Introduction to Learning and Data

Hiroshi Shimodaira[*]

January-March 2017

Informatics 2B consists of two threads.

1. **Algorithms and Data Structures**: how to efficiently store data and efficient algorithms for basic tasks such as sorting and searching. (Kyriakos Kalorkoti — KK)
2. **Learning and Data**: building models to describe a data set and making predictions about new data. (Hiroshi Shimodaira)

Prerequisites You should have taken Informatics 1; in particular Inf1 — Data and Analysis is relevant to this thread. We use some Calculus and some Linear Algebra in this course, but we will revise and refresh what is needed as we go along.

Books There is no required textbook, but some good books include:

- Simon Rogers and Mark Girolami, *A First Course in Machine Learning*, 2nd Ed., CRC, 2016.
- Tom Mitchell, *Machine Learning*, McGraw Hill, 1997.
- Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, 2003.

Some more advanced books include:

- Ian H. Witten and Eibe Franke, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, 2005.
- Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.
- David J.C. MacKay, *Information Theory, Inference, and Learning Algorithms*, CUP, 2003.
- R Duda, P Hart and D Stork, *Pattern Classification*, 2nd Ed., Wiley, 2001.
- Kevein P. Murphy, *Machine Learning*, The MIT Press, 2012.
- T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*, 2nd Ed., Springer 2009. (`http://statweb.stanford.edu/~tibs/ElemStatLearn`)

This book demonstrates some of the ideas we cover in this course, applied in a ‘Web 2.0’ setting:

- Toby Segaran, *Programming Collective Intelligence*, O’Reilly, 2007.

[*]©2014-2017 University of Edinburgh. All rights reserved. This note is heavily based on notes inherited from Steve Renals and Iain Murray.

Online resources Wikipedia (`http://en.wikipedia.org/`) has good coverage of several of the topics that we will cover.

1.1 Motivation

Given some data, what might we do with it? In the algorithms and data structures thread you will learn about how to design and analyse algorithms and data structures to *sort* data and to *search* for items in a data set, in a correct, efficient and simple manner. In this thread we are more concerned with algorithms and models that can make *predictions* about new data items, given the data that has already been observed. For example, we may want to:

- assign a new data item to its class (*classification*);
- predict the next item in a sequence (*time series prediction*);
- predict an output given a new input (*regression*).

Léon Bottou (`http://leon.bottou.org/research/largescale`) puts things in an interesting way:

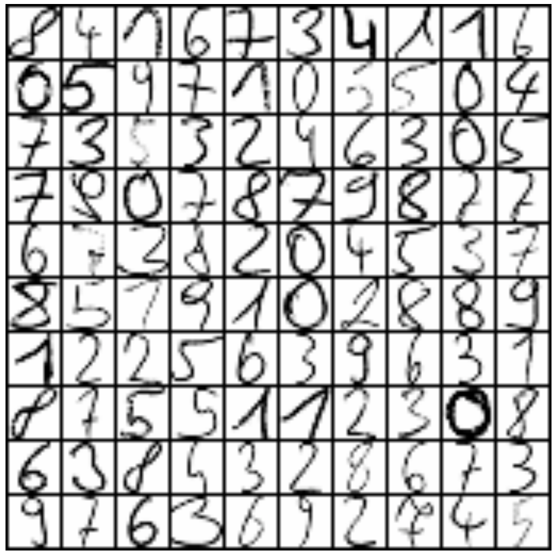
> For the sake of the argument, assume that there are only two categories of computers. The first category, the *makers*, directly mediate human activity. They implement the dialogue between sellers and buyers, run the accounting department, control industrial processes, route telecommunications, etc. The makers generate and collect huge quantities of data. The second category, the *thinkers*, analyse this data, build models, and draw conclusions that direct the activity of the makers.

In this thread we are concerned with the thinkers. Let’s look at some examples.

Handwritten digit recognition Figure 1.1a shows some examples of the digits 0–9, written by people, then scanned and digitised. Each digit *image* is associated with one of the ten digit *classes*. If an image of a new digit is observed, the task is to classify it correctly. To do this, we must decide on a representation for the images, and learn a way of associating such a representation to a class. This is a classification problem.

Load forecasting For the electricity industry, being able to predict the demand for power — as far in advance as possible — is extremely important in order to schedule maintenance, buy fuel as cheaply as possible, minimise overhead, and so on. A typical problem is to predict the hourly electricity demand several days in advance. There are many factors to consider such as weather conditions, time of day, day of the week, holidays, and so on. The problems to be addressed are how the factors are to be represented, and given such a representation how to estimate the load. This is a regression problem.

Predicting currency exchange rates Given a history of currency exchange rates (the exchange rate recorded each day, hour, minute, . . .) how accurately can the next value in the time series be predicted?


(a) Set of digitised characters

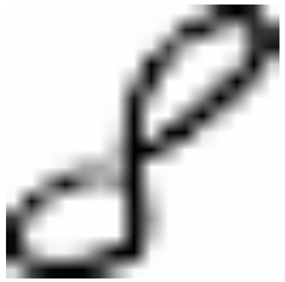

(b) Digitised character ‘8’

Figure 1.1: Handwritten digits, captured by a camera and digitised. This data was collected and distributed by Alex Seewald, `http://alex.seewald.at/digits/`

Recommender systems Given my shopping history, how can an online store recommend items to me that I might want to buy?

Spam Filtering Filter out the unwanted spam mail messages, whilst not losing any of the non-spam messages.

Speech recognition What is the sequence of words that corresponds to an acoustic signal?

Medical diagnosis Given a set of measurements relating to a patient, what condition should be diagnosed?

Robot navigation Given a set of observations about the world, what actions should a robot take to achieve a goal?

1.2 Data

In this thread we are interested in discovering patterns in data, and using those patterns to make predictions or perform classifications. People are naturally good at exploiting patterns, Witten and Frank give several examples: hunters seek patterns in animal migrations, farmers seek patterns in crop growth, politicians look for patterns in the electorate, lovers look for patterns in their partners’

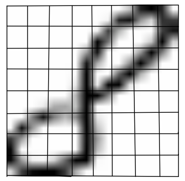


Figure 1.2: Preprocessed digit image on an 8x8 grid

responses to them. Indeed, the process of science involves finding patterns in data, making predictions based on those patterns, and verifying the predictions.

Data comes from many places:

- Sensors such as microphones, cameras, machine monitoring
- Social/economic records such as currency exchange rates, prices
- Artificially created such as possible parses of a sentence, possible object code generated by a compiler

And it needs to be kept in a machine readable electronic form, but it can come in different forms:

- Numeric: numbers such as sound pressure levels obtained from a microphone
- Nominal: categories such as a red, blue and green; or the words in a language
- Ordinal: symbolic data that can be ranked such as grades from “very easy” to “very difficult”

1.2.1 Example: Handwritten digits

As an example let’s look at how we might represent a handwritten digit, such as the digit 8 in Figure 1.1b. First, we need to extract an image of an individual digit, since the raw data is likely to be sequences of characters (e.g., an address on an envelope, a handwritten cheque). This process, called segmentation, is in itself non-trivial! Once the individual digit images are extracted they need to be normalised, for example they need to be scaled to be the same size (imagine that each digit is scaled to fit in a rectangle of a fixed size). These *preprocessing* operations result in a set of segmented and normalised digit images (Figure 1.1a). We will assume that there exists an automatic process that can perform the segmentation and normalisation operations.

Binary representation

Preprocessing removes quite a lot of the variability between images: the location and scale of the images are now the same. Once the images are all fitted to similar sized boxes, the next task is to find a way to represent them in away that is suitable for the process or machine that will associate digit images with the 10 possible classes (0–9). One way to do this is to divide each box into a grid. Figure 1.2 shows the digit image from Figure 1.1b on an 8×8 grid. We could represent each cell in the grid by a binary value: 0 if there is no ink in that location, 1 if there is. This would result in something like the one shown in Figure 1.3 for the example image.

We now have a grid of numbers. How can we represent such a grid? One way is as a 64-bit ($8 \times 8 = 64$)

0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	1
0	0	0	1	1	0	1	0
0	0	0	1	1	1	0	0
0	0	1	1	1	0	0	0
1	1	1	1	0	0	0	0
1	0	0	1	1	0	0	0
1	1	1	1	0	0	0	0

Figure 1.3: Binary representation of the digit image with an 8x8 grid

binary number:
0000011100001001000110100001110000111000111100001001100011110000
This has the advantage of being compact, but treating the bits as part of a single number means that some are more significant than others—changing the first bit has a bigger effect than changing the last bit.

Another way is as a 8-by-8 matrix, e.g. $B = (b_{ij}),\ 1 \le i \le 8,\ 1 \le j \le 8$, so that each element b_{ij} corresponds to the grid (i, j), i.e., the cell at i'th row and j'th column.

$$B = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Binary vector representation

Recognising digit images will always require a way of measuring the similarity between images (we assume that images of the same digit class are in some way similar). Thus we would like a representation that tells us that the above binary grid is similar to this one:

0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	1
0	0	0	1	1	0	1	0
0	0	0	1	1	1	0	0
0	0	0	1	1	0	0	0
0	1	1	1	0	0	0	0
1	0	0	1	1	0	0	0
1	1	1	1	0	0	0	0

A natural way to do this is by converting the corresponding 8-by-8 matrix into a 64-element vector [1]:
$(0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, \dots, 0, 0, 0)^T$

[1]The conversion shown here was done in row-wise, i.e., the first row of the matrix was copied into the vector first, followed by copying the second row, and so on. It could be done in column-wise, of course.

Notation: $(\cdot)^T$ means transpose, allowing us to write a column vector on a single line to save space. Having the two images represented as vectors, **a** and **b** for example, we can now tell how they are similar to each other by calculating the distance (e.g. Euclidean distance) between **a** and **b**.

You will have come across 2- and 3-dimensional vectors when you have studied vector geometry. The concept can be extended to higher dimensions and the mathematics (scalar product, distance, and so on) remains the same. We are representing the images in a high-dimensional space (64 dimensions in this case), where each dimension corresponds to a pixel value.

Real-valued vectors

It was not completely accurate to write that we assigned the binary number to each grid location as "0 if there is no ink in that location, 1 if there is". In fact what we did was to choose a threshold on how much grey colour there was in each location. If a location was above threshold we assigned it value 1, otherwise it was assigned 0.

An alternative to hard thresholding would be to set the value for each location to a real number, between 0 and 1, such that 0 is completely white and 1 is completely black, with numbers in between corresponding to the amount of grey. In this case our vector might look like:
$(0.00, 0.00, 0.00, 0.00, 0.02, 0.55, 0.66, 0.78, 0.00, 0.00, 0.00, 0.01, 0.70, \dots, 0.00, 0.00, 0.00)^T$

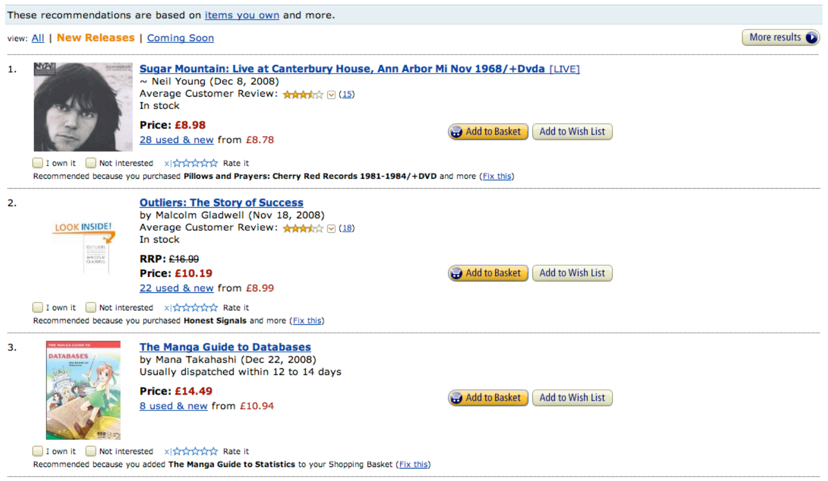
This type of representation is used in many problems, and we will be dealing with such multidimensional vector representations in a lot of this course. They are often referred to as *feature vectors*.

1.2.2 Example: Recommender Systems

You can get recommendations for things like films, games, music by asking your friends and by reading reviews. And you know some of your friends have tastes similar to your own (and some of them don't). But what if there is a new film your friends haven't seen yet? Or if none of your friends have quite the same likes and dislikes as you? A *recommender system* (also called *collaborative filtering*) works by looking at large groups of people and seeing which among them buy or rent or download similar things to you. Such a system can then combines the lists of things that they like, to provide a list of things that you might like.

The film rental company Netflix uses recommender systems to suggest films you might like to watch. They instituted the Netflix Prize (`http://www.netflixprize.com`) to improved the quality of recommender systems. The description provided by Netflix gives a good idea of what recommender systems do, and the framework in which they are constructed:

> Netflix is all about connecting people to the movies they love. To help customers find those movies, we've developed our world-class movie recommendation system: Cinematch. Its job is to predict whether someone will enjoy a movie based on how much they liked or disliked other movies. We use those predictions to make personal movie recommendations based on each customer's unique tastes. And while Cinematch is doing pretty well, it can always be made better.
>
> Now there are a lot of interesting alternative approaches to how Cinematch works that we haven't tried. Some are described in the literature, some aren't. We're curious whether any of these can beat Cinematch by making better predictions. Because, frankly, if there is a much better approach it could make a big difference to our customers and our business.
>
> So, we thought we'd make a contest out of finding the answer. It's "easy" really. We provide you with a lot of anonymous rating data, and a prediction accuracy bar that is 10% better than



These recommendations are based on items you own and more.

view: All | New Releases | Coming Soon

More results

1. Sugar Mountain: Live at Canterbury House, Ann Arbor Mi Nov 1968/+Dvda [LIVE]
~ Neil Young (Dec 8, 2008)
Average Customer Review: (15)
In stock
Price: £8.98
28 used & new from £8.78
Add to Basket | Add to Wish List
I own it | Not interested | Rate it
Recommended because you purchased **Pillows and Prayers: Cherry Red Records 1981-1984/+DVD** and more (Fix this)

2. Outliers: The Story of Success
by Malcolm Gladwell (Nov 18, 2008)
Average Customer Review: (18)
In stock
RRP: ~~£16.99~~
Price: £10.19
22 used & new from £8.99
Add to Basket | Add to Wish List
I own it | Not interested | Rate it
Recommended because you purchased **Honest Signals** and more (Fix this)

3. The Manga Guide to Databases
by Mana Takahashi (Dec 22, 2008)
Usually dispatched within 12 to 14 days
Price: £14.49
8 used & new from £10.94
Add to Basket | Add to Wish List
I own it | Not interested | Rate it
Recommended because you added **The Manga Guide to Statistics** to your Shopping Basket (Fix this)

Figure 1.4: Automatically generated recommendations from `http://www.amazon.co.uk`

> what Cinematch can do on the same training data set. (Accuracy is a measurement of how closely predicted ratings of movies match subsequent actual ratings.)

(The bar was finally beaten in summer 2009, after 3 years of competition.)

What does the data used by a recommender system look like? You can think of it as a large matrix of people (recommenders) vs. items (e.g. films). Each cell (i, j) of the matrix gives the value that person i gives to item j. In the case of Netflix, this is a whole number from 1–5, representing the rating that the person gave to the film (1-star to 5-stars); for a website recommender it might be 1 (liked), 0 (no preference), -1 (didn't like); for an online store it might be 2 (bought), 1 (browsed), 0 (didn't buy).

In many cases this matrix is sparse: most people have not seen most films.

1.3 Learning

We are interested in learning from the data that we have observed, in order to make predictions about new data points. But what do we mean by learning? Chambers' dictionary defines the verb *learn* as follows:

> to be informed; to get to know; to gain knowledge, skill, or ability.

It is not obvious to see how this translates to the computational setting. What does it mean to inform a computer? How can we determine or measure whether a computer has gained a skill? Rather than worrying about such philosophical issues, we will use the following definition, based on that in Tom Mitchell's book:

> *Machine learning is the study of models and algorithms that improve automatically through experience.*

There are still some problems with this definition:

1. What do we mean by *improve*?

2. What is *experience*?

For a system to improve then it must have some kind of purpose, for example classifying digit images to the correct digit class. If the system improves, then it classifies more accurately. Putting it another way, it makes fewer errors. In machine learning we need to be able to measure improvement in terms of a mathematical function that may measure the error, or measure how well the system models the data. The choice of this function, and the way the system is modified to improve the function, is the at the core of machine learning.

By experience we mean the observed data. In the case of digit recognition, experience corresponds to a set of images together with their correct classes. In the case of a film recommender system, it is a set of ratings given to films by a large group of users.

1.4 Notation

The following is the default notation used in the course. Please note that different version may be used sometimes for the ease of readability.

x	A scalar
$\mathbf{x}$	A column vector : $\mathbf{x} = (x_1, x_2, \ldots, x_D)^T$, where D is the dimension of vector
$\mathbf{x}^T$	Transpose of vector $\mathbf{x}$, meaning a row vector if $\mathbf{x}$ is a column vector
x_n or x_d	n'th sample (scalar), or d'th element of vector $\mathbf{x}$
$\mathbf{x}_n$	n'th sample (vector): $\mathbf{x}_n = (x_{n1}, x_{n2}, \ldots, x_{nD})^T$
$\lVert\mathbf{x}\rVert$	Euclidean norm or L^2 norm: $\lVert\mathbf{x}\rVert = \sqrt{\sum_{d=1}^{D} x_d^2} = \sqrt{\mathbf{x}^T\mathbf{x}}$, also known as magnitude
$\lVert\mathbf{x}\rVert_1$	L^1 norm, i.e. $\sum_{d=1}^{D} \lvert x_d\rvert$
$\mathbf{u}\cdot\mathbf{v}$	dot (inner or scalar) product of $\mathbf{u}$ and $\mathbf{v}$, i.e., $\mathbf{u}\cdot\mathbf{v} = \lVert\mathbf{u}\rVert\,\lVert\mathbf{v}\rVert\cos\theta = \mathbf{u}^T\mathbf{v} = \sqrt{\sum_{d=1}^{D} u_d v_d}$
$\mathbf{A}$	A matrix: $\mathbf{A} = (a_{ij})$. If $\mathbf{A}$ is a M-by-N matrix, $\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_N)$
A_{ij}	Element of matrix $\mathbf{A}$ at i'th row and j'th column, i.e. a_{ij}
$\mathbf{I}$ or $\mathbf{I}_d$	Identity matrix of size d by d (ones on diagonal, zeros off)
$\mathbf{A}^T$	Transpose of matrix $\mathbf{A}$, i.e. $\mathbf{A}^T = (a_{ji})$
$\mathbf{A}^{-1}$	Inverse of matrix $\mathbf{A}$, i.e. $\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$
$\lvert\mathbf{A}\rvert$ or $\det(\mathbf{A})$	Determinant of matrix $\mathbf{A}$
$\mathrm{tr}(\mathbf{A})$	Trace of matrix $\mathbf{A}$
$\{\mathbf{x}_n\}_1^N$	A set of samples: $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$
$\bar{x}$	sample mean of x
s^2, s_x^2, or $\mathrm{Var}\,(x)$	sample variance of x
s_{xy}	sample covariance of x and y. NB: $s_{xx} = s_x^2$
μ	population mean
σ^2	population variance
$\exp(x)$	The natural exponential function of x, i.e. e^x
$\ln(x)$	The natural logarithm of x
δ_{ij}	Kronecker delta, i.e. $\delta_{ij} = 1$ if $i = j$, 0 if $i \neq j$

Similarity and recommender systems

Hiroshi Shimodaira[*]

January-March 2017

In this chapter we shall look at how to measure the *similarity* between items. To be precise we'll look at a measure of the *dissimilarity* or *distance* between feature vectors, as well as a direct measurement of similarity. We shall then see how such measures can be used to suggest items in collaborative filtering and recommender systems.

2.1 Distances

We often want to compare two feature vectors, to measure how different (or how similar) they are. We hope that similar patterns will behave in a similar way. For example if we are performing handwriting recognition, a low distance between digit feature vectors (derived from images) might indicate that they should be given the same label. If we are building a recommender system for an online shop, similar user feature vectors (derived from their purchasing or browsing histories) might indicate users with similar tastes. The distance between two items depends on both the representation used by the feature vectors and on the distance measure used.

If the feature vectors are binary (i.e., all elements are 0 or 1) then the *Hamming distance* is a possible distance measure. For real valued vectors, the Euclidean distance is often used: this is familiar from 2- or 3-dimensional geometry, and may also be generalised to higher dimensions.

2.1.1 Hamming distance

The Hamming distance between two binary sequences of equal length is the number of positions for which the corresponding symbols are different. For example the Hamming Distance between 10101010 and 11101001 is 3.

2.1.2 Euclidean distance

The Euclidean distance is already familiar to you from 2- and 3-dimensional geometry. The Euclidean distance $r_2(\mathbf{u}, \mathbf{v})$ between two 2-dimensional vectors $\mathbf{u} = (u_1, u_2)^T$ and $\mathbf{v} = (v_1, v_2)^T$ is given by:

$$r_2(\mathbf{u}, \mathbf{v}) = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2} = \sqrt{\sum_{k=1}^{2} (u_k - v_k)^2} \,. \tag{2.1}$$

[*]©2014-2017 University of Edinburgh. All rights reserved. This note is heavily based on notes inherited from Steve Renals and Iain Murray.

(Superscript T symbolises the transpose operation so we can write a column vector easily within a line, e.g., $\binom{x}{y}$ as $(x, y)^T$.)

Generalising to higher dimensions, the *Euclidean distance* between two D-dimensional vectors $\mathbf{u} = (u_1, u_2, u_3, \ldots, u_D)^T$ and $\mathbf{v} = (v_1, v_2, v_3, \ldots, v_D)^T$ is given by:

$$r_2(\mathbf{u}, \mathbf{v}) = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2 + \cdots + (u_D - v_D)^2} = \sqrt{\sum_{d=1}^{D} (u_d - v_d)^2} \,. \tag{2.2}$$

It is often the case that we are not interested in the precise distances, just in a comparison between distances. For example, we may be interested in finding the closest point (nearest neighbour) to a point in a data set. In this case it is not necessary to take the square root.

Other distance measures are possible for example the Manhattan (or city-block) metric:

$$r_1(\mathbf{u}, \mathbf{v}) = |u_1 - v_1| + |u_2 - v_2| + \cdots + |u_D - v_D| = \sum_{d=1}^{D} |u_d - v_d| \,. \tag{2.3}$$

The notation $|a|$ indicates the absolute value of a. More generally it is possible to use other powers, giving rise to a more general form (known as the p-norm or L^p-norm):

$$r_p(\mathbf{u}, \mathbf{v}) = \left(\sum_{d=1}^{D} |u_d - v_d|^p \right)^{1/p} \,. \tag{2.4}$$

We will be mostly concerned with the familiar Euclidean distance in this course.

2.2 A simple recommender system

Table 2.1 shows the ratings that a few well-known US film critics gave to a small group of movies. We shall use this data to develop a simple recommender system. Unlike a realistic system, in this case every person has rated every film.

We can represent this data as a matrix, whose rows correspond to a particular critic and whose columns correspond to a film, and where the value of each element (c, m) is the score given by critic c to film (movie) m:

$$\begin{pmatrix} 3 & 7 & 4 & 9 & 9 & 7 \\ 7 & 5 & 5 & 3 & 8 & 8 \\ 7 & 5 & 5 & 0 & 8 & 4 \\ 5 & 6 & 8 & 5 & 9 & 8 \\ 5 & 8 & 8 & 8 & 10 & 9 \\ 7 & 7 & 8 & 4 & 7 & 8 \end{pmatrix} .$$

If we want a feature vector per critic then we can just take the rows:

$$\begin{aligned} \mathbf{x}_1 &= (3, 7, 4, 9, 9, 7)^T \\ \mathbf{x}_2 &= (7, 5, 5, 3, 8, 8)^T \\ &\vdots \\ \mathbf{x}_6 &= (7, 7, 8, 4, 7, 8)^T \,, \end{aligned}$$

where $\mathbf{x}_1$ corresponds to David Denby, $\mathbf{x}_2$ corresponds to Todd McCarthy, and so on.

	Australia	*Body of Lies*	*Burn After Reading*	*Hancock*	*Milk*	*Revolutionary Road*
David Denby (New Yorker)	3	7	4	9	9	7
Todd McCarthy (Variety)	7	5	5	3	8	8
Joe Morgenstern (Wall St Journal)	7	5	5	0	8	4
Claudia Puig (USA Today)	5	6	8	5	9	8
Peter Travers (Rolling Stone)	5	8	8	8	10	9
Kenneth Turan (LA Times)	7	7	8	4	7	8

Table 2.1: Ratings given to six movies by six film critics (from `http://www.metacritic.com`).

If the critics have each reviewed a set of M films (movies), then we can imagine each critic defining a point in an M-dimensional space, given by that critic's review scores. The points are hard to visualise in more than three dimensions (three films). Figure 2.1 shows a two dimensional version in which the six critics are placed in a space defined by their reviews of two films. As more users are made available they can be plotted on the chart, according to their ratings for those films. We make the assumption that the closer two people are in this review space, then the more similar are their tastes.

Consider a new user who rates *Hancock* as 2, and *Revolutionary Road* as 7. This user is also shown in the review space in Figure 2.1. Based on these two films, to which critic is the user most similar? It is easy to see from the graph that the closest critic to the user is McCarthy. In this 2-dimensional case Euclidean distance is:

$$r_2(\text{User}, \text{McCarthy}) = \sqrt{(2-3)^2 + (7-8)^2} = \sqrt{2} \approx 1.4$$

We can go ahead and use Equation (2.1) to compute the Euclidean distances between six critics in the 6-dimensional review space:

	Denby	McCarthy	Morgenstern	Puig	Travers	Turan
Denby		7.7	10.6	6.2	5.2	7.9
McCarthy	7.7		5.0	4.4	7.2	3.9
Morgenstern	10.6	5.0		7.5	10.7	6.8
Puig	6.2	4.4	7.5		3.9	3.2
Travers	5.2	7.2	10.7	3.9		5.6
Turan	7.9	3.9	6.8	3.2	5.6	

Thus the two closest critics, based on these films, are Claudia Puig and Kenneth Turan.

Consider a new user who has not seen Hancock, Australia or Milk, but has supplied ratings to the other three films:

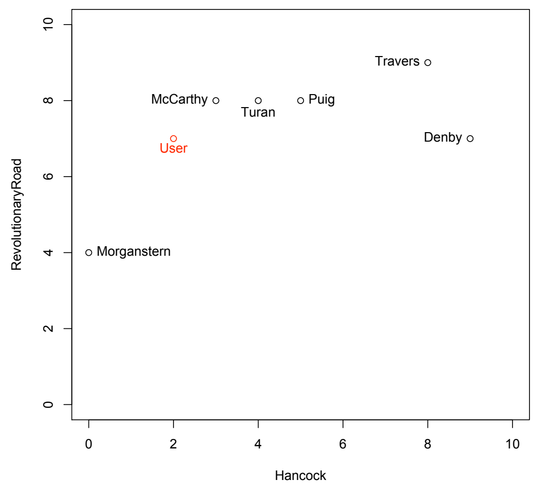


Figure 2.1: Plot of critics in a 2-dimensional review space for *Hancock* and *Revolutionary Road*. For the pair of films under consideration the Euclidean distance between two points provides a measure of how different two reviewers are.

	Body of Lies	*Burn After Reading*	*Revolutionary Road*
User2	6	9	6

Using a 3-dimensional space defined by the films that User2 has rated, we can compute the distances to each critic:

Critic	$r_2(\text{critic}, \text{user2})$
Denby	$\sqrt{27} \approx 5.2$
McCarthy	$\sqrt{21} \approx 4.6$
Morgenstern	$\sqrt{21} \approx 4.6$
Puig	$\sqrt{5} \approx 2.2$
Travers	$\sqrt{14} \approx 3.7$
Turan	$\sqrt{6} \approx 2.4$

Thus, based on these three films, User2 is most similar to Claudia Puig. Can we use this information to build a simple recommender system? Or, more specifically, can we we use this information to decide which film out of *Milk*, *Hancock* and *Australia* the system should recommend to User2 based on their expressed preferences?

We would like to rely on the most similar critics, so we convert our distance measure into a similarity measure:

$$\text{sim}(\mathbf{u}, \mathbf{v}) = \frac{1}{1 + r_2(\mathbf{u}, \mathbf{v})} . \tag{2.5}$$

We have chosen an ad hoc measure of similarity based on Euclidean distance. However, it has some desirable properties: a distance of 0 corresponds to a similarity of 1 (the largest value it can take); a distance of ∞ corresponds to a similarity of 0 (the smallest it can take). We now use this measure to list the critics' similarity to User2:

Critic	sim(critic, user2)
Denby	0.16
McCarthy	0.18
Morgenstern	0.18
Puig	0.31
Travers	0.21
Turan	0.29

One way to make a recommendation for User2 would be to choose the most similar critic, and then to choose the movie that they ranked most highly. This approach has a couple of drawbacks: (1) it does nothing to smooth away any peculiarities of that critic's rankings; and (2) in the case when recommender systems are applied (e.g., online shops) the population of "critics" may be very large indeed, and there may be quite a large number of similar user profiles. An alternative way to make a ranking for User2 would be to weight the rating of each critic by the similarity to User2. An overall score for each film can be obtained by summing these weighted rankings. If u is the user, and we have C critics, then the estimated score given to film m by u, $\mathrm{sc}_u(m)$, is obtained as follows:

$$\mathrm{sc}_u(m) = \frac{1}{\sum_{c=1}^{C} \mathrm{sim}(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_c)} \sum_{c=1}^{C} \mathrm{sim}(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_c) \cdot x_{cm}\,, \qquad (2.6)$$

where $\tilde{\mathbf{x}}_u$ is a vector of ratings for the films seen by u, and $\tilde{\mathbf{x}}_c$ is the vector of corresponding film ratings from critic c. In this example, $\tilde{\mathbf{x}}_u$ and $\tilde{\mathbf{x}}_c$ are 3-dimensional vectors, whereas the original $\mathbf{x}_c$ is a 6-dimensional vector. The term $1/\sum_{c=1}^{C} \mathrm{sim}(\tilde{\mathbf{x}}_u, \tilde{\mathbf{x}}_c)$ is used to normalise the weighted sum of scores to estimate the user's score for the film.

We can compute an estimate of User2's score for each film using Equation (2.6). We'll make the computation explicit in a table:

		Australia		*Hancock*		*Milk*	
	Similarity	Score	Sim · Score	Score	Sim · Score	Score	Sim · Score
Denby	0.16	3	0.48	9	1.44	9	1.44
McCarthy	0.18	7	1.26	3	0.54	8	1.44
Morgenstern	0.18	7	1.26	0	0.00	8	1.44
Puig	0.31	5	1.55	5	1.55	9	2.79
Travers	0.21	5	1.05	8	1.68	10	2.10
Turan	0.29	7	2.03	4	1.16	7	2.03
Total	1.33		7.63		6.37		11.24
Est. Score			5.7		4.8		8.5

So the recommender system would propose *Milk* to User2. But more than just proposing a single film, it provides an estimate of the rating that User2 would provide to each film based on User2's ratings of other films, the estimated similarity of User2 to each critic, and the ratings of the critics to films unseen by User2.

Some questions:

- What do we do if not all the critics have seen the same set of movies? Are the distance/similarity methods between different pairs of people comparable if they are computed across different spaces (i.e., different sets of ratings)? Is there something we can do to make them more comparable?
- How do we deal with the fact that some critics may score more highly on average than others? Or that some critics have a wider spread of scores than others?

We can also solve the transposed problem: instead of measuring the similarity between people, we can measure the similarity between films. In this case we will have a space whose dimension is the number of critics, and each point in the space corresponds to a film. Transposing the previous data matrix:

$$\begin{pmatrix} 3 & 7 & 4 & 9 & 9 & 7 \\ 7 & 5 & 5 & 3 & 8 & 8 \\ 7 & 5 & 5 & 0 & 8 & 4 \\ 5 & 6 & 8 & 5 & 9 & 8 \\ 5 & 8 & 8 & 8 & 10 & 9 \\ 7 & 7 & 8 & 4 & 7 & 8 \end{pmatrix}^T = \begin{pmatrix} 3 & 7 & 7 & 5 & 5 & 7 \\ 7 & 5 & 5 & 6 & 8 & 7 \\ 4 & 5 & 5 & 8 & 8 & 8 \\ 9 & 3 & 0 & 5 & 8 & 4 \\ 9 & 8 & 8 & 9 & 10 & 7 \\ 7 & 8 & 4 & 8 & 9 & 8 \end{pmatrix},$$

each row corresponds to a feature vector for a film:

$$\mathbf{w}_1 = (3, 7, 7, 5, 5, 7)^T$$
$$\vdots$$
$$\mathbf{w}_6 = (7, 8, 4, 8, 9, 8)^T\,,$$

where $\mathbf{w}_1$ corresponds to *Australia*, $\mathbf{w}_2$ corresponds to *Body of Lies*, and so on.

We can then go ahead and compute the distances between films in the space of critics' ratings, in a similar way to before:

	Australia	*Body of Lies*	*Burn After Reading*	*Hancock*	*Milk*	*Revolutionary Road*
Australia		5.8	5.3	10.9	8.9	7.2
Body of Lies	5.8		3.7	6.6	5.9	4.0
Burn After Reading	5.3	3.7		8.9	7.0	4.5
Hancock	10.9	6.6	8.9		10.9	8.4
Milk	8.9	5.9	7.0	10.9		4.8
Revolutionary Road	7.2	4.0	4.5	8.4	4.8	

Therefore if a user, for whom we have no history of ratings chooses *Body of Lies*, then based on our stored critics' ratings we would recommend *Burn After Reading* and *Revolutionary Road* as the two most similar films.

To summarise:

1. We represented the rating data from C critics about M films (movies) as a $C \times M$ matrix.
2. Each row of this data matrix corresponds to a critic's feature vector in "review space".
3. We can compute the distance between feature vectors, to give measure of dissimilarity between critics.

4. We can use this information to construct a simple recommender system.
5. If we take the transpose of the data matrix, then each row corresponds to a film's feature vector; distance measures between these vectors correspond to dissimilarity between films.

2.3 Similarity using correlation

2.3.1 Normalisation

So far our estimate of similarity has been based on the Euclidean distance between feature vectors in a review space. But this distance is not well normalised. For example, two critics may rank a set of films in the same order, but if one critic gives consistently higher scores to all movies than the other, then the Euclidean distance will be large and the estimated similarity will be small. In the data we have been working with (Table 2.1) some critics do give higher scores on average: the mean review ratings per critic range from 4.8 to 8.0.

One way to *normalise* the scores given by each critic is to transform each score into a *standard score*[1]. The standard scores are defined such that the set of scores given by each critic have the same sample mean and sample standard deviation. We first compute the sample mean and sample standard deviation for each critic. Consider an M-dimensional feature vector corresponding to critic c, $\mathbf{x}_c = (x_{c1}, x_{c2}, \ldots, x_{cM})$, where x_{cm} is critic c's rating for movie m. We can compute the sample mean $\bar{x}_c$ and sample standard deviation s_c for critic c as follows[2]:

$$\bar{x}_c = \frac{1}{M}\sum_{m=1}^{M} x_{cm} \tag{2.7}$$

$$s_c = \sqrt{\frac{1}{M-1}\sum_{m=1}^{M}(x_{cm}-\bar{x}_c)^2}\,. \tag{2.8}$$

We then use these statistics to normalise x_{cm} (the critic c's score for movie m) to a standard score,

$$z_{cm} = \frac{x_{cm}-\bar{x}_c}{s_c}\,. \tag{2.9}$$

The z scores for a critic c are normalised with a mean of 0 (obtained by subtracting the mean score from the x_c scores) and a sample standard deviation of 1 (obtained by dividing by the sample standard deviation of the x_c scores). Thus using these normalised scores for each critic removes the offset effect of differing means and the spread effect of differing variances.

2.3.2 Pearson Correlation Coefficient

There are many other ways that we could measure similarity. One measure that has been used a lot in data mining, and collaborative filtering in particular, is a measure based on the *correlation* between

[1]Standard scores are also called *z-values*, *z-scores*, and *normal scores*.

[2]The 'sample standard deviation' is the square-root of the 'sample variance'. Different books give different definitions of the 'sample variance', which is an estimate of the 'true' variance of a population from a limited number of samples. Equation (2.8) uses an 'unbiased' estimate of the true variance. Another version of variance replaces '$M-1$' with 'M', which is normally called a 'population variance', meaning the samples you have got are assumed to be the whole population rather than its subset. This way of calculating variance gives a 'biased' estimate if it is used as an estimate of the true variance of a population. For large sample sizes, as commonly-found in machine learning, the difference usually doesn't matter. Matlab/Octave use the '$M-1$' estimator by default, and Python's `scipy` package uses the 'M' estimator by default, although both have options to use the other.

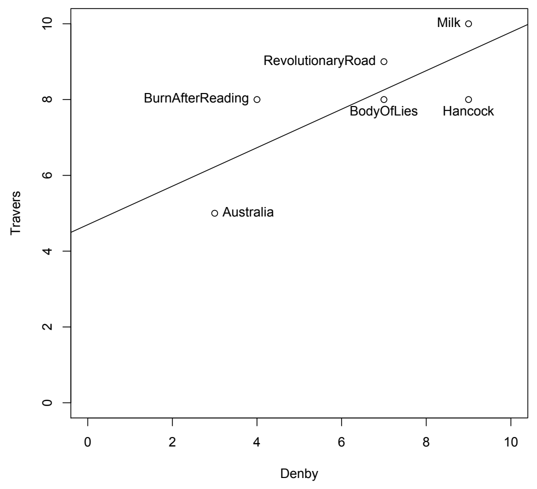


Figure 2.2: Plot of films in a 2-dimensional space defined by the ratings of David Denby and Peter Travers. The best fit straight line for these points is also plotted.

users' ratings. Rather than considering the distance between feature vectors as a way to estimate similarity, we can consider the correlation between the critics scores. Figures 2.2 and 2.3 each plot films in terms of the ratings of two specified critics, along with a best fit straight line. If the ratings of the two critics are closely related (similar) then the best-fit line will (almost) touch every item; if the films are generally far from the best fit line then the review scores are not well associated (dissimilar). We can see that the scores of Travers and Denby (Figure 2.2) are much better correlated than the scores of McCarthy and Denby (Figure 2.3); although Denby has lower ratings on average than that of Travers, this does not affect the correlation.

To estimate the correlation between two sets of scores we use the *Pearson Correlation Coefficient*. To estimate this we first normalise the scores for each critic to stand scores, using Equations (2.7), (2.8) and (2.9). We can then compute the Pearson correlation coefficient between critics c and d, r_{cd} as [3] :

$$r_{cd} = \frac{1}{M-1}\sum_{m=1}^{M} z_{cm}\, z_{dm} \tag{2.10}$$

$$= \frac{1}{M-1}\sum_{m=1}^{M}\left(\frac{x_{cm}-\bar{x}_c}{s_c}\right)\left(\frac{x_{dm}-\bar{x}_d}{s_d}\right). \tag{2.11}$$

If z_{cm} tends to be large when z_{dm} is large and z_{cm} tends to be small when z_{dm} is small, then the correlation coefficient will tend towards 1. If z_{cm} tends to be large when z_{dm} is small and z_{cm} tends to be small when z_{dm} is large, then the correlation coefficient will tend towards -1. If there is no relation between critics c and d, then their correlation coefficient will tend towards 0.

In the above examples, the correlation between Denby and Travers is 0.76 and the correlation between

[3] If we define *sample covariance* as $s_{cd} = \frac{1}{M-1}\sum_{m=1}^{M}(x_{cm}-\bar{x}_c)(x_{dm}-\bar{x}_d)$, Equation (2.11) can be rewritten as $r_{cd} = \frac{s_{cd}}{s_c s_d}$.

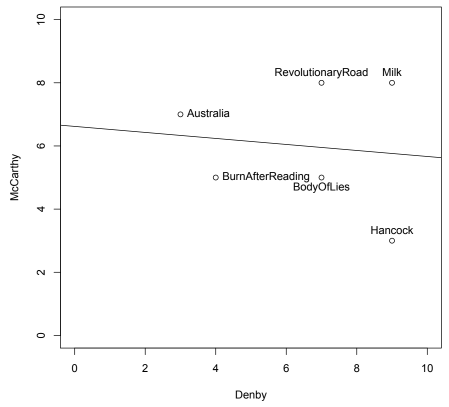


Figure 2.3: Plot of films in a 2-dimensional space defined by the ratings of David Denby and Todd McCarthy. The best fit straight line for these points is also plotted

Denby and McCarthy is −0.12. For comparison the similarities based on Euclidean distance are 0.16 and 0.11.

Exercise: Using the Python function you will write in Tutorial 3, show that the similarities between critics obtained using the correlation coefficient are as below. Compare these similarities with those obtained using the Euclidean distance.

Similarities based on Euclidean distances between critics:

	Denby	McCarthy	Morgenstern	Puig	Travers	Turan
Denby	1	0.11	0.09	0.14	0.16	0.11
McCarthy	0.11	1	0.17	0.19	0.12	0.20
Morgenstern	0.09	0.17	1	0.12	0.09	0.13
Puig	0.14	0.19	0.12	1	0.20	0.24
Travers	0.16	0.12	0.09	0.20	1	0.15
Turan	0.11	0.20	0.13	0.24	0.15	1

Pearson correlation coefficient similarities between critics

	Denby	McCarthy	Morgenstern	Puig	Travers	Turan
Denby	1	-0.12	-0.36	0.14	0.76	-0.55
McCarthy	-0.12	1	0.75	0.53	0.18	0.61
Morgenstern	-0.36	0.75	1	0.44	-0.04	0.64
Puig	0.14	0.53	0.44	1	0.73	0.65
Travers	0.76	0.18	-0.04	0.73	1	0.08
Turan	-0.55	0.61	0.64	0.65	0.08	1

2.4 Reading

Further reading on recommender systems in chapter 2 of Segaran.

Clustering and Visualisation of Data

Hiroshi Shimodaira[*]

January-March 2017

Cluster analysis aims to partition a data set into meaningful or useful groups, based on distances between data points. In some cases the aim of cluster analysis is to obtain greater understanding of the data, and it is hoped that the clusters capture the natural structure of the data. In other cases cluster analysis does not necessarily add to understanding of the data, but enables it to be processed more efficiently.

Cluster analysis can be contrasted with *classification*. Classification is a *supervised* learning process: there is a training set in which each data item has a label. For example, in handwriting recognition the training set may correspond to a set of images of handwritten digits, together with a label ("zero" to "nine") for each digit. In the test set the label for each image is unknown: it is the job of the classifier to predict a label for each test item.

Clustering, on the other hand, is an *unsupervised* procedure in which the training set does not contain any labels. The aim of a clustering algorithm is to group such a data set into clusters, based on the unlabelled data alone. In many situations there is no "true" set of clusters. For example consider the twenty data points shown in Figure 3.1 (a). It is reasonable to divide this set into two clusters (Figure 3.1 (b)), four clusters (Figure 3.1 (c)) or five clusters (Figure 3.1 (d)).

There are many reasons to perform clustering. Most commonly it is done to better understand the data (*data interpretation*), or to efficiently code the data set (*data compression*).

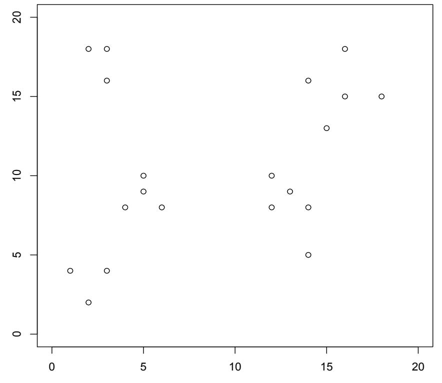
Data interpretation: Automatically dividing a set of data items into groups is an important way to analyse and describe the world. Automatic clustering has been used to cluster documents (such as web pages), user preference data (of the type discussed in the previous chapter), and many forms of scientific observational data in fields ranging from astronomy to psychology to biology.

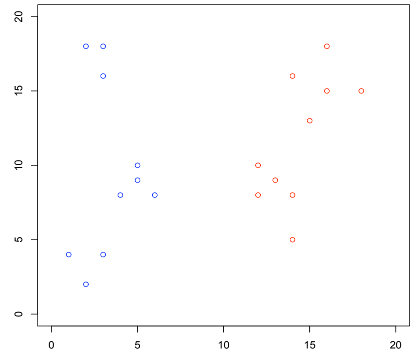
Data compression: Clustering may be used to compress data by representing each data item in a cluster by a single cluster *prototype*, typically at the centre of the cluster. Consider D-dimensional data which has been clustered into K clusters. Rather than representing a data item as a D-dimensional vector, we could store just its cluster index (an integer from 1 to K). This representation, known as *vector canalisation*, reduces the required storage for a large data set at the cost of some information loss. Vector quantisation is used in image, video and audio compression.

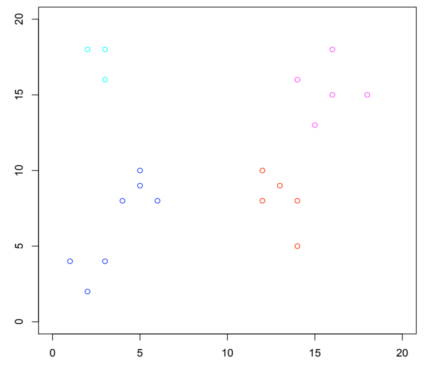
3.1 Types of clustering

There are two main approaches to clustering: hierarchical and partitional. *Hierarchical clustering* forms a tree of nested clusters in which, at each level in the tree, a cluster is the union of its children.

[*]©2014-2017 University of Edinburgh. All rights reserved. This note is heavily based on notes inherited from Steve Renals and Iain Murray.


(a) original data


(b) two clusters


(c) four clusters

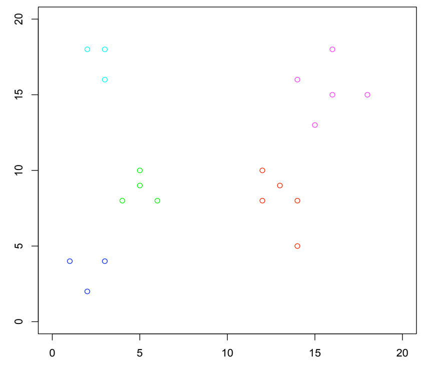

(d) five clusters

Figure 3.1: Clustering a set of 20 two-dimensional data points.

Partitional clustering does not have a nested or hierarchical structure, it simply divides the data set into a fixed number of non-overlapping clusters, with each data point assigned to exactly one cluster. The most commonly employed partitional clustering algorithm, K-means clustering, is discussed below. Whatever approach to clustering is employed, the core operations are distance computations: computing the distance between two data points, between a data point and a cluster prototype, or between two cluster prototypes.

There are two main approaches to hierarchical clustering. In *top-down clustering* algorithms, all the data points are initially collected in a single top-level cluster. This cluster is then split into two (or more) sub-clusters, and each these sub-clusters is further split. The algorithms continues to build a tree structure in a top down fashion, until the leaves of the tree contain individual data points. An alternative approach is *agglomerative hierarchical clustering*, which acts in a bottom-up way. An agglomerative clustering algorithm starts with each data point defining a one-element cluster. Such an algorithm operates by repeatedly merging the two closest clusters until a single cluster is obtained.

3.2 K-means clustering

K-means clustering aims to divide a set of D-dimensional data points into K clusters. The number of clusters, K, must be specified, it is not determined by the clustering: thus it will always attempt to find K clusters in the data, whether they really exist or not.

Each cluster is defined by its cluster centre, and clustering proceeds by assigning each of the input data points to the cluster with the closest centre, using a Euclidean distance metric. The centre of each cluster is then re-estimated as the *centroid* of the points assigned to it. The process is then iterated. The algorithm is:

- *Initialise* K cluster centres, $\{\mathbf{m}_k\}_1^K$
- While not *converged*:
  - Assign each data vector $\mathbf{x}_n$ $(1 \leq n \leq N)$ to the closest cluster centre;
  - Recompute each cluster mean as the mean of the vectors assigned to that cluster

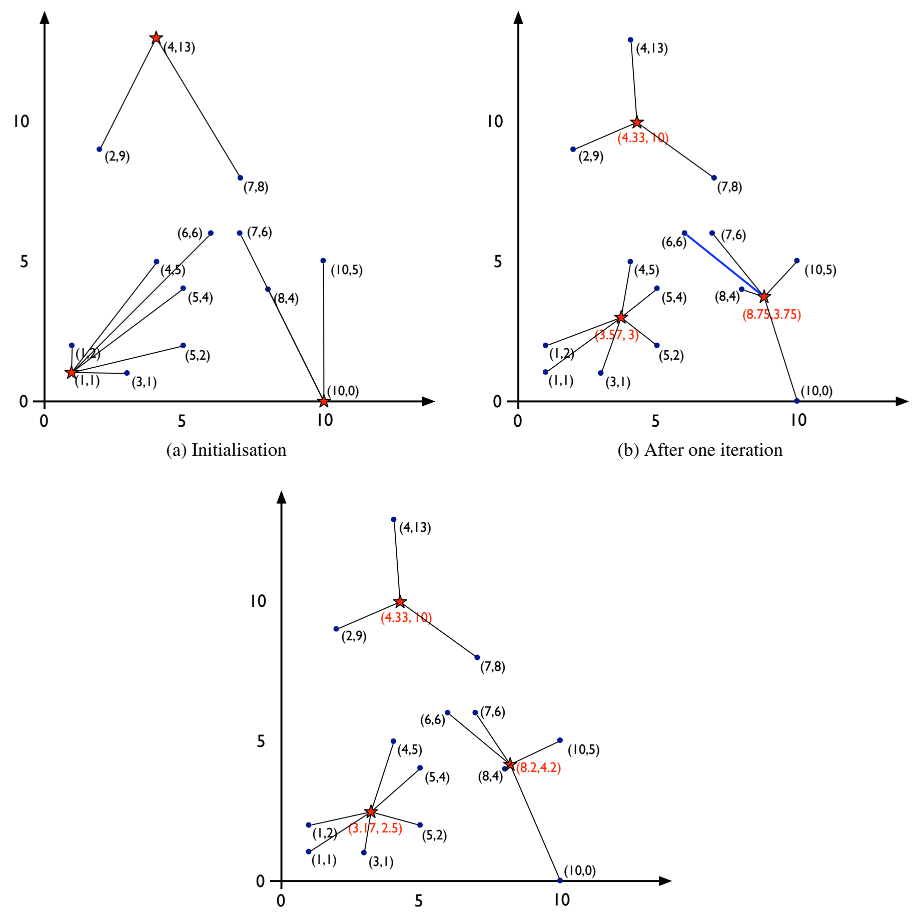
The algorithm requires a distance measure to be defined in the data space, and the Euclidean distance is often used.

The initialisation method needs to be further specified. There are several possible ways to initialise the cluster centres, including:

- Choose random data points as cluster centres
- Randomly assign data points to K clusters and compute means as initial centres
- Choose data points with extreme values
- Find the mean for the whole data set then perturb into K means

All of these work reasonably, and there is no “best” way. However, as discussed below, the initialisation has an effect on the final clustering: different initialisations lead to different cluster solutions.

The algorithm iterates until it converges. Convergence is reached when the assignment of points to clusters does not change after an iteration. An attractive feature of K-means is that convergence is

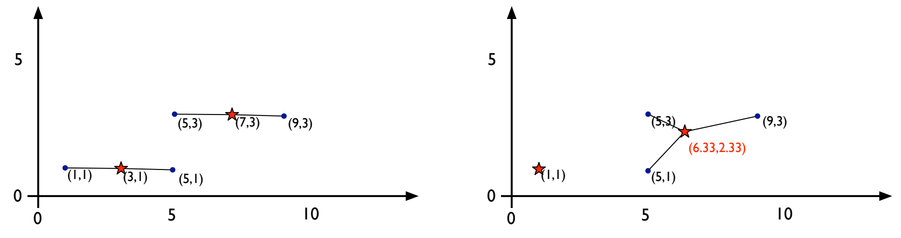


(a) Initialisation

(b) After one iteration

(c) After two iterations

Figure 3.2: Example of K-means algorithm applied to 14 data points, $K = 3$. The lines indicate the distances from each point to the centre of the cluster to which it is assigned. Here only one point (6,6) moves cluster after updating the means. In general, multiple points can be reassigned after each update of the centre positions.



(a) Within-cluster sum-squared error = 4 (b) Within-cluster sum-squared error = 3.33

Figure 3.3: Two different converged clusterings for the same data set, but starting from different initialisations.

guaranteed. However, the number of iterations required to reach convergence is not guaranteed. For large datasets it is often sensible to specify a maximum number of iterations, especially since a good clustering solution is often reached after a few iterations. Figure 3.2 illustrates the K-means clustering process. Figure 3.3 illustrates how different initialisations can lead to different solutions.

3.3 Mean squared error function

K-means clustering is an intuitively sensible algorithm, but is it possible to get a better mathematical description of what is going on? To compare two different clusterings into K clusters we can use the *mean squared error* function, which can be thought of as measuring the *scatter* of the data points relative to their cluster centres. Thus if we have two sets of K clusters, it makes sense to prefer the one with the smallest mean squared error: the clustering with the lowest scatter of data points relative to their cluster centres.

Let us define an indicator variable z_{nk}, such that $z_{nk} = 1$ if the n th data point $\mathbf{x}_n$ belongs to cluster k and $z_{nk} = 0$ otherwise. Then we can write the mean squared error as

$$E = \frac{1}{N} \sum_{k=1}^{K} \sum_{n=1}^{N} z_{nk} \|\mathbf{x}_n - \mathbf{m}_k\|^2, \tag{3.1}$$

where $\mathbf{m}_k$ is the centre of cluster k, N is the number of data points in total, and $\|\cdot\|$ denotes the Euclidean norm (i.e. L^2-norm) of a vector. Another way of viewing this error function is in terms of the squared deviation of the data points belonging to a cluster from the cluster centre, summed over all centres. This may be regarded as a variance measure for each cluster, and hence K-means is sometimes referred to as *minimum variance* clustering. Like all variance-based approaches this criterion is dependent on the scale of the data. In the introduction we said that the aim of clustering was to discover a structure of clusters such that points in the same group are close to each other, and far from points in other clusters. The mean squared error only addresses the first of these: it does not include a between-clusters term.

Depending on the initialisation of the cluster centres, K-means can converge to different solutions; this is illustrated in Figure 3.3. The same data set of 4 points, can have two different clusterings, depending on where the initial cluster centres are placed. Both these solutions are *local minima* of the error

function, but they have different error values. For the solution in Figure 3.3a, the error is:

$$E = \frac{((4+4)+(4+4))}{4} = 4\,.$$

The second solution (Figure 3.3b) has a lower error:

$$E = \frac{0 + (32/9 + 20/9 + 68/9)}{4} = \frac{10}{3} < 4\,.$$

We have discussed the *batch* version of K-means. The online (sample-by-sample) version in which a point is assigned to a cluster and the cluster centre is immediately updated is less vulnerable to local minima.

There are many variants of K-means clustering that have been designed to improve the efficiency and to find lower error solutions.

3.4 Dimensionality reduction and data visualisation

Another way of obtaining better understanding of the data is to visualise it. For example, we could easily see the shapes of data clusters or spot outliers if they are plotted in a two or three dimensional space. It is, however, not straightforward to visualise high-dimensional data.

A simple solution would be to pick up only two components out of D ones and plot them in the same manner as we did for Figure 1 in Note 2, in which we picked up two films, “Hancock” and “Revolutionary Road” out of 6 films to plot critics. We would, however, need to draw more number of plots with different combinations of films to grasp the distribution of data.

A more general way of visualising high-dimensional data is to transform the data to the one in a two-dimensional space. For example, we can apply a linear transformation or mapping using a unit vector $\mathbf{u} = (u_1, \ldots, u_D)^T$ in the original D-dimensional vector space.[1] Calculating a dot-product (Euclidean inner-product) between $\mathbf{u}$ and $\mathbf{x}_n$ gives a scalar y_n:

$$y_n = \mathbf{u} \cdot \mathbf{x}_n = \mathbf{u}^T \mathbf{x}_n = u_1 x_{n1} + \cdots + u_D x_{nD} \tag{3.2}$$

which can be regarded as the orthogonal projection of $\mathbf{x}_n$ on the axis defined by $\mathbf{u}$.

We now consider another unit vector $\mathbf{v}$ that is orthogonal to $\mathbf{u}$, and project $\mathbf{x}_n$ orthogonally on it to get another scalar z_n:

$$z_n = \mathbf{v} \cdot \mathbf{x}_n = \mathbf{v}^T \mathbf{x}_n = v_1 x_{n1} + \cdots + v_D x_{nD}\,. \tag{3.3}$$

You will see that $\mathbf{x}_n$ is mapped to a point $(y_n, z_n)^T$ in a two-dimensional space, and the whole $\{\mathbf{x}_n\}_1^N$ can be mapped to $\left\{(y_n, z_n)^T\right\}_1^N$ in the same manner. It is easy to see that the resultant plots depend on the choice of $\mathbf{u}$ and $\mathbf{v}$. One option is to choose the pair of vectors that maximise the total variance of the projected data:

$$\begin{aligned} &\max_{\mathbf{u},\mathbf{v}} \ \mathrm{Var}(y) + \mathrm{Var}(z) \\ &\text{subject to} \ \ \|\mathbf{u}\| = 1, \|\mathbf{v}\| = 1, \mathbf{u} \perp \mathbf{v}\,. \end{aligned} \tag{3.4}$$

This means that we try to find a two dimensional space, i.e., a plane, such that the projected data on the plane spread as wide as possible rather than a plane on which the data are concentrated in a small area.

[1] $NB : \|\mathbf{u}\| = 1 bydefinition$

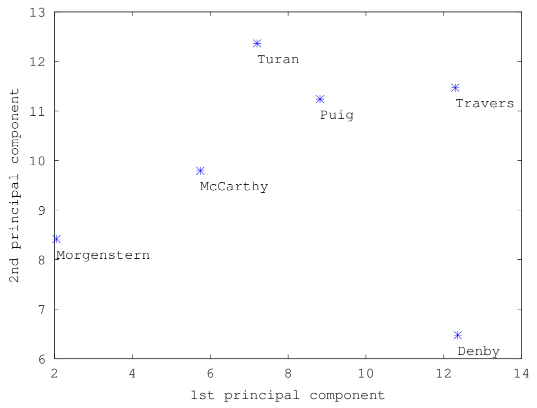


Figure 3.4: Plot of the critics in the 2-dimensional space defined by the first two principal components. (see Note 2)

It is known that the optimal projection vectors are given by the eigen vectors of the sample *covariance matrix*[2], **S**, defined as

$$\mathbf{S} = \frac{1}{N-1}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T \tag{3.5}$$

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n\,. \tag{3.6}$$

If scalar representation is preferred to the matrix one above, the element s_{ij} at i'th row and j'th column of **S**, is given as:

$$s_{ij} = \frac{1}{N-1}\sum_{n=1}^{N}(x_{ni} - m_i)(x_{nj} - m_j) \tag{3.7}$$

$$m_i = \frac{1}{N}\sum_{n=1}^{N}x_{ni}\,. \tag{3.8}$$

To be explicit, let $\{\lambda_1, \ldots, \lambda_D\}$ be the eigen values sorted in decreasing order so that λ_1 is the largest and λ_D is the smallest, the optimal projection vectors $\mathbf{u}^*$ and $\mathbf{v}^*$ are the eigen vectors that correspond to the two largest eigen values λ_1 and λ_2. Figure 3.4 depicts the scatter plot with the method for the example shown in Note 2.[3]

Generally speaking, if we would like to effectively transform $\mathbf{x}_n$ in a D-dimensional space to $\mathbf{y}_n$ in a lower ℓ-dimensional space ($\ell < D$), it can be done with the eigen vectors, $\mathbf{p}_1, \ldots, \mathbf{p}_\ell$, that correspond

[2]Covariance matrix is an extension of the variance for univariate case to the multivariate (multi-dimensional) case. We will see more details about covariance matrices in Note 8, where we consider Gaussian distributions.

[3]Someone might wonder how each film (review scores) contributes to the principal components. This can be confirmed with 'factor loading(s)', which is the correlation coefficient between the principal component and the review scores of the film.

to the ℓ largest eigen values $\lambda_1, \ldots, \lambda_\ell$:

$$\mathbf{y}_n = \begin{pmatrix} \mathbf{p}_1^T \\ \vdots \\ \mathbf{p}_\ell^T \end{pmatrix}\mathbf{x}_n = \begin{pmatrix} \mathbf{p}_1^T\mathbf{x}_n \\ \vdots \\ \mathbf{p}_\ell^T\mathbf{x}_n \end{pmatrix}. \tag{3.9}$$

This technique is called *principal component analysis (PCA)* and widely used for data visualisation, *dimensionality reduction*, data compression, and feature extraction. [4]

3.5 Summary

In this chapter we have:

1. introduced the notion of clustering a data set into non-overlapping groups using hierarchical or partitional approaches;
2. described the most important partitional algorithm, K-means clustering;
3. defined a within-cluster mean squared error function for K-means clustering;
4. introduced the notion of dimensionality reduction for data visualisation;
5. described the technique, principal component analysis (PCA) for dimensionality reduction.

The key properties of K-means are that it:

- Is an automatic procedure for clustering unlabelled data.
- Requires a pre-specified number of clusters.
- Chooses a set of clusters with the minimum within-cluster variance.
- Is guaranteed to converge (eventually, in a finite number of steps).
- Provides a locally optimal solution, dependent on the initialisation.

3.6 Reading

Further reading

- Bishop, section 9.1 (on clustering)
- Bishop, section 12.1 (on principal component analysis)
- Segaran, chapter 3 (on clustering)

[4]It is almost always the case that reducing dimensionality involves degradation, i.e., loss of information. The ratio, $\sum_{i=1}^{\ell}\lambda_i / \sum_{i=1}^{D}\lambda_i$, indicates how much amount of information retains after the conversion.

Classification and K-nearest neighbours

Hiroshi Shimodaira[*]

January-March 2017

In the previous chapter we looked at how we can use a distance measure to organise a data set into groupings. In this chapter—and for most of the rest of the course—we look at the case where each data point has a *label*, identifying it as a member of a class. In the recognition of handwritten digits, for example, there are ten possible labels corresponding to the digits "0"–"9". The problem of classification is to predict the correct label for an input feature vector. The classifier is obtained using a set of training data containing feature vectors and their labels. This is referred to as *supervised* learning, since the label for a training vector acts as supervision for the classifier when it is learning from training data.

Let's introduce some notation: $\mathbf{x} = (x_1,\ x_2, \ldots, x_D)^T$ is a D-dimensional (input) feature vector, which has class label c. The *training set* is a set of N feature vectors and their class labels; the n'th training item consists of a feature vector $\mathbf{x}_n$ and its class label c_n. The j'th element of the n'th feature vector is written x_{nj}. A learning algorithm is used to train a classifier using the training set. *Testing* involves assigning a class to an unlabelled feature vector $\mathbf{x}$.

We can measure how accurate a classifier is using an *error function*. A suitable error function for classification is the number of items that are misclassified—i.e., the number of times the classifier assigns a class label different from the true class label. The classification error is often expressed as the percentage of the total number of items that is misclassified, the "error rate". We also need to specify on which set of data the error function was measured. The "training set error rate" is the percentage of misclassifications that the classifier makes on the training set after the learning algorithm has been applied. The "test set error rate" refers to errors made by the classifier on a set of data not used by the learning algorithm—the test set.[1]

4.1 A simple example

Consider the problem of determining apples from pears. In this example case we have two features for each piece of fruit: its circumference (at the widest point) and its height, each measured in cm. Apples are "more spherical" than pears which tend to be longer than they are wide. But some pears are relatively short and some apples are taller than expected. In this case we have an input vector of the form $\mathbf{x} = (x_1, x_2)^T$, where x_1 is the circumference and x_2 the height. The class C can take two values A or P (standing for apples and pears).

We have a set of training data: height and circumference measurements, along with class labels. In Figure 4.1 we plot this two dimensional training data for the two classes. We can see that pears tend to have a greater height and smaller circumference compared with apples; however it is not possible to draw a straight line to separate the two classes (Figure 4.1).

[*]©2014-2017 University of Edinburgh. All rights reserved. This note is heavily based on notes inherited from Steve Renals and Iain Murray.

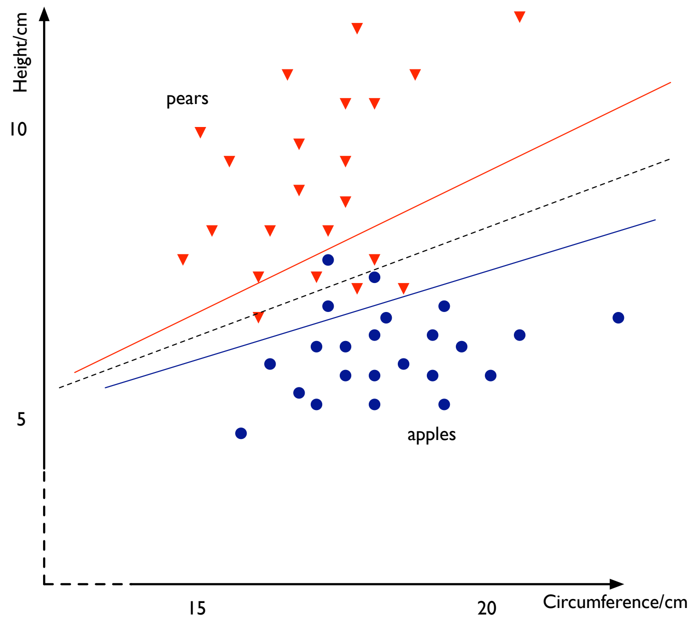


Figure 4.1: Training data for apples and pears. It is not possible to draw a straight line to perfectly separate the classes in this feature space. Three possible lines are drawn, but each results in a number of misclassifications.

We now have three new, unlabelled examples which we would like to classify (represented as stars in Figure 4.2):

- $(16, 10)$: all the training data in the region of this point is classified as P, so we classify this point as P.
- $(19, 6)$: looking at the training data it seems obvious to class this as A.
- $(17, 7)$: its not obvious in which class this example should be classified; the feature vector gives us evidence whether we have an apple or a pear, but does not enable us to make an unambiguous classification.

We can draw a straight line such that one side of it corresponds to one class and the other side to the other—as in the three possible lines shown in Figure 4.1. Such a line is called a *decision boundary*; if the data was three-dimensional, then the decision boundary would be defined by a plane. For one-dimensional data, a decision boundary can simply be a point on the real line. Intuitively it seems possible to find an optimal decision boundary, based on minimising the number of misclassifications in the training set. Although we use the training set to determine the decision boundary, we are most interested in making optimal decisions on a test set.

[1]The technique of using separate data sets for training and testing is referred to as *cross validation*.

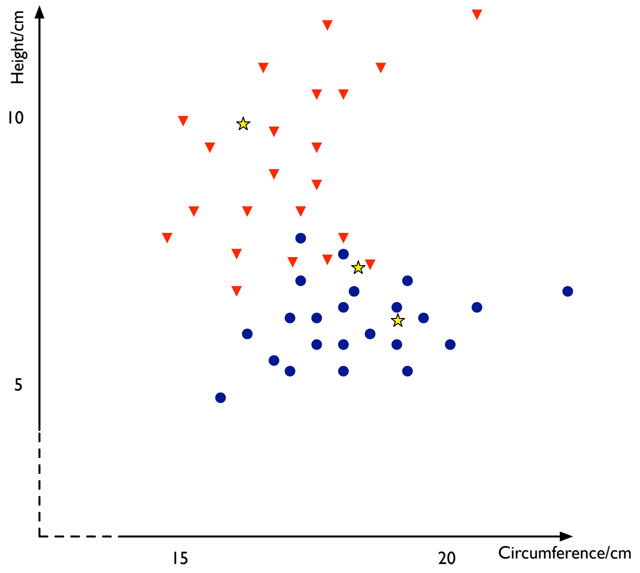


Figure 4.2: The training data for apples and pears, together with three test points.

4.2 K-Nearest Neighbour

An intuitive way to decide how to classify an unlabelled test item is to look at the training data points nearby, and make the classification according to the classes of those nearby labelled data points. This intuition is formalised in a classification approach called K-nearest neighbour (k-NN) classification.

Nearest neighbour classification has a very simple basis: to classify a test item, find the item in the training set which is closest and assign the test item to the same class as the selected training item. If there happens to be an identical item in the training set then it makes sense to assign the test item to the same class. Otherwise, the class of the member in the training set which is most similar to the test item is our best guess. We use a distance measure (e.g., Euclidean distance) to determine similarity. If we have a representation for which the distance measure is a reasonable measure of similarity, then the nearest neighbour method will work well.

4.2.1 K-Nearest Neighbour classification algorithm

Rather than just using the single closest point, the K-nearest neighbour approach looks at the K points in the training set that are closest to the test point; the test point is classified according to the class to which the majority of the K-nearest neighbours belong. For the first two items above, the value of K is not really important: $(16, 10)$ is classified as P and $(19, 6)$ is classified as A, no matter how many nearest neighbours are considered. However the third example above, $(17, 7)$, is ambiguous, and this is reflected in the sensitivity of the classifier to the value of K:

- 1-nearest: classified as pear

- 2-nearest: tie (one apple and one pair are nearest neighbours). In this case, just choose randomly between the two classes
- 3-nearest: classified as apple
- 5-nearest: classified as pear
- 9-nearest: classified as apple

K-nearest neighbour is very efficient at training time, since training simply consists of storing the training set.[2] Testing is much slower, since it can potentially involve measuring the distance between the test point and every training point, which can make K-nearest neighbour impractical for large data sets.

We can write the K-nearest neighbour algorithm precisely as follows, where X is the training data set with class labels so that $X = \{(\mathbf{x}, c)\}$, Z is the test set, there are C possible classes, and r is the distance metric (typically the Euclidean distance):

- For each test example $\mathbf{z} \in Z$:
  - Compute the distance $r(\mathbf{z}, \mathbf{x})$ between $\mathbf{z}$ and each training example $(\mathbf{x}, c) \in X$
  - Select $U_k(\mathbf{z}) \subseteq X$, the set of the k nearest training examples to $\mathbf{z}$
  - Decide the class of $\mathbf{z}$ by the majority voting:

$$c(\mathbf{z}) = \arg\max_{j \in \{1,\ldots,C\}} \sum_{(\mathbf{x},c) \in U_k(\mathbf{z})} \delta_{jc} \tag{4.1}$$

where the Kronecker delta $\delta_{jc} = 1$ when $j = c$ and zero otherwise. It is sometimes possible to store the training data set in a data structure that enables the nearest neighbours to be found efficiently, without needing to compare with every training data point (in the average case). One such data structure is the k-d tree. However, these approaches are usually only fast in practice with fairly low-dimensional feature vectors, or if approximations are made.

4.2.2 Decision boundaries by K-NN classification

K-nearest neighbour classifiers make their decisions on the basis of local information. Rather than trying to draw decision boundaries across the whole space (as in Figure 4.1), K-nearest neighbour techniques make decisions based on a few local points. As such they can be quite susceptible to noise, especially if K is small: a small shift in the location of a test point can result in a different classification since a new point from the training data set becomes the nearest neighbour. As K becomes larger, the classification decision boundary becomes smoother since several training points contribute to the classification decision.

What do the decision boundaries look like for nearest neighbour classification? Consider a special case of 1-nearest neighbour classification in which each training data point belongs to a distinct class from the others. In that case each training data point defines a region around it; test points within this region will be classified to the same class as the training data point. These regions are illustrated for a simple case in Figure 4.3, where the boundaries of regions are shown as dotted lines. As you may

[2] In practice, responsible machine learning practitioners will try out different choices of K, and different distance measures, possibly optimising free parameters of a distance measure. Then training requires testing different choices, and becomes expensive.

understand now, each boundary is given as the perpendicular bisector of the line segment between the two corresponding data points. This partitioning formed by a set of data points is sometimes called *Voronoi diagram* or *Voronoi tessellation*.

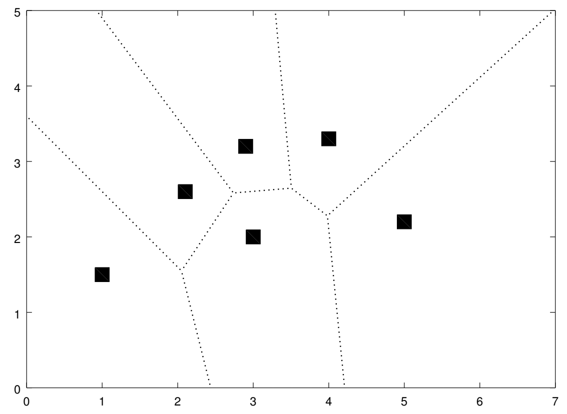


Figure 4.3: Decision boundaries by 1-NN for data points of distinct classes from each other

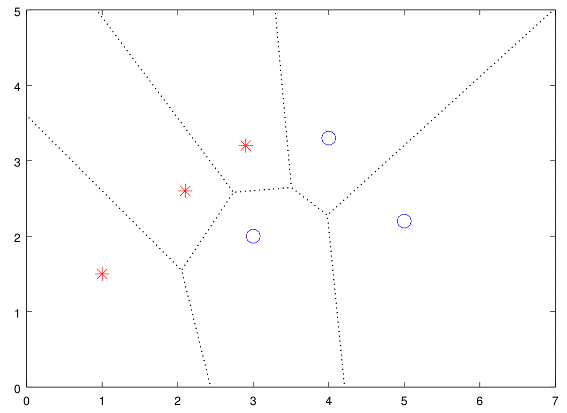


Figure 4.4: Nearest neighbour regions for a training data set of two classes, where training samples of one class are shown with '*' in red, those of the other class are shown with '○' in blue. The Euclidean distance is used as the distance measure in this example.

Now, we assume that each data points belongs to either of two classes, say, red or blue shown classes as is shown in in Figure 4.4,

To obtain the decision boundary we combine those boundaries which are between regions of different

classes, as illustrated in Figure 4.5. The resultant boundary is referred to as being *piecewise linear*. Figures 4.6 and 4.7 show the case of three classes.

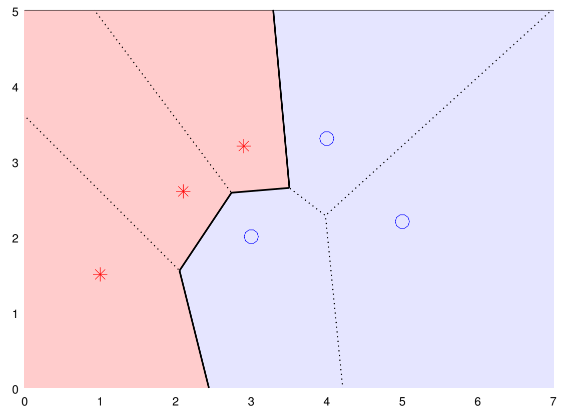


Figure 4.5: Decision boundary and decision regions for a 1-nearest neighbour classifier for two classes.

4.3 Reading

For further reading on K-nearest neighbour see

- Mitchell, sections 8.1 and 8.2
- Duda, Hart and Stork, sections 4.4, 4.5 and 4.6

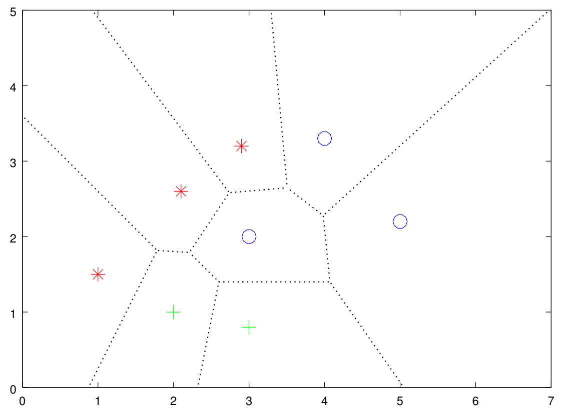


Figure 4.6: Nearest neighbour regions for a training data set of three classes.

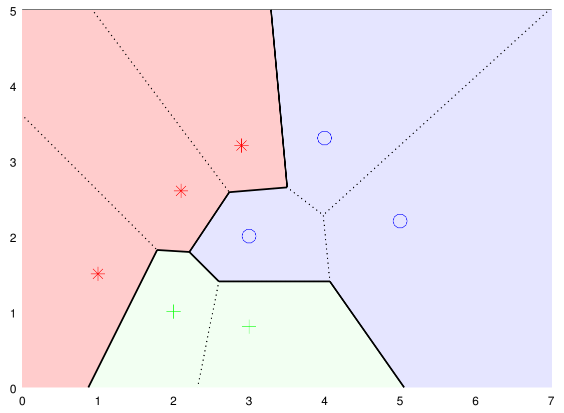


Figure 4.7: Decision boundary and decision regions for a 1-nearest neighbour classifier for three classes.

Introduction to Statistical Pattern Recognition and Optimisation

Hiroshi Shimodaira*

January-March 2017

When we are trying to make decisions, uncertainty arises due to:

- Inaccurate or incomplete information about the situation (for example due to noisy or inaccurate sensors).
- A lack of complete knowledge about the situation (for example we may only have information arising from the available sensors, perhaps missing out on other sources of information).

We must assume that only uncertain and partial information is available, and the system must make decisions taking this into account. The mathematics of probability provides the way to deal with uncertainty, and tells us how to update our knowledge and beliefs if new information becomes available. This chapter introduces the use of probability and statistics for pattern recognition and learning.

5.1 A simple example

Consider the problem of determining the sex of fish. We could carefully examine the fish, which would be slow and tedious. For many species of fish it is known that male fish tend to be longer than female fish. It is much easier and faster to measure the length of a fish than to determine its sex directly. Thus we would like a system to determine the sex of fish from their length, as accurately as possible.[1]

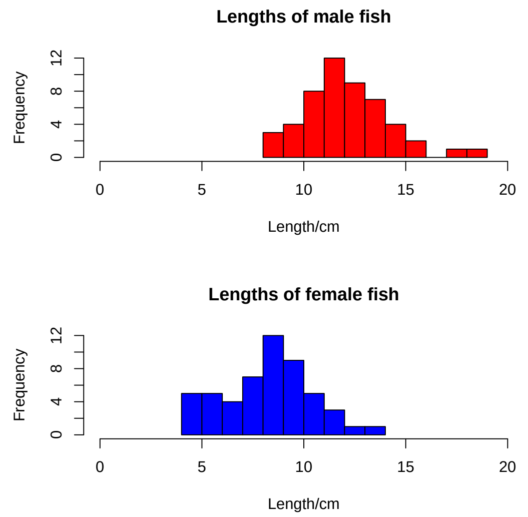
We will consider the following scenario: we have a set of *labelled data* that we can use as a training set. This is a set of measurements of the length of each fish, together with the class label (male or female). In Figure 5.1a we plot this dataset as two histograms, one for for each class (sex), showing the number of fish of each length in each class. The length data comes as integer values (cm); later we'll look at directly modelling continuous valued data.

We now have four new, unlabelled examples with lengths 5 cm, 9 cm, 12 cm, 16 cm. How do we classify each of these?

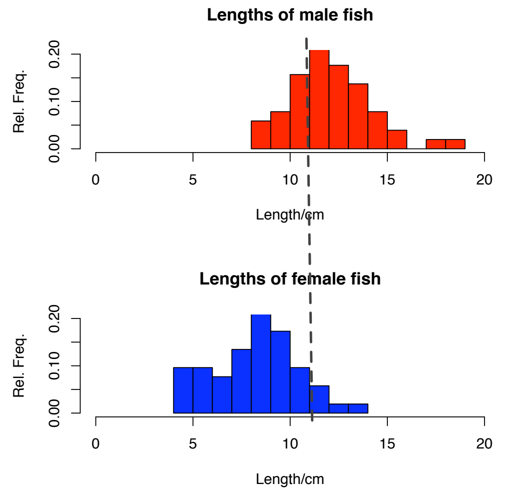
- 5 cm: It seems clear that it should be classified female, since we have never seen a male fish this short. (But can we be *sure* that we will *never* see a male fish of this length?)

*©2014-2017 University of Edinburgh. All rights reserved. This note is heavily based on notes inherited from Steve Renals and Iain Murray.

[1]This is based on a real problem; there is a classic paper (R. Redner and H. Walker, "Mixture densities, maximum likelihood, and the EM algorithm", SIAM Rev. 26 (1984), pp. 195–239.) which uses as an example the determination of the sex of halibut using their length as the feature, based on data from the Seattle Fisheries Commission.



(a) Histograms showing observed frequencies of lengths of male and female fish.



(b) Histograms showing relative frequencies of lengths of male and female fish. The dotted line shows a possible decision boundary between male and female, based on the length.

Figure 5.1: Histograms showing lengths of male and female fish.

- 16 cm: Likewise, it seems safe to classify the 16 cm fish as male
- 9 cm: We have observed both male and female fish that are 9 cm long, but about twice as many females of that length as males. So it would seem sensible to classify that fish as female.
- 12 cm: We have also observed both male and female fish that are 12 cm long, but more male than female, so it seems to make sense to classify that fish as male.

For the 9 cm and 12 cm cases (in particular), it is not possible to make an unambiguous classification. The length of the fish gives us evidence whether it is male or female, but the length alone is not enough information to be sure.

Rather than plotting the histograms with the observed frequencies for each length, we could plot them using the *relative frequencies*—the proportions of male and female fish that are each length (Figure 5.1b).

We can assign a particular value of the length to be a *decision boundary*—that value of the length at which we assign all shorter fish to be female and longer ones to be male. A possible decision boundary is marked by the dotted line in Figure 5.1b. Intuitively it seems like that this is a good decision point, based on minimising the number of misclassifications in the training set. While we use the training set to determine the decision boundary, we are most interested in classifying new data (a test set).

We can think of the relative frequencies as *probability estimates*; estimates of the probability of the length given that the fish is male (or female). If we call the length X and the class S, then we can write that the relative frequencies are estimates of $P(X = \ell \mid S = \text{male})$ and $P(X = \ell \mid S = \text{female})$: the probability that the length X has value ℓ given that the fish is male (female).

Question: Imagine that you know that there are 10 times as many male fish as female fish. Would you still make the same classifications?

5.2 Probability refresher

Before jumping into the use of probabilities for pattern recognition and learning, let's briefly revise basic probability.

Some events are *deterministic*: if you drop an apple, it will hit the floor (event A *causes* event B).

Some events are random. They may be intrinsically random (such as radioactive decay) or they may be deterministic, but we do not have enough information (or it is prohibitive to do the computations). For example tossing a coin is a deterministic physical process: if we had enough information (force of the toss, weight of the coin, etc.) then it would be possible to compute the outcome of the toss precisely (using Newtonian mechanics). Forecasting the weather is another example in this category, but much more complex!

Probabilistic models are the correct model to use when acting under uncertainty: we never have the *whole truth*. We can use probability theory to make decisions based on what we know.

5.2.1 What is probability?

This turns out to be a deep and controversial question! Indeed, there is still debate about the correct interpretation of probability. There are two basic positions:

- Probability is a *frequency limit*: e.g., tossing a fair coin N times, $n(\text{heads})/N \rightarrow 1/2$ as $N \rightarrow \infty$;

- Probability is a *degree of belief*: e.g., given a set of symptoms a doctor may believe you a have a particular disease with probability $P(\text{disease} \mid \text{symptoms})$;

It turns out that it is possible to derive the mathematics of probability starting from a small set of axioms.

Sample space

A *trial* is the basic event which we want to model (e.g., toss of a coin, roll of a dice). A *Sample space* — set of all possible outcomes of a trial (e.g., $\{1, 2, 3, 4, 5, 6\}$ in the case of rolling a dice).

If the outcome of an experiment is A, then the complement of A is written $\overline{A}$ "not A", the set of all other outcomes. The sample space is $\{A, \overline{A}\}$. In this case we consider A as representing an event that may be true or false, and we have a sample space {true, false}, or $\{1, 0\}$.

Symmetry

We would like to assign probabilities to events in sample space, such that equivalent (symmetric, interchangeable) outcomes should be assigned the same probability (e.g., tossing a fair coin). And we would like to constrain the total probability of all outcomes to be 1. If a sample space is composed of N symmetric events (e.g., rolling a fair dice), then the probability of each event in the sample space should be $1/N$.

Independence

Two events A and B are *independent* if the outcome of A does not influence the outcome of B (and vice-versa). The sample space of two independent events is their Cartesian Product.

5.2.2 Boxes example (Bishop, 2006)

We have two boxes, 1 red and 1 green. In the red box we have 2 apples and 6 oranges. In the green box we have 3 apples and 1 orange. Suppose we randomly choose a box, and from the chosen box we randomly choose a piece of fruit. We choose the red box 40% of the time, and the green box 60% of the time.

In this example we have two *random variables*:

- B the identity of the box, which can take two values r (red) or g (green)
- F the type of fruit, with two possible values a (apple) and o (orange)

We can define the probability of an event as the fraction of times that event occurs (as $N \rightarrow \infty$). We can write the probability of selecting the red box as $P(B = r) = 4/10$, and the probability of selecting the green box as $P(B = g) = 6/10$. We only have two boxes and we must choose one of them (B can only take values r or g), so

$$P(B = r) + P(B = g) = 1.$$

We'll come back to this example.

5.2.3 Joint and Conditional Probability

Assume we have two random variables X and Y. X can take one of I values $x_1, \ldots, x_i, \ldots, x_I$ and Y can take one of J values $y_1, \ldots, y_j, \ldots, y_J$. Consider N trials where we sample the values of X and Y, and let n_{ij} be the number of times $X = x_i$ and $Y = y_j$. If we consider the limit $N \to \infty$, then we can define the *joint probability* that X has value x_i and Y has value y_j as:

$$P(X = x_i, Y = y_j) = \frac{n_{ij}}{N}\,.$$

Sometimes, to avoid clutter, we write $P(x_i, y_j)$. The ordering of the terms is irrelevant here: $P(X = x_i, Y = y_j) = P(Y = y_j, X = x_i)$.

Let n_i be the number of times $X = x_i$, irrespective of the value of Y. Then we can write:

$$P(X = x_i) = \frac{n_i}{N}\,,$$

which can also be abbreviated as $P(x_i)$. From the definition of n_i,

$$n_i = \sum_{j=1}^{J} n_{ij}\,.$$

Therefore we can write

$$P(X = x_i) = \sum_{j=1}^{J} P(X = x_i, Y = y_j)\,, \tag{5.1}$$

which is called the *law of total probability* or the *sum rule*.

The *conditional probability* $P(X = x_i \mid Y = y_j)$, read as the probability that $X = x_i$ given that $Y = y_j$, is obtained by only considering events when $Y = y_j$, and is the proportion of the time when $X = x_i$ in that case:

$$P(X = x_i \mid Y = y_j) = \frac{n_{ij}}{n_j}\,.$$

Now, since

$$\frac{n_{ij}}{N} = \frac{n_{ij}}{n_j}\frac{n_j}{N}\,,$$

we may write the *product rule* which relates the joint probability to the conditional probability:

$$P(X = x_i, Y = y_j) = P(X = x_i \mid Y = y_j)\,P(Y = y_j)\,. \tag{5.2}$$

If we want to write the distribution for an arbitrary value of the random variable X we can write $P(X)$. Using this more compact notation we can write the sum and product rules as:

$$P(X) = \sum_Y P(X, Y) \tag{5.3}$$

$$P(X, Y) = P(X \mid Y)\,P(Y) = P(Y \mid X)\,P(X)\,. \tag{5.4}$$

You can read $P(X, Y)$ as "the probability of X *and* Y", read $P(Y \mid X)$ as "the probability of Y *given* X", and read $P(X)$ as "the probability of X". $\sum_Y$ refers to the sum over all values that Y can take.

5.2.4 Bayes' Theorem

Re-expressing (5.4), we can write:

$$P(Y \mid X) = \frac{P(X \mid Y)\,P(Y)}{P(X)}\,. \tag{5.5}$$

This is known as *Bayes' Theorem*. It is extremely useful for computing $P(Y \mid X)$ when $P(X \mid Y)$ is known. Bayes' theorem and the law of total probability are at the centre of statistical pattern recognition and machine learning.

Using the law of total probability (sum rule) we can expand the denominator as:

$$P(X) = \sum_Y P(X \mid Y)\,P(Y)\,,$$

and we can write Bayes' theorem as

$$P(Y \mid X) = \frac{P(X \mid Y)\,P(Y)}{\sum_Y P(X \mid Y)\,P(Y)}\,. \tag{5.6}$$

5.2.5 Boxes example continued

In this example we had:

$$\begin{aligned} P(B = r) &= 4/10 \\ P(B = g) &= 6/10\,. \end{aligned}$$

The probability of picking an apple from the red box (conditional probability of picking an apple given that red box was chosen) is the fraction of apples in the red box (1/4) and is written as $P(F = a \mid B = r)$. We can write the set of conditional probabilities of picking a type of fruit given a box:

$$\begin{aligned} P(F = a \mid B = r) &= 1/4 \\ P(F = o \mid B = r) &= 3/4 \\ P(F = a \mid B = g) &= 3/4 \\ P(F = o \mid B = g) &= 1/4\,. \end{aligned}$$

As a check, we can verify that each conditional distribution is normalised:
$P(F = a \mid B = r) + P(F = o \mid B = r) = 1$ and $P(F = a \mid B = g) + P(F = o \mid B = g) = 1$.

We can use the law of total probability to evaluate the overall probability of choosing an apple:

$$\begin{aligned} P(F = a) &= P(F = a \mid B = r)\,P(B = r) + P(F = a \mid B = g)\,P(B = g) \\ &= 1/4 \cdot 4/10 + 3/4 \cdot 6/10 \\ &= 22/40 = 11/20\,. \end{aligned}$$

And the probability of choosing an orange is:

$$P(F = o) = 1 - P(F = a) = 9/20\,.$$

Now suppose we are told that an apple was chosen, but we don't know which box it came from. We can use Bayes' theorem to evaluate the conditional probability that the red box was chosen, given that an apple was picked:

$$\begin{aligned} P(B=r \mid F=a) &= \frac{P(F=a \mid B=r)\, P(B=r)}{P(F=a)} \\ &= \frac{1/4 \cdot 4/10}{11/20} \\ &= 2/11\,. \end{aligned}$$

So, the probability that red box was chosen given that an apple was picked is 2/11, and

$$P(B=g \mid F=a) = 1 - P(B=r \mid F=a) = 1 - 2/11 = 9/11\,,$$

the probability that green box was chosen given that an apple was picked is 9/11.

We can interpret Bayes' theorem as follows. Without having picked any fruit, the best information we have about the probabilities of the two boxes are given by the *prior probabilities* $P(B)$. Once we have picked a fruit, then we have some additional information and we can use Bayes' theorem to compute the *posterior probabilities* $P(B|F)$. Without having observed any fruit, the probability of choosing the green box is 6/10. If an apple is observed we can incorporate this information, finding that the posterior probability is 9/11. Observing the apple makes it more probable that the box we selected was the green one.

5.3 Bayes' Theorem and Pattern Classification

Bayes' theorem is at the heart of statistical pattern recognition. Let's look at how, in general terms, we can express a pattern classification problem using Bayes' theorem.

Consider a pattern classification problem in which there are K classes. Let C denote the class, taking values $1, \ldots, K$. The observed input data, which is a D-dimensional feature vector, is denoted by X. Once the training set is used to train the classifier, a new, unlabelled data point $\mathbf{x}$ is observed. To make a classification we could compute the *posterior probabilities* (also called *a posteriori* probabilities) $P(C=k|X=\mathbf{x})$, for every class $k = 1, \ldots, K$; we can then classify $\mathbf{x}$ by assigning it to the class with the highest posterior probability, k_{max}. We can write this operation as:

$$k_{max} = \arg\max_{k \in \{1,\ldots,K\}} P(C=k|X=\mathbf{x})\,. \tag{5.7}$$

This procedure is sometimes called MAP (maximum a posteriori) decision rule. The max operator returns a probability that is the maximum value of $P(C=k|X=\mathbf{x})$ over all values of C; the arg max operator returns the argument (the value of k_{max}) corresponding to the maximum probability. More compactly, we can write $P(C=k|X=\mathbf{x})$ as $P(C_k|\mathbf{x})$.[2]

MAP classification of $\mathbf{x}$ requires estimates of the conditional probability of each class, k, given $\mathbf{x}$. We can re-express the required conditional probabilities using Bayes' theorem:

$$P(C_k|\mathbf{x}) = \frac{P(\mathbf{x}|C_k)\, P(C_k)}{P(\mathbf{x})}\,. \tag{5.8}$$

We have expressed the posterior probability $P(C_k|\mathbf{x})$ as a product of:

[2]We will simply use C_k to denote $C = k$ for the ease of readability.

- the *likelihood*[3] of the class k, $P(\mathbf{x}|C_k)$, given data $\mathbf{x}$;
- the *prior probability*, $P(C_k)$, of the class k.

There is also the denominator, $P(\mathbf{x})$, to consider. We can compute this term using the law of total probability:

$$P(\mathbf{x}) = \sum_{\ell} P(\mathbf{x}|C_\ell)\, P(C_\ell)\,.$$

However, because $P(\mathbf{x})$ is the same for all classes, we do not need to consider it when finding the most probable class, so that $P(C_k|\mathbf{x})$ is proportional to $P(\mathbf{x}|C_k)\, P(C_k)$, i.e.

$$P(C_k|\mathbf{x}) \propto P(\mathbf{x}|C_k)\, P(C_k)\,. \tag{5.9}$$

If we return to the fish example, we can see why re-expressing the posterior probability using Bayes' theorem is a useful thing to do.

5.4 Fish example redux

We have 200 examples of fish lengths, 100 male and 100 female. Let class $C = M$ represent male, and $C = F$ represent female. Remember that we are assuming fish come in integer lengths—or, more realistically, if a fish has a length between 9.5 and 10.5, then $X = 10$. Let the number of male and female fish from with length x be given by $n_M(x)$ and $n_F(x)$, and the total number of fish in each class k are denoted N_M and N_F. These counts are given in table 5.1 for our example dataset.

We can estimate the likelihoods $P(x|M)$ and $P(x|F)$ as the counts in each class for length x divided by the total number of examples in that class:

$$\begin{aligned} P(x|M) &\approx \frac{n_M(x)}{N_M} \\ P(x|F) &\approx \frac{n_F(x)}{N_F}\,. \end{aligned}$$

To estimate these likelihoods, we consider the fish in each class separately (probability estimates are conditional on the class). We obtain estimates of $P(x|M)$ and $P(x|F)$ (and NOT estimates of $P(M|x)$ and $P(F\,|\,x)$). Thus we have estimated the likelihoods of the length given each class using relative frequencies (using the training set of 100 examples from each class). These are also tabulated in table 5.1. (These are probability *estimates* since N_M and N_F are finite, which is always the case for the real world.)

We can use Bayes' theorem to estimate the posterior probabilities of each class given the data:

$$\begin{aligned} P(M\,|\,x) &= \frac{P(x\,|\,M)\, P(M)}{P(x)} \propto P(x\,|\,M)\, P(M) \\ P(F\,|\,x) &= \frac{P(x\,|\,F)\, P(F)}{P(x)} \propto P(x\,|\,F)\, P(F)\,. \end{aligned}$$

[3]In common English usage, "likelihood" is more-or-less a synonym for probability, perhaps with extra connotations of an event being hypothetical. In technical statistical usage, the likelihood is a property of an explanation of some data: a model or its parameters. The posterior probability of a parameter, model, or class, is proportional to its prior multiplied by its likelihood. Therefore saying "likelihood of the data", while commonly-seen and acceptable informal English, conflicts with the 'correct' statistical usage: "likelihood of the model (given the data)".

x	$n_M(x)$	$P(x\|M)$	$n_F(x)$	$P(x\|F)$
1	0	0.00	0	0.00
2	0	0.00	0	0.00
3	0	0.00	0	0.00
4	0	0.00	2	0.02
5	1	0.01	7	0.07
6	2	0.02	8	0.08
7	2	0.02	10	0.10
8	2	0.02	14	0.14
9	7	0.07	21	0.21
10	8	0.08	19	0.19
11	14	0.14	10	0.10
12	22	0.22	4	0.04
13	19	0.19	2	0.02
14	11	0.11	1	0.01
15	6	0.06	1	0.01
16	2	0.02	1	0.01
17	2	0.02	0	0.00
18	1	0.01	0	0.00
19	1	0.01	0	0.00
20	0	0.00	0	0.00

Table 5.1: Example fish lengths for male and female, tabulated showing counts of each length per class, and relative frequencies used to estimate likelihoods for each class. (NB: These are different from the example data shown in section 5.1.)

In the case of this two-class problem:

$$\begin{aligned} P(x) &= P(x\,|\,M)\,P(M) + P(x\,|\,F)\,P(F) \\ P(M) + P(F) &= 1 \\ P(M\,|\,x) + P(F\,|\,x) &= 1\,. \end{aligned}$$

If we want to compare the posterior probabilities of M and F given the data we can take their ratio:

$$\begin{aligned} \frac{P(M\,|\,x)}{P(F\,|\,x)} &= \frac{P(x\,|\,M)\,P(M)/P(x)}{P(x\,|\,F)\,P(F)/P(x)} \\ &= \frac{P(x\,|\,M)\,P(M)}{P(x\,|\,F)\,P(F)}\,. \end{aligned} \tag{5.10}$$

In this case, if the ratio (5.10) is greater than 1 then x is classified as M, if x is less than 1 then x is classified as F. As mentioned above, the denominator term $P(x)$ cancels.

Let's look at the same four test points as before. In this case let us assume that male and female fish have equal prior probabilities, that is $P(M) = P(F) = 1/2$.

1. $X = 5$
$$\frac{P(M\,|\,X{=}5)}{P(F\,|\,X{=}5)} = \frac{P(X{=}5\,|\,M)\,P(M)}{P(X{=}5\,|\,F)\,P(F)} = \frac{0.01 \cdot 0.5}{0.07 \cdot 0.5} = 1/7$$
Hence classify as $X{=}5$ as female (F).

2. $X = 16$
$$\frac{P(M\,|\,X{=}16)}{P(F\,|\,X{=}16)} = \frac{P(X{=}16\,|\,M)\,P(M)}{P(X{=}16\,|\,F)\,P(F)} = \frac{0.02 \cdot 0.5}{0.01 \cdot 0.5} = 2$$
Hence classify as $X{=}16$ as male (M).

3. $X = 9$
$$\frac{P(M\,|\,X{=}9)}{P(F\,|\,X{=}9)} = \frac{P(X{=}9\,|\,M)\,P(M)}{P(X{=}9\,|\,F)\,P(F)} = \frac{0.07 \cdot 0.5}{0.21 \cdot 0.5} = 1/3$$
Hence classify as $X{=}9$ as female (F).

4. $X = 12$
$$\frac{P(M\,|\,X{=}12)}{P(F\,|\,X{=}12)} = \frac{P(X{=}12\,|\,M)\,P(M)}{P(X{=}12\,|\,F)\,P(F)} = \frac{0.22 \cdot 0.5}{0.04 \cdot 0.5} = 5.5$$
Hence classify as $X{=}12$ as male (M).

Equal prior probabilities mean that we are equally likely to find a male or a female fish. If we believe that one sex is more prevalent than the other, then we can adjust the prior probability accordingly—and Bayes' theorem incorporates this information. Consider $X{=}11$; if the priors probabilities are equal (0.5), then we classify that value as male (M) since

$$\frac{P(M\,|\,X{=}11)}{P(F\,|\,X{=}11)} = \frac{P(X{=}11\,|\,M)\,P(M)}{P(X{=}11\,|\,F)\,P(F)} = \frac{0.14 \cdot 0.5}{0.10 \cdot 0.5} = 1.4$$

However, if we know there are twice as many females as males (i.e., $P(M) = 1/3$, $P(F) = 2/3$), then the ratio becomes

$$\frac{P(M\,|\,X{=}11)}{P(F\,|\,X{=}11)} = \frac{P(X{=}11\,|\,M)\,P(M)}{P(X{=}11\,|\,F)\,P(F)} = \frac{0.14 \cdot /3}{0.10 \cdot 2/3} = 0.7$$

and we classify the point as female (F).

The values we have been using for $P(x\,|\,M)$ and $P(x\,|\,F)$ are estimates, based on the relative frequencies—technically we refer to these as *maximum likelihood estimates*.[4]

Some questions. Assuming equal prior probabilities:

1. What is the value of $P(M \mid X=4)$?
2. What is the value of $P(F \mid X=18)$?
3. You observe data point $X=20$. To which class should it be assigned?

Comment on your answers, and any changes you might make to the probability estimates.

5.5 Optimisation problem

In section 5.3, we studied the basic idea of statistical pattern classification based on probabilities. To make the framework work properly, the probabilities need to be estimated from the training data as accurate as possible. To that end, we define a certain criterion to find what are supposed to be 'best'. This concept of optimality or optimisation plays an essential role in pattern recognition. For example, we will study parameter estimation of Gaussian distributions in Chapter 8, and training of neural networks in Chapters 11 and 12, both of which are defined as optimisation problems. We have already seen two examples of optimisation problem in Chapter 3. One is the mean squared error function for K-means clustering shown in Equation (3.1):

$$E = \frac{1}{N}\sum_{k=1}^{K}\sum_{n=1}^{N} z_{nk}\|\mathbf{x}_n - \mathbf{m}_k\|^2 \tag{5.11}$$

which we wanted to minimise with respect to the set of mean vectors $\{\mathbf{m}_k\}_1^K$. This can be formulated as the following optimisation problem:

$$\min_{\{\mathbf{m}_k\}_1^K} \frac{1}{N}\sum_{k=1}^{K}\sum_{n=1}^{N} z_{nk}\|\mathbf{x}_n - \mathbf{m}_k\|^2\,. \tag{5.12}$$

Recall that the K-means clustering is an algorithm to find a local minimum solution of the optimisation problem. The second example is found in Equation (3.4), which is a formulation for PCA to find the first two principal components, $\mathbf{u}$ and $\mathbf{v}$, such that

$$\begin{aligned} &\max_{\mathbf{u},\mathbf{v}} \ \mathrm{Var}\,(y) + \mathrm{Var}\,(z) \\ &\text{subject to } \ \|\mathbf{u}\| = 1, \|\mathbf{v}\| = 1, \mathbf{u} \perp \mathbf{v}\,. \end{aligned} \tag{5.13}$$

Optimisation is a large field of study, which can be categorised in terms of types of variable (continuous or discrete) and types of constraint (unconstrained or constrained). The two examples above both fall into the category of continuous optimisation, but the former is a unconstrained optimisation problem, whereas the latter is a constrained one. In this section, we only consider continuous and unconstrained optimisation whose typical form [5] is given as

$$\min_{\mathbf{x}} f(\mathbf{x}) \tag{5.14}$$

[4]There are more sophisticated ways to deal with a model's unknown parameters, although these are beyond the scope of this course. For example, fully Bayesian methods consider all possible values of the parameters, weighted by the plausibility (posterior probability) of each setting.

[5] It does not matter whether we consider max or min, as $\max f(\mathbf{x})$ is equivalent to $\min -f(\mathbf{x})$.

where $\mathbf{x} \in \mathcal{R}^D$ and f is a smooth function such that $f : \mathcal{R}^D \to \mathcal{R}$. The function f to be optimised is called an *objective function*, and the solutions of the optimisation problem is called *optimal solutions*. It is not generally possible to find analytic (closed-form) solutions, and iterative optimisation algorithms are normally required as we have seen in the K-means clustering, but we here consider very simple cases where closed forms exist.

It is easy to see that the optimal solutions satisfy the following condition:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = 0, \text{ for } i = 1, \ldots, D \tag{5.15}$$

where $\frac{\partial f(\mathbf{x})}{\partial x_i}$ denotes the *partial derivative* of $f(\mathbf{x})$ with respect to the scalar variable x_i. This can be written in a vector form:

$$\left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_D}\right)^T = \mathbf{0} \tag{5.16}$$

where $\mathbf{0} = (0, \ldots, 0)^T$ is a zero vector in a D-dimensional vector space. The left hand side of this equation is called a *gradient* or a *gradient vector* of $f(\mathbf{x})$ and often denoted as $\nabla f(\mathbf{x})$ or $\mathrm{grad} f(\mathbf{x})$. Note that the condition above (i.e. $\nabla f(\mathbf{x}) = \mathbf{0}$) is not generally a sufficient condition for optimal solutions, but a necessary condition. However, the examples shown below consider a rather simple and specific case of quadratic functions in which the condition is sufficient to find optimal solutions.

5.5.1 Optimisation of a quadratic function of one variable

A simple yet meaningful example of optimisation problem would be the minimisation of the following quadratic function of a scalar variable x:

$$f(x) = ax^2 + bx + c \tag{5.17}$$

where $a > 0$, and b, c are arbitrary constants. By rewriting it into

$$ax^2 + bx + c = a\left(x + \frac{b}{2a}\right)^2 - \frac{b^2}{4a} + c \tag{5.18}$$

we can easily see that the solution is $x = -\frac{b}{2a}$. This can be also confirmed by using the necessary condition shown Equation (5.15) to get

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = 2ax + b = 0 \tag{5.19}$$

which gives the same solution.

5.5.2 Optimisation of a quadratic function of two variables

We now consider the following function g of two variables, x and y :

$$g(x, y) = ax^2 + by^2 + cxy + dx + ey + f \tag{5.20}$$

where, for simplicity's sake, we assume $a > 0$, $b > 0$, $c^2 < 4ab$, and d, e, f are arbitrary constants. Taking partial derivatives with respect to x and y yields

$$\frac{\partial g}{\partial x} = 2ax + cy + d = 0 \tag{5.21}$$

$$\frac{\partial g}{\partial y} = 2by + cx + e = 0\,. \tag{5.22}$$

To find the optimal solution (x, y), we rewrite the above into a system of linear equations in matrix form:

$$\begin{pmatrix} 2a & c \\ c & 2b \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -d \\ -e \end{pmatrix}. \tag{5.23}$$

Thus

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2a & c \\ c & 2b \end{pmatrix}^{-1} \begin{pmatrix} -d \\ -e \end{pmatrix} \tag{5.24}$$

$$= \frac{1}{4ab - c^2} \begin{pmatrix} -2bd + ce \\ cd - 2ae \end{pmatrix}. \tag{5.25}$$

Question: Extend the above example to a quadratic function of three variables, $x, y,$ and z. How about a case of n variables?

5.5.3 Line of best fit (Least square error line fitting)

Consider a set of N observations $\{\mathbf{p}_n\}_1^N$ in a 2D space, where $\mathbf{p}_n = (x_n, y_n)^T$, for which we would like to find the best fit line $y = ax + b$, where a and b are the parameters of the line.[6] As an objective function, we can consider the mean squared error between the predicted value $\hat{y}_i$ from x_i and the actual value y_i, so that the objective function E is given as:

$$E = \frac{1}{N} \sum_{n=1}^{N} (\hat{y}_n - y_n)^2 \tag{5.26}$$

where

$$\hat{y}_n = ax_n + b\,. \tag{5.27}$$

Thus, the optimisation problem is defined as $\min_{a,b} E$, i.e.,

$$\min_{a,b} \frac{1}{N} \sum_{n=1}^{N} (\hat{y}_n - y_n)^2\,. \tag{5.28}$$

Note that, as opposed to previous examples, this objective function E is not a function of x, but a function of a and b.

To solve the problem, we at first rewrite E by substituting Equation (5.27) into Equation (5.26).

$$E = \frac{1}{N} \sum_{n=1}^{N} (ax_n + b - y_n)^2\,. \tag{5.29}$$

The partial derivations of E with respect to a and b are given as

$$\frac{\partial E}{\partial a} = \frac{2}{N} \sum_{n=1}^{N} (ax_n + b - y_n)x_n = \frac{2}{N} \left(a \sum_{n=1}^{N} x_n^2 + b \sum_{n=1}^{N} x_n - \sum_{n=1}^{N} x_n y_n \right) \tag{5.30}$$

$$\frac{\partial E}{\partial b} = \frac{2}{N} \sum_{n=1}^{N} (ax_n + b - y_n) = \frac{2}{N} \left(a \sum_{n=1}^{N} x_n + b \sum_{n=1}^{N} 1 - \sum_{n=1}^{N} y_n \right). \tag{5.31}$$

[6] This is regarded as a *regression line* on y with x, whereas we can also consider a regression line on x with y, which is defined as $x = cy + d$.

By making the both partial derivatives, $\frac{\partial E}{\partial b}$ and $\frac{\partial E}{\partial b}$, equal to zero, and ignoring irrelevant coefficients, we get a system of linear equations in a and b:

$$a \sum_{n=1}^{N} x_n^2 + b \sum_{n=1}^{N} x_n = \sum_{n=1}^{N} x_n y_n \tag{5.32}$$

$$a \sum_{n=1}^{N} x_n + b \sum_{n=1}^{N} 1 = \sum_{n=1}^{N} y_n\,. \tag{5.33}$$

In matrix form,

$$\begin{pmatrix} \sum_{n=1}^{N} x_n^2 & \sum_{n=1}^{N} x_n \\ \sum_{n=1}^{N} x_n & \sum_{n=1}^{N} 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum_{n=1}^{N} x_n y_n \\ \sum_{n=1}^{N} y_n \end{pmatrix}. \tag{5.34}$$

So, the solution of the optimisation problem is given as

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum_{n=1}^{N} x_n^2 & \sum_{n=1}^{N} x_n \\ \sum_{n=1}^{N} x_n & \sum_{n=1}^{N} 1 \end{pmatrix}^{-1} \begin{pmatrix} \sum_{n=1}^{N} x_n y_n \\ \sum_{n=1}^{N} y_n \end{pmatrix} \tag{5.35}$$

$$= \frac{1}{N \sum_{n=1}^{N} x_n^2 - (\sum_{n=1}^{N} x_n)^2} \begin{pmatrix} N \sum_{n=1}^{N} x_n y_n - (\sum_{n=1}^{N} x_n)(\sum_{n=1}^{N} y_n) \\ -(\sum_{n=1}^{N} x_n)(\sum_{n=1}^{N} x_n y_n) + (\sum_{n=1}^{N} x_n^2)(\sum_{n=1}^{N} y_n) \end{pmatrix}. \tag{5.36}$$

This can be simplified further, which is left as an exercise for the reader.

5.6 Summary

The main things covered in this chapter are:

- A simple example to demonstrate why we need to take uncertainty into account in pattern recognition
- A review of some of the main probability concepts that we will need: Independence; Conditional probability; The law of total probability; Bayes' theorem.
- The application of Bayes' theorem to statistical pattern recognition
- The basic idea of optimisation widely used in statistical pattern recognition

5.7 Reading

- Duda, Hart and Stork: Chapter 1, section 2.1
- Bishop: Sections 1.2.1, 1.2.2, 1.2.3

Naive Bayes

Hiroshi Shimodaira[*]

January-March 2017

In the previous chapter we introduced the use of Bayes' Theorem for pattern classification. For a test vector **x**, we estimate the posterior probability $P(C_k|\mathbf{x})$ for each class k. To classify **x** we choose the class with the largest estimated posterior probability. To do this we re-express the posterior probabilities using Bayes' Theorem:

$$\begin{aligned} P(C_k|\mathbf{x}) &= \frac{P(\mathbf{x}|C_k)\,P(C_k)}{P(\mathbf{x})} \\ &\propto P(\mathbf{x}|C_k)\,P(C_k)\,. \end{aligned}$$

Thus for each class we need to provide an estimate of the likelihood $P(\mathbf{x}|C_k)$ and the prior $P(C_k)$.

To estimate the likelihood we need a statistical *model* which can provide a likelihood estimate. In the "fish" example, the likelihood was estimated using a histogram for each class: the likelihood of a fish with length x coming from class k was estimated as the relative frequency of fish of length x estimated from the training data for class k.

Imagine if in addition to the length we also had some other information such as weight and circumference. In this case, the input feature vector for each fish would contain 3 elements: (length, weight, circumference). If we have 20 possible values for each feature, then the number of bins in a histogram of these vectors would be $20^3 = 8000$. In this case 100 examples per class would mean that it is not possible to observe examples for the vast majority of the bins. We would need many more examples to obtain reasonable estimates based on relative frequencies. As the number of feature dimensions increases, the total number of possible feature vectors increases *exponentially*.

This severe difficulty that arises from moving to spaces of higher dimension was termed the *curse of dimensionality* by Richard Bellman in the 1950s. This is a critical problem: it becomes impossible to reliably estimate histograms once we have more than a few dimensions, even if we have millions of examples.

In this chapter we shall look at an approach to the problem called the Naive Bayes approximation. After showing how it works in a small, fictitious example, we'll go on to see how this approach may be applied to the problem of text classification, which uses high-dimensional feature vectors (e.g., 10^4- to 10^7-dimensional).

[*]©2014-2017 University of Edinburgh. All rights reserved. This note is heavily based on notes inherited from Steve Renals and Iain Murray.

6.1 The Naive Bayes assumption

One way to deal with the curse of dimensionality is to assume that the different feature dimensions are *independent*.[1] If we are using histograms of relative frequencies to estimate likelihoods, then this means that we construct, for each class, D 1-dimensional histograms, rather than a single D-dimensional histogram. If we assume that each dimension can take M values, we now only need to estimate $D \times M$ relative frequencies, rather than M^D relative frequencies!

Consider a D-dimensional feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_D)$. We write the probability of a data point **x** given class k as a joint distribution of the D components of **x** (Equation (6.1)). We can re-express any joint distribution as a product of conditional distributions using successive applications of the product rule (Equation (6.2)). The result is known as the chain rule of probability, which in this case is:

$$\begin{aligned} P(\mathbf{x}|C_k) &= P(x_1, x_2, \ldots, x_D|C_k) && (6.1) \\ &= P(x_1|x_2, \ldots, x_D, C_k)\,P(x_2|x_3, \ldots, x_D, C_k)\ldots P(x_{D-1}|x_D, C_k)\,P(x_D|C_k)\,. && (6.2) \end{aligned}$$

This decomposition in itself doesn't address the curse of dimensionality, since the first term on the right-hand side of (6.2) is conditioned on $(D-1)$ terms, the second on $(D-2)$ terms, and so on.

However we can simplify things if we *naively* assume that the individual feature dimensions $(x_1, x_2, \ldots, x_D)$ are independent, that is:

$$\begin{aligned} P(x_1|x_2, \ldots, x_D, C_k) &= P(x_1|C_k) \\ P(x_2|x_3, \ldots, x_D, C_k) &= P(x_2|C_k) \\ &\vdots \end{aligned}$$

This is called the *Naive Bayes* assumption. The assumption is drastic and rarely true: for example, in the above case Naive Bayes states that the length of a male fish is independent of its weight and circumference. However making this approximation allows us to have a much simpler form for the likelihood, since (6.2) is simplified to:

$$P(\mathbf{x}|C_k) = P(x_1|C_k)\,P(x_2|C_k)\ldots P(x_D|C_k) = \prod_{d=1}^{D} P(x_d|C_k)\,. \qquad (6.3)$$

We have approximated the probability of a D-dimensional feature vector as a product of D probabilities of the 1-dimensional feature vectors.

Using this assumption, we can express Bayes' theorem as follows:

$$\begin{aligned} P(C_k|\mathbf{x}) &= \frac{P(\mathbf{x}|C_k)\,P(C_k)}{P(\mathbf{x})} && (6.4) \\ &= \frac{\prod_{d=1}^{D} P(x_d|C_k)\,P(C_k)}{\prod_{d=1}^{D} P(x_d)} \;\propto\; P(C_k)\prod_{d=1}^{D} P(x_d|C_k) && (6.5) \end{aligned}$$

6.2 Example

The following (fictitious) example comes from the book *Data Mining* by Witten and Frank.

[1]We normally assume that features are 'conditionally independent' – a relaxed version of independence, in which features are mutually independent in the class.

Consider a game which is played or not depending on the weather conditions: outlook (sunny / overcast / rainy), temperature (hot / mild / cool), humidity (high / normal) and windy (true / false). The input variable is 4-dimensional, with 2 or 3 values per dimension. There are 36 possible combinations ($3 \times 3 \times 2 \times 2$).

We have an input training set of 14 examples, shown in table 6.1. As is usually the case in machine learning, there are fewer training examples than possible settings of the input variables; most possible conditions are not directly observed. We can tabulate the data in terms of the frequencies for each of the four input variables (table 6.2), and in terms of the relative frequencies (table 6.3). The relative frequencies can be used as probability estimates, for example $P(T=h \mid \text{Play}=Y) = 2/9$.

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	NO
sunny	hot	high	true	NO
overcast	hot	high	false	YES
rainy	mild	high	false	YES
rainy	cool	normal	false	YES
rainy	cool	normal	true	NO
overcast	cool	normal	true	YES
sunny	mild	high	false	NO
sunny	cool	normal	false	YES
rainy	mild	normal	false	YES
sunny	mild	normal	true	YES
overcast	mild	high	true	YES
overcast	hot	normal	false	YES
rainy	mild	high	true	NO

Table 6.1: Training data for the weather example

Outlook	Y	N
sunny	2	3
overcast	4	0
rainy	3	2

Temperature	Y	N
hot	2	2
mild	4	2
cool	3	1

Humidity	Y	N
high	3	4
normal	6	1

Windy	Y	N
false	6	2
true	3	3

Table 6.2: Play counts for different weather conditions.

Outlook	Y	N
sunny	2/9	3/5
overcast	4/9	0/5
rainy	3/9	2/5

Temperature	Y	N
hot	2/9	2/5
mild	4/9	2/5
cool	3/9	1/5

Humidity	Y	N
high	3/9	4/5
normal	6/9	1/5

Windy	Y	N
false	6/9	2/5
true	3/9	3/5

Table 6.3: Play relative frequencies for different weather conditions. There was play on 9/14 cases.

We are given the following test example:

Outlook	Temp.	Humidity	Windy	Play
sunny	cool	high	true	?

This example's feature vector, **x**, was not observed in the training set. We can generalise from the other examples by using Naive Bayes:

$$\begin{aligned}
P(\text{play}=Y \mid \mathbf{x}) &\propto P(\text{play}=Y) \cdot P(O=s \mid \text{play}=Y) \cdot P(T=c \mid \text{play}=Y) \\
&\quad \cdot P(H=h \mid \text{play}=Y) \cdot P(W=t \mid \text{play}=Y) \\
&= \frac{9}{14} \cdot \frac{2}{9} \cdot \frac{3}{9} \cdot \frac{3}{9} \cdot \frac{3}{9} \\
&\approx 0.0053 \\
P(\text{play}=N \mid \mathbf{x}) &\propto P(\text{play}=N) \cdot P(O=s \mid \text{play}=N) \cdot P(T=c \mid \text{play}=N) \\
&\quad \cdot P(H=h \mid \text{play}=N) \cdot P(W=t \mid \text{play}=N) \\
&= \frac{5}{14} \cdot \frac{3}{5} \cdot \frac{1}{5} \cdot \frac{4}{5} \cdot \frac{3}{5} \\
&\approx 0.0206
\end{aligned}$$

And the ratio of posterior probabilities is:

$$\frac{P(\text{play}=Y \mid \mathbf{x})}{P(\text{play}=N \mid \mathbf{x})} = \frac{0.0053}{0.0206} \approx 0.26$$

Hence we classify **x** as play $=N$.

6.3 Conclusion

In this chapter we have introduced the Naive Bayes approximation. When we have a multidimensional feature vector, Naive Bayes approximated the likelihoods by considering each feature dimension to be independent:

$$P(x_1, x_2, \ldots, x_D|C) \approx P(x_1|C) \cdot P(x_2|C) \cdots P(x_D|C)\,,$$

which approximates a D-dimensional distribution as D 1-dimensional distributions. If each dimension can take M different values, we need only estimate $M \times D$ probabilities, rather than M^D probabilities.

Text Classification using Naive Bayes

Hiroshi Shimodaira[*]

January-March 2017

Text classification is the task of classifying documents by their content: that is, by the words of which they are comprised. Perhaps the best-known current text classification problem is email *spam filtering*: classifying email messages into spam and non-spam (ham).

7.1 Document models

Text classifiers often don't use any kind of deep representation about language: often a document is represented as a *bag of words*. (A bag is like a set that allows repeating elements.) This is an extremely simple representation: it only knows which words are included in the document (and how many times each word occurs), and throws away the word order!

Consider a document $\mathcal{D}$, whose class is given by C. In the case of email spam filtering there are two classes $C = S$ (spam) and $C = H$ (ham). We classify $\mathcal{D}$ as the class which has the highest posterior probability $P(C \,|\, \mathcal{D})$, which can be re-expressed using Bayes' Theorem:

$$P(C \,|\, \mathcal{D}) = \frac{P(\mathcal{D}|C)\, P(C)}{P(\mathcal{D})} \propto P(\mathcal{D}|C)\, P(C)\,. \tag{7.1}$$

We shall look at two probabilistic models of documents, both of which represent documents as a bag of words, using the Naive Bayes assumption. Both models represent documents using feature vectors whose components correspond to word types. If we have a vocabulary V, containing $|V|$ word types, then the feature vector dimension $D = |V|$.

Bernoulli document model: a document is represented by a feature vector with binary elements taking value 1 if the corresponding word is present in the document and 0 if the word is not present.

Multinomial document model: a document is represented by a feature vector with integer elements whose value is the frequency of that word in the document.

Example: Consider the vocabulary:

$$V = \{blue, red, dog, cat, biscuit, apple\}\,.$$

In this case $|V| = D = 6$. Now consider the (short) document "the blue dog ate a blue biscuit". If $\mathbf{d}^B$ is the Bernoulli feature vector for this document, and $\mathbf{d}^M$ is the multinomial feature vector, then we

[*]©2014-2017 University of Edinburgh. All rights reserved. This note is heavily based on notes inherited from Steve Renals and Iain Murray.

would have:

$$\mathbf{d}^B = (1, 0, 1, 0, 1, 0)^T$$
$$\mathbf{d}^M = (2, 0, 1, 0, 1, 0)^T$$

To classify a document we use Equation (7.1), which requires estimating the likelihoods of the document given the class, $P(\mathcal{D}|C)$ and the class prior probabilities $P(C)$. To estimate the likelihood, $P(\mathcal{D}|C)$, we use the Naive Bayes assumption applied to whichever of the two document models we are using.

7.2 The Bernoulli document model

As mentioned above, in the Bernoulli model a document is represented by a binary vector, which represents a point in the space of words. If we have a vocabulary V containing a set of $|V|$ words, then the t'th element of a document vector corresponds to word w_t in the vocabulary. Let $\mathbf{b}$ be the feature vector for the document $\mathcal{D}$; then the t'th element of $\mathbf{b}$, written b_t, is either 0 or 1 representing the absence or presence of word w_t in the document.

Let $P(w_t|C_k)$ be the probability of word w_t occurring in a document of class k; the probability of w_t not occurring in a document of this class is given by $(1-P(w_t|C_k))$. If we make the naive Bayes assumption, that the probability of each word occurring in the document is independent of the occurrences of the other words, then we can write the document likelihood $P(\mathcal{D} \,|\, C)$ in terms of the individual word likelihoods $P(w_t|C_k)$:

$$P(\mathcal{D}|C_k) \;=\; P(\mathbf{b}|C_k) = \prod_{t=1}^{|V|} \left[b_t\, P(w_t|C_k) + (1-b_t)\,(1-P(w_t|C_k))\right]\,. \tag{7.2}$$

This product goes over all words in the vocabulary. If word w_t is present, then $b_t = 1$ and the required probability is $P(w_t \,|\, C_k)$; if word w_t is not present, then $b_t = 0$ and the required probability is $1 - P(w_t|C_k)$. We can imagine this as a model for generating document feature vectors of class k, in which the document feature vector is modelled as a collection of $|V|$ weighted coin tosses, the t th having a probability of success equal to $P(w_t|C_k)$.

The *parameters* of the likelihoods are the probabilities of each word given the document class $P(w_t|C_k)$; the model is also parameterised by the prior probabilities, $P(C_k)$. We can learn (estimate) these parameters from a training set of documents labelled with class k. Let $n_k(w_t)$ be the number of documents of class k in which w_t is observed; and let N_k be the total number of documents of that class. Then we can estimate the parameters of the word likelihoods as:[1]

$$\hat{P}(w_t \mid C_k) = \frac{n_k(w_t)}{N_k}\,, \tag{7.3}$$

the relative frequency of documents of class k that contain word w_t. If there are N documents in total in the training set, then the prior probability of class k may be estimated as the relative frequency of documents of class k:

$$\hat{P}(C_k) = \frac{N_k}{N}\,. \tag{7.4}$$

Thus given a training set of documents (each labelled with a class), and a set of K classes, we can estimate a Bernoulli text classification model as follows:

[1]We sometimes use $\hat{P}(\,)$ to explicitly denote the estimated probability as opposed to the (theoretical) true one, $P(\,)$.

1. Define the vocabulary V; the number of words in the vocabulary defines the dimension of the feature vectors

2. Count the following in the training set:

   - N the total number of documents
   - N_k the number of documents labelled with class k, for $k=1,\ldots,K$
   - $n_k(w_t)$ the number of documents of class k containing word w_t for $k=1,\ldots,K, t=1,\ldots,|V|$

3. Estimate the likelihoods $P(w_t \mid C_k)$ using Equation (7.3)

4. Estimate the priors $P(C_k)$ using Equation (7.4)

To classify an unlabelled document $\mathcal{D}$, we estimate the posterior probability for each class k combining Equation (7.1) and Equation (7.2):

$$\begin{aligned} P(C_k\,|\,\mathbf{b}) &\propto P(\mathbf{b}\,|\,C_k)\,P(C_k) \\ &\propto P(C_k)\prod_{t=1}^{|V|}\left[b_t\,P(w_t|C_k) + (1-b_t)\,(1-P(w_t|C_k))\right]. \end{aligned} \tag{7.5}$$

Example

Consider a set of documents, each of which is related either to *Sports* (S) or to *Informatics* (I). Given a training set of 11 documents, we would like to estimate a Naive Bayes classifier, using the Bernoulli document model, to classify unlabelled documents as S or I.

We define a vocabulary of eight words:

$$V = \begin{bmatrix} w_1 = \text{goal}, \\ w_2 = \text{tutor}, \\ w_3 = \text{variance}, \\ w_4 = \text{speed}, \\ w_5 = \text{drink}, \\ w_6 = \text{defence}, \\ w_7 = \text{performance}, \\ w_8 = \text{field} \end{bmatrix}$$

Thus each document is represented as an 8-dimensional binary vector.

The training data is presented below as a matrix for each class, in which each row represents an 8-dimensional document vector

$$\mathbf{B}^{\text{Sport}} = \begin{pmatrix} 1&0&0&0&1&1&1&1 \\ 0&0&1&0&1&1&0&0 \\ 0&1&0&1&0&1&1&0 \\ 1&0&0&1&0&1&0&1 \\ 1&0&0&0&1&0&1&1 \\ 0&0&1&1&0&0&1&1 \end{pmatrix}$$

$$\mathbf{B}^{\text{Inf}} = \begin{pmatrix} 0&1&1&0&0&0&1&0 \\ 1&1&0&1&0&0&1&1 \\ 0&1&1&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&1&0&1&0&1&0 \end{pmatrix}$$

Classify the following into Sports or Informatics using a Naive Bayes classifier.

1. $\mathbf{b}_1 = (1,0,0,1,1,1,0,1)^T$
2. $\mathbf{b}_2 = (0,1,1,0,1,0,1,0)^T$

Solution:

The total number of documents in the training set $N=11$; $N_S=6$, $N_I=5$.

Using (7.4), we can estimate the prior probabilities from the training data as:

$$\hat{P}(S) = \frac{6}{11}; \qquad \hat{P}(I) = \frac{5}{11}$$

The document count $n_k(w)$ in the training data set, and estimated word likelihoods using (7.3) are given as:

	$n_S(w)$	$\hat{P}(w\|S)$	$n_I(w)$	$\hat{P}(w\|I)$
w_1	3	3/6	1	1/5
w_2	1	1/6	3	4/5
w_3	2	2/6	3	3/5
w_4	3	3/6	1	1/5
w_5	3	3/6	1	1/5
w_6	4	4/6	1	2/5
w_7	4	4/6	3	3/5
w_8	4	4/6	1	1/5

We use (7.5) to compute the posterior probabilities of the two test vectors and hence classify them.

1. $\mathbf{b}_1 = (1,0,0,1,1,1,0,1)^T$

$$\begin{aligned} \hat{P}(S\,|\,\mathbf{b}_1) &\propto \hat{P}(S)\prod_{t=1}^{8}\left[b_{1t}\hat{P}(w_t|S) + (1-b_{1t})(1-\hat{P}(w_t|S))\right] \\ &\propto \frac{6}{11}\left(\frac{1}{2}\times\frac{5}{6}\times\frac{2}{3}\times\frac{1}{2}\times\frac{1}{2}\times\frac{2}{3}\times\frac{1}{3}\times\frac{2}{3}\right) = \frac{5}{891} \approx 5.6\times10^{-3} \\ \hat{P}(I\,|\,\mathbf{b}_1) &\propto \hat{P}(I)\prod_{t=1}^{8}\left[b_{1t}\hat{P}(w_t|I) + (1-b_{1t})(1-\hat{P}(w_t|I))\right] \\ &\propto \frac{5}{11}\left(\frac{1}{5}\times\frac{2}{5}\times\frac{2}{5}\times\frac{1}{5}\times\frac{1}{5}\times\frac{1}{5}\times\frac{2}{5}\times\frac{1}{5}\right) = \frac{8}{859375} \approx 9.3\times10^{-6} \end{aligned}$$

Classify this document as S.

2. $\mathbf{b}_2 = (0, 1, 1, 0, 1, 0, 1, 0)^T$

$$\hat{P}(S\,|\,\mathbf{b}_2) \propto \hat{P}(S)\prod_{t=1}^{8}\left[b_{2t}\hat{P}(w_t\,|\,S) + (1-b_{2t})(1-\hat{P}(w_t\,|\,S))\right]$$
$$\propto \frac{6}{11}\left(\frac{1}{2}\times\frac{1}{6}\times\frac{1}{3}\times\frac{1}{2}\times\frac{1}{2}\times\frac{1}{3}\times\frac{2}{3}\times\frac{1}{3}\right) = \frac{12}{42768} \approx 2.8\times10^{-4}$$

$$\hat{P}(I\,|\,\mathbf{b}_1) \propto \hat{P}(I)\prod_{t=1}^{8}\left[b_{1t}\hat{P}(w_t\,|\,I) + (1-b_{1t})(1-\hat{P}(w_t\,|\,I))\right]$$
$$\propto \frac{5}{11}\left(\frac{4}{5}\times\frac{3}{5}\times\frac{3}{5}\times\frac{4}{5}\times\frac{1}{5}\times\frac{4}{5}\times\frac{3}{5}\times\frac{4}{5}\right) = \frac{34560}{4296875} \approx 8.0\times10^{-3}$$

Classify as I.

7.3 The multinomial distribution *(NE)*

Before discussing the multinomial document model, it is important to be familiar with the multinomial distribution.

We first need to be able to count the number of distinct arrangements of a set of items, when some of the items are *indistinguishable*. For example: Using all the letters, how many distinct sequences can you make from the word "Mississippi"? There are 11 letters to permute, but "i" and "s" occur four times and "p" twice. If these letters were distinct (e.g., if they were labelled i_1, i_2, etc.) then there would be 11! permutations. However of these permutations there are 4! that are the same if the subscripts are removed from the "i"s. This means that we can reduce the size of the total sample space by a factor of 4! to take account of the four occurrences of "i". Likewise there is a factor of 4! for "s" and a factor of 2! for "p" (and a factor of 1! for "m"). This gives the total number distinct permutations as:

$$\frac{11!}{4!\,4!\,2!\,1!} = 34650$$

Generally if we have n items of D types, with n_1 of type 1, n_2 of type 2 and n_D of type D (such that $n_1 + n_2 + \ldots + n_D = n$), then the number of distinct permutations is given by:

$$\frac{n!}{n_1!\,n_2!\ldots n_D!}$$

These numbers are called the *multinomial coefficients*.

Now suppose a population contains items of $D \geq 2$ different types and that the proportion of items that are of type t is p_t $(t = 1, \ldots, D)$, with

$$\sum_{t=1}^{D} p_t = 1 \qquad p_t > 0,\ \text{for all } t.$$

Suppose n items are drawn at random (with replacement) and let x_t denote the number of items of type t. The vector $\mathbf{x} = (x_1, \ldots, x_D)^T$ has a *multinomial distribution* with parameters n and $p_1, \ldots, p_D$, defined by:

$$P(\mathbf{x}) = \frac{n!}{x_1!\,x_2!\ldots x_D!}\,p_1^{x_1}p_2^{x_2}\ldots p_D^{x_D}$$
$$= \frac{n!}{\prod_{t=1}^{D} x_t!}\prod_{t=1}^{D} p_t^{x_t}\,. \tag{7.6}$$

The $\prod_{t=1}^{D} p_t^{x_t}$ product gives the probability of one sequence of outcomes with counts $\mathbf{x}$. The multinomial coefficient, counts the number of such sequences that there are.

7.4 The multinomial document model

In the multinomial document model, the document feature vectors capture the frequency of words, not just their presence or absence. Let $\mathbf{x}$ be the multinomial model feature vector for document $\mathcal{D}$. The t'th element of $\mathbf{x}$, written x_t, is the count of the number of times word w_t occurs in document $\mathcal{D}$. Let $n = \sum_t x_t$ be the total number of words in document $\mathcal{D}$.

Let $P(w_t\,|\,C_k)$ again be the probability of word w_t occurring in class k, this time estimated using the word frequency information from the document feature vectors. We again make the naive Bayes assumption, that the probability of each word occurring in the document is independent of the occurrences of the other words. We can then write the document likelihood $P(\mathcal{D}\,|\,C_k)$ as a multinomial distribution (Equation7.6), where the number of draws corresponds to the length of the document, and the proportion of drawing item t is the probability of word type t occurring in a document of class k, $P(w_t\,|\,C_k)$.

$$P(\mathcal{D}\,|\,C) = P(\mathbf{x}\,|\,C_k) = \frac{n!}{\prod_{t=1}^{|V|} x_t!}\prod_{t=1}^{|V|} P(w_t\,|\,C_k)^{x_t}$$
$$\propto \prod_{t=1}^{|V|} P(w_t\,|\,C_k)^{x_t}\,. \tag{7.7}$$

We often won't need the normalisation term $(n!/\prod_t x_t!)$, because it does not depend on the class k. The numerator of the right hand side of this expression can be interpreted as the product of word likelihoods for each word in the document, with repeated words taking part for each repetition.

As for the multinomial model, the parameters of the likelihood are the probabilities of each word given the document class $P(w_t\,|\,C_k)$, and the model parameters also include the prior probabilities $P(C_k)$. To estimate these parameters from a training set of documents, $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$, let z_{ik} be an indicator variable which equals 1 when $\mathcal{D}_i$ belongs to class k, and equals 0 otherwise. If N is again the total number of documents, then we have:

$$\hat{P}(w_t \mid C_k) = \frac{\sum_{i=1}^{N} x_{it}\,z_{ik}}{\sum_{s=1}^{|V|}\sum_{i=1}^{N} x_{is}\,z_{ik}} \tag{7.8}$$

$$= \frac{n_k(w_t)}{\sum_{s=1}^{|V|} n_k(w_s)}\,, \tag{7.9}$$

an estimate of the probability $P(w_t \mid C_k)$ as the relative frequency of w_t in documents of class k with respect to the total number of words in documents of that class, where $n_k(w_t) = \sum_{i=1}^{N} x_{it}\,z_{ik}$.[2]

The prior probability of class k is estimated as before (Equation (7.4).

Thus given a training set of documents (each labelled with a class) and a set of K classes, we can estimate a multinomial text classification model as follows:

1. Define the vocabulary V; the number of words in the vocabulary defines the dimension of the feature vectors.

[2] We previously used the same variable $n_k(w_t)$ in Equation (7.3) for Bernoulli model to denote the number of documents of class k containing word w_t, whereas it now represents the frequency of word w_t in the documents of class k. You should now understand Equation (7.9) is more general than Equation (7.3). The actual difference lies in the fact that the two document models employ different types of feature vector - Bernoulli feature vector and multinomial feature vector.

2. Count the following in the training set:
   - N : the total number of documents
   - N_k : the number of documents labelled with class k, for each class $k=1,\dots,K$
   - x_{it} : the frequency of word w_t in document $\mathcal{D}_i$, computed for every word w_t in V (Alternatively, $n_k(w_t)$: the frequency of word w_t in the documents of class k)
3. Estimate the likelihoods $P(w_t \mid C_k)$ using (7.8) or (7.9).
4. Estimate the priors $P(C_k)$ using (7.4).

To classify an unlabelled document $\mathcal{D}$, we estimate the posterior probability for each class combining (7.1) and (7.7):

$$\begin{aligned} P(C_k|\mathcal{D}) &= P(C_k|\mathbf{x}) \\ &\propto P(\mathbf{x}|C_k)\,P(C_k) \\ &\propto P(C_k)\prod_{t=1}^{|V|} P(w_t|C_k)^{x_t}\,. \end{aligned} \tag{7.10}$$

Unlike the Bernoulli model, words that do not occur in the document (i.e., for which $x_t=0$) do not affect the probability (since $p^0=1$).

Note that we can rewrite the posterior probability in terms of words u which occur in the document:

$$P(C_k|\mathcal{D}) \propto P(C_k)\prod_{j=1}^{\mathrm{len}(\mathcal{D})} P(u_j|C_k) \tag{7.11}$$

where u_j is the j'th word in document $\mathcal{D}$. This form will be preferable to the original one in some applications, which you will see in the following example.

Example

Consider the same example in Section 7.2, except that this time we will use the multinomial model instead of Bernoulli model.

Each document $\mathcal{D}_i$ is now represented as an 8-dimensional row vector $\mathbf{m}_i$; element m_{it} is the frequency of word w_t in document $\mathcal{D}_i$.

The training data is presented below as a matrix for each class, in which each row represents an 8-dimensional document vector.

$$\mathbf{M}^{\text{Sport}} = \begin{pmatrix} 2&0&0&0&1&2&3&1\\ 0&0&1&0&2&1&0&0\\ 0&1&0&1&0&2&1&0\\ 1&0&0&2&0&1&0&1\\ 2&0&0&0&1&0&1&3\\ 0&0&1&2&0&0&2&1 \end{pmatrix}$$

$$\mathbf{M}^{\text{Inf}} = \begin{pmatrix} 0&1&1&0&0&0&1&0\\ 1&2&0&1&0&0&1&1\\ 0&1&1&0&0&2&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&1&0&1&0&1&0 \end{pmatrix}$$

Classify the following test documents into Sports or Informatics.

1. $\mathcal{D}_1 = w_5\ w_1\ w_6\ w_8\ w_1\ w_2\ w_6$
2. $\mathcal{D}_2 = w_3\ w_5\ w_2\ w_7$

Solution:

Let $n_k(w)$ be the frequency of word w in all documents of class k, giving likelihood estimate,

$$\hat{P}(w|S) = \frac{n_S(w)}{\sum_{v\in V} n_S(v)}\,, \quad \hat{P}(w|I) = \frac{n_I(w)}{\sum_{v\in V} n_I(v)}\,,$$

	$n_S(w)$	$\hat{P}(w\|S)$	$n_I(w)$	$\hat{P}(w\|I)$
w_1	5	5/36	1	1/16
w_2	1	1/36	4	4/16
w_3	2	2/36	3	3/16
w_4	5	5/36	1	1/16
w_5	4	4/36	1	1/16
w_6	6	6/36	2	2/16
w_7	7	7/36	3	3/16
w_8	6	6/36	1	1/16

We have now estimated the model parameters.

1. $\mathcal{D}_1 = w_5\ w_1\ w_6\ w_8\ w_1\ w_2\ w_6$

$$\begin{aligned} \hat{P}(\mathcal{D}_1|S) &= \hat{P}(w_5|S)\cdot\hat{P}(w_1|S)\cdot\hat{P}(w_6|S)\cdot\hat{P}(w_8|S)\cdot\hat{P}(w_1|S)\cdot\hat{P}(w_2|S)\cdot\hat{P}(w_6|S) \\ &= \frac{4}{36}\times\frac{5}{36}\times\frac{6}{36}\times\frac{6}{36}\times\frac{5}{36}\times\frac{1}{36}\times\frac{6}{36} = \frac{4\times5^2\times6^3}{36^7} \approx 2.76\times10^{-7} \end{aligned}$$

$$\hat{P}(S|\mathcal{D}_1) \propto \hat{P}(S)\,\hat{P}(\mathcal{D}_1|S) = \frac{6}{11}\cdot\frac{4\times5^2\times6^3}{36^7} \approx 1.50\times10^{-7}$$

$$\begin{aligned} \hat{P}(\mathcal{D}_1|I) &= \hat{P}(w_5|I)\cdot\hat{P}(w_1|I)\cdot\hat{P}(w_6|I)\cdot\hat{P}(w_8|I)\cdot\hat{P}(w_1|I)\cdot\hat{P}(w_2|I)\cdot\hat{P}(w_6|I) \\ &= \frac{1}{16}\times\frac{1}{16}\times\frac{2}{16}\times\frac{1}{16}\times\frac{1}{16}\times\frac{4}{16}\times\frac{2}{16} = \frac{2^4}{16^7} \approx 5.96\times10^{-7} \end{aligned}$$

$$\hat{P}(I|\mathcal{D}_1) \propto \hat{P}(I)\,\hat{P}(\mathcal{D}_1|I) = \frac{5}{11}\cdot\frac{2^4}{16^6} \approx 2.71\times10^{-8}$$

$\hat{P}(S|\mathcal{D}_1) > \hat{P}(I|\mathcal{D}_1)$, thus we classify $\mathcal{D}_1$ as S.

We have not normalised by $\hat{P}(\mathcal{D}_1)$, hence the above are joint probabilities, proportional to the posterior probability. To obtain the posterior:

$$\hat{P}(S|\mathcal{D}_1) = \frac{\hat{P}(S)\,\hat{P}(\mathcal{D}_1|S)}{\hat{P}(S)\,\hat{P}(\mathcal{D}_1|S) + \hat{P}(I)\,\hat{P}(\mathcal{D}_1|I)} = \frac{1.50\times10^{-7}}{1.50\times10^{-7} + 2.71\times10^{-8}} \approx 0.847$$

$$\hat{P}(I|\mathcal{D}_1) = 1 - \hat{P}(S|\mathcal{D}_1) \approx 0.153$$

2. $\mathcal{D}_2 = w_3\ w_5\ w_2\ w_7$

$$\hat{P}(\mathcal{D}_2|S) = \hat{P}(w_3|S)\cdot\hat{P}(w_5|S)\cdot\hat{P}(w_2|S)\cdot\hat{P}(w_7|S)$$

$$= \frac{2}{36}\times\frac{4}{36}\times\frac{1}{36}\times\frac{7}{36} = \frac{2^3\times 7}{36^4} \approx 3.33\times 10^{-5}$$

$$\hat{P}(S|\mathcal{D}_2) \propto \hat{P}(S)\,\hat{P}(\mathcal{D}_2|S) = \frac{6}{11}\cdot\frac{2^3\cdot 7}{36^4} \approx 1.82\times 10^{-5}$$

$$\hat{P}(\mathcal{D}_2|I) = \hat{P}(w_3|I)\cdot\hat{P}(w_5|I)\cdot\hat{P}(w_2|I)\cdot\hat{P}(w_7|I)$$

$$= \frac{3}{16}\times\frac{1}{16}\times\frac{4}{16}\times\frac{3}{16} = \frac{3^2\cdot 4}{16^4} \approx 5.49\times 10^{-4}$$

$$\hat{P}(I|\mathcal{D}_2) \propto \hat{P}(I)\,\hat{P}(\mathcal{D}_2|I) = \frac{5}{11}\cdot\frac{3^2\cdot 4}{16^4} \approx 2.50\times 10^{-4}$$

$\hat{P}(I|\mathcal{D}_2) > \hat{P}(S|\mathcal{D}_2)$, thus we classify $\mathcal{D}_2$ as I.

We have not normalised by $\hat{P}(\mathcal{D}_2)$, hence the above are joint probabilities, proportional to the posterior probability. To obtain the posterior:

$$\hat{P}(S|\mathcal{D}_2) = \frac{\hat{P}(S)\,\hat{P}(\mathcal{D}_2|S)}{\hat{P}(S)\,\hat{P}(\mathcal{D}_2|S) + \hat{P}(I)\,\hat{P}(\mathcal{D}_2|I)} = \frac{1.82\times 10^{-5}}{1.82\times 10^{-5} + 2.50\times 10^{-4}} \approx 0.0679$$

$$\hat{P}(I|\mathcal{D}_2) = 1 - \hat{P}(S|\mathcal{D}_2) \approx 0.932$$

7.5 The Zero Probability Problem

A drawback of relative frequency estimates—Equation (7.8) for the multinomial model—is that zero counts result in estimates of zero probability. This is a bad thing because the Naive Bayes equation for the likelihood (7.7) involves taking a product of probabilities: if any one of the terms of the product is zero, then the whole product is zero. This means that the probability of the document belonging to the class in question is zero—which means it is impossible.

Just because a word does not occur in a document class in the training data does not mean that it cannot occur in any document of that class.

The problem is that Equation (7.8) *underestimates* the likelihoods of words that do not occur in the data. Even if word w is not observed for class k in the training set, we would still like $P(w \mid C_k) > 0$. Since probabilities must sum to 1, if unobserved words have underestimated probabilities, then those words that are observed must have overestimated probabilities. Therefore, one way to alleviate the problem is to remove a small amount of probability allocated to observed events and distribute this across the unobserved events. A simple way to do this, sometimes called *Laplace's law of succession* or *add one smoothing*, adds a count of one to each word type. If there are W word types in total, then Equation (7.8) may be replaced with:

$$P_{\text{Lap}}(w_t \mid C_k) = \frac{1 + \sum_{i=1}^{N} x_{it}\, z_{ik}}{|V| + \sum_{s=1}^{|V|}\sum_{i=1}^{N} x_{is}\, z_{ik}} = \frac{1 + n_k(w_t)}{|V| + \sum_{s=1}^{|V|} n_k(w_s)} \tag{7.12}$$

The denominator was increased to take account of the $|V|$ extra "observations" arising from the "add 1" term, ensuring that the probabilities are still normalised.

Question: The Bernoulli document model can also suffer from the zero probability model. How would you apply add one smoothing in this case?

7.6 Comparing the two models

The Bernoulli and the multinomial document models are both based on a bag of words. However there are a number of differences, which we summarise here:

1. **Underlying model of text:**
   Bernoulli: a document can be thought of as being generated from a multidimensional Bernoulli distribution: the probability of a word being present can be thought of as a (weighted) coin flip with probability $P(w_t|C)$.
   Multinomial: a document is formed by drawing words from a multinomial distribution: you can think of obtaining the next word in the document by rolling a (weighted) $|V|$-sided dice with probabilities $P(w_t|C)$.

2. **Document representation:**
   Bernoulli: binary vector, elements indicating presence or absence of a word.
   Multinomial: integer vector, elements indicating frequency of occurrence of a word.

3. **Multiple occurrences of words:**
   Bernoulli: ignored.
   Multinomial: taken into account.

4. **Behaviour with document length:**
   Bernoulli: best for short documents.
   Multinomial: longer documents are OK.

5. **Behaviour with "the":**
   Bernoulli: since "the" is present in almost every document, $P(\text{“the”}|C) \approx 1.0$.
   Multinomial: since probabilities are based on relative frequencies of word occurrence in a class, $P(\text{“the”}|C) \approx 0.05$.

7.7 Conclusion

In this chapter we have shown how the Naive Bayes approximation can be used for document classification, by constructing distributions over words. The classifiers require a *document model* to estimate $P(\text{document} \mid \text{class})$. We looked at two document models that we can use with the Naive Bayes approximation:

- **Bernoulli document model:** a document is represented by a binary feature vector, whose elements indicate absence or presence of corresponding word in the document.

- **Multinomial document model:** a document is represented by an integer feature vector, whose elements indicate frequency of corresponding word in the document.

Reference: Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*, Cambridge University Press. 2008. (Chapter 13 Text classification and Naive Bayes) `http://www-nlp.stanford.edu/IR-book/`

Gaussians

Hiroshi Shimodaira[*]

January-March 2017

In this chapter we introduce the basics of how to build probabilistic models of continuous-valued data, including the most important probability distribution for continuous data: the Gaussian, or Normal, distribution. We discuss both the univariate Gaussian (the Gaussian distribution for one-dimensional data) and the multivariate Gaussian distribution (the Gaussian distribution for multi-dimensional data).

8.1 Continuous random variables

First we review the concepts of the cumulative distribution function and the probability density function of a continuous random variable.

Many events that we want to model probabilistically are described by real numbers rather than discrete symbols or integers. In this case we must use *continuous random variables*. Some examples of continuous random variables include:

- The sum of two numbers drawn randomly from the interval $0 < x < 1$;
- The length of time for a bus to arrive at the bus-stop;
- The height of a member of a population.

8.1.1 Cumulative distribution function

We will develop the way we model continuous random variables using a simple example.

Consider waiting for a bus, which runs every 30 minutes. We shall make the idealistic assumption that the buses are always exactly on time, thus a bus arrives every 30 minutes. If you are waiting for a bus, but don't know the timetable, then the precise length of time you need to wait is unknown. Let the continuous random variable X denote the length of time you need to wait for a bus.

Given the above information we may assume that X never takes values above 30 or below 0. We can write this as:

$$\begin{aligned} P(X < 0) &= 0 \\ P(0 \le X \le 30) &= 1 \\ P(X > 30) &= 0 \end{aligned}$$

[*]©2014-2017 University of Edinburgh. All rights reserved. This note is heavily based on notes inherited from Steve Renals and Iain Murray.

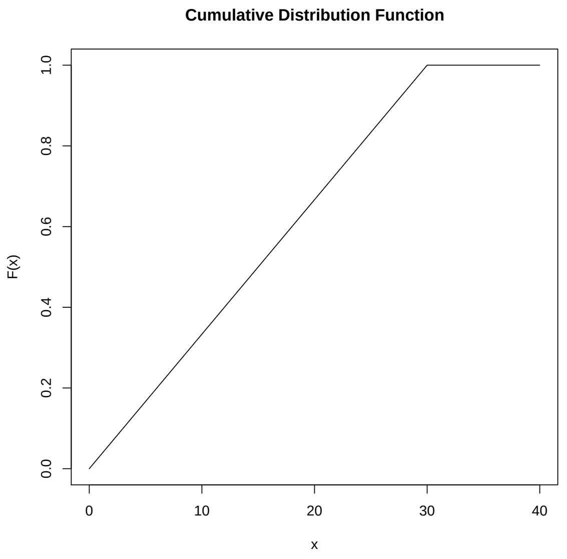


Figure 8.1: Cumulative distribution function of random variable X in the "bus" example.

The probability distribution function for a random variable assigns a probability to each value that the variable may take. It is impossible to write down a probability distribution function for a continuous random variable X, since $P(X=x) = 0$ for all x. This is because X is continuous and can take infinitely many values (and $1/\infty=0$). However we can write down a *cumulative distribution* $F(X)$, which gives the probability of X taking a value that is less than or equal to x. For the current example:

$$F(x) = P(X \le x) = \begin{cases} 0 & x < 0 \\ (x-0)/30 = x/30 & 0 \le x \le 30 \\ 1 & x > 30 \end{cases} \tag{8.1}$$

In writing down this cumulative distribution, we have made the (reasonable) assumption that the probability of a bus arriving increases in proportion to the interval of time waited (in the region 0–30 minutes). This cumulative density function is plotted in Figure 8.1.

Cumulative distribution functions have the following properties:

(i) $F(-\infty) = 0$;

(ii) $F(\infty) = 1$;

(iii) If $a \le b$ then $F(a) \le F(b)$.

To obtain the probability of falling in an interval we can do the following:

$$P(a < X \le b) = P(X \le b) - P(X \le a) = F(b) - F(a). \tag{8.2}$$

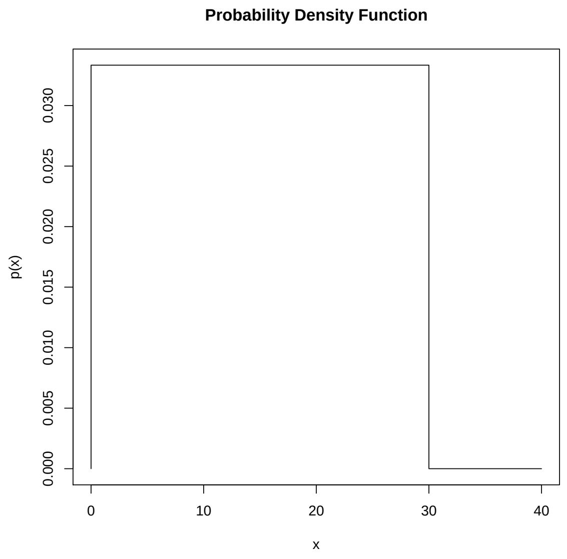


Figure 8.2: Probability density function of random variable X in the "bus" example.

For the "bus" example:

$$\begin{aligned} P(15 < X \leq 21) &= F(21) - F(15) \\ &= 0.7 - 0.5 = 0.2 \end{aligned}$$

8.1.2 Probability density function

Although we cannot define a probability distribution function for a continuous random variable, we can define a closely related function, called the *probability density function* (pdf), $p(x)$:

$$\begin{aligned} p(x) &= \frac{\mathrm{d}}{\mathrm{d}x} F(x) = F'(x) \\ F(x) &= \int_{-\infty}^{x} p(x)\,\mathrm{d}x\,. \end{aligned}$$

The pdf is the gradient of the cdf. Note that $p(x)$ is *not* the probability that X has value x. However, the pdf is proportional to the probability that X lies in a small interval centred on x. The pdf is the usual way of describing the probabilities associated with a continuous random variable X. We usually write probabilities using upper case P and probability densities using lower case p.

We can immediately write down the pdf for the "bus" example:

$$p(x) = \begin{cases} 0 & x < 0 \\ 1/30 & 0 \leq x \leq 30 \\ 0 & x > 30 \end{cases}$$

Figure 8.2 shows a graph of this pdf. X is said to be *uniform* on the interval $(0, 30)$.

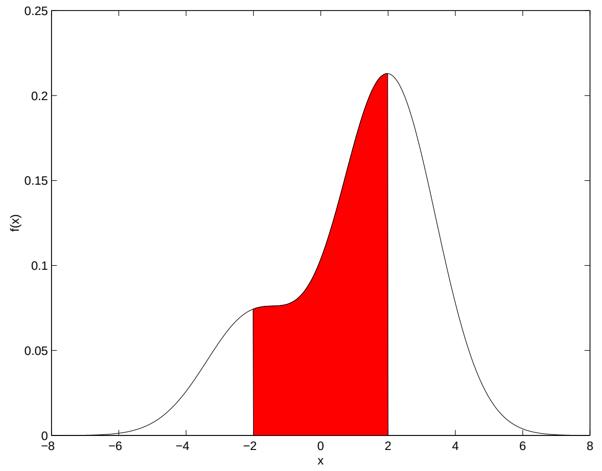


Figure 8.3: $P(-2 < X \leq 2)$ is the shaded area under the pdf.

The probability that the random variable lies in interval (a, b) is the area under the pdf between a and b:

$$\begin{aligned} P(a < X \leq b) &= F(b) - F(a) \\ &= \int_{-\infty}^{b} p(x)\,\mathrm{d}x - \int_{-\infty}^{a} p(x)\,\mathrm{d}x \\ &= \int_{a}^{b} p(x)\,\mathrm{d}x\,. \end{aligned}$$

This integral is illustrated in Figure 8.3. The total area under the pdf equals 1, the probability that x takes on some value between $-\infty$ and ∞. We can also confirm that $F(\infty) - F(-\infty) = 1 - 0 = 1$.

8.2 The Gaussian distribution

The *Gaussian* (or *Normal*) distribution is the most commonly encountered (and easily analysed) continuous distribution. It is also a reasonable model for many situations (the famous "bell curve").

If a (scalar) variable has a Gaussian distribution, then it has a probability density function with this form:

$$p(x|\mu, \sigma^2) = N(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right). \tag{8.3}$$

The Gaussian is described by two parameters:

- the mean μ (location)
- the variance σ^2 (dispersion)

We write $p(x|\mu, \sigma^2)$ because the pdf of x depends on the parameters. Sometimes (to slim down the notation) we simply write $p(x)$.

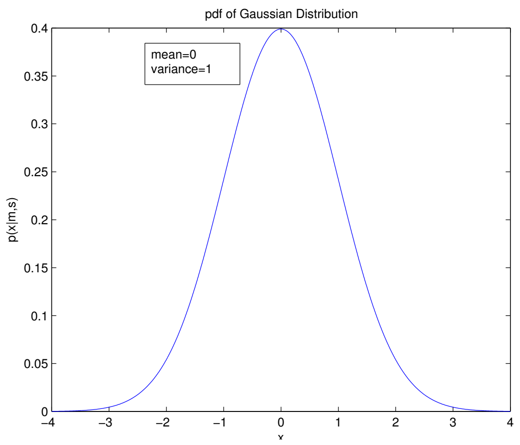


Figure 8.4: One dimensional Gaussian ($\mu = 0$, $\sigma^2 = 1$)

All Gaussians have the same shape, with the location controlled by the mean, and the dispersion (horizontal scaling) controlled by the variance.[1] Figure 8.4 shows a one-dimensional Gaussian with zero mean and unit variance ($\mu = 0$, $\sigma^2 = 1$.)

In (8.3), the mean occurs in the exponential part of the pdf, $\exp(-0.5(x-\mu)^2/\sigma^2)$. This term will have a maximum ($\exp(0) = 1$) when $x = \mu$; thus the peak of the Gaussian corresponds to the mean, and we can think of it as the location parameter.

In one dimension, the variance can be thought of as controlling the width of the Gaussian pdf. Since the area under the pdf must equal 1, this means that the wide Gaussians have lower peaks than narrow Gaussians. This explains why the variance occurs twice in the formula for a Gaussian. In the exponential part $\exp(-0.5(x-\mu)^2/\sigma^2)$, the variance parameter controls the width: for larger values of σ^2, the value of the exponential decreases more slowly as x moves away from the mean. The term $1/\sqrt{2\pi\sigma^2}$ is the *normalisation* constant, which scales the whole pdf to ensure that it integrates to 1. This term decreases with σ^2: hence as σ^2 decreases so the pdf gets a taller (but narrower) peak. The behaviour of the pdf with respect to the variance parameter is shown in Figure 8.5.

[1]To be precise, the width of a distribution scales with its standard deviation, σ, i.e. the square root of the variance.

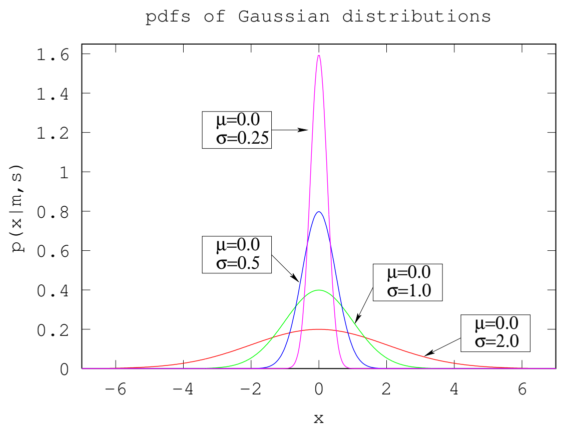


Figure 8.5: Four Gaussian pdfs with zero mean and different standard deviations.

8.3 Parameter estimation

A Gaussian distribution has two parameters the mean (μ) and the variance(σ^2). In machine learning or pattern recognition we are not given the parameters, we have to estimate them from data. As in the case of Naive Bayes we can choose the parameters such that they maximise the likelihood of the model generating the training data. In the case of the Gaussian distribution the *maximum likelihood estimate* (MLE) of the mean and the variance[2] results in:

$$\hat{\mu} = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{8.4}$$

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - \hat{\mu})^2 , \tag{8.5}$$

where x_n denotes the feature value of n'th sample, and N is the number of samples in total.

8.4 Example

A pattern recognition problem has two classes, S and T. Some observations are available for each class:

Class S:	10	8	10	10	11	11
Class T:	12	9	15	10	13	13

We assume that each class may be modelled by a Gaussian. Using the above data, estimate the parameters of the Gaussian pdf for each class, and sketch the pdf for each class.

[2]The maximum likelihood estimate of the variance turns out to be a biased form of the sample variance that is normalised by $N-1$ rather than N.

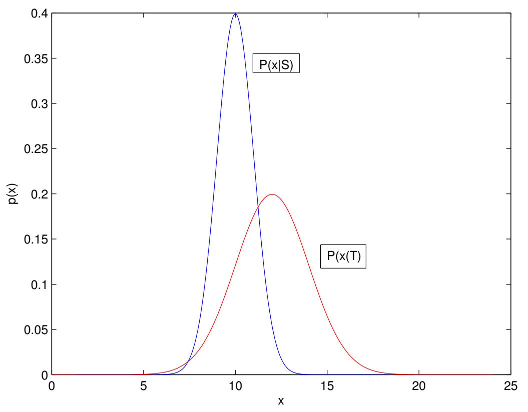


Figure 8.6: Estimated Gaussian pdfs for class S ($\hat{\mu}=10$, $\hat{\sigma}^2=1$) and class class T ($\hat{\mu}=12$, $\hat{\sigma}^2=4$)

The mean and variance of each pdf are estimated with MLE shown in Equation (8.4) and Equation (8.5).

$$\begin{aligned}
\hat{\mu}_S &= \frac{(10+8+10+10+11+11)}{6} = 10 \\
\hat{\sigma}^2_S &= \frac{(10-10)^2+(8-10)^2+(10-10)^2+(10-10)^2+(11-10)^2+(11-10)^2}{6} = 1 \\
\hat{\mu}_T &= \frac{(12+9+15+10+13+13)}{6} = 12 \\
\hat{\sigma}^2_T &= \frac{(12-12)^2+(9-12)^2+(15-12)^2+(10-12)^2+(13-12)^2+(13-12)^2}{6} = 4
\end{aligned}$$

The process of estimating the parameters from the training data is sometimes referred to as *fitting* the distribution to the data.

Figure 8.6 shows the pdfs for each class. The pdf for class T is twice the width of that for class S: the width of a distribution scales with its standard deviation, not its variance.

8.5 The multidimensional Gaussian distribution and covariance

The univariate (one-dimensional) Gaussian may be extended to the multivariate (multi-dimensional) case. The D-dimensional Gaussian is parameterised by a mean vector, $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_D)^T$, and a *covariance matrix*[3], $\boldsymbol{\Sigma} = (\sigma_{ij})$, and has a probability density

$$p(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right). \tag{8.6}$$

The 1-dimensional Gaussian is a special case of this pdf. The covariance matrix gives the variance of each variable (dimension) along the leading diagonal, and the off-diagonal elements measure the

[3]$\boldsymbol{\Sigma}$ is a D-by-D square matrix, and σ_{ij} or Σ_{ij} denotes its element at i'th row and j'th column.

correlations between the variables. The argument to the exponential $\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$ is an example of a *quadratic form*. $|\boldsymbol{\Sigma}|$ is the determinant of the covariance matrix $\boldsymbol{\Sigma}$.

The mean vector $\boldsymbol{\mu}$ is the expectation of $\mathbf{x}$:

$$\boldsymbol{\mu} = E[\mathbf{x}] .$$

The covariance matrix $\boldsymbol{\Sigma}$ is the expectation of the deviation of $\mathbf{x}$ from the mean:

$$\boldsymbol{\Sigma} = E[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T] . \tag{8.7}$$

From (8.7) it follows that $\boldsymbol{\Sigma} = (\sigma_{ij})$ is a $D\times D$ symmetric matrix; that is $\boldsymbol{\Sigma}^T = \boldsymbol{\Sigma}$:

$$\sigma_{ij} = E[(x_i-\mu_i)(x_j-\mu_j)] = E[(x_j-\mu_j)(x_i-\mu_i)] = \sigma_{ji} .$$

Note that σ_{ij} denotes not the standard deviation but the *covariance* between i'th and j'th elements of $\mathbf{x}$. For example, in the 1-dimensional case, $\sigma_{11} = \sigma^2$.

It is helpful to consider how the covariance matrix may be interpreted. The sign of the covariance, σ_{ij}, helps to determine the relationship between two components:

- If x_j is large when x_i is large, then $(x_j-\mu_j)(x_i-\mu_i)$ will tend to be positive;[4]
- If x_j is small when x_i is large, then $(x_j-\mu_j)(x_i-\mu_i)$ will tend to be negative.

If variables are highly correlated (large covariance) then this may indicate that one does not give much extra information if the other is known. If two components of the input vector, x_i and x_j, are statistically independent then the covariance between them is zero, $\sigma_{ij}=0$.

Correlation coefficient The values of the elements of the covariance matrix depend on the unit of measurement: consider the case when x is measured in metres, compared when x is measured in millimetres. It is useful to define a measure of dispersion that is independent of the unit of measurement. To do this we may define the *correlation coefficient*[5] between features x_i and x_j, $\rho(x_i, x_j)$:

$$\rho(x_i,x_j) = \rho_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}} . \tag{8.8}$$

The correlation coefficient ρ_{ij} is obtained by normalising the covariance σ_{ij} by the square root of the product of the variances σ_{ii} and σ_{jj}, and satisfies $-1 \le \rho_{ij} \le 1$:

$$\begin{aligned}
\rho(x,y) &= +1 \qquad \text{if } y = ax+b \quad a>0 \\
\rho(x,y) &= -1 \qquad \text{if } y = ax+b \quad a<0 .
\end{aligned}$$

The correlation coefficient is both scale-independent and location-independent, i.e.:

$$\rho(x_i,x_j) = \rho(ax_i+b, cx_j+d) . \tag{8.9}$$

[4]Note that x_i in this section denotes the i'th element of a vector $\mathbf{x}$ (which is a vector of D random variables) rather than the i'th sample in a data set $\{x_1, \ldots, x_N\}$.

[5]This is normally referred as "Pearson's correlation coefficient", whose another version for sampled data was discussed in Note 2.

8.6 The 2-dimensional Gaussian distribution

Let's look at a two dimensional case, with the following *inverse* covariance matrix[6]:

$$\boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix}^{-1} = \frac{1}{\sigma_{11}\sigma_{22} - \sigma_{12}\sigma_{21}} \begin{pmatrix} \sigma_{22} & -\sigma_{12} \\ -\sigma_{21} & \sigma_{11} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix}.$$

(Remember the covariance matrix is symmetric so $a_{12}=a_{21}$.) To avoid clutter, assume that $\boldsymbol{\mu}=(0,0)^T$, then the quadratic form is:

$$\begin{aligned} \mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x} &= \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ &= \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{12}x_1 + a_{22}x_2 \end{pmatrix} \\ &= a_{11}x_1^2 + 2a_{12}x_1x_2 + a_{22}x_2^2 . \end{aligned}$$

Thus we see that the argument to the exponential expands as a quadratic of D variables ($D=2$ in this case).[7]

In the case of a diagonal covariance matrix:

$$\boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \sigma_{11} & 0 \\ 0 & \sigma_{22} \end{pmatrix}^{-1} = \begin{pmatrix} \frac{1}{\sigma_{11}} & 0 \\ 0 & \frac{1}{\sigma_{22}} \end{pmatrix} = \begin{pmatrix} a_{11} & 0 \\ 0 & a_{22} \end{pmatrix},$$

and the quadratic form has no cross terms:

$$\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x} = a_{11}x_1^2 + a_{22}x_2^2 .$$

In the multidimensional case the normalisation term in front of the exponential is $\frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}}$. Recall that the determinant of a matrix can be regarded as a measure of its size. And the dependence on the dimension reflects the fact that the volume increases with dimension.

Consider a two-dimensional Gaussian with the following mean vector and covariance matrix:

$$\boldsymbol{\mu} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

We refer to this as a *spherical* Gaussian since the probability distribution has spherical (circular) symmetry. The covariance matrix is diagonal (so the off-diagonal correlations are 0), and the variances are equal (1). This pdf is illustrated in the plots of this pdf in Figure 8.7a.

Now consider a two-dimensional Gaussian with the following mean vector and covariance matrix:

$$\boldsymbol{\mu} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix}$$

In this case the covariance matrix is again diagonal, but the variances are not equal. Thus the resulting pdf has an elliptical shape, illustrated in Figure 8.7b.

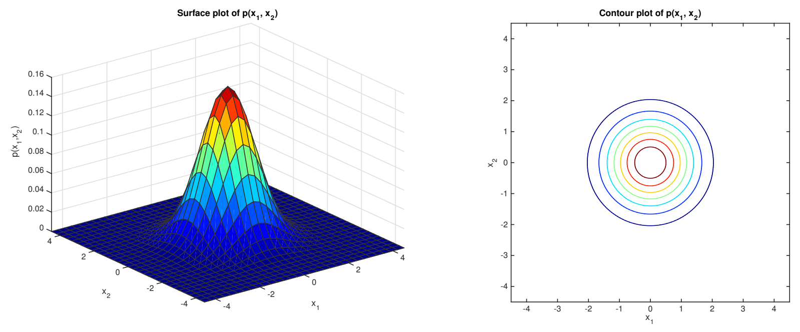
Now consider the following two-dimensional Gaussian:

$$\boldsymbol{\mu} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & -1 \\ -1 & 4 \end{pmatrix}$$

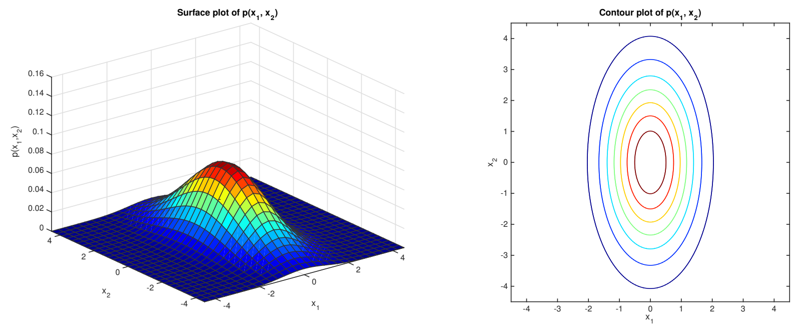
In this case we have a full covariance matrix (off-diagonal terms are non-zero). The resultant pdf is shown in Figure 8.7c.

[6]The inverse covariance matrix is sometimes called the *precision matrix*.

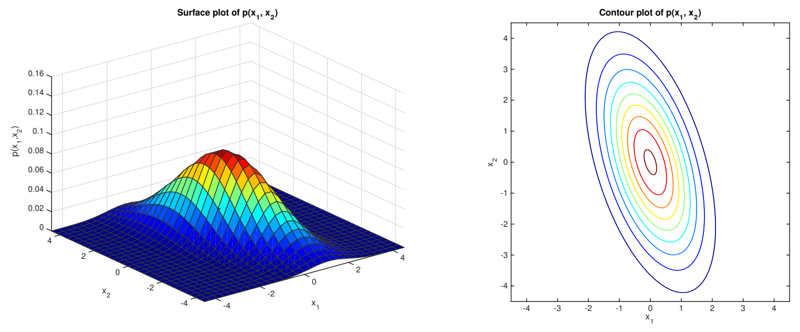
[7]Any covariance matrix is *positive semi-definite*, meaning $\mathbf{x}^T\boldsymbol{\Sigma}\,\mathbf{x} \geq 0$ for any real-valued vector $\mathbf{x}$. The inverse of covariance matrix, if it exists, is also positive semi-definite, i.e., $\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x} \geq 0$.



(a) Spherical Gaussian (diagonal covariance, equal variances)



(b) Gaussian with diagonal covariance matrix



(c) Gaussian with full covariance matrix

Figure 8.7: Surface and contour plots of 2–dimensional Gaussian with different covariance structures

8.7 Parameter estimation

It is possible to show that the mean vector and covariance matrix that maximise the likelihood of the Gaussian generating the training data are given by: [8]

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n \tag{8.10}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{n=1}^{N} (\mathbf{x}_n - \hat{\boldsymbol{\mu}})(\mathbf{x}_n - \hat{\boldsymbol{\mu}})^T . \tag{8.11}$$

Alternatively, in a scalar representation:

$$\hat{\mu}_i = \frac{1}{N}\sum_{n=1}^{N} x_{ni}, \qquad \text{for } i = 1, \ldots, D \tag{8.12}$$

$$\hat{\sigma}_{ij} = \frac{1}{N}\sum_{n=1}^{N} (x_{ni} - \hat{\mu}_i)(x_{nj} - \hat{\mu}_j) \qquad \text{for } i, j = 1, \ldots, D . \tag{8.13}$$

As an example consider the data points displayed in Figure 8.8a. To fit a Gaussian to these samples we compute the mean and variance with MLE. The resulting Gaussian is superimposed as a contour map on the training data in Figure 8.8b.

8.8 Bayes' theorem and Gaussians

Many of the rules for combining probabilities that were outlined at the start of the course, are similar for probability density functions. For example, if x and y are continuous random variables, with probability density functions (pdfs) $p(x)$, and $p(y)$:

$$p(x, y) = p(x \mid y)\, p(y) \tag{8.14}$$

$$p(x) = \int p(x, y)\, \mathrm{d}y , \tag{8.15}$$

where $p(x \mid y)$ is the pdf of x given y.

Indeed we may mix probabilities of discrete variables and probability densities of continuous variables, for example if x is continuous and z is discrete:

$$p(x, z) = p(x \mid z)\, P(z) . \tag{8.16}$$

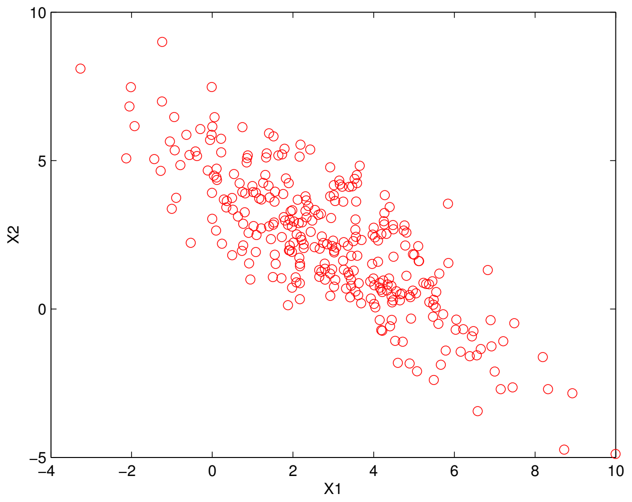
Proving that this is so requires a branch of mathematics called measure theory.

We can thus write Bayes' theorem for continuous data x and discrete class k as:

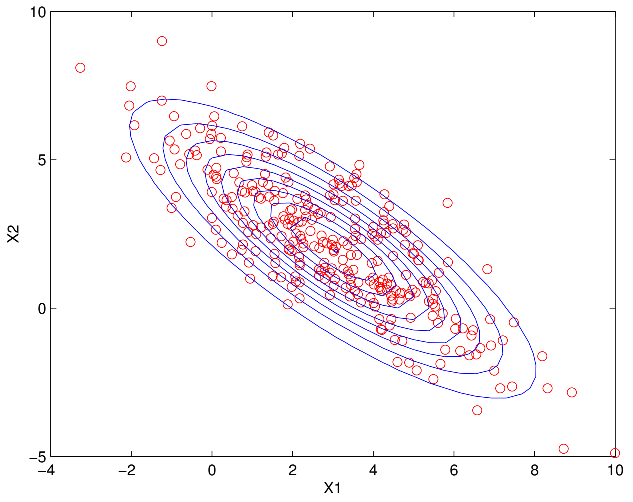
$$\begin{aligned} P(C_k \mid x) &= \frac{p(x \mid C_k)\, P(C_k)}{p(x)} \\ &= \frac{p(x \mid C_k)\, P(C_k)}{\sum_{\ell=1}^{K} p(x \mid C_\ell)\, P(C_\ell)} \qquad (8.17) \end{aligned}$$

$$P(C_k \mid x) \propto p(x \mid C_k)\, P(C_k) \tag{8.18}$$

[8]Again the estimated covariance matrix with MLE is a biased estimator, rather than the unbiased estimator that is normalised by $N-1$.



(a) Training data



(b) Estimated Gaussian

Figure 8.8: Fitting a Gaussian to a set of two-dimensional data samples

The posterior probability of the class given the data is proportional to the probability density of the data times the prior probability of the class.

We can thus use Bayes' theorem for pattern recognition with continuous random variables.

If the pdf of continuous random variable x given class k is represented as a Gaussian with mean μ_k and variance σ_k^2, then we can write: [9]

$$\begin{aligned} P(C_k \mid x) &\propto p(x \mid C_k)\, P(C_k) \\ &\propto N(x\,;\,\mu_k, \sigma_k^2)\, P(C_k) \\ &\propto \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(\frac{-(x-\mu_k)^2}{2\,\sigma_k^2}\right) P(C_k)\,. \end{aligned} \tag{8.19}$$

We call $p(x \mid C_k)$ the likelihood of class k (given the observation x).

8.9 Appendix: Plotting Gaussians with Matlab

`plotgauss1D` is a function to plot a one-dimensional Gaussian with mean `mu` and variance `sigma2`:

```
function plotgauss1D(mu, sigma2)
% plot 1 dimension Gaussian with mean mu and variance sigma2
sd = sqrt(sigma2);  % std deviation
x = mu-3*sd:0.02:mu+3*sd; % location of points at which x is calculated
g = 1/(sqrt(2*pi)*sd)*exp(-0.5*(x-mu).^2/sigma2);
plot(x,g);
```

Recall that the standard deviation (SD) is the square root of the variance. It is a fact that about 0.68 of the probability mass of a Gaussian is within 1 SD (either side) of the mean, about 0.95 is within 2 SDs of the mean, and over 0.99 is within 3 SDs of the mean. Thus plotting a Gaussian for x ranging from $\mu - 3\sigma$ to $\mu + 3\sigma$ captures over 99% of the probability mass, and we take these as the ranges for the plot.

The following Matlab function plots two-dimensional Gaussians as a surface or a contour plot (and was used for the plots in the previous section). We could easily write it to take a (2-dimensional) mean vector and 2x2 covariance matrix, but it can be convenient to write the covariance matrix in terms of variances σ_{jj} and correlation coefficient, ρ_{jk}. Recall that:

$$\rho_{jk} = \frac{\sigma_{jk}}{\sqrt{\sigma_{jj}\sigma_{kk}}}\,. \tag{8.20}$$

Then we may write:

$$\sigma_{jk} = \rho_{jk}\sqrt{\sigma_{jj}}\sqrt{\sigma_{kk}} \tag{8.21}$$

where $\sqrt{\sigma_{jj}}$ is the standard deviation of dimension j. The following code does the job:

```
function plotgauss2D(xmu, ymu, xvar, yvar, rho)

% make a contour plot  and a surface plot of a 2D Gaussian
% xmu, ymu - mean of x, y
```

[9]The suffix k in μ_k and σ_k denotes the class number rather than the k' th element of vector.

```
% xvar, yvar - variance of x, y
% rho correlation coefficient between x and y

xsd = sqrt(xvar);  % std deviation on x axis
ysd = sqrt(yvar);  % std deviation on y axis

if (abs(rho) >= 1.0)
  disp('error: rho must lie between -1 and 1');
  return
end
covxy = rho*xsd*ysd;  % calculation of the covariance

C = [xvar covxy; covxy yvar];  % the covariance matrix
A = inv(C);    % the inverse covariance matrix

% plot between +-2SDs along each dimension
maxsd = max(xsd,ysd);
x = xmu-2*maxsd:0.1:xmu+2*maxsd; % location of points at which x is calculated
y = ymu-2*maxsd:0.1:ymu+2*maxsd; % location of points at which y is calculated

[X, Y] = meshgrid(x,y); % matrices used for plotting

% Compute value of Gaussian pdf at each point in the grid
z = 1/(2*pi*sqrt(det(C))) *
exp(-0.5 * (A(1,1)*(X-xmu).^2 + 2*A(1,2)*(X-xmu).*(Y-ymu) + A(2,2)*(Y-ymu).^2));

surf(x,y,z);
figure;
contour(x,y,z);
```

The above code computes the vectors `x` and `y` over which the function will be plotted. `meshgrid` takes these vectors and forms the set of all pairs (`[X, Y]`) over which the pdf is to be estimated. `surf` plots the function as a surface, and `contour` plots it as a contour map, or plan. You can use the Matlab help to find out more about plotting surfaces.

In the equation for the Gaussian pdf in `plotgauss2D`, because we are evaluating over points in a grid, we write out the quadratic form fully. More generally, if we want to evaluate a D-dimensional Gaussian pdf for a data point **x**, we can use a Matlab function like the following:

```
function y=evalgauss1(mu, covar, x)
% EVALGAUSS1 - evauate a Gaussian pdf

% y=EVALGAUSS1(MU, COVAR, X) evaluates a multivariate Gaussian with
% mean MU and covariance COVAR for a data point X


[d b] = size(covar);

% Check that the covariance matrix is square
if (d ~= b)
```

```
  error('Covariance matrix should be square');
end

% force MU and X into column vectors
mu = reshape(mu, d, 1);
x = reshape(x, d, 1);

% subtract the mean from the data point
x = x-mu;

invcovar = inv(covar);

y = 1/sqrt((2*pi)^d*det(covar)) * exp (-0.5*x'*invcovar*x);
```

However, for efficiency it is usually better to estimate the Gaussian pdfs for a set of data points together. The following function, from the Netlab toolbox, takes an $n \times d$ matrix `x`, where each row corresponds to a data point.

```
function y = gauss(mu, covar, x)
% Y = GAUSS(MU, COVAR, X) evaluates a multi-variate Gaussian  density
% in D-dimensions at a set of points given by the rows of the matrix X.
% The Gaussian density has mean vector MU and covariance matrix COVAR.
%
% Copyright (c) Ian T Nabney (1996-2001)

[n, d] = size(x);
[j, k] = size(covar);

% Check that the covariance matrix is the correct dimension
if ((j ~= d) | (k ~=d))
  error('Dimension of the covariance matrix and data should match');
end

invcov = inv(covar);
mu = reshape(mu, 1, d);    % Ensure that mu is a row vector

x = x - ones(n, 1)*mu;    % Replicate mu and subtract from each data point
fact = sum(((x*invcov).*x), 2);

y = exp(-0.5*fact);

y = y./sqrt((2*pi)^d*det(covar));
```

Check that you understand how this function works. Note that `sum(a,2)` sums along rows of matrix `a` to return a column vector of the row sums. (`sum(a,1)` sums down columns to return a row vector.)

Classification with Gaussians

Hiroshi Shimodaira*

January-March 2017

In the previous chapter we looked at probabilistic models of continuous variables; in particular we introduced the Gaussian (Normal) probability distribution, probably the most important probability distribution for continuous variables.

9.1 Classification

As before we use Bayes' theorem for classification, to relate the probability density function of the data given the class to the posterior probability of the class given the data.

First we consider the univariate case, with a continuous random variable x, whose pdf, given class $C = k$, is a Gaussian with mean μ_k and variance σ_k^2. Using Bayes' theorem we can write:

$$\begin{aligned} P(C_k\,|\,x) = \frac{p(x\,|\,C_k)\,P(C_k)}{p(x)} &\propto p(x\,|\,C_k)\,P(C_k) \\ &\propto N(x;\mu_k,\sigma_k^2)\,P(C_k) \\ &\propto \frac{1}{\sqrt{2\pi\sigma_k^2}}\exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)P(C_k)\,, \end{aligned} \tag{9.1}$$

where $p(x\,|\,C_k)$ is the likelihood of class k given observation x. [1]

Log likelihoods and log probabilities When dealing with Gaussians, it is often useful to take logs, and define $LL(x\,|\,C_k)$ to denote $\ln p(x\,|\,C_k)$. We can write the log likelihood of the Gaussian pdf $\ln p(x\,|\,\mu_k,\sigma_k^2)$:

$$\begin{aligned} LL(x\,|\,\mu_k,\sigma_k^2) &= \ln p(x\,|\,\mu_k,\sigma_k^2) \\ &= \ln\left[\frac{1}{\sqrt{2\pi\sigma_k^2}}\exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)\right] \\ &= -\ln\left(\sqrt{2\pi\sigma_k^2}\right) - \frac{(x-\mu_k)^2}{2\sigma_k^2} \\ &= \frac{1}{2}\left(-\ln(2\pi) - \ln\sigma_k^2 - \frac{(x-\mu_k)^2}{\sigma_k^2}\right). \end{aligned} \tag{9.2}$$

*©2014-2017 University of Edinburgh. All rights reserved. This note is heavily based on notes inherited from Steve Renals and Iain Murray.

[1]As was mentioned in Note 8, although $p(x\,|\,C_k)$ is generally called the class-conditional density function of x, it should be called the likelihood function of C_k when classification is concerned.

We can use Bayes' theorem to write log posterior probability $LP(C_k\,|\,x)$:

$$\begin{aligned} LP(C_k\,|\,x) &= LL(x\,|\,C_k) + LP(C_k) + \text{const.} \\ &= \frac{1}{2}\left(-\ln(2\pi) - \ln\sigma_k^2 - \frac{(x-\mu_k)^2}{\sigma_k^2}\right) + \ln P(C_k) + \text{const.}\,, \end{aligned} \tag{9.3}$$

where $LP(C_k)$ is the log prior probability of class k, and 'const.' is the log of the constant of proportionality, $p(x)$, in Equation (9.1), Bayes' theorem.

Log probability ratio If C_1 and C_2 are modelled by Gaussians with means μ_1 and μ_2, and variances σ_1^2 and σ_2^2, then we can write the log odds (ratio of posterior probabilities) as follows:

$$\ln\frac{P(C_1\,|\,x)}{P(C_2\,|\,x)} = \ln P(C_1\,|\,x) - \ln P(C_2\,|\,x) \tag{9.4}$$

$$= \frac{1}{2}\left(-\ln(2\pi) - \ln\sigma_1^2 - \frac{(x-\mu_1)^2}{\sigma_1^2}\right) - \frac{1}{2}\left(-\ln(2\pi) - \ln\sigma_2^2 - \frac{(x-\mu_2)^2}{\sigma_2^2}\right) + (\ln P(C_1) - \ln P(C_2)) \tag{9.5}$$

$$= -\frac{1}{2}\left(\frac{(x-\mu_1)^2}{\sigma_1^2} - \frac{(x-\mu_2)^2}{\sigma_2^2} + \ln\sigma_1^2 - \ln\sigma_2^2\right) + \ln P(C_1) - \ln P(C_2)\,. \tag{9.6}$$

Look at the right hand side of Equation (9.6). The first two terms are the variance-weighted Euclidean distances from the means, the second two terms are log variances (arising from the normalisation terms) and the third two terms are the log prior probabilities. Think through why, intuitively, each of these terms should affect which class you believe best explains an observation.

9.2 Example: Univariate Gaussian classifier

We return to our previous pattern recognition example in Note 8, a problem with two classes, S and T. Some observations of scalar values are available for each class:

Class S:	10	8	10	10	11	11
Class T:	12	9	15	10	13	13

We assume that each class may be modelled by a Gaussian. The maximum likelihood estimators of the mean and variance of each pdf are:

$$\begin{aligned} \hat{\mu}_S &= 10 \qquad \hat{\sigma}_S^2 = 1 \\ \hat{\mu}_T &= 12 \qquad \hat{\sigma}_T^2 = 4 \end{aligned}$$

The following unlabelled data points are available:

$$x_1 = 10 \qquad x_2 = 11 \qquad x_3 = 6$$

To which class should each of the data points be assigned? Assume the two classes have equal prior probabilities.

Since this is a two class problem, it is convenient to calculate the log posterior probability ratios for each case. (In a multiclass problem, we would calculate the log posterior probability for each class.)

$$\begin{aligned}\ln\frac{P(S \mid X=x)}{P(T \mid X=x)} &= -\frac{1}{2}\left(\frac{(x-\mu_S)^2}{\sigma_S^2} - \frac{(x-\mu_T)^2}{\sigma_T^2} + \ln\sigma_S^2 - \ln\sigma_T^2\right) + \ln P(S) - \ln P(T)\\ &= -\frac{1}{2}\left(\frac{(x-\mu_S)^2}{\sigma_S^2} - \frac{(x-\mu_T)^2}{\sigma_T^2} + \ln\sigma_S^2 - \ln\sigma_T^2\right)\\ &= -\frac{1}{2}\left((x-10)^2 - \frac{(x-12)^2}{4} - \ln 4\right).\end{aligned}$$

If the log ratio is less than 0, then assign to class T, otherwise assign to class S.

- $x_1 = 10$:

$$\begin{aligned}\ln\frac{P(S \mid X=x_1)}{P(T \mid X=x_1)} &= -\frac{1}{2}\left((x_1-10)^2 - \frac{(x_1-12)^2}{4} - \ln 4\right)\\ &= -\frac{1}{2}(0 - 1 - \ln 4)\\ &\approx 1.19\end{aligned}$$

- $x_2 = 11$:

$$\begin{aligned}\ln\frac{P(S \mid X=x_2)}{P(T \mid X=x_2)} &= -\frac{1}{2}\left((x_2-10)^2 - \frac{(x_2-12)^2}{4} - \ln 4\right)\\ &= -\frac{1}{2}(1 - 0.25 - \ln 4)\\ &\approx 0.32\end{aligned}$$

- $x_3 = 6$:

$$\begin{aligned}\ln\frac{P(S \mid X=x_3)}{P(T \mid X=x_3)} &= -\frac{1}{2}\left((x_3-10)^2 - \frac{(x_3-12)^2}{4} - \ln 4\right)\\ &= -\frac{1}{2}(16 - 9 - \ln 4)\\ &\approx -2.81\end{aligned}$$

We assign x_1 to S; x_2 to S; x_3 to T.

Now assume that the two classes do not have equal prior probabilities, in fact $P(S)=0.3$, $P(T)=0.7$. Including this prior information, to which class should each of the above test data points $\{x_1, x_2, x_3\}$ be assigned?

Again compute the log posterior probability ratios:

$$\begin{aligned}\ln\frac{P(S \mid X=x)}{P(T \mid X=x)} &= -\frac{1}{2}\left(\frac{(x-\mu_S)^2}{\sigma_S^2} - \frac{(x-\mu_T)^2}{\sigma_T^2} + \ln\sigma_S^2 - \ln\sigma_T^2\right) + \ln P(S) - \ln P(T)\\ &= -\frac{1}{2}\left((x-10)^2 - \frac{(x-12)^2}{4} - \ln 4\right) + \ln P(S) - \ln P(T)\\ &= -\frac{1}{2}\left((x-10)^2 - \frac{(x-12)^2}{4} - \ln 4\right) + \ln(3/7).\end{aligned}$$

Reclassifying use the prior probability information:

- $x_1 = 10$:

$$\begin{aligned}\ln\frac{P(S \mid X=x_1)}{P(T \mid X=x_1)} &= -\frac{1}{2}\left((x_1-10)^2 - \frac{(x_1-12)^2}{4} - \ln 4\right) + \ln(3/7)\\ &= -\frac{1}{2}(0 - 1 - \ln 4) + \ln(3/7)\\ &\approx 0.34\end{aligned}$$

- $x_2 = 11$:

$$\begin{aligned}\ln\frac{P(S \mid X=x_2)}{P(T \mid X=x_2)} &= -\frac{1}{2}\left((x_2-10)^2 - \frac{(x_2-12)^2}{4} - \ln 4\right) + \ln(3/7)\\ &= -\frac{1}{2}(1 - 0.25 - \ln 4) + \ln(3/7)\\ &\approx -0.53\end{aligned}$$

- $x_3 = 6$:

$$\begin{aligned}\ln\frac{P(S \mid X=x_3)}{P(T \mid X=x_3)} &= -\frac{1}{2}\left((x_3-10)^2 - \frac{(x_3-12)^2}{4} - \ln 4\right) + \ln(3/7)\\ &= -\frac{1}{2}(16 - 9 - \ln 4) + \ln(3/7)\\ &\approx -3.66\end{aligned}$$

We now assign x_1 to S; x_2 to T; x_3 to T.

9.3 Multivariate Gaussian classifier

Now consider D-dimensional data $\mathbf{x}$ from class C modelled using a multivariate Gaussian:

$$\begin{aligned}p(\mathbf{x}|C) = p(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) &= N(\mathbf{x};\boldsymbol{\mu},\boldsymbol{\Sigma})\\ &= \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right).\end{aligned} \tag{9.7}$$

The log likelihood is:

$$LL(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \ln p(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = -\frac{D}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}). \tag{9.8}$$

And we can write the log posterior probability:

$$\ln P(C|\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) - \frac{1}{2}\ln|\boldsymbol{\Sigma}| + \ln P(C) + \text{constant}. \tag{9.9}$$

9.4 Example: Multivariate Gaussian Classifier

Consider the following problem. We have two-dimensional data from three classes (A, B, C). The classes may be assumed to have equal prior probabilities. Our training data is in files `trainA.dat`, `trainB.dat`, and `trainC.dat`, with test data in files `testA.dat`, `testB.dat`,and `testC.dat`.

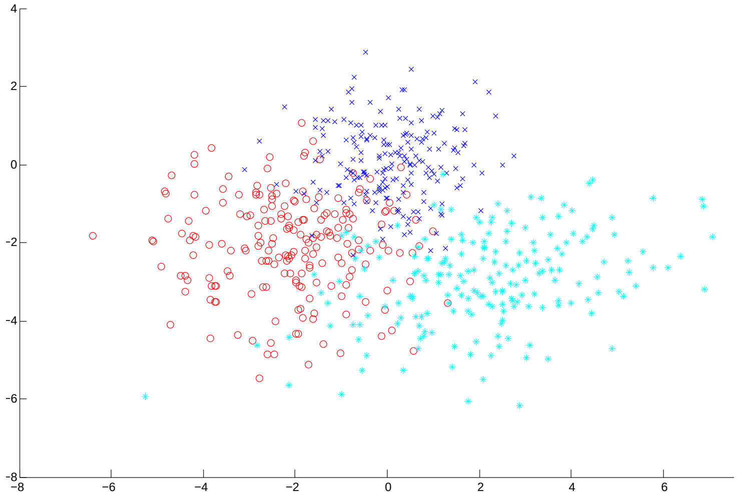


Figure 9.1: Training data from three classes: A (red circles), B (blue crosses) and C (cyan stars)

These files can be downloaded as `http://www.inf.ed.ac.uk/teaching/courses/inf2b/labs/MGC.zip`.

There are 200 points from each class for training, and a further 100 points from each class for testing. The data was generated from Gaussian distributions with the following parameters:

$$\hat{\mu}_A = \begin{pmatrix} -2 \\ -2 \end{pmatrix}, \quad \hat{\Sigma}_A = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix},$$
$$\hat{\mu}_B = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \hat{\Sigma}_B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$
$$\hat{\mu}_C = \begin{pmatrix} 2 \\ -3 \end{pmatrix}, \quad \hat{\Sigma}_C = \begin{pmatrix} 4.0 & 1.0 \\ 1.0 & 1.5 \end{pmatrix}.$$

We can load it into Matlab as follows, then plot the three classes on the same scatter plot (Figure 9.1).

```
% load training and test data
xa = load('trainA.dat');
xb = load('trainB.dat');
xc = load('trainC.dat');
testa = load('testA.dat');
testb = load('testB.dat');
testc = load('testC.dat');
alltest = [testa; testb; testc];

% plot the training data
figure;
hold on;
scatter(xa(:, 1), xa(:,2), 'r', 'o');
```

```
scatter(xb(:, 1), xb(:,2), 'b', 'x');
scatter(xc(:, 1), xc(:,2), 'c', '*');
```

Each row of the matrices `xa`, `xb`, etc. corresponds to a 2-dimensional training data point. To model each class with a Gaussian density, we can immediately estimate the mean and covariance parameters with the sample mean and covariance (computed using the built-in Matlab functions `mean` and `cov`):

```
% train the gaussians
mua = mean(xa);
mub = mean(xb);
muc = mean(xc);

covara = cov(xa);
covarb = cov(xb);
covarc = cov(xc);
```

If there are n data points then `cov(x)` computes the covariance normalising by $1/(n-1)$; `cov(x,1)` normalises by $1/n$.

The resulting estimates for the mean and covariance of each class are as follows:

$$\hat{\mu}_A = \begin{pmatrix} -2.1 \\ -2.1 \end{pmatrix}, \quad \hat{\Sigma}_A = \begin{pmatrix} 2.0 & -0.1 \\ -0.1 & 1.6 \end{pmatrix},$$
$$\hat{\mu}_B = \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix}, \quad \hat{\Sigma}_B = \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 0.9 \end{pmatrix},$$
$$\hat{\mu}_C = \begin{pmatrix} 2.1 \\ -2.9 \end{pmatrix}, \quad \hat{\Sigma}_C = \begin{pmatrix} 4.0 & 0.8 \\ 0.8 & 1.4 \end{pmatrix}.$$

Compare these with the true values above.

We can plot the resulting Gaussians as contour plots over the training data points (Figure 9.2).

Figure 9.3 shows the testing points, labelled with their true classes, together with the Gaussians estimated from the training data.

We can now go ahead and classify each testing point. For test points in class A, we can do the following:

```
testaOut = [gauss(mua,covara,testa) gauss(mub,covarb,testa)
            gauss(muc, covarc, testa)];
[maxaOut, classa] = max(testaOut,[],2);
```

The first line applies each of the three Gaussians to all the test points; the second line determines which class has the highest probability. We can do this for each set of training data. Figures 9.4, 9.5, and 9.6 shows how the test data from each class was classified.

We can look at the results using a *confusion matrix*. The column of a confusion matrix correspond to the predicted classes (i.e., classifier outputs). The rows correspond to the actual (true) class labels. The number at position (r, c) is the number of patterns from true class r that were classified as class c. The number of correctly classified patterns is obtained by summing the numbers on the leading diagonal:

		Predicted class		
Test Data		A	B	C
Actual	A	77	15	8
class	B	5	88	7
	C	9	2	89

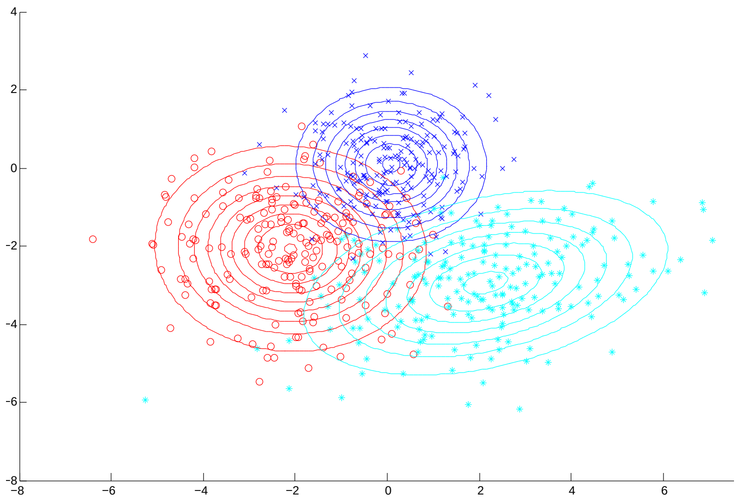


Figure 9.2: Gaussian distributions estimated from training data for each class

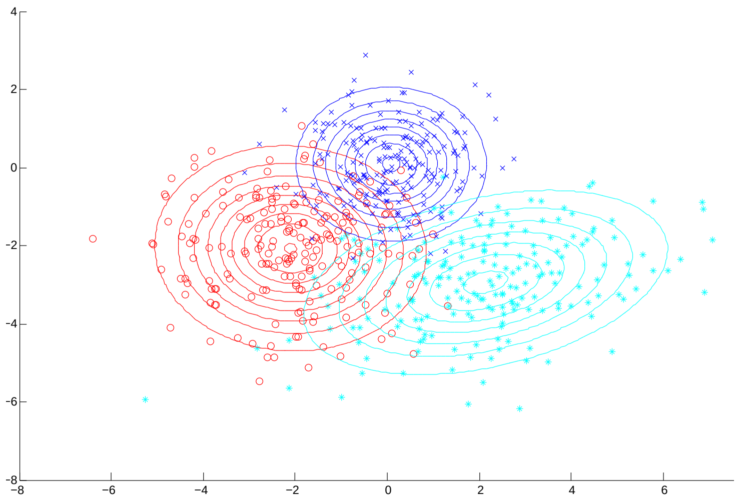


Figure 9.3: Test points, with true class labels, and distributions estimated from training data.

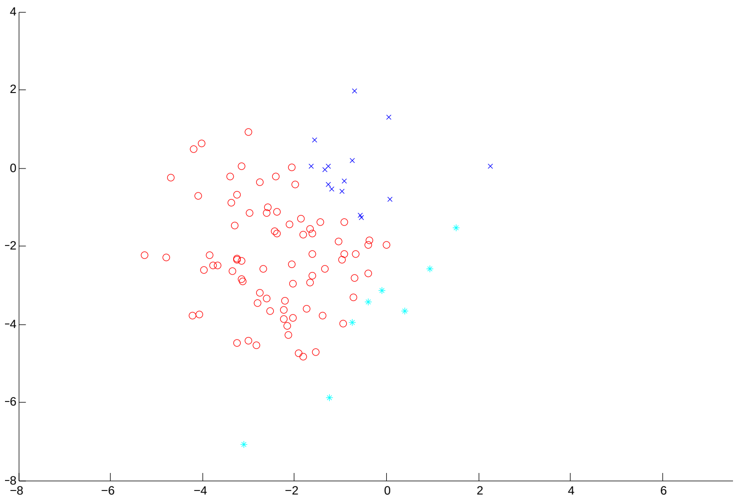


Figure 9.4: Classification of test points from class A

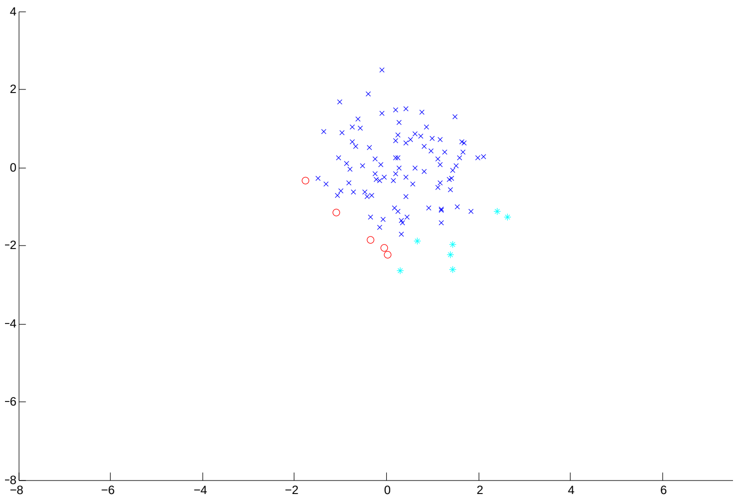


Figure 9.5: Classification of test points from class B

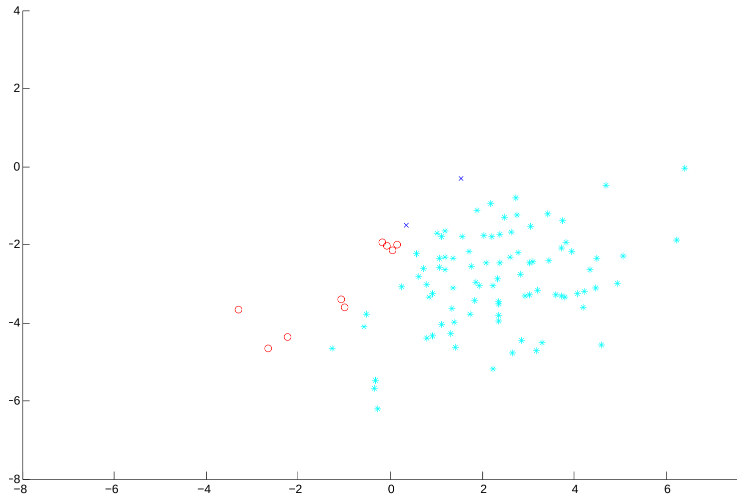


Figure 9.6: Classification of test points from class C

From the confusion matrix we can see that the overall proportion of test patterns correctly classified is $(77 + 88 + 89)/300 = 254/300 \approx 0.85$.

9.5 Application case study

We discuss the use of a multidimensional Gaussian classifier applied in a medical task. The details are as follows:

Task Predict recovery of patients entering hospital with severe head injuries.

Input features 6 categorical variables recorded for each patient: Age, plus 5 responses to stimulation (eye, motor, verbal), graded on scales (e.g., limb movement graded from 1 (nil) to 7 (normal)).

Output classes Three output classes: 1 (death); 2 (severe disability); 3 (good recovery).

Data set 500 patients in both the training and test sets.

Imbalanced classes 50% class 1, 10% class 2, 40% class 3.

Missing data Some feature values in some patterns were missing.

(D. M. Titterington *et al.* (1981), "Comparison of discrimination techniques applied to a complex data set of head-injured patients", *J Roy Stat Soc Ser A*, 144(2), 145–175.)

This problem was tackled by modelling each class using a multivariate Gaussian. Training thus consisted of estimating the mean, covariance matrix and prior probability for each class. The mean vector and covariance matrix for each class was estimated using maximum likelihood (i.e., estimated using sample mean and sample covariance). The prior probabilities were estimated as the relative frequency for each class.

The missing values in the training set filled in with the class mean; since the class is unknown in the test set, the missing values in the test set were filled in with the overall mean.

At test time, each test data point was assigned to the class with the highest posterior probability.

First it is useful to see how well the system performs when tested on the training data. This is the confusion matrix that was obtained:

		Predicted class		
Training Data		1	2	3
Actual	1	209	0	50
class	2	22	1	29
	3	15	1	173

Overall 76.6% ((209 + 1 + 173)/500 = 383/500) of the training patterns were correctly classified.

The following confusion matrix was obtained on the test data:

		Predicted class		
Test Data		1	2	3
Actual	1	188	0	59
class	2	19	1	28
	3	29	2	171

Overall 72.0% ((188 + 1 + 171)/500 = 360/500) of the test patterns were correctly classified.

There are various points which may be noted from this experiment:

- The classification accuracy on the training set was better than the test set, but only a relative improvement of about 6.4%.
- Class two, which corresponded to only 10% of the training set, was rarely selected by the system.
- The baseline strategy, which could be obtained by choosing the class with the highest prior probability (class 1) for each example (i.e., ignoring the data) would achieve about 50% correct.
- The main confusion concerned items from class 1 being misclassified as class 3 (about 10% of all the data in both training and test sets).

Discriminant functions

Hiroshi Shimodaira[*]

January-March 2017

In the previous chapter we saw how we can combine a Gaussian probability density function with class prior probabilities using Bayes' theorem to estimate class-conditional posterior probabilities. For each point in the input space we can estimate the posterior probability of each class, assigning that point to the class with the maximum posterior probability. We can view this process as dividing the input space into decision regions, separated by decision boundaries. In the next section we investigate whether the maximum posterior probability rule is indeed the best decision rule (in terms of minimising the number of errors). In the following sections we introduce discriminant functions which define the decision boundaries, and investigate the form of decision functions induced by Gaussian pdfs with different constraints on the covariance matrix.

10.1 Decision boundaries

We may assign each point in the input space as a particular class. This divides the input space into *decision regions* $\mathcal{R}_k$, such that a point falling in $\mathcal{R}_k$ is assigned to class C_k. In the general case, a decision region $\mathcal{R}_k$ need not be contiguous, but may consist of several disjoint regions each associated with class C_k. The boundaries between these regions are called *decision boundaries*.

Figure 10.1 shows the decision regions that result from assigning each point to the class with the maximum posterior probability, using the Gaussians estimated for classes A, B and C from the example in the previous chapter.

10.1.1 Placement of decision boundaries

Estimating posterior probabilities for each class results in the input space being divided into decision regions, if each point is classified as the class with the highest posterior probability. But is this an optimal placement of decision boundaries?

Consider a 1-dimensional feature space (x) and two classes C_1 and C_2. A reasonable criterion for the placement of decision boundaries is one that *minimises the probability of misclassification*. To estimate the probability of misclassification we need to consider the two ways that a point can be classified wrongly:

1. assigning x to C_1 when it belongs to C_2 (x is in decision region $\mathcal{R}_1$ when it belongs to class C_2);
2. assigning x to C_2 when it belongs to C_1 (x is in $\mathcal{R}_2$ when it belongs to C_1).

[*]©2014-2017 University of Edinburgh. All rights reserved. This note is heavily based on notes inherited from Steve Renals and Iain Murray.

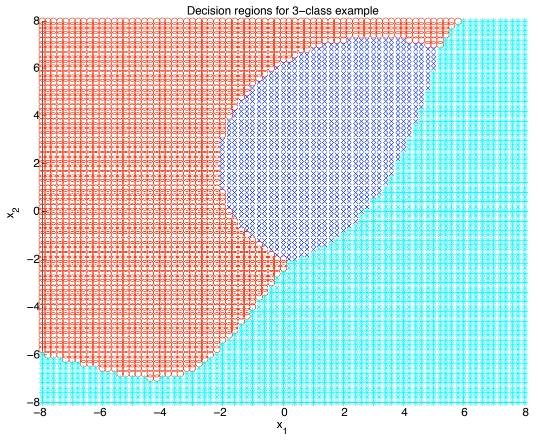


Figure 10.1: Decision regions for the three-class two-dimensional problem from the previous chapter. Class A (red), class B (blue), class C (cyan).

Thus the probability of the total error may be written as:

$$P(\text{error}) = P(x \in \mathcal{R}_2, C_1) + P(x \in \mathcal{R}_1, C_2)\,.$$

Expanding the terms on the right hand side as conditional probabilities, we may write:

$$P(\text{error}) = P(x \in \mathcal{R}_2 \,|\, C_1)\, P(C_1) + P(x \in \mathcal{R}_1 \,|\, C_2)\, P(C_2)\,. \tag{10.1}$$

10.1.2 Overlapping Gaussians

Figure 10.2 illustrates two overlapping Gaussian distributions (assuming equal priors). Two possible decision boundaries are illustrated and the two regions of error are coloured.

We can obtain $P(x \in \mathcal{R}_2 \,|\, C_1)$ by integrating $p(x|C_1)$ within $\mathcal{R}_2$, and similarly for $P(x \in \mathcal{R}_1 \,|\, C_2)$, and thus rewrite (10.1) as:

$$P(\text{error}) = \int_{\mathcal{R}_2} p(x|C_1)\, P(C_1)\, \mathrm{d}x + \int_{\mathcal{R}_1} p(x|C_2)\, P(C_2)\, \mathrm{d}x\,. \tag{10.2}$$

Minimising the probability of misclassification is equivalent to minimising $P(\text{error})$. From (10.2) we can see that this is achieved as follows, for a given x:

- if $p(x|C_1)\, P(C_1) > p(x|C_2)\, P(C_2)$, then point x should be in region $\mathcal{R}_1$;
- if $p(x|C_2)\, P(C_2) > p(x|C_1)\, P(C_1)$, then point x should be in region $\mathcal{R}_2$.

The probability of misclassification is thus minimised by assigning each point to the class with the maximum posterior probability.

It is possible to extend this justification for a decision rule based on the maximum posterior probability to D-dimensional feature vectors and K classes. In this case consider the probability of a pattern being

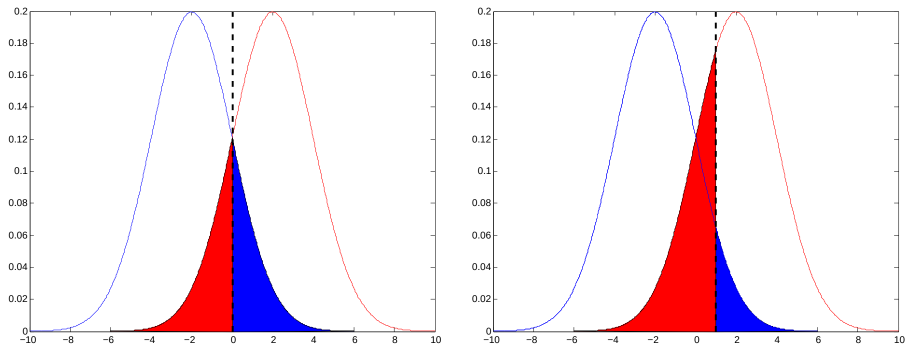


Figure 10.2: Overlapping Gaussian pdfs. Two possible decision boundaries are shown by the dashed line. The decision boundary on the left hand plot is optimal, assuming equal priors. The overall probability of error is given by the area of the shaded regions under the pdfs.

correctly classified:

$$\begin{aligned}P(\text{correct}) &= \sum_{k=1}^{K} P(\mathbf{x} \in \mathcal{R}_k, C_k)\\ &= \sum_{k=1}^{K} P(\mathbf{x} \in \mathcal{R}_k \,|\, C_k)\, P(C_k)\\ &= \sum_{k=1}^{K} \int_{\mathcal{R}_k} p(\mathbf{x}\,|\,C_k)\, P(C_k)\, \mathrm{d}x\,.\end{aligned}$$

This performance measure is maximised by choosing the $\mathcal{R}_k$ such that each $\mathbf{x}$ is assigned to the class k that maximises $p(\mathbf{x}\,|\,C_k)\,P(C_k)$. This procedure is equivalent to assigning each $\mathbf{x}$ to the class with the maximum posterior probability.

Thus the maximum posterior probability decision rule is equivalent to minimising the probability of misclassification. However, to obtain this result we assumed both that the class-conditional models are correct, and that the models are well-estimated from the data.

10.2 Discriminant functions

If we have a set of K classes then we may define a set of K *discriminant functions* $y_k(\mathbf{x})$, one for each class. Data point $\mathbf{x}$ is assigned to class C_k if

$$y_k(\mathbf{x}) > y_\ell(\mathbf{x}) \qquad \text{for all } \ell \neq k.$$

In other words: assign $\mathbf{x}$ to the class C_k whose discriminant function $y_k(\mathbf{x})$ is biggest.

This is precisely what we did in the previous chapter when classifying based on the values of the log posterior probability. Thus the log posterior probability of class C_k given a data point $\mathbf{x}$ is a possible discriminant function:

$$y_k(\mathbf{x}) = \ln P(C_k\,|\,\mathbf{x}) = \ln p(\mathbf{x}\,|\,C_k) + \ln P(C_k) + \text{const.}$$

The posterior probability could also be used as a discriminant function, with the same results: choosing the class with the largest posterior probability is an identical decision rule to choosing the class with the largest log posterior probability.

As discussed above, classifying a point as the class with the largest (log) posterior probability corresponds to the decision rule which minimises the probability of misclassification. In that sense, it forms an optimal discriminant function. A decision boundary occurs at points in the input space where discriminant functions are equal. If the region of input space classified as class C_k and the region classified as class C_ℓ are contiguous, then the decision boundary separating them is given by:

$$y_k(\mathbf{x}) = y_\ell(\mathbf{x})\,.$$

Decision boundaries are not changed by monotonic transformations (such as taking the log) of the discriminant functions.

Formulating a pattern classification problem in terms of discriminant functions is useful since it is possible to estimate the discriminant functions directly from data, without having to estimate probability density functions on the inputs. Direct estimation of the decision boundaries is sometimes referred to as *discriminative* modelling. In contrast, the models that we have considered so far are *generative* models: they could generate new ‘fantasy’ data by choosing a class label, and then sampling an input from its class-conditional model.

10.3 Discriminant functions for class-conditional Gaussians

What is the form of the discriminant function when using a Gaussian pdf? As before, we take the discriminant function as the log posterior probability:

$$\begin{aligned}y_k(\mathbf{x}) &= \ln P(C_k\,|\,\mathbf{x}) = \ln p(\mathbf{x}\,|\,C_k) + \ln P(C_k) + \text{const.}\\ &= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) - \frac{1}{2}\ln|\boldsymbol{\Sigma}_k| + \ln P(C_k)\,. \qquad (10.3)\end{aligned}$$

We have dropped the term $-1/2\ln(2\pi)$, since it is a constant that occurs in the discriminant function for each class. The first term on the right hand side of (10.3) is quadratic in the elements of $\mathbf{x}$ (i.e., if you multiply out the elements, there will be some terms containing x_i^2 or x_ix_j).

10.4 Linear discriminants

Let’s consider the case in which the Gaussian pdfs for each class all share the same covariance matrix. That is, for all classes C_k, $\boldsymbol{\Sigma}_{\boldsymbol{k}} = \boldsymbol{\Sigma}$. In this case $\boldsymbol{\Sigma}$ is class-independent (since it is equal for all classes), therefore the term $-1/2\ln|\boldsymbol{\Sigma}|$ may also be dropped from the discriminant function and we have:

$$y_k(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) + \ln P(C_k)\,.$$

If we explicitly expand the quadratic matrix-vector expression we obtain the following:

$$y_k(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x} - \mathbf{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \boldsymbol{\mu}_k^T\boldsymbol{\Sigma}^{-1}\mathbf{x} + \boldsymbol{\mu}_k^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k) + \ln P(C_k)\,. \qquad (10.4)$$

The mean $\boldsymbol{\mu_k}$ depends on class C_k, but (as stated before) the covariance matrix is class-independent. Therefore, terms that do not include the mean or the prior probabilities are class independent, and may be dropped. Thus we may drop $\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x}$ from the discriminant.

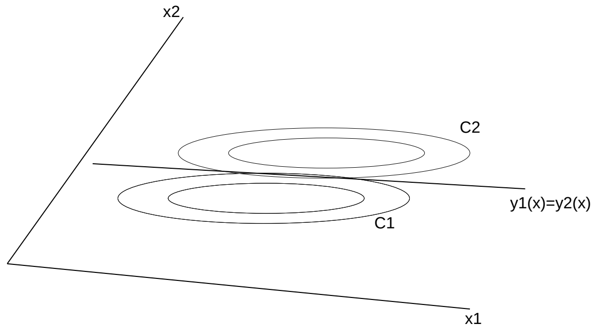


Figure 10.3: Discriminant function for equal covariance Gaussians

We can simplify this discriminant function further. It is a fact that for a symmetric matrix $\mathbf{M}$ and vectors $\mathbf{a}$ and $\mathbf{b}$:

$$\mathbf{a}^T\mathbf{M}\mathbf{b} = \mathbf{b}^T\mathbf{M}\mathbf{a}\,.$$

Now since the covariance matrix $\boldsymbol{\Sigma}$ is symmetric, it follows that $\boldsymbol{\Sigma}^{-1}$ is also symmetric.[1] Therefore:

$$\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k = \boldsymbol{\mu}_k^T\boldsymbol{\Sigma}^{-1}\mathbf{x}\,.$$

We can thus simplify (10.4) as:

$$y_k(\mathbf{x}) = \boldsymbol{\mu}_k^T\boldsymbol{\Sigma}^{-1}\mathbf{x} - \frac{1}{2}\boldsymbol{\mu}_k^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \ln P(C_k)\,. \tag{10.5}$$

This equation has three terms on the right hand side, but only the first depends on $\mathbf{x}$. We can define two new variables $\mathbf{w_k}$ (D-dimension vector) and w_{k0}, which are derived from $\boldsymbol{\mu}_k$, $P(C_k)$, and $\boldsymbol{\Sigma}$:

$$\mathbf{w}_k^T = \boldsymbol{\mu}_k^T\boldsymbol{\Sigma}^{-1} \tag{10.6}$$

$$w_{k0} = -\frac{1}{2}\boldsymbol{\mu}_k^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \ln P(C_k) = -\frac{1}{2}\mathbf{w}_k^T\boldsymbol{\mu}_k + \ln P(C_k)\,. \tag{10.7}$$

Substituting (10.6) and (10.7) into (10.5) we obtain:

$$y_k(\mathbf{x}) = \mathbf{w}_k^T\mathbf{x} + w_{k0}\,. \tag{10.8}$$

This is a linear equation in D dimensions. We refer to $\mathbf{w}_k$ as the *weight vector* and w_{k0} as the *bias* for class C_k.

We have thus shown that the discriminant function for a Gaussian which shares the same covariance matrix with the Gaussians pdfs of all the other classes may be written as (10.8). We call such discriminant functions *linear discriminants*: they are linear functions of $\mathbf{x}$. If $\mathbf{x}$ is two-dimensional, the decision boundaries will be straight lines, illustrated in Figure 10.3. In three dimensions the decision boundaries will be planes. In D-dimensions the decision boundaries are called *hyperplanes*.

10.5 Spherical Gaussians with equal covariance

Let's look at an even more constrained case, where not only do all the classes share a covariance matrix, but that covariance matrix is *spherical*: the off-diagonal terms (covariances) are all zero, and

[1]It also follows that $\mathbf{x}^T\Sigma^{-1}\mathbf{x} \geq 0$ for any $\mathbf{x}$.

the diagonal terms (variances) are equal for all components. In this case the matrix may be defined by a single number, σ^2, the value of the variances:

$$\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$$

$$\boldsymbol{\Sigma}^{-1} = \frac{1}{\sigma^2}\mathbf{I}$$

where $\mathbf{I}$ is the identity matrix.

Since this is a special case of Gaussians with equal covariance, the discriminant functions are linear, and may be written as (10.8). However, we can get another view of the discriminant functions if we write them as:

$$y_k(\mathbf{x}) = -\frac{||\mathbf{x}-\boldsymbol{\mu}_k||^2}{2\sigma^2} + \ln P(C_k)\,. \tag{10.9}$$

If the prior probabilities are equal for all classes, the decision rule simply assigns an unseen vector to the nearest class mean (using the Euclidean distance). In this case the class means may be regarded as class *templates* or *prototypes*.

Exercise: Show that (10.9) is indeed reduced to a linear discriminant.

10.6 Two-class linear discriminants

To get some more insights into linear discriminants, we can look at another special case: two-class problems. Two class problems occur quite often in practice, and they are more straightforward to think about because we are considering a single decision boundary between the two classes.

In the two-class case it is possible to use a single discriminant function: for example one which takes value zero at the decision boundary, negative values for one class and positive values for the other. A suitable discriminant function in this case is the log odds (log ratio of posterior probabilities):

$$\begin{aligned} y(\mathbf{x}) &= \ln\frac{P(C_1|\mathbf{x})}{P(C_2|\mathbf{x})} = \ln\frac{p(\mathbf{x}|C_1)}{p(\mathbf{x}|C_2)} + \ln\frac{P(C_1)}{P(C_2)} \\ &= \ln p(\mathbf{x}|C_1) - \ln p(\mathbf{x}|C_2) + \ln P(C_1) - \ln P(C_2)\,. \end{aligned} \tag{10.10}$$

Feature vector $\mathbf{x}$ is assigned to class C_1 when $y(\mathbf{x}) > 0$; $\mathbf{x}$ is assigned to class C_2 when $y(\mathbf{x}) < 0$. The decision boundary is defined by $y(\mathbf{x}) = 0$.

If the pdf for each class is a Gaussian, and the covariance matrix is shared, then the discriminant function is linear:

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0\,,$$

where $\mathbf{w}$ is a function of the class-dependent means and the class-independent covariance matrix, and the w_0 is a function of the means, the covariance matrix and the prior probabilities.

The decision boundary for the two-class linear discriminant corresponds to a $(D-1)$-dimensional hyperplane in the input space. Let $\mathbf{a}$ and $\mathbf{b}$ be two points on the decision boundary. Then:

$$y(\mathbf{a}) = 0 = y(\mathbf{b})\,.$$

And since $y(\mathbf{x})$ is a linear discriminant:

$$\mathbf{w}^T\mathbf{a} + w_0 = 0 = \mathbf{w}^T\mathbf{b} + w_0.$$

And a little rearranging gives us:

$$\mathbf{w}^T(\mathbf{a}-\mathbf{b}) = 0\,. \tag{10.11}$$

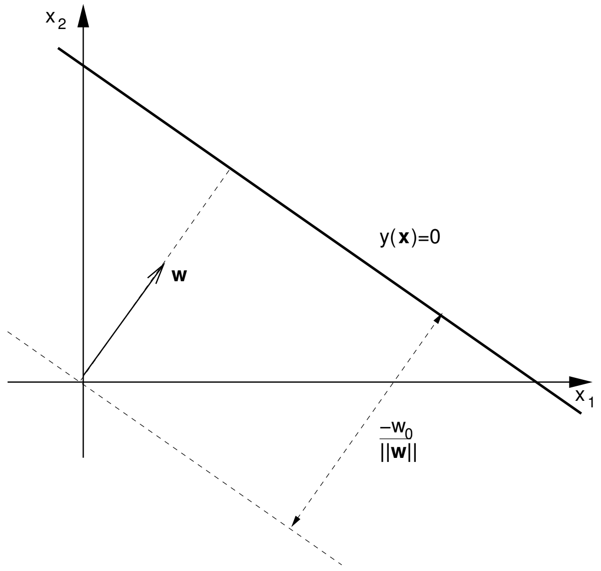


Figure 10.4: Geometry of a two-class linear discriminant

In three dimensions, Equation (10.11) is the equation of a plane, with $\mathbf{w}$ being the vector normal to the plane. In higher dimensions, this equation describes a hyperplane, and $\mathbf{w}$ is normal to any vector lying on the hyperplane. The hyperplane is the decision boundary in this two-class problem.

If $\mathbf{x}$ is a point on the hyperplane, then the normal distance from the hyperplane to the origin is given by:

$$\ell = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||} = -\frac{w_0}{||\mathbf{w}||} \qquad \text{(using } y(\mathbf{x}) = 0\text{)},$$

which is illustrated in Figure 10.4.

Single Layer Neural Networks

Hiroshi Shimodaira[*]

January-March 2017

We have shown that if we have a pattern classification problem in which each class k is modelled by a pdf $p(\mathbf{x}|C_k)$, then we can define discriminant functions $y_k(\mathbf{x})$ which define the decision regions and the boundaries between classes. If the classes have Gaussian pdfs and all share the same covariance matrix, then the discriminant functions are linear.

If we are concerned with performing pattern classification, we do not have to first estimate the pdfs for each class, and then derive the discriminant functions. We can find the discriminant functions directly. We will start by fitting linear discriminant functions. We will then move on to more complex discriminant functions, suitable for use with generative processes that would be hard to model directly. The technology we will use to represent the discriminant functions will be 'neural networks'.

The neural networks that we will consider are simply functions that take a d-dimensional input, and return an output (either a scalar or a vector). These functions have free-parameters, known as 'weights and biases'. Learning a neural network model entails fitting the weight and bias parameters so that the input-output mapping is useful for some task. In these notes, we will learn functions that give low classification error when used as discriminant functions.

11.1 Network representation

Consider the set of linear discriminant functions for a K class problem:

$$\begin{aligned} y_1(\mathbf{x}) &= \mathbf{w}_1^T\mathbf{x} + w_{10} \\ &\vdots \\ y_K(\mathbf{x}) &= \mathbf{w}_K^T\mathbf{x} + w_{K0}\,. \end{aligned} \tag{11.1}$$

This can be converted into a matrix-vector form:

$$\begin{pmatrix} y_1 \\ \vdots \\ y_K \end{pmatrix} = \begin{pmatrix} w_{10} & w_{11} & \dots & w_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K0} & w_{K1} & \dots & w_{Kd} \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{pmatrix}, \tag{11.2}$$

or

$$\mathbf{y} = \mathbf{W}\dot{\mathbf{x}}\,, \tag{11.3}$$

where $y_k = y_k(\mathbf{x})$, $\mathbf{y} = (y_1,\dots,y_K)^T$, $\mathbf{W} = (\dot{\mathbf{w}}_1,\dots,\dot{\mathbf{w}}_K)^T$, $\dot{\mathbf{w}}_k = (w_{k0}, w_{k1},\dots,w_{kd})^T = (w_{k0}, \mathbf{w}_k^T)^T$, and $\dot{\mathbf{x}} = (1, \mathbf{x}^T)^T$, which is an augmented vector of $\mathbf{x}$. Hereafter, we sometimes use $\mathbf{x}$ to denote $\dot{\mathbf{x}}$ for simplicity.

[*]©2014-2017 University of Edinburgh. All rights reserved. This note is heavily based on notes inherited from Steve Renals and Iain Murray.

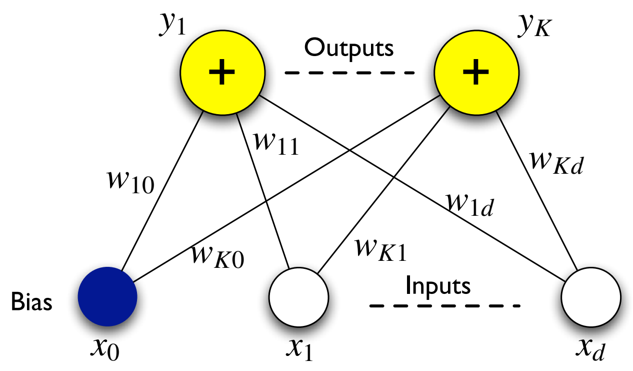


Figure 11.1: Single layer neural network with d-dimensional input vector $\mathbf{x} = (x_1,\dots,x_d)^T$ and output vector $\mathbf{y} = (y_1,\dots,y_K)^T$ corresponding to K classes. The input vector is augmented with an additional variable $x_0 = 1$ which is the bias node. The model is parameterised by the weight matrix $\mathbf{W}$, whose elements w_{ki} are the weights from input x_i to output y_k (and whose rows are the discriminant functions w_k). The bias of output y_k is given by w_{k0}.

We can regard this mapping from $\mathbf{x}$ to $\mathbf{y}$ as a *single-layer neural network*, in which the weight vector for class k connects the input vector ($\mathbf{x}$) to an output corresponding to class k. We can collect the weight vectors together as the rows of a $K \times (d+1)$ weight matrix $\mathbf{W}$, in which element w_{ki} connects input x_i to output y_k. To avoid treating the biases w_{k0} separately, we define an additional input x_0, which always takes the value 1 (hence we have a $(d+1)$ dimension input vector). The bias for class k, w_{k0}, is the weight between x_0 and y_k. The resulting network is illustrated in figure 11.1.

The equation of such a single-layer neural network (in matrix-vector form) is:

$$\mathbf{y} = \mathbf{W}\mathbf{x} \tag{11.4}$$

or, in terms of the individual components:

$$y_k = \sum_{i=0}^{d} w_{ki} x_i\,. \tag{11.5}$$

We can summarise the terminology we have just introduced:

Input vector $\mathbf{x} = (x_0, x_1,\dots,x_d)^T$

Output vector $\mathbf{y} = (y_1,\dots,y_K)^T$

Weight matrix $\mathbf{W}$: w_{ki} is the weight from input x_i to output y_k

11.2 The training problem

How do we train the weight matrix in a single layer neural network defined by equation (11.4)?

The network is trained using a *training set* that contains N input/output pairs $\{(\mathbf{x}_n, \mathbf{t}_n) : 1 \leq n \leq N\}$, where $\mathbf{t}_n = (t_{n1}, \ldots, t_{nK})^T$ is the *target output* vector for input vector $\mathbf{x}_n$. For a classification problem, if the correct class is k, then:

$$\begin{aligned} t_{nk} &= 1 \\ t_{n\ell} &= 0 \qquad \forall \ell \neq k\,. \end{aligned}$$

Representing the target label in a vector in this way is called a "1-from-K" coding.

The single-layer neural network training problem may be stated as follows:

> Given a training set, what values should the elements of the weight matrix $\mathbf{W}$ take, so as to *minimise* an *error function* defined in terms of the outputs $\mathbf{y}_n$?

The error function should measure the distance of the output vectors $\mathbf{y}_n$ from the corresponding target vectors $\mathbf{t}_n$ for all n. A natural way to do this is by taking the (squared) Euclidean distance, and we define the *sum-of-squares* error function which computes the sum of squared Euclidean distances between $\mathbf{t}_n$ and $\mathbf{y}_n$ for all members of the training set $1 \leq n \leq N$. In matrix form we can write:

$$E(\mathbf{W}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{t}_n\|^2\,. \tag{11.6}$$

Or equivalently, we can write the error in terms of the components:

$$\begin{aligned} E(\mathbf{W}) &= \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}(y_{nk} - t_{nk})^2 \\ &= \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}\left(\sum_{i=0}^{d} w_{ki}x_{ni} - t_{nk}\right)^2\,. \end{aligned} \tag{11.7}$$

This error function $E(\mathbf{W})$ is a smooth function of the weights. We train the network parameters $\mathbf{W}$ so as to minimise the value of the error function $E(\mathbf{W})$ given the training set.

To find the minimum of a function we look for the point where its gradient is 0. [1] In this case we look for where the derivative of E with respect to the weights equals 0. Since E is a quadratic function of the weights, the derivatives will be linear functions of the weights. There are two ways of solving the equation to find the optimal weight matrix:

- An exact approach (*pseudoinverse* solution)
- Iterative approaches such as
  - Iteratively reweighted least squares (IRLS), which is an application of the Newton–Raphson algorithm
  - Gradient descent

We will consider *gradient descent*: although IRLS is preferable for single-layer neural networks (it is much faster), gradient descent can also be used in more complicated settings (such as training multi-layer neural networks).

[1]See Section 5.5 for gradient and optimisation problem.

11.3 Gradient descent

Gradient descent is an important optimisation technique, that may be used whenever it is possible to compute the derivatives of the error function with respect to the parameter to be optimised. For single layer neural networks, this means taking the derivative of the error function $E(\mathbf{W})$ with respect to the weight matrix $\mathbf{W}$.

The idea of gradient descent is that to minimise an error function with respect to the parameters, we want to take small steps in a *downhill* direction. We take small steps because the gradient is not uniform, and if we take too big a step we may end up going uphill again! When considering this form of optimisation, we are considering another multidimensional space, *weight space*. This is a $K \cdot (d + 1)$ dimension space, and a specific weight matrix $\mathbf{W}$ corresponds to a point in weight space. The error function evaluates the error value for a point in weight space (given the training set).

The gradient of E given $\mathbf{W}$ is written as $\nabla_{\mathbf{W}}E$, the vector of partial derivatives of E with respect to the elements of $\mathbf{W}$:

$$\nabla_{\mathbf{W}}E = \left(\frac{\partial E}{\partial w_{10}}, \ldots, \frac{\partial E}{\partial w_{ki}}, \ldots, \frac{\partial E}{\partial w_{Kd}}\right)^T\,. \tag{11.8}$$

Descending in weight space means adjusting the weight matrix $\mathbf{W}$ by moving a small direction down the gradient, which is the direction along which E decreases most rapidly. This means adjusting the weight factor in the direction of $-\nabla_{\mathbf{W}}E$, or adjusting each weight w_{ki} by adding a factor $-\eta \cdot \partial E/\partial w_{ki}$, where η is a small constant called the *step size* or *learning rate*.

The operation of gradient descent is as follows:

1. Start with a guess for the weight matrix $\mathbf{W}$ (e.g., small randomly chosen weights).
2. Update the weights by adjusting the weight matrix by a small distance in the direction in weight space along which E decreases most rapidly: i.e., in the direction of $-\nabla_{\mathbf{W}}E$.
3. Recompute the error, and goto 2, terminating either after a fixed number of steps, or when the error stops decreasing by more than a threshold.

If we write the value of a weight at iteration τ as w_{ki}^{τ}, then its updated value is given by:

$$w_{ki}^{\tau+1} = w_{ki}^{\tau} - \eta\frac{\partial E}{\partial w_{ki}}\,. \tag{11.9}$$

The learning rate η specifies how much the parameters should be adjusted along the direction of the gradient.

11.4 Gradient descent for a single-layer neural network

To apply gradient descent to a single-layer neural network, in which we minimise E with respect to $\mathbf{W}$, it is necessary to differentiate E (equation (11.7) with respect to each weight w_{ki}:

$$\frac{\partial E}{\partial w_{ki}} = \sum_{n=1}^{N}\left(\sum_{j=0}^{d} w_{kj}x_{nj} - t_{nk}\right)x_{ni} \tag{11.10}$$

$$= \sum_{n=1}^{N}(y_{nk} - t_{nk})\,x_{ni}\,. \tag{11.11}$$

Define δ_{nk} as the difference between the network output and the target for class k [2], considering the n'th example:

$$\delta_{nk} = y_{nk} - t_{nk} . \tag{11.12}$$

We can think of δ_{nk} as the error on output k for the n'th training example.

We may thus write the partial derivatives in terms of the δs:

$$\frac{\partial E}{\partial w_{ki}} = \sum_{n=1}^{N} \delta_{nk} x_{ni} . \tag{11.13}$$

So the derivative for the weight connecting input i to output k is calculated using the product of the error at output k, δ_{nk}, and the input value x_{ni}, summed over all the training set. This derivative is 'local' to the weight concerned: it does not require knowledge of other weights in the weight matrix, nor other elements of the input, output or target vectors.[3]

Combining the expression for the derivatives (11.13) with the expression for gradient descent update (11.9), we obtain:

$$w_{ki}^{\tau+1} = w_{ki}^{\tau} - \eta \sum_{n=1}^{N} \delta_{nk} x_{ni} . \tag{11.14}$$

This is the gradient descent *learning rule* for the weights of a single-layer neural network. (This gradient descent rule is also known as the delta rule, the Widrow–Hoff rule and LMS learning.)

We may write the algorithm for gradient descent training as follows:

```
 1: procedure GRADIENTDESCENTTRAINING($\mathbf{X}, \mathbf{T}, \mathbf{W}$)
 2:     initialise $\mathbf{W}$ to small random numbers
 3:     while not converged do
 4:         for all $k, i$: $\Delta w_{ki} \leftarrow 0$
 5:         for $n \leftarrow 1, N$ do
 6:             for $k \leftarrow 1, K$ do
 7:                 $y_{nk} \leftarrow \sum_{i=0}^{d} w_{ki} x_{ni}$
 8:                 $\delta_{nk} \leftarrow y_{nk} - t_{nk}$
 9:                 for $i \leftarrow 1, d$ do
10:                     $\Delta w_{ki} \leftarrow \Delta w_{ki} + \delta_{nk} \cdot x_{ni}$
11:                 end for
12:             end for
13:         end for
14:         for all $k, i$: $w_{ki} \leftarrow w_{ki} - \eta \cdot \Delta w_{ki}$
15:     end while
16: end procedure
```

11.4.1 The bias parameter

We previously saw a geometric interpretation of the bias parameter for a two-class linear discriminant: the normal distance of the separating hyperplane from the origin. Here we provide another interpretation of the bias.

[2] Note that δ_{nk} here is not the Kronecker delta!

[3] Local computations are convenient and scale well. The locality of computation has also been used to argue for the 'biological plausibility' of neural network models of learning. Although any relation to biological neurons is abstract; real neurons are complicated.

If we set the derivatives to 0, as above, and explicitly consider only the bias parameter, then we can write:

$$\frac{\partial E}{\partial w_{k0}} = \sum_{n=1}^{N} \left(\sum_{j=1}^{d} w_{kj} x_{nj} + w_{k0} - t_{nk} \right) = 0 . \tag{11.15}$$

If we write the average inputs and targets as follows:

$$\bar{x}_i = \frac{1}{N} \sum_{n=1}^{N} x_{ni} \tag{11.16}$$

$$\bar{t}_k = \frac{1}{N} \sum_{n=1}^{N} t_{nk} . \tag{11.17}$$

Then we may write the solution for the bias as

$$\sum_{j=1}^{d} w_{kj} \bar{x}_j + w_{k0} - \bar{t}_k = 0 \tag{11.18}$$

$$w_{k0} = \bar{t}_k - \sum_{j=1}^{d} w_{kj} \bar{x}_j . \tag{11.19}$$

This means that we may interpret the bias as compensating for the difference in the mean of the targets and the network outputs, averaged over the training set.

11.5 Logistic discriminants

We can generalise linear discriminants by applying a nonlinear function to them. Consider a two-class linear discriminant, to which we apply an *activation function*, g:

$$y(\mathbf{x}) = g(\mathbf{w}^T \mathbf{x} + w_0) . \tag{11.20}$$

If g is monotonically increasing, then (11.20) may be regarded as a linear discriminant, since the decision boundary will still be linear. We may represent this as a two-class, single output, single layer neural network (figure 11.2).

For convenience we can define an *activation value a*:

$$a = \mathbf{w}^T \mathbf{x} + w_0 , \tag{11.21}$$

so that

$$y(\mathbf{x}) = g(a) . \tag{11.22}$$

11.5.1 Logistic sigmoid activation function

If we have a two class problem, with classes 1 and 2, then we can express the posterior probability of C_1 using Bayes' theorem:

$$P(C_1 | \mathbf{x}) = \frac{p(\mathbf{x}|C_1)\, P(C_1)}{p(\mathbf{x}|C_1)\, P(C_1) + p(\mathbf{x}|C_2)\, P(C_2)} . \tag{11.23}$$

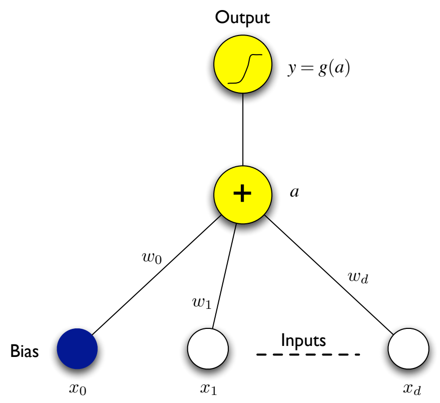


Figure 11.2: Single layer neural network representation of a generalised linear discriminant. In this case the output activation function is a logistic sigmoid.

If we divide top and bottom of the right hand side by $p(\mathbf{x}|C_1)\,P(C_1)$, then we obtain:

$$P(C_1|\mathbf{x}) = \frac{1}{1 + \frac{p(\mathbf{x}|C_2)\,P(C_2)}{p(\mathbf{x}|C_1)\,P(C_1)}}\,. \tag{11.24}$$

If we define a as the ratio of log posterior probabilities (log odds):

$$a = \ln \frac{p(\mathbf{x}|C_1)\,P(C_1)}{p(\mathbf{x}|C_2)\,P(C_2)} \tag{11.25}$$

and substitute into (11.24) we obtain:

$$P(C_1|\mathbf{x}) = g(a) = \frac{1}{1 + \exp(-a)}\,. \tag{11.26}$$

Here, $g(a)$ is the *logistic sigmoid activation function*, plotted in figure 11.3. Sigmoid means 'S'-shaped: the function maps $(-\infty, \infty)$ onto $(0, 1)$ — it is a "squashing function". If $|a|$ is small then $g(a)$ is approximately linear: so a network with logistic sigmoid activation functions approximates a linear network when the weights (and hence the inputs to the activation function) are small. As a increases, $g(a)$ saturates to 1, and as a decreases to become large and negative $g(a)$ saturates to 0.

For a single layer neural network:

$$a = \mathbf{w}^T\mathbf{x} + w_0\,. \tag{11.27}$$

If we have a single-layer neural network, with one output, and a logistic sigmoid activation function g on the output node (11.20), then from (11.26) and (11.27) we see that the posterior probability may be written:

$$P(C_1|\mathbf{x}) = g(a) = g(\mathbf{w}^T\mathbf{x} + w_0)\,. \tag{11.28}$$

This corresponds to the single layer neural network of equation (11.20) and figure 11.2.

Therefore, for a two class problem (which may be represented with a single output), a single layer neural network with a logistic sigmoid activation function on the output may be regarded as providing a posterior probability estimate. This is an interesting result, since it means we may obtain a posterior

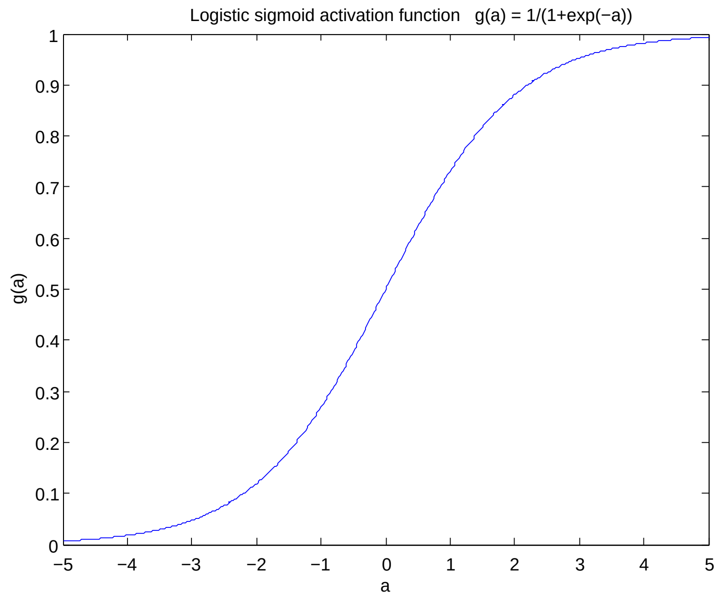


Figure 11.3: Logistic sigmoid function, $g(a) = \dfrac{1}{1 + e^{-a}}$

probability estimate by training the weights of a single-layer neural network by gradient descent, with no need to estimate (or assume) Gaussian pdfs. The optimal setting of the weights of such a network, given a training set, need not be the same as those obtained using a Gaussian classifier with a shared covariance matrix.

This single layer neural network, giving $P(C_1|\mathbf{x}) = 1/(1 + \exp(-\mathbf{w}^T\mathbf{x} - w_0))$, is known as the *logistic regression* binary classifier. Logistic regression is a 'work-horse' of practical machine learning: it's widely used, and important to understand.

11.5.2 Gradient descent training

Gradient descent training of a single-layer neural network with a logistic sigmoid activation function is similar to training a linear network. The principal difference is that we must take of the contribution of the activation function to the gradient.

Consider a network with sigmoid activation functions on the outputs. We can write the "forward" equations of the network as:

$$y_{nk} = g(a_{nk}) \tag{11.29}$$

$$a_{nk} = \sum_{i=0}^{d} w_{ki} x_{ni}\,. \tag{11.30}$$

In this case, the sum-of-squares error function is given by:

$$E = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}(y_{nk} - t_{nk})^2 \tag{11.31}$$

$$= \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}(g(a_{nk}) - t_{nk})^2 . \tag{11.32}$$

The partial derivative of the error function with respect to weight w_{ki} for pattern n is:

$$\frac{\partial E_n}{\partial w_{ki}} = \frac{\partial E_n}{\partial y_{nk}}\frac{\partial y_{nk}}{\partial a_{nk}}\frac{\partial a_{nk}}{\partial w_{ki}} \tag{11.33}$$

$$= \delta_{nk}\, g'(a_{nk})\, x_{ni} \tag{11.34}$$

where $\delta_{nk} = (y_{nk} - t_{nk})$, as before. $g'(a)$ is the derivative of the sigmoid activation function. It turns out that

$$g'(a) = g(a)(1 - g(a)) . \tag{11.35}$$

(You should check this!)

Therefore the total gradient is:

$$\frac{\partial E}{\partial w_{ki}} = \sum_{n=1}^{N}\frac{\partial E_n}{\partial w_{ki}} \tag{11.36}$$

$$= \sum_{n=1}^{N} g'(a_{nk})\,\delta_{nk}\, x_{ni} . \tag{11.37}$$

11.6 Softmax

If we have more than two classes then a suitable discriminant function for class k is related to the log posterior probability:

$$a_k = \ln p(\mathbf{x}|C_k)\, P(C_k) , \tag{11.38}$$

where a_k is called the activation value of output k.

If we substitute the activation into Bayes' theorem, we find that the posterior probability is given by the following expression:

$$P(C_k|\mathbf{x}) = \frac{\exp(a_k)}{\sum_{\ell=1}^{K}\exp(a_\ell)} , \tag{11.39}$$

which is sometimes referred to as the *softmax* or normalised exponential.

We can use the softmax as the output function for a multi-class single layer neural network:

$$y_k = \frac{\exp(a_k)}{\sum_{\ell=1}^{K}\exp(a_\ell)} \tag{11.40}$$

$$a_k = \sum_{i=0}^{d} w_{ki}x_i . \tag{11.41}$$

This form of output function guarantees that the K output values will sum to 1, a necessary condition for probability estimates.

We can perform gradient descent training when there is a softmax activation function on the outputs. We have to be a little more careful when estimating the derivatives, since the output y_k depends on the other outputs y_ℓ, through the normalisation term in the denominator. Therefore we need to consider the partial derivative of y_k with respect to all the activation values a_ℓ when computing the gradient:

$$\frac{\partial E_n}{\partial w_{ki}} = \frac{\partial E_n}{\partial a_{nk}}\frac{\partial a_{nk}}{\partial w_{ki}} \tag{11.42}$$

$$\frac{\partial E_n}{\partial a_{nk}} = \sum_{\ell=1}^{K}\frac{\partial E_n}{\partial y_{n\ell}}\frac{\partial y_{n\ell}}{\partial a_{nk}} \tag{11.43}$$

$$= \sum_{\ell=1}^{K}\delta_{n\ell}\frac{\partial y_{n\ell}}{\partial a_{nk}} \tag{11.44}$$

$\partial E_n/\partial y_{n\ell}$ equals δ_{nk}, as before. The other derivative turns out to be:

$$\frac{\partial y_{n\ell}}{\partial a_{nk}} = y_\ell I_{k\ell} - y_\ell y_k \tag{11.45}$$

where $I_{k\ell} = 1$ if $k = \ell$, and $I_{k\ell} = 0$ otherwise.

11.7 "Online" gradient descent

The error function is usually calculated by summing over all input patterns, where E_n is the contribution to the error from a single pattern:

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) .$$

It is possible to use the gradient of E_n to update the weights one pattern at a time:

$$w_{ki}^{(\tau+1)} = w_{ki}^{(\tau)} - \eta\frac{\partial E_n}{\partial w_{ki}} . \tag{11.46}$$

This is referred to as *online gradient descent*. It is usually preferable to choose patterns randomly when training online. In *stochastic gradient descent* a training pattern is randomly chosen, the error and its derivatives computed for that pattern, and the weights are updated using (11.46). Online gradient descent can be useful for real-time, adaptive applications. Stochastic gradient descent is an efficient procedure when using large data sets.

11.8 Matlab: Training single layer neural networks

It is straightforward to use single-layer neural networks in Matlab, using the Netlab toolbox. To define a single layer neural network, with 2 inputs, 2 outputs and a linear output function, use the function `glm`:

```
slnet = glm(2,2,'linear');
```

To train this network with gradient descent, we need to set a few options (look at the help page on `graddesc` to learn more):

```
options=foptions;
options(1)=1; %% display/log error values
options(3)=0.01; %% convergence criterion
options(14)=100; %% maximum number of training iterations
options(18)=0.001; %% learning rate (eta)
```

Training a network in Netlab is done using the netopt function which takes the specific training algorithm to be used as an argument. To train this net using gradient descent, where `xtrain` is the matrix of training data, and `ttrain` contains the corresponding targets, one row per pattern.

```
[slnet options errlog] =
      netopt(slnet, options, xtrain, ttrain, 'graddesc');
plot(errlog);
```

For comparison try using the IRLS trainer (`glmtrain`).

The second statement plots the graph of the error versus learning rate. Gradient descent is sensitive to the value of the learning rate: if it is too high then the step down the gradient is too big, and the error function can increase instead of decreasing, and is likely to become unstable, increasing rapidly in later iterations. On the other hand, if the step size is too small, the error rate will decrease more slowly than possible.

11.8.1 Testing

Once the network is trained, the function `glmfwd(net,x)` can be used to run the test data `xtest` through the network:

```
testOut=glmfwd(slnet,xtest);
```

If `ttest` contains the targets for the test data (i.e., the correct classification), the following piece of Matlab can be used to compute the classification error:

```
    [maxOut, classified] = max(testOut,[],2);
    [tmpOut, answerClasses] = max(ttest,[],2);
    numberMisclassified = size(find(classified-answerClasses));
    percentError = 100.0 * numberMisclassified / size(answerClasses);
```

MATLAB tricks:

- `max(testOut,[],2);` means work along dimension 2 — i.e., returns the maximum value for each row.
- `[maxOut, classified] = max(testOut,[],2);` returns the max values for each row in `maxOut`, and the index in `classified`.
- `find(classified-answerClasses)` returns the indices of non-zero elements (i.e., elements where the output class is different to the target class).
- `size(find(classified-answerClasses))` thus returns the number of misclassified patterns.

To train a single-layer neural network with a logistic sigmoid activation function in Matlab. If `xtrain` and `ttrain` contain the input vectors and targets, then:

```
options=foptions;
options(1)=1;
options(18)=0.01;
ttrain = [ta;tb];

slnet=glm(2,1,'logistic')
[slnet options errlog] =
      netopt(slnet, options, xtrain, ttrain, 'graddesc');
```

For the softmax activation function, the third argument to glm should be `'softmax'`.

Technical note: (For a good reason) Netlab uses a different error function for logistic sigmoid output compared with linear output. This means that the error values produced by `netopt` for the logistic and linear activation functions are not comparable.

11.8.2 Tasks: Single Layer neural networks

The data for this lab consists of two files `ecoli-train.netlab` and `ecoli-test.netlab`; these are available from:
`http://www.inf.ed.ac.uk/teaching/courses/inf2b/labs/ecoli-train.netlab`
`http://www.inf.ed.ac.uk/teaching/courses/inf2b/labs/ecoli-test.netlab`

The lab is concerned with classifying a real data set. The data consists of 5 real-valued inputs and a discrete class. There are four classes (numbered 1–4). There are 230 training patterns in total (107 for class 1, 58 for class 2, 39 for class 3 and 26 for class 4). There are also 77 test patterns. The data is to do with molecular biology, but for this example you don't need to be concerned with that (for information the original README file that corresponds to the data set is available on the lab3 web page). The data has been reformatted into a form suitable for Netlab. Download the files `ecoli-train.netlab` and `ecoli-test.netlab`, and use the method `datread` to load them, e.g.:

```
[ecxtrain, ecttrain, nin, nout, ndata] = datread('ecoli-train.netlab');
[ecxtest, ecttest, nin, nout, ndata] = datread('ecoli-test.netlab');
```

Carry out the following tasks.

1. Train a single-layer neural network on this data using gradient descent. Observe how the error decreases per iteration. Experiment with various learning rates. In each case compute the error rate on the test set.
2. Compare IRLS training with gradient descent training.
3. Repeat the experiments using a single layer neural network with a softmax output activation function.

11.9 Summary

This chapter has introduced several important concepts:

- The representation of a set of discriminant functions as a single-layer neural network.
- Direct training of the parameters of a single-layer neural network.
- Minimisation of error function by gradient descent in parameter space.
- The logistic sigmoid and softmax activation functions, and their relation to posterior probabilities.
- Online or stochastic gradient descent.

Multi-Layer Neural Networks

Hiroshi Shimodaira*

January-March 2017

In the previous chapter, we saw how single-layer linear networks could be generalised by applying an output activation function such as a sigmoid. We can further generalise such networks by applying a set of fixed nonlinear transforms ϕ_j to the input vector $\mathbf{x}$. For a single output network:

$$y(\mathbf{x}, \mathbf{w}) = g\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right) . \tag{12.1}$$

Again g is a nonlinear activation function such as a sigmoid. If the functions $\phi_j(\mathbf{x})$ are fixed and non-adaptive they are sometimes referred to as basis functions. Using such basis functions broadens the class of available discriminant functions. Our goal in this chapter is to investigate how to make these basis functions adaptive: that is to have their own parameters (e.g., another weight matrix) that may be estimated automatically from a training data set.

Given enough fixed basis functions, and infinite training data, it is possible to approximate any continuous function. However, for finite datasets, 'Deep' networks, which use parameterised basis functions, often work better than 'shallow' networks, which have a large number of fixed basis functions.

12.1 Multi-layer Perceptrons

In this section we build up a multi-layer neural network model, step by step. This multi-layer network has different names: multi-layer perceptron (MLP), feed-forward neural network, artificial neural network (ANN), backprop network.[1]

The first layer involves M linear combinations of the d-dimensional inputs:

$$b_j = \sum_{i=0}^{d} w_{ji}^{(1)} x_i , \qquad j = 1, 2, \ldots, M . \tag{12.2}$$

As before $x_0 = 1$, with the weights leading out from it corresponding to the biases. The quantities b_j are called *activations*, and the parameters $w_{ji}^{(1)}$ are the weights. The superscript '(1)' indicates that this is the first layer of the network. Each of the activations is then transformed by a nonlinear activation function g, typically a sigmoid:

$$z_j = h(b_j) = \frac{1}{1 + \exp(-b_j)} \tag{12.3}$$

*©2014-2017 University of Edinburgh. All rights reserved. This note is heavily based on notes inherited from Steve Renals and Iain Murray.

[1]MLP is probably the most frequently used name, although it is not strictly accurate. The perceptron is a single layer network with discontinuous step function nonlinear activation functions, rather than the continuous nonlinear activation functions used here.

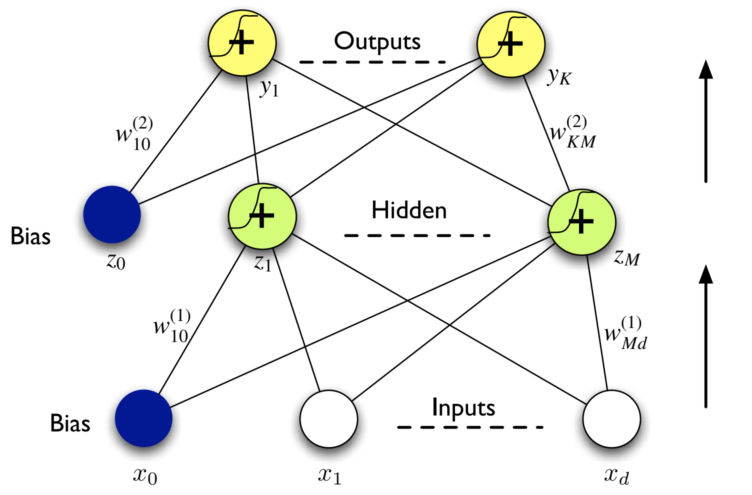


Figure 12.1: Network diagram for a multi-layer perceptron (MLP) with two layers of weights.

The outputs z_j correspond to the outputs of the basis functions in (12.1). In the context of neural networks, the quantities z_j are interpreted as the output of *hidden units*—so called because they do not have values specified by the problem (as is the case for input units) or target values used in training (as is the case for output units).

In the second layer, the outputs of the hidden units are linearly combined to give the activations of the K output units:

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j \qquad k = 1, 2, \ldots, K . \tag{12.4}$$

Again $z_0 = 1$, corresponding to the bias. This transformation is the second layer of the neural network parameterised by weights $w_{kj}^{(2)}$. The output units are transformed using an activation function; again a sigmoid may be used:

$$y_k = g(a_k) = \frac{1}{1 + \exp(-a_k)} , \tag{12.5}$$

or for multiclass problems, a softmax activation function:

$$g(a_k) = \frac{\exp(a_k)}{\sum_{\ell=1}^{K} \exp(a_\ell)} .$$

These equations may be combined to give the overall equation that describes the *forward propagation* through the network, and describes how an output vector is computed from an input vector, given the weight matrices:

$$y_k = g\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{d} w_{ji}^{(1)} x_i\right)\right) \tag{12.6}$$

This is illustrated in figure 12.1.

12.2 MLP Training: Back-propagation of error

Similar to single-layer neural networks, we can train a network using gradient descent. This involves defining an error function E, and then evaluating the derivatives $\partial E/\partial w^{(2)}_{kj}$ and $\partial E/\partial w^{(1)}_{ji}$. The evaluation of these error derivatives proceeds using a version of the chain rule of differentiation, referred to as *back-propagation of error*, or just *backprop*.

When training single-layer neural networks, the error gradients are the product of the derivative of the error at the output of the weights and the value at the input to the weight. This interpretation is still possible for the hidden-to-output weights $w^{(2)}_{kj}$, for which a target output is available and the input is obtained from the hidden units. However target values are not available for hidden units, and so it is not possible to train the input-to-hidden weights in precisely the same way. This is sometimes called the *credit assignment* problem: what is the "error" of a hidden unit? how does the value of a particular input-to-hidden weight affect the overall error? The solution to this problem is found by systematically deriving expressions for the relevant derivatives using the chain rule of differentiation.

12.2.1 Error function

To train an MLP we need to define an error function. Again we use the sum-of-squares error function, obtained by summing over a training set of N examples:

$$E = \sum_{n=1}^{N} E_n \tag{12.7}$$

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_{nk} - t_{nk})^2 . \tag{12.8}$$

The values of y_{nk} may be computed for each pattern using the MLP forward propagation equation (12.6). To avoid clutter, we'll drop the '(1)' and '(2)' superscripts when writing down weights.

To obtain the overall error gradients, we sum over the training examples:

$$\frac{\partial E}{\partial w_{kj}} = \sum_{n=1}^{N} \frac{\partial E_n}{\partial w_{kj}} \tag{12.9}$$

$$\frac{\partial E}{\partial w_{ji}} = \sum_{n=1}^{N} \frac{\partial E_n}{\partial w_{ji}} . \tag{12.10}$$

12.2.2 Hidden-to-output weights

First we would like to compute the error gradients for the hidden-to-output weights, $\partial E_n/\partial w_{kj}$. Now we can write E_n in terms of these weights:

$$\begin{aligned} E_n &= \frac{1}{2}\sum_{k=1}^{K}(g(a_{nk}) - t_{nk})^2 \\ &= \frac{1}{2}\sum_{k=1}^{K}\left(g\left(\sum_{j=0}^{M} w_{kj} z_{nj}\right) - t_{nk}\right)^2 . \end{aligned} \tag{12.11}$$

The derivatives of the error with respect to w_{kj} can be broken down as follows:

$$\frac{\partial E_n}{\partial w_{kj}} = \frac{\partial E_n}{\partial a_{nk}} \frac{\partial a_{nk}}{\partial w_{kj}} . \tag{12.12}$$

The gradient of the error E_n with respect to the activations a_{nk} is often referred to as the error signal and given the notation δ_{nk}, analogous to what we had for single layer neural networks.

$$\delta_{nk} = \frac{\partial E_n}{\partial a_{nk}} \tag{12.13}$$

And since:

$$\frac{\partial a_{nk}}{\partial w_{kj}} = z_{nj} \tag{12.14}$$

we may substitute (12.13) and (12.14) into (12.12) to obtain:

$$\frac{\partial E_n}{\partial w_{kj}} = \delta_{nk}\, z_{nj} \tag{12.15}$$

where:

$$\delta_{nk} = \frac{\partial E_n}{\partial y_{nk}} \cdot \frac{\partial y_{nk}}{\partial a_{nk}} = (y_{nk} - t_{nk})\, g'(a_{nk}) , \tag{12.16}$$

similar to single-layer neural networks with a nonlinear activation function.

12.2.3 Input-to-hidden weights

Now we would like to compute the error gradients for the input-to-hidden weights, $\partial E / \partial w_{ji}$. To do this we need to make sure that we take into account all the ways in which hidden unit j (and hence weight w_{ji}) can influence the error. To do this let's look at δ_{nj}, the error signal for hidden unit j:

$$\begin{aligned}\delta_{nj} &= \frac{\partial E_n}{\partial b_{nj}} \\ &= \sum_{k=1}^{K} \frac{\partial E_n}{\partial a_{nk}} \frac{\partial a_{nk}}{\partial b_{nj}} \\ &= \sum_{k=1}^{K} \delta_{nk} \frac{\partial a_{nk}}{\partial b_{nj}} \,. \end{aligned} \tag{12.17}$$

Since hidden unit j can influence the error through all the output units (since it is connected to all of them), we must sum over all the output units' contributions to δ_{nj}. We need the expression for $\partial a_{nk} / \partial b_{nj}$, obtained by differentiating (12.4) and the hidden unit activation function (12.3):

$$\begin{aligned}\frac{\partial a_{nk}}{\partial b_{nj}} &= \frac{\partial a_{nk}}{\partial z_{nj}} \frac{\partial z_{nj}}{\partial b_{nj}} \\ &= w_{kj}\, h'(b_{nj}) \,. \end{aligned} \tag{12.18}$$

Substituting (12.18) into (12.17) we obtain:

$$\delta_{nj} = h'(b_{nj}) \sum_{k=1}^{K} \delta_{nk}\, w_{kj} \,. \tag{12.19}$$

This is the famous *back-propagation of error* (backprop) equation. By applying the chain rule of differentiation, backprop obtains the δ values for hidden units by "back-propagating" the δ values of the outputs, weighted by the hidden-to-output weight matrix. This is illustrated in figure 12.2. The derivatives of the input-to-hidden weights can thus be evaluated using:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial b_{nj}} \frac{\partial b_{nj}}{\partial w_{ji}} = \delta_j\, x_i \,. \tag{12.20}$$

This approach can be recursively applied to further hidden layers.

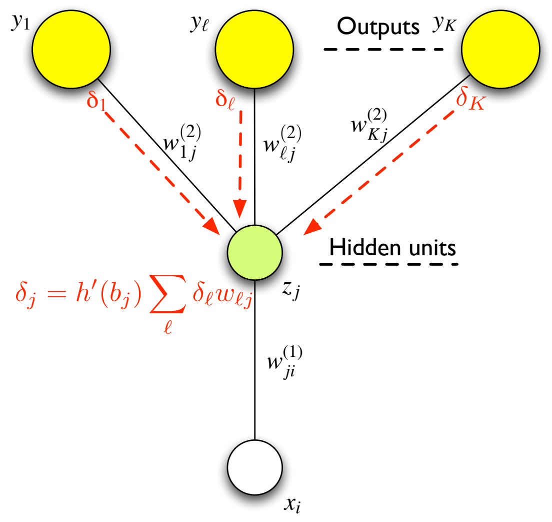


Figure 12.2: Back-propagation of error signals in an MLP

12.2.4 Back-propagation algorithm

The back-propagation of error algorithm is summarised as follows:

1. Apply the N input vectors from the training set, $\mathbf{x}_n$, to the network and forward propagate using (12.6) to obtain the set of output vectors $\mathbf{y}_n$
2. Using the target vectors $\mathbf{t}_n$ compute the error E using (12.7) and (12.8).
3. Evaluate the error signals δ_{nk} for each output unit using (12.16).
4. Evaluate the error signals δ_{nk} for each hidden unit using back-propagation of error (12.19).
5. Use (12.15) and (12.20) to evaluate the derivatives for each training pattern, obtaining the overall derivatives using (12.9) and (12.10).