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 θ_1 and θ_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

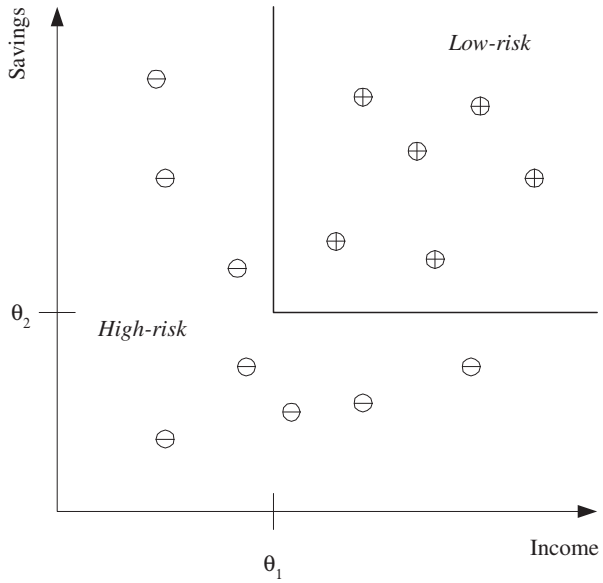

Figure 1.1 Example of a training dataset where each circle corresponds to one data instance with input values in the corresponding axes and its sign indicates the class. For simplicity, only two customer attributes, income and savings, are taken as input and the two classes are low-risk ('+') and high-risk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

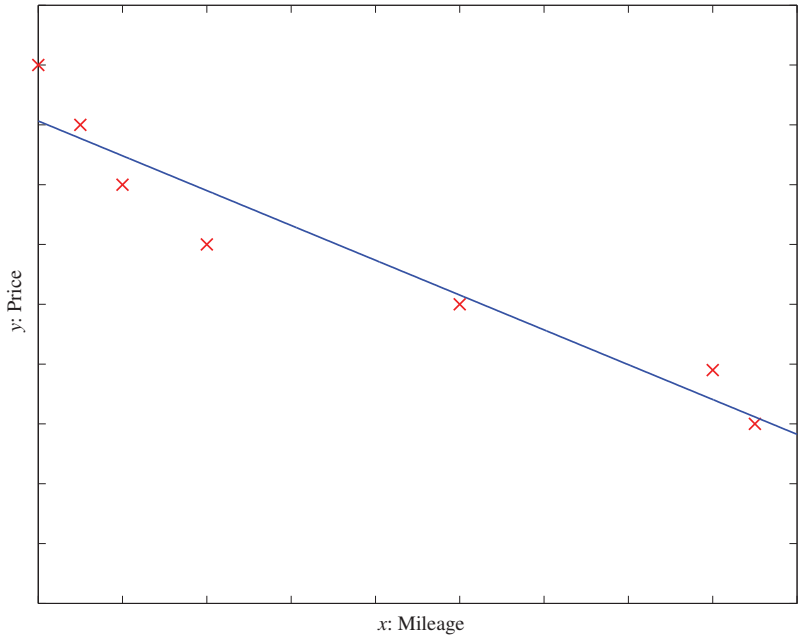

Figure 1.2 A training dataset of used cars and the function fitted. For simplicity, mileage is taken as the only input attribute and a linear model is used.

program optimizes the parameters, θ, 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: 16×16 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 100×100 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

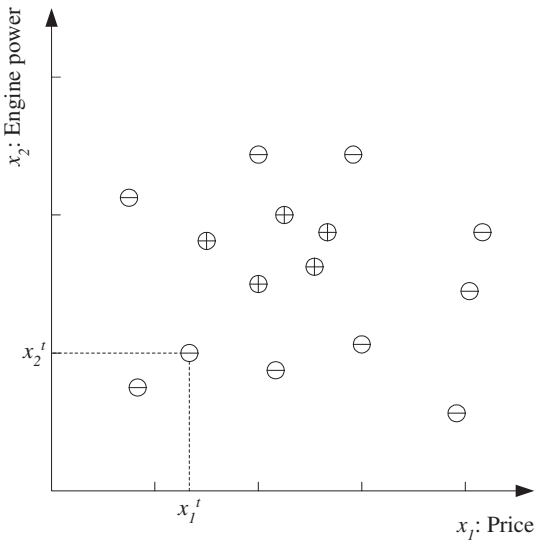

Figure 2.1 Training set for the class of a "family car." Each data point corresponds to one example car, and the coordinates of the point indicate the price and engine power of that car. '+' denotes a positive example of the class (a family car), and '−' denotes a negative example (not a family car);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

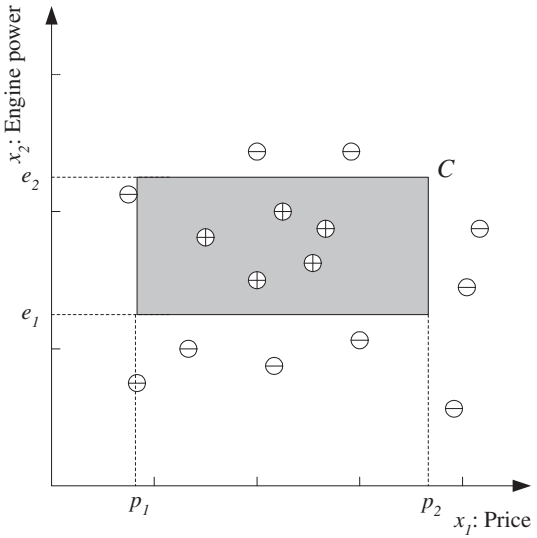

Figure 2.2 Example of a hypothesis class. The class of family car is a rectangle in the price-engine power space.

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

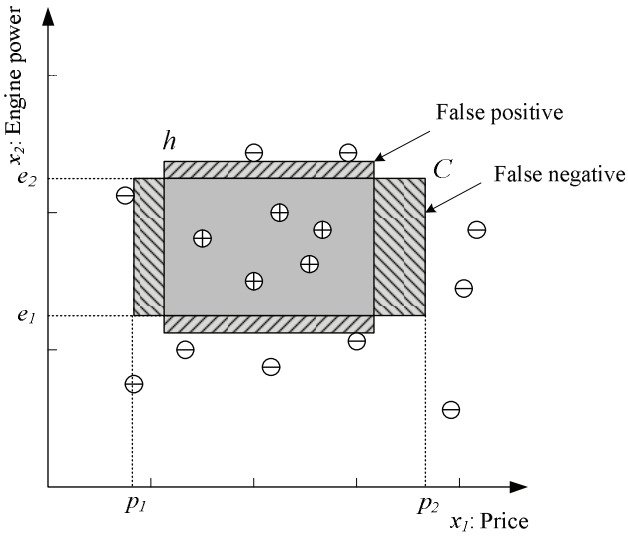

Figure 2.3 C is the actual class and h is our induced hypothesis. The point where C is 1 but h is 0 is a false negative, and the point where C is 0 but h is 1 is a false positive. Other points—namely, true positives and true negatives—are correctly classified.

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

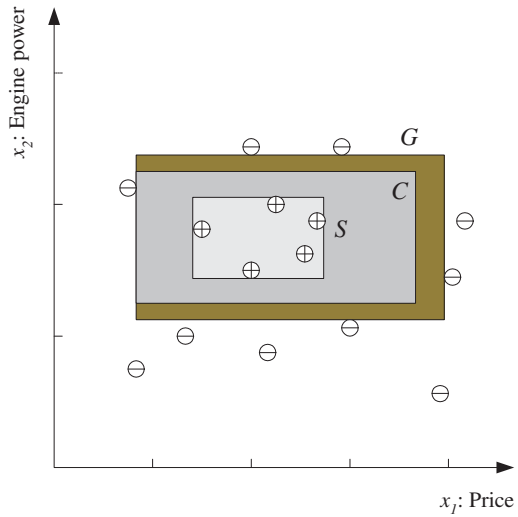

Figure 2.4 S is the most specific and G is the most general hypothesis.

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.

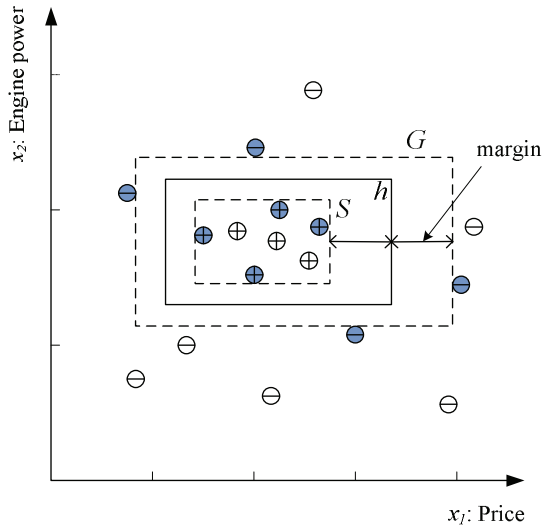

Figure 2.5 We choose the hypothesis with the largest margin, for best separation. The shaded instances are those that define (or support) the margin; other instances can be removed without affecting h.

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-

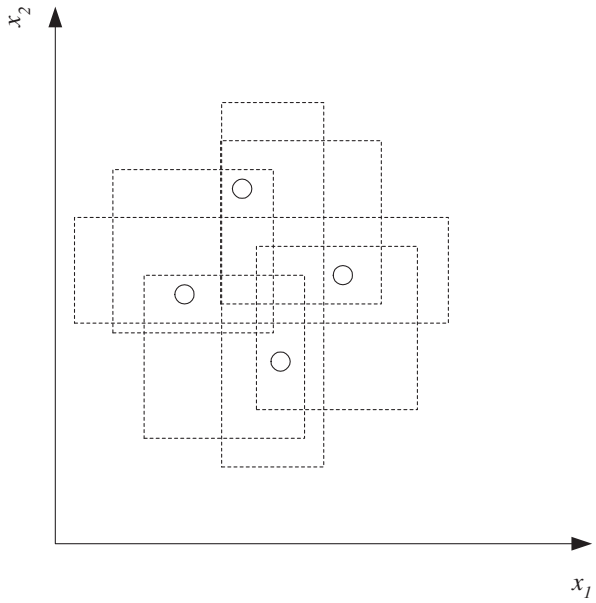

Figure 2.6 An axis-aligned rectangle can shatter four points. Only rectangles covering two points are shown.

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 ϵ, 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 ϵ. 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 δ. We have the inequality

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

So if we choose N and δ 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

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

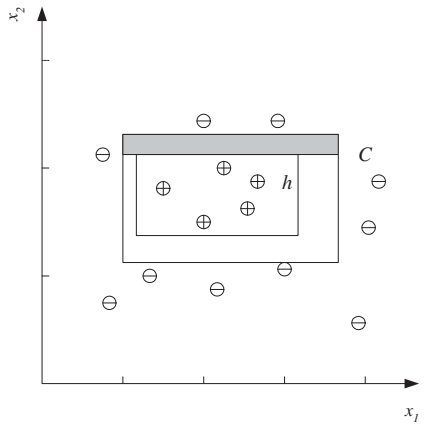

Figure 2.7 The difference between h and C is the sum of four rectangular strips, one of which is shaded.

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 ϵ. We can have arbitrary large confidence by decreasing δ and arbitrary small error by decreasing ϵ, 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

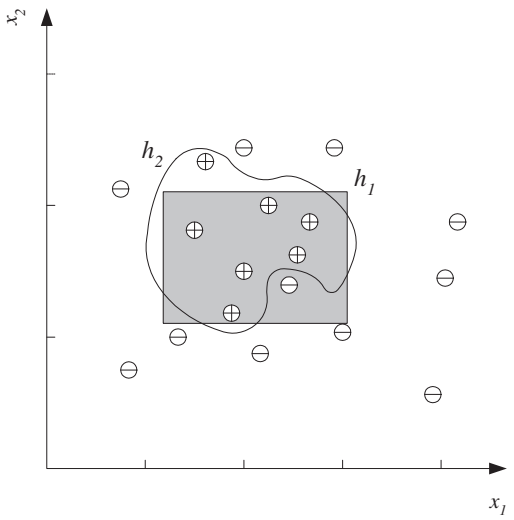

Figure 2.8 When there is noise, there is not a simple boundary between the positive and negative instances, and zero misclassification error may not be possible with a simple hypothesis. A rectangle is a simple hypothesis with four parameters defining the corners. An arbitrary closed form can be drawn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

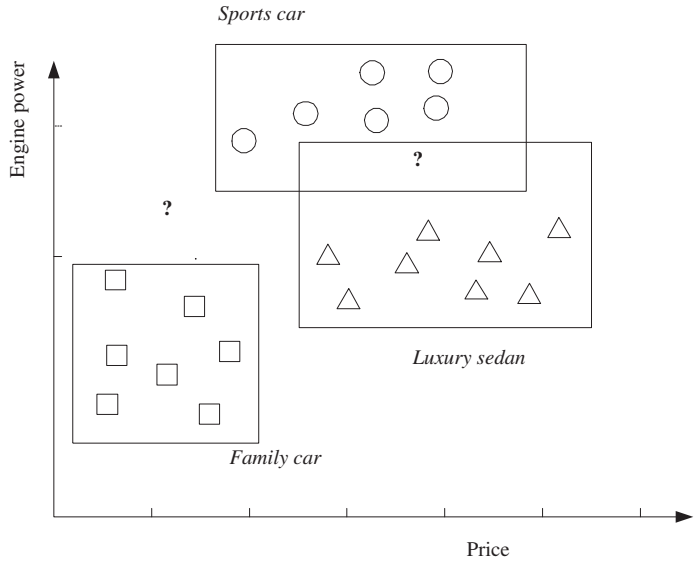

Figure 2.9 There are three classes: family car, sports car, and luxury sedan. There are three hypotheses induced, each one covering the instances of one class and leaving outside the instances of the other two classes. '?' are reject regions where no, or more than one, class is chosen.

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 ϵ 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

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

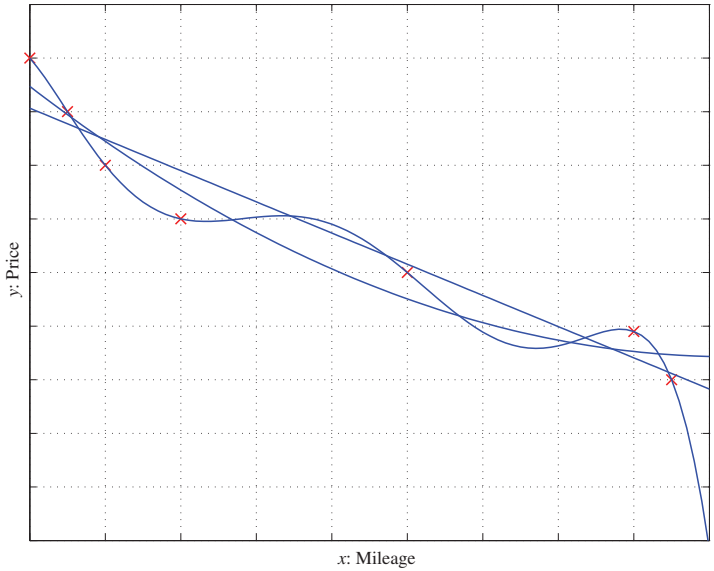

Figure 2.10 Linear, second-order, and sixth-order polynomials are fitted to the same set of points. The highest order gives a perfect fit, but given this much data it is very unlikely that the real curve is so shaped. The second order seems better than the linear fit in capturing the trend in the training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

   $g(\cdot)$ defines the hypothesis class $\mathcal{H}$, and a particular value of θ 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 θ; 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, θ. 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 θ^* 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.

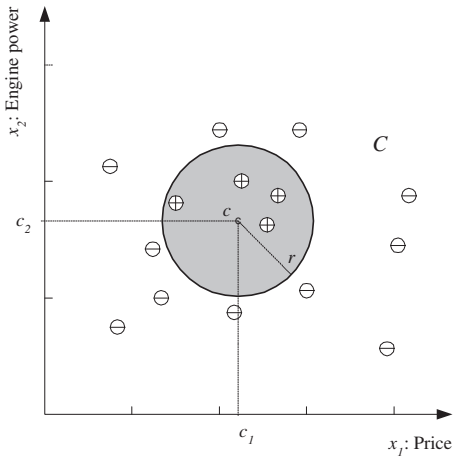

Figure 2.11 Hypothesis class is a circle with two parameters, the coordinates of its center and its radius.

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.

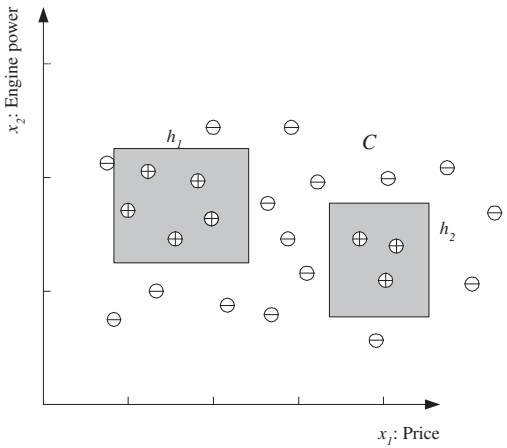

Figure 2.12 Hypothesis class is a union of two rectangles.

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.

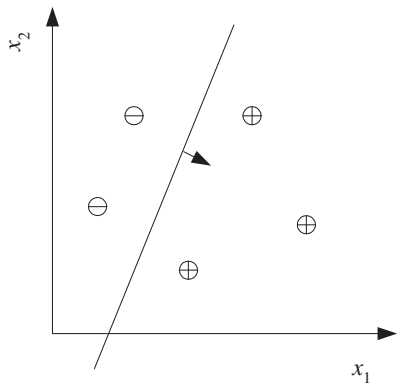

Figure 2.13 A line separating positive and negative instances.

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 α_i as the decision to assign the input to class C_i and λ_{ik} as the *loss* incurred for taking action α_i when the input actually belongs to C_k. Then the *expected risk* for taking action α_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 α_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 α_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*, α_{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).

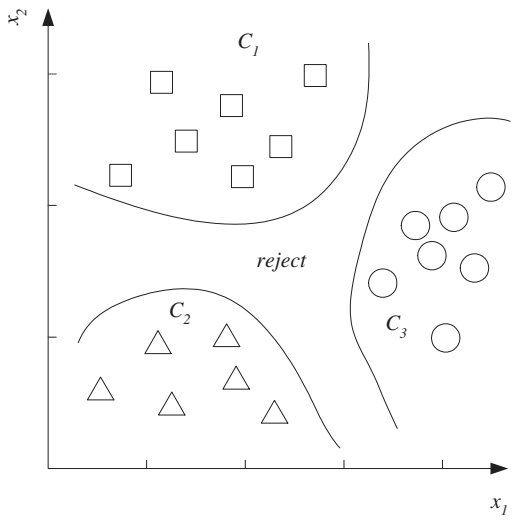

Figure 3.1 Example of decision regions and decision boundaries.

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
α_1: Choose C_1	0	10
α_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 α_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
α_1: Choose C_1	0	10
α_2: Choose C_2	5	0
α_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 α_1 if

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

We choose α_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 λ 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.

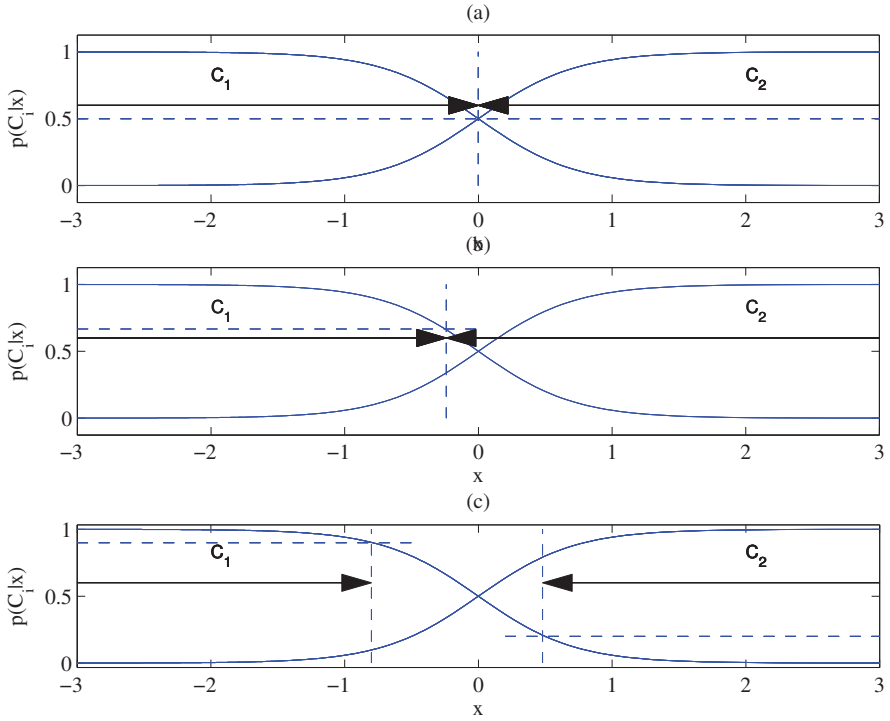

Figure 3.2 The boundary changes as the misclassification losses change. (a) The boundary is where the two posteriors are equal when both misclassifications are equally costly. (b) When the losses are not symmetric, the boundary shifts toward the class that incurs higher risk when misclassified. (c) When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

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

LIKELIHOOD

We want to find θ that makes sampling x^t from $p(x|\theta)$ as likely as possible. Because x^t are independent, the *likelihood* of parameter θ 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 θ that makes $\mathcal{X}$ the most likely to be drawn. We thus search for θ 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 θ, and let $d = d(\mathcal{X})$ be an estimator of θ. To evaluate the quality of this estimator, we can measure how much it is different from θ, 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 θ values, then we say that d is an *unbiased estimator* of θ. For example, with x^t drawn from some density with mean μ, the sample average, m, is an unbiased estimator of the mean, μ, 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 μ, if we take many such samples, $\mathcal{X}_i$, and estimate many $m_i = m(\mathcal{X}_i)$, *their* average will get close to μ 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 μ. Let us now check, s^2, the MLE of σ^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 σ^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 θ (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, θ, 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

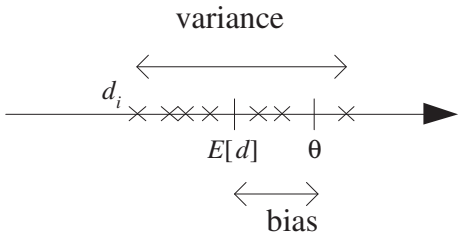

Figure 4.1 θ is the parameter to be estimated. d_i are several estimates (denoted by '$\times$') over different samples $\mathcal{X}_i$. Bias is the difference between the expected value of d and θ. Variance is how much d_i are scattered around the expected value. We would like both to be small.

need the sample), and we model this uncertainty by viewing θ as a random variable and by defining a prior density for it, $p(\theta)$. For example, let us say we are told that θ is approximately normal and with 90 percent confidence, θ 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 θ 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 θ, which tells us the likely θ 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 θ, the sufficient statistics, we know everything about the distribution. Thus we are taking an average over predictions using all values of θ, 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 θ, 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 θ 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 θ, 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 μ. 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 μ_0, σ_0^2, and σ^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 μ_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 σ_0^2 is small, that is, when we have little prior uncertainty regarding the correct value of θ, or when N is small, our prior guess μ_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 μ_i is the mean income of such customers and σ_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)

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

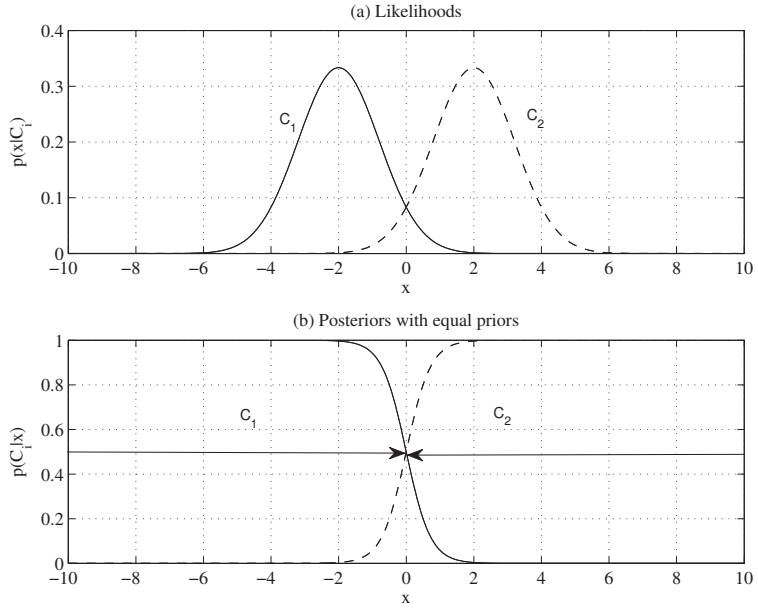

Figure 4.2 (a) Likelihood functions and (b) posteriors with equal priors for two classes when the input is one-dimensional. Variances are equal and the posteriors intersect at one point, which is the threshold of decision.

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).

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

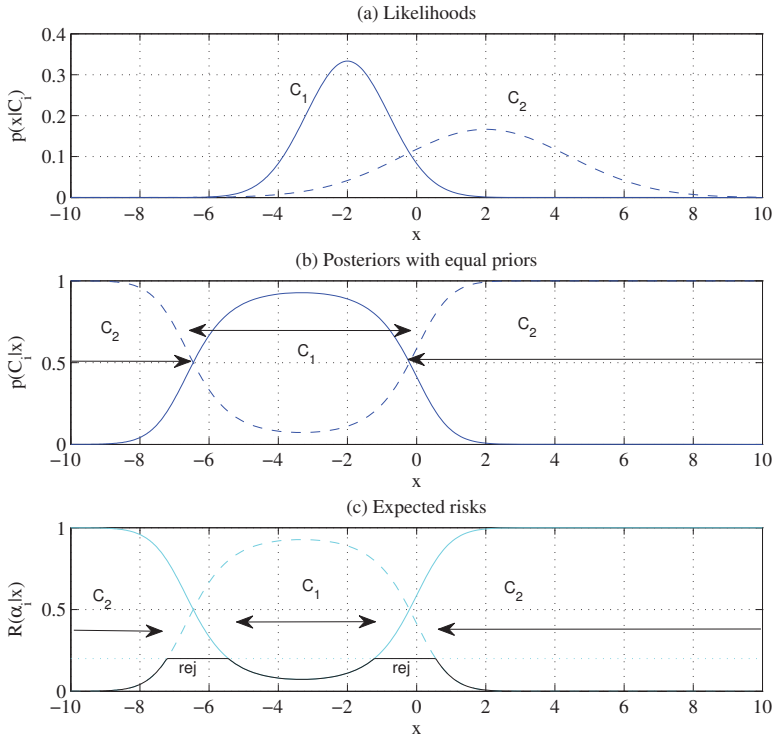

Figure 4.3 (a) Likelihood functions and (b) posteriors with equal priors for two classes when the input is one-dimensional. Variances are unequal and the posteriors intersect at two points. In (c), the expected risks are shown for the two classes and for reject with $\lambda = 0.2$ (section 3.3).

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

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 μ_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 θ. If we assume that ϵ is zero mean Gaussian with constant variance σ^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 θ. 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

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

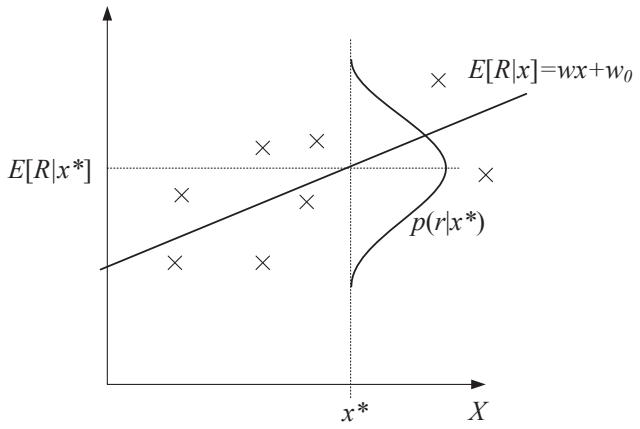

Figure 4.4 Regression assumes 0 mean Gaussian noise added to the model; here, the model is linear.

$p(r|x)$ 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 θ 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 θ 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, σ^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.

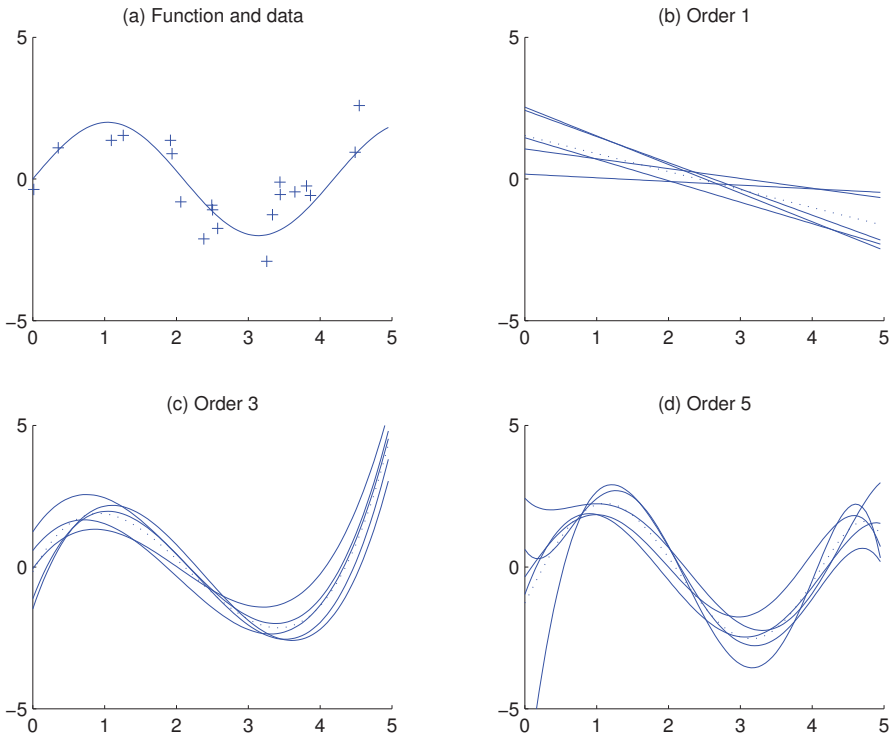

Figure 4.5 (a) Function, $f(x) = 2\sin(1.5x)$, and one noisy ($\mathcal{N}(0,1)$) dataset sampled from the function. Five samples are taken, each containing twenty instances. (b), (c), (d) are five polynomial fits, namely, $g_i(\cdot)$, of order 1, 3, and 5. For each case, dotted line is the average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

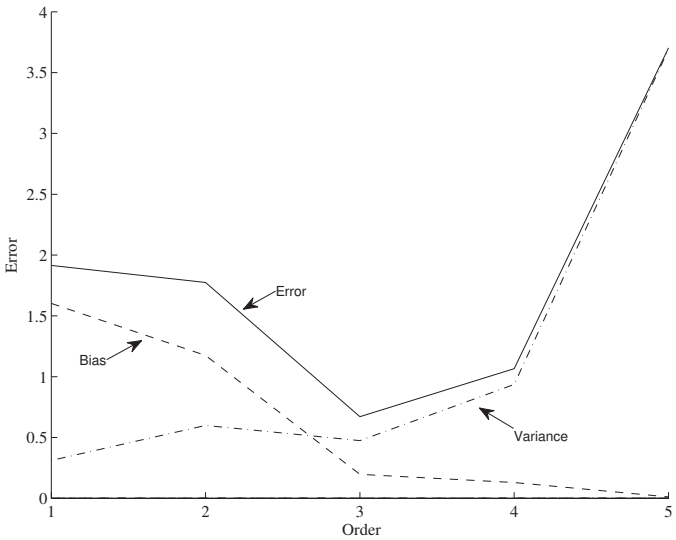

Figure 4.6 In the same setting as that of figure 4.5, using one hundred models instead of five, bias, variance, and error for polynomials of order 1 to 5. Order 1 has the smallest variance. Order 5 has the smallest bias. As the order is increased, bias decreases but variance increases. Order 3 has the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

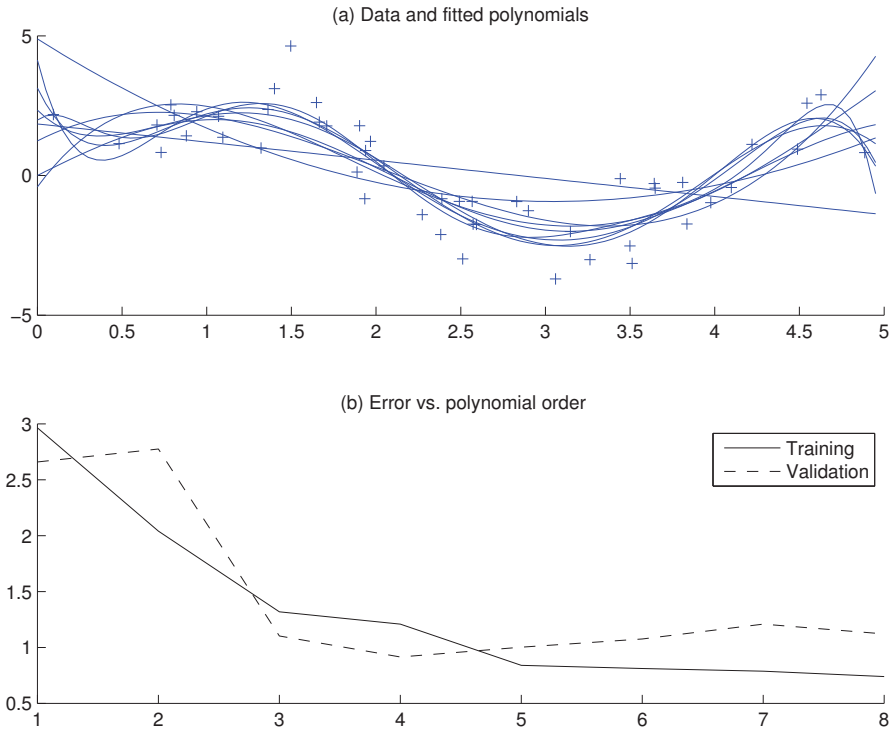

Figure 4.7 In the same setting as that of figure 4.5, training and validation sets (each containing 50 instances) are generated. (a) Training data and fitted polynomials of order from 1 to 8. (b) Training and validation errors as a function of the polynomial order. The "elbow" is at 3.

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 λ 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 λ is taken too large, only very simple models are allowed and we risk introducing bias. λ 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 σ^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 λ_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 λ, 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

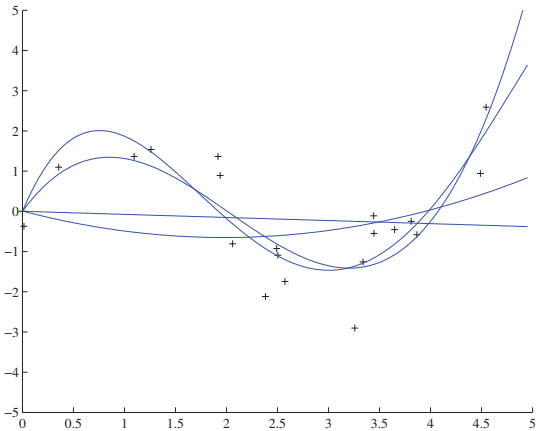

Figure 4.8 In the same setting as that of figure 4.5, polynomials of order 1 to 4 are fitted. The magnitude of coefficients increase as the order of the polynomial increases. They are as follows: $1 : [-0.0769, 0.0016]^T$, $2 : [0.1682, -0.6657, 0.0080]^T$, $3 : [0.4238, -2.5778, 3.4675, -0.0002]^T$,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

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 μ and σ, and the code that calculates m and s from the sample. Do the same using the Bayes' estimator assuming a prior distribution for μ.
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 λ on bias and variance?

    SOLUTION: λ 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.