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

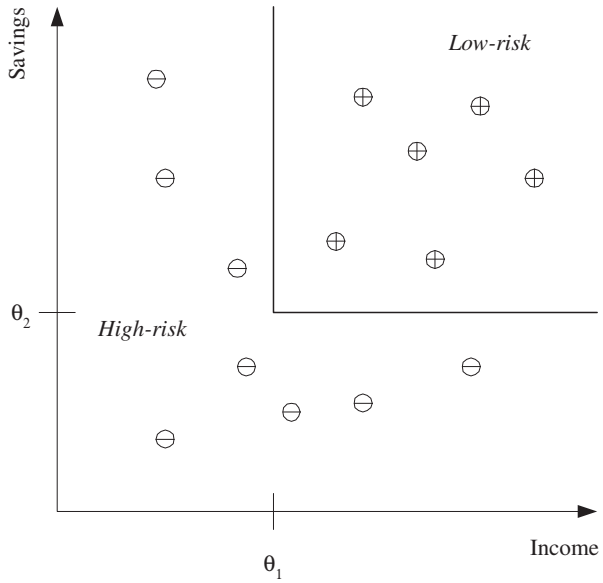

Figure 1.1 Example of a training dataset where each circle corresponds to one data instance with input values in the corresponding axes and its sign indicates the class. For simplicity, only two customer attributes, income and savings, are taken as input and the two classes are low-risk ('+') and high-risk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

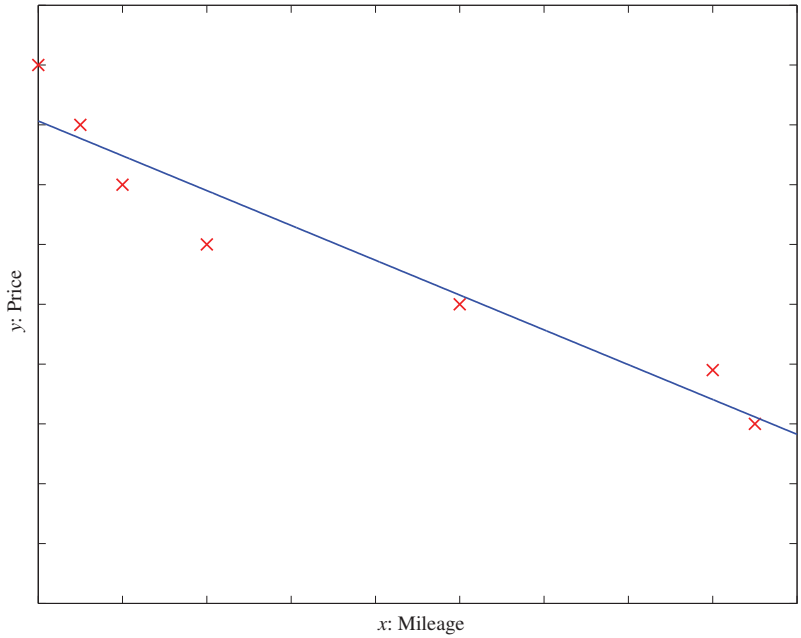

Figure 1.2 A training dataset of used cars and the function fitted. For simplicity, mileage is taken as the only input attribute and a linear model is used.

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.

8 *Nonparametric Methods*

In the previous chapters, we discussed the parametric and semiparametric approaches where we assumed that the data is drawn from one or a mixture of probability distributions of known form. Now, we discuss the nonparametric approach that is used when no such assumption can be made about the input density and the data speaks for itself. We consider the nonparametric approaches for density estimation, classification, outlier detection, and regression and see how the time and space complexity can be checked.

8.1 Introduction

IN PARAMETRIC methods, whether for density estimation, classification, or regression, we assume a model valid over the whole input space. In regression, for example, when we assume a linear model, we assume that for any input, the output is the same linear function of the input. In classification when we assume a normal density, we assume that all examples of the class are drawn from this same density. The advantage of a parametric method is that it reduces the problem of estimating a probability density function, discriminant, or regression function to estimating the values of a small number of parameters. Its disadvantage is that this assumption does not always hold and we may incur a large error if it does not. If we cannot make such assumptions and cannot come up with a parametric model, one possibility is to use a semiparametric mixture model as we saw in chapter 7 where the density is written as a disjunction of a small number of parametric models.

NONPARAMETRIC ESTIMATION

In *nonparametric estimation*, all we assume is that *similar inputs have similar outputs*. This is a reasonable assumption: The world is smooth,

and functions, whether they are densities, discriminants, or regression functions, change slowly. Similar instances mean similar things. We all love our neighbors because they are so much like us.

Therefore, our algorithm is composed of finding the similar past instances from the training set using a suitable distance measure and interpolating from them to find the right output. Different nonparametric methods differ in the way they define similarity or interpolate from the similar training instances. In a parametric model, all of the training instances affect the final global estimate, whereas in the nonparametric case, there is no single global model; local models are estimated as they are needed, affected only by the nearby training instances.

Nonparametric methods do not assume any a priori parametric form for the underlying densities; in a looser interpretation, a nonparametric model is not fixed but its complexity depends on the size of the training set, or rather, the complexity of the problem inherent in the data.

INSTANCE-BASED
MEMORY-BASED
LEARNING

In machine learning literature, nonparametric methods are also called *instance-based* or *memory-based learning* algorithms, since what they do is store the training instances in a lookup table and interpolate from these. This implies that all of the training instances should be stored and storing all requires memory of $\mathcal{O}(N)$. Furthermore, given an input, similar ones should be found, and finding them requires computation of $\mathcal{O}(N)$. Such methods are also called *lazy* learning algorithms, because unlike the *eager* parametric models, they do not compute a model when they are given the training set but postpone the computation of the model until they are given a test instance. In the case of a parametric approach, the model is quite simple and has a small number of parameters, of order $\mathcal{O}(d)$, or $\mathcal{O}(d^2)$, and once these parameters are calculated from the training set, we keep the model and no longer need the training set to calculate the output. N is generally much larger than d (or d^2), and this increased need for memory and computation is the disadvantage of the nonparametric methods.

We start by estimating a density function, and discuss its use in classification. We then generalize the approach to regression.

8.2 Nonparametric Density Estimation

As usual in density estimation, we assume that the sample $\mathcal{X} = \{x^t\}_{t=1}^N$ is drawn independently from some unknown probability density $p(\cdot)$. $\hat{p}(\cdot)$

is our estimator of $p(\cdot)$. We start with the univariate case where x^t are scalars and later generalize to the multidimensional case.

The nonparametric estimator for the cumulative distribution function, $F(x)$, at point x is the proportion of sample points that are less than or equal to x

$$\hat{F}(x) = \frac{\#\{x^t \leq x\}}{N} \tag{8.1}$$

where $\#\{x^t \leq x\}$ denotes the number of training instances whose x^t is less than or equal to x. Similarly, the nonparametric estimate for the density function, which is the derivative of the cumulative distribution, can be calculated as

$$\hat{p}(x) = \frac{1}{h}\left[\frac{\#\{x^t \leq x+h\} - \#\{x^t \leq x\}}{N}\right] \tag{8.2}$$

h is the length of the interval and instances x^t that fall in this interval are assumed to be "close enough." The techniques given in this chapter are variants where different heuristics are used to determine the instances that are close and their effects on the estimate.

8.2.1 Histogram Estimator

HISTOGRAM

The oldest and most popular method is the *histogram* where the input space is divided into equal-sized intervals named *bins*. Given an origin x_o and a bin width h, the bins are the intervals $[x_o + mh, x_o + (m+1)h)$ for positive and negative integers m and the estimate is given as

$$\hat{p}(x) = \frac{\#\{x^t \text{ in the same bin as } x\}}{Nh} \tag{8.3}$$

In constructing the histogram, we have to choose both an origin and a bin width. The choice of origin affects the estimate near boundaries of bins, but it is mainly the bin width that has an effect on the estimate: With small bins, the estimate is spiky, and with larger bins, the estimate is smoother (see figure 8.1). The estimate is 0 if no instance falls in a bin and there are discontinuities at bin boundaries. Still, one advantage of the histogram is that once the bin estimates are calculated and stored, we do not need to retain the training set.

NAIVE ESTIMATOR

The *naive estimator* (Silverman 1986) frees us from setting an origin. It is defined as

$$\hat{p}(x) = \frac{\#\{x - h/2 < x^t \leq x + h/2\}}{Nh} \tag{8.4}$$

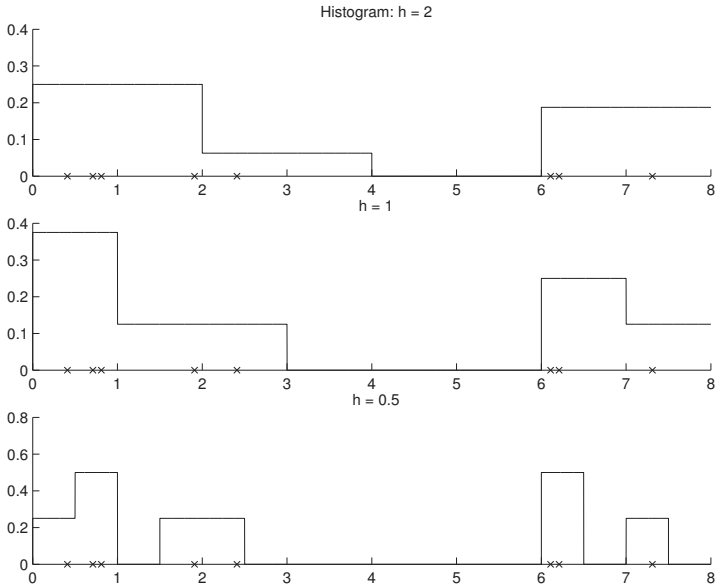

Figure 8.1 Histograms for various bin lengths. '×' denote data points.

and is equal to the histogram estimate where x is always at the center of a bin of size h (see figure 8.2). The estimator can also be written as

$$\hat{p}(x) = \frac{1}{Nh} \sum_{t=1}^{N} w\left(\frac{x - x^t}{h}\right) \tag{8.5}$$

with the *weight function* defined as

$$w(u) = \begin{cases} 1 & \text{if } |u| < 1/2 \\ 0 & \text{otherwise} \end{cases}$$

This is as if each x^t has a symmetric region of influence of size h around it and contributes 1 for an x falling in its region. Then the nonparametric estimate is just the sum of influences of x^t whose regions include x. Because this region of influence is "hard" (0 or 1), the estimate is not a continuous function and has jumps at $x^t \pm h/2$.

8.2.2 Kernel Estimator

KERNEL FUNCTION To get a smooth estimate, we use a smooth weight function called a *kernel*

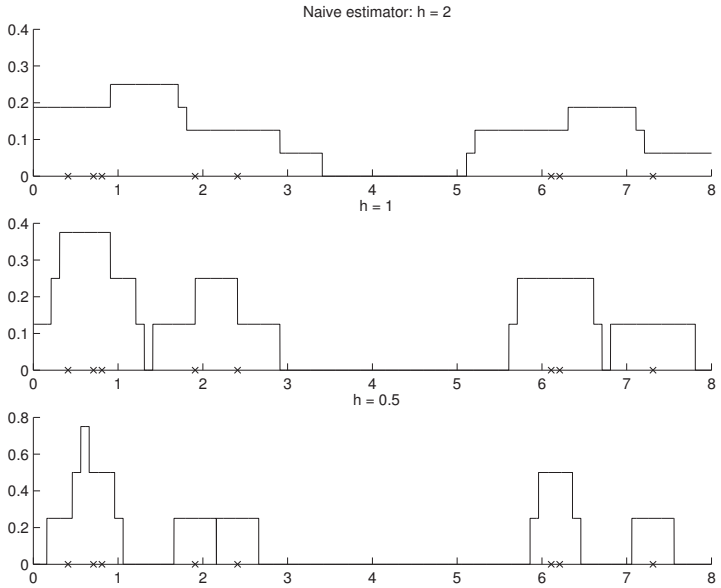

Figure 8.2 Naive estimate for various bin lengths.

function. The most popular is the Gaussian kernel:

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{u^2}{2}\right] \tag{8.6}$$

KERNEL ESTIMATOR
PARZEN WINDOWS

The *kernel estimator*, also called *Parzen windows*, is defined as

$$\hat{p}(x) = \frac{1}{Nh} \sum_{t=1}^{N} K\left(\frac{x - x^t}{h}\right) \tag{8.7}$$

The kernel function $K(\cdot)$ determines the shape of the influences and the window width h determines the width. Just like the naive estimate is the sum of "boxes," the kernel estimate is the sum of "bumps." All the x^t have an effect on the estimate at x, and this effect decreases smoothly as $|x - x^t|$ increases.

To simplify calculation, $K(\cdot)$ can be taken to be 0 if $|x - x^t| > 3h$. There exist other kernels easier to compute that can be used, as long as $K(u)$ is maximum for $u = 0$ and decreasing symmetrically as $|u|$ increases.

When h is small, each training instance has a large effect in a small region and no effect on distant points. When h is larger, there is more

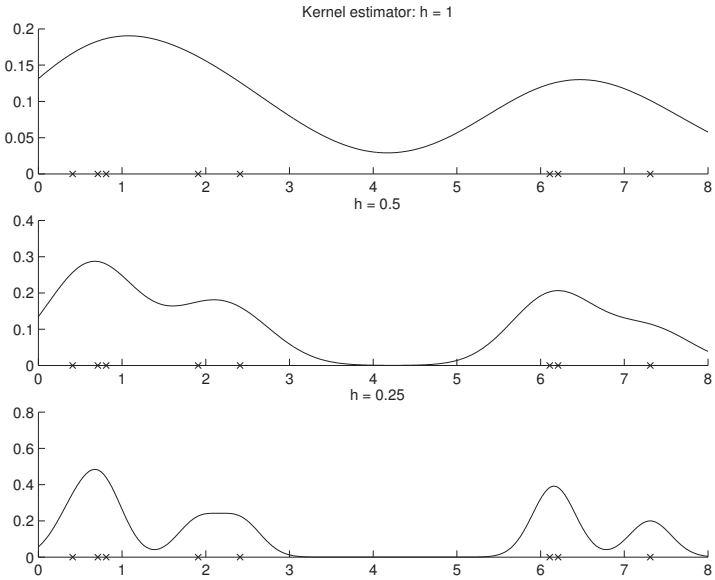

Figure 8.3 Kernel estimate for various bin lengths.

overlap of the kernels and we get a smoother estimate (see figure 8.3). If $K(\cdot)$ is everywhere nonnegative and integrates to 1, namely, if it is a legitimate density function, so will $\hat{p}(\cdot)$ be. Furthermore, $\hat{p}(\cdot)$ will inherit all the continuity and differentiability properties of the kernel $K(\cdot)$, so that, for example, if $K(\cdot)$ is Gaussian, then $\hat{p}(\cdot)$ will be smooth having all the derivatives.

One problem is that the window width is fixed across the entire input space. Various adaptive methods have been proposed to tailor h as a function of the density around x.

8.2.3 *k*-Nearest Neighbor Estimator

The nearest neighbor class of estimators adapts the amount of smoothing to the *local* density of data. The degree of smoothing is controlled by k, the number of neighbors taken into account, which is much smaller than N, the sample size. Let us define a distance between a and b, for example, $|a - b|$, and for each x, we define

$$d_1(x) \leq d_2(x) \leq \cdots \leq d_N(x)$$

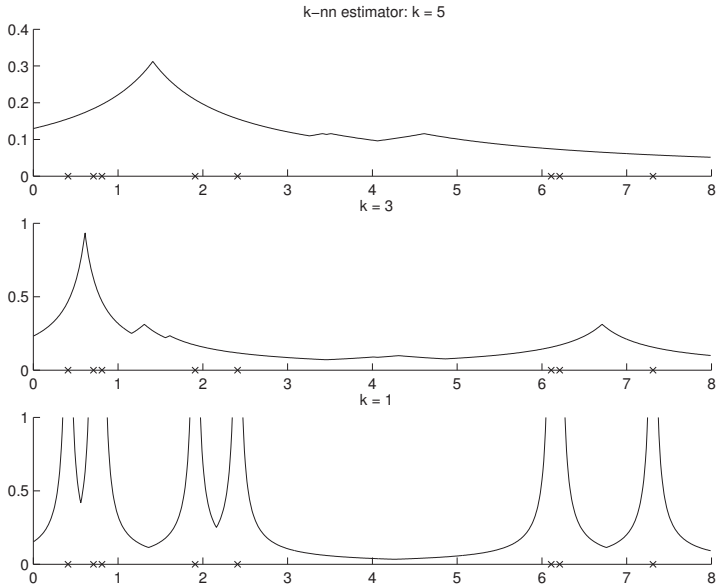

Figure 8.4 k-nearest neighbor estimate for various k values.

to be the distances arranged in ascending order, from x to the points in the sample: $d_1(x)$ is the distance to the nearest sample, $d_2(x)$ is the distance to the next nearest, and so on. If x^t are the data points, then we define $d_1(x) = \min_t |x - x^t|$, and if i is the index of the closest sample, namely, $i = \arg\min_t |x - x^t|$, then $d_2(x) = \min_{j \neq i} |x - x^j|$, and so forth.

k-NEAREST NEIGHBOR ESTIMATE

The *k-nearest neighbor* (k-nn) density estimate is

$$\hat{p}(x) = \frac{k}{2Nd_k(x)} \tag{8.8}$$

This is like a naive estimator with $h = 2d_k(x)$, the difference being that instead of fixing h and checking how many samples fall in the bin, we fix k, the number of observations to fall in the bin, and compute the bin size. Where density is high, bins are small, and where density is low, bins are larger (see figure 8.4).

The k-nn estimator is not continuous; its derivative has a discontinuity at all $\frac{1}{2}(x^{(j)} + x^{(j+k)})$ where $x^{(j)}$ are the order statistics of the sample. The k-nn is not a probability density function since it integrates to ∞, not 1.

To get a smoother estimate, we can use a kernel function whose effect decreases with increasing distance

$$\hat{p}(x) = \frac{1}{N d_k(x)} \sum_{t=1}^{N} K\left(\frac{x - x^t}{d_k(x)}\right) \tag{8.9}$$

This is like a kernel estimator with adaptive smoothing parameter $h = d_k(x)$. $K(\cdot)$ is typically taken to be the Gaussian kernel.

8.3 Generalization to Multivariate Data

Given a sample of d-dimensional observations $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$, the multivariate kernel density estimator is

$$\hat{p}(\boldsymbol{x}) = \frac{1}{N h^d} \sum_{t=1}^{N} K\left(\frac{\boldsymbol{x} - \boldsymbol{x}^t}{h}\right) \tag{8.10}$$

with the requirement that

$$\int_{\Re^d} K(\boldsymbol{x}) d\boldsymbol{x} = 1$$

The obvious candidate is the multivariate Gaussian kernel:

$$K(\boldsymbol{u}) = \left(\frac{1}{\sqrt{2\pi}}\right)^d \exp\left[-\frac{\|\boldsymbol{u}\|^2}{2}\right] \tag{8.11}$$

CURSE OF DIMENSIONALITY

However, care should be applied to using nonparametric estimates in high-dimensional spaces because of the *curse of dimensionality*: Let us say $\boldsymbol{x}$ is eight-dimensional, and we use a histogram with ten bins per dimension, then there are 10^8 bins, and unless we have lots of data, most of these bins will be empty and the estimates in there will be 0. In high dimensions, the concept of "close" also becomes blurry so we should be careful in choosing h.

For example, the use of the Euclidean norm in equation 8.11 implies that the kernel is scaled equally on all dimensions. If the inputs are on different scales, they should be normalized to have the same variance. Still, this does not take correlations into account, and better results are achieved when the kernel has the same form as the underlying distribution

$$K(\boldsymbol{u}) = \frac{1}{(2\pi)^{d/2}|\mathbf{S}|^{1/2}} \exp\left[-\frac{1}{2}\boldsymbol{u}^T \mathbf{S}^{-1} \boldsymbol{u}\right] \tag{8.12}$$

where $\mathbf{S}$ is the sample covariance matrix. This corresponds to using Mahalanobis distance instead of the Euclidean distance.

8.4 Nonparametric Classification

When used for classification, we use the nonparametric approach to estimate the class-conditional densities, $p(\boldsymbol{x}|C_i)$. The kernel estimator of the class-conditional density is given as

$$\hat{p}(\boldsymbol{x}|C_i) = \frac{1}{N_i h^d} \sum_{t=1}^{N} K\left(\frac{\boldsymbol{x}-\boldsymbol{x}^t}{h}\right) r_i^t \tag{8.13}$$

where r_i^t is 1 if $\boldsymbol{x}^t \in C_i$ and 0 otherwise. N_i is the number of labeled instances belonging to C_i: $N_i = \sum_t r_i^t$. The MLE of the prior density is $\hat{P}(C_i) = N_i/N$. Then, the discriminant can be written as

$$\begin{aligned} g_i(\boldsymbol{x}) &= \hat{p}(\boldsymbol{x}|C_i)\hat{P}(C_i) \\ &= \frac{1}{Nh^d} \sum_{t=1}^{N} K\left(\frac{\boldsymbol{x}-\boldsymbol{x}^t}{h}\right) r_i^t \end{aligned} \tag{8.14}$$

and $\boldsymbol{x}$ is assigned to the class for which the discriminant takes its maximum. The common factor $1/(Nh^d)$ can be ignored. So each training instance votes for its class and has no effect on other classes; the weight of vote is given by the kernel function $K(\cdot)$, typically giving more weight to closer instances.

For the special case of k-nn estimator, we have

$$\hat{p}(\boldsymbol{x}|C_i) = \frac{k_i}{N_i V^k(\boldsymbol{x})} \tag{8.15}$$

where k_i is the number of neighbors out of the k nearest that belong to C_i and $V^k(\boldsymbol{x})$ is the volume of the d-dimensional hypersphere centered at $\boldsymbol{x}$, with radius $r = \|\boldsymbol{x} - \boldsymbol{x}_{(k)}\|$ where $\boldsymbol{x}_{(k)}$ is the k-th nearest observation to $\boldsymbol{x}$ (among all neighbors from all classes of $\boldsymbol{x}$): $V^k = r^d c_d$ with c_d as the volume of the unit sphere in d dimensions, for example, $c_1 = 2, c_2 = \pi, c_3 = 4\pi/3$, and so forth. Then

$$\hat{P}(C_i|\boldsymbol{x}) = \frac{\hat{p}(\boldsymbol{x}|C_i)\hat{P}(C_i)}{\hat{p}(\boldsymbol{x})} = \frac{k_i}{k} \tag{8.16}$$

k-NN CLASSIFIER

The *k-nn classifier* assigns the input to the class having most examples among the k neighbors of the input. All neighbors have equal vote, and the class having the maximum number of voters among the k neighbors is chosen. Ties are broken arbitrarily or a weighted vote is taken. k is generally taken to be an odd number to minimize ties: Confusion is

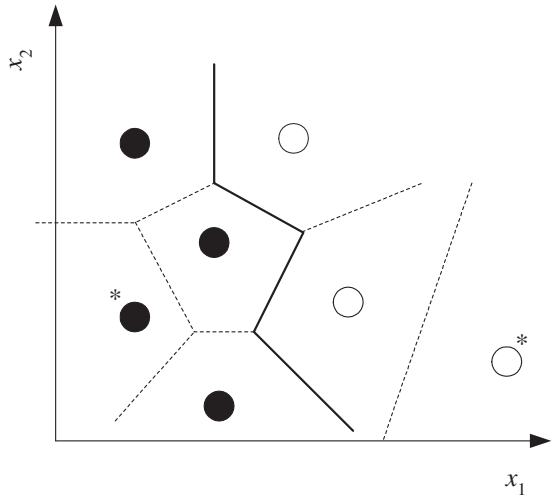

Figure 8.5 Dotted lines are the Voronoi tesselation and the straight line is the class discriminant. In condensed nearest neighbor, those instances that do not participate in defining the discriminant (marked by '*') can be removed without increasing the training error.

generally between two neighboring classes. A special case of k-nn is the *nearest neighbor classifier* where $k = 1$ and the input is assigned to the class of the nearest pattern. This divides the space in the form of a *Voronoi tesselation* (see figure 8.5).

NEAREST NEIGHBOR CLASSIFIER

VORONOI TESSELATION

8.5 Condensed Nearest Neighbor

Time and space complexity of nonparametric methods are proportional to the size of the training set, and *condensing* methods have been proposed to decrease the number of stored instances without degrading performance. The idea is to select the smallest subset $\mathcal{Z}$ of $\mathcal{X}$ such that when $\mathcal{Z}$ is used in place of $\mathcal{X}$, error does not increase (Dasarathy 1991).

CONDENSED NEAREST NEIGHBOR

The best-known and earliest method is *condensed nearest neighbor* where 1-nn is used as the nonparametric estimator for classification (Hart 1968). 1-nn approximates the discriminant in a piecewise linear manner, and only the instances that define the discriminant need be kept; an in-

$\mathcal{Z} \leftarrow \emptyset$
Repeat
  For all $\boldsymbol{x} \in \mathcal{X}$ (in random order)
    Find $\boldsymbol{x}' \in \mathcal{Z}$ such that $\|\boldsymbol{x} - \boldsymbol{x}'\| = \min_{\boldsymbol{x}^j \in \mathcal{Z}} \|\boldsymbol{x} - \boldsymbol{x}^j\|$
    If class($\boldsymbol{x}$)$\neq$class($\boldsymbol{x}'$) add $\boldsymbol{x}$ to $\mathcal{Z}$
Until $\mathcal{Z}$ does not change

Figure 8.6 Condensed nearest neighbor algorithm.

stance inside the class regions need not be stored as *its* nearest neighbor is of the same class and its absence does not cause any error (on the training set) (figure 8.5). Such a subset is called a consistent subset, and we would like to find the minimal consistent subset.

Hart proposed a greedy algorithm to find $\mathcal{Z}$ (figure 8.6). The algorithm starts with an empty $\mathcal{Z}$ and passing over the instances in $\mathcal{X}$ one by one in a random order, checks whether they can be classified correctly by 1-nn using the instances already stored in $\mathcal{Z}$. If an instance is misclassified, it is added to $\mathcal{Z}$; if it is correctly classified, $\mathcal{Z}$ is unchanged. We should pass over the training set a few times until no further instances are added. The algorithm does a local search and depending on the order in which the training instances are seen, different subsets may be found, which may have different accuracies on the validation data. Thus it does not guarantee finding the minimal consistent subset, which is known to be NP-complete (Wilfong 1992).

Condensed nearest neighbor is a greedy algorithm that aims to minimize training error and complexity, measured by the size of the stored subset. We can write an augmented error function

$$E'(\mathcal{Z}|\mathcal{X}) = E(\mathcal{X}|\mathcal{Z}) + \lambda|\mathcal{Z}| \tag{8.17}$$

where $E(\mathcal{X}|\mathcal{Z})$ is the error on $\mathcal{X}$ storing $\mathcal{Z}$. $|\mathcal{Z}|$ is the cardinality of $\mathcal{Z}$, and the second term penalizes complexity. As in any regularization scheme, λ represents the trade-off between the error and complexity such that for small λ, error becomes more important, and as λ gets larger, complex models are penalized more. Condensed nearest neighbor is one method to minimize equation 8.17, but other algorithms to optimize it can also be devised.

8.6 Distance-Based Classification

The k-nearest neighbor classifier assigns an instance to the class most heavily represented among its neighbors. It is based on the idea that the more similar the instances, the more likely it is that they belong to the same class. We can use the same approach for classification as long as we have a reasonable similarity or distance measure (Chen et al. 2009).

Most classification algorithms can be recast as a distance-based classifier. For example, in section 5.5, we saw the parametric approach with Gaussian classes, and there, we talked about the *nearest mean classifier* where we choose C_i if

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_i) = \min_{j=1}^{K} \mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_j) \tag{8.18}$$

In the case of hyperspheric Gaussians where dimensions are independent and all are in the same scale, the distance measure is the Euclidean:

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_i) = \|\boldsymbol{x} - \boldsymbol{m}_i\|$$

Otherwise it is the Mahalanobis distance:

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_i) = (\boldsymbol{x} - \boldsymbol{m}_i)^T \mathbf{S}_i^{-1} (\boldsymbol{x} - \boldsymbol{m}_i)$$

where $\mathbf{S}_i$ is the covariance matrix of C_i.

In the semiparametric approach where each class is written as a mixture of Gaussians, we can say roughly speaking that we choose C_i if among all cluster centers of all classes, one that belongs to C_i is the closest:

$$\min_{l=1}^{k_i} \mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_{il}) = \min_{j=1}^{K} \min_{l=1}^{k_j} \mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_{jl}) \tag{8.19}$$

where k_j is the number of clusters of C_j and $\boldsymbol{m}_{jl}$ denotes the center of cluster l of C_j. Again, the distance used is the Euclidean or Mahalanobis depending on the shape of the clusters.

The nonparametric case can be even more flexible: Instead of having a distance measure per class or per cluster, we can have a different one for each neighborhood, that is, for each small region in the input space. In other words, we can define *locally adaptive distance functions* that we can then use in classification, for example, with k-nn (Hastie and Tibshirani 1996; Domeniconi, Peng, and Gunopulos 2002; Ramanan and Baker 2011).

DISTANCE LEARNING

The idea of *distance learning* is to parameterize $\mathcal{D}(\boldsymbol{x}, \boldsymbol{x}^t|\theta)$, learn θ from a labeled sample in a supervised manner, and then use it with k-nn (Bellet, Habrard, and Sebban 2013). The most common approach is to use the Mahalanobis distance:

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{x}^t|\mathbf{M}) = (\boldsymbol{x} - \boldsymbol{x}^t)^T \mathbf{M} (\boldsymbol{x} - \boldsymbol{x}^t) \tag{8.20}$$

where the parameter is the positive definite matrix $\mathbf{M}$. An example is the *large margin nearest neighbor* algorithm (Weinberger and Saul 2009) where $\mathbf{M}$ is estimated so that for all instances in the training set, the distance to a neighbor with the same label is always less than the distance to a neighbor with a different label—we discuss this algorithm in detail in section 13.13.

LARGE MARGIN NEAREST NEIGHBOR

When the input dimensionality is high, to avoid overfitting, one approach is to add sparsity constraints on $\mathbf{M}$. The other approach is to use a low-rank approximation where we factor $\mathbf{M}$ as $\mathbf{L}^T\mathbf{L}$ and $\mathbf{L}$ is $k \times d$ with $k < d$. In this case:

$$\begin{aligned} \mathcal{D}(\boldsymbol{x}, \boldsymbol{x}^t|\mathbf{M}) &= (\boldsymbol{x} - \boldsymbol{x}^t)^T \mathbf{M} (\boldsymbol{x} - \boldsymbol{x}^t) = (\boldsymbol{x} - \boldsymbol{x}^t)^T \mathbf{L}^T \mathbf{L} (\boldsymbol{x} - \boldsymbol{x}^t) \\ &= (\mathbf{L}(\boldsymbol{x} - \boldsymbol{x}^t))^T (\mathbf{L}(\boldsymbol{x} - \boldsymbol{x}^t)) = (\mathbf{L}\boldsymbol{x} - \mathbf{L}\boldsymbol{x}^t)^T (\mathbf{L}\boldsymbol{x} - \mathbf{L}\boldsymbol{x}^t)) \\ &= (\boldsymbol{z} - \boldsymbol{z}^t)^T (\boldsymbol{z} - \boldsymbol{z}^t) = \|\boldsymbol{z} - \boldsymbol{z}^t\|^2 \end{aligned} \tag{8.21}$$

where $\boldsymbol{z} = \mathbf{L}\boldsymbol{x}$ is the k-dimensional projection of $\boldsymbol{x}$, and we learn $\mathbf{L}$ instead of $\mathbf{M}$. We see that the Mahalanobis distance in the original d-dimensional $\boldsymbol{x}$ space corresponds to the (squared) Euclidean distance in the new k-dimensional space. This implies the three-way relationship between distance estimation, dimensionality reduction, and feature extraction: The ideal distance measure is defined as the Euclidean distance in a new space whose (fewest) dimensions are extracted from the original inputs in the best possible way. This is demonstrated in figure 8.7.

HAMMING DISTANCE

With discrete data, *Hamming distance* that counts the number of nonmatching attributes can be used:

$$HD(\boldsymbol{x}, \boldsymbol{x}^t) = \sum_{j=1}^{d} 1(x_j \neq x_j^t) \tag{8.22}$$

where

$$1(a) = \begin{cases} 1 & \text{if } a \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

This framework can be used with application-dependent similarity or distance measures as well. We may have specialized similarity/distance

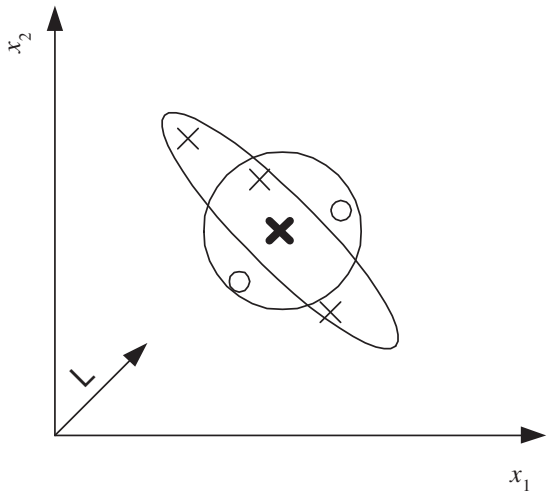

Figure 8.7 The use of Mahalanobis vs. Euclidean distance in k-nearest neighbor classification. There are two classes indicated by '$\circ$' and '$\times$'. The bold '$\times$' is the test instance and $k = 3$. Points that are of equal Euclidean distance define a circle that here leads to misclassification. We see that there is a certain correlation structure that can be captured by the Mahalanobis distance; it defines an ellipse and leads to correct classification. We also see that if the data is projected on the direction showed by L, we can do correct classification in that reduced one-dimensional space.

scores for matching image parts in vision, sequence alignment scores in bioinformatics, and document similarity measures in natural language processing; these can all be used without explicitly needing to represent those entities as vectors and using a general-purpose distance such as the Euclidean distance. In chapter 13, we will talk about kernel functions that have a similar role.

As long as we have a similarity score function between two instances $S(\boldsymbol{x}, \boldsymbol{x}^t)$, we can define a *similarity-based representation* $\boldsymbol{x}'$ of instance $\boldsymbol{x}$ as the N-dimensional vector of scores with all the training instances, $\boldsymbol{x}^t, t = 1, \ldots, N$:

$$\boldsymbol{x}' = [s(\boldsymbol{x}, \boldsymbol{x}^1), s(\boldsymbol{x}, \boldsymbol{x}^2), \ldots, s(\boldsymbol{x}, \boldsymbol{x}^N)]^T$$

This can then be used as a vector to be handled by any learner (Pekalska

and Duin 2002); in the context of kernel machines, we will call this the *empirical kernel map* (section 13.7).

8.7 Outlier Detection

An *outlier*, *novelty*, or *anomaly* is an instance that is very much different from other instances in the sample. An outlier may indicate an abnormal behavior of the system; for example, in a dataset of credit card transactions, it may indicate fraud; in an image, outliers may indicate anomalies, for example, tumors; in a dataset of network traffic, outliers may be intrusion attempts; in a health-care scenario, an outlier indicates a significant deviation from patient's normal behavior. Outliers may also be recording errors—for example, due to faulty sensors—that should be detected and discarded to get reliable statistics.

OUTLIER DETECTION

Outlier detection is not generally cast as a supervised, two-class classification problem of seperating typical instances and outliers, because generally there are very few instances that can be labeled as outliers and they do not fit a consistent pattern that can be easily captured by a two-class classifier. Instead, it is the typical instances that are modeled; this is sometimes called *one-class classification*. Once we model the typical instances, any instance that does not fit the model (and this may occur in many different ways) is an anomaly. Another problem that generally occurs is that the data used to train the outlier detector is unlabeled and may contain outliers mixed with typical instances.

ONE-CLASS CLASSIFICATION

Outlier detection basically implies spotting what does not normally happen; that is, it is density estimation followed by checking for instances with too small probability under the estimated density. As usual, the fitted model can be parametric, semiparametric, or nonparametric. In the parametric case (section 5.4), for example, we can fit a Gaussian to the whole data and any instance having a low probability, or equally, with high Mahalanobis distance to the mean, is a candidate for being an outlier. In the semiparametric case (section 7.2), we fit, for example, a mixture of Gaussians and check whether an instance has small probability; this would be an instance that is far from its nearest cluster center or one that forms a cluster by itself.

Still when the data that is used for fitting the model itself includes outliers, it makes more sense to use a nonparametric density estimator, because the more parametric a model is, the less robust it will be to the

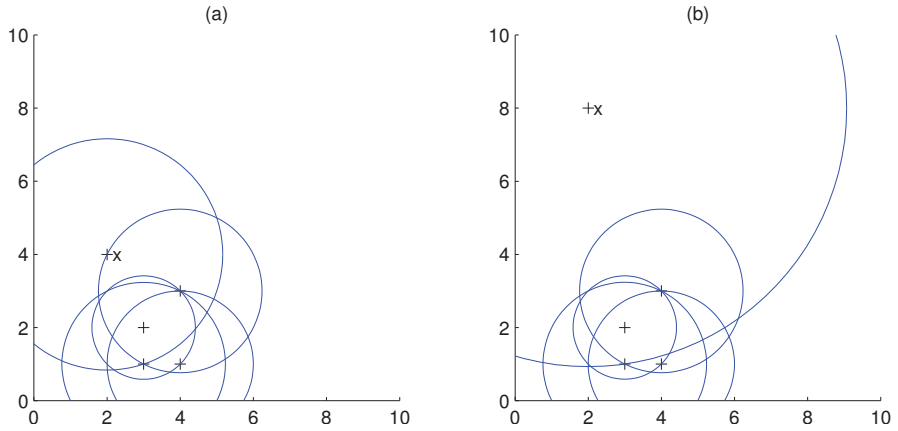

Figure 8.8 Training instances are shown by '+', '×' is the query, and the radius of the circle centered at an instance is equal to the distance to the third nearest neighbor. (a) LOF of '×' is close to 1 and it is not an outlier. (b) LOF of '×' is much larger than 1 and it is likely to be an outlier.

presence of outliers—for example, a single outlier may seriously corrupt the estimated mean and covariance of a Gaussian.

In nonparametric density estimation, as we discussed in the preceding sections, the estimated probability is high where there are many training instances nearby and the probability decreases as the neighborhood becomes more sparse. One example is the *local outlier factor* that compares the denseness of the neighborhood of an instance with the average denseness of the neighborhoods of its neighbors (Breunig et al. 2000). Let us define $d_k(\boldsymbol{x})$ as the distance between instance $\boldsymbol{x}$ and its k-th nearest neighbor. Let us define $\mathcal{N}(\boldsymbol{x})$ as the set of training instances that are in the neighborhood of $\boldsymbol{x}$, for example, its k nearest neighbors. Consider $d_k(\boldsymbol{s})$ for $\boldsymbol{s} \in \mathcal{N}(\boldsymbol{x})$. We compare $d_k(\boldsymbol{x})$ with the average of $d_k(\boldsymbol{s})$ for such $\boldsymbol{s}$:

LOCAL OUTLIER FACTOR

$$\text{LOF}(\boldsymbol{x}) = \frac{d_k(\boldsymbol{x})}{\sum_{\boldsymbol{s} \in \mathcal{N}(\boldsymbol{x})} d_k(\boldsymbol{s}) / |\mathcal{N}(\boldsymbol{x})|} \tag{8.23}$$

If $\text{LOF}(\boldsymbol{x})$ is close to 1, $\boldsymbol{x}$ is not an outlier; as it gets larger, the probability that it is an outlier increases (see figure 8.8).

8.8 Nonparametric Regression: Smoothing Models

In regression, given the training set $\mathcal{X} = \{x^t, r^t\}$ where $r^t \in \Re$, we assume

$$r^t = g(x^t) + \epsilon$$

In parametric regression, we assume a polynomial of a certain order and compute its coefficients that minimize the sum of squared error on the training set. Nonparametric regression is used when no such polynomial can be assumed; we only assume that close x have close $g(x)$ values. As in nonparametric density estimation, given x, our approach is to find the neighborhood of x and average the r values in the neighborhood to calculate $\hat{g}(x)$. The nonparametric regression estimator is also called a *smoother* and the estimate is called a *smooth* (Härdle 1990). There are various methods for defining the neighborhood and averaging in the neighborhood, similar to methods in density estimation. We discuss the methods for the univariate x; they can be generalized to the multivariate case in a straightforward manner using multivariate kernels, as in density estimation.

SMOOTHER

8.8.1 Running Mean Smoother

If we define an origin and a bin width and average the r values in the bin as in the histogram, we get a *regressogram* (see figure 8.9)

REGRESSOGRAM

$$\hat{g}(x) = \frac{\sum_{t=1}^{N} b(x, x^t) r^t}{\sum_{t=1}^{N} b(x, x^t)} \tag{8.24}$$

where

$$b(x, x^t) = \begin{cases} 1 & \text{if } x^t \text{ is the same bin with } x \\ 0 & \text{otherwise} \end{cases}$$

Having discontinuities at bin boundaries is disturbing as is the need to fix an origin. As in the naive estimator, in the *running mean smoother*, we define a bin symmetric around x and average in there (figure 8.10).

RUNNING MEAN SMOOTHER

$$\hat{g}(x) = \frac{\sum_{t=1}^{N} w\left(\frac{x-x^t}{h}\right) r^t}{\sum_{t=1}^{N} w\left(\frac{x-x^t}{h}\right)} \tag{8.25}$$

where

$$w(u) = \begin{cases} 1 & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

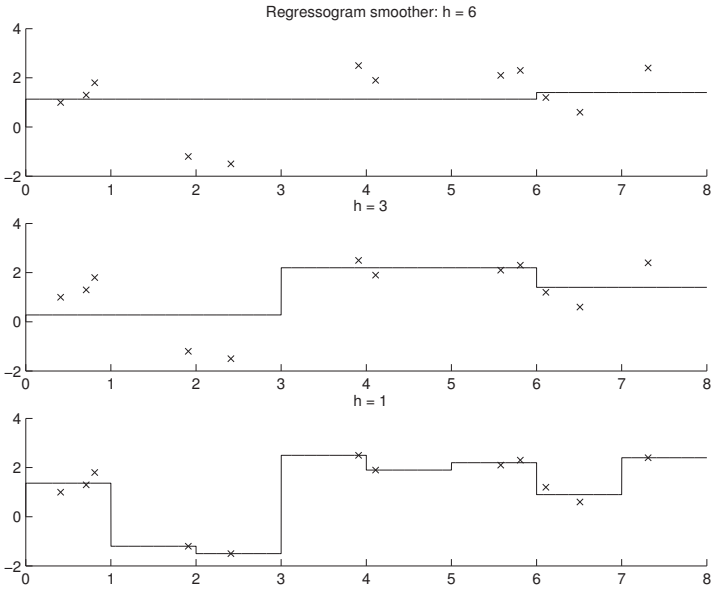

Figure 8.9 Regressograms for various bin lengths. '×' denote data points.

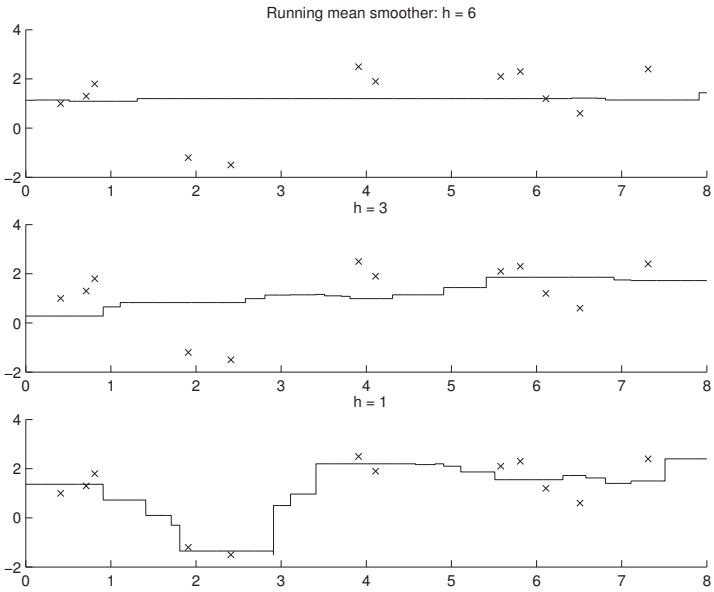

Figure 8.10 Running mean smooth for various bin lengths.

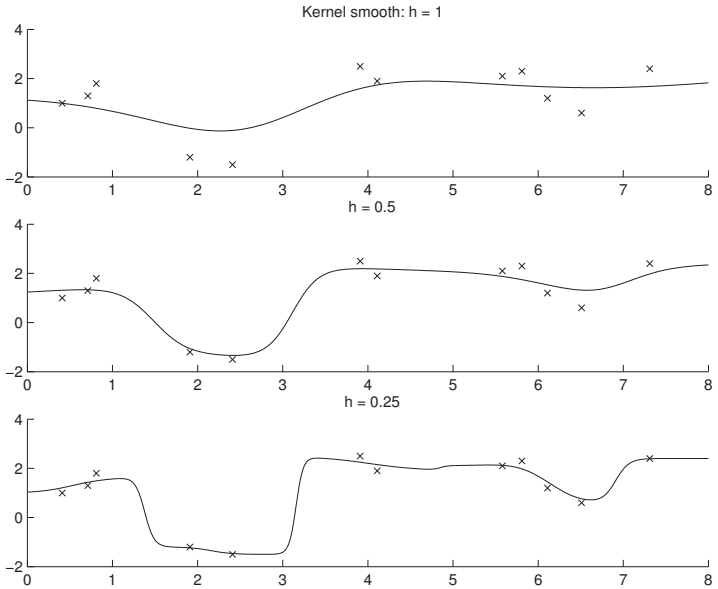

Figure 8.11 Kernel smooth for various bin lengths.

This method is especially popular with evenly spaced data, such as time series. In applications where there is noise, we can use the median of the r^t in the bin instead of their mean.

8.8.2 Kernel Smoother

KERNEL SMOOTHER

As in the kernel estimator, we can use a kernel giving less weight to further points, and we get the *kernel smoother* (see figure 8.11):

$$\hat{g}(x) = \frac{\sum_t K\left(\frac{x-x^t}{h}\right) r^t}{\sum_t K\left(\frac{x-x^t}{h}\right)} \tag{8.26}$$

k-NN SMOOTHER

Typically a Gaussian kernel $K(\cdot)$ is used. Instead of fixing h, we can fix k, the number of neighbors, adapting the estimate to the density around x, and get the *k-nn smoother*.

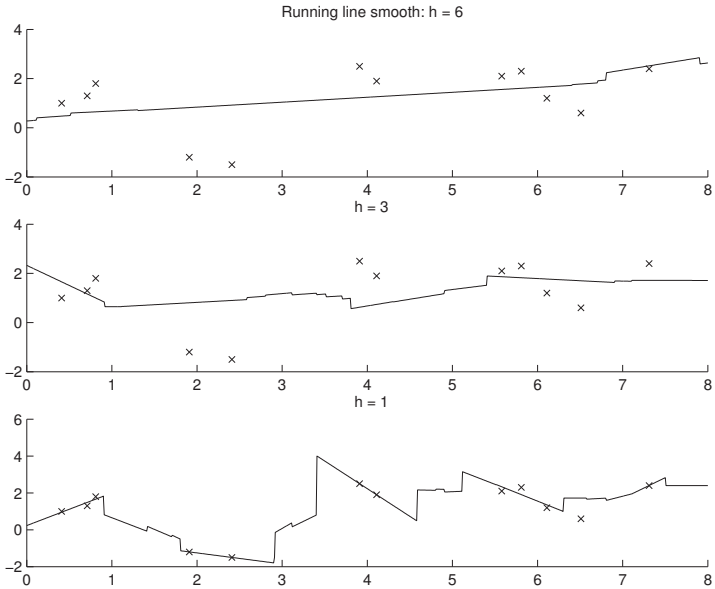

Figure 8.12 Running line smooth for various bin lengths.

8.8.3 Running Line Smoother

Instead of taking an average and giving a constant fit at a point, we can take into account one more term in the Taylor expansion and calculate a linear fit. In the *running line smoother*, we can use the data points in the neighborhood, as defined by h or k, and fit a local regression line (see figure 8.12).

RUNNING LINE SMOOTHER

In the *locally weighted running line smoother*, known as *loess*, instead of a hard definition of neighborhoods, we use kernel weighting such that distant points have less effect on error.

LOCALLY WEIGHTED RUNNING LINE SMOOTHER

8.9 How to Choose the Smoothing Parameter

In nonparametric methods, for density estimation or regression, the critical parameter is the smoothing parameter as used in bin width or kernel spread h, or the number of neighbors k. The aim is to have an estimate that is less variable than the data points. As we have discussed previously, one source of variability in the data is noise and the other is the

variability in the unknown underlying function. We should smooth just enough to get rid of the effect of noise—not less, not more. With too large h or k, many instances contribute to the estimate at a point and we also smooth the variability due to the function (oversmoothing); with too small h or k, single instances have a large effect and we do not even smooth over the noise (undersmoothing). In other words, small h or k leads to small bias but large variance. Larger h or k decreases variance but increases bias. Geman, Bienenstock, and Doursat (1992) discuss bias and variance for nonparametric estimators.

SMOOTHING SPLINES

This requirement is explicitly coded in a regularized cost function as used in *smoothing splines*:

$$\sum_t [r^t - \hat{g}(x^t)]^2 + \lambda \int_a^b [\hat{g}''(x)]^2 dx \tag{8.27}$$

The first term is the error of fit. $[a, b]$ is the input range; $\hat{g}''(\cdot)$ is the *curvature* of the estimated function $\hat{g}(\cdot)$ and as such measures the variability. Thus the second term penalizes fast-varying estimates. λ trades off variability and error where, for example, with large λ, we get smoother estimates.

Cross-validation is used to tune h, k, or λ. In density estimation, we choose the parameter value that maximizes the likelihood of the validation set. In a supervised setting, trying a set of candidates on the training set (see figure 8.13), we choose the parameter value that minimizes the error on the validation set.

8.10 Notes

k-nearest neighbor and kernel-based estimation were proposed sixty years ago, but because of the need for large memory and computation, the approach was not popular for a long time (Aha, Kibler, and Albert 1991). With advances in parallel processing and with memory and computation getting cheaper, such methods have recently become more widely used. Textbooks on nonparametric estimation are Silverman 1986 and Scott 1992. Dasarathy 1991 is a collection of many papers on k-nn and editing/condensing rules; Aha 1997 is another collection.

The nonparametric methods are very easy to parallelize on a Single Instruction Multiple Data (SIMD) machine; each processor stores one training instance in its local memory and in parallel computes the kernel

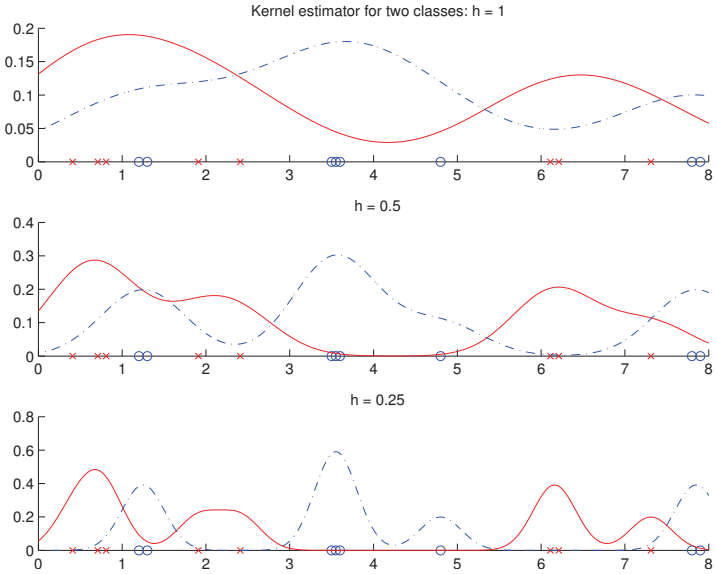

Figure 8.13 Kernel estimate for various bin lengths for a two-class problem. Plotted are the conditional densities, $p(x|C_i)$. It seems that the top one oversmooths and the bottom undersmooths, but whichever is the best will depend on where the validation data points are.

function value for that instance (Stanfill and Waltz 1986). Multiplying with a kernel function can be seen as a convolution, and we can use Fourier transformation to calculate the estimate more efficiently (Silverman 1986). It has also been shown that spline smoothing is equivalent to kernel smoothing.

CASE-BASED REASONING

In artificial intelligence, the nonparametric approach is called *case-based reasoning*. The output is found by interpolating from known similar past "cases." This also allows for some knowledge extraction: The given output can be justified by listing these similar past cases.

Due to its simplicity, k-nn is the most widely used nonparametric classification method and is quite successful in practice in a variety of applications. One nice property is that they can be used even with very few labeled instances; for example, in a forensic application, we may have only one face image per person.

It has been shown (Cover and Hart 1967; reviewed in Duda, Hart, and

Stork 2001) that in the large sample case when $N \to \infty$, the risk of nearest neighbor ($k = 1$) is never worse than twice the Bayes' risk (which is the best that can be achieved), and, in that respect, it is said that "half of the available information in an infinite collection of classified samples is contained in the nearest neighbor" (Cover and Hart 1967, 21). In the case of k-nn, it has been shown that the risk asymptotes to the Bayes' risk as k goes to infinity.

The most critical factor in nonparametric estimation is the distance metric used. With discrete attributes, we can simply use the Hamming distance where we just sum up the number of nonmatching attributes. More sophisticated distance functions are discussed in Wettschereck, Aha, and Mohri 1997 and Webb 1999.

Distance estimation or metric learning is a popular research area; see Bellet, Habrard, and Sebban 2013 for a comprehensive recent survey. The different ways similarity measures can be used in classification are discussed by Chen et al. (2009); examples of local distance methods in computer vision are given in Ramanan and Baker 2011.

Outlier/anomaly/novelty detection arises as an interesting problem in various contexts, from faults to frauds, and in detecting significant deviations from the past data, for example, churning customers. It is a very popular research area, and two comprehensive surveys include those by Hodge and Austin (2004) and Chandola, Banerjee, and Kumar (2009).

Nonparametric regression is discussed in detail in Härdle 1990. Hastie and Tibshirani (1990) discuss smoothing models and propose *additive models* where a multivariate function is written as a sum of univariate estimates. Locally weighted regression is discussed in Atkeson, Moore, and Schaal 1997. These models bear much similarity to radial basis functions and mixture of experts that we discuss in chapter 12.

ADDITIVE MODELS

In the condensed nearest neighbor algorithm, we saw that we can keep only a subset of the training instances, those that are close to the boundary, and we can define the discriminant using them only. This idea bears much similarity to the *support vector machines* that we discuss in chapter 13. There we also discuss various kernel functions to measure similarity between instances and how we can choose the best. Writing the prediction as a sum of the combined effects of training instances also underlies *Gaussian processes* (chapter 16), where a kernel function is called a *covariance function*.

8.11 Exercises

1. How can we have a smooth histogram?

   SOLUTION: We can interpolate between the two nearest bin centers. We can consider the bin centers as x^t, consider the histogram values as r^t, and use any interpolation scheme, linear or kernel-based.

2. Show equation 8.16.

   SOLUTION: Given that

   $$\hat{p}(\boldsymbol{x}|C_i) = \frac{k_i}{N_i V^k(\boldsymbol{x})} \quad \text{and} \quad \hat{P}(C_i) = \frac{N_i}{N}$$

   we can write

   $$\begin{aligned}\hat{P}(C_i|\boldsymbol{x}) &= \frac{\hat{p}(\boldsymbol{x}|C_i)\hat{P}(C_i)}{\sum_j \hat{p}(\boldsymbol{x}|C_j)\hat{P}(C_j)} = \frac{\frac{k_i}{N_i V^k(\boldsymbol{x})}\frac{N_i}{N}}{\sum_j \frac{k_j}{N_j V^k(\boldsymbol{x})}\frac{N_j}{N}} \\ &= \frac{k_i}{\sum_j k_j} = \frac{k_i}{k}\end{aligned}$$

3. Parametric regression (section 5.8) assumes Gaussian noise and hence is not robust to outliers; how can we make it more robust ?

4. How can we detect outliers after hierarchical clustering (section 7.8) ?

5. How does condensed nearest neighbor behave if $k > 1$?

   SOLUTION: When $k > 1$, to get full accuracy without any misclassification, it may be necessary to store an instance multiple times so that the correct class gets the majority of the votes. For example, if $k = 3$ and $\boldsymbol{x}$ has two neighbors both belonging to a different class, we need to store $\boldsymbol{x}$ twice (i.e., it gets added in two epochs), so that if $\boldsymbol{x}$ is seen during test, the majority (two in this case) out of three neighbors belong to the correct class.

6. In condensed nearest neighbor, an instance previously added to $\mathcal{Z}$ may no longer be necessary after a later addition. How can we find such instances that are no longer necessary?

7. In a regressogram, instead of averaging in a bin and doing a constant fit, we can use the instances falling in a bin and do a linear fit (see figure 8.14). Write the code and compare this with the regressogram proper.

8. Write the error function for loess discussed in section 8.8.3.

   SOLUTION: The output is calculated using a linear model $g(x) = ax+b$, where, in the running line smoother, we minimize

   $$E(a, b|x, \mathcal{X}) = \sum_t w\left(\frac{x - x^t}{h}\right)[r^t - (ax^t + b)]^2$$

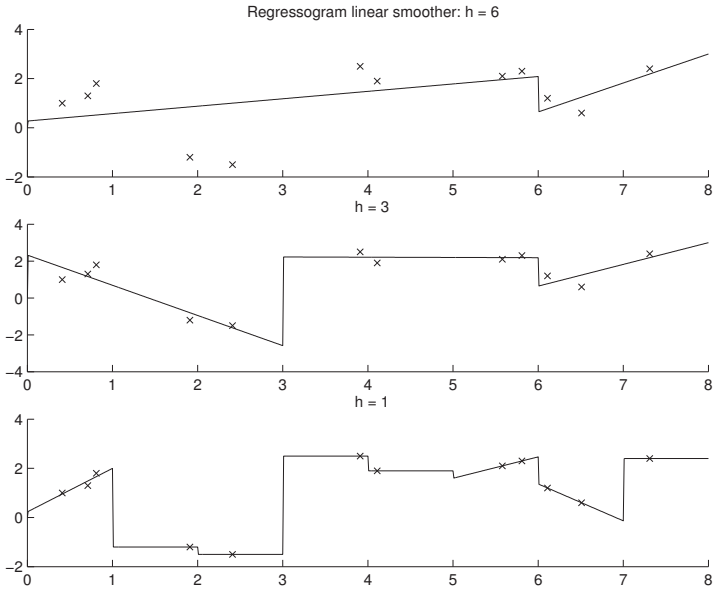

Figure 8.14 Regressograms with linear fits in bins for various bin lengths.

and

$$w(u) = \begin{cases} 1 & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

Note that we do not have one error function but rather, for each test input x, we have another error function taking into account only the data closest to x, which is minimized to fit a line in that neighborhood.

Loess is the weighted version of running line smoother where a kernel function $K(\cdot) \in (0,1)$ replaces the $w(\cdot) \in \{0,1\}$:

$$E(a,b|x,\mathcal{X}) = \sum_t K\left(\frac{x-x^t}{h}\right)[r^t - (ax^t+b)]^2$$

9. Propose an incremental version of the running mean estimator, which, like the condensed nearest neighbor, stores instances only when necessary.
10. Generalize kernel smoother to multivariate data.
11. In the running smoother, we can fit a constant, a line, or a higher-degree polynomial at a test point. How can we choose among them?

    SOLUTION: By cross-validation.

12. In the running mean smoother, besides giving an estimate, can we also calculate a confidence interval indicating the variance (uncertainty) around the estimate at that point?

8.12 References

Aha, D. W., ed. 1997. Special Issue on Lazy Learning. *Artificial Intelligence Review* 11 (1–5): 7–423.

Aha, D. W., D. Kibler, and M. K. Albert. 1991. "Instance-Based Learning Algorithm." *Machine Learning* 6:37–66.

Atkeson, C. G., A. W. Moore, and S. Schaal. 1997. "Locally Weighted Learning." *Artificial Intelligence Review* 11:11–73.

Bellet, A., A. Habrard, and M. Sebban. 2013. "A Survey on Metric Learning for Feature Vectors and Structured Data." *arXiv:1306.6709v2*.

Breunig, M. M., H.-P. Kriegel, R. T. Ng, and J. Sander. 2000. "LOF: Identifying Density-Based Local Outliers." In *ACM SIGMOD International Conference on Management of Data*, 93–104. New York: ACM Press.

Chandola, V., A. Banerjee, and V. Kumar. 2009. "Anomaly Detection: A Survey." *ACM Computing Surveys* 41 (3): 15:1–15:58.

Chen, Y., E. K. Garcia, M. R. Gupta, A. Rahimi, and L. Cazzanti. 2009. "Similarity-Based Classification: Concepts and Algorithms." *Journal of Machine Learning Research* 11:747–776.

Cover, T. M., and P. E. Hart. 1967. "Nearest Neighbor Pattern Classification." *IEEE Transactions on Information Theory* 13:21–27.

Dasarathy, B. V. 1991. *Nearest Neighbor Norms: NN Pattern Classification Techniques*. Los Alamitos, CA: IEEE Computer Society Press.

Domeniconi, C., J. Peng, and D. Gunopulos. 2002. "Locally Adaptive Metric Nearest-Neighbor Classification." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24:1281–1285.

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

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

Härdle, W. 1990. *Applied Nonparametric Regression*. Cambridge, UK: Cambridge University Press.

Hart, P. E. 1968. "The Condensed Nearest Neighbor Rule." *IEEE Transactions on Information Theory* 14:515–516.

Hastie, T. J., and R. J. Tibshirani. 1990. *Generalized Additive Models*. London: Chapman and Hall.

Hastie, T. J., and R. J. Tibshirani. 1996. "Discriminant Adaptive Nearest Neighbor Classification." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18:607–616.

Hodge, V. J., and J. Austin. 2004. "A Survey of Outlier Detection Methodologies." *Artificial Intelligence Review* 22:85–126.

Pekalska, E., and R. P. W. Duin. 2002. "Dissimilarity Representations Allow for Building Good Classifiers." *Pattern Recognition Letters* 23:943–956.

Ramanan, D., and S. Baker. 2011. "Local Distance Functions: A Taxonomy, New Algorithms, and an Evaluation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33:794–806.

Scott, D. W. 1992. *Multivariate Density Estimation*. New York: Wiley.

Silverman, B. W. 1986. *Density Estimation in Statistics and Data Analysis*. London: Chapman and Hall.

Stanfill, C., and D. Waltz. 1986. "Toward Memory-Based Reasoning." *Communications of the ACM* 29:1213–1228.

Webb, A. 1999. *Statistical Pattern Recognition*. London: Arnold.

Weinberger, K. Q., and L. K. Saul. 2009. "Distance Metric Learning for Large Margin Classification." *Journal of Machine Learning Research* 10:207–244.

Wettschereck, D., D. W. Aha, and T. Mohri. 1997. "A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms." *Artificial Intelligence Review* 11:273–314.

Wilfong, G. 1992. "Nearest Neighbor Problems." *International Journal on Computational Geometry and Applications* 2:383–416.