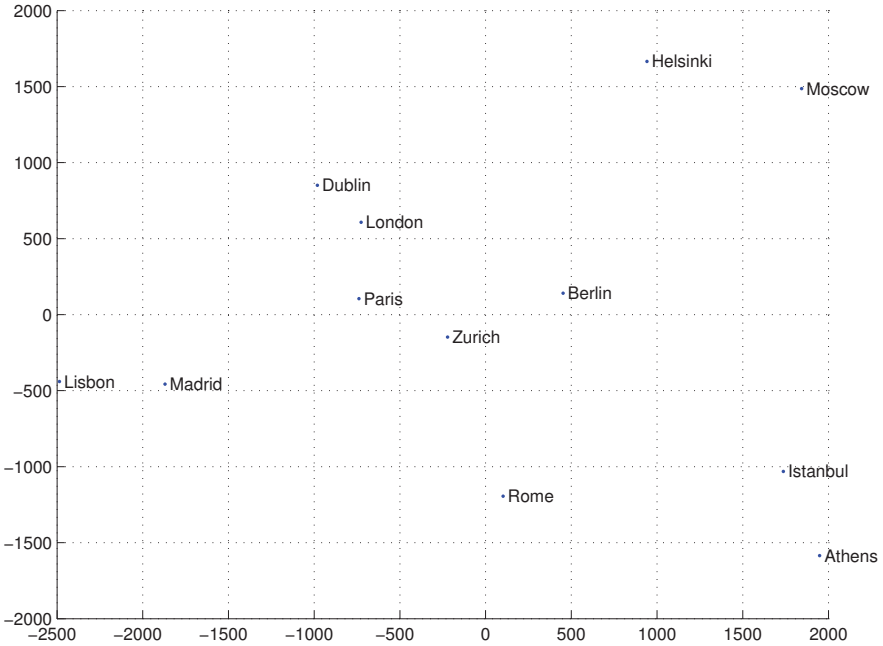


**Figure 6.9** Map of Europe drawn by MDS. Pairwise road travel distances between these cities are given as input, and MDS places them in two dimensions such that these distances are preserved as well as possible.

where $b_{rs}$ is defined as

$$b_{rs} = \sum_{j=1}^{d} x_j^r x_j^s \tag{6.33}$$

To constrain the solution, we center the data at the origin and assume

$$\sum_{t=1}^{N} x_j^t = 0, \forall j = 1, \ldots, d$$

Then, summing up equation 6.32 on $r$, $s$, and both $r$ and $s$, and defining

$$T = \sum_{t=1}^{N} b_{tt} = \sum_t \sum_j (x_j^t)^2$$

we get

$$\sum_r d_{rs}^2 = T + N b_{ss}$$

$$\sum_s d_{rs}^2 = Nb_{rr} + T$$
$$\sum_r \sum_s d_{rs}^2 = 2NT$$

When we define

$$d_{\bullet s}^2 = \frac{1}{N}\sum_r d_{rs}^2 \ , \ d_{r\bullet}^2 = \frac{1}{N}\sum_s d_{rs}^2 \ , \ d_{\bullet\bullet}^2 = \frac{1}{N^2}\sum_r \sum_s d_{rs}^2$$

and using equation 6.32, we get

$$b_{rs} = \frac{1}{2}(d_{r\bullet}^2 + d_{\bullet s}^2 - d_{\bullet\bullet}^2 - d_{rs}^2) \tag{6.34}$$

Having now calculated $b_{rs}$ and knowing that $\mathbf{B} = \mathbf{X}\mathbf{X}^T$ as defined in equation 6.33, we can use feature embedding (section 6.4). We know from the spectral decomposition that $\mathbf{X} = \mathbf{C}\mathbf{D}^{1/2}$ can be used as an approximation for $\mathbf{X}$, where $\mathbf{C}$ is the matrix whose columns are the eigenvectors of $\mathbf{B}$ and $\mathbf{D}^{1/2}$ is a diagonal matrix with square roots of the eigenvalues on the diagonals. Looking at the eigenvalues of $\mathbf{B}$, we decide on a dimensionality $k$ lower than $d$ (and $N$), as we did in PCA and FA. Let us say $\boldsymbol{c}_j$ are the eigenvectors with $\lambda_j$ as the corresponding eigenvalues. Note that $\boldsymbol{c}_j$ is $N$-dimensional. Then we get the new dimensions as

$$z_j^t = \sqrt{\lambda_j} c_j^t, j = 1, \ldots, k, t = 1, \ldots, N \tag{6.35}$$

That is, the new coordinates of instance $t$ are given by the $t$th elements of the eigenvectors, $\boldsymbol{c}_j, j = 1, \ldots, k$, after normalization.

We know that principal component analysis and feature embedding do the same job. This shows that PCA does the same work with MDS and does it more cheaply if $d < N$. PCA done on the correlation matrix rather than the covariance matrix equals doing MDS with standardized Euclidean distances where each variable has unit variance.

In the general case, we want to find a mapping $\boldsymbol{z} = \boldsymbol{g}(\boldsymbol{x}|\theta)$, where $\boldsymbol{z} \in \Re^k, \boldsymbol{x} \in \Re^d$, and $\boldsymbol{g}(\boldsymbol{x}|\theta)$ is the mapping function from $d$ to $k$ dimensions defined up to a set of parameters $\theta$. Classical MDS, which we discussed previously, corresponds to a linear transformation

$$\boldsymbol{z} = \boldsymbol{g}(\boldsymbol{x}|\mathbf{W}) = \mathbf{W}^T\boldsymbol{x} \tag{6.36}$$

but in a general case, a nonlinear mapping can also be used; this is called *Sammon mapping*. The normalized error in mapping is called the *Sam-*

SAMMON MAPPING

*mon stress* and is defined as

$$
\begin{aligned}
E(\theta|\mathcal{X}) &= \sum_{r,s} \frac{(\|\boldsymbol{z}^r - \boldsymbol{z}^s\| - \|\boldsymbol{x}^r - \boldsymbol{x}^s\|)^2}{\|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2} \\
&= \sum_{r,s} \frac{(\|\boldsymbol{g}(\boldsymbol{x}^r|\theta) - \boldsymbol{g}(\boldsymbol{x}^s|\theta)\| - \|\boldsymbol{x}^r - \boldsymbol{x}^s\|)^2}{\|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2}
\end{aligned} \tag{6.37}
$$

One can use any regression method for $\boldsymbol{g}(\cdot|\theta)$ and estimate $\theta$ to minimize the stress on the training data $\mathcal{X}$. If $\boldsymbol{g}(\cdot)$ is nonlinear in $\boldsymbol{x}$, this will then correspond to a nonlinear dimensionality reduction.

In the case of classification, one can include class information in the distance (see Webb 1999) as

$$d'_{rs} = (1-\alpha)d_{rs} + \alpha c_{rs}$$

where $c_{rs}$ is the "distance" between the classes $\boldsymbol{x}^r$ and $\boldsymbol{x}^s$ belong to. This interclass distance should be supplied subjectively and $\alpha$ is optimized using cross-validation.

## 6.8 Linear Discriminant Analysis

LINEAR DISCRIMINANT ANALYSIS

*Linear discriminant analysis* (LDA) is a supervised method for dimensionality reduction for classification problems. We start with the case where there are two classes, then generalize to $K > 2$ classes.

Given samples from two classes $C_1$ and $C_2$, we want to find the direction, as defined by a vector $\boldsymbol{w}$, such that when the data are projected onto $\boldsymbol{w}$, the examples from the two classes are as well separated as possible. As we saw before,

$$z = \boldsymbol{w}^T\boldsymbol{x} \tag{6.38}$$

is the projection of $\boldsymbol{x}$ onto $\boldsymbol{w}$ and thus is a dimensionality reduction from $d$ to 1.

$\boldsymbol{m}_1$ and $m_1$ are the means of samples from $C_1$ before and after projection, respectively. Note that $\boldsymbol{m}_1 \in \Re^d$ and $m_1 \in \Re$. We are given a sample $\mathcal{X} = \{\boldsymbol{x}^t, r^t\}$ such that $r^t = 1$ if $\boldsymbol{x}^t \in C_1$ and $r^t = 0$ if $\boldsymbol{x}^t \in C_2$.

$$
\begin{aligned}
m_1 &= \frac{\sum_t \boldsymbol{w}^T\boldsymbol{x}^t r^t}{\sum_t r^t} = \boldsymbol{w}^T\boldsymbol{m}_1 \\
m_2 &= \frac{\sum_t \boldsymbol{w}^T\boldsymbol{x}^t (1-r^t)}{\sum_t (1-r^t)} = \boldsymbol{w}^T\boldsymbol{m}_2
\end{aligned} \tag{6.39}
$$

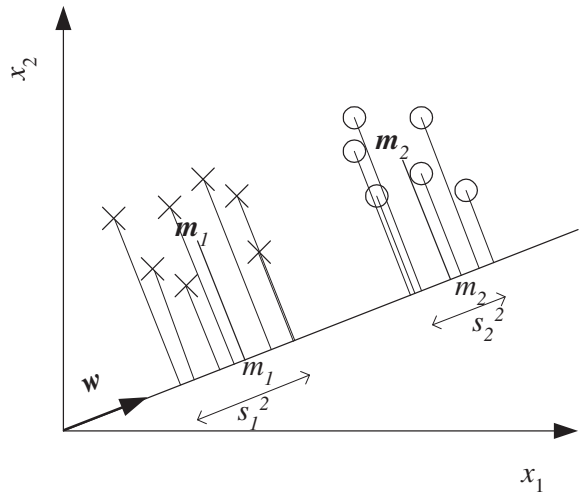


**Figure 6.10** Two-dimensional, two-class data projected on $\boldsymbol{w}$.

SCATTER

The *scatter* of samples from $C_1$ and $C_2$ after projection are

$$s_1^2 = \sum_t (\boldsymbol{w}^T \boldsymbol{x}^t - m_1)^2 r^t$$

$$s_2^2 = \sum_t (\boldsymbol{w}^T \boldsymbol{x}^t - m_2)^2 (1 - r^t) \tag{6.40}$$

FISHER'S LINEAR DISCRIMINANT

After projection, for the two classes to be well separated, we would like the means to be as far apart as possible and the examples of classes be scattered in as small a region as possible. So we want $|m_1 - m_2|$ to be large and $s_1^2 + s_2^2$ to be small (see figure 6.10). *Fisher's linear discriminant* is $\boldsymbol{w}$ that maximizes

$$J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} \tag{6.41}$$

Rewriting the numerator, we get

$$\begin{aligned} (m_1 - m_2)^2 &= (\boldsymbol{w}^T \boldsymbol{m}_1 - \boldsymbol{w}^T \boldsymbol{m}_2)^2 \\ &= \boldsymbol{w}^T (\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^T \boldsymbol{w} \\ &= \boldsymbol{w}^T \mathbf{S}_B \boldsymbol{w} \end{aligned} \tag{6.42}$$

BETWEEN-CLASS SCATTER MATRIX

where $\mathbf{S}_B = (\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^T$ is the *between-class scatter matrix*. The

denominator is the sum of scatter of examples of classes around their means after projection and can be rewritten as

$$\begin{aligned} s_1^2 &= \sum_t (\boldsymbol{w}^T \boldsymbol{x}^t - m_1)^2 r^t \\ &= \sum_t \boldsymbol{w}^T (\boldsymbol{x}^t - \boldsymbol{m}_1)(\boldsymbol{x}^t - \boldsymbol{m}_1)^T \boldsymbol{w} r^t \\ &= \boldsymbol{w}^T \mathbf{S}_1 \boldsymbol{w} \end{aligned} \tag{6.43}$$

where

$$\mathbf{S}_1 = \sum_t r^t (\boldsymbol{x}^t - \boldsymbol{m}_1)(\boldsymbol{x}^t - \boldsymbol{m}_1)^T \tag{6.44}$$

WITHIN-CLASS SCATTER MATRIX

is the *within-class scatter matrix* for $C_1$. $\mathbf{S}_1 / \sum_t r^t$ is the estimator of $\boldsymbol{\Sigma}_1$. Similarly, $s_2^2 = \boldsymbol{w}^T \mathbf{S}_2 \boldsymbol{w}$ with $\mathbf{S}_2 = \sum_t (1 - r^t)(\boldsymbol{x}^t - \boldsymbol{m}_2)(\boldsymbol{x}^t - \boldsymbol{m}_2)^T$, and we get

$$s_1^2 + s_2^2 = \boldsymbol{w}^T \mathbf{S}_W \boldsymbol{w}$$

where $\mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2$ is the total within-class scatter. Note that $s_1^2 + s_2^2$ divided by the total number of samples is the variance of the pooled data. Equation 6.41 can be rewritten as

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T \mathbf{S}_B \boldsymbol{w}}{\boldsymbol{w}^T \mathbf{S}_W \boldsymbol{w}} = \frac{|\boldsymbol{w}^T (\boldsymbol{m}_1 - \boldsymbol{m}_2)|^2}{\boldsymbol{w}^T \mathbf{S}_W \boldsymbol{w}} \tag{6.45}$$

Taking the derivative of $J$ with respect to $\boldsymbol{w}$ and setting it equal to 0, we get

$$\frac{\boldsymbol{w}^T (\boldsymbol{m}_1 - \boldsymbol{m}_2)}{\boldsymbol{w}^T \mathbf{S}_W \boldsymbol{w}} 2 \left( (\boldsymbol{m}_1 - \boldsymbol{m}_2) - \frac{\boldsymbol{w}^T (\boldsymbol{m}_1 - \boldsymbol{m}_2)}{\boldsymbol{w}^T \mathbf{S}_W \boldsymbol{w}} \mathbf{S}_W \boldsymbol{w} \right) = 0$$

Given that $\boldsymbol{w}^T (\boldsymbol{m}_1 - \boldsymbol{m}_2) / \boldsymbol{w}^T \mathbf{S}_W \boldsymbol{w}$ is a constant, we have

$$\boldsymbol{w} = c \mathbf{S}_W^{-1} (\boldsymbol{m}_1 - \boldsymbol{m}_2) \tag{6.46}$$

where $c$ is some constant. Because it is the direction that is important for us and not the magnitude, we can just take $c = 1$ and find $\boldsymbol{w}$.

Remember that when $p(\boldsymbol{x}|C_i) \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma})$, we have a linear discriminant where $\boldsymbol{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$, and we see that Fisher's linear discriminant is optimal if the classes are normally distributed. Under the same assumption, a threshold, $w_0$, can also be calculated to separate the two classes. But Fisher's linear discriminant can be used even when the classes are not normal. We have projected the samples from $d$ dimensions to 1,

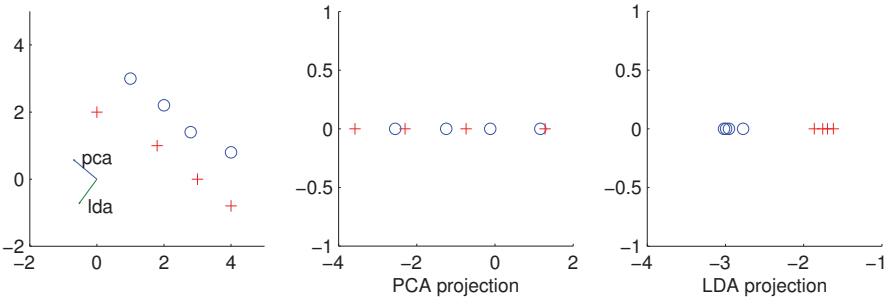


**Figure 6.11** Two-dimensional synthetic data, directions found by PCA and LDA and projections along these directions are shown. LDA uses class information and as expected, does a much better job in terms of class separation.

and any classification method can be used afterward. In figure 6.11, we see two-dimensional synthetic data with two classes. As we see, and as expected, because it uses the class information, LDA direction is superior to the PCA direction in terms of the ease of discrimination afterwards.

In the case of $K > 2$ classes, we want to find the matrix $\mathbf{W}$ such that

$$\mathbf{z} = \mathbf{W}^T \mathbf{x} \tag{6.47}$$

where $\mathbf{z}$ is $k$-dimensional and $\mathbf{W}$ is $d \times k$. The within-class scatter matrix for $C_i$ is

$$\mathbf{S}_i = \sum_t r_i^t (\mathbf{x}^t - \mathbf{m}_i)(\mathbf{x}^t - \mathbf{m}_i)^T \tag{6.48}$$

where $r_i^t = 1$ if $\mathbf{x}^t \in C_i$ and 0 otherwise. The total within-class scatter is

$$\mathbf{S}_W = \sum_{i=1}^K \mathbf{S}_i \tag{6.49}$$

When there are $K > 2$ classes, the scatter of the means is calculated as how much they are scattered around the overall mean

$$\mathbf{m} = \frac{1}{K} \sum_{i=1}^K \mathbf{m}_i \tag{6.50}$$

and the between-class scatter matrix is

$$\mathbf{S}_B = \sum_{i=1}^K N_i (\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^T \tag{6.51}$$

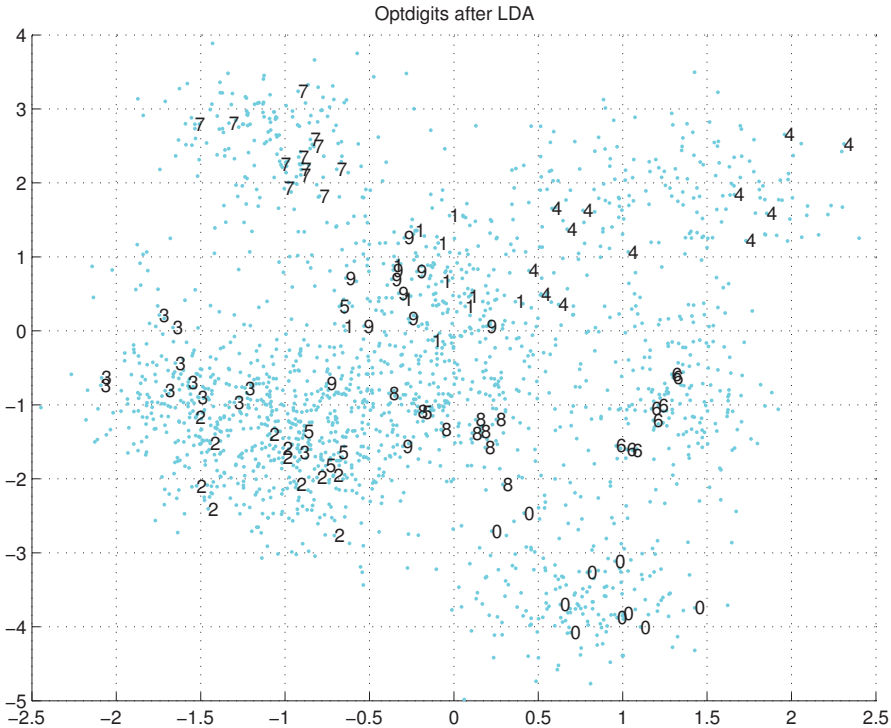


**Figure 6.12** Optdigits data plotted in the space of the first two dimensions found by LDA. Comparing this with figure 6.5, we see that LDA, as expected, leads to a better separation of classes than PCA. Even in this two-dimensional space (there are nine altogether), we can discern separate clouds for different classes.

with $N_i = \sum_t r_i^t$. The between-class scatter matrix after projection is $\mathbf{W}^T\mathbf{S}_B\mathbf{W}$ and the within-class scatter matrix after projection is $\mathbf{W}^T\mathbf{S}_W\mathbf{W}$. These are both $k \times k$ matrices. We want the first scatter to be large, that is, after the projection, in the new $k$-dimensional space we want class means to be as far apart from each other as possible. We want the second scatter to be small, that is, after the projection, we want samples from the same class to be as close to their mean as possible. For a scatter (or covariance) matrix, a measure of spread is the determinant, remembering that the determinant is the product of eigenvalues and that an eigenvalue gives the variance along its eigenvector (component). Thus we

are interested in the matrix $\mathbf{W}$ that maximizes

$$J(\mathbf{W}) = \frac{|\mathbf{W}^T \mathbf{S}_B \mathbf{W}|}{|\mathbf{W}^T \mathbf{S}_W \mathbf{W}|} \tag{6.52}$$

The largest eigenvectors of $\mathbf{S}_W^{-1}\mathbf{S}_B$ are the solution. $\mathbf{S}_B$ is the sum of $K$ matrices of rank 1, namely, $(\boldsymbol{m}_i - \boldsymbol{m})(\boldsymbol{m}_i - \boldsymbol{m})^T$, and only $K-1$ of them are independent. Therefore, $\mathbf{S}_B$ has a maximum rank of $K-1$ and we take $k = K-1$. Thus we define a new lower, $(K-1)$-dimensional space where the discriminant is then to be constructed (see figure 6.12). Though LDA uses class separability as its goodness criterion, any classification method can be used in this new space for estimating the discriminants.

We see that to be able to apply LDA, $\mathbf{S}_W$ should be invertible. If this is not the case, we can first use PCA to get rid of singularity and then apply LDA to its result; however, we should make sure that PCA does not reduce dimensionality so much that LDA does not have anything left to work on.

## 6.9 Canonical Correlation Analysis

In all the methods discussed previously, we assume we have a single source of data returning us a single set of observations. Sometimes, for the same object or event, we have two types of variables. For example, in speech recognition, in addition to the acoustic information, we may also have the visual information of the lip movements while the word is uttered; in retrieval, we may have image data and text annotations. Frequently, these two sets of variables are correlated, and we want to take this correlation into account while reducing dimensionality to a joint space. This is the idea in *canonical correlation analysis* (CCA) (Rencher 1995).

CANONICAL CORRELATION ANALYSIS

Let us say we have a dataset with two sets of variables $\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{y}^t\}_{t=1}^N$ where $\boldsymbol{x}^t \in \Re^d$ and $\boldsymbol{y}^t \in \Re^e$. Note that both of these are inputs and this is an unsupervised problem; if there is a required output for classification or regression, that is handled afterward as in PCA (section 6.3).

The *canonical correlation* is measured as the amount of correlation between the $\boldsymbol{x}$ dimensions and the $\boldsymbol{y}$ dimensions. Let us define a notation: $\mathbf{S}_{xx} = \text{Cov}(\boldsymbol{x}) = E[(\boldsymbol{x} - \boldsymbol{\mu}_x)^2]$ is the covariance matrix of the $\boldsymbol{x}$ dimensions and is $d \times d$—this is the $\boldsymbol{\Sigma}$ matrix that we use frequently, in PCA for example. Now, we also have the $e \times e$ covariance matrix of the $\boldsymbol{y}$, namely, $\mathbf{S}_{yy} = \text{Cov}(\boldsymbol{y})$. We also have the two cross-covariance matrices, namely,

$\mathbf{S}_{xy} = \text{Cov}(\boldsymbol{x}, \boldsymbol{y}) = E[(\boldsymbol{x} - \boldsymbol{\mu}_x)(\boldsymbol{y} - \boldsymbol{\mu}_y)]$, which is $d \times e$, and the other cross-covariance matrix $\mathbf{S}_{yx} = \text{Cov}(\boldsymbol{y}, \boldsymbol{x}) = E[(\boldsymbol{y} - \boldsymbol{\mu}_y)(\boldsymbol{x} - \boldsymbol{\mu}_x)]$, which is $e \times d$.

We are interested in the two vectors $\boldsymbol{w}$ and $\boldsymbol{v}$ such that when $\boldsymbol{x}$ is projected along $\boldsymbol{w}$ and $\boldsymbol{y}$ is projected along $\boldsymbol{v}$, we have maximum correlation. That is, we want to maximize

$$\begin{aligned} \rho &= \text{Corr}(\boldsymbol{w}^T\boldsymbol{x}, \boldsymbol{v}^T\boldsymbol{y}) = \frac{\text{Cov}(\boldsymbol{w}^T\boldsymbol{x}, \boldsymbol{v}^T\boldsymbol{y})}{\sqrt{\text{Var}(\boldsymbol{w}^T\boldsymbol{x})}\sqrt{\text{Var}(\boldsymbol{v}^T\boldsymbol{y})}} \\ &= \frac{\boldsymbol{w}^T\text{Cov}(\boldsymbol{x}, \boldsymbol{y})\boldsymbol{v}}{\sqrt{\boldsymbol{w}^T\text{Var}(\boldsymbol{x})\boldsymbol{w}}\sqrt{\boldsymbol{v}^T\text{Var}(\boldsymbol{y})\boldsymbol{v}}} = \frac{\boldsymbol{w}^T\mathbf{S}_{xy}\boldsymbol{v}}{\sqrt{\boldsymbol{w}^T\mathbf{S}_{xx}\boldsymbol{w}}\sqrt{\boldsymbol{v}^T\mathbf{S}_{yy}\boldsymbol{v}}} \end{aligned} \tag{6.53}$$

Equally, we can say that what we want is to maximize $\boldsymbol{w}^T\mathbf{S}_{xy}\boldsymbol{v}$ subject to $\boldsymbol{w}^T\mathbf{S}_{xx}\boldsymbol{w} = 1$ and $\boldsymbol{v}^T\mathbf{S}_{yy}\boldsymbol{v} = 1$. Writing these as Lagrangian terms as we do in PCA and then taking derivatives with respect to $\boldsymbol{w}$ and $\boldsymbol{v}$ and setting them equal to 0, we see that $\boldsymbol{w}$ should be an eigenvector of $\mathbf{S}_{xx}^{-1}\mathbf{S}_{xy}\mathbf{S}_{yy}^{-1}\mathbf{S}_{yx}$ and similarly $\boldsymbol{v}$ should be an eigenvector of $\mathbf{S}_{yy}^{-1}\mathbf{S}_{yx}\mathbf{S}_{xx}^{-1}\mathbf{S}_{xy}$ (Hardoon, Szedmak, and Shawe-Taylor 2004).

Because we are interested in maximizing the correlation, we choose the two eigenvectors with the highest eigenvalues—let us call them $\boldsymbol{w}_1$ and $\boldsymbol{v}_1$—and the amount of correlation is equal to their (shared) eigenvalue $\lambda_1$. The eigenvalues of **AB** are the same as those of **BA** as long as **AB** and **BA** are square, but the eigenvectors are not the same: $\boldsymbol{w}_1$ is $d$-dimensional whereas $\boldsymbol{v}_1$ is $e$-dimensional.

Just as we do in PCA, we can decide how many pairs of eigenvectors $(\boldsymbol{w}_i, \boldsymbol{v}_i)$ to use by looking at the relative value of the corresponding eigenvalue:

$$\frac{\lambda_i}{\sum_{j=1}^{s}\lambda_j}$$

where $s = \min(d, e)$ is the maximum possible rank. We need to keep enough of them to conserve the correlation in the data.

Let us say we choose $k$ as the dimensionality, then we get the *canonical variates* by projecting the training instances along them:

$$a_i^t = \boldsymbol{w}_i^T\boldsymbol{x}^t, b_i^t = \boldsymbol{v}_i^T\boldsymbol{y}^t, i = 1, \ldots, k \tag{6.54}$$

which we can write in matrix form as

$$\boldsymbol{a}^t = \mathbf{W}^T\boldsymbol{x}^t, \boldsymbol{b}^t = \mathbf{V}^T\boldsymbol{y}^t \tag{6.55}$$

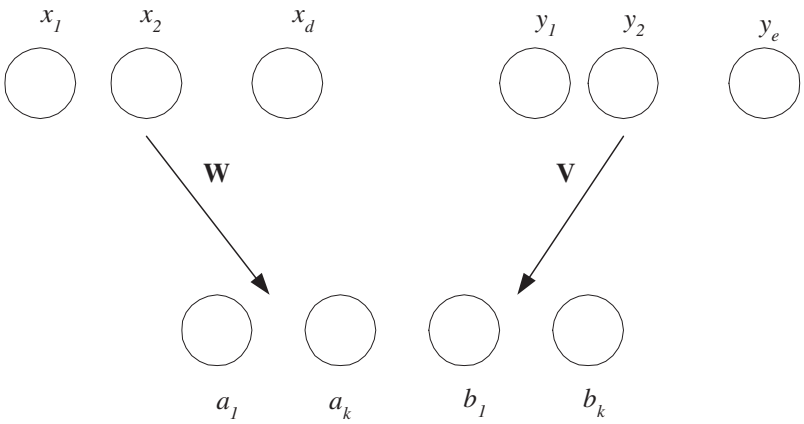


**Figure 6.13** Canonical correlation analysis uses two sets of variables $\boldsymbol{x}$ and $\boldsymbol{y}$ and projects each so that the correlation after projections is maximized.

where $\mathbf{W}$ is the $d \times k$ matrix whose columns are $\boldsymbol{w}_i$ and $\mathbf{V}$ is the $e \times k$ matrix whose columns are $\boldsymbol{v}_i$ (figure 6.13). This vector of $(a_i, b_i)$ pairs now constitutes our new, lower-dimensional representation that we can then use, for example, for classification. These new features are nonredundant: The values of $a_i$ are uncorrelated, and each $a_i$ is uncorrelated with all $b_j, j \neq i$.

For CCA to make sense, the two sets of variables need to be dependent. In the case of retrieval, for example, Hardoon, Szedmak, and Shawe-Taylor 2004, there is dependence: The word "sky" is associated with a lot of blue color in the image, so it makes sense to use CCA. But this is not always the case. For example, in user authentication, we may have the signature and iris images, but there is no reason to assume any dependence between them. In such a case, it would be better to do dimensionality reduction separately on signature and iris images, hence recovering dependence between features in the same set. It only makes sense to use CCA if we can also assume dependence between features of separate sets. Rencher (1995) discusses tests to check whether $\mathbf{S}_{xy} = \mathbf{0}$, that is, whether $\boldsymbol{x}$ and $\boldsymbol{y}$ are independent. One interesting note is that if $\boldsymbol{x}$ are the observed variables and if class labels are given as $\boldsymbol{y}$ using 1-of-$K$ encoding, CCA finds the same solution as Fisher's LDA (section 6.8).

In factor analysis, we give a generative interpretation of dimensionality

reduction: We assume that there are hidden variables $\mathbf{z}$ that in combination cause the observed variables $\mathbf{x}$. Here, we can similarly think of hidden variables that generate $\mathbf{x}$ *and* $\mathbf{y}$; actually, we may consider $\mathbf{a}$ and $\mathbf{b}$ together constituting $\mathbf{z}$, the representation in the latent space.

It is possible to generalize CCA for more than two sets of variables. Bach and Jordan (2005) give a probabilistic interpretation of CCA where more than two sets of variables are possible.

## 6.10 Isomap

Principal component analysis, which we discussed in section 6.3, works when the data lies in a linear subspace. However, this may not hold in many applications. Take, for example, face recognition where a face is represented as a two-dimensional, say $100 \times 100$ image. In this case, each face is a point in $10,000$ dimensions. Now let us say that we take a series of pictures as a person slowly rotates his or her head from right to left. The sequence of face images we capture follows a trajectory in the $10,000$-dimensional space, and this is not linear. Now consider the faces of many people. The trajectories of all their faces as they rotate their faces define a manifold in the $10,000$-dimensional space, and this is what we want to model. The similarity between two faces cannot simply be written in terms of the sum of the pixel differences, and hence Euclidean distance is not a good metric. It may even be the case that images of two different people with the same pose have smaller Euclidean distance between them than the images of two different poses of the same person. This is not what we want. What should count is the distance along the manifold, which is called the *geodesic distance*. *Isometric feature mapping* (Isomap) (Tenenbaum, de Silva, and Langford 2000) estimates this distance and applies multidimensional scaling (section 6.7), using it for dimensionality reduction.

GEODESIC DISTANCE
ISOMETRIC FEATURE MAPPING

Isomap uses the geodesic distances between all pairs of data points. For neighboring points that are close in the input space, Euclidean distance can be used; for small changes in pose, the manifold is locally linear. For faraway points, geodesic distance is approximated by the sum of the distances between the points along the way over the manifold. This is done by defining a graph whose nodes correspond to the $N$ data points and whose edges connect neighboring points (those with distance less than some $\epsilon$ or one of the $n$ nearest) with weights corresponding

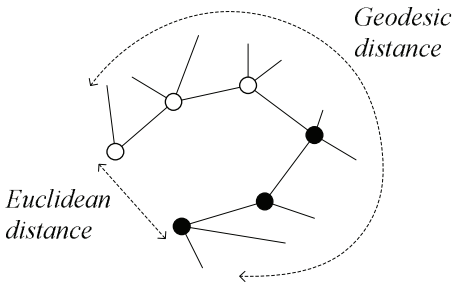


**Figure 6.14** Geodesic distance is calculated along the manifold as opposed to the Euclidean distance that does not use this information. After multidimensional scaling, these two instances from two classes will be mapped to faraway positions in the new space, though they are close in the original space.

to Euclidean distances. The geodesic distance between any two points is calculated as the length of the shortest path between the corresponding two nodes. For two points that are not close by, we need to hop over a number of intermediate points along the way, and therefore the distance will be the distance along the manifold, approximated as the sum of local Euclidean distances (see figure 6.14).

Two nodes $r$ and $s$ are connected if $\|\boldsymbol{x}^r - \boldsymbol{x}^s\| < \epsilon$ (while making sure that the graph is connected), or if $\boldsymbol{x}^s$ is one of the $n$ neighbors of $\boldsymbol{x}^r$ (while making sure that the distance matrix is symmetric), and we set the edge length to $\|\boldsymbol{x}^r - \boldsymbol{x}^s\|$. For any two nodes $r$ and $s$, $d_{rs}$ is the length of the shortest path between them. We then apply MDS on $d_{rs}$ to reduce dimensionality to $k$ using feature embedding by observing the proportion of variance explained. This will have the effect of placing $r$ and $s$ that are far apart in the geodesic space also far apart in the new $k$-dimensional space even if they are close in terms of Euclidean distance in the original $d$-dimensional space.

It is clear that the graph distances provide a better approximation as the number of points increases, though there is the trade-off of longer execution time; if time is critical, one can subsample and use a subset of "landmark points" to make the algorithm faster. The parameter $\epsilon$ needs to be carefully tuned; if it is too small, there may be more than one connected component, and if it is too large, "shortcut" edges may be added that corrupt the low-dimensional embedding (Balasubramanian et al. 2002).

One problem with Isomap, as with MDS, because it uses feature embedding, is that it places the $N$ points in a low-dimensional space, but it does not learn a general mapping function that will allow mapping a new test point; the new point should be added to the dataset and the whole algorithm needs to be run once more using $N + 1$ instances.

## 6.11 Locally Linear Embedding

LOCALLY LINEAR EMBEDDING

*Locally linear embedding* (LLE) recovers global nonlinear structure from locally linear fits (Roweis and Saul 2000). The idea is that each local patch of the manifold can be approximated linearly and given enough data, each point can be written as a linear, weighted sum of its neighbors (again either defined using a given number of neighbors, $n$, or distance threshold, $\epsilon$). Given $\boldsymbol{x}^r$ and its neighbors $\boldsymbol{x}^s_{(r)}$ in the original space, one can find the reconstruction weights $\mathbf{W}_{rs}$ that minimize the error function

$$\mathcal{E}^w(\mathbf{W}|\mathcal{X}) = \sum_r \|\boldsymbol{x}^r - \sum_s \mathbf{W}_{rs}\boldsymbol{x}^s_{(r)}\|^2 \tag{6.56}$$

using least squares subject to $\mathbf{W}_{rr} = 0, \forall r$ and $\sum_s \mathbf{W}_{rs} = 1$.

The idea in LLE is that the reconstruction weights $\mathbf{W}_{rs}$ reflect the intrinsic geometric properties of the data that we expect to be also valid for local patches of the manifold, that is, the new space we are mapping the instances to (see figure 6.15). The second step of LLE is hence to now keep the weights $\mathbf{W}_{rs}$ fixed and let the new coordinates $\boldsymbol{z}^r$ take whatever values they need respecting the interpoint constraints given by the weights:

$$\mathcal{E}^z(\mathcal{Z}|\mathbf{W}) = \sum_r \|\boldsymbol{z}^r - \sum_s \mathbf{W}_{rs}\boldsymbol{z}^s\|^2 \tag{6.57}$$

Nearby points in the original $d$-dimensional space should remain close and similarly colocated with respect to one another in the new, $k$-dimensional space. Equation 6.57 can be rewritten as

$$\mathcal{E}^z(\mathcal{Z}|\mathbf{W}) = \sum_{r,s} \mathbf{M}_{rs}(\boldsymbol{z}^r)^T\boldsymbol{z}^s \tag{6.58}$$

where

$$\mathbf{M}_{rs} = \delta_{rs} - \mathbf{W}_{rs} - \mathbf{W}_{sr} + \sum_i \mathbf{W}_{ir}\mathbf{W}_{is} \tag{6.59}$$

$\mathbf{M}$ is sparse (only a small percentage of data points are neighbors of a data point: $n \ll N$), symmetric, and positive semidefinite. As in other

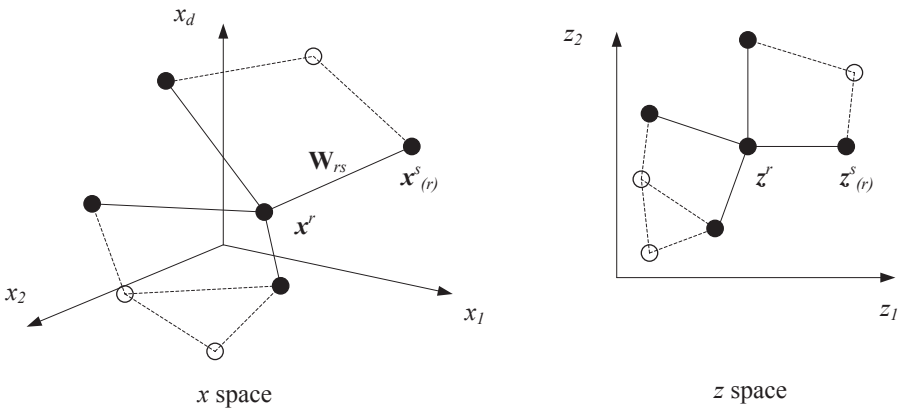


**Figure 6.15** Local linear embedding first learns the constraints in the original space and next places the points in the new space respecting those constraints. The constraints are learned using the immediate neighbors (shown with continuous lines) but also propagate to second-order neighbors (shown dashed).

dimensionality reduction methods, we require that the data be centered at the origin, $E[\mathbf{z}] = 0$, and that the new coordinates be uncorrelated and unit length: $\text{Cov}(\mathbf{z}) = \mathbf{I}$. The solution to equation 6.58 subject to these two constraints is given by the $k + 1$ eigenvectors with the smallest eigenvalues; we ignore the lowest one and the other $k$ eigenvectors give us the new coordinates.

Because the $n$ neighbors span a space of dimensionality $n - 1$ (you need distances to three points to uniquely specify your location in two dimensions), LLE can reduce dimensionality up to $k \leq n-1$. It is observed (Saul and Roweis 2003) that some margin between $k$ and $n$ is necessary to obtain a good embedding. Note that if $n$ (or $\epsilon$) is small, the graph (that is constructed by connecting each instance to its neighbors) may no longer be connected and it may be necessary to run LLE separately on separate components to find separate manifolds in different parts of the input space. On the other hand, if $n$ (or $\epsilon$) is taken large, some neighbors may be too far for the local linearity assumption to hold and this may corrupt the embedding. It is possible to use different $n$ (or $\epsilon$) in different parts of the input space based on some prior knowledge, but how this can be done is open to research (Saul and Roweis 2003).

As with Isomap, LLE solution is the set of new coordinates for the $N$ points, but we do not learn a mapping and hence cannot find $\boldsymbol{z}'$ for a new $\boldsymbol{x}'$. There are two solutions to this:

1. Using the same idea, one can find the $n$ neighbors of $\boldsymbol{x}'$ in the original $d$-dimensional space and first learn the reconstruction weights $\boldsymbol{w}_j$ that minimizes

$$\mathcal{E}^w(\boldsymbol{w}|\mathcal{X}) = \|\boldsymbol{x}' - \sum_s \boldsymbol{w}_s \boldsymbol{x}^s\|^2 \tag{6.60}$$

and then use them to reconstruct $\boldsymbol{z}'$ in the new $k$-dimensional space:

$$\boldsymbol{z}' = \sum_s \boldsymbol{w}_s \boldsymbol{z}^s \tag{6.61}$$

Note that this approach can also be used to interpolate from an Isomap (or MDS) solution. The drawback however is the need to store the whole set of $\{\boldsymbol{x}^t, \boldsymbol{z}^t\}_{t=1}^N$.

2. Using $\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{z}^t\}_{t=1}^N$ as a training set, one can train any regressor, $\boldsymbol{g}(\boldsymbol{x}^t|\theta)$—for example, a multilayer perceptron (chapter 11)—as a generalizer to approximate $\boldsymbol{z}^t$ from $\boldsymbol{x}^t$, whose parameters $\theta$ is learned to minimize the regression error:

$$\mathcal{E}(\theta|\mathcal{X}) = \sum_t \|\boldsymbol{z}^t - \boldsymbol{g}(\boldsymbol{x}^t|\theta)\|^2 \tag{6.62}$$

Once training is done, we can calculate $\boldsymbol{z}' = \boldsymbol{g}(\boldsymbol{x}'|\theta)$. The model $\boldsymbol{g}(\cdot)$ should be carefully chosen to be able to learn the mapping. There may no longer be a unique optimum and hence there are all the usual problems related to minimization, that is, initialization, local optima, convergence, and so on.

In both Isomap and LLE, there is local information that is propagated over neighbors to get a global solution. In Isomap, the geodesic distance is the sum of local distances; in LLE, the final optimization in placing $\boldsymbol{z}^t$ takes into account all local $\mathbf{W}_{rs}$ values. Let us say $a$ and $b$ are neighbors and $b$ and $c$ are neighbors. Though $a$ and $c$ may not be neighbors, there is dependence between $a$ and $c$ either through the graph, $d_{ac} = d_{ab} + d_{bc}$, or the weights $\mathbf{W}_{ab}$ and $\mathbf{W}_{bc}$. In both algorithms, the global nonlinear organization is found by integrating local linear constraints that overlap partially.

## 6.12 Laplacian Eigenmaps

Consider the data instance $\boldsymbol{x}^r \in \Re^d, r = 1, \ldots, N$ and its projection $\boldsymbol{z}^r \in \Re^k$. Let us say that we are given a similarity value $B_{rs}$ between pairs of instances possibly calculated in some high-dimensional space such that it takes its maximum value if $r$ and $s$ are the same and decreases as they become dissimilar. Assume that the minimum possible value is 0 and that it is symmetric: $B_{rs} = B_{sr}$ (Belkin and Nyogi 2003). The aim is to

$$\min \sum_{r,s} \|\boldsymbol{z}^r - \boldsymbol{z}^s\|^2 B_{rs} \tag{6.63}$$

Two instances that should be similar, that is, $r$ and $s$ whose $B_{rs}$ is high, should be placed nearby in the new space; hence $\boldsymbol{z}^r$ and $\boldsymbol{z}^s$ should be close. Whereas the more they are dissimilar, the less we care for their relative position in the new space. $B_{rs}$ are calculated in the original space; for example, if we use the dot product, the method would work similar to the way multidimensional scaling does:

$$B_{rs} = (\boldsymbol{x}^r)^T \boldsymbol{x}_s$$

LAPLACIAN EIGENMAPS

But what is done in *Laplacian eigenmaps*, similar to Isomap and LLE, is that we care for similarities only locally (Belkin and Nyogi 2003). We define a neighborhood either through some maximum $\epsilon$ distance between $\boldsymbol{x}^r$ and $\boldsymbol{x}^s$, or a $k$-nearest neighborhood, and outside of that we set $B_{rs}$ to 0. In the neighborhood, we use the Gaussian kernel to convert Euclidean distance to a similarity value:

$$B_{rs} = \exp\left[-\frac{\|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2}{2\sigma^2}\right]$$

for some user-defined $\sigma$ value. $\mathbf{B}$ can be seen as defining a weighted graph.

For the case of $k = 1$ (we reduce dimensionality to 1), we can rewrite equation 6.63 as

$$\begin{aligned}
\min \quad & \frac{1}{2}\sum_{r,s}(z_r - z_s)^2 B_{rs} \\
= \quad & \frac{1}{2}\left(\sum_{r,s} B_{rs} z_r^2 - 2\sum_{r,s} B_{rs} z_r z_s + \sum_{r,s} B_{rs}(z_s)^2\right) \\
= \quad & \frac{1}{2}\left(\sum_{r} d_r z_r^2 - 2\sum_{r,s} B_{rs} z_r z_s + \sum_{s} d_s z_s^2\right)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_r d_r z_r^2 - \sum_r \sum_s B_{rs} z_r z_s \\
&= \boldsymbol{z}^T \mathbf{D} \boldsymbol{z} - \boldsymbol{z}^T \mathbf{B} \boldsymbol{z} \qquad (6.64)
\end{aligned}
$$

where $d_r = \sum_s B_{rs}$. $\mathbf{D}$ is the diagonal matrix of $d_r$, and $\boldsymbol{z}$ is the $N$-dimensional column vector whose dimension $r$, $z_r$ is the new coordinate for $\boldsymbol{x}^r$. We define the *graph Laplacian*

$$\mathbf{L} = \mathbf{D} - \mathbf{B} \qquad (6.65)$$

and the aim is to minimize $\boldsymbol{z}^T \mathbf{L} \boldsymbol{z}$. For a unique solution, we require $\|\boldsymbol{z}\| = 1$. Just as in feature embedding, we get the coordinates in the new space directly without any extra projection and it can be shown that $\boldsymbol{z}$ should be an eigenvector of $\mathbf{L}$, and because we want to minimize, we choose the eigenvector with the smallest eigenvalue. Note, however, that there is at least one eigenvector with eigenvalue 0 and that should be ignored. That eigenvector has all its elements equal to each other: $\boldsymbol{c} = (1/\sqrt{N})\mathbf{1}^T$. The corresponding eigenvalue is 0 because

$$\mathbf{L}\boldsymbol{c} = \mathbf{D}\boldsymbol{c} - \mathbf{B}\boldsymbol{c} = 0$$

$\mathbf{D}$ has row sums in its diagonal, and the dot product of a row of $\mathbf{B}$ and $\mathbf{1}$ also takes a weighted sum; in this case, for equation 6.64 to be 0, with $B_{ij}$ nonnegative, $z_i$ and $z_j$ should be equal for all pairs of $i, j$, and for the norm to be 1, all should be $1/\sqrt{N}$. So, we need to skip the eigenvector with eigenvalue 0 and if we want to reduce dimensionality to $k > 1$, we need to take the next $k$.

The Laplacian eigenmap is a feature embedding method; that is, we find the coordinates in the new space directly and have no explicit model for mapping that we can later use for new instances.

We can compare equation 6.63 with equation 6.37 (Sammon stress in MDS). Here, the similarity in the original space is represented implicitly in $B_{rs}$ whereas in MDS, it is explicitly written as $\|\boldsymbol{x}^r - \boldsymbol{x}^s\|$. Another difference is that in MDS, we check for similarity between all pairs, whereas here the constraints are local (which are then propagated because those local neighborhoods partially overlap—as in Isomap and LLE).

For the four-dimensional Iris data, results after projection to two dimensions are given for MDS and Laplacian eigenmaps in figure 6.16. MDS here is equivalent to PCA, whereas we see that the Laplacian eigenmap projects similar instances nearby in the new space. This is why this method is a good way to preprocess the data before clustering; *spectral clustering*, which we discuss in section 7.7, uses this idea.

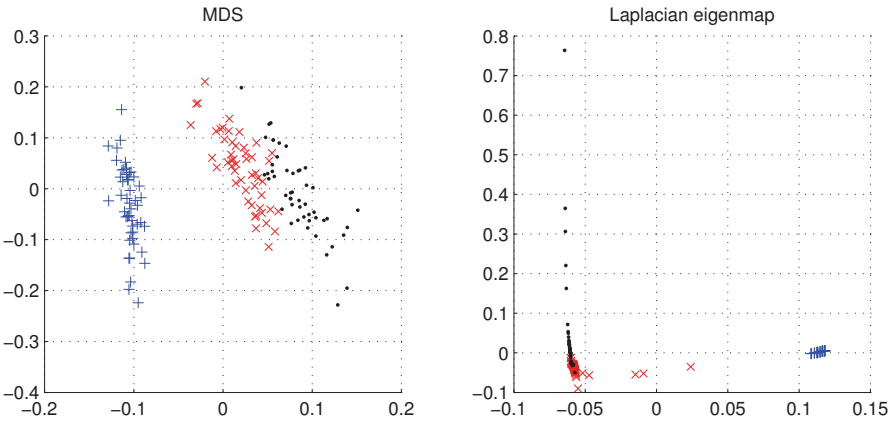


**Figure 6.16** Iris data reduced to two dimensions using multidimensional scaling and Laplacian eigenmaps. The latter leads to a more dense placement of similar instances.

## 6.13 Notes

Subset selection in regression is discussed in Miller 1990. The forward and backward search procedures we discussed are local search procedures. Fukunaga and Narendra (1977) proposed a branch and bound procedure. At considerable more expense, you can use a stochastic procedure like simulated annealing or genetic algorithms to search more widely in the the search space.

There are also *filtering* algorithms for feature selection where heuristic measures are used to calculate the "relevance" of a feature in a preprocessing stage without actually using the learner. For example, in the case of classification, instead of training a classifier and testing it at each step, one can use a separability measure, like the one used in linear discriminant analysis, to measure the quality of the new space in separating classes from each other (McLachlan 1992). With the cost of computation going down, it is best to include the learner in the loop because there is no guarantee that the heuristic used by the filter will match the bias of the learner that uses the features; no heuristic can replace the actual validation accuracy. A survey of feature selection methods is given by Guyon and Elisseeff (2003).

Projection methods work with numeric inputs, and discrete variables

should be represented by 0/1 dummy variables, whereas subset selection can use discrete inputs directly. Finding the eigenvectors and eigenvalues is quite straightforward and is part of any linear algebra package. Factor analysis was introduced by the British psychologist Charles Spearman to find the single factor for intelligence which explains the correlation between scores on various intelligence tests. The existence of such a single factor, called $g$, is highly disputed. More information on multidimensional scaling can be found in Cox and Cox 1994.

The projection methods we discussed are batch procedures in that they require that the whole sample be given before the projection directions are found. Mao and Jain (1995) discuss online procedures for doing PCA and LDA, where instances are given one by one and updates are done as new instances arrive. Another possibility for doing a nonlinear projection is when the estimator in Sammon mapping is taken as a nonlinear function, for example, a multilayer perceptron (section 11.11) (Mao and Jain 1995). It is also possible but much harder to do nonlinear factor analysis. When the models are nonlinear, it is difficult to come up with the right nonlinear model. You also need to use complicated optimization and approximation methods to solve for the model parameters.

Laplacian eigenmaps use the idea of feature embedding such that given pairwise similarities are preserved; the same idea is also used in kernel machines where pairwise similarities are given by a kernel function, and in chapter 13, we talk about "kernel" PCA, LDA, and CCA. Just as we implement polynomial regression by using linear regression where we consider high-order terms as additional inputs (section 5.8), we can do nonlinear dimensionality reduction by mapping to a new space by using nonlinear basis functions. That is the idea in kernel methods that allow us to go further than dot product or Euclidean distance for similarity calculation.

Matrix decomposition methods are quite popular in various big data applications because they allow us to explain a large data matrix using smaller matrices. One example application is *recommendation systems* where we may have millions of movies and millions of customers and entries are customer ratings. Note that most entries will be missing and the aim is to fill in those missing values and then do a recommendation based on those predicted values (Koren, Bell, and Volinsky 2009).

RECOMMENDATION SYSTEMS

There is a trade-off between feature extraction and decision making. If the feature extractor is good, the task of the classifier (or regressor) becomes trivial, for example, when the class code is extracted as a new

feature from the existing features. On the other hand, if the classifier is good enough, then there is no need for feature extraction; it does its automatic feature selection or combination internally. We live between these two ideal worlds.

There exist algorithms that do some feature selection internally, though in a limited way. Decision trees (chapter 9) do feature selection while generating the decision tree, and multilayer perceptrons (chapter 11) do nonlinear feature extraction in the hidden nodes. We expect to see more development along this line in embedding feature extraction in the actual step of classification/regression.

## 6.14 Exercises

1. Assuming that the classes are normally distributed, in subset selection, when one variable is added or removed, how can the new discriminant be calculated quickly? For example, how can the new $\mathbf{S}_{new}^{-1}$ be calculated from $\mathbf{S}_{old}^{-1}$?
2. Using Optdigits from the UCI repository, implement PCA. For various number of eigenvectors, reconstruct the digit images and calculate the reconstruction error (equation 6.12).
3. Plot the map of your state/country using MDS, given the road travel distances as input.
4. In Sammon mapping, if the mapping is linear, namely, $g(\boldsymbol{x}|\mathbf{W}) = \mathbf{W}^T\boldsymbol{x}$, how can $\mathbf{W}$ that minimizes the Sammon stress be calculated?
5. In figure 6.11, we see a synthetic two-dimensional data where LDA does a better job than PCA. Draw a similar dataset where PCA and LDA find the same good direction. Draw another where neither PCA nor LDA find a good direction.
6. Redo exercise 3, this time using Isomap where two cities are connected only if there is a direct road between them that does not pass through any other city.
7. In Isomap, instead of using Euclidean distance, we can also use Mahalanobis distance between neighboring points. What are the advantages and disadvantages of this approach, if any?
8. Multidimensional scaling can work as long as we have the pairwise distances between objects. We do not actually need to represent the objects as vectors at all as long as we have some measure of similarity. Can you give an example?

   SOLUTION: Let us say we have a database of documents. Then if $d_{rs}$ denotes the number of terms that documents $r$ and $s$ share, we can use MDS to map

these documents to a lower-dimensional space, for example, to visualize them and check for structure. Note that here we can count the number of shared terms without needing to explicitly represent the documents as vectors using, for example, the bag of words representation.

9. How can we incorporate class information into Isomap or LLE such that instances of the same class are mapped to nearby locations in the new space?

   SOLUTION: We can include an additional penalty term in calculating distances for instances belonging to different classes; MDS will then map instances of the same class to nearby points.

10. In factor analysis, how can we find the remaining ones if we already know some of the factors?

    SOLUTION: If we already know some of the factors, we can find their loadings by regression and then remove their effect from the data. We will then get the residual of what is not explained by those factors and look for additional factors that can explain this residual.

11. Discuss an application where there are hidden factors (not necessarily linear) and where factor analysis would be expected to work well.

    SOLUTION: One example is the data of student grades at a university. The grade a student gets for a set of courses depends on a number of hidden factors—for example, the student's aptitude for the subject, the amount of time he/she can allocate to studying, the comfort of his/her lodging, and so on.

## 6.15 References

Balasubramanian, M., E. L. Schwartz, J. B. Tenenbaum, V. de Silva, and J. C. Langford. 2002. "The Isomap Algorithm and Topological Stability." *Science* 295:7.

Bach, F., and M. I. Jordan. 2005. *A Probabilistic Interpretation of Canonical Correlation Analysis*. Technical Report 688, Department of Statistics, University of California, Berkeley.

Belkin, M., and P. Niyogi. 2003. "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation." *Neural Computation* 15:1373–1396.

Chatfield, C., and A. J. Collins. 1980. *Introduction to Multivariate Analysis*. London: Chapman and Hall.

Cox, T. F., and M. A. A. Cox. 1994. *Multidimensional Scaling*. London: Chapman and Hall.

Flury, B. 1988. *Common Principal Components and Related Multivariate Models*. New York: Wiley.

Fukunaga, K., and P. M. Narendra. 1977. "A Branch and Bound Algorithm for Feature Subset Selection." *IEEE Transactions on Computers* C-26:917–922.

Guyon, I., and A. Elisseeff. 2003. "An Introduction to Variable and Feature Selection." *Journal of Machine Learning Research* 3:1157–1182.

Hardoon, D. R., S. Szedmak, J. Shawe-Taylor. 2004. "Canonical Correlation Analysis: An Overview with Application to Learning Methods." *Neural Computation* 16:2639–2664.

Hastie, T. J., R. J. Tibshirani, and A. Buja. 1994. "Flexible Discriminant Analysis by Optimal Scoring." *Journal of the American Statistical Association* 89:1255–1270.

Kohavi, R., and G. John. 1997. "Wrappers for Feature Subset Selection." *Artificial Intelligence* 97:273–324.

Koren, Y., R. Bell, and C. Volinsky. 2009. "Matrix Factorization Techniques for Recommender Systems." *IEEE Computer* 42 (8): 30–37.

Landauer, T. K., D. Laham, and M. Derr. 2004. "From Paragraph to Graph: Latent Semantic Analysis for Information Visualization." *Proceedings of the National Academy of Sciences* 101 (suppl. 1): 5214–5219.

Lee, D. D., and H. S. Seung. 1999. "Learning the Parts of Objects by Non-Negative Matrix Factorization," *Nature* 401 (6755): 788–791.

Mao, J., and A. K. Jain. 1995. "Artificial Neural Networks for Feature Extraction and Multivariate Data Projection." *IEEE Transactions on Neural Networks* 6: 296–317.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition*. New York: Wiley.

Miller, A. J. 1990. *Subset Selection in Regression*. London: Chapman and Hall.

Pudil, P., J. Novovičová, and J. Kittler. 1994. "Floating Search Methods in Feature Selection." *Pattern Recognition Letters* 15:1119–1125.

Rencher, A. C. 1995. *Methods of Multivariate Analysis*. New York: Wiley.

Roweis, S. T., and L. K. Saul. 2000. "Nonlinear Dimensionality Reduction by Locally Linear Embedding." *Science* 290:2323–2326.

Saul, K. K., and S. T. Roweis. 2003. "Think Globally, Fit Locally: Unsupervised Learning of Low Dimensional Manifolds." *Journal of Machine Learning Research* 4:119–155.

Strang, G. 2006. *Linear Algebra and Its Applications*, 4th ed. Boston: Cengage Learning.

Tenenbaum, J. B., V. de Silva, and J. C. Langford. 2000. "A Global Geometric Framework for Nonlinear Dimensionality Reduction." *Science* 290:2319–2323.

Tipping, M. E., and C. M. Bishop. 1999. "Probabilistic Principal Component Analysis." *Journal of the Royal Statistical Society Series B* 61:611–622.

Turk, M., and A. Pentland. 1991. "Eigenfaces for Recognition." *Journal of Cognitive Neuroscience* 3:71–86.

Webb, A. 1999. *Statistical Pattern Recognition*. London: Arnold.

# 7 *Clustering*

*In the parametric approach, we assumed that the sample comes from a known distribution. In cases when such an assumption is untenable, we relax this assumption and use a semiparametric approach that allows a mixture of distributions to be used for estimating the input sample. Clustering methods allow learning the mixture parameters from data. In addition to probabilistic modeling, we discuss vector quantization, spectral clustering, and hierarchical clustering.*

## 7.1 Introduction

IN CHAPTERS 4 and 5, we discussed the parametric method for density estimation where we assumed that the sample $\mathcal{X}$ is drawn from some parametric family, for example, Gaussian. In parametric classification, this corresponds to assuming a certain density for the class densities $p(\boldsymbol{x}|C_i)$. The advantage of any parametric approach is that given a model, the problem reduces to the estimation of a small number of parameters, which, in the case of density estimation, are the sufficient statistics of the density, for example, the mean and covariance in the case of Gaussian densities.

Though parametric approaches are used quite frequently, assuming a rigid parametric model may be a source of bias in many applications where this assumption does not hold. We thus need more flexible models. In particular, assuming Gaussian density corresponds to assuming that the sample, for example, instances of a class, forms one single group in the $d$-dimensional space, and as we saw in chapter 5, the center and the shape of this group is given by the mean and the covariance respectively.

In many applications, however, the sample is not one group; there may be several groups. Consider the case of optical character recognition: There are two ways of writing the digit 7; the American writing is '7', whereas the European writing style has a horizontal bar in the middle (to tell it apart from the European '1', which keeps the small stroke on top in handwriting). In such a case, when the sample contains examples from both continents, the class for the digit 7 should be represented as the disjunction of two groups. If each of these groups can be represented by a Gaussian, the class can be represented by a *mixture* of two Gaussians, one for each writing style.

A similar example is in speech recognition where the same word can be uttered in different ways, due to different pronounciation, accent, gender, age, and so forth. Thus when there is not a single, universal prototype, all these different ways should be represented in the density to be statistically correct.

SEMIPARAMETRIC DENSITY ESTIMATION

We call this approach *semiparametric density estimation*, as we still assume a parametric model for each group in the sample. We discuss the *nonparametric* approach in chapter 8, which is used when there is no structure to the data and even a mixture model is not applicable. In this chapter, we focus on density estimation and defer supervised learning to chapter 12.

## 7.2 Mixture Densities

MIXTURE DENSITY

The *mixture density* is written as

$$p(\mathbf{x}) = \sum_{i=1}^{k} p(\mathbf{x}|\mathcal{G}_i) P(\mathcal{G}_i) \tag{7.1}$$

MIXTURE COMPONENTS
GROUPS
CLUSTERS
COMPONENT DENSITIES
MIXTURE PROPORTIONS

where $\mathcal{G}_i$ are the *mixture components*. They are also called *group* or *clusters*. $p(\mathbf{x}|\mathcal{G}_i)$ are the *component densities* and $P(\mathcal{G}_i)$ are the *mixture proportions*. The number of components, $k$, is a hyperparameter and should be specified beforehand. Given a sample and $k$, learning corresponds to estimating the component densities and proportions. When we assume that the component densities obey a parametric model, we need only estimate their parameters. If the component densities are multivariate Gaussian, we have $p(\mathbf{x}|\mathcal{G}_i) \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, and $\Phi = \{P(\mathcal{G}_i), \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}_{i=1}^{k}$ are the parameters that should be estimated from the iid sample $\mathcal{X} = \{\mathbf{x}^t\}_t$.

Parametric classification is a bona fide mixture model where groups, $\mathcal{G}_i$, correspond to classes, $C_i$, component densities $p(\boldsymbol{x}|\mathcal{G}_i)$ correspond to class densities $p(\boldsymbol{x}|C_i)$, and $P(\mathcal{G}_i)$ correspond to class priors, $P(C_i)$:

$$p(\boldsymbol{x}) = \sum_{i=1}^{K} p(\boldsymbol{x}|C_i)P(C_i)$$

In this *supervised* case, we know how many groups there are and learning the parameters is trivial because we are given the labels, namely, which instance belongs to which class (component). We remember from chapter 5 that when we are given the sample $\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{r}^t\}_{t=1}^N$, where $r_i^t = 1$ if $\boldsymbol{x}^t \in C_i$ and 0 otherwise, the parameters can be calculated using maximum likelihood. When each class is Gaussian distributed, we have a Gaussian mixture, and the parameters are estimated as

$$\begin{aligned} \hat{P}(C_i) &= \frac{\sum_t r_i^t}{N} \\ \boldsymbol{m}_i &= \frac{\sum_t r_i^t \boldsymbol{x}^t}{\sum_t r_i^t} \\ \mathbf{S}_i &= \frac{\sum_t r_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T}{\sum_t r_i^t} \end{aligned} \tag{7.2}$$

The difference in this chapter is that the sample is $\mathcal{X} = \{\boldsymbol{x}^t\}_t$: We have an *unsupervised learning* problem. We are given only $\boldsymbol{x}^t$ and not the labels $\boldsymbol{r}^t$, that is, we do not know which $\boldsymbol{x}^t$ comes from which component. So we should estimate both: First, we should estimate the labels, $r_i^t$, the component that a given instance belongs to; and, second, once we estimate the labels, we should estimate the parameters of the components given the set of instances belonging to them. We first discuss a simple algorithm, $k$-means clustering, for this purpose and later on show that it is a special case of the *expectation-maximization* (EM) algorithm.

## 7.3 $k$-Means Clustering

Let us say we have an image that is stored with 24 bits/pixel and can have up to 16 million colors. Assume we have a color screen with 8 bits/pixel that can display only 256 colors. We want to find the best 256 colors among all 16 million colors such that the image using only the 256 colors in the palette looks as close as possible to the original image. This is *color quantization* where we map from high to lower resolution. In the general

COLOR QUANTIZATION

case, the aim is to map from a continuous space to a discrete space; this process is called *vector quantization*.

VECTOR QUANTIZATION

Of course we can always quantize uniformly, but this wastes the colormap by assigning entries to colors not existing in the image, or would not assign extra entries to colors frequently used in the image. For example, if the image is a seascape, we expect to see many shades of blue and maybe no red. So the distribution of the colormap entries should reflect the original density as close as possible placing many entries in high-density regions, discarding regions where there is no data.

REFERENCE VECTORS

Let us say we have a sample of $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$. We have $k$ *reference vectors*, $\boldsymbol{m}_j, j = 1, \ldots, k$. In our example of color quantization, $\boldsymbol{x}^t$ are the image pixel values in 24 bits and $\boldsymbol{m}_j$ are the color map entries also in 24 bits, with $k = 256$.

Assume for now that we somehow have the $\boldsymbol{m}_j$ values; we discuss how to learn them shortly. Then in displaying the image, given the pixel, $\boldsymbol{x}^t$, we represent it with the most similar entry, $\boldsymbol{m}_i$ in the color map, satisfying

$$\|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\|$$

CODEBOOK VECTORS
CODE WORDS

That is, instead of the original data value, we use the closest value we have in the alphabet of reference vectors. $\boldsymbol{m}_i$ are also called *codebook vectors* or *code words*, because this is a process of *encoding/decoding* (see figure 7.1): Going from $\boldsymbol{x}^t$ to $i$ is a process of encoding the data using the codebook of $\boldsymbol{m}_i, i = 1, \ldots, k$ and, on the receiving end, generating $\boldsymbol{m}_i$ from $i$ is decoding. Quantization also allows *compression*: For example, instead of using 24 bits to store (or transfer over a communication line) each $\boldsymbol{x}^t$, we can just store/transfer its index $i$ in the colormap using 8 bits to index any one of 256, and we get a compression rate of almost 3; there is also the color map to store/transfer.

COMPRESSION

Let us see how we can calculate $\boldsymbol{m}_i$: When $\boldsymbol{x}^t$ is represented by $\boldsymbol{m}_i$, there is an error that is proportional to the distance, $\|\boldsymbol{x}^t - \boldsymbol{m}_i\|$. For the new image to look like the original image, we should have these distances as small as possible for all pixels. The total *reconstruction error* is defined as

RECONSTRUCTION ERROR

$$E(\{\boldsymbol{m}_i\}_{i=1}^k | \mathcal{X}) = \sum_t \sum_i b_i^t \|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2 \tag{7.3}$$

where

$$b_i^t = \begin{cases} 1 & \text{if } \|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\| \\ 0 & \text{otherwise} \end{cases} \tag{7.4}$$

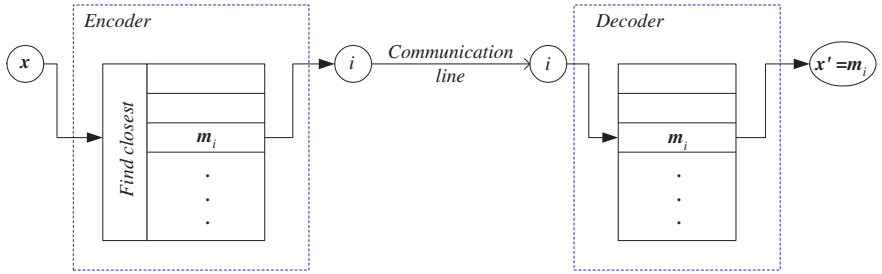


**Figure 7.1** Given $\boldsymbol{x}$, the encoder sends the index of the closest code word and the decoder generates the code word with the received index as $\boldsymbol{x}'$. Error is $\|\boldsymbol{x}' - \boldsymbol{x}\|^2$.

*k*-MEANS CLUSTERING

The best reference vectors are those that minimize the total reconstruction error. $b_i^t$ also depend on $\boldsymbol{m}_i$, and we cannot solve this optimization problem analytically. We have an iterative procedure named *k-means clustering* for this: First, we start with some $\boldsymbol{m}_i$ initialized randomly. Then at each iteration, we first use equation 7.4 and calculate $b_i^t$ for all $\boldsymbol{x}^t$, which are the *estimated labels*; if $b_i^t$ is 1, we say that $\boldsymbol{x}^t$ belongs to the group of $\boldsymbol{m}_i$. Then, once we have these labels, we minimize equation 7.3. Taking its derivative with respect to $\boldsymbol{m}_i$ and setting it to 0, we get

$$\boldsymbol{m}_i = \frac{\sum_t b_i^t \boldsymbol{x}^t}{\sum_t b_i^t} \tag{7.5}$$

The reference vector is set to the mean of all the instances that it represents. Note that this is the same as the formula for the mean in equation 7.2, except that we place the estimated labels $b_i^t$ in place of the labels $r_i^t$. This is an iterative procedure because once we calculate the new $\boldsymbol{m}_i$, $b_i^t$ change and need to be recalculated, which in turn affect $\boldsymbol{m}_i$. These two steps are repeated until $\boldsymbol{m}_i$ stabilize (see figure 7.2). The pseudocode of the *k*-means algorithm is given in figure 7.3.

One disadvantage is that this is a local search procedure, and the final $\boldsymbol{m}_i$ highly depend on the initial $\boldsymbol{m}_i$. There are various methods for initialization:

- We can simply take randomly selected $k$ instances as the initial $\boldsymbol{m}_i$.
- The mean of all data can be calculated and small random vectors may be added to the mean to get the $k$ initial $\boldsymbol{m}_i$.

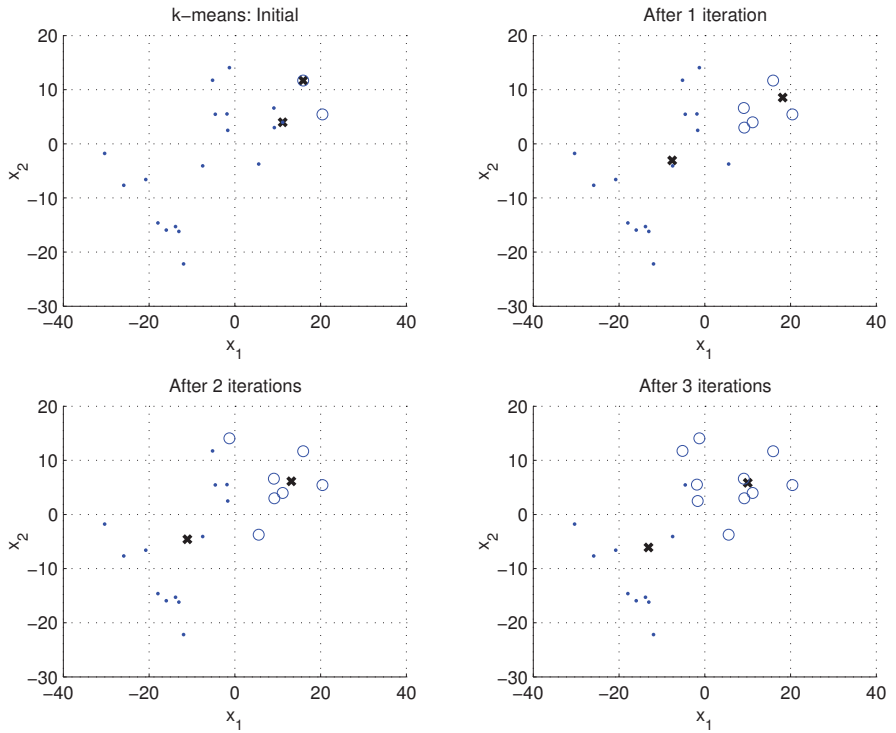


**Figure 7.2** Evolution of $k$-means. Crosses indicate center positions. Data points are marked depending on the closest center.

- We can calculate the principal component, divide its range into $k$ equal intervals, partitioning the data into $k$ groups, and then take the means of these groups as the initial centers.

After convergence, all the centers should cover some subset of the data instances and be useful; therefore, it is best to initialize centers where we believe there is data.

LEADER CLUSTER ALGORITHM

There are also algorithms for adding new centers *incrementally* or deleting empty ones. In *leader cluster algorithm*, an instance that is far away from existing centers (defined by a threshold value) causes the creation of a new center at that point (we discuss such a neural network algorithm, ART, in chapter 12). Or, a center that covers a large number of instances ($\sum_t b_i^t / N > \theta$) can be split into two (by adding a small random vector to

Initialize $\mathbf{m}_i, i = 1, \ldots, k$, for example, to $k$ random $\mathbf{x}^t$
Repeat
  For all $\mathbf{x}^t \in \mathcal{X}$
    $b_i^t \leftarrow \begin{cases} 1 & \text{if } \|\mathbf{x}^t - \mathbf{m}_i\| = \min_j \|\mathbf{x}^t - \mathbf{m}_j\| \\ 0 & \text{otherwise} \end{cases}$
  For all $\mathbf{m}_i, i = 1, \ldots, k$
    $\mathbf{m}_i \leftarrow \sum_t b_i^t \mathbf{x}^t / \sum_t b_i^t$
Until $\mathbf{m}_i$ converge

**Figure 7.3** $k$-means algorithm.

one of the two copies to make them different). Similarly, a center that covers too few instances can be removed and restarted from some other part of the input space.

$k$-means algorithm is for clustering, that is, for finding groups in the data, where the groups are represented by their centers, which are the typical representatives of the groups. Vector quantization is one application of clustering, but clustering is also used for *preprocessing* before a later stage of classification or regression. Given $\mathbf{x}^t$, when we calculate $b_i^t$, we do a mapping from the original space to the $k$-dimensional space, that is, to one of the corners of the $k$-dimensional hypercube. A regression or discriminant function can then be learned in this new space; we discuss such methods in chapter 12.

## 7.4 Expectation-Maximization Algorithm

In $k$-means, we approached clustering as the problem of finding codebook vectors that minimize the total reconstruction error. In this section, our approach is probabilistic and we look for the component density parameters that maximize the likelihood of the sample. Using the mixture model of equation 7.1, the log likelihood given the sample $\mathcal{X} = \{\mathbf{x}^t\}_t$ is

$$\begin{aligned} \mathcal{L}(\Phi|\mathcal{X}) &= \log \prod_t p(\mathbf{x}^t|\Phi) \\ &= \sum_t \log \sum_{i=1}^k p(\mathbf{x}^t|\mathcal{G}_i) P(\mathcal{G}_i) \end{aligned} \tag{7.6}$$

where $\Phi$ includes the priors $P(\mathcal{G}_i)$ and also the sufficient statistics of the

component densities $p(\mathbf{x}^t|\mathcal{G}_i)$. Unfortunately, we cannot solve for the parameters analytically and need to resort to iterative optimization.

EXPECTATION-MAXIMIZATION

The *expectation-maximization* algorithm (Dempster, Laird, and Rubin 1977; Redner and Walker 1984) is used in maximum likelihood estimation where the problem involves two sets of random variables of which one, $X$, is observable and the other, $Z$, is hidden. The goal of the algorithm is to find the parameter vector $\Phi$ that maximizes the likelihood of the observed values of $X$, $\mathcal{L}(\Phi|\mathcal{X})$. But in cases where this is not feasible, we associate the extra *hidden variables* $Z$ and express the underlying model using both, to maximize the likelihood of the joint distribution of $X$ and $Z$, the *complete* likelihood $\mathcal{L}_c(\Phi|\mathcal{X},\mathcal{Z})$.

Since the $Z$ values are not observed, we cannot work directly with the complete data likelihood $\mathcal{L}_c$; instead, we work with its expectation, $\mathcal{Q}$, given $\mathcal{X}$ and the current parameter values $\Phi^l$, where $l$ indexes iteration. This is the *expectation* (E) step of the algorithm. Then in the *maximization* (M) step, we look for the new parameter values, $\Phi^{l+1}$, that maximize this. Thus

$$\begin{aligned}\text{E-step} &: \mathcal{Q}(\Phi|\Phi^l) = E[\mathcal{L}_c(\Phi|\mathcal{X},\mathcal{Z})|\mathcal{X},\Phi^l]\\ \text{M-step} &: \Phi^{l+1} = \arg\max_{\Phi} \mathcal{Q}(\Phi|\Phi^l)\end{aligned}$$

Dempster, Laird, and Rubin (1977) proved that an increase in $\mathcal{Q}$ implies an increase in the incomplete likelihood

$$\mathcal{L}(\Phi^{l+1}|\mathcal{X}) \geq \mathcal{L}(\Phi^l|\mathcal{X})$$

In the case of mixtures, the hidden variables are the sources of observations, namely, which observation belongs to which component. If these were given, for example, as class labels in a supervised setting, we would know which parameters to adjust to fit that data point. The EM algorithm works as follows: in the E-step we estimate these labels given our current knowledge of components, and in the M-step we update our component knowledge given the labels estimated in the E-step. These two steps are the same as the two steps of $k$-means; calculation of $b_i^t$ (E-step) and reestimation of $\mathbf{m}_i$ (M-step).

We define a vector of *indicator variables* $\mathbf{z}^t = \{z_1^t, \ldots, z_k^t\}$ where $z_i^t = 1$ if $\mathbf{x}^t$ belongs to cluster $\mathcal{G}_i$, and 0 otherwise. $\mathbf{z}$ is a multinomial distribution from $k$ categories with prior probabilities $\pi_i$, shorthand for $P(\mathcal{G}_i)$.

Then

$$P(\mathbf{z}^t) = \prod_{i=1}^{k} \pi_i^{z_i^t} \tag{7.7}$$

The likelihood of an observation $\mathbf{x}^t$ is equal to its probability specified by the component that generated it:

$$p(\mathbf{x}^t|\mathbf{z}^t) = \prod_{i=1}^{k} p_i(\mathbf{x}^t)^{z_i^t} \tag{7.8}$$

$p_i(\mathbf{x}^t)$ is shorthand for $p(\mathbf{x}^t|\mathcal{G}_i)$. The joint density is

$$p(\mathbf{x}^t, \mathbf{z}^t) = P(\mathbf{z}^t)p(\mathbf{x}^t|\mathbf{z}^t)$$

and the complete data likelihood of the iid sample $\mathcal{X}$ is

$$\begin{aligned}
\mathcal{L}_c(\Phi|\mathcal{X}, \mathcal{Z}) &= \log \prod_t p(\mathbf{x}^t, \mathbf{z}^t|\Phi) \\
&= \sum_t \log p(\mathbf{x}^t, \mathbf{z}^t|\Phi) \\
&= \sum_t \log P(\mathbf{z}^t|\Phi) + \log p(\mathbf{x}^t|\mathbf{z}^t, \Phi) \\
&= \sum_t \sum_i z_i^t [\log \pi_i + \log p_i(\mathbf{x}^t|\Phi)]
\end{aligned}$$

**E-step**: We define

$$\begin{aligned}
\mathcal{Q}(\Phi|\Phi^l) &\equiv E\left[\log P(X, Z)|\mathcal{X}, \Phi^l\right] \\
&= E\left[\mathcal{L}_c(\Phi|\mathcal{X}, \mathcal{Z})|\mathcal{X}, \Phi^l\right] \\
&= \sum_t \sum_i E[z_i^t|\mathcal{X}, \Phi^l][\log \pi_i + \log p_i(\mathbf{x}^t|\Phi^l)]
\end{aligned}$$

where

$$\begin{aligned}
E[z_i^t|\mathcal{X}, \Phi^l] &= E[z_i^t|\mathbf{x}^t, \Phi^l] \quad \mathbf{x}^t \text{ are iid} \\
&= P(z_i^t = 1|\mathbf{x}^t, \Phi^l) \quad z_i^t \text{ is a 0/1 random variable} \\
&= \frac{p(\mathbf{x}^t|z_i^t = 1, \Phi^l)P(z_i^t = 1|\Phi^l)}{p(\mathbf{x}^t|\Phi^l)} \quad \text{Bayes' rule} \\
&= \frac{p_i(\mathbf{x}^t|\Phi^l)\pi_i}{\sum_j p_j(\mathbf{x}^t|\Phi^l)\pi_j} \\
&= \frac{p(\mathbf{x}^t|\mathcal{G}_i, \Phi^l)P(\mathcal{G}_i)}{\sum_j p(\mathbf{x}^t|\mathcal{G}_j, \Phi^l)P(\mathcal{G}_j)} \\
&= P(\mathcal{G}_i|\mathbf{x}^t, \Phi^l) \equiv h_i^t
\end{aligned} \tag{7.9}$$

We see that the expected value of the hidden variable, $E[z_i^t]$, is the posterior probability that $\boldsymbol{x}^t$ is generated by component $\mathcal{G}_i$. Because this is a probability, it is between 0 and 1 and is a "soft" label, as opposed to the 0/1 "hard" label of $k$-means.

**M-step**: We maximize $\mathcal{Q}$ to get the next set of parameter values $\Phi^{l+1}$:

$$\Phi^{l+1} = \arg\max_{\Phi} \mathcal{Q}(\Phi|\Phi^l)$$

which is

$$\begin{aligned}\mathcal{Q}(\Phi|\Phi^l) &= \sum_t \sum_i h_i^t [\log \pi_i + \log p_i(\boldsymbol{x}^t|\Phi^l)] \\ &= \sum_t \sum_i h_i^t \log \pi_i + \sum_t \sum_i h_i^t \log p_i(\boldsymbol{x}^t|\Phi^l) \end{aligned} \tag{7.10}$$

The second term is independent of $\pi_i$ and using the constraint that $\sum_i \pi_i = 1$ as the Lagrangian, we solve for

$$\nabla_{\pi_i} \sum_t \sum_i h_i^t \log \pi_i - \lambda \left( \sum_i \pi_i - 1 \right) = 0$$

and get

$$\pi_i = \frac{\sum_t h_i^t}{N} \tag{7.11}$$

which is analogous to the calculation of priors in equation 7.2.

Similarly, the first term of equation 7.10 is independent of the components and can be dropped while estimating the parameters of the components. We solve for

$$\nabla_{\Phi} \sum_t \sum_i h_i^t \log p_i(\boldsymbol{x}^t|\Phi) = 0 \tag{7.12}$$

If we assume Gaussian components, $\hat{p}_i(\boldsymbol{x}^t|\Phi) \sim \mathcal{N}(\boldsymbol{m}_i, \mathbf{S}_i)$, the M-step is

$$\begin{aligned}\boldsymbol{m}_i^{l+1} &= \frac{\sum_t h_i^t \boldsymbol{x}^t}{\sum_t h_i^t} \\ \mathbf{S}_i^{l+1} &= \frac{\sum_t h_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i^{l+1})(\boldsymbol{x}^t - \boldsymbol{m}_i^{l+1})^T}{\sum_t h_i^t} \end{aligned} \tag{7.13}$$

where, for Gaussian components in the E-step, we calculate

$$h_i^t = \frac{\pi_i |\mathbf{S}_i|^{-1/2} \exp[-(1/2)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T \mathbf{S}_i^{-1} (\boldsymbol{x}^t - \boldsymbol{m}_i)]}{\sum_j \pi_j |\mathbf{S}_j|^{-1/2} \exp[-(1/2)(\boldsymbol{x}^t - \boldsymbol{m}_j)^T \mathbf{S}_j^{-1} (\boldsymbol{x}^t - \boldsymbol{m}_j)]} \tag{7.14}$$

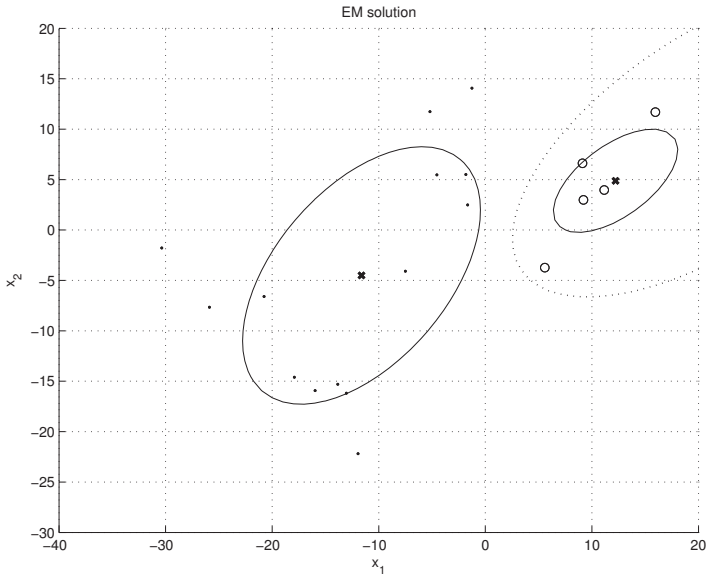


**Figure 7.4** Data points and the fitted Gaussians by EM, initialized by one $k$-means iteration of figure 7.2. Unlike in $k$-means, as can be seen, EM allows estimating the covariance matrices. The data points labeled by greater $h_i$, the contours of the estimated Gaussian densities, and the separating curve of $h_i = 0.5$ (dashed line) are shown.

Again, the similarity between equations 7.13 and 7.2 is not accidental; the estimated soft labels $h_i^t$ replace the actual (unknown) labels $r_i^t$.

EM is initalized by $k$-means. After a few iterations of $k$-means, we get the estimates for the centers $\boldsymbol{m}_i$, and using the instances covered by each center, we estimate the $\mathbf{S}_i$ and $\sum_t b_i^t / N$ give us the $\pi_i$. We run EM from that point on, as shown in figure 7.4.

Just as in parametric classification (section 5.5), with small samples and large dimensionality we can regularize by making simplifying assumptions. When $\hat{p}_i(\boldsymbol{x}^t|\Phi) \sim \mathcal{N}(\boldsymbol{m}_i, \mathbf{S})$, the case of a shared covariance matrix, equation 7.12 reduces to

$$\min_{\boldsymbol{m}_i, \mathbf{S}} \sum_t \sum_i h_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)^T \mathbf{S}^{-1} (\boldsymbol{x}^t - \boldsymbol{m}_i) \tag{7.15}$$

When $\hat{p}_i(\boldsymbol{x}^t|\Phi) \sim \mathcal{N}(\boldsymbol{m}_i, s^2\mathbf{I})$, the case of a shared diagonal matrix, we

have

$$\min_{\mathbf{m}_i,s} \sum_t \sum_i h_i^t \frac{\|\mathbf{x}^t - \mathbf{m}_i\|^2}{s^2} \tag{7.16}$$

which is the reconstruction error we defined in $k$-means clustering (equation 7.3). The difference is that now

$$h_i^t = \frac{\exp\left[-(1/2s^2)\|\mathbf{x}^t - \mathbf{m}_i\|^2\right]}{\sum_j \exp\left[-(1/2s^2)\|\mathbf{x}^t - \mathbf{m}_j\|^2\right]} \tag{7.17}$$

is a probability between 0 and 1. $b_i^t$ of $k$-means clustering makes a hard 0/1 decision, whereas $h_i^t$ is a *soft label* that assigns the input to a cluster with a certain probability. When $h_i^t$ are used instead of $b_i^t$, an instance contributes to the update of parameters of all components, to each proportional to that probability. This is especially useful if the instance is close to the midpoint between two centers.

We thus see that $k$-means clustering is a special case of EM applied to Gaussian mixtures where inputs are assumed independent with equal and shared variances, all components have equal priors, and labels are hardened. $k$-means thus pave the input density with circles, whereas EM in the general case uses ellipses of arbitrary shapes, orientations, and coverage proportions.

## 7.5 Mixtures of Latent Variable Models

When full covariance matrices are used with Gaussian mixtures, even if there is no singularity, one risks overfitting if the input dimensionality is high and the sample is small. To decrease the number of parameters, assuming a common covariance matrix may not be right since clusters may really have different shapes. Assuming diagonal matrices is even more risky because it removes all correlations.

The alternative is to do dimensionality reduction in the clusters. This decreases the number of parameters while still capturing the correlations. The number of free parameters is controlled through the dimensionality of the reduced space.

When we do factor analysis (section 6.5) in the clusters, we look for *latent* or *hidden variables* or *factors* that generate the data in the clusters (Bishop 1999):

$$p(\mathbf{x}^t|\mathcal{G}_i) \sim \mathcal{N}(\mathbf{m}_i, \mathbf{V}_i\mathbf{V}_i^T + \mathbf{\Psi}_i) \tag{7.18}$$

where $\mathbf{V}_i$ and $\mathbf{\Psi}_i$ are the factor loadings and specific variances of cluster $\mathcal{G}_i$. Rubin and Thayer (1982) give EM equations for factor analysis. It is possible to extend this in mixture models to find *mixtures of factor analyzers* (Ghahramani and Hinton 1997). In the E-step, in equation 7.9, we use equation 7.18, and in the M-step, we solve equation 7.12 for $\mathbf{V}_i$ and $\mathbf{\Psi}_i$ instead of $\mathbf{S}_i$. Similarly, one can also do PCA in groups, which is called *mixtures of probabilistic principal component analyzers* (Tipping and Bishop 1999).

MIXTURES OF FACTOR ANALYZERS

MIXTURES OF PROBABILISTIC PRINCIPAL COMPONENT ANALYZERS

We can of course use EM to learn $\mathbf{S}_i$ and then do FA or PCA separately in each cluster, but doing EM is better because it couples these two steps of clustering and dimensionality reduction and does a soft partitioning. An instance contributes to the calculation of the latent variables of all groups, weighted by $h_i^t$.

## 7.6 Supervised Learning after Clustering

Clustering, just as the dimensionality reduction methods discussed in chapter 6, can be used for two purposes. First, it can be used for data exploration, to understand the structure of data. Second, it can be used to map data to a new space where supervised learning is easier.

Dimensionality reduction methods are used to find correlations between variables and thus group variables; clustering methods, on the other hand, are used to find similarities between instances and thus group instances. If such groups are found, these may be named (by application experts) and their attributes be defined. One can choose the group mean as the representative prototype of instances in the group, or the possible range of attributes can be written. This allows a simpler description of the data. For example, if the customers of a company seem to fall in one of $k$ groups, called *segments*, customers being defined in terms of their demographic attributes and transactions with the company, then a better understanding of the customer base will be provided that will allow the company to provide different strategies for different types of customers; this is part of *customer relationship management* (CRM). Likewise, the company will also be able to develop strategies for those customers who do not fall in any large group, and who may require attention—for example, churning customers.

CUSTOMER SEGMENTATION

CUSTOMER RELATIONSHIP MANAGEMENT

Frequently, clustering is also used as a preprocessing stage. Just like the dimensionality reduction methods of chapter 6 allowed us to make

a mapping to a new space, after clustering, we also map to a new $k$-dimensional space where the dimensions are $h_i$ (or $b_i$ at the risk of loss of information). In a supervised setting, we can then learn the discriminant or regression function in this new space. The difference from dimensionality reduction methods like PCA however is that $k$, the dimensionality of the new space, can be larger than $d$, the original dimensionality.

When we use a method like PCA, where the new dimensions are combinations of the original dimensions, to represent any instance in the new space, all dimensions contribute; that is, all $z_j$ are nonzero. In the case of a method like clustering where the new dimensions are defined locally, there are many more new dimensions, $b_j$, but only one (or if we use $h_j$, few) of them have a nonzero value. In the former case, where there are few dimensions but all contribute, we have a *distributed representation*; in the latter case, where there are many dimensions but few contribute, we have a *local representation*.

DISTRIBUTED VS. LOCAL REPRESENTATION

One advantage of preceding a supervised learner with unsupervised clustering or dimensionality reduction is that the latter does not need labeled data. Labeling the data is costly. We can use a large amount of unlabeled data for learning the cluster parameters and then use a smaller labeled data to learn the second stage of classification or regression. Unsupervised learning is called "learning what normally happens" (Barrow 1989). When followed by a supervised learner, we first learn what normally happens and then learn what that means. We discuss such methods in chapter 12.

MIXTURE OF MIXTURES

In the case of classification, when each class is a mixture model composed of a number of components, the whole density is a *mixture of mixtures*:

$$p(\mathbf{x}|C_i) = \sum_{j=1}^{k_i} p(\mathbf{x}|\mathcal{G}_{ij})P(\mathcal{G}_{ij})$$

$$p(\mathbf{x}) = \sum_{i=1}^{K} p(\mathbf{x}|C_i)P(C_i)$$

where $k_i$ is the number of components making up $p(\mathbf{x}|C_i)$ and $\mathcal{G}_{ij}$ is the component $j$ of class $i$. Note that different classes may need different number of components. Learning the parameters of components is done separately for each class (probably after some regularization) as we discussed previously. This is better than fitting many components to data from all classes and then labeling them later with classes.

## 7.7 Spectral Clustering

Instead of clustering in the original space, a possibility is to first map the data to a new space with reduced dimensionality such that similarities are made more apparent and then cluster in there. Any feature selection or extraction method can be used for this purpose, and one such method is the Laplacian eigenmaps of section 6.12, where the aim is to place the data instances in such a way that given pairwise similarities are preserved.

After such a mapping, points that are similar are placed nearby, and this is expected to enhance the performance of clustering—for example, by using $k$-means. This is the idea behind *spectral clustering* (von Luxburg 2007). There are hence two steps:

SPECTRAL CLUSTERING

1. In the original space, we define a local neighborhood (by either fixing the number of neighbors or a distance threshold), and then for instances that are in the same neighborhood, we define a similarity measure—for example, using the Gaussian kernel—that is inversely proportional to the distance between them. Remember that instances not in the same local neighborhood are assigned a similarity of 0 and hence can be placed anywhere with respect to each other. Given this Laplacian, instances are positioned in the new space using feature embedding.

2. We run $k$-means clustering with the new data coordinates in this new space.

We remember from section 6.12 that when $\mathbf{B}$ is the matrix of pairwise similarities and $\mathbf{D}$ is the diagonal degree matrix with $d_i = \sum_j B_{ij}$ on the diagonals, the graph Laplacian is defined as

$$\mathbf{L} = \mathbf{D} - \mathbf{B}$$

This is the unnormalized Laplacian. There are two ways to normalize. One is closely related to the random walk (Shi and Malik 2000) and the other constructs a symmetric matrix (Ng, Jordan, and Weiss 2002). They may lead to better performance in clustering:

$$\begin{aligned} \mathbf{L}_{rw} &= \mathbf{I} - \mathbf{D}^{-1}\mathbf{B} \\ \mathbf{L}_{sym} &= \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{B}\mathbf{D}^{-1/2} \end{aligned}$$

It is always a good idea to do dimensionality reduction before clustering using Euclidean distance if there are redundant or correlated features. Using Laplacian eigenmaps makes more sense than multidimensional scaling proper or principal components analysis because those two check for the preservation of pairwise similarities between *all* pairs of instances whereas here with Laplacian eigenmaps, we care about preserving the similarity between neighboring instances only and in a manner that is inversely proportional to the distance between them. This has the effect that instances that are nearby in the original space, probably within the same cluster, will be placed very close in the new space, thus making the work of $k$-means easier, whereas those that are some distance away, probably belonging to different clusters, will be placed far apart. The graph should always be connected; that is, the local neighborhood should be large enough to connect clusters. Remember that the number of eigenvectors with eigenvalue 0 is the number of components and that it should be 1.

Note that though similarities are local, they propagate. Consider three instances, $a$, $b$, and $c$. Let us say $a$ and $b$ lie in the same neighborhood and so do $b$ and $c$, but not $a$ and $c$. Still, because $a$ and $b$ will be placed nearby and $b$ and $c$ will be placed nearby, $a$ will lie close to $c$ too, and they will probably be assigned to the same cluster. Consider now $a$ and $d$ that are not in the neighborhood with too many intermediate nodes between them; these two will not be placed nearby and it is very unlikely that they will be assigned to the same cluster.

Depending on which graph Laplacian is used and depending on the neighborhood size or the spread of the Gaussian, different results can be obtained, so one should always try for different parameters (von Luxburg 2009).

## 7.8 Hierarchical Clustering

We discussed clustering from a probabilistic point of view as fitting a mixture model to the data, or in terms of finding code words minimizing reconstruction error. There are also methods for clustering that use only similarities of instances, without any other requirement on the data; the aim is to find groups such that instances in a group are more similar to each other than instances in different groups. This is the approach taken by *hierarchical clustering*.

HIERARCHICAL CLUSTERING

This needs the use of a similarity, or equivalently a distance, measure defined between instances. Generally Euclidean distance is used, where we have to make sure that all attributes have the same scale. This is a special case of the *Minkowksi distance* with $p = 2$:

$$d_m(\mathbf{x}^r, \mathbf{x}^s) = \left[\sum_{j=1}^{d} (x_j^r - x_j^s)^p\right]^{1/p}$$

*City-block distance* is easier to calculate:

$$d_{cb}(\mathbf{x}^r, \mathbf{x}^s) = \sum_{j=1}^{d} |x_j^r - x_j^s|$$

AGGLOMERATIVE CLUSTERING

DIVISIVE CLUSTERING

An *agglomerative clustering* algorithm starts with $N$ groups, each initially containing one training instance, merging similar groups to form larger groups, until there is a single one. A *divisive clustering* algorithm goes in the other direction, starting with a single group and dividing large groups into smaller groups, until each group contains a single instance.

SINGLE-LINK CLUSTERING

At each iteration of an agglomerative algorithm, we choose the two closest groups to merge. In *single-link clustering*, this distance is defined as the smallest distance between all possible pair of elements of the two groups:

$$d(\mathcal{G}_i, \mathcal{G}_j) = \min_{\mathbf{x}^r \in \mathcal{G}_i, \mathbf{x}^s \in \mathcal{G}_j} d(\mathbf{x}^r, \mathbf{x}^s) \tag{7.19}$$

Consider a weighted, completely connected graph with nodes corresponding to instances and edges between nodes with weights equal to the distances between the instances. Then the single-link method corresponds to constructing the minimal spanning tree of this graph.

COMPLETE-LINK CLUSTERING

In *complete-link clustering*, the distance between two groups is taken as the largest distance between all possible pairs:

$$d(\mathcal{G}_i, \mathcal{G}_j) = \max_{\mathbf{x}^r \in \mathcal{G}_i, \mathbf{x}^s \in \mathcal{G}_j} d(\mathbf{x}^r, \mathbf{x}^s) \tag{7.20}$$

These are the two most frequently used measures to choose the two closest groups to merge. Other possibilities are the average-link method that uses the average of distances between all pairs and the centroid distance that measures the distance between the centroids (means) of the two groups.

DENDROGRAM

Once an agglomerative method is run, the result is generally drawn as a hierarchical structure known as the *dendrogram*. This is a tree where

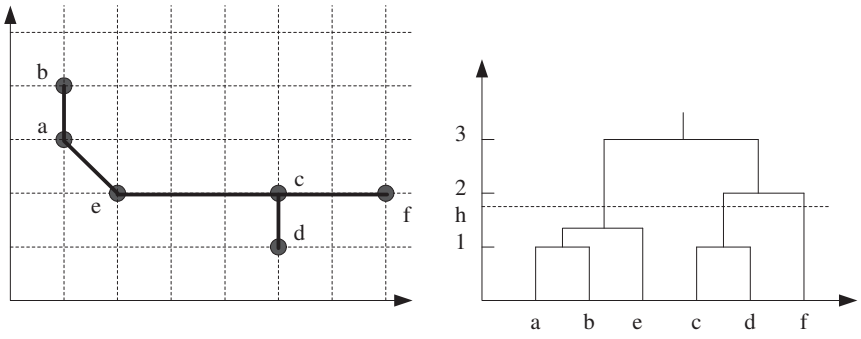


**Figure 7.5** A two-dimensional dataset and the dendrogram showing the result of single-link clustering is shown. Note that leaves of the tree are ordered so that no branches cross. The tree is then intersected at a desired value of $h$ to get the clusters.

leaves correspond to instances, which are grouped in the order in which they are merged. An example is given in figure 7.5. The tree can then be intersected at any level to get the wanted number of groups.

Single-link and complete-link methods calculate the distance between groups differently that affect the clusters and the dendrogram. In the single-link method, two instances are grouped together at level $h$ if the distance between them is less than $h$, or if there is an intermediate sequence of instances between them such that the distance between consecutive instances is less than $h$. On the other hand, in the complete-link method, all instances in a group have a distance less than $h$ between them. Single-link clusters may be elongated due to this "chaining" effect. (In figure 7.5, what if there were an instance halfway between $e$ and $c$?) Complete-link clusters tend to be more compact.

## 7.9 Choosing the Number of Clusters

Like any learning method, clustering also has its knob to adjust complexity; it is $k$, the number of clusters. Given any $k$, clustering will always find $k$ centers, whether they really are meaningful groups, or whether they are imposed by the method we use. There are various ways we can use to fine-tune $k$:

- In some applications such as color quantization, $k$ is defined by the application.
- Plotting the data in two dimensions using PCA may be used in uncovering the structure of data and the number of clusters in the data.
- An incremental approach may also help: Setting a maximum allowed distance is equivalent to setting a maximum allowed reconstruction error per instance.
- In some applications, validation of the groups can be done manually by checking whether clusters actually code meaningful groups of the data. For example, in a data mining application, application experts may do this check. In color quantization, we may inspect the image visually to check its quality (despite the fact that our eyes and brain do not analyze an image pixel by pixel).

Depending on what type of clustering method we use, we can plot the reconstruction error or log likelihood as a function of $k$ and look for the "elbow." After a large enough $k$, the algorithm will start dividing groups, in which case there will not be a large decrease in the reconstruction error or large increase in the log likelihood. Similarly, in hierarchical clustering, by looking at the differences between levels in the tree, we can decide on a good split.

## 7.10 Notes

Mixture models are frequently used in statistics. Dedicated textbooks are those by Titterington, Smith, and Makov (1985) and McLachlan and Basford (1988). McLachlan and Krishnan (1997) discuss recent developments in the EM algorithm, how its convergence can be accelerated, and various variants. In signal processing, $k$-means is called the *Linde-Buzo-Gray* (LBG) algorithm (Gersho and Gray 1992). It is used frequently in both statistics and signal processing in a large variety of applications and has many variants, one of which is *fuzzy k-means*. The *fuzzy membership* of an input to a component is also a number between 0 and 1 (Bezdek and Pal 1995). Alpaydın (1998) compares $k$-means, fuzzy $k$-means, and EM on Gaussian mixtures. A comparison of EM and other learning algorithms for the learning of Gaussian mixture models is given by Xu and Jordan (1996). On small data samples, an alternative to simplifying assumptions

FUZZY $k$-MEANS

is to use a Bayesian approach (Ormoneit and Tresp 1996). Moerland (1999) compares mixtures of Gaussians and mixtures of latent variable models on a set of classification problems, showing the advantage of latent variable models empirically. A book on clustering methods is by Jain and Dubes (1988) and survey articles are by Jain, Murty, and Flynn (1999) and Xu and Wunsch (2005).

One of the advantages of spectral clustering and hierarchical clustering is that we do not need a vectorial representation of the instances, as long as we can define a similarity/distance measure between pairs of instances. The problem of representing an arbitrary data structure—documents, graphs, web pages, and so on—as a vector such that Euclidean distance is meaningful is always a tedious task and leads to artificial representations, such as the bag of words. Being able to use (dis)similarity measures directly defined on the original structure is always a good idea, and we will have the same advantage with kernel functions when we talk about kernel machines in chapter 13.

## 7.11 Exercises

1. In image compression, $k$-means can be used as follows: The image is divided into nonoverlapping $c \times c$ windows and these $c^2$-dimensional vectors make up the sample. For a given $k$, which is generally a power of two, we do $k$-means clustering. The reference vectors and the indices for each window is sent over the communication line. At the receiving end, the image is then reconstructed by reading from the table of reference vectors using the indices. Write the computer program that does this for different values of $k$ and $c$. For each case, calculate the reconstruction error and the compression rate.

2. We can do $k$-means clustering, partition the instances, and then calculate $\mathbf{S}_i$ separately in each group. Why is this not a good idea?

   SOLUTION: There are basically two reasons: First, $k$-means does hard partitioning but it is always better to do a soft partitioning (using $h_i^t \in (0,1)$ instead of $b_i^t \in \{0,1\}$) so that instances (in between two clusters) can contribute to the parameters (the covariance matrix in this case) of more than one cluster allowing a smooth transition between clusters.

   Second, $k$-means proper uses the Euclidean distance and we remember that Euclidean distance implies features that have the same scale and are independent. Using $\mathbf{S}_i$ implies the use of Mahalanobis distance and hence taking care of differences in scale and dependencies.

3. Derive the M-step equations for $\mathbf{S}$ in the case of shared arbitrary covariance

matrix $\mathbf{S}$ (equation 7.15) and $s^2$ in the case of shared diagonal covariance matrix (equation 7.16).

4. Define a multivariate Bernoulli mixture where inputs are binary and derive the EM equations.

   SOLUTION: When the components are multivariate Bernouilli, we have binary vectors that are $d$-dimensional. Assuming that the dimensions are independent, we have (see section 5.7)

$$p_i(\boldsymbol{x}^t|\Phi) = \prod_{j=1}^{d} p_{ij}^{x_j^t}(1-p_{ij})^{1-x_j^t}$$

   where $\Phi^l = \{p_{i1}^l, p_{i2}^l, \ldots, p_{id}^l\}_{i=1}^k$. The E-step does not change (equation 7.9). In the M-step, for the component parameters $p_{ij}, i = 1, \ldots, k, j = 1, \ldots, d$, we maximize

$$\begin{aligned}\mathcal{Q}' &= \sum_t \sum_i h_i^t \log p_i(\boldsymbol{x}^t|\phi^l) \\ &= \sum_t \sum_i h_i^t \sum_j x_j^t \log p_{ij}^l + (1-x_j^t)\log(1-p_{ij}^l)\end{aligned}$$

   Taking the derivative with respect to $p_{ij}$ and setting it equal to 0, we get

$$p_{ij}^{l+1} = \frac{\sum_t h_i^t x_j^t}{\sum_t h_j^t}$$

   Note that this is the same as in equation 5.31, except that estimated "soft" labels $h_i^t$ replace the supervised labels $r_i^t$.

5. In the mixture of mixtures approach for classification, how can we fine-tune $k_i$, the number of components for class $C_i$?

EDIT DISTANCE

6. *Edit distance* between two strings—for example, gene sequences—is the number of character operations (insertions, deletions, substitutions) it takes to convert one string into another. List the advantages of doing spectral clustering using the edit distance as opposed to vanilla $k$-means using Euclidean distance on strings.

7. How can we do hierarchical clustering with binary input vectors—for example, for text clustering using the bag of words representation?

8. What are the similarities and differences between average-link clustering and $k$-means?

   SOLUTION: They both measure similarity by looking at the average of instances that fall in a cluster. Note, however, that in a hierarchical scheme, there are clusters at different resolutions.

9. In hierarchical clustering, how can we have locally adaptive distances? What are the advantages and disadvantages of this?

10. How can we make $k$-means robust to outliers?

    SOLUTION: An outlier is an instance that is very far from *all* centers. We would not want outliers to affect the solution. One possibility is to not take such instances into account when calculating the parameters—for example, means and covariances. Note that to detect an outlier, we can use the Mahalanobis distance or the likelihood, but we cannot use the posterior. We discuss a nonparametric method for detecting outliers in section 8.7.

11. Having generated a dendrogram, can we "prune" it?

## 7.12 References

Alpaydın, E. 1998. "Soft Vector Quantization and the EM Algorithm." *Neural Networks* 11:467–477.

Barrow, H. B. 1989. "Unsupervised Learning." *Neural Computation* 1:295–311.

Bezdek, J. C., and N. R. Pal. 1995. "Two Soft Relatives of Learning Vector Quantization." *Neural Networks* 8:729–743.

Bishop, C. M. 1999. "Latent Variable Models." In *Learning in Graphical Models*, ed. M. I. Jordan, 371–403. Cambridge, MA: MIT Press.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. "Maximum Likelihood from Incomplete Data via the EM Algorithm." *Journal of Royal Statistical Society* B 39:1–38.

Gersho, A., and R. M. Gray. 1992. *Vector Quantization and Signal Compression*. Boston: Kluwer.

Ghahramani, Z., and G. E. Hinton. 1997. *The EM Algorithm for Mixtures of Factor Analyzers*. Technical Report CRG TR-96-1, Department of Computer Science, University of Toronto (revised Feb. 1997).

Jain, A. K., and R. C. Dubes. 1988. *Algorithms for Clustering Data*. New York: Prentice Hall.

Jain, A. K., M. N. Murty, and P. J. Flynn. 1999. "Data Clustering: A Review." *ACM Computing Surveys* 31:264–323.

McLachlan, G. J., and K. E. Basford. 1988. *Mixture Models: Inference and Applications to Clustering*. New York: Marcel Dekker.

McLachlan, G. J., and T. Krishnan. 1997. *The EM Algorithm and Extensions*. New York: Wiley.

Moerland, P. 1999. "A Comparison of Mixture Models for Density Estimation." In *International Conference on Artificial Neural Networks*, ed. D. Willshaw and A. Murray, 25–30. London, UK: IEE Press.

Ng, A., M. I. Jordan, and Y. Weiss. 2002. "On Spectral Clustering: Analysis and an Algorithm." In *Advances in Neural Information Processing Systems 14*, ed. T. Dietterich, S. Becker, and Z. Ghahramani, 849–856. Cambridge, MA: MIT Press.

Ormoneit, D., and V. Tresp. 1996. "Improved Gaussian Mixture Density Estimates using Bayesian Penalty Terms and Network Averaging." In *Advances in Neural Information Processing Systems 8*, ed. D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, 542–548. Cambridge, MA: MIT Press.

Redner, R. A., and H. F. Walker. 1984. "Mixture Densities, Maximum Likelihood and the EM Algorithm." *SIAM Review* 26:195–239.

Rubin, D. B., and D. T. Thayer. 1982. "EM Algorithms for ML Factor Analysis." *Psychometrika* 47:69–76.

Shi, J., and J. Malik. 2000. "Normalized Cuts and Image Segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22:888–905.

Tipping, M. E., and C. M. Bishop. 1999. "Mixtures of Probabilistic Principal Component Analyzers." *Neural Computation* 11:443–482.

Titterington, D. M., A. F. M. Smith, and E. E. Makov. 1985. *Statistical Analysis of Finite Mixture Distributions*. New York: Wiley.

von Luxburg, U. 2007. "A Tutorial on Spectral Clustering." *Statistical Computing* 17:395–416.

Xu, L., and M. I. Jordan. 1996. "On Convergence Properties of the EM Algorithm for Gaussian Mixtures." *Neural Computation* 8:129–151.

Xu, R., and D. Wunsch II. 2005. "Survey of Clustering Algorithms." *IEEE Transactions on Neural Networks* 16:645–678.

# 8 *Nonparametric Methods*

*In the previous chapters, we discussed the parametric and semiparametric approaches where we assumed that the data is drawn from one or a mixture of probability distributions of known form. Now, we discuss the nonparametric approach that is used when no such assumption can be made about the input density and the data speaks for itself. We consider the nonparametric approaches for density estimation, classification, outlier detection, and regression and see how the time and space complexity can be checked.*

## 8.1 Introduction

IN PARAMETRIC methods, whether for density estimation, classification, or regression, we assume a model valid over the whole input space. In regression, for example, when we assume a linear model, we assume that for any input, the output is the same linear function of the input. In classification when we assume a normal density, we assume that all examples of the class are drawn from this same density. The advantage of a parametric method is that it reduces the problem of estimating a probability density function, discriminant, or regression function to estimating the values of a small number of parameters. Its disadvantage is that this assumption does not always hold and we may incur a large error if it does not. If we cannot make such assumptions and cannot come up with a parametric model, one possibility is to use a semiparametric mixture model as we saw in chapter 7 where the density is written as a disjunction of a small number of parametric models.

NONPARAMETRIC ESTIMATION

In *nonparametric estimation*, all we assume is that *similar inputs have similar outputs*. This is a reasonable assumption: The world is smooth,

and functions, whether they are densities, discriminants, or regression functions, change slowly. Similar instances mean similar things. We all love our neighbors because they are so much like us.

Therefore, our algorithm is composed of finding the similar past instances from the training set using a suitable distance measure and interpolating from them to find the right output. Different nonparametric methods differ in the way they define similarity or interpolate from the similar training instances. In a parametric model, all of the training instances affect the final global estimate, whereas in the nonparametric case, there is no single global model; local models are estimated as they are needed, affected only by the nearby training instances.

Nonparametric methods do not assume any a priori parametric form for the underlying densities; in a looser interpretation, a nonparametric model is not fixed but its complexity depends on the size of the training set, or rather, the complexity of the problem inherent in the data.

INSTANCE-BASED
MEMORY-BASED
LEARNING

In machine learning literature, nonparametric methods are also called *instance-based* or *memory-based learning* algorithms, since what they do is store the training instances in a lookup table and interpolate from these. This implies that all of the training instances should be stored and storing all requires memory of $\mathcal{O}(N)$. Furthermore, given an input, similar ones should be found, and finding them requires computation of $\mathcal{O}(N)$. Such methods are also called *lazy* learning algorithms, because unlike the *eager* parametric models, they do not compute a model when they are given the training set but postpone the computation of the model until they are given a test instance. In the case of a parametric approach, the model is quite simple and has a small number of parameters, of order $\mathcal{O}(d)$, or $\mathcal{O}(d^2)$, and once these parameters are calculated from the training set, we keep the model and no longer need the training set to calculate the output. $N$ is generally much larger than $d$ (or $d^2$), and this increased need for memory and computation is the disadvantage of the nonparametric methods.

We start by estimating a density function, and discuss its use in classification. We then generalize the approach to regression.

## 8.2 Nonparametric Density Estimation

As usual in density estimation, we assume that the sample $\mathcal{X} = \{x^t\}_{t=1}^N$ is drawn independently from some unknown probability density $p(\cdot)$. $\hat{p}(\cdot)$

is our estimator of $p(\cdot)$. We start with the univariate case where $x^t$ are scalars and later generalize to the multidimensional case.

The nonparametric estimator for the cumulative distribution function, $F(x)$, at point $x$ is the proportion of sample points that are less than or equal to $x$

$$\hat{F}(x) = \frac{\#\{x^t \leq x\}}{N} \tag{8.1}$$

where $\#\{x^t \leq x\}$ denotes the number of training instances whose $x^t$ is less than or equal to $x$. Similarly, the nonparametric estimate for the density function, which is the derivative of the cumulative distribution, can be calculated as

$$\hat{p}(x) = \frac{1}{h}\left[\frac{\#\{x^t \leq x+h\} - \#\{x^t \leq x\}}{N}\right] \tag{8.2}$$

$h$ is the length of the interval and instances $x^t$ that fall in this interval are assumed to be "close enough." The techniques given in this chapter are variants where different heuristics are used to determine the instances that are close and their effects on the estimate.

### 8.2.1 Histogram Estimator

HISTOGRAM

The oldest and most popular method is the *histogram* where the input space is divided into equal-sized intervals named *bins*. Given an origin $x_o$ and a bin width $h$, the bins are the intervals $[x_o + mh, x_o + (m+1)h)$ for positive and negative integers $m$ and the estimate is given as

$$\hat{p}(x) = \frac{\#\{x^t \text{ in the same bin as } x\}}{Nh} \tag{8.3}$$

In constructing the histogram, we have to choose both an origin and a bin width. The choice of origin affects the estimate near boundaries of bins, but it is mainly the bin width that has an effect on the estimate: With small bins, the estimate is spiky, and with larger bins, the estimate is smoother (see figure 8.1). The estimate is 0 if no instance falls in a bin and there are discontinuities at bin boundaries. Still, one advantage of the histogram is that once the bin estimates are calculated and stored, we do not need to retain the training set.

NAIVE ESTIMATOR

The *naive estimator* (Silverman 1986) frees us from setting an origin. It is defined as

$$\hat{p}(x) = \frac{\#\{x - h/2 < x^t \leq x + h/2\}}{Nh} \tag{8.4}$$

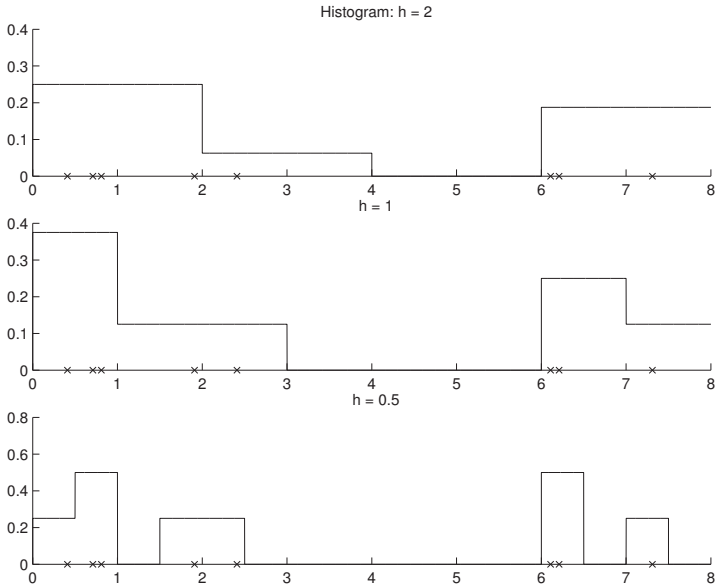


**Figure 8.1** Histograms for various bin lengths. '×' denote data points.

and is equal to the histogram estimate where $x$ is always at the center of a bin of size $h$ (see figure 8.2). The estimator can also be written as

$$\hat{p}(x) = \frac{1}{Nh} \sum_{t=1}^{N} w\left(\frac{x - x^t}{h}\right) \tag{8.5}$$

with the *weight function* defined as

$$w(u) = \begin{cases} 1 & \text{if } |u| < 1/2 \\ 0 & \text{otherwise} \end{cases}$$

This is as if each $x^t$ has a symmetric region of influence of size $h$ around it and contributes 1 for an $x$ falling in its region. Then the nonparametric estimate is just the sum of influences of $x^t$ whose regions include $x$. Because this region of influence is "hard" (0 or 1), the estimate is not a continuous function and has jumps at $x^t \pm h/2$.

### 8.2.2 Kernel Estimator

KERNEL FUNCTION To get a smooth estimate, we use a smooth weight function called a *kernel*

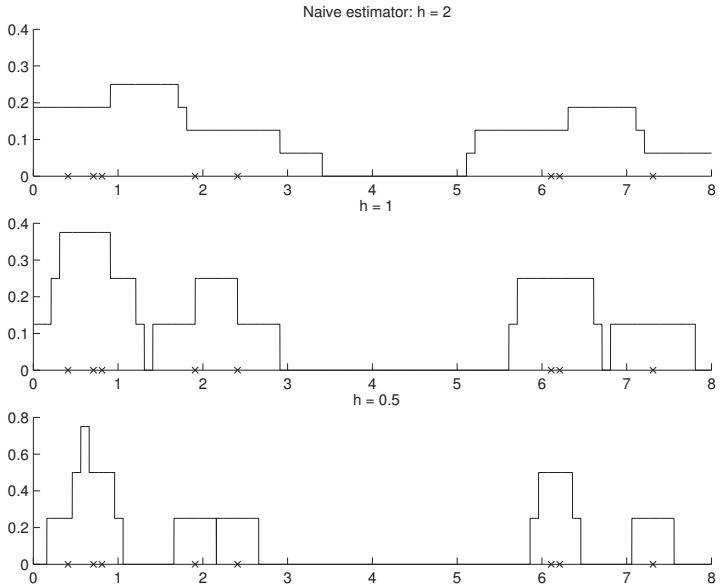


**Figure 8.2** Naive estimate for various bin lengths.

*function*. The most popular is the Gaussian kernel:

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{u^2}{2}\right] \tag{8.6}$$

KERNEL ESTIMATOR
PARZEN WINDOWS

The *kernel estimator*, also called *Parzen windows*, is defined as

$$\hat{p}(x) = \frac{1}{Nh} \sum_{t=1}^{N} K\left(\frac{x - x^t}{h}\right) \tag{8.7}$$

The kernel function $K(\cdot)$ determines the shape of the influences and the window width $h$ determines the width. Just like the naive estimate is the sum of "boxes," the kernel estimate is the sum of "bumps." All the $x^t$ have an effect on the estimate at $x$, and this effect decreases smoothly as $|x - x^t|$ increases.

To simplify calculation, $K(\cdot)$ can be taken to be 0 if $|x - x^t| > 3h$. There exist other kernels easier to compute that can be used, as long as $K(u)$ is maximum for $u = 0$ and decreasing symmetrically as $|u|$ increases.

When $h$ is small, each training instance has a large effect in a small region and no effect on distant points. When $h$ is larger, there is more