Introduction to Machine Learning

Ethem Alpaydın

The MIT Press

Solutions Manual

Please email remarks, suggestions, corrections to
alpaydin@boun.edu.tr

Version 1 Printed on January 10, 2007

1 *Introduction*

1. *Imagine you have two possibilities: You can fax a document, that is, send the image, or you 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?*

   The text file typically is shorter than the image file but a faxed document can also contain diagrams, pictures, etc. After using an OCR, we lose properties such as font, size, etc (unless we also recognize and transmit such information) or the personal touch if it is handwritten text. OCR may not be perfect, and for ambigious cases, OCR should identify those image blocks and transmit them as they are. A fax machine is cheaper and easier to find than a computer with scanner and OCR software.

   OCR is good if we have high volume, good quality documents; for documents of few pages with small amount of text, it is better to transmit the image.

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?*

   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 you typically see in vouchers and banking slips, which are used with OCR software, and you may have already noticed how the characters in these fonts have been slightly changed to minimize the similarities between them. Bar-

code readers are still used because reading barcodes is still a better (cheaper, more reliable, more available) technology than reading characters.

3. *Assume we are given the task to build a system that can distinguish junk e-mail. What is in a junk e-mail that lets us know that it is junk? How can the computer detect junk through a syntactic analysis? What would you like the computer to do if it detects a junk e-mail—delete it automatically, move it to a different file, or just highlight it on the screen?*

   Typically, spam filters check for the existence/absence of words and symbols. Words such as “opportunity”, ”viagra”, ”dollars” as well as characters such as ’$’, '!' 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 (One very frequently used method for spam filtering is the *naive Bayes' classifier* which we discuss in Section 5.7).

   The spam filters do not work with 100 percent reliability and frequently make errors in classification. If a junk mail is not filtered and showed to the user, this is not good, but it is not as bad as filtering a good mail as spam. 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.

   Note that filtering spam will probably never be solved completely as the spammers keep finding novel ways to outdo the filters: They use digit ‘0’ instead of the letter ’O’, digit ‘1’ instead of letter ‘l’ to pass the word tests, add pieces of texts from regular messages for the mail to be considered not spam, or send it as image not as text (and lately distort the image in small random amounts to that it is not always the same image). Still, 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 you are given the task of building an automated taxi. Define the constraints. What are the inputs? What is the output? How can you communicate with the passenger? Do you need to communicate with the other automated taxis, that is, do you need a “language”?*

An automated taxi should be able to pick a passenger and drive him/her to a destination. It should have some positioning system (GPS/GIS) and should have other sensors (cameras) to be able to sense cars, pedestrians, obstacles etc on the road. The output should be the sequence of actions to reach the destination in the smallest time with the minimum inconvenience to the passenger. The automated taxi needs to communicate with the passenger to receive commands and may also need to interact with other automated taxis to exhange information about road traffic or scheduling, load balancing, etc.

5. *In basket analysis, we want to find the dependence between two items X and Y. Given a database of customer transactions, how can you find these dependencies? How would you generalize this to more than two items?*

   This is discussed in Section 3.9.

6. *How can you predict the next command to be typed by the user? Or the next page to be downloaded over the Web? When would such a prediction be useful? When would it be annoying?*

   These are also other applications of basket analysis. The result of any statistical estimation has the risk of being wrong. That is, such dependencies should always be taken as an advice which the user can then adopt or refuse. Assuming them to be true and taking automatic action accordingly would be annoying.

2 *Supervised Learning*

1. *Write the computer program that finds S and G from a given training set.*

   The Matlab code given in `ex2_1.m` does not consider multiple possible generalizations of S or specializations of G and therefore may not work for small datasets. An example run is given in figure 2.1.

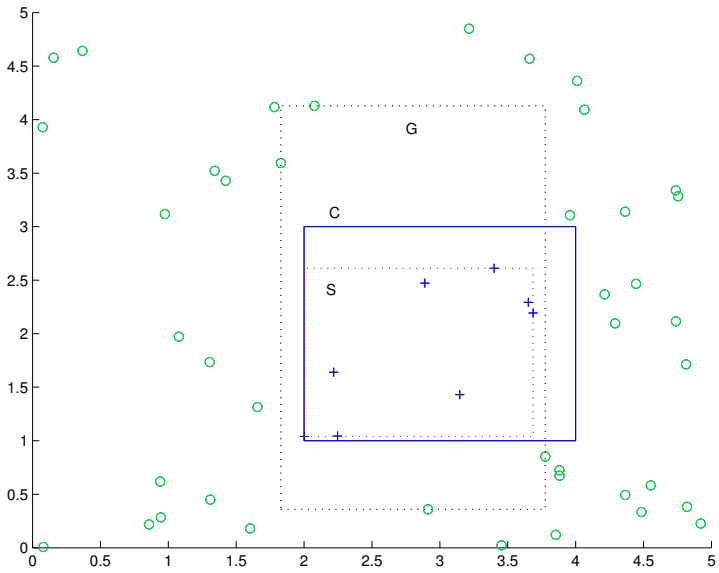


Figure 2.1 '+'/'o' are the positive and negative examples. C, S and G are the actual concept, the most specific hypothesis and the most general hypothesis respectively.

2. *Imagine you are given the training instances one at a time, instead of all at once. How can you incrementally adjust S and G in such a case? (Hint: See the candidate elimination algorithm in Mitchell 1997.)*

   The *candidate elimination* algoritm proposed by Mitchell starts with S as the null set and G as containing the whole input space. At each instance x, S and G are updated as follows (Mitchell, 1997; p. 33):

   - If x is a positive example, remove any $g \in G$ that covers x and expand any $s \in S$ that does not cover x
   - If x is a negative example, remove any $s \in S$ that covers x and restrict any $g \in G$ that does cover x

   The important point is that when we are restricting a $g \in G$ (specialization) or expanding a $s \in S$ (generalization), there may be more than one way of doing it, and this creates multiple hypotheses in S or G. For example, in figure 2.2, if we see a negative example at $(20000, 2000)$ after two positive examples $G = (-\infty < x < \infty, -\infty < y < \infty)$ splits in two: $G = (-\infty < x < 20000, -\infty < y < \infty), (-\infty < x < \infty, -\infty < y < 2000)$. These are two different ways of specializing G so that it does not include any positive example.

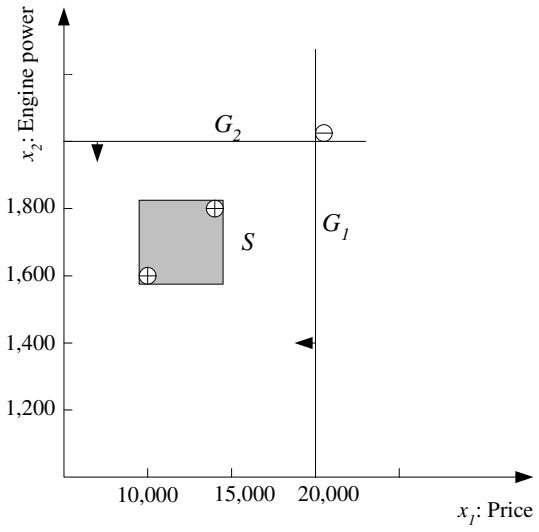


Figure 2.2 There are two specializations of G.

3. *Why is it better to use the average of S and G as the final hypothesis?*

   If there is noise, instances may be slightly changed; in such a case, using halfway between S and G will make the hypothesis robust to such small perturbations.

4. *Let us say our hypothesis class is a circle instead of a rectangle. What are the parameters? How can the parameters of a circle hypothesis be calculated in such a case? What if it is an ellipse? Why does it make more sense to use an ellipse instead of a circle? How can you generalize your code to $K > 2$ classes?*

   In the case of a circle, the parameters are the center and the radius (see figure 2.3). We then need to find the tightest circle that includes all the positive examples as S and G will be the largest circle that includes all the positive examples and no negative example:

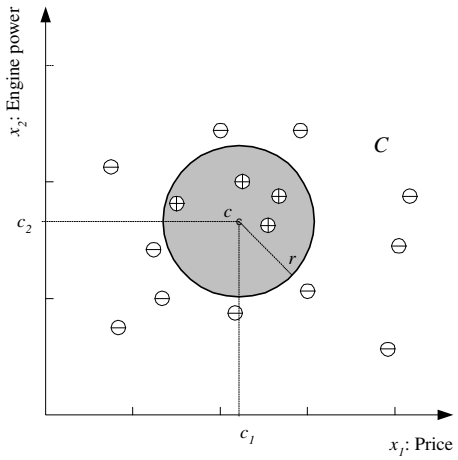


Figure 2.3 Hypothesis class is a circle with two parameters, the coordinates of its center and its radius.

   It makes more sense to use an ellipse because the two axes need not have the same scale and an ellipse has two separate parameters for the widths in the two axes rather than a single radius.

   When there are $K > 2$ classes, we need a separate circle/ellipse for each class. For each class C_i, there will be one hypothesis which takes

all elements of C_i as positive examples and instances of all $C_j, j \neq i$ as negative examples.

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

   In the case when there is a single rectangle, all the positive instances should form one single group; by increasing the number of rectangles, we get flexibility. With two rectangles for example (see figure 2.4), the positive instances can form two, possibly disjoint clusters in the input space. Note that each rectangle corresponds to a conjunction on the two input attributes and having multiple rectangles, corresponds to a disjunction. Any logical formula can be written as a disjunction of conjunctions. In the worst case ($m = N$), we can have a separate rectangle for each positive instance.

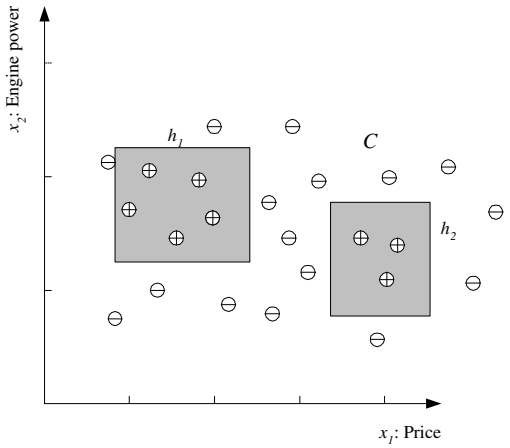


Figure 2.4 Hypothesis class is a union of two rectangles.

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

   The region of ambiguity is between S and G. It would be best to be given queries there so that we can make this region of doubt smaller. If a given instance there turns out to be positive, this means we can

make S larger up to that instance; if it is negative, this means we can shrink G up until there.

7. *In equation 2.12, we summed up the squares of the differences between the actual value and the estimated value. This error function is the one most frequently used, but it is one of several possible error functions. Because it sums up the squares of the differences, it is not robust to outliers. What would be a better error function to implement* robust regression*?*

   As we see in Chapter 4, the squared error corresponds to assuming that there is Gaussian noise. If the noise comes from a distribution with long tails, then summing up squared differences cause a few, far away points, i.e., outliers, to corrupt the fitted line.

   To decrease the effect of outliers, we can sum up the absolute value of differences instead of squaring them:

$$E(g|\mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N}|r^t - g(x^t)|$$

   but note that we lose from differentiability. Support vector regression which we discuss in Chapter 10 uses an error function (see equation 10.61 and figure 10.13) which uses absolute difference and also has a term that neglects the error due to very small differences.

8. *Derive equation 2.16.*

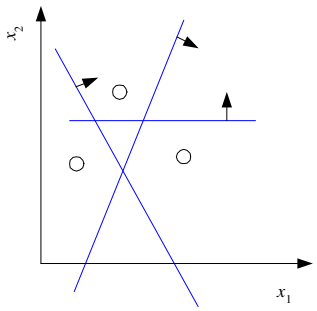
   We take the derivative of the sum of squared errors with respect to the two parameters, set them equal to 0 and solve these two equations in two unknowns:

$$\begin{aligned}
E(w_1, w_0|\mathcal{X}) &= \frac{1}{N}\sum_{i=1}^{N}\left[r^t - (w_1x^t + w_0)\right]^2 \\
\frac{\partial E}{\partial w_0} &= \sum_t \left[r^t - (w_1x^t + w_0)\right] = 0 \\
\sum_t r^t &= w_1\sum_t x^t + Nw_0 \\
w_0 &= \sum_t r^t/N - w_1\sum_t x^t/N = \overline{r} - w_1\overline{x} \\
\frac{\partial E}{\partial w_1} &= \sum_t \left[r^t - (w_1x^t + w_0)\right]x^t = 0
\end{aligned}$$

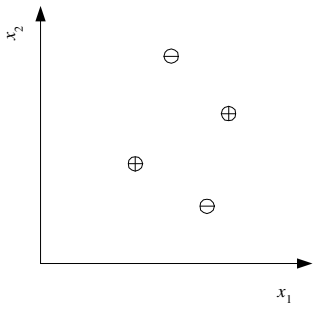
$$\begin{aligned}
\sum_t r^t x^t &= w_1 \sum_t (x^t)^2 + w_0 \sum_t x^t \\
\sum_t r^t x^t &= w_1 \sum_t (x^t)^2 + (\overline{r} - w_1 \overline{x}) \sum_t x^t \\
\sum_t r^t x^t &= w_1 \left(\sum_t (x^t)^2 - \overline{x} \sum_t x^t \right) + \overline{r} \sum_t x^t \\
\sum_t r^t x^t &= w_1 \left(\sum_t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

9. *Assume our hypothesis class is the set of lines, and we use a line to separate the positive and negative examples, instead of bounding the positive examples as in a rectangle, leaving the negatives outside (see figure 2.10). Show that the VC dimension of a line is 3.*

   As we see in figure 2.5 below, for all possible labeling of three points, there exist a line to separate positive and negative examples. With four points, no matter how we place these four points in two dimensions, there is at least one labeling where we cannot draw a line such that on one side lie all the positives and on the other lie all the negatives.



All possible labelings of three points can be separated using a line.



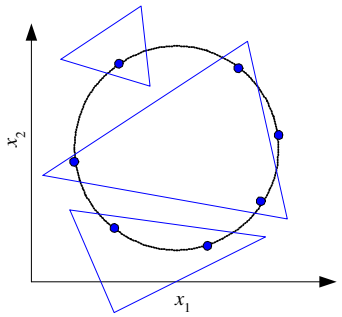
These four points cannot be separated using a line.

Figure 2.5 With a line, we can shatter three points but not four.

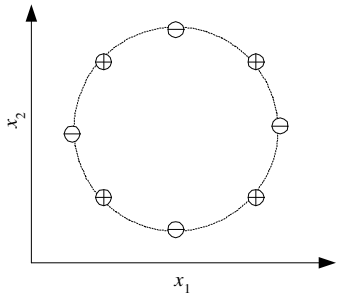
10. *Show that the VC dimension of the triangle hypothesis class is 7 in two*

dimensions. (Hint: For best separation, it is best to place the seven points equidistant on a circle.)

As we can see in figure 2.6, for all possible labeling of seven points, we can draw a triangle to separate the positive and negative examples. We cannot do the same when there are eight points.



These seven points can be separated using a triangle no matter how they are labeled.



These eight points with this labeling cannot be separated using a triangle.

Figure 2.6 A triangle can shatter seven points but not eight.

3 *Bayesian Decision Theory*

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

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

   Write the discriminant function in terms of the likelihood ratio.

   We can define a discriminant function as

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

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

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

   If the priors are equal, the discriminant is the likelihood ratio (see also the next exercise).

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

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

   Write the discriminant function in terms of the log odds.

   We define a discriminant function as

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

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

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

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

3. *In a two-class, two-action problem, if the loss function is* $\lambda_{11} = \lambda_{22} = 0$, $\lambda_{12} = 10$, *and* $\lambda_{21} = 1$, *write the optimal decision rule.*

   We calculate the expected risks of the two actions:

$$\begin{aligned} R(\alpha_1|x) &= \lambda_{11}P(C_1|x) + \lambda_{12}P(C_2|x) = 10P(C_2|x) \\ R(\alpha_2|x) &= \lambda_{21}P(C_1|x) + \lambda_{22}P(C_2|x) = P(C_1|x) \end{aligned}$$

   and we choose C_1 if $R(\alpha_1|x) < R(\alpha_2|x)$, or if $P(C_1|x) > 10P(C_2|x)$, $P(C_1|x) > 10/11$. Assigning accidentally an instance of C_2 to C_1 is so bad that we need to be very sure before assigning an instance to C_1.

4. *Somebody tosses a fair coin and if the result is heads, you get nothing, otherwise you get \$5. How much would you pay to play this game? What if the win is \$500 instead of \$5?*

   With five dollars, the expected earning is $(1/2) \cdot 0 + (1/2) \cdot 5 = 2.5$ dollars and one can bet up that amount to play the game. With a reward of 500 dollars, the expected earning is 250 and one can bet up that amount. However research has shown that people are risk aversive, in the sense that though they may risk small amounts, they do not like to risk larger amounts, even though odds may be more favorable. See for example, A. Tversky, D. Kahneman. 1981. "The framing of decisions and the psychology of choice," *Science* 211: 453–458.

5. *In figure 3.4, calculate* $P(C|W)$.

$$\begin{aligned} P(C|W) &= \frac{P(W|C)P(C)}{P(W)} \\ P(W) &= P(W|C)P(C) + P(W|\sim C)P(\sim C) \\ P(W|\sim C) &= P(W|R,S,\sim C)P(R,S|\sim C) \\ &\quad + P(W|\sim R,S,\sim C)P(\sim R,S|\sim C) \end{aligned}$$

$$
\begin{aligned}
& +P(W|R,\sim S,\sim C)P(R,\sim S|\sim C) \\
& +P(W|\sim R,\sim S,\sim C)P(\sim R,\sim S|\sim C) \\
= \quad & P(W|R,S)P(R|\sim C)P(S|\sim C) \\
& +P(W|\sim R,S)P(\sim R|\sim C)P(S|\sim C) \\
& +P(W|R,\sim S)P(R|\sim C)P(\sim S|\sim C) \\
& +P(W|\sim R,\sim S)P(\sim R|\sim C)P(\sim S|\sim C)
\end{aligned}
$$

6. *In figure 3.5, calculate $P(F|C)$.*

$$P(F|C) = P(F|R)P(R|C) + P(F|\sim R)P(\sim R|C)$$

7. *Given the structure in figure 3.5 and a sample X containing observations as*

Cloudy	*Sprinkler*	*Rain*	*Wet grass*	*Roof*
No	*Yes*	*No*	*Yes*	*Yes*
Yes	*No*	*Yes*	*No*	*No*
⋮	⋮	⋮	⋮	⋮

how do you learn the probabilities?

We estimate the probabilities by counting the proportions of occurrences:

$$
\begin{aligned}
P(C) &= \frac{\#\{\text{cases where cloudy = 'yes'}\}}{\#\{\text{cases}\}} \\
P(S|C) &= \frac{P(S,C)}{P(C)} = \frac{\#\{\text{cases where sprinkler = 'yes' and cloudy = 'yes'}\}}{\#\{\text{cases where cloudy = 'yes'}\}} \\
&\vdots
\end{aligned}
$$

8. *Generalize the confidence and support formulas for basket analysis to calculate k-dependencies, namely, $P(Y|X_1,\ldots,X_k)$.*

We are interested in rules of the form $X_1, X_2, \ldots, X_k \rightarrow Y$:

- Support:

$$P(X_1, X_2, \ldots, X_k, Y) = \frac{\#\{\text{customers who bought } X_1 \text{ and } \ldots X_k \text{ and } Y\}}{\#\{\text{customers}\}}$$

- Confidence:

$$P(Y|X_1, X_2, \ldots, X_k) = \frac{P(X_1, X_2, \ldots, X_k, Y)}{P(X_1, X_2, \ldots, X_k)} = \frac{\#\{\text{customers who bought } X_1 \text{ and } \ldots X_k \text{ and } Y\}}{\#\{\text{customers who bought } X_1 \text{ and } \ldots X_k\}}$$

Note that people who bought X_1, X_2, X_3 over a certain number should have bought X_1, X_2 and X_1, X_3 and X_2, X_3 over the same amount. So one can expand k-dependencies from $(k-1)$-dependencies. This is the basic idea behind the Apriori algorithm.

9. *If, in basket analysis, associated with each item sold, we also have a number indicating how much the customer enjoyed the product, for example, in a scale of 0 to 10, how can you use this extra information to calculate which item to propose to a customer?*

   One can come up with a weighted confidence measure, or considering that these are numeric values now, one can calculate correlations and propose a new product that has the highest positive correlation with the current one, if the correlation is high enough.

4 *Parametric Methods*

1. *Write the code that generates a Bernoulli sample with given parameter p, and the code that calculates $\hat{p}$ from the sample.*

   The Matlab code is given in `ex4_1.m`.

2. *Write the log likelihood for a multinomial sample and show equation 4.6.*

   We add the constraint as a Lagrange term and maximize it:

$$
\begin{aligned}
J(p_i) &= \sum_i \sum_t x_i^t \log p_i + \lambda(1 - \sum_i p_i) \\
\frac{\partial J}{p_i} &= \frac{\sum_t x_i^t}{p_i} - \lambda = 0 \\
\lambda &= \frac{\sum_t x_i^t}{p_i} \Rightarrow p_i \lambda = \sum_t x_i^t \\
\sum_i p_i \lambda &= \sum_i \sum_t x_i^t \Rightarrow \lambda = \sum_t \sum_i x_i^t \\
p_i &= \frac{\sum_t x_i^t}{\sum_t \sum_i x_i^t} = \frac{\sum_t x_i^t}{N} \text{ because } \sum_i x_i^t = 1
\end{aligned}
$$

3. *Write the code that generates a normal sample with given μ and σ, and the code that calculates m and s from the sample. Do the same using the Bayes' estimator assuming a prior distribution for μ.*

   The Matlab code is given in `ex4_3.m` and we see an example output in figure 4.1.

4. *Given two normal distributions $p(x|C_1) \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $p(x|C_2) \sim \mathcal{N}(\mu_2, \sigma_2^2)$ and $P(C_1)$ and $P(C_2)$, calculate the Bayes' discriminant points analytically.*

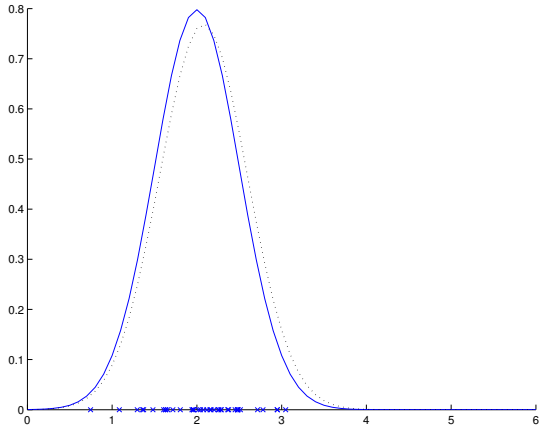


Figure 4.1 Actual Gaussian density (continuous line), a sample drawn from it (×), and a Gaussian fitted to the sample (dotted line).

We have

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

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

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

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

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

Note that if the variances are equal, $a = 0$ and there is one root, that is, the two posteriors intersect at a single x value.

5. *What is the likelihood ratio*

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

in the case of Gaussian densities?

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

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

$$\begin{aligned}\frac{p(x|C_1)}{p(x|C_2)} &= \exp\left[-\frac{(x-\mu_1)^2}{2\sigma^2} + \frac{(x-\mu_2)^2}{2\sigma^2}\right] \\ &= \exp\left[\frac{(\mu_1-\mu_2)}{\sigma^2}x + \frac{\mu_2^2-\mu_1^2}{2\sigma^2}\right] \\ &= \exp(wx + w_0)\end{aligned}$$

If we calculate the odds assuming equal priors, we have

$$\begin{aligned}\frac{P(C_1|x)}{P(C_2|x)} &= \frac{P(C_1|x)}{1-P(C_1|x)} = \frac{p(x|C_1)}{p(x|C_2)}\frac{P(C_1)}{P(C_2)} \\ &= \exp(wx + w_0) \\ P(C_1|x) &= \frac{\exp(wx+w_0)}{1+\exp(wx+w_0)} = \frac{1}{1+\exp(-(wx+w_0))}\end{aligned}$$

which is the logistic function in terms of differences of means etc.

6. *For a two-class problem, generate normal samples for two classes with different variances, then use parametric classification to estimate the discriminant points. Compare these with the theoretical values.*

The Matlab code is given in `ex4_6.m`, and its output plot is in figure 4.2.

Output:

```
Real values C1:(3.0,1.0)- C2:(2.0,0.3)
Estimates C1:(3.45,1.23)- C2:(2.15,0.26)
Actual intersect pts: 2.49 1.31
Estimated intersect pts: 2.64 1.53
```

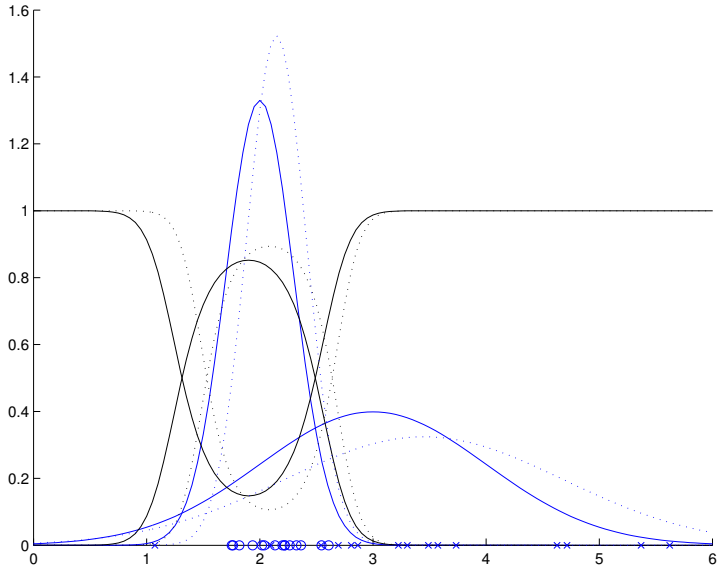


Figure 4.2 Actual densities, posteriors, drawn samples and the fitted densities and posteriors for a two-class problem. The continuous lines are the actual densities and posteriors and the dotted lines are the estimations using the data points shown by '×' and '∘'.

7. *Assume a linear model and then add 0-mean Gaussian noise to generate a sample. Divide your sample into two as training and validation sets. Use linear regression using the training half. Compute error on the validation set. Do the same for polynomials of degrees 2 and 3 as well.*

   The Matlab code given in `ex4_7.m` samples data from $2\sin(1.5x) + \mathcal{N}(0,1)$, and fits five times polynomials of order 1, 2 and 3. It also calculates their bias and variance. This is the code used to generate figure 4.5 of the book (with orders 1, 3 and 5). The plots are given in figure 4.3 (This is figure 4.5 in the book):

   Output:

```
Order 1 B=1.80 V=0.29 E=2.09
Order 2 B=1.25 V=0.38 E=1.63
Order 3 B=0.28 V=0.30 E=0.58
```

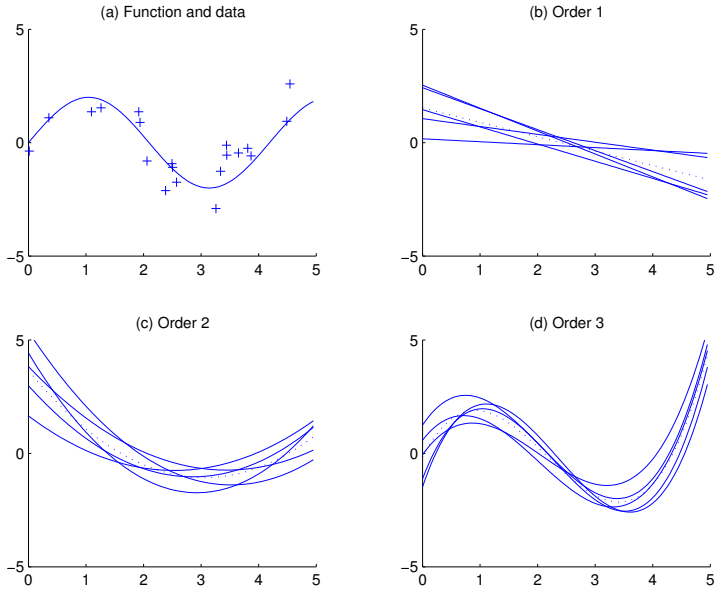


Figure 4.3 Function, one random noisy sample and five fitted polynomials of order 1, 2 and 3.

8. *When the training set is small, the contribution of variance to error may be more than that of bias and in such a case, we may prefer a simple model even though we know that it is too simple for the task. Can you give an example?*

   With small datasets and complex models, the fit will have high variance because there is not enough data to sufficiently constrain the model and the fit will follow noise. In such a case, it is better to use a simple model though it risks having high bias.

   If we run the program of exercise 4.7 here with six data points (`ex4_8.m` has $N = 6$ on line 2), we get the result in figure 4.4 and the following output

```
Order 1 B=1.66 V=0.37 E=2.03
```

```
Order 2 B=1.31 V=1.59 E=2.90
Order 3 B=3.74 V=2.86 E=6.60
```

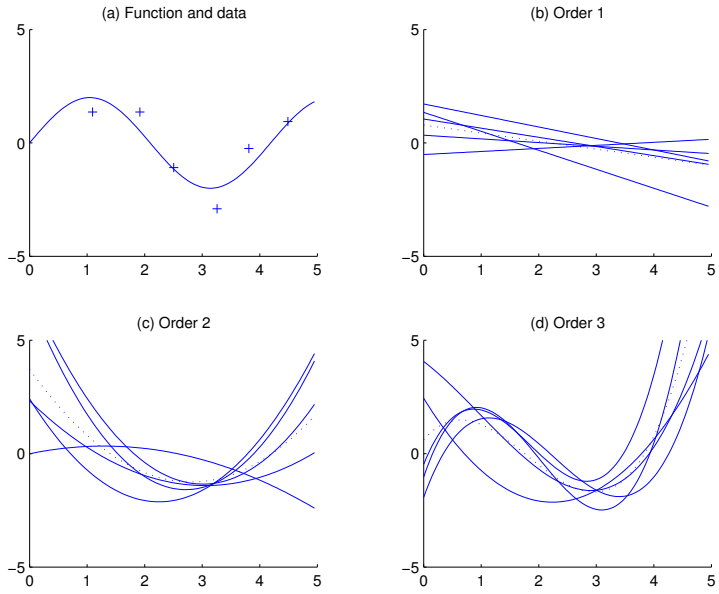


Figure 4.4 Function, one random noisy sample and five fitted polynomials of order 1, 2 and 3.

We see that a linear fit has less error though a third-degree polynomial is ideally the best.

9. *Let us say, given the samples* $\mathcal{X}_i = \{x_i^t, r_i^t\}$*, we define* $g_i(x) = r_i^1$*, namely, our estimate for any* x *is the* r *value of the first instance in the (unordered) dataset* $\mathcal{X}_i$*. What can you say about its bias and variance, as compared with* $g_i(x) = 2$ *and* $g_i(x) = \sum_t r_i^t/N$*? What if the sample is ordered, so that* $g_i(x) = \min_t r_i^t$*?*

   Taking any instance has less bias than taking a constant but has higher variance. It has higher variance than the average and it may have higher bias. If the sample is ordered so that the instance we pick is the minimum, variance decreases (minimums tend to get more similar to each other) and bias may also increase. The Matlab code is given in `ex4_9.m` and running it, we get the plot of figure 4.5 with the following output:

```
Order 0 B=1.87 V=0.10 E=1.97
First inst B=2.63 V=2.30 E=4.93
Min inst B=14.42 V=0.53 E=14.95
```

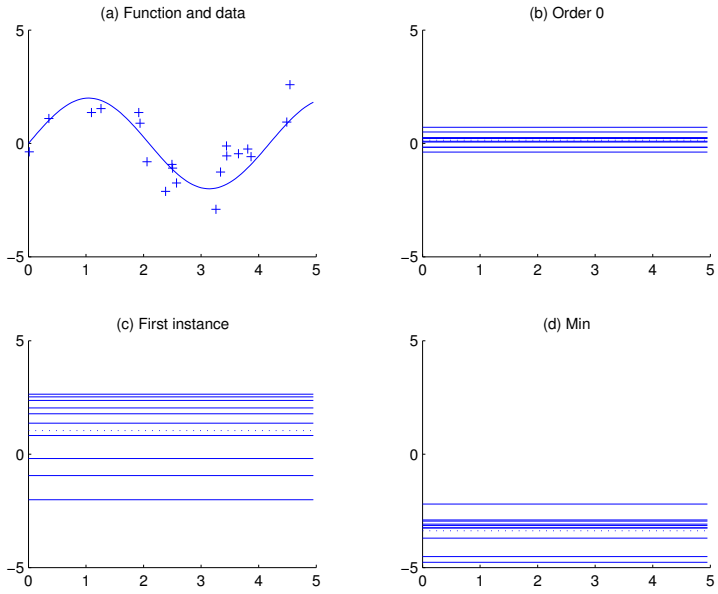


Figure 4.5 Function, one random noisy sample and the fitted models are: $\sum_t r^t/N$, r^1, $\min_t r^t$.

5 *Multivariate Methods*

1. *Show equation 5.11.*

   Given that

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

   we have

$$\begin{aligned} |\Sigma| &= \sigma_1^2\sigma_2^2 - \rho^2\sigma_1^2\sigma_2 = \sigma_1^2\sigma_2(1-\rho^2) \\ |\Sigma|^{1/2} &= \sigma_1\sigma_2\sqrt{1-\rho^2} \\ \Sigma^{-1} &= \frac{1}{\sigma_1^2\sigma_2(1-\rho^2)} \begin{bmatrix} \sigma_2^2 & -\rho\sigma_1\sigma_2 \\ -\rho\sigma_1\sigma_2 & \sigma_1^2 \end{bmatrix} \end{aligned}$$

   and $(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$ can be expanded as

$$\begin{aligned} &[x_1-\mu_1 \; x_2-\mu_2] \begin{bmatrix} \frac{\sigma_2^2}{\sigma_1^2\sigma_2(1-\rho^2)} & -\frac{\rho\sigma_1\sigma_2}{\sigma_1^2\sigma_2(1-\rho^2)} \\ -\frac{\rho\sigma_1\sigma_2}{\sigma_1^2\sigma_2(1-\rho^2)} & \frac{\sigma_1^2}{\sigma_1^2\sigma_2(1-\rho^2)} \end{bmatrix} \begin{bmatrix} x_1-\mu_1 \\ x_2-\mu_2 \end{bmatrix} \\ &= \frac{1}{1-\rho^2}\left[\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2 - 2\rho\left(\frac{x_1-\mu_1}{\sigma_1}\right)\left(\frac{x_2-\mu_2}{\sigma_2}\right) + \left(\frac{x_2-\mu_2}{\sigma_2}\right)^2\right] \end{aligned}$$

2. *Generate a sample from a multivariate normal density $\mathcal{N}(\boldsymbol{\mu},\Sigma)$, calculate $\boldsymbol{m}$ and $\mathbf{S}$, and compare them with $\boldsymbol{\mu}$ and Σ. Check how your estimates change as the sample size changes.*

   The Matlab code is given in `ex5_2.m`. The result is given in figure 5.1 for different sample sizes.

3. *Generate samples from two multivariate normal densities $\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), i = 1, 2$, and calculate the Bayes' optimal discriminant for the four cases in table 5.1.*

   The Matlab code is given in `ex5_3.m` and its output is given in figure 5.2.

4. *For a two-class problem, for the four cases of Gaussian densities in table 5.1, derive*

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

   Using Bayes' rule, we have

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

   Given that

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

   we have the following four cases:

   (a) $\boldsymbol{\Sigma}_1 \neq \boldsymbol{\Sigma}_2$:

   $$\log \frac{p(\boldsymbol{x}|C_1)}{p(\boldsymbol{x}|C_2)} = \log \frac{|\boldsymbol{\Sigma}_1|^{-1/2}\exp[-(1/2)(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1)]}{|\boldsymbol{\Sigma}_2|^{-1/2}\exp[-(1/2)(\boldsymbol{x}-\boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2)]}$$

   (b) $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \boldsymbol{\Sigma}$:

   $$\begin{aligned}\log \frac{p(\boldsymbol{x}|C_1)}{p(\boldsymbol{x}|C_2)} &= -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2) \\ &= \boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2) + \frac{1}{2}(\boldsymbol{\mu}_2-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_2+\boldsymbol{\mu}_1)\end{aligned}$$

   We can write

   $$\log \frac{P(C_1|\boldsymbol{x})}{P(C_2|\boldsymbol{x})} = \boldsymbol{w}^T\boldsymbol{x} + w_0$$

   where

   $$\begin{aligned}\boldsymbol{w} &= \boldsymbol{\Sigma}_1^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2) \\ w_0 &= -\frac{1}{2}(\boldsymbol{\mu}_1+\boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_2) + \log\frac{P(C_1)}{P(C_2)}\end{aligned}$$

(c) $\mathbf{\Sigma}$ diagonal with $\sigma_j^2, j = 1, \ldots, d$:

$$\log \frac{P(C_1|\boldsymbol{x})}{P(C_2|\boldsymbol{x})} = \sum_{j=1}^{d} \left(\frac{\mu_{1j} - \mu_{2j}}{\sigma_j} \right)^2 x_j - \frac{1}{2} \sum_{j=1}^{d} \left(\frac{\mu_{1j}^2 - \mu_{2j}^2}{\sigma_j^2} \right) + \log \frac{P(C_1)}{P(C_2)}$$

(d) $\mathbf{\Sigma}$ diagonal with σ^2:

$$\log \frac{P(C_1|\boldsymbol{x})}{P(C_2|\boldsymbol{x})} = \frac{1}{\sigma^2} \sum_{j=1}^{d} (\mu_{1j} - \mu_{2j})^2 x_j - \frac{1}{2\sigma^2} \sum_{j=1}^{d} (\mu_{1j}^2 - \mu_{2j}^2) + \log \frac{P(C_1)}{P(C_2)}$$

5. *Let us say we have two variables x_1 and x_2 and we want to make a quadratic fit using them, namely*

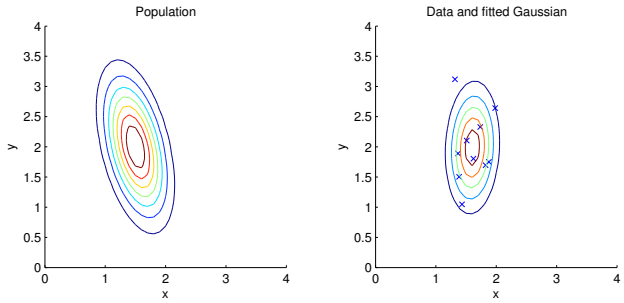
$$f(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 (x_1)^2 + w_5 (x_2)^2$$

How can we find $w_i, i = 0, \ldots, 5$, given a sample of $\mathcal{X} = \{x_1^t, x_2^t, r^t\}$?

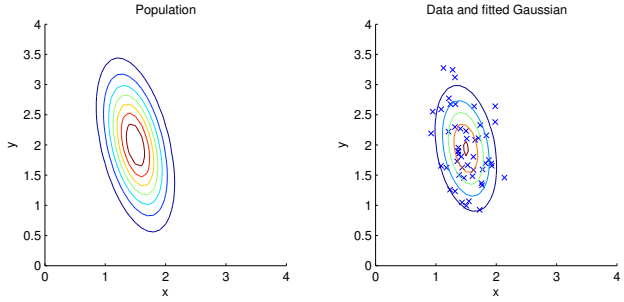
We write the fit as

$$f(x_1, x_2) = w_0 + w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + w_5 z_5$$

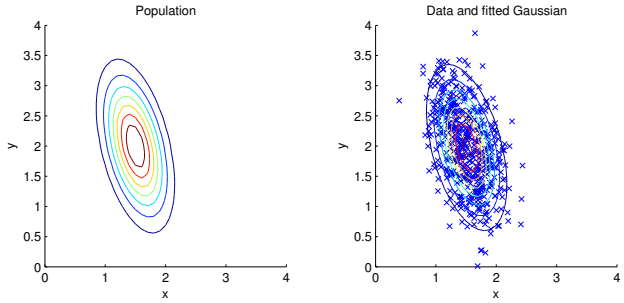
where $z_1 = x_1$, $z_2 = x_2$, $z_3 = x_1 x_2$, $z_4 = (x_1)^2$, and $z_5 = (x_2)^2$. We can then use linear regression to learn $w_i, i = 0, \ldots, 5$. The linear fit in the five-dimensional $(z_1, z_2, z_3, z_4, z_5)$ corresponds to a quadratic fit in the two-dimensional (x_1, x_2) space. We discuss such generalized linear models in more detail (and other nonlinear basis functions) in chapter 10.



(a) $N = 10$



(b) $N = 50$



(c) $N = 500$

Figure 5.1 Gaussian population and the fitted Gaussian for different sample sizes.

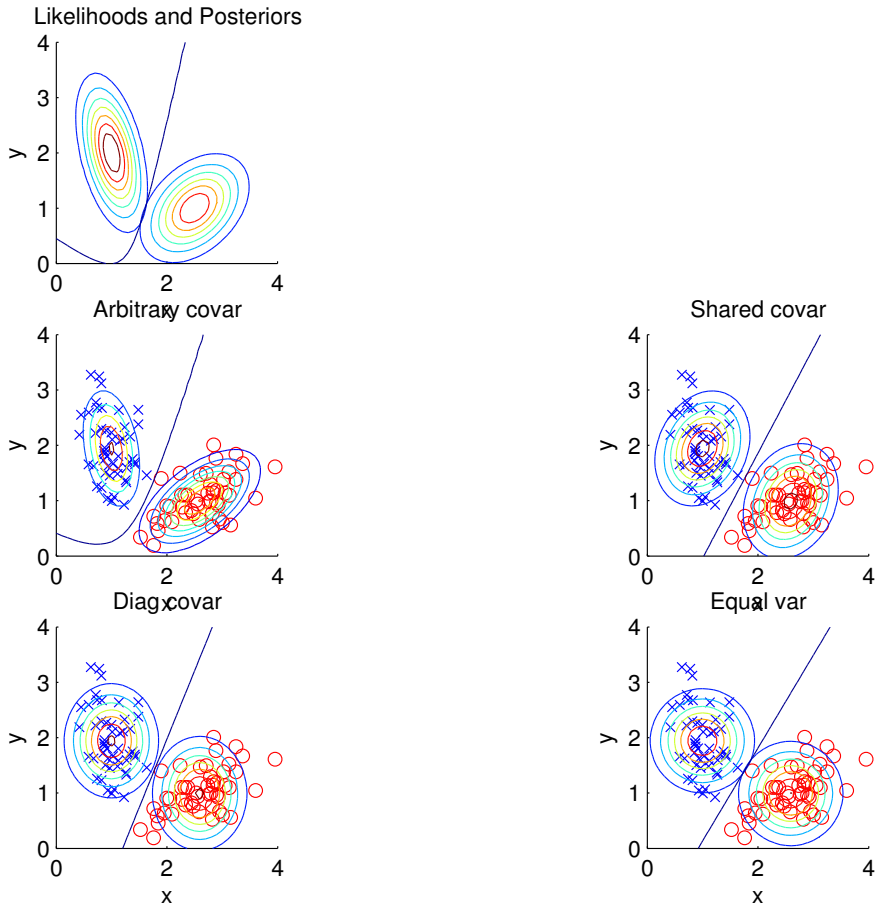


Figure 5.2 With two classes, population densities and the optimal Bayes' discriminant is shown. Then, data are sampled and the estimated likelihoods and posterior discriminants are shown for the four cases.

6 *Dimensionality Reduction*

1. *Assuming that the classes are normally distributed, in subset selection, when one variable is added or removed, how can the new discriminant be calculated quickly? For example, how can the new* $\mathbf{S}_{new}^{-1}$ *be calculated from* $\mathbf{S}_{old}^{-1}$*?*

   When we add a new variable, we are adding a row and column to the covariance matrix. It can be shown that if we partition a symmetric nonsingular matrix $\mathbf{A}$ into (Rencher, 1995; p. 28)

   $$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \boldsymbol{a}_{12} \\ \boldsymbol{a}_{12}^T & a_{22} \end{bmatrix}$$

   the inverse is given by

   $$\mathbf{A}^{-1} = \frac{1}{b}\begin{bmatrix} b\mathbf{A}_{11}^{-1} + \mathbf{A}_{11}^{-1}\boldsymbol{a}_{12}\boldsymbol{a}_{12}^T\mathbf{A}_{11}^{-1} & -\mathbf{A}_{11}^{-1}\boldsymbol{a}_{12} \\ -\boldsymbol{a}_{12}^T\mathbf{A}_{11}^{-1} & 1 \end{bmatrix}$$

   where $b = a_{22} - \boldsymbol{a}_{12}^T\mathbf{A}_{11}^{-1}\boldsymbol{a}_{12}$. So given $\mathbf{A}_{11}$ and $\mathbf{A}_{11}^{-1}$, we can easily calculate $\mathbf{A}^{-1}$ when $\boldsymbol{a}_{12}$ and a_{22} (covariances of the new variable with the existing variables, and its variance) are added.

   Another possibility is to break down the Mahalanobis distance calculation (if we have the previous values stored). Let $\mathbf{A}$ be partitioned as (T. Cormen, C. Leiserson, R. Rivest, C. Stein. 2001. *Introduction to Algorithms*, The MIT Press. 2nd edition, p. 761)

   $$\mathbf{A} = \begin{bmatrix} \mathbf{A}_k & \mathbf{B}^T \\ \mathbf{B} & \mathbf{C} \end{bmatrix}$$

where $\mathbf{A}_k$ is the leading $k \times k$ submatrix of $\mathbf{A}$ and $\boldsymbol{x}$ similarly broken into two parts, we can write

$$\begin{aligned}\boldsymbol{x}^T\mathbf{A}\boldsymbol{x} &= [\boldsymbol{y}^T\boldsymbol{z}^T]\begin{bmatrix}\mathbf{A}_k & \mathbf{B}^T\\ \mathbf{B} & \mathbf{C}\end{bmatrix}\begin{bmatrix}\boldsymbol{y}\\ \boldsymbol{z}\end{bmatrix}\\ &= (\boldsymbol{y}+\mathbf{A}_k^{-1}\mathbf{B}^T\boldsymbol{z})^T\mathbf{A}_k(\boldsymbol{y}+\mathbf{A}_k^{-1}\mathbf{B}^T\boldsymbol{z}) + \boldsymbol{z}^T(\mathbf{C}-\mathbf{B}\mathbf{A}_k^{-1}\mathbf{B}^T)\boldsymbol{z}\end{aligned}$$

In our case of adding one feature, $\boldsymbol{y}$ will be d-dimensional and z will be a scalar. If we already have the d-dimensional $\boldsymbol{y}^T\mathbf{A}_k\boldsymbol{y}$ values calculated and stored, we can use the equality above to calculate the $(d+1)$-dimensional $\boldsymbol{x}^T\mathbf{A}\boldsymbol{x}$ values easily.

2. *Using Optdigits from the UCI repository, implement PCA. For various number of eigenvectors, reconstruct the digit images and calculate the reconstruction error (equation 6.12).*

   The Matlab code is given in `ex6_2.m` (which also generates figure 6.3 in the book). The contour plot of an example '6' and its reconstruction is given in figure 6.1. The original 64 dimensional data and its reconstruction are as follows:

```
tf =
     0     0     0     0     0     0     0     0
     0     0     0     2     2     0     0     0
     0     9    15    15    16    16    12     1
    10    14     3     2    15     5    10    11
    12     4     0     6    12     0     4    14
     0     0     0     1    15     3    11    12
     0     0     0     0     4    14    14     1
     0     0     0     0     0     0     0     0

tfn =
      0  -0.0000   0.0023   0.0010   0.0034   0.0852   0.0572  -0.0000
 0.0844   1.2471   2.9502   3.6126   4.2342   3.2692   1.3435   0.0996
 3.2523   9.7849  14.2097  14.2816  12.7876  13.9419  11.7729   2.9551
10.8280  13.9309  10.1036   7.0768   7.8097   8.2514  10.9618  11.4303
 8.5466   9.3411   3.6087   2.5096   6.3869   6.0329  10.4288  16.4897
-0.4493   1.5186   2.1009   3.1204   7.5014  10.8696  13.2739  11.3677
-1.3608  -1.5062   0.5341   2.1385   3.2335   6.8721   7.8419   4.2543
-0.1859  -0.1063   0.0335   0.0034   0.0000   0.0789   0.4969   0.4434
```

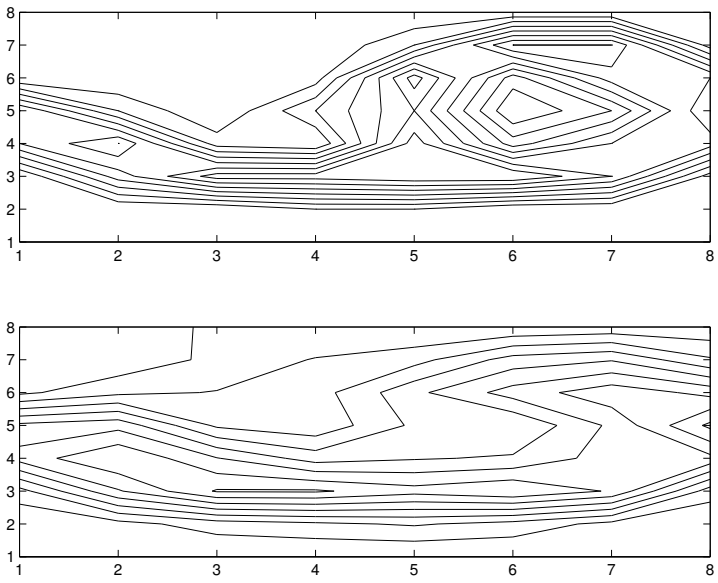


Figure 6.1 A '6' and its reconstruction using the top two eigenvectors.

3. *Plot the map of your state/country using MDS, given the road travel distances as input.*

   The Matlab code that generates figure 6.6 in the book is given in `ex6_3.m`. It generates the map of Europe from pairwise distances between 12 cities.

4. *In Sammon mapping, if the mapping is linear, namely,* $g(\boldsymbol{x}|\mathbf{W}) = \mathbf{W}^T\boldsymbol{x}$*, how can* $\mathbf{W}$ *that minimizes the Sammon stress be calculated?*

   MDS already does this linear mapping. Another possibility is to plug in the linear model in Sammon stress (equation 6.29) and find $\mathbf{W}$ that minimizes it. One can for example use an iterative, gradient-descent procedure; this is especially interesting when the mapping is nonlinear. We discuss this in chapter 11 when we use multilayer perceptrons (p. 265).

7 *Clustering*

1. *In image compression, k-means can be used as follows: The image is divided into nonoverlapping $c \times c$ windows and these c^2-dimensional vectors make up the sample. For a given k, which is generally a power of two, we do k-means clustering. The reference vectors and the indices for each window is sent over the communication line. At the receiving end, the image is then reconstructed by reading from the table of reference vectors using the indices. Write the computer program that does this for different values of k and c. For each case, calculate the reconstruction error and the compression rate.*

   This code is very straightforward, given the pseudocode of *k*-means in figure 7.3 (p. 139) of the book. One word of caution: When calculating the compression rate, do not forget to take into account the storage needed for the reference vectors. Given an $n \times n$ image, $c \times c$ windows, m reference vectors, the index of each window is coded using $\log_2 m$ bits. There are $(n/c)^2$ indices that should be so coded; so the image is transferred using $(n/c)^2 \log_2 m$ bits. There are m reference vectors each of dimensionality c^2. If we need b bits for each dimension, we need mc^2b bits to store the reference vectors.

2. *We can do k-means clustering, partition the instances, and then calculate $\mathbf{S}_i$ separately in each group. Why is this not a good idea?*

   Because it does hard partitioning. It is always better to do a soft partitioning (using $h_i^t \in (0,1)$ instead of $b_i^t \in \{0,1\}$) where instances (in between two clusters) can contribute to the parameters (the covariance matrix in this case) of more than one cluster, thereby effectively increasing the training set of each cluster and allowing a smooth transition between clusters.

3. *Derive the M-step equations for* $\mathbf{S}$ *in the case of shared arbitrary covariance matrix* $\mathbf{S}$ *(equation 7.15) and* s^2*, in the case of shared diagonal covariance matrix (equation 7.16).*

   In the case of a shared arbitrary covariance matrix, in the E-step, we have

$$h_i^t = \frac{\pi_i \exp[-(1/2)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T \mathbf{S}_i^{-1}(\boldsymbol{x}^t - \boldsymbol{m}_i)]}{\sum_j \pi_j \exp[-(1/2)(\boldsymbol{x}^t - \boldsymbol{m}_j)^T \mathbf{S}_j^{-1}(\boldsymbol{x}^t - \boldsymbol{m}_j)]}$$

   and in the M-step for the component parameters, we have

$$\min_{\boldsymbol{m}_i, \mathbf{S}} \sum_t \sum_i h_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)^T \mathbf{S}^{-1} (\boldsymbol{x}^t - \boldsymbol{m}_i)$$

   The update equation for $\boldsymbol{m}_i$ does not change but for the common covariance matrix, we have

$$\begin{aligned} \mathbf{S}^l &= \frac{\sum_t \sum_i h_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i^{l+1})(\boldsymbol{x}^t - \boldsymbol{m}_i^{l+1})^T}{\sum_t \sum_i h_i^t} \\ &= \frac{\sum_t \sum_i h_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i^{l+1})(\boldsymbol{x}^t - \boldsymbol{m}_i^{l+1})^T}{N} \end{aligned}$$

   Another way we can see this is by considering that

$$\mathbf{S} = \sum_i P(G_i)\mathbf{S}_i = \sum_i \left(\frac{\sum_t h_i^t}{N} \right) \mathbf{S}_i$$

   In the case of a shared diagonal matrix, for the E-step we have

$$h_i^t = \frac{\pi_i \exp\left[-(1/2s^2)\|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2\right]}{\sum_j \pi_j \exp\left[-(1/2s^2)\|\boldsymbol{x}^t - \boldsymbol{m}_j\|^2\right]}$$

   and in the M-step, we have

$$\min_{\boldsymbol{m}_i, s} \sum_t \sum_i h_i^t \frac{\|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2}{s^2}$$

   The update equation for the shared variance is

$$s^2 = \frac{\sum_t \sum_i \sum_{k=1}^d h_i^t (x_k^t - m_{ik})^2}{Nd}$$

4. *Define a multivariate Bernoulli mixture where inputs are binary and derive the EM equations.*

   When the components are multivariate Bernouilli, we have d-dimensional binary vectors. Assuming that the dimensions are independent, we have (section 5.7)

$$p_i(\boldsymbol{x}^t|\Phi) = \prod_{j=1}^{d} p_{ij}^{x_j^t}(1-p_{ij})^{1-x_j^t}$$

   where $\Phi^l = \{p_{i1}^l, p_{i2}^l, \ldots, p_{id}^l\}_{i=1}^k$. The E-step does not change (equation 7.9). In the M-step, for the component parameters $p_{ij}, i = 1, \ldots, k, j = 1, \ldots, d$, we maximize

$$\begin{aligned}\mathcal{Q}' &= \sum_t \sum_i h_i^t \log p_i(\boldsymbol{x}^t|\phi^l) \\ &= \sum_t \sum_i h_i^t \sum_j x_j^t \log p_{ij}^l + (1-x_j^t)\log(1-p_{ij}^l)\end{aligned}$$

   Taking the derivative with respect to p_{ij} and setting equal to 0, we get

$$p_{ij}^{l+1} = \frac{\sum_t h_i^t x_j^t}{\sum_t h_j^t}$$

   Note that this is the same as equation 5.31 except for estimated "soft" labels h_i^t replacing the supervised labels r_i^t.

8 *Nonparametric Methods*

1. *Show equation 8.17.*

   Given that

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

   we can write

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

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

   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 has 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 (that is 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.

3. *In a regressogram, instead of averaging in a bin and doing a constant fit, one can use the instances falling in a bin and do a linear fit (see figure 8.11). Write the code and compare this with the regressogram proper.*

   The Matlab code is given in `ex8_3.m` which is the code used to generate figure 8.11 of the book. Compared with the regressogram proper, it is

more costly both in terms of space and computation: We need one more parameter (slope) for each bin; we need computation during test to calculate the output; we need more computation during training to calculate the linear fit in each bin.

4. *Write the error function for loess discussed in section 8.6.3.*

   The output is calculated using a linear model

   $$g(x) = ax + b$$

   where, in the running line smoother, a, b minimize

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

   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 $w(\cdot) \in \{0, 1\}$:

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

5. *Propose an incremental version of the running mean estimator, which, like the condensed nearest neighbor, stores instances only when necessary.*

   Let us say that we are using nearest neighbor regression, that is, given test input x, we find closest $x^t \in \mathcal{X}$ and give its desired output r^t as the output. In such a case, in the condensed nearest neighbor regression, we do not need to store a given pair (x, r) if (x', r') is the closest and $|r' - r| < \theta$ where θ is some tolerance threshold.

   We modify the pseudocode of figure 8.6 (on p. 164 of the book) for condensed nearest neighbor classification as shown in figure 8.1. Instead of checking whether the class labels are the same, we check if the outputs are close enough.

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

Figure 8.1 Condensed nearest neighbor regression algorithm.

This can be generalized to any nonparametric smoother: We check if the estimate using current $\mathcal{Z}$ gives an acceptable output (within a certain tolerance), if not, we add the current pair to $\mathcal{Z}$. Again (see exercise 8.2 above), it may be necessary to store the current pair multiple times.

6. *Generalize kernel smoother to multivariate data.*

   We can just use a multivariate kernel, for example, d-variate Gaussian, in equation 8.21. Again, we have the usual concerns of the suitability of using Euclidean distance vs. estimating a local covariance and using Mahalanobis distance.

9 *Decision Trees*

1. *Generalize the Gini index (equation 9.5) and the misclassification error (equation 9.6) for $K > 2$ classes. Generalize misclassification error to risk, taking a loss function into account.*

   - Gini index with $K > 2$ classes: $\phi(p_1, p_2, \ldots, p_K) = \sum_{i=1}^{K} \sum_{j<i} p_i p_j$
   - Misclassification error: $\phi(p_1, p_2, \ldots, p_K) = 1 - \max_{i=1}^{K} p_i$
   - Risk: $\phi_\Lambda(p_1, p_2, \ldots, p_K) = \min_{i=1}^{K} \sum_{k=1}^{K} \lambda_{ik} p_k$ where Λ is the $K \times K$ loss matrix (equation 3.7).

2. *For a numeric input, instead of a binary split, one can use a ternary split with two thresholds and three branches as*

   $$x_j < w_{ma},\ w_{ma} \leq x_j < w_{mb},\ x_j \geq w_{mb}$$

   Propose a modification of the tree induction method to learn the two thresholds, w_{ma}, w_{mb}. What are the advantages and the disadvantages of such a node over a binary node?

   For the numeric attributes, in the pseudocode given in figure 9.3 of the book, instead of one split threshold, we need to try all possible pairs of splits and choose the best; in calculating the entropy after the split, we need to sum up over the three sets corresponding to the instances taking the three branches.

   The complexity of finding the best pair is $\mathcal{O}(N_m^2)$ instead of $\mathcal{O}(N_m)$ and each node stores two thresholds instead of one and has three branches instead of two. The advantage is that one ternary node splits an input into three, whereas this requires two successive binary nodes. Which one is better depends on the data at hand; if we have hypotheses that

require bounded intervals (for example, rectangles), a ternary node may be advantageous.

3. *Propose a tree induction algorithm with backtracking.*

   The decision tree induction algorithm is greedy and stops when there is no further decrease in entropy. One possibility is to backtrack: Let us say that we have a tree where no matter now we add a node, entropy does not decrease further; we can go back, undo the last decision, choose the next best split and continue, to see whether we get to a better state. In fact, the decision tree induction does depth-first search and any other search mechanism, best-first search, breadth-first search, etc. can be used which can search wider in the space (of possible trees) at the expense of more space and computation.

4. *In generating a univariate tree, a discrete attribute with n possible values can be represented by n 0/1 dummy variables and then treated as n separate numeric attributes. What are the advantages and disadvantages of this approach?*

   If we do this, we get binary splits instead of n-ary splits. For example, if we have three values {red,green,blue}, a discrete node has a ternary split whereas doing this, we can have a split such as {red, not-red}, thereby grouping all the complements of a value in the other branch. Whether this leads to simpler trees or not depends on the data.

5. *Derive a learning algorithm for sphere trees (equation 9.21). Generalize to ellipsoid trees.*

   A sphere node uses a binary decision node (see figure 9.1):

   $$f_m(\mathbf{x}) : \|\mathbf{x} - \mathbf{c}_m\| \leq \alpha_m$$

   and a simple algorithm is as follows: Calculate the centroid of instances reaching node m and take it as $\mathbf{c}_m$. Then sort all instances according to their (Euclidean) distance to the center and for all possible $N_m - 1$ values between them, check what the entropy would be and choose the best as the radius α_m. In an ellipsoid tree, we calculate the covariance matrix of the instances and use the Mahalanobis distance from an instance to the center.

6. *In a regression tree, we discussed that in a leaf node, instead of calculating the mean, we can do a linear regression fit and make the response*

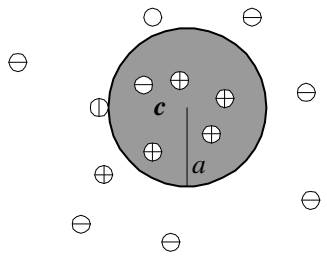


Figure 9.1 A sphere decision node.

at the leaf dependent on the input. Propose a similar method for classification trees.

This implies that at each leaf, we will have a linear classifier trained with instances reaching there. That linear classifier will generate posterior probabilities for the different classes and those probabilities will be used in the entropy calculation. That is, it is not necessary that a leaf is pure, that is, contains instances of only one class; it is enough that the classifier in it generates posterior probabilities close to 0 or 1.

7. *Propose a rule induction algorithm for regression.*

   Rule induction is similar to tree induction except that at each split, we keep one of the branches (better one) and ignore the other, going depth-first and generating a single path, instead of a tree. Each node corresponds to one condition, the path corresponds to their conjunction and the leaf is the average of their outputs. We would like to find a subset which is as large as possible and on which we have small estimation error (MSE in regression); we can combine these two criteria of accuracy and complexity using some MDL measure. Once we get a rule, we remove the subset it covers from the training set. We then run the same algorithm again to get another rule that covers another subset, until we cover all the training set.

10 *Linear Discrimination*

1. *For each of the following basis function, describe where it is nonzero:*

   a. $\sin(x_1)$

      This is used when a trigonometric mapping is necessary, for example, in robot arm kinematics, or in recurring phenomena, for example, seasonal repeating behavior in time-series.

   b. $\exp(-(x_1 - a)^2/c)$

      This is a bell-shaped function with a as its center and c its spread. Roughly, speaking it is nonzero between $(a - 2\sqrt{c}, a + 2\sqrt{c})$.

   c. $\exp(-\|\boldsymbol{x} - \boldsymbol{a}\|^2/c)$

      This is a d-dimensional bell-shaped function with $\boldsymbol{a}$ as its center and c its spread in d dimensions.

   d. $\log(x_2)$

      This is useful when x_2 has a wide scale and a log transformation is useful.

   e. $1(x_1 > c)$

      This is similar to a univariate split of a decision tree node.

   f. $1(ax_1 + bx_2 > c)$

      This defines a multivariate oblique split.

2. *For the two-dimensional case of figure 10.2, show equations 10.4 and 10.5.*

   Given figure 10.1, first let us take input $\boldsymbol{x}_0$ on the hyperplane. The angle between $\boldsymbol{x}_0$ and $\boldsymbol{w}$ is a_0 and because it is on the hyperplane

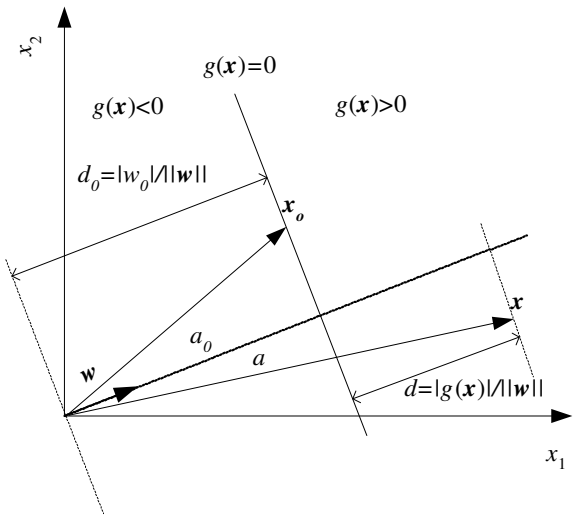


Figure 10.1 The geometric interpretation of the linear discriminant.

$g(\boldsymbol{x}_0) = 0$. Then

$$\begin{aligned} g(\boldsymbol{x}_0) &= \boldsymbol{w}^T\boldsymbol{x}_0 + w_0 = \|\boldsymbol{w}\|\|\boldsymbol{x}_0\|\cos a_0 + w_0 = 0 \\ d_0 &= \|\boldsymbol{x}_0\|\cos a_0 = \frac{|w_0|}{\|\boldsymbol{w}\|} \end{aligned}$$

For any $\boldsymbol{x}$ with angle a to $\boldsymbol{w}$, similarly we have

$$\begin{aligned} g(\boldsymbol{x}) &= \boldsymbol{w}^T\boldsymbol{x} + w_0 = \|\boldsymbol{w}\|\|\boldsymbol{x}\|\cos a + w_0 \\ d &= \|\boldsymbol{x}\|\cos a - \frac{w_0}{\|\boldsymbol{w}\|} = \frac{g(\boldsymbol{x}) - w_0 + w_0}{\|\boldsymbol{w}\|} = \frac{g(\boldsymbol{x})}{\|\boldsymbol{w}\|} \end{aligned}$$

3. *Show that the derivative of the softmax, $y_i = \exp(a_i)/\sum_j \exp(a_j)$, is $\partial y_i/\partial a_j = y_i(\delta_{ij} - y_j)$, where δ_{ij} is 1 if $i = j$ and 0 otherwise.*

   Given that

$$y_i = \frac{\exp a_i}{\sum_j \exp a_j}$$

   for $i = j$, we have

$$\frac{\partial y_i}{\partial a_i} = \frac{\exp a_i \left(\sum_j \exp a_j\right) - \exp a_i \exp a_j}{\left(\sum_j \exp a_j\right)^2}$$

$$
\begin{aligned}
&= \frac{\exp a_i}{\sum_j \exp a_j}\left(\frac{\sum_j \exp a_j - \exp a_i}{\sum_j \exp a_j}\right) \\
&= y_i(1-y_i)
\end{aligned}
$$

and for $i \neq j$, we have

$$
\begin{aligned}
\frac{\partial y_i}{\partial a_j} &= \frac{-\exp a_i \exp a_j}{\left(\sum_j \exp a_j\right)^2} \\
&= -\left(\frac{\exp a_i}{\sum_j \exp a_j}\right)\left(\frac{\sum_j \exp a_j}{\sum_j \exp a_j}\right) \\
&= y_i(0-y_j)
\end{aligned}
$$

which we can combine in one equation as

$$\frac{\partial y_i}{\partial a_j} = y_i(\delta_{ij} - y_j)$$

4. *With $K = 2$, show that using two softmax outputs is equal to using one sigmoid output.*

$$y_1 = \frac{\exp o_1}{\exp o_1 + \exp o_2} = \frac{1}{1+\exp(o_2 - o_1)} = \frac{1}{1+\exp(-(o_1 - o_2))}$$

So for example if we have $o_1 = \boldsymbol{w}_1^T \boldsymbol{x}$, we have

$$y_1 = \frac{\exp \boldsymbol{w}_1^T \boldsymbol{x}}{\exp \boldsymbol{w}_1^T \boldsymbol{x} + \exp \boldsymbol{w}_2^T \boldsymbol{x}} = \text{sigmoid}((\boldsymbol{w}_1 - \boldsymbol{w}_2)^T \boldsymbol{x})$$

and $y_2 = 1 - y_1$.

5. *How can we learn $\mathbf{W}_i$ in equation 10.34?*

For example if we have two inputs x_1, x_2, we can open up equation 10.34 as

$$
\begin{aligned}
\log \frac{p(x_1, x_2|C_i)}{p(x_1, x_2|C_K)} &= \mathbf{W}_{i11}x_1^2 + \mathbf{W}_{i12}x_1x_2 + \mathbf{W}_{i21}x_2x_1 + \mathbf{W}_{i22}x_2^2 \\
&+ w_{i1}x_1 + w_{i2}x_2 + w_{i0}
\end{aligned}
$$

Then we can use gradient-descent and take derivative with respect to any $\mathbf{W}_{jkl}$ to calculate an update rule (as in equation 10.33):

$$\Delta \mathbf{W}_{jkl} = \eta \sum_t (r_j^t - y_j^t) x_k^t x_l^t$$

11 *Multilayer Perceptrons*

1. *Show the perceptron that calculates NOT of its input.*

$$y = s(-x + 0.5)$$

2. *Show the perceptron that calculates NAND of its two inputs.*

$$y = s(-x_1 - x_2 + 1.5)$$

3. *Show the perceptron that calculates the parity of its three inputs.*

$$\begin{aligned} h_1 &= s(-x_1 - x_2 + 2x_3 - 1.5) \quad 001 \\ h_2 &= s(-x_1 + 2x_2 - x_3 - 1.5) \quad 010 \\ h_3 &= s(2x_1 - x_2 - x_3 - 1.5) \quad 100 \\ h_4 &= s(x_1 + x_2 + x_3 - 2.5) \quad 111 \\ y &= s(h_1 + h_2 + h_3 + h_4 - 0.5) \end{aligned}$$

The four hidden units corresponding to the four cases of (x_1, x_2, x_3) values where the parity is 1, namely, 001, 010, 100, and 111. They are then OR'd to calculate the overall output. Note that another possibility is to calculate the three bit parity is in terms of two-bit parity (XOR) as: (x_1 XOR x_2) XOR x_3.

4. *Derive the update equations when the hidden units use tanh, instead of the sigmoid. Use the fact that* $\tanh' = (1 - \tanh^2)$.

The only difference from the case where hidden units use sigmoid is that the derivative of the hidden unit activations will change. In updating the first layer weights (equation 11.16), instead of $z_h(1 - z_h)$, we will have $(1 - z_h^2)$.

5. *Derive the update equations for an MLP with two hidden layers.*

$$
\begin{aligned}
z_{1h} &= \text{sigmoid}(\boldsymbol{w}_{1h}^T \boldsymbol{x}) = \text{sigmoid}\left(\sum_{j=1}^{d} w_{1hj} x_j + w_{1h0}\right), \; h = 1, \ldots, H_1 \\
z_{2l} &= \text{sigmoid}(\boldsymbol{w}_{2l}^T \boldsymbol{z}_1) = \text{sigmoid}\left(\sum_{h=0}^{H_1} w_{2lh} z_{1h} + w_{2l0}\right), \; l = 1, \ldots, H_2 \\
y_i &= \boldsymbol{v}_i^T \boldsymbol{z}_2 = \sum_{l=1}^{H_2} v_{il} z_{2l} + v_0
\end{aligned}
$$

Let us take the case of regression:

$$E = \frac{1}{2} \sum_t \sum_i (r_i^t - y_i^t)^2$$

We just continue backpropagating, that is continue the chain rule, and we can write error on a layer as a function of the error in the layer after, carrying the supervised error in the output layer to layers before:

$$
\begin{aligned}
err_i &\equiv r_i^t - y_i^t \text{ and } \Delta v_{il} = \eta \sum_t err_i z_{2l} \\
err_{2l} &\equiv \left[\sum_i err_i v_i\right] z_{2l}(1 - z_{2l}) \text{ and } \Delta w_{2lh} = \eta \sum_t err_{2l} z_{1h} \\
err_{1h} &\equiv \left[\sum_l err_{2l} w_{2lh}\right] z_{1h}(1 - z_{1h}) \text{ and } \Delta w_{hj} = \eta \sum_t err_{1h} x_j
\end{aligned}
$$

6. *Parity is cyclic shift invariant, for example, "0101" and "1010" have the same parity. Propose a multilayer perceptron to learn the parity function using this hint.*

   One can generate virtual examples by adding shifted versions of instances to the training set. Or, one can define local hidden units with weight sharing to keep track of local parity which are then combined to calculate the overall parity.

7. *In cascade correlation, what are the advantages of freezing the previously existing weights?*

   The main advantage is that it allows us to train a single layer at each step which is faster than training multiple hidden layers. Keeping all

but one of the set of parameters fixed is similar to *backfitting algorithms*; we go over a set of additive models, while keeping all the others fixed, we update the parameters of one model to minimize the residual from the other models. This makes sense in cascade correlation because each new unit is adde to learn what has not yet been learned by the previous layer(s).

8. *Derive the update equations for an MLP implementing Sammon mapping that minimizes Sammon stress (equation 11.40).*

$$\begin{aligned} E(\theta|\mathcal{X}) &= \sum_{r,s}\left[\frac{\|\boldsymbol{g}(\boldsymbol{x}^r|\theta)-\boldsymbol{g}(\boldsymbol{x}^s|\theta)\|-\|\boldsymbol{x}^r-\boldsymbol{x}^s\|}{\|\boldsymbol{x}^r-\boldsymbol{x}^s\|}\right]^2 \\ &= \frac{1}{\sum_{r,s}\|\boldsymbol{x}^r-\boldsymbol{x}^s\|^2}\sum_{r,s}\left(\|\boldsymbol{g}(\boldsymbol{x}^r|\theta)-\boldsymbol{g}(\boldsymbol{x}^s|\theta)\|-\|\boldsymbol{x}^r-\boldsymbol{x}^s\|\right)^2 \end{aligned}$$

The MLP is written as

$$g(x|\boldsymbol{v},\boldsymbol{W}) = \sum_h v_h z_h + v_0 \text{ and } z_h = \text{sigmoid}\left(\sum_j w_{hj}x_j + w_{h0}\right)$$

The update rules are found using gradient-descent:

$$\begin{aligned} \Delta v_h &= -\eta\frac{1}{\sum_{r,s}\|\boldsymbol{x}^r-\boldsymbol{x}^s\|^2}\sum_{r,s}\left(\|\boldsymbol{g}(\boldsymbol{x}^r|\theta)-\boldsymbol{g}(\boldsymbol{x}^s|\theta)\|-\|\boldsymbol{x}^r-\boldsymbol{x}^s\|\right)\left(z_h^r-z_h^s\right) \\ \Delta w_{hj} &= -\eta\frac{1}{\sum_{r,s}\|\boldsymbol{x}^r-\boldsymbol{x}^s\|^2}\sum_{r,s}\left(\|\boldsymbol{g}(\boldsymbol{x}^r|\theta)-\boldsymbol{g}(\boldsymbol{x}^s|\theta)\|-\|\boldsymbol{x}^r-\boldsymbol{x}^s\|\right) \\ &\quad\cdot\; v_h\left[z_h^r(1-z_h^r)x_j^r - z_h^s(1-z_h^s)x_j^s\right] \end{aligned}$$

12 *Local Models*

1. *Show an RBF network that implements XOR.*

   There are two possibilities: (a) We can have two circular Gaussians centered on the two positive instances and the second layer ORs them, (b) We can have one elliptic Gaussian centered on (0.5, 0.5) with negative correlation to cover the two positive instances

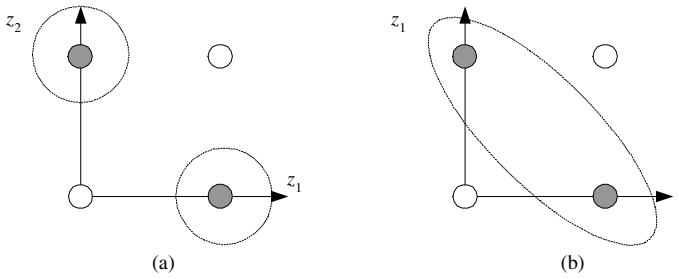


Figure 12.1 Two ways of implementing XOR with RBF.

2. *Derive the update equations for the RBF network for classification (equations 12.20 and 12.21).*

   Because of the use of cross-entropy and softmax, the update equations will be the same with equations 12.17, 12.18, and 12.19 (see equation 10.33 for a similar derivation).

3. *Show how the system given in equation 12.22 can be trained.*

   There are two sets of parameters: $\boldsymbol{v}, v_0$ of the default model and $w_h, \boldsymbol{m}_h, s_h$ of the exceptions. Using gradient-descent and starting from

random values, we can update both iteratively. We update $\boldsymbol{v}, v_0$ as if we are training a linear model and $w_h, \boldsymbol{m}_h, s_h$ as if we are training a RBF network.

Another possibility is to separate their training: First, we train the linear default model and then once it converges, we freeze its weights and calculate the residuals, that is, differences not explained by the default. We train the RBF on these residuals, so that the RBF learns the exceptions, that is, instances not covered by the default "rule." See E. Alpaydın. 1997. *REx: Learning A Rule and Exceptions*, International Computer Science Institute TR-97-040, Berkeley CA.

4. *Compare the number of parameters of a mixture of experts architecture with an RBF network.*

   With d inputs, K classes and H Gaussians, an RBF network needs Hd parameters for the centers, H parameters for the spreads and $(H+1)K$ parameters for the second layer weights. For the case of a MoE, for each second layer weight, we need a $d+1$ dimensional vector of the linear model, but there is no bias, thereby we have $HK(d+1)$ parameters.

   Note that the number of parameters in the first layer is the same with RBF and it is the same whether we have Gaussian or softmax gating: For each hidden unit, in the case of Gaussian gating, we need d parameters for the center and 1 for the spread; in the case of softmax gating, the linear model has $d+1$ parameters (d inputs and a bias).

5. *Formalize a mixture of experts architecture where the experts and the gating network are multilayer perceptrons. Derive the update equations for regression and classification.*

   The output is

$$y_i^t = \sum_{h=1}^{H} w_{ih}^t g_h^t$$

   The experts are MLP with M hidden units:

$$\begin{aligned} w_{ih}^t &= \sum_{k=1}^{M} v_{ihk} b_k^t + v_{ih0} \\ b_k^t &= \text{sigmoid}\left(\sum_{j=1}^{d} u_{kj} x_j^t + u_{k0}\right) \end{aligned}$$

The gating networks are MLP with G hidden units:

$$\begin{aligned} g_h^t &= \frac{\exp o_h^t}{\sum_l \exp o_l^t} \\ o_h^t &= \sum_{k=1}^{G} m_{hk} a_k^t + m_{h0} \\ a_k^t &= \text{sigmoid}\left(\sum_{j=1}^{d} n_{kj} x_j^t + n_{k0}\right) \end{aligned}$$

As usual, we use gradient-descent:

$$\begin{aligned} E(\{v,u,m,n\}|\mathcal{X}) &= \frac{1}{2}\sum_t \sum_i (r_i^t - y_i^t)^2 \\ \Delta v_{ihk} &= \eta \sum_t (r_i^t - y_i^t) g_h^t b_k^t \\ \Delta u_{kj} &= \eta \sum_t \sum_h \sum_i (r_i^t - y_i^t) g_h^t v_{ihk} b_k^t (1 - b_k^t) x_j^t \\ \Delta m_{hk} &= \eta \sum_t \sum_i (r_i^t - y_i^t)(w_{ih}^t - y_i^t) g_h^t a_k^t \\ \Delta n_{kj} &= \eta \sum_t \sum_h \sum_i (r_i^t - y_i^t)(w_{ih}^t - y_i^t) g_h^t m_{hk} a_k^t (1 - a_k^t) x_j^t \end{aligned}$$

In classification, we have

$$\begin{aligned} y_i^t &= \frac{\exp \sum_{h=1}^{H} w_{ih}^t g_h^t}{\sum_k \exp \sum_{h=1}^{H} w_{kh}^t g_h^t} \\ E(\{v,u,m,n\}|\mathcal{X}) &= -\sum_t \sum_i r_i^t \log y_i^t \end{aligned}$$

Update equations turn out to be the same.

6. *Derive the update equations for the cooperative mixture of experts for classification.*

$$\begin{aligned} E &= -\sum_t \sum_i r_i^t \log y_i^t \\ y_i^t &= \frac{\exp \sum_{h=1}^{H} w_{ih}^t g_h^t}{\sum_k \exp \sum_{h=1}^{H} w_{kh}^t g_h^t} \end{aligned}$$

$$
\begin{aligned}
w_{ih}^t &= \boldsymbol{v}_{ih}^T \boldsymbol{x}^t \\
g_h^t &= \frac{\exp \boldsymbol{m}_h^T \boldsymbol{x}^t}{\sum_l \exp \boldsymbol{m}_l^T \boldsymbol{x}^t}
\end{aligned}
$$

We use gradient-descent:

$$
\begin{aligned}
\Delta v_{ih} &= \eta \sum_t (r_i^t - y_i^t) g_h^t b_k^t \\
\Delta \boldsymbol{m}_h &= -\eta \sum_t \sum_i \sum_k \frac{\partial E}{\partial y_i^t} \frac{\partial y_i^t}{\partial g_k^t} \frac{\partial g_k^t}{\partial p_h^t} \frac{\partial p_h^t}{\partial \boldsymbol{m}_h} \\
&= \eta \sum_t \sum_i (r_i^t - y_i^t) \sum_k w_{ik} g_k^t (\delta_{kh} - g_h^t) \boldsymbol{x}^t \\
&= \eta \sum_t \sum_i (r_i^t - y_i^t) \left[\sum_k w_{ik} g_k^t \delta_{kh} - \sum_k w_{ik} g_k^t g_h^t \right] \boldsymbol{x}^t \\
&= \eta \sum_t \sum_i (r_i^t - y_i^t) (w_{ih} - y_i^t) g_h^t \boldsymbol{x}^t
\end{aligned}
$$

remembering that $\delta_{kh} = 0$ for $k \neq h$ and $\sum_k w_{ik} g_k^t = y_i^t$. This is for softmax gating; for the case of Gaussian gating, the last term changes to:

$$
\frac{\partial p_h^t}{\partial \boldsymbol{m}_h} = \frac{(\boldsymbol{x}^t - \boldsymbol{m}_h)}{2 s_h^2}
$$

7. *Derive the update equations for the competitive mixture of experts for classification.*

Let us see regression first:

$$
\begin{aligned}
\mathcal{L} &= \sum_t \log \sum_h g_h^t \exp \left[-\frac{1}{2} \sum_i (r_i^t - y_{ih}^t)^2 \right] \\
y_{ih}^t &= \boldsymbol{v}_{ih}^T \boldsymbol{x}^t \\
g_h^t &= \frac{\exp \boldsymbol{m}_h^T \boldsymbol{x}}{\sum_l \exp \boldsymbol{m}_l^T \boldsymbol{x}}
\end{aligned}
$$

We use gradient-ascent to maximize the log likelihood:

$$
\begin{aligned}
\Delta \boldsymbol{v}_{ih} &= \eta \sum_t \frac{g_h^t \exp \left[-(1/2) \sum_i (r_i^t - y_{ih}^t)^2 \right]}{\sum_l g_l^t \exp \left[-(1/2) \sum_i (r_i^t - y_{il}^t)^2 \right]} (r_i^t - y_{ih}^t) \boldsymbol{x}^t \\
&= \eta \sum_t (r_i^t - y_{ih}^t) f_h^t \boldsymbol{x}^t
\end{aligned}
$$

where

$$f_h^t = \frac{g_h^t \exp\left[-(1/2)\sum_i (r_i^t - y_{ih}^t)^2\right]}{\sum_l g_l^t \exp\left[-(1/2)\sum_i (r_i^t - y_{il}^t)^2\right]}$$

Similarly,

$$\begin{aligned}
\Delta \boldsymbol{m}_h &= \eta \sum_t \left[\sum_l \frac{\exp\left[-(1/2)\sum_i (r_i^t - y_{ih}^t)^2\right]}{\sum_j g_l^t \exp\left[-(1/2)\sum_i (r_i^t - y_{ij}^t)^2\right]} g_l^t (\delta_{lh} - g_h^t) \right] \boldsymbol{x}^t \\
&= \eta \sum_t \left[\sum_l f_l^t (\delta_{lh} - g_h^t) \right] \boldsymbol{x}^t \\
&= \eta \sum_t \left[\sum_l f_l^t \delta_{lh} - \sum_l f_l^t g_h^t \right] \boldsymbol{x}^t \\
&= \eta \sum_t (f_h^t - g_h^t) \boldsymbol{x}^t
\end{aligned}$$

given that $\sum_l f_l^t = 1$.

For the case of classification, the log likelihood is

$$\mathcal{L} = \sum_t \log \sum_h g_h^t \exp\left[\sum_i r_i^t \log y_{ih}^t\right]$$

and we get the same update equations where

$$f_h^t = \frac{g_h^t \exp\left[\sum_i r_i^t \log y_{ih}^t\right]}{\sum_l g_l^t \exp\left[\sum_i r_i^t \log y_{il}^t\right]}$$

8. *Formalize the hierarchical mixture of experts architecture with two levels. Derive the update equations using gradient descent for regression and classification.*

   The following is from (Jordan and Jacobs, 1994). I have only slightly changed the notation to match the notation of the book.

   Let us see the case of regression with a single output: y is the overall output, y_i are the outputs on the first level and y_{ij} are the outputs on the second level, which are leaves for a model with two levels. Similarly, g_i are the gating outputs on the first level and $g_{j|i}$ are the outputs

on the second level: gating value of expert j on the second level given that we have chosen the branch i on the first level:

$$\begin{aligned} y &= \sum_i g_i y_i \\ y_i &= \sum_j g_{j|i} y_{ij} \quad \text{and} \quad g_i = \frac{\exp \boldsymbol{m}_i^T \boldsymbol{x}}{\sum_k \exp \boldsymbol{m}_k^T \boldsymbol{x}} \\ y_{ij} &= \boldsymbol{v}_{ij}^T \boldsymbol{x} \quad \text{and} \quad g_{j|i} = \frac{\exp \boldsymbol{m}_{ij}^T \boldsymbol{x}}{\sum_l \exp \boldsymbol{m}_{il}^T \boldsymbol{x}} \end{aligned}$$

In regression, the error to be minimized is (note that we are using a competitive version here):

$$E = \sum_t \log \sum_i g_i^t \sum_j g_{j|i}^t \exp\left[-\frac{1}{2}(r^t - y_{ij}^t)^2\right]$$

and using gradient-descent, we get the following update equations:

$$\begin{aligned} \Delta \boldsymbol{v}_{ij} &= \eta \sum_t f_i^t f_{j|i}^t (r^t - y^t) \boldsymbol{x}^t \\ \Delta \boldsymbol{m}_i &= \eta \sum_t (f_i^t - g_i^t) \boldsymbol{x}^t \\ \Delta \boldsymbol{m}_{ij} &= \eta \sum_t f_i^t (f_{j|i}^t - g_{j|i}^t) \boldsymbol{x}^t \end{aligned}$$

where we make use of the following posteriors:

$$\begin{aligned} f_i^t &= \frac{g_i^t \sum_j g_{j|i}^t \exp[-(1/2)(r^t - y_{ij}^t)^2]}{\sum_k g_k^t \sum_j g_{j|k}^t \exp[-(1/2)(r^t - y_{kj}^t)^2]} \\ f_{j|i}^t &= \frac{g_{j|i}^t \exp[-(1/2)(r^t - y_{ij}^t)^2]}{\sum_l g_{l|i}^t \exp[-(1/2)(r^t - y_{il}^t)^2]} \\ f_{ij}^t &= \frac{g_i^t g_{j|i}^t \exp[-(1/2)(r^t - y_{ij}^t)^2]}{\sum_k g_k^t \sum_l g_{l|k}^t \exp[-(1/2)(r^t - y_{kl}^t)^2]} \end{aligned}$$

Note how we multiply the gating values on the path starting from the root to a leaf expert.

For the case of classification with $K > 2$ classes, one possibility is to have K separate HMEs as above (having single output experts), whose outputs we softmax to maximize the loglikelihood:

$$\begin{aligned} \mathcal{L} &= \sum_t \log \sum_i g_i^t \sum_j g_{j|i}^t \exp \left[\sum_c r_c^t \log p_c^t \right] \\ p_c^t &= \frac{\exp y_c^t}{\sum_k \exp y_k^t} \end{aligned}$$

where each y_c^t denotes the output of one single-output HME. The more interesting case of a single multiclass HME where experts have K softmax outputs is discussed in S. R. Waterhouse, A. J. Robinson. 1994. "Classification using Hierarchical Mixtures of Experts." *Proceedings of the 1994 IEEE Workshop on Neural Networks for Signal Processing*, pp. 177–186. IEEE Press.

13 *Hidden Markov Models*

1. *Given the observable Markov model with three states, S_1, S_2, S_3, initial probabilities*

   $$\mathbf{\Pi} = [0.5, 0.2, 0.3]^T$$

   and transition probabilities

   $$\mathbf{A} = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$

   generate 100 sequences of 1,000 states.

   The Matlab code is given in `ex13_1.m`.

2. *Using the data generated by the previous exercise, estimate* Π, $\mathbf{A}$ *and compare with the parameters used to generate the data.*

   The Matlab code is given in `ex13_2.m`. Its output is

   $$\begin{aligned} \hat{\mathbf{\Pi}} &= [0.54, 0.20, 0.26]^T \\ \hat{\mathbf{A}} &= \begin{bmatrix} 0.3987 & 0.3005 & 0.3008 \\ 0.2057 & 0.5937 & 0.2005 \\ 0.0998 & 0.1001 & 0.8001 \end{bmatrix} \end{aligned}$$

3. *Formalize a second-order Markov model. What are the parameters? How can we calculate the probability of a given state sequence? How can the parameters be learned for the case of a observable model?*

In a second-order MM, the current state depends on the two previous states:

$$a_{ijk} \equiv P(q_{t+2} = S_k | q_{t+1} = S_j, q_t = S_i)$$

Initial state probability defines the probability of the first state:

$$\pi_i \equiv P(q_1 = S_i)$$

We also need parameters to define the probability of the second state given the first state:

$$\theta_{ij} \equiv P(q_2 = S_j | q_1 = S_i)$$

Given a second-order observable MM with parameters $\lambda = (\Pi, \Theta, \mathbf{A})$, the probability of an observed state sequence is

$$\begin{aligned} P(O = Q|\lambda) &= P(q_1)P(q_2|q_1)\prod_{t=3}^{T} P(q_t|q_{t-1}, q_{t-2}) \\ &= \pi_{q_1}\theta_{q_2q_1}a_{q_3q_2q_1}a_{q_4q_3q_2}\cdots a_{q_Tq_{T-1}q_{T-2}} \end{aligned}$$

The probabilities are estimated as proportions:

$$\begin{aligned} \hat{\pi}_i &= \frac{\sum_k 1(q_1^k = S_i)}{K} \\ \hat{\theta}_{ij} &= \frac{\sum_k 1(q_2^k = S_j \text{ and } q_1^k = S_i)}{\sum_k 1(q_1^k = S_i)} \\ \hat{a}_{ijk} &= \frac{\sum_k \sum_{t=3}^{T} 1(q_t^k = S_k \text{ and } q_{t-1}^k = S_j \text{ and } q_{t-2}^k = S_i)}{\sum_k \sum_{t=3}^{T} 1(q_{t-1}^k = S_j \text{ and } q_{t-2}^k = S_i)} \end{aligned}$$

4. *Show that any second- (or higher-order) Markov model can be converted to a first-order Markov model.*

   In a second-order model, each state depends on the two previous states. We can define a new set of states corresponding to the Cartesian product of the original set of states with itself. A first-order model on this new N^2 states corrresponds to a second-order model on the original N states.

5. *Some researchers define a Markov model as generating an observation while traversing an arc, instead of on arrival to a state. Is this model any more powerful than what we have discussed?*

   Similar to the case of the previous exercise, if the output depends not only on the current state but also on the next state, we can define new states corresponding to this pair and have the output generated by this (joint) state.

6. *Generate training and validation sequences from an HMM of your choosing. Then train different HMMs by varying the number of hidden states on the same training set and calculate the validation likelihoods. Observe how the validation likelihood changes as the number of states increases.*

   `ex13_6a.m` samples from an HMM, and `ex13_6b.m` learns the parameters given a sample.

14 Assessing and Comparing Classification Algorithms

1. *We can simulate a classifier with error probability p by drawing samples from a Bernoulli distribution. Doing this, implement the binomial, approximate, and t tests for $p_0 \in (0, 1)$. Repeat these tests at least 1,000 times for several values of p and calculate the probability of rejecting the null hypothesis. What do you expect the probability of reject to be when $p_0 = p$?*

   The Matlab code is given in `ex14_1.m`. When $p_0 = p$, we expect the probability of reject to be equal to α, maximum allowed type I probability (see figure 14.1).

2. *Assume $x^t \sim \mathcal{N}(\mu, \sigma^2)$ where σ^2 is known. How can we test for $H_0 : \mu \geq \mu_0$ vs. $H_1 : \mu < \mu_0$?*

   Under H_0

   $$z = \frac{\sqrt{N}(m - \mu_0)}{\sigma} \sim \mathcal{Z}$$

   We accept H_0 if $z \in (-z_\alpha, \infty)$.

3. *The K-fold cross-validated t test only tests for the equality of error rates. If the test rejects, we do not know which classification algorithm has the lower error rate. How can we test whether the first classification algorithm does not have higher error rate than the second one? Hint: We have to test $H_0 : \mu \leq 0$ vs. $H_1 : \mu > 0$.*

   Using the notation of section 14.7.2, under $H_0 : \mu \leq 0$ vs. $H_1 : \mu > 0$

   $$t = \frac{\sqrt{k}(m - 0)}{s} \sim t_{K-1}$$

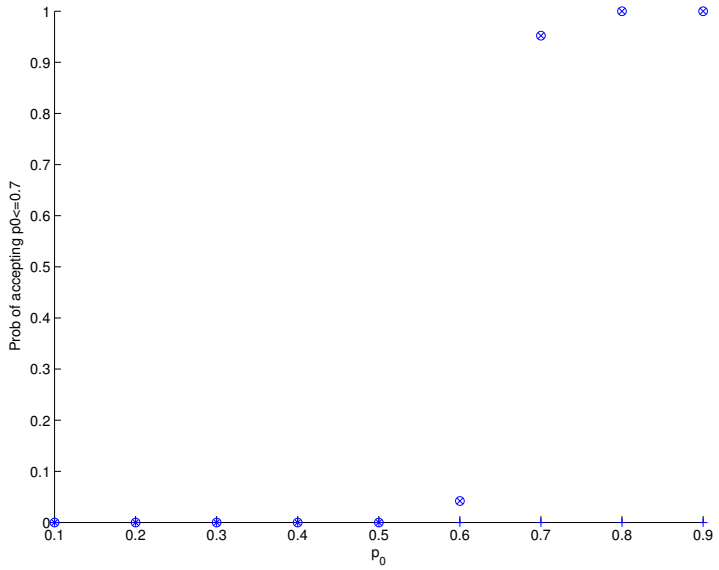


Figure 14.1 Probability of acceptance of the three tests of $H_0 : p \leq 0..7$ for different values of p ('+': Binomial test, 'x': Approximate normal test, 'o': t test; the three tests agree for $N = 250$ and 500 iterations). Note that for $p = 0.7$, the probability of acceptance is 0.95, which is expected because $\alpha = 0.05$.

We accept H_0 if $t \in (-\infty, t_{\alpha,K-1})$. If we accept, we cannot say anything but if we reject, we can say that the second algorithm has less error rate.

4. *Let us say we have three classification algorithms. How can we order these three from best to worst?*

   Given three algorithms, if we have a linear ordering, such as 2 has significantly less error than 1 and 3, 1 has significantly less error than 3, then we have a perfect order: $2 < 1 < 3$. Unfortunately, this is rarely the case; frequently we have one algorithm whose error rate is significantly different from the other two whereas there is no significant difference between the two. This implies that to be able to define an order, we need an additional criterion, and the best candidate for this is complexity (time and/or space). That is we order algorithms in terms of error rates by looking at the results of pairwise tests and two algorithms whose error rates are not statistically significantly different

are ordered in terms of complexity. So let us say 1 has significantly less error than 2 and 3, there is no difference between 2 and 3 and 3 is simpler than 2, then we have the ordering: $1 < 3 < 2$.

For a generalization of this idea to an arbitrary number of algorithms, see O. T. Yıldız, E. Alpaydın. 2006. "Ordering and Finding the Best of $K > 2$ Supervised Learning Algorithms." *IEEE Transactions on Pattern Analysis and Machine Inteligence* 28: 392–402.

15 *Combining Multiple Learners*

1. *If each base-learner is iid and correct with probability $p > 1/2$, what is the probability that a majority vote over L classifiers gives the correct answer?*

$$P(X \geq \lfloor L/2 \rfloor + 1) = \sum_{i=\lfloor L/2 \rfloor + 1}^{L} \left(\begin{array}{c} L \\ i \end{array} \right) p^i (1-p)^{L-i}$$

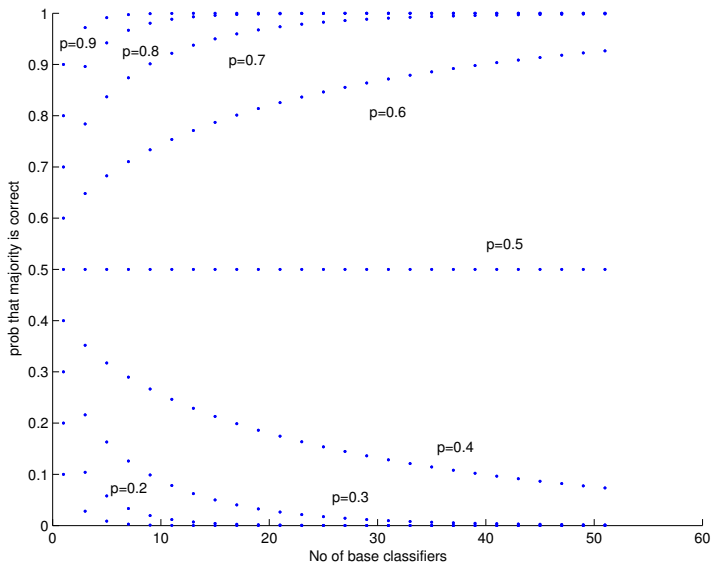


Figure 15.1 Probability that a majority vote is correct as a function of the number of base-learners for different p. The probabity increases only for $p > 0.5$.

2. *In bagging, to generate the L training sets, what would be the effect of using L-fold cross-validation instead of bootstrap?*

   With L-fold cross-validation, each training set has $L-1$ parts of the original set and different sets share $L-2$ parts. For example, for $L = 10$, each training set contains 90 percent of the dataset whereas this percentage is 63.2 percent for bagging. Different training sets share 80 percent of the data which is higher than the percentage of data shared by two bootstrap samples.

3. *Propose an incremental algorithm for learning error-correcting output codes where new two-class problems are added as they are needed to better solve the multiclass problem.*

   At each iteration, we look at the confusion matrix and find the pair of classes that are most confused. We then generate a new dichotomizer, that is, add a new column to the ecoc matrix where one of the most confused classes is labeled as $+1$, the other is labeled as -1 and all other classes are labeled as 0. We continue adding until we overfit over a validation set.

4. *What is the difference between voting and stacking using a linear perceptron as the combiner function?*

   If the voting system is also trained, the only difference would be that with stacking, the weights need not be positive or sum up to 1, and there is also a bias term. Of course, the main advantage of stacking is when the combiner is nonlinear.

5. *In cascading, why do we require $\theta_{j+1} \geq \theta_j$?*

   Instances on which the confidence is less than θ_j have already been filtered out by d_j; we require the threshold to increase so that we can have higher confidences.

16 *Reinforcement Learning*

1. *Given the grid world in figure 16.10, if the reward on reaching on the goal is 100 and* $\gamma = 0.9$*, calculate manually* $Q^*(s,a)$*,* $V^*(S)$*, and the actions of optimal policy.*

   The Matlab code given in `ex16_1.m` can be used.

2. *With the same configuration given in exercise 1, use Q learning to learn the optimal policy.*

   The code is given in `ex16_1.m`.

3. *In exercise 1, how does the optimal policy change if another goal state is added to the lower-right corner? What happens if a state of reward* -100 *(a very bad state) is defined in the lower-right corner?*

   The Matlab code given in `ex16_3.m` has both a goal state and a bad state with negative reward. We see that the optimal policy navigates around the bad state.

4. *Instead of having* $\gamma < 1$*, we can have* $\gamma = 1$ *but with a negative reward of* $-c$ *for all intermediate (nongoal) states. What is the difference?*

   The Matlab code is given in `ex16_4.m` where $\gamma = 1$ and $c = -1$. The Q values converge to reward minus the number of steps to the goal.

5. *In exercise 1, assume that the reward on arrival to the goal state is normal distributed with mean 100 and variance 40. Assume also that the actions are also stochastic in that when the robot advances in a direction, it moves in the intended direction with probability 0.5 and there is a 0.25 probability that it moves in one of the lateral directions. Learn* $Q(s,a)$ *in this case.*

   The Matlab code is given in `ex16_5.m`.

6. *Assume we are estimating the value function for states $V(s)$ and that we want to use TD(λ) algorithm. Derive the tabular value iteration update.*

   The temporal error at time t is

   $$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

   All state values are updated as

   $$V(s) \leftarrow V(s) + \eta \delta_t e_t(s), \ \forall s$$

   where the eligibility of states decay in time:

   $$e_t(s) = \begin{cases} 1 & \text{if } s = s_t \\ \gamma \lambda e_{t-1}(s) & \text{otherwise} \end{cases}$$

7. *Using equation 16.22, derive the weight update equations when a multilayer perceptron is used to estimate Q.*

   Let us say for simplicity we have one-dimensional state value s_t and one-dimensional action value a_t and let us assume a linear model:

   $$Q(s,a) = w_1 s + w_2 a + w_3$$

   We can update the three parameters w_1, w_2, w_3 using gradient-descent (equation 16.21):

   $$\begin{aligned} \Delta w_1 &= \eta \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] s_t \\ \Delta w_2 &= \eta \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] a_t \\ \Delta w_3 &= \eta \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \end{aligned}$$

   In the case of a multilayer perceptron, only the last term will differ to update the weights on all layers.

   In the case of Sarsa(λ), $\boldsymbol{e}$ is three-dimensional: e^1 for w_1, e^2 for w_2, and e^3 for w_0. We update the eligibilities (equation 16.23):

   $$\begin{aligned} e_t^1 &= \gamma \lambda e_{t-1}^1 + s_t \\ e_t^2 &= \gamma \lambda e_{t-1}^2 + a_t \\ e_t^3 &= \gamma \lambda e_{t-1}^3 \end{aligned}$$

and we update the weights using the eligibilities (equation 16.22):

$$\begin{aligned}
\Delta w_1 &= \eta \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] e_t^1 \\
\Delta w_2 &= \eta \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] e_t^2 \\
\Delta w_3 &= \eta \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] e_t^3
\end{aligned}$$

8. *Give an example of a reinforcement learning application that can be modeled by a POMDP. Define the states, actions, observations, and reward.*

   An example is a soccer playing robot. A robot has sensors, typically a camera, but cannot fully observe its state, that is, its exact position in the field. Using the camera, the robot can see only a part of the field, maybe the goals, the boundaries or beacons, but this is not enough to pinpoint its location accurately. It has actuators for action, to move in the field. The goal is to win the game.