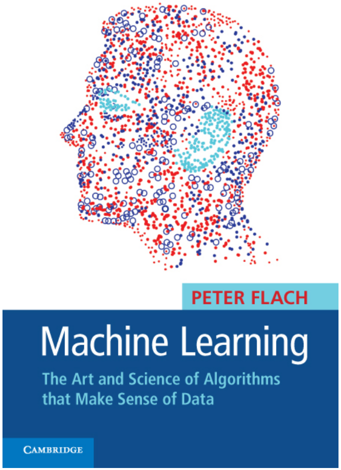
Machine Learning

The Art and Science of Algorithms that Make Sense of Data

Peter A. Flach

Intelligent Systems Laboratory, University of Bristol, United Kingdom

Edited by Tomasz Pawlak to match requirements of course of
Applications of Computational Intelligence Methods
at Poznan University of Technology, Faculty of Computing.



These slides accompany the above book published by Cambridge University Press in 2012, and are made freely available for teaching purposes (the copyright remains with the author, however).

The material is divided in four difficulty levels A (basic) to D (advanced); this PDF includes all material up to level B.

Table of contents I

Table of contents II

Table of contents III

Table of contents IV

Assassinating spam e-mail

SpamAssassin is a widely used open-source spam filter. It calculates a score for an incoming e-mail, based on a number of built-in rules or ‘tests’ in SpamAssassin’s terminology, and adds a ‘junk’ flag and a summary report to the e-mail’s headers if the score is 5 or more.

```
-0.1 RCVD_IN_MXRATE_WL       RBL: MXRate recommends allowing
                             [123.45.6.789 listed in sub.mxrate.net]
 0.6 HTML_IMAGE_RATIO_02     BODY: HTML has a low ratio of text to image area
 1.2 TVD_FW_GRAPHIC_NAME_MID BODY: TVD_FW_GRAPHIC_NAME_MID
 0.0 HTML_MESSAGE            BODY: HTML included in message
 0.6 HTML_FONx_FACE_BAD      BODY: HTML font face is not a word
 1.4 SARE_GIF_ATTACH         FULL: Email has a inline gif
 0.1 BOUNCE_MESSAGE          MTA bounce message
 0.1 ANY_BOUNCE_MESSAGE      Message is some kind of bounce message
 1.4 AWL                     AWL: From: address is in the auto white-list
```

From left to right you see the score attached to a particular test, the test identifier, and a short description including a reference to the relevant part of the e-mail. As you see, scores for individual tests can be negative (indicating evidence suggesting the e-mail is ham rather than spam) as well as positive. The overall score of 5.3 suggests the e-mail might be spam.

Linear classification

Suppose we have only two tests and four training e-mails, one of which is spam (see Table 1).

- ☞ Both tests succeed for the spam e-mail;
- ☞ for one ham e-mail neither test succeeds,
- ☞ for another the first test succeeds and the second doesn’t,
- ☞ and for the third ham e-mail the first test fails and the second succeeds.

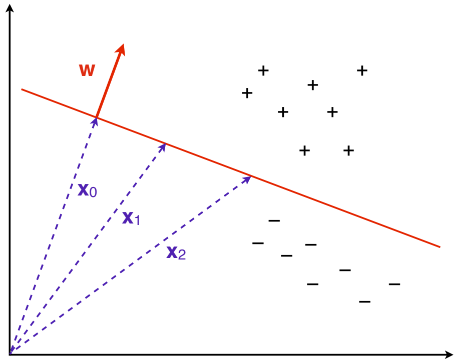
Spam filtering as a classification task

E-mail	x_1	x_2	Spam?	$4x_1+4x_2$
1	1	1	1	8
2	0	0	0	0
3	1	0	0	4
4	0	1	0	4

It is easy to see that assigning both tests a weight of 4 correctly 'classifies' these four e-mails into spam and ham. In the mathematical notation introduced in Background 1 we could describe this classifier as $4x_1+4x_2>5$ or $(4,4)\cdot(x_1,x_2)>5$.

In fact, any weight between 2.5 and 5 will ensure that the threshold of 5 is only exceeded when both tests succeed. We could even consider assigning different weights to the tests – as long as each weight is less than 5 and their sum exceeds 5 – although it is hard to see how this could be justified by the training

Linear classification in two dimensions



The straight line separates the positives from the negatives. It is defined by $\mathbf{w} \cdot \mathbf{x}_i = t$, where $\mathbf{w}$ is a vector perpendicular to the decision boundary and pointing in the direction of the positives, t is the decision threshold, and $\mathbf{x}_i$ points to a point on the decision boundary. In particular, $\mathbf{x}_0$ points in the same direction as $\mathbf{w}$, from which it follows that $\mathbf{w} \cdot \mathbf{x}_0 = ||\mathbf{w}||\,||\mathbf{x}_0|| = t$ ($||\mathbf{x}||$ denotes the length of the vector $\mathbf{x}$).

Homogeneous coordinates

It is sometimes convenient to simplify notation further by introducing an extra constant 'variable' $x_0 = 1$, the weight of which is fixed to $w_0 = -t$.

The extended data point is then $\mathbf{x}^{\circ} = (1, x_1, \ldots, x_n)$ and the extended weight vector is $\mathbf{w}^{\circ} = (-t, w_1, \ldots, w_n)$, leading to the decision rule $\mathbf{w}^{\circ} \cdot \mathbf{x}^{\circ} > 0$ and the decision boundary $\mathbf{w}^{\circ} \cdot \mathbf{x}^{\circ} = 0$.

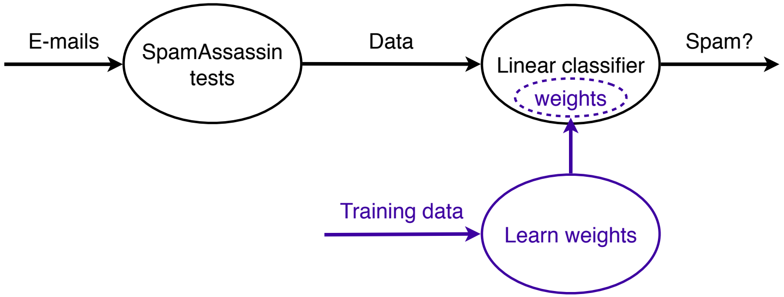
Thanks to these so-called homogeneous coordinates the decision boundary passes through the origin of the extended coordinate system, at the expense of needing an additional dimension.

☞ note that this doesn't really affect the data, as all data points and the 'real' decision boundary live in the plane $x_0 = 1$.

Important point to remember

Machine learning is the systematic study of algorithms and systems that improve their knowledge or performance with experience.

Machine learning for spam filtering



At the top we see how SpamAssassin approaches the spam e-mail classification task: the text of each e-mail is converted into a data point by means of SpamAssassin's built-in tests, and a linear classifier is applied to obtain a 'spam or ham' decision. At the bottom (in blue) we see the bit that is done by machine learning.

Overfitting

Imagine you are preparing for your *Machine Learning 101* exam. Helpfully, Professor Flach has made previous exam papers and their worked answers available online. You begin by trying to answer the questions from previous papers and comparing your answers with the model answers provided.

Unfortunately, you get carried away and spend all your time on memorising the model answers to all past questions. Now, if the upcoming exam completely consists of past questions, you are certain to do very well. But if the new exam asks different questions about the same material, you would be ill-prepared and get a much lower mark than with a more traditional preparation.

In this case, one could say that you were *overfitting* the past exam papers and that the knowledge gained didn't *generalise* to future exam questions.

A Bayesian classifier I

Bayesian spam filters maintain a vocabulary of words and phrases – potential spam or ham indicators – for which statistics are collected from a training set.

- ☞ For instance, suppose that the word ‘Viagra’ occurred in four spam e-mails and in one ham e-mail. If we then encounter a new e-mail that contains the word ‘Viagra’, we might reason that the odds that this e-mail is spam are 4:1, or the probability of it being spam is 0.80 and the probability of it being ham is 0.20.
- ☞ The situation is slightly more subtle because we have to take into account the prevalence of spam. Suppose that I receive on average one spam e-mail for every six ham e-mails. This means that I would estimate the odds of an unseen e-mail being spam as 1:6, i.e., non-negligible but not very high either.

A Bayesian classifier II

☞ If I then learn that the e-mail contains the word ‘Viagra’, which occurs four times as often in spam as in ham, I need to combine these two odds. As we shall see later, Bayes’ rule tells us that we should simply multiply them: 1:6 times 4:1 is 4:6, corresponding to a spam probability of 0.4.

In this way you are combining two independent pieces of evidence, one concerning the prevalence of spam, and the other concerning the occurrence of the word ‘Viagra’, pulling in opposite directions.

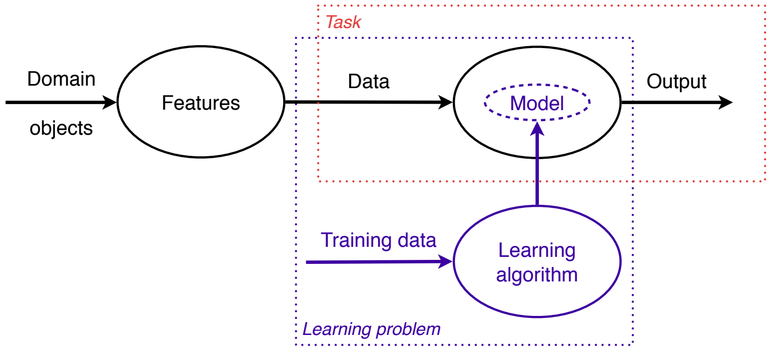
The nice thing about this ‘Bayesian’ classification scheme is that it can be repeated if you have further evidence. For instance, suppose that the odds in favour of spam associated with the phrase ‘blue pill’ is estimated at 3:1, and suppose our e-mail contains both ‘Viagra’ and ‘blue pill’, then the combined odds are 4:1 times 3:1 is 12:1, which is ample to outweigh the 1:6 odds associated with the low prevalence of spam (total odds are 2:1, or a spam probability of 0.67, up from 0.40 without the ‘blue pill’).

A rule-based classifier

- if the e-mail contains the word ‘Viagra’ then estimate the odds of spam as 4:1;
- otherwise, if it contains the phrase ‘blue pill’ then estimate the odds of spam as 3:1;
- otherwise, estimate the odds of spam as 1:6.

The first rule covers all e-mails containing the word ‘Viagra’, regardless of whether they contain the phrase ‘blue pill’, so no overcounting occurs. The second rule *only* covers e-mails containing the phrase ‘blue pill’ but not the word ‘Viagra’, by virtue of the ‘otherwise’ clause. The third rule covers all remaining e-mails: those which neither contain neither ‘Viagra’ nor ‘blue pill’.

How machine learning helps to solve a task



An overview of how machine learning is used to address a given task. A task (red box) requires an appropriate mapping – a model – from data described by features to outputs. Obtaining such a mapping from training data is what constitutes a learning problem (blue box).

Important point to remember

Tasks are addressed by models, whereas learning problems are solved by learning algorithms that produce models.

Important point to remember

Machine learning is concerned with using the right features to build the right models that achieve the right tasks.

What's next?

1. The ingredients of machine learning
   - Tasks: the problems that can be solved with machine learning
     - Looking for structure
   - Models: the output of machine learning
     - Geometric models
     - Probabilistic models
     - Logical models
     - Grouping and grading
   - Features: the workhorses of machine learning
     - Many uses of features
     - Feature construction and transformation

Important point to remember

Models lend the machine learning field diversity, but tasks and features give it unity.

What's next?

Tasks for machine learning

The most common machine learning tasks are *predictive*, in the sense that they concern predicting a target variable from features.

- Binary and multi-class classification: categorical target
- Regression: numerical target
- Clustering: hidden target

Descriptive tasks are concerned with exploiting underlying structure in the data.

Example 1.1, p.15

Measuring similarity

If our e-mails are described by word-occurrence features as in the text classification example, the similarity of e-mails would be measured in terms of the words they have in common. For instance, we could take the number of common words in two e-mails and divide it by the number of words occurring in either e-mail (this measure is called the *Jaccard coefficient*).

Suppose that one e-mail contains 42 (different) words and another contains 112 words, and the two e-mails have 23 words in common, then their similarity would be $\frac{23}{42+112-23} = \frac{23}{130} = 0.18$. We can then cluster our e-mails into groups, such that the average similarity of an e-mail to the other e-mails in its group is much larger than the average similarity to e-mails from other groups.

Looking for structure

Looking for structure I

Consider the following matrix:

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 2 & 2 & 3 \end{pmatrix}$$

Imagine these represent ratings by six different people (in rows), on a scale of 0 to 3, of four different films – say *The Shawshank Redemption*, *The Usual Suspects*, *The Godfather*, *The Big Lebowski*, (in columns, from left to right). *The Godfather* seems to be the most popular of the four with an average rating of 1.5, and *The Shawshank Redemption* is the least appreciated with an average rating of 0.5. Can you see any structure in this matrix?

Looking for structure II

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 2 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- ☞ The right-most matrix associates films (in columns) with genres (in rows): *The Shawshank Redemption* and *The Usual Suspects* belong to two different genres, say drama and crime, *The Godfather* belongs to both, and *The Big Lebowski* is a crime film and also introduces a new genre (say comedy).
- ☞ The tall, 6-by-3 matrix then expresses people's preferences in terms of genres.

Looking for structure III

- Finally, the middle matrix states that the crime genre is twice as important as the other two genres in terms of determining people's preferences.

Table 1.1, p.18

Machine learning settings

	Predictive model	*Descriptive model*
Supervised learning	classification, regression	subgroup discovery
Unsupervised learning	predictive clustering	descriptive clustering, association rule discovery

The rows refer to whether the training data is labelled with a target variable, while the columns indicate whether the models learned are used to predict a target variable or rather describe the given data.

What's next?

Machine learning models

Machine learning models can be distinguished according to their main intuition:

- ☞ *Geometric* models use intuitions from geometry such as separating (hyper-)planes, linear transformations and distance metrics.
- ☞ *Probabilistic* models view learning as a process of reducing uncertainty, modelled by means of probability distributions.
- ☞ *Logical* models are defined in terms of easily interpretable logical expressions.

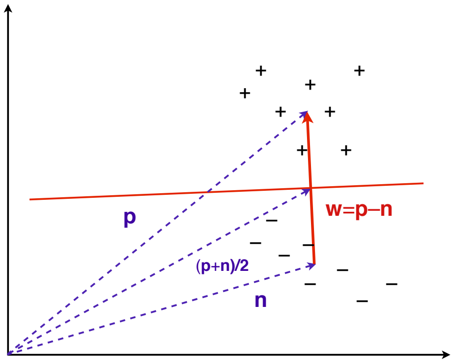
Alternatively, they can be characterised by their *modus operandi*:

- ☞ *Grouping models* divide the instance space into segments; in each segment a very simple (e.g., constant) model is learned.
- ☞ *Grading models* learning a single, global model over the instance space.

Geometric models

Figure 1.1, p.22

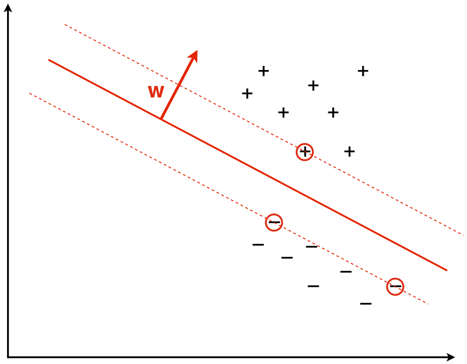
Basic linear classifier



The basic linear classifier constructs a decision boundary by half-way intersecting the line between the positive and negative centres of mass. It is described by the equation $\mathbf{w} \cdot \mathbf{x} = t$, with $\mathbf{w} = \mathbf{p} - \mathbf{n}$; the decision threshold can be found by noting that $(\mathbf{p} + \mathbf{n})/2$ is on the decision boundary, and hence $t = (\mathbf{p} - \mathbf{n}) \cdot (\mathbf{p} + \mathbf{n})/2 = (||\mathbf{p}||^2 - ||\mathbf{n}||^2)/2$, where $||\mathbf{x}||$ denotes the length of vector $\mathbf{x}$.

Figure 1.2, p.23

Support vector machine



The decision boundary learned by a support vector machine from the linearly separable data from Figure 1. The decision boundary maximises the margin, which is indicated by the dotted lines. The circled data points are the support vectors.

Probabilistic models

Table 1.2, p.26

A simple probabilistic model

Viagra	lottery	$P(Y = \text{spam}\|\text{Viagra}, \text{lottery})$	$P(Y = \text{ham}\|\text{Viagra}, \text{lottery})$
0	0	0.31	**0.69**
0	1	**0.65**	0.35
1	0	**0.80**	0.20
1	1	0.40	**0.60**

‘Viagra’ and ‘lottery’ are two Boolean features; Y is the class variable, with values ‘spam’ and ‘ham’. In each row the most likely class is indicated in bold.

Decision rule

Assuming that X and Y are the only variables we know and care about, the posterior distribution $P(Y|X)$ helps us to answer many questions of interest.

- ☞ For instance, to classify a new e-mail we determine whether the words 'Viagra' and 'lottery' occur in it, look up the corresponding probability $P(Y = \mathsf{spam}|\mathsf{Viagra}, \mathsf{lottery})$, and predict spam if this probability exceeds 0.5 and ham otherwise.
- ☞ Such a recipe to predict a value of Y on the basis of the values of X and the posterior distribution $P(Y|X)$ is called a *decision rule*.

Example 1.2, p.26

Missing values I

Suppose we skimmed an e-mail and noticed that it contains the word ‘lottery’ but we haven’t looked closely enough to determine whether it uses the word ‘Viagra’. This means that we don’t know whether to use the second or the fourth row in Table 1.2 to make a prediction. This is a problem, as we would predict spam if the e-mail contained the word ‘Viagra’ (second row) and ham if it didn’t (fourth row). The solution is to average these two rows, using the probability of ‘Viagra’ occurring in any e-mail (spam or not):

$$\begin{aligned} P(Y|\text{lottery}) =& P(Y|\text{Viagra}=0,\text{lottery})P(\text{Viagra}=0) \\ &+ P(Y|\text{Viagra}=1,\text{lottery})P(\text{Viagra}=1) \end{aligned}$$

Example 1.2, p.26

Missing values II

For instance, suppose for the sake of argument that one in ten e-mails contain the word 'Viagra', then $P(\text{Viagra} = 1) = 0.10$ and $P(\text{Viagra} = 0) = 0.90$. Using the above formula, we obtain
$P(Y = \text{spam}|\text{lottery} = 1) = 0.65 \cdot 0.90 + 0.40 \cdot 0.10 = 0.625$ and
$P(Y = \text{ham}|\text{lottery} = 1) = 0.35 \cdot 0.90 + 0.60 \cdot 0.10 = 0.375$. Because the occurrence of 'Viagra' in any e-mail is relatively rare, the resulting distribution deviates only a little from the second row in Table 1.2.

Likelihood ratio

As a matter of fact, statisticians work very often with different conditional probabilities, given by the *likelihood function* $P(X|Y)$.

- ☞ I like to think of these as thought experiments: if somebody were to send me a spam e-mail, how likely would it be that it contains exactly the words of the e-mail I'm looking at? And how likely if it were a ham e-mail instead?
- ☞ What really matters is not the magnitude of these likelihoods, but their ratio: how much more likely is it to observe this combination of words in a spam e-mail than it is in a non-spam e-mail.
- ☞ For instance, suppose that for a particular e-mail described by X we have $P(X|Y = \text{spam}) = 3.5 \cdot 10^{-5}$ and $P(X|Y = \text{ham}) = 7.4 \cdot 10^{-6}$, then observing X in a spam e-mail is nearly five times more likely than it is in a ham e-mail.
- ☞ This suggests the following decision rule (maximum a posteriori, MAP): predict spam if the likelihood ratio is larger than 1 and ham otherwise.

Important point to remember

Use likelihoods if you want to ignore the prior distribution or assume it uniform, and posterior probabilities otherwise.

Example 1.3, p.28

Posterior odds

$$\frac{P(Y=\text{spam}|\text{Viagra}=0,\text{lottery}=0)}{P(Y=\text{ham}|\text{Viagra}=0,\text{lottery}=0)} = \frac{0.31}{0.69} = 0.45$$

$$\frac{P(Y=\text{spam}|\text{Viagra}=1,\text{lottery}=1)}{P(Y=\text{ham}|\text{Viagra}=1,\text{lottery}=1)} = \frac{0.40}{0.60} = 0.67$$

$$\frac{P(Y=\text{spam}|\text{Viagra}=0,\text{lottery}=1)}{P(Y=\text{ham}|\text{Viagra}=0,\text{lottery}=1)} = \frac{0.65}{0.35} = 1.9$$

$$\frac{P(Y=\text{spam}|\text{Viagra}=1,\text{lottery}=0)}{P(Y=\text{ham}|\text{Viagra}=1,\text{lottery}=0)} = \frac{0.80}{0.20} = 4.0$$

Using a MAP decision rule we predict ham in the top two cases and spam in the bottom two. Given that the full posterior distribution is all there is to know about the domain in a statistical sense, these predictions are the best we can do: they are *Bayes-optimal*.

Table 1.3, p.29

Example marginal likelihoods

Y	$P(\text{Viagra} = 1 \mid Y)$	$P(\text{Viagra} = 0 \mid Y)$
spam	0.40	0.60
ham	0.12	0.88

Y	$P(\text{lottery} = 1 \mid Y)$	$P(\text{lottery} = 0 \mid Y)$
spam	0.21	0.79
ham	0.13	0.87

Example 1.4, p.30

Using marginal likelihoods

Using the marginal likelihoods from Table 1.3, we can approximate the likelihood ratios (the previously calculated odds from the full posterior distribution are shown in brackets):

$$\frac{P(\text{Viagra}=0|Y=\text{spam})}{P(\text{Viagra}=0|Y=\text{ham})}\frac{P(\text{lottery}=0|Y=\text{spam})}{P(\text{lottery}=0|Y=\text{ham})}=\frac{0.60}{0.88}\frac{0.79}{0.87}=0.62 \qquad (0.45)$$

$$\frac{P(\text{Viagra}=0|Y=\text{spam})}{P(\text{Viagra}=0|Y=\text{ham})}\frac{P(\text{lottery}=1|Y=\text{spam})}{P(\text{lottery}=1|Y=\text{ham})}=\frac{0.60}{0.88}\frac{0.21}{0.13}=1.1 \qquad (1.9)$$

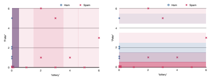
$$\frac{P(\text{Viagra}=1|Y=\text{spam})}{P(\text{Viagra}=1|Y=\text{ham})}\frac{P(\text{lottery}=0|Y=\text{spam})}{P(\text{lottery}=0|Y=\text{ham})}=\frac{0.40}{0.12}\frac{0.79}{0.87}=3.0 \qquad (4.0)$$

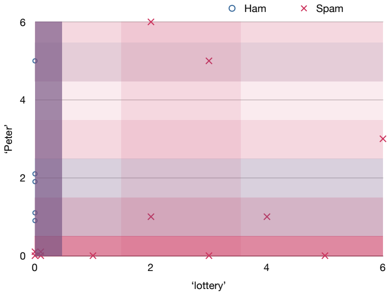
$$\frac{P(\text{Viagra}=1|Y=\text{spam})}{P(\text{Viagra}=1|Y=\text{ham})}\frac{P(\text{lottery}=1|Y=\text{spam})}{P(\text{lottery}=1|Y=\text{ham})}=\frac{0.40}{0.12}\frac{0.21}{0.13}=5.4 \qquad (0.67)$$

We see that, using a maximum likelihood decision rule, our very simple model arrives at the Bayes-optimal prediction in the first three cases, but not in the fourth ('Viagra' and 'lottery' both present), where the marginal likelihoods are actually very misleading.

Figure 1.3, p.31

The Scottish classifier



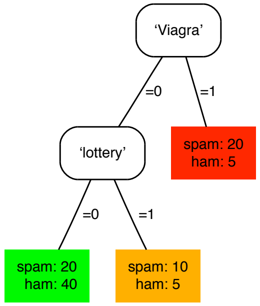


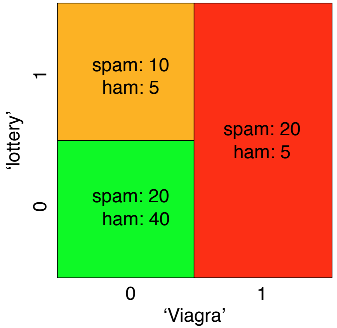
(top) Visualisation of two marginal likelihoods as estimated from a small data set. The colours indicate whether the likelihood points to spam or ham. **(bottom)** Combining the two marginal likelihoods gives a pattern not unlike that of a Scottish tartan.

Logical models

Figure 1.4, p.32

A feature tree





(left) A feature tree combining two Boolean features. Each internal node or split is labelled with a feature, and each edge emanating from a split is labelled with a feature value. Each leaf therefore corresponds to a unique combination of feature values. Also indicated in each leaf is the class distribution derived from the training set. **(right)** A feature tree partitions the instance space into rectangular regions, one for each leaf. We can clearly see that the majority of ham lives in the lower left-hand corner.

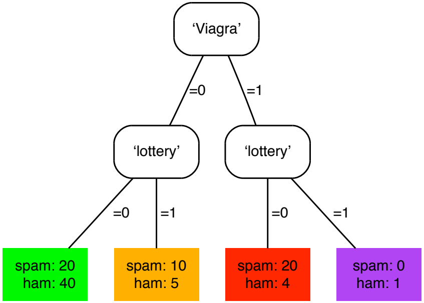
Example 1.5, p.33

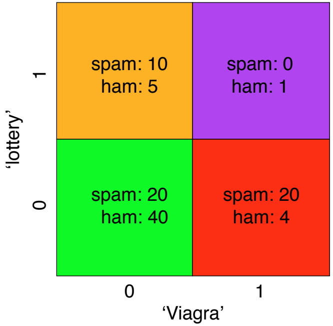
Labelling a feature tree

- ☞ The leaves of the tree in Figure 1.4 could be labelled, from left to right, as ham – spam – spam, employing a simple decision rule called *majority class*.
- ☞ Alternatively, we could label them with the proportion of spam e-mail occurring in each leaf: from left to right, 1/3, 2/3, and 4/5.
- ☞ Or, if our task was a regression task, we could label the leaves with predicted real values or even linear functions of some other, real-valued features.

Figure 1.5, p.33

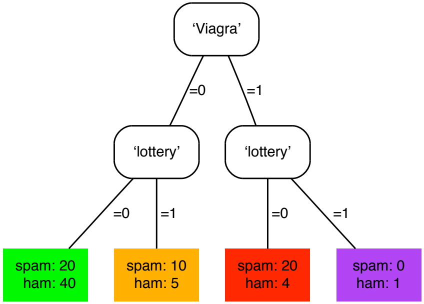
A complete feature tree





(left) A complete feature tree built from two Boolean features. **(right)** The corresponding instance space partition is the finest partition that can be achieved with those two features.

Formulation of rules from a tree



For each path from the root to a leaf:

- Collect all comparisons from the itermediate nodes
- Join the comparisons using AND
- Use majority class from the leaf as decision

Example 1.6, p.34

Overlapping rules

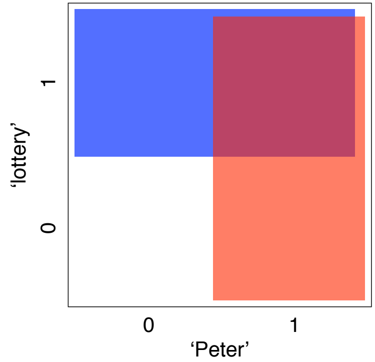
Consider the following rules:

$\cdot$**if** lottery = 1 **then** Class = Y = spam$\cdot$
$\cdot$**if** Peter = 1 **then** Class = Y = ham$\cdot$

As can be seen in Figure 1.6, these rules overlap for $\text{lottery} = 1 \wedge \text{Peter} = 1$, for which they make contradictory predictions. Furthermore, they fail to make any predictions for $\text{lottery} = 0 \wedge \text{Peter} = 0$.

Figure 1.6, p.35

Overlapping rules

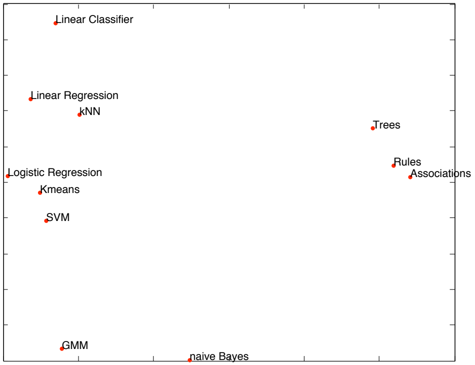


The effect of overlapping rules in instance space. The two rules make contradictory predictions in the top right-hand corner, and no prediction at all in the bottom left-hand corner.

Grouping and grading

Figure 1.7, p.37

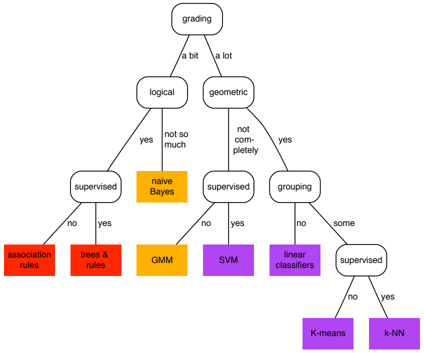
Mapping machine learning models



A ‘map’ of some of the models that will be considered in this book. Models that share characteristics are plotted closer together: logical models to the right, geometric models on the top left and probabilistic models on the bottom left. The horizontal dimension roughly ranges from grading models on the left to grouping models on the right.

Figure 1.8, p.38

ML taxonomy



A taxonomy describing machine learning methods in terms of the extent to which they are grading or grouping models, logical, geometric or a combination, and supervised or unsupervised. The colours indicate the type of model, from left to right: logical (red), probabilistic (orange) and geometric (purple).

What's next?

Example 1.7, p.39

The MLM data set

Suppose we have a number of learning models that we want to describe in terms of a number of properties:

- the extent to which the models are geometric, probabilistic or logical;
- whether they are grouping or grading models;
- the extent to which they can handle discrete and/or real-valued features;
- whether they are used in supervised or unsupervised learning; and
- the extent to which they can handle multi-class problems.

The first two properties could be expressed by discrete features with three and two values, respectively; or if the distinctions are more gradual, each aspect could be rated on some numerical scale. A simple approach would be to measure each property on an integer scale from 0 to 3, as in Table 1.4. This table establishes a data set in which each row represents an instance and each column a feature.

Table 1.4, p.39

The MLM data set

Model	geom	stats	logic	group	grad	disc	real	sup	unsup	multi
Trees	1	0	3	3	0	3	2	3	2	3
Rules	0	0	3	3	1	3	2	3	0	2
naive Bayes	1	3	1	3	1	3	1	3	0	3
kNN	3	1	0	2	2	1	3	3	0	3
Linear Classifier	3	0	0	0	3	1	3	3	0	0
Linear Regression	3	1	0	0	3	0	3	3	0	1
Logistic Regression	3	2	0	0	3	1	3	3	0	0
SVM	2	2	0	0	3	2	3	3	0	0
Kmeans	3	2	0	1	2	1	3	0	3	1
GMM	1	3	0	0	3	1	3	0	3	1
Associations	0	0	3	3	0	3	1	0	3	1

The MLM data set describing properties of machine learning models.

Many uses of features

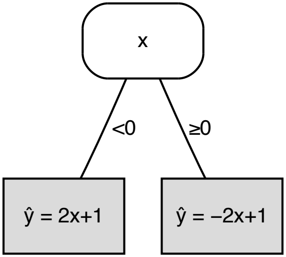
Example 1.8, p.41

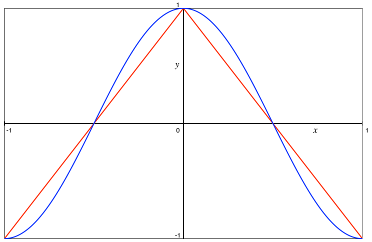
Many uses of features

Suppose we want to approximate $y = \cos \pi x$ on the interval $-1 \leq x \leq 1$. A linear approximation is not much use here, since the best fit would be $y = 0$. However, if we split the x-axis in two intervals $-1 \leq x < 0$ and $0 \leq x \leq 1$, we could find reasonable linear approximations on each interval. We can achieve this by using x both as a *splitting feature* and as a *regression variable* (Figure 1.9).

Figure 1.9, p.41

A small regression tree



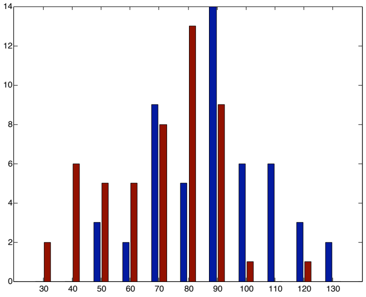


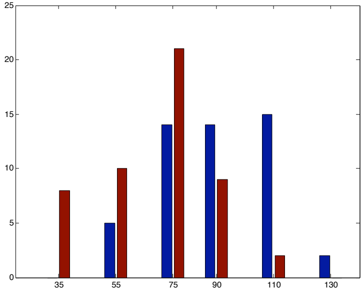
(left) A regression tree combining a one-split feature tree with linear regression models in the leaves. Notice how x is used as both a splitting feature and a regression variable. **(right)** The function $y = \cos \pi x$ on the interval $-1 \leq x \leq 1$, and the piecewise linear approximation achieved by the regression tree.

Feature construction and transformation

Figure 1.10, p.42

Class-sensitive discretisation





(left) Artificial data depicting a histogram of body weight measurements of people with (blue) and without (red) diabetes, with eleven fixed intervals of 10 kilograms width each. **(right)** By joining the first and second, third and fourth, fifth and sixth, and the eighth, ninth and tenth intervals, we obtain a discretisation such that the proportion of diabetes cases increases from left to right. This discretisation makes the feature more useful in predicting diabetes.

Example 1.9, p.43

The kernel trick

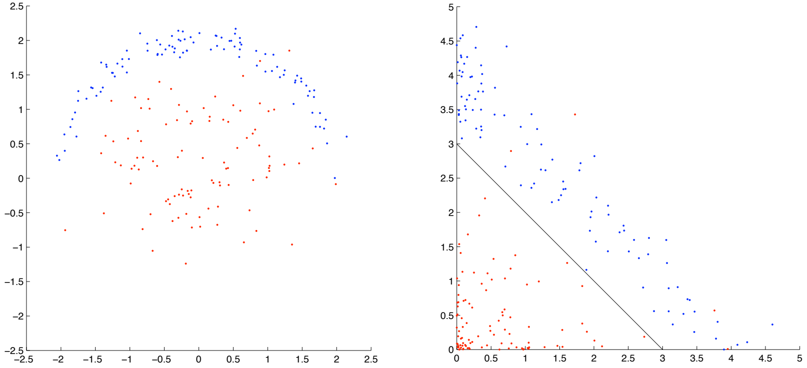
Let $\mathbf{x}_1 = (x_1, y_1)$ and $\mathbf{x}_2 = (x_2, y_2)$ be two data points, and consider the mapping $(x, y) \mapsto (x^2, y^2, \sqrt{2}xy)$ to a three-dimensional feature space. The points in feature space corresponding to $\mathbf{x}_1$ and $\mathbf{x}_2$ are $\mathbf{x}'_1 = (x_1^2, y_1^2, \sqrt{2}x_1y_1)$ and $\mathbf{x}'_2 = (x_2^2, y_2^2, \sqrt{2}x_2y_2)$. The dot product of these two feature vectors is

$$\mathbf{x}'_1 \cdot \mathbf{x}'_2 = x_1^2x_2^2 + y_1^2y_2^2 + 2x_1y_1x_2y_2 = (x_1x_2 + y_1y_2)^2 = (\mathbf{x}_1 \cdot \mathbf{x}_2)^2$$

That is, by squaring the dot product in the original space we obtain the dot product in the new space *without actually constructing the feature vectors*! A function that calculates the dot product in feature space directly from the vectors in the original space is called a *kernel* – here the kernel is $\kappa(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2)^2$.

Figure 1.11, p.43

Non-linearly separable data



(left) A linear classifier would perform poorly on this data. **(right)** By transforming the original (x,y) data into $(x',y') = (x^2,y^2)$, the data becomes more 'linear', and a linear decision boundary $x'+y' = 3$ separates the data fairly well. In the original space this corresponds to a circle with radius $\sqrt{3}$ around the origin.

What's next?

Symbols used in the following slides

Suppose the following symbols:

- ☞ $\mathscr{X}$ – set of all instances (the universe)
- ☞ $\mathscr{L}$ – set of all labels (the universe)
- ☞ $\mathscr{C}$ – set of all classes (the universe)
- ☞ $\mathscr{Y}$ – set of all outputs (the universe)
- ☞ Tr – training set of labelled instances $(x, l(x))$, where $l : \mathscr{X} \rightarrow \mathscr{L}$
- ☞ Te – test set of labelled instances $(x, l(x))$, where $l : \mathscr{X} \rightarrow \mathscr{L}$

Table 2.1, p.52

Predictive machine learning scenarios

Task	*Label space*	*Output space*	*Learning problem*
Classification	$\mathscr{L} = \mathscr{C}$	$\mathscr{Y} = \mathscr{C}$	learn an approximation $\hat{c} : \mathscr{X} \rightarrow \mathscr{C}$ to the true labelling function c
Scoring and ranking	$\mathscr{L} = \mathscr{C}$	$\mathscr{Y} = \mathbb{R}^{\lvert\mathscr{C}\rvert}$	learn a model that outputs a score vector over classes
Probability estimation	$\mathscr{L} = \mathscr{C}$	$\mathscr{Y} = [0,1]^{\lvert\mathscr{C}\rvert}$	learn a model that outputs a probability vector over classes
Regression	$\mathscr{L} = \mathbb{R}$	$\mathscr{Y} = \mathbb{R}$	learn an approximation $\hat{f} : \mathscr{X} \rightarrow \mathbb{R}$ to the true labelling function f

What's next?

Classification

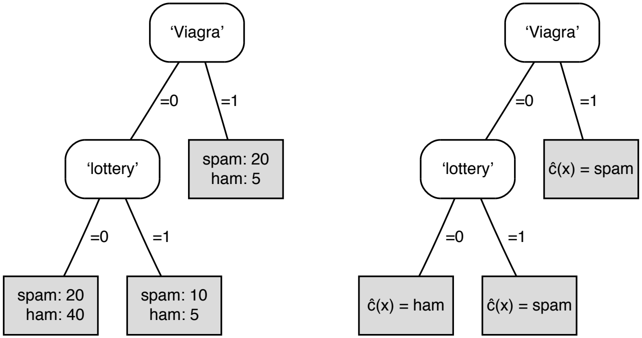
A *classifier* is a mapping $\hat{c} : \mathscr{X} \to \mathscr{C}$, where $\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$ is a finite and usually small set of *class labels*. We will sometimes also use C_i to indicate the set of examples of that class.

We use the 'hat' to indicate that $\hat{c}(x)$ is an estimate of the true but unknown function $c(x)$. Examples for a classifier take the form $(x, c(x))$, where $x \in \mathscr{X}$ is an instance and $c(x)$ is the true class of the instance (sometimes contaminated by noise).

Learning a classifier involves constructing the function $\hat{c}$ such that it matches c as closely as possible (and not just on the training set, but ideally on the entire instance space $\mathscr{X}$).

Figure 2.1, p.53

A decision tree



(left) A feature tree with training set class distribution in the leaves. **(right)** A decision tree obtained using the majority class decision rule.

Assessing classification performance

Table 2.2, p.54

Contingency table

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**30**	**20**	50
Actual ⊖	**10**	**40**	50
	40	60	100

	⊕	⊖	
⊕	**20**	**30**	50
⊖	**20**	**30**	50
	40	60	100

(left) A two-class contingency table or confusion matrix depicting the performance of the decision tree in Figure 2.1. Numbers on the descending diagonal indicate correct predictions, while the ascending diagonal concerns prediction errors. **(right)** A contingency table with the same marginals but independent rows and columns.

Statistics from contingency table

Let's label numbers of a classifier's predictions on a test set as in the table:

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**TP**	**FN**	Pos
Actual ⊖	**FP**	**TN**	Neg
	0	0	0

Where abbreviations stand for:

- TP – true positives
- FP – false positives
- FN – false negatives
- TN – true negatives
- Pos – number of positive examples
- Neg – number of negative examples

Table 2.3, p.57

Performance measures I

Measure	*Definition*	*Equal to*	*Estimates*
number of positives	$Pos = \sum_{x \in Te} I[c(x) = \oplus]$		
number of negatives	$Neg = \sum_{x \in Te} I[c(x) = \ominus]$	$\|Te\| - Pos$	
number of true positives	$TP = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]$		
number of true negatives	$TN = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \ominus]$		
number of false positives	$FP = \sum_{x \in Te} I[\hat{c}(x) = \oplus, c(x) = \ominus]$	$Neg - TN$	
number of false negatives	$FN = \sum_{x \in Te} I[\hat{c}(x) = \ominus, c(x) = \oplus]$	$Pos - TP$	
proportion of positives	$pos = \frac{1}{\|Te\|} \sum_{x \in Te} I[c(x) = \oplus]$	$Pos/\|Te\|$	$P(c(x) = \oplus)$
proportion of negatives	$neg = \frac{1}{\|Te\|} \sum_{x \in Te} I[c(x) = \ominus]$	$1 - pos$	$P(c(x) = \ominus)$
class ratio	$clr = pos/neg$	Pos/Neg	
(*) accuracy	$acc = \frac{1}{\|Te\|} \sum_{x \in Te} I[\hat{c}(x) = c(x)]$		$P(\hat{c}(x) = c(x))$
(*) error rate	$err = \frac{1}{\|Te\|} \sum_{x \in Te} I[\hat{c}(x) \neq c(x)]$	$1 - acc$	$P(\hat{c}(x) \neq c(x))$

Table 2.3, p.57

Performance measures II

Measure	*Definition*	*Equal to*	*Estimates*
true positive rate, sensitivity, recall	$tpr = \frac{\sum_{x \in Te} I[\hat{c}(x)=c(x)=\oplus]}{\sum_{x \in Te} I[c(x)=\oplus]}$	TP/Pos	$P(\hat{c}(x) = \oplus \| c(x) = \oplus)$
true negative rate, specificity	$tnr = \frac{\sum_{x \in Te} I[\hat{c}(x)=c(x)=\ominus]}{\sum_{x \in Te} I[c(x)=\ominus]}$	TN/Neg	$P(\hat{c}(x) = \ominus \| c(x) = \ominus)$
false positive rate, false alarm rate	$fpr = \frac{\sum_{x \in Te} I[\hat{c}(x)=\oplus, c(x)=\ominus]}{\sum_{x \in Te} I[c(x)=\ominus]}$	$FP/Neg = 1 - tnr$	$P(\hat{c}(x) = \oplus \| c(x) = \ominus)$
false negative rate	$fnr = \frac{\sum_{x \in Te} I[\hat{c}(x)=\ominus, c(x)=\oplus]}{\sum_{x \in Te} I[c(x)=\oplus]}$	$FN/Pos = 1 - tpr$	$P(\hat{c}(x) = \ominus \| c(x) = \oplus)$
precision, confidence	$prec = \frac{\sum_{x \in Te} I[\hat{c}(x)=c(x)=\oplus]}{\sum_{x \in Te} I[\hat{c}(x)=\oplus]}$	$TP/(TP+FP)$	$P(c(x) = \oplus \| \hat{c}(x) = \oplus)$

Table: A summary of different quantities and evaluation measures for classifiers on a test set Te. Symbols starting with a capital letter denote absolute frequencies (counts), while lower-case symbols denote relative frequencies or ratios. All except those indicated with (*) are defined only for binary classification.

Example 2.1, p.56

Accuracy as a weighted average

Suppose a classifier's predictions on a test set are as in the following table:

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**60**	**15**	75
Actual ⊖	**10**	**15**	25
	70	30	100

From this table, we see that the true positive rate is $tpr = 60/75 = 0.80$ and the true negative rate is $tnr = 15/25 = 0.60$. The overall accuracy is $acc = (60+15)/100 = 0.75$, which is no longer the average of true positive and negative rates. However, taking into account the proportion of positives $pos = 0.75$ and the proportion of negatives $neg = 1 - pos = 0.25$, we see that

$$acc = pos \cdot tpr + neg \cdot tnr$$

Visualising classification performance

Degrees of freedom

The following contingency table:

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**TP**	**FN**	Pos
Actual ⊖	**FP**	**TN**	Neg
	0	0	0

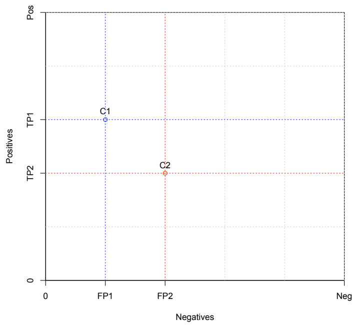
contains 9 values, however some of them depend on others: e.g., marginal sums depend on rows and columns, respectively. Actually, we need only 4 values to determine the rest of them. Thus, we say that this table has **4 degrees of freedom**. In general table having $(k+1)^2$ entries has k^2 degrees of freedom.

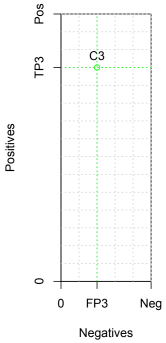
In the following, we assume that Pos, Neg, **TP** and **FP** are enough to reconstruct whole table.

Figure 2.2, p.58

A coverage plot

Let there be classifiers C1, C2 and C3.

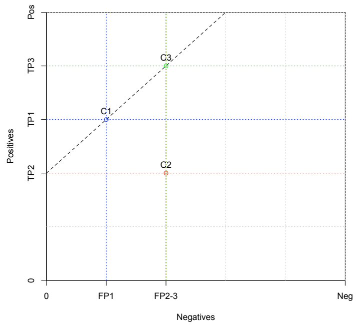


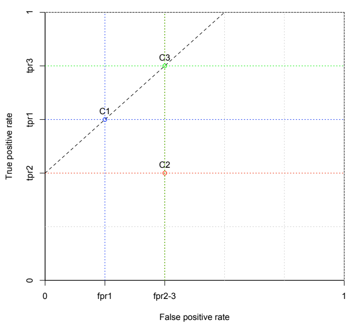


(left) A coverage plot depicting the two contingency tables in Table 2.2. The plot is square because the class distribution is uniform. From the plot we immediately see that C1 is better than C2. **(right)** Coverage plot for Example 2.1, with a class ratio $clr = 3$.

Figure 2.3, p.59

An ROC plot

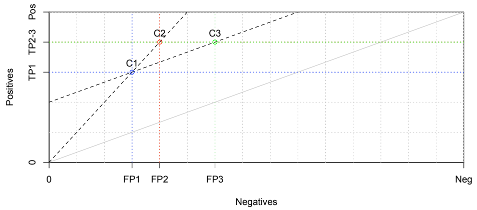


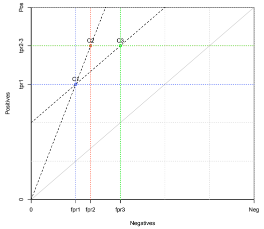


(left) C1 and C3 dominate C2, but neither dominates the other. **The diagonal line having slope of 1 indicates that all classifiers on this line achieve equal accuracy.** **(right)** Receiver Operating Characteristic (ROC) plot: a merger of the two coverage plots in Figure 2.2, employing normalisation to deal with the different class distributions. **The diagonal line having slope of 1 indicates that all classifiers on this line have the same average recall** (average of positive and negative recalls).

Figure 2.4, p.61

Comparing coverage and ROC plots





(left) In a coverage plot, accuracy isometrics have a slope of 1, and average recall isometrics are parallel to the ascending diagonal. **(right)** In the corresponding ROC plot, average recall isometrics have a slope of 1; the accuracy isometric here has a slope of 3, corresponding to the ratio of negatives to positives in the data set.

What's next?

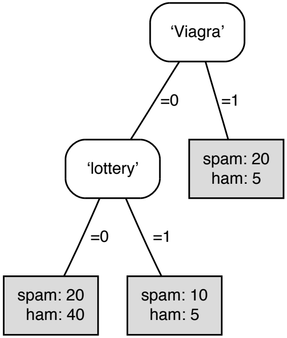
Scoring classifier

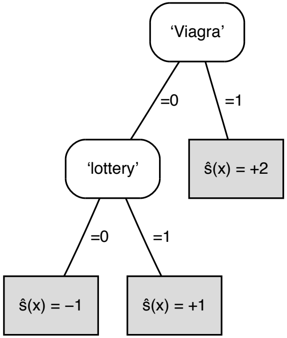
A *scoring classifier* is a mapping $\hat{\mathbf{s}}: \mathscr{X} \rightarrow \mathbb{R}^k$, i.e., a mapping from the instance space to a k-vector of real numbers.
The boldface notation indicates that a scoring classifier outputs a vector $\hat{\mathbf{s}}(x) = (\hat{s}_1(x), \ldots, \hat{s}_k(x))$ rather than a single number; $\hat{s}_i(x)$ is the score assigned to class C_i for instance x.
This score indicates how likely it is that class label C_i applies.

If we only have two classes, it usually suffices to consider the score for only one of the classes; in that case, we use $\hat{s}(x)$ to denote the score of the positive class for instance x.

Figure 2.5, p.62

A scoring tree





(left) A feature tree with training set class distribution in the leaves. **(right)** A scoring tree using the logarithm of the class ratio as scores; spam is taken as the positive class.

Margins and loss functions

If we take the true class $c(x)$ as $+1$ for positive examples and -1 for negative examples, then the quantity $z(x) = c(x)\hat{s}(x)$ is positive for correct predictions and negative for incorrect predictions: this quantity is called the *margin* assigned by the scoring classifier to the example.

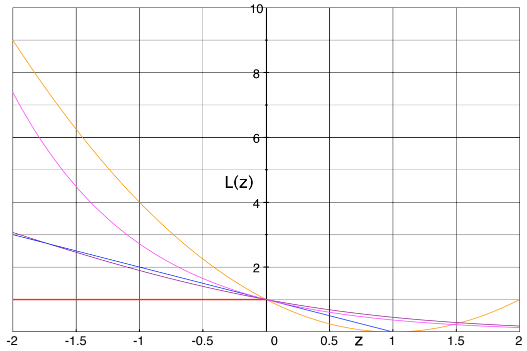
We would like to reward large positive margins, and penalise large negative values. This is achieved by means of a so-called *loss function* $L : \mathbb{R} \mapsto [0, \infty)$ which maps each example's margin $z(x)$ to an associated loss $L(z(x))$.

We will assume that $L(0) = 1$, which is the loss incurred by having an example on the decision boundary. We furthermore have $L(z) \geq 1$ for $z < 0$, and usually also $0 \leq L(z) < 1$ for $z > 0$ (Figure 2.6).

The average loss over a test set Te is $\frac{1}{|Te|} \sum_{x \in Te} L(z(x))$.

Figure 2.6, p.63

Loss functions



From bottom-left (*i*) 0–1 loss $L_{01}(z) = 1$ if $z \leq 0$, and $L_{01}(z) = 0$ if $z > 0$;
(*ii*) hinge loss $L_{\mathrm{h}}(z) = (1 - z)$ if $z \leq 1$, and $L_{\mathrm{h}}(z) = 0$ if $z > 1$;
(*iii*) logistic loss $L_{\log}(z) = \log_2(1 + \exp(-z))$;
(*iv*) exponential loss $L_{\exp}(z) = \exp(-z)$;
(*v*) squared loss $L_{\mathrm{sq}}(z) = (1 - z)^2$ (this can be set to 0 for $z > 1$, just like hinge loss).

Assessing and visualising ranking performance

Example 2.2, p.64

Ranking example

- The scoring tree in Figure 2.5 produces the following ranking: $[20+,5-][10+,5-][20+,40-]$. Here, $20+$ denotes a sequence of 20 positive examples, and instances in square brackets $[\dots]$ are tied.
- By selecting a split point in the ranking we can turn the ranking into a classification. In this case there are four possibilities:
  - (A) setting the split point before the first segment, and thus assigning all segments to the negative class;
  - (B) assigning the first segment to the positive class, and the other two to the negative class;
  - (C) assigning the first two segments to the positive class; and
  - (D) assigning all segments to the positive class.
- In terms of actual scores, this corresponds to (A) choosing any score larger than 2 as the threshold; (B) choosing a threshold between 1 and 2; (C) setting the threshold between -1 and 1; and (D) setting it lower than -1.

Example 2.3, p.65

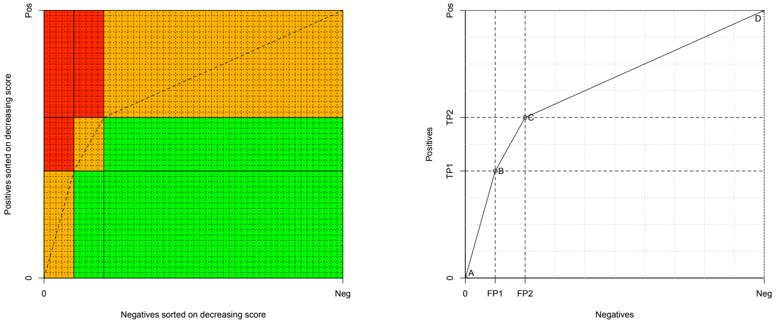
Ranking accuracy

The *ranking error rate* is defined as

$$rank\text{-}err = \frac{\sum_{x \in Te^{\oplus}, x' \in Te^{\ominus}} I[\hat{s}(x) < \hat{s}(x')] + \frac{1}{2} I[\hat{s}(x) = \hat{s}(x')]}{Pos \cdot Neg}$$

- ☞ The 5 negatives in the right leaf are scored higher than the 10 positives in the middle leaf and the 20 positives in the left leaf, resulting in $50 + 100 = 150$ ranking errors.
- ☞ The 5 negatives in the middle leaf are scored higher than the 20 positives in the left leaf, giving a further 100 ranking errors.
- ☞ In addition, the left leaf makes 800 half ranking errors (because 20 positives and 40 negatives get the same score), the middle leaf 50 and the right leaf 100.
- ☞ In total we have 725 ranking errors out of a possible $50 \cdot 50 = 2500$, corresponding to a ranking error rate of 29% or a ranking accuracy of 71%.

Figure 2.7, p.66 Coverage curve



(left) Each cell in the grid denotes a unique pair of one positive and one negative example: the green cells indicate pairs that are correctly ranked by the classifier, the red cells represent ranking errors, and the orange cells are half-errors due to ties. **(right)** The coverage curve of a tree-based scoring classifier has one line segment for each leaf of the tree, and one (FP, TP) pair for each possible threshold on the score.

Important point to remember

ROC curve is obtained from the coverage curve by normalizing the axes to range $[0,1]$.
The area under the ROC curve is the ranking accuracy.

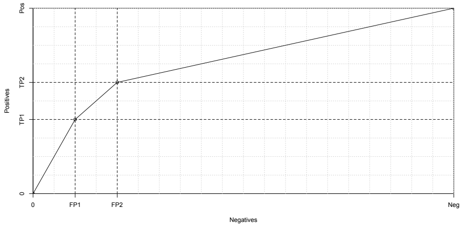
Example 2.4, p.67

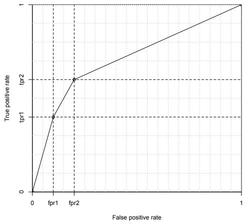
Class imbalance

- ☞ Suppose we feed the scoring tree in Figure 2.5 an extended test set, with an additional batch of 50 negatives.
- ☞ The added negatives happen to be identical to the original ones, so the net effect is that the number of negatives in each leaf doubles.
- ☞ As a result the coverage curve changes (because the class ratio changes), but the ROC curve stays the same (Figure 2.8).
- ☞ Note that the ranking accuracy stays the same as well: while the classifier makes twice as many ranking errors, there are also twice as many positive–negative pairs, so the ranking error rate doesn't change.

Figure 2.8, p.67

Class imbalance





(left) A coverage curve obtained from a test set with class ratio $clr = 1/2$. **(right)** The corresponding (axis-normalised) ROC curve is the same as the one corresponding to the coverage curve in Figure 2.7 (right). The ranking accuracy is the Area Under the ROC Curve (AUC).

Rankings from grading classifiers

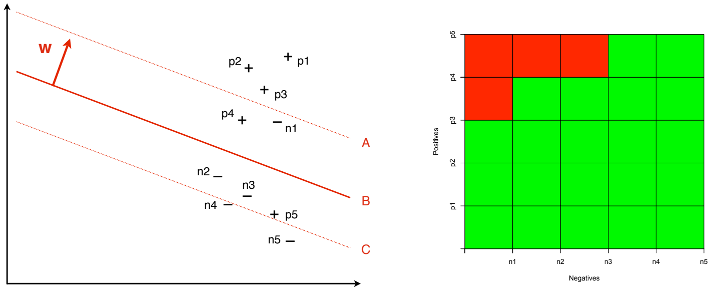
Figure 2.9 (left) shows a linear classifier (the decision boundary is denoted B) applied to a small data set of five positive and five negative examples, achieving an accuracy of 0.80.

We can derive a score from this linear classifier by taking the distance of an example from the decision boundary; if the example is on the negative side we take the negative distance. This means that the examples are ranked in the following order: p1 – p2 – p3 – n1 – p4 – n2 – n3 – p5 – n4 – n5.

This ranking incurs four ranking errors: n1 before p4, and n1, n2 and n3 before p5. Figure 2.9 (right) visualises these four ranking errors in the top-left corner. The AUC of this ranking is $21/25 = 0.84$.

Figure 2.9, p.68

Rankings from grading classifiers



(left) A linear classifier induces a ranking by taking the signed distance to the decision boundary as the score. This ranking only depends on the orientation of the decision boundary: the three lines result in exactly the same ranking. **(right)** The grid of correctly ranked positive–negative pairs (in green) and ranking errors (in red).

Tuning rankers

Example 2.5, p.70

Tuning your spam filter I

You have carefully trained your Bayesian spam filter, and all that remains is setting the decision threshold. You select a set of six spam and four ham e-mails and collect the scores assigned by the spam filter. Sorted on decreasing score these are 0.89 (spam), 0.80 (spam), 0.74 (ham), 0.71 (spam), 0.63 (spam), 0.49 (ham), 0.42 (spam), 0.32 (spam), 0.24 (ham), and 0.13 (ham).

If the class ratio of 6 spam against 4 ham is representative, you can select the optimal point on the ROC curve using an isometric with slope 4/6. As can be seen in Figure 2.11, this leads to putting the decision boundary between the sixth spam e-mail and the third ham e-mail, and we can take the average of their scores as the decision threshold (0.28).

Example 2.5, p.70

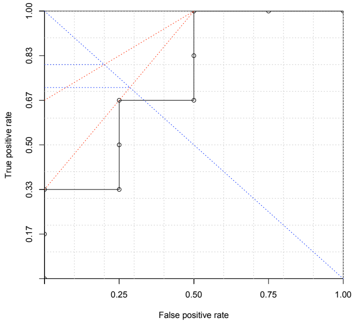
Tuning your spam filter II

An alternative way of finding the optimal point is to iterate over all possible split points – from before the top ranked e-mail to after the bottom one – and calculate the number of correctly classified examples at each split: 4 – 5 – 6 – 5 – 6 – 7 – 6 – 7 – 8 – 7 – 6. The maximum is achieved at the same split point, yielding an accuracy of 0.80.

A useful trick to find out which accuracy an isometric in an ROC plot represents is to intersect the isometric with the descending diagonal. Since accuracy is a weighted average of the true positive and true negative rates, and since these are the same in a point on the descending diagonal, we can read off the corresponding accuracy value on the y-axis.

Figure 2.11, p.71

Finding the optimal point



Selecting the optimal point on an ROC curve. The top dotted line is the accuracy isometric, with a slope of 2/3. The lower isometric doubles the value (or prevalence) of negatives, and allows a choice of thresholds. By intersecting the isometrics with the descending diagonal we can read off the achieved accuracy on the y-axis.

What's next?

Class probability estimation

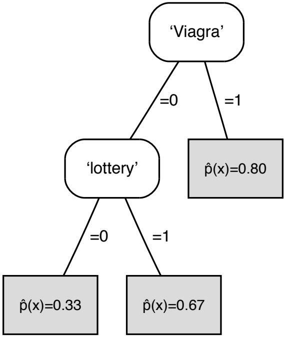
A *class probability estimator* – or probability estimator in short – is a scoring classifier that outputs probability vectors over classes, i.e., a mapping $\hat{\mathbf{p}} : \mathscr{X} \to [0,1]^k$. We write $\hat{\mathbf{p}}(x) = \big(\hat{p}_1(x), \ldots, \hat{p}_k(x)\big)$, where $\hat{p}_i(x)$ is the probability assigned to class C_i for instance x, and $\sum_{i=1}^{k} \hat{p}_i(x) = 1$.

If we have only two classes, the probability associated with one class is 1 minus the probability of the other class; in that case, we use $\hat{p}(x)$ to denote the estimated probability of the positive class for instance x.

As with scoring classifiers, we usually do not have direct access to the true probabilities $p_i(x)$.

Figure 2.12, p.73

Probability estimation tree



A probability estimation tree derived from the feature tree in Figure 1.4.

Assessing class probability estimates

Mean squared probability error

We can define the *squared error* (*SE*) of the predicted probability vector $\hat{\mathbf{p}}(x) = \left(\hat{p}_1(x), \ldots, \hat{p}_k(x)\right)$ as

$$\mathrm{SE}(x) = \frac{1}{2}\sum_{i=1}^{k}(\hat{p}_i(x) - I[c(x) = C_i])^2$$

and the *mean squared error* (*MSE*) as the average squared error over all instances in the test set:

$$\mathrm{MSE}(Te) = \frac{1}{|Te|}\sum_{x \in Te}\mathrm{SE}(x)$$

The factor 1/2 in Equation 2.6 ensures that the squared error per example is normalised between 0 and 1: the worst possible situation is that a wrong class is predicted with probability 1, which means two 'bits' are wrong.
For two classes this reduces to a single term $(\hat{p}(x) - I[c(x) = \oplus])^2$ only referring to the positive class.

Example 2.6, p.74

Squared error

Suppose one model predicts $(0.70, 0.10, 0.20)$ for a particular example x in a three-class task, while another appears much more certain by predicting $(0.99, 0, 0.01)$.

- If the first class is the actual class, the second prediction is clearly better than the first: the SE of the first prediction is $((0.70-1)^2+(0.10-0)^2+(0.20-0)^2)/2 = 0.07$, while for the second prediction it is $((0.99-1)^2+(0-0)^2+(0.01-0)^2)/2 = 0.0001$. The first model gets punished more because, although mostly right, it isn't quite sure of it.
- However, if the third class is the actual class, the situation is reversed: now the SE of the first prediction is $((0.70-0)^2+(0.10-0)^2+(0.20-1)^2)/2 = 0.57$, and of the second $((0.99-0)^2+(0-0)^2+(0.01-1)^2)/2 = 0.98$. The second model gets punished more for not just being wrong, but being presumptuous.

Which probabilities achieve lowest MSE?

Returning to the probability estimation tree in Figure 2.12, we calculate the squared error per leaf as follows (left to right):

$$\mathrm{SE}_1 = 20(0.33-1)^2 + 40(0.33-0)^2 = 13.33$$
$$\mathrm{SE}_2 = 10(0.67-1)^2 + 5(0.67-0)^2 = 3.33$$
$$\mathrm{SE}_3 = 20(0.80-1)^2 + 5(0.80-0)^2 = 4.00$$

which leads to a mean squared error of $\mathrm{MSE} = \frac{1}{100}(\mathrm{SE}_1 + \mathrm{SE}_2 + \mathrm{SE}_3) = 0.21$. Changing the predicted probabilities in the left-most leaf to 0.40 for spam and 0.60 for ham, or 0.20 for spam and 0.80 for ham, results in a higher squared error:

$$\mathrm{SE}_1' = 20(0.40-1)^2 + 40(0.40-0)^2 = 13.6$$
$$\mathrm{SE}_1'' = 20(0.20-1)^2 + 40(0.20-0)^2 = 14.4$$

Predicting probabilities obtained from the class distributions in each leaf is optimal in the sense of lowest MSE.

Why predicting empirical probabilities is optimal

The reason for this becomes obvious if we rewrite the expression for two-class squared error of a leaf as follows, using the notation $n^{\oplus}$ and $n^{\ominus}$ for the numbers of positive and negative examples in the leaf:

$$\begin{aligned} n^{\oplus}(\hat{p}-1)^2 + n^{\ominus}\hat{p}^2 = (n^{\oplus}+n^{\ominus})\hat{p}^2 - 2n^{\oplus}\hat{p} + n^{\oplus} &= (n^{\oplus}+n^{\ominus})\left[\hat{p}^2 - 2\dot{p}\hat{p} + \dot{p}\right] \\ &= (n^{\oplus}+n^{\ominus})\left[(\hat{p}-\dot{p})^2 + \dot{p}(1-\dot{p})\right] \end{aligned}$$

where $\dot{p} = n^{\oplus}/(n^{\oplus}+n^{\ominus})$ is the relative frequency of the positive class among the examples covered by the leaf, also called the *empirical probability*. As the term $\dot{p}(1-\dot{p})$ does not depend on the predicted probability $\hat{p}$, we see immediately that we achieve lowest squared error in the leaf if we assign $\hat{p} = \dot{p}$.

Smoothing empirical probabilities

It is almost always a good idea to *smooth* these relative frequencies. The most common way to do this is by means of the *Laplace correction*:

$$\dot{p}_i(S) = \frac{n_i + 1}{|S| + k}$$

In effect, we are adding uniformly distributed *pseudo-counts* to each of the k alternatives, reflecting our prior belief that the empirical probabilities will turn out uniform.

We can also apply non-uniform smoothing by setting

$$\dot{p}_i(S) = \frac{n_i + m \cdot \pi_i}{|S| + m}$$

This smoothing technique, known as the *m-estimate*, allows the choice of the number of pseudo-counts m as well as the prior probabilities π_i. The Laplace correction is a special case of the m-estimate with $m = k$ and $\pi_i = 1/k$.

What's next?

3. Beyond binary classification
   - Handling more than two classes
     - Multi-class classification
     - Multi-class scores and probabilities
   - Regression
   - Unsupervised and descriptive learning
     - Predictive and descriptive clustering
     - Other descriptive models

What's next?

Multi-class classification

Example 3.1, p.82

Performance of multi-class classifiers I

Consider the following three-class confusion matrix (plus marginals):

	Predicted			
	15	**2**	**3**	20
Actual	**7**	**15**	**8**	30
	2	**3**	**45**	50
	24	20	56	100

- The accuracy of this classifier is $(15+15+45)/100 = 0.75$.
- We can calculate per-class precision and recall: for the first class this is $15/24 = 0.63$ and $15/20 = 0.75$ respectively, for the second class $15/20 = 0.75$ and $15/30 = 0.50$, and for the third class $45/56 = 0.80$ and $45/50 = 0.90$.

Example 3.1, p.82

Performance of multi-class classifiers II

☞ We could average these numbers to obtain single precision and recall numbers for the whole classifier, or we could take a weighted average taking the proportion of each class into account. For instance, the weighted average precision is $0.20 \cdot 0.63 + 0.30 \cdot 0.75 + 0.50 \cdot 0.80 = 0.75$.

☞ Another possibility is to perform a more detailed analysis by looking at precision and recall numbers for each pair of classes: for instance, when distinguishing the first class from the third precision is $15/17 = 0.88$ and recall is $15/18 = 0.83$, while distinguishing the third class from the first these numbers are $45/48 = 0.94$ and $45/47 = 0.96$ (can you explain why these numbers are much higher in the latter direction?).

Construction of multi-class classifiers

Suppose we want to build k-class classifier, but have ability to train only two-class ones. We have two alternative schemes to do so:

- ☞ One-versus-rest – we train k binary classifiers separately for each class C_i from $C_1, ..., C_k$, where C_i is treated as $\oplus$, and all remaining classes as $\ominus$
- ☞ One-versus-one – we train at least $\frac{k(k-1)}{2}$ classifiers for each pair of classess C_i and C_j treating them as $\oplus$ and $\ominus$, respectively. Different one-versus-one schemes can be described by means of **output code** matrix:

$$\begin{pmatrix} +1 & +1 & 0 \\ -1 & 0 & +1 \\ 0 & -1 & -1 \end{pmatrix}$$

  where each column describes a binary classification task, using the class in the row with $+1$ entry as $\oplus$ and the class in the row with -1 entry as $\ominus$.

Example 3.2, p.85

One-versus-one voting I

A one-versus-one code matrix for $k = 4$ classes is as follows:

$$\begin{pmatrix} +1 & +1 & +1 & 0 & 0 & 0 \\ -1 & 0 & 0 & +1 & +1 & 0 \\ 0 & -1 & 0 & -1 & 0 & +1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{pmatrix}$$

Suppose our six pairwise classifiers predict $w = +1\ -1\ +1\ -1\ +1\ +1$. We can interpret this as votes for $C_1 - C_3 - C_1 - C_3 - C_2 - C_3$; i.e., three votes for C_3, two votes for C_1 and one vote for C_2. More generally, the i-th classifier's vote for the j-th class can be expressed as $(1 + w_i c_{ji})/2$, where c_{ji} is the entry in the j-th row and i-th column of the code matrix.

Example 3.2, p.85

One-versus-one voting II

However, this overcounts the 0 entries in the code matrix; since every class participates in $k-1$ pairwise binary tasks, and there are $l = k(k-1)/2$ tasks, the number of zeros in every row is $k(k-1)/2-(k-1) = (k-1)(k-2)/2 = l(k-2)/k$ (3 in our case). For each zero we need to subtract half a vote, so the number of votes for C_j is

$$\begin{aligned} v_j &= \left(\sum_{i=1}^{l} \frac{1+w_i c_{ji}}{2}\right) - l\frac{k-2}{2k} = \left(\sum_{i=1}^{l} \frac{w_i c_{ji}-1}{2}\right) + l - l\frac{k-2}{2k} \\ &= -d_j + l\frac{2k-k+2}{2k} = \frac{(k-1)(k+2)}{4} - d_j \end{aligned}$$

where $d_j = \sum_i (1-w_i c_{ji})/2$ is a bit-wise distance measure.

Example 3.2, p.85

One-versus-one voting III

In other words, the distance and number of votes for each class sum to a constant depending only on the number of classes; with three classes this is 4.5. This can be checked by noting that

- the distance between w and the first code word is 2.5 (two votes for C_1);
- with the second code word, 3.5 (one vote for C_2);
- with the third code word, 1.5 (three votes for C_3);
- and 4.5 with the fourth code word (no votes).

So voting and distance-based decoding are equivalent in this case.

Example 3.3, p.86

Loss-based decoding

Continuing the previous example, suppose the scores of the six pairwise classifiers are $(+5, -0.5, +4, -0.5, +4, +0.5)$. This leads to the following margins, in matrix form:

$$\begin{pmatrix} +5 & -0.5 & +4 & 0 & 0 & 0 \\ -5 & 0 & 0 & -0.5 & +4 & 0 \\ 0 & +0.5 & 0 & +0.5 & 0 & +0.5 \\ 0 & 0 & -4 & 0 & -4 & -0.5 \end{pmatrix}$$

Using 0–1 loss we ignore the magnitude of the margins and thus predict C_3 as in the voting-based scheme of Example 3.2. Using exponential loss $L(z) = \exp(-z)$, we obtain the distances $(4.67, 153.08, 4.82, 113.85)$. Loss-based decoding would therefore (just) favour C_1, by virtue of its strong wins against C_2 and C_4; in contrast, all three wins of C_3 are with small margin.

Multi-class scores and probabilities

Example 3.4, p.87

Coverage counts as scores I

Suppose we have three classes and three binary classifiers which either predict positive or negative (there is no reject option).

The first classifier classifies 8 examples of the first class as positive, no examples of the second class, and 2 examples of the third class. For the second classifier these counts are 2, 17 and 1, and for the third they are 4, 2 and 8.

Suppose a test instance is predicted as positive by the first and third classifiers. We can add the coverage counts of these two classifiers to obtain a score vector of $(12,2,10)$. Likewise, if all three classifiers 'fire' for a particular test instance (i.e., predict positive), the score vector is $(14,19,11)$.

Example 3.4, p.87

Coverage counts as scores II

We can describe this scheme conveniently using matrix notation:

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 8 & 0 & 2 \\ 2 & 17 & 1 \\ 4 & 2 & 8 \end{pmatrix} = \begin{pmatrix} 12 & 2 & 10 \\ 14 & 19 & 11 \end{pmatrix}$$

The middle matrix contains the class counts (one row for each classifier). The left 2-by-3 matrix contains, for each example, a row indicating which classifiers fire for that example. The right-hand side then gives the combined counts for each example.

Example 3.5, p.88

Multi-class AUC I

Assume we have a multi-class scoring classifier that produces a k-vector of scores $\hat{\mathbf{s}}(x) = (\hat{s}_1(x), \ldots, \hat{s}_k(x))$ for each test instance x.

☞ By restricting attention to $\hat{s}_i(x)$ we obtain a scoring classifier for class C_i against the other classes, and we can calculate the one-versus-rest AUC for C_i in the normal way.

☞ By way of example, suppose we have three classes, and the one-versus-rest AUCs come out as 1 for the first class, 0.8 for the second class and 0.6 for the third class. Thus, for instance, all instances of class 1 receive a higher first entry in their score vectors than any of the instances of the other two classes.

☞ The average of these three AUCs is 0.8, which reflects the fact that, if we uniformly choose an index i, and we select an instance x uniformly among class C_i and another instance x' uniformly among all instances not from C_i, then the expectation that $\hat{s}_i(x) > \hat{s}_i(x')$ is 0.8.

Example 3.5, p.88

Multi-class AUC II

- Suppose now C_1 has 10 instances, C_2 has 20 and C_3 70.
- The weighted average of the one-versus-rest AUCs is then 0.68: that is, if we uniformly choose x *without reference to the class*, and then choose x' uniformly from among all instances not of the same class as x', the expectation that $\hat{s}_i(x) > \hat{s}_i(x')$ is 0.68.
- This is lower than before, because it is now more likely that a random x comes from class C_3, whose scores do a worse ranking job.

One-versus-one AUC

We can obtain similar averages from one-versus-one AUCs.

- ☞ For instance, we can define AUC_{ij} as the AUC obtained using scores $\hat{s}_i$ to rank instances from classes C_i and C_j. Notice that $\hat{s}_j$ may rank these instances differently, and so $\text{AUC}_{ji} \neq \text{AUC}_{ij}$.
- ☞ Taking an unweighted average over all $i \neq j$ estimates the probability that, for uniformly chosen classes i and $j \neq i$, and uniformly chosen $x \in C_i$ and $x' \in C_j$, we have $\hat{s}_i(x) > \hat{s}_i(x')$.
- ☞ The weighted version of this estimates the probability that the instances are correctly ranked if we don't pre-select the class.

Example 3.7, p.90

Multi-class probabilities I

In Example 3.4 we can divide the class counts by the total number of positive predictions. This results in the following class distributions: $(0.80, 0, 0.20)$ for the first classifier, $(0.10, 0.85, 0.05)$ for the second classifier, and $(0.29, 0.14, 0.57)$ for the third. The probability distribution associated with the combination of the first and third classifiers is

$$\frac{10}{24}(0.80, 0, 0.20) + \frac{14}{24}(0.29, 0.14, 0.57) = (0.50, 0.08, 0.42)$$

which is the same distribution as obtained by normalising the combined counts $(12, 2, 10)$. Similarly, the distribution associated with all three classifiers is

$$\frac{10}{44}(0.80, 0, 0.20) + \frac{20}{44}(0.10, 0.85, 0.05) + \frac{14}{44}(0.29, 0.14, 0.57) = (0.32, 0.43, 0.25)$$

Example 3.7, p.90

Multi-class probabilities II

Matrix notation describes this very succinctly as

$$\begin{pmatrix} 10/24 & 0 & 14/24 \\ 10/44 & 20/44 & 14/44 \end{pmatrix} \begin{pmatrix} 0.80 & 0.00 & 0.20 \\ 0.10 & 0.85 & 0.05 \\ 0.29 & 0.14 & 0.57 \end{pmatrix} = \begin{pmatrix} 0.50 & 0.08 & 0.42 \\ 0.32 & 0.43 & 0.25 \end{pmatrix}$$

The middle matrix is a row-normalised version of the middle matrix in Equation 3.1. *Row normalisation* works by dividing each entry by the sum of the entries in the row in which it occurs. As a result the entries in each row sum to one, which means that each row can be interpreted as a probability distribution. The left matrix combines two pieces of information: (*i*) which classifiers fire for each example (for instance, the second classifier doesn't fire for the first example); and (*ii*) the coverage of each classifier. The right-hand side then gives the class distribution for each example. Notice that the product of row-normalised matrices again gives a row-normalised matrix.

What's next?

Real-valued targets

A *function estimator*, also called a *regressor*, is a mapping $\hat{f} : \mathscr{X} \to \mathbb{R}$. The regression learning problem is to learn a function estimator from examples $(x_i, f(x_i))$.

Note that we switched from a relatively low-resolution target variable to one with infinite resolution. Trying to match this precision in the function estimator will almost certainly lead to overfitting – besides, it is highly likely that some part of the target values in the examples is due to fluctuations that the model is unable to capture.

It is therefore entirely reasonable to assume that the examples are noisy, and that the estimator is only intended to capture the general trend or shape of the function.

Example 3.8, p.92

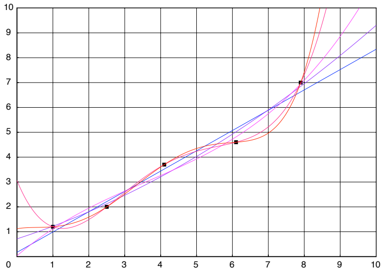
Line fitting example

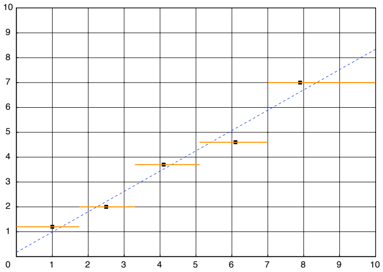
Consider the following set of five points:

x	y
1.0	1.2
2.5	2.0
4.1	3.7
6.1	4.6
7.9	7.0

We want to estimate y by means of a polynomial in x. Figure 3.2 (left) shows the result for degrees of 1 to 5 using ☞*linear regression*, which will be explained in Chapter 7. The top two degrees fit the given points exactly (in general, any set of n points can be fitted by a polynomial of degree no more than $n-1$), but they differ considerably at the extreme ends: e.g., the polynomial of degree 4 leads to a decreasing trend from $x=0$ to $x=1$, which is not really justified by the data.

Figure 3.2, p.92 Fitting polynomials to data





(left) Polynomials of different degree fitted to a set of five points. From bottom to top in the top right-hand corner: degree 1 (straight line), degree 2 (parabola), degree 3, degree 4 (which is the lowest degree able to fit the points exactly), degree 5. **(right)** A piecewise constant function learned by a grouping model; the dotted reference line is the linear function from the left figure.

Overfitting again

An n-degree polynomial has $n+1$ parameters: e.g., a straight line $y = a \cdot x + b$ has two parameters, and the polynomial of degree 4 that fits the five points exactly has five parameters.

A piecewise constant model with n segments has $2n-1$ parameters: n y-values and $n-1$ x-values where the 'jumps' occur.

So the models that are able to fit the points exactly are the models with more parameters.

A rule of thumb is that, to avoid overfitting, the number of parameters estimated from the data must be considerably less than the number of data points.

What's next?

Table 3.1, p.95

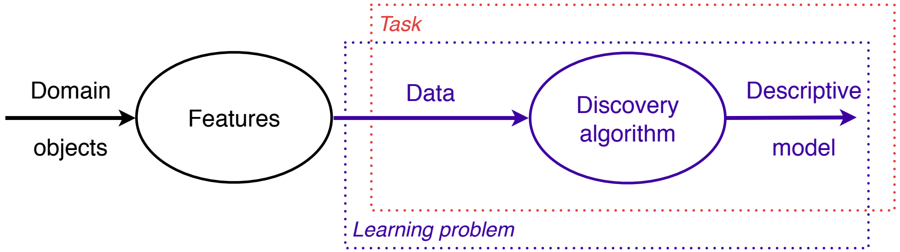
Unsupervised and descriptive learning

	Predictive model	*Descriptive model*
Supervised learning	classification, regression	**subgroup discovery**
Unsupervised learning	**predictive clustering**	**descriptive clustering**, **association rule discovery**

The learning settings indicated in **bold** are introduced in the remainder of this chapter.

Figure 3.4, p.96

Descriptive learning



In descriptive learning the task and learning problem coincide: we do not have a separate training set, and the task is to produce a descriptive model of the data.

Predictive and descriptive clustering

Predictive and descriptive clustering

One way to understand clustering is as learning a new labelling function from unlabelled data. So we could define a 'clusterer' in the same way as a classifier, namely as a mapping $\hat{q}:\mathscr{X}\rightarrow\mathscr{C}$, where $\mathscr{C}=\{C_1,C_2,\ldots,C_k\}$ is a set of new labels. This corresponds to a *predictive* view of clustering, as the domain of the mapping is the entire instance space, and hence it generalises to unseen instances.

A *descriptive* clustering model learned from given data $D\subseteq\mathscr{X}$ would be a mapping $\hat{q}:D\rightarrow\mathscr{C}$ whose domain is D rather than $\mathscr{X}$. In either case the labels have no intrinsic meaning, other than to express whether two instances belong to the same cluster. So an alternative way to define a clusterer is as an equivalence relation $\hat{q}\subseteq\mathscr{X}\times\mathscr{X}$ or $\hat{q}\subseteq D\times D$ or, equivalently, as a partition of $\mathscr{X}$ or D.

Distance-based clustering I

Most distance-based clustering methods depend on the possibility of defining a 'centre of mass' or *exemplar* for an arbitrary set of instances, such that the exemplar minimises some distance-related quantity over all instances in the set, called its *scatter*. A good clustering is then one where the scatter summed over each cluster – the *within-cluster scatter* – is much smaller than the scatter of the entire data set.

This analysis suggests a definition of the clustering problem as finding a partition $D = D_1 \uplus \ldots \uplus D_K$ that minimises the within-cluster scatter. However, there are a few issues with this definition:

- the problem as stated has a trivial solution: set $K = |D|$ so that each 'cluster' contains a single instance from D and thus has zero scatter;
- if we fix the number of clusters K in advance, the problem cannot be solved efficiently for large data sets (it is NP-hard).

Distance-based clustering II

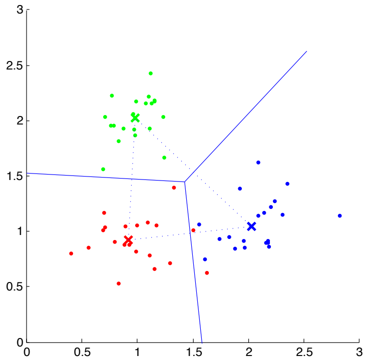
The first problem is the clustering equivalent of overfitting the training data. It could be dealt with by penalising large K. Most approaches, however, assume that an educated guess of K can be made. This leaves the second problem, which is that finding a globally optimal solution is intractable for larger problems. This is a well-known situation in computer science and can be dealt with in two ways:

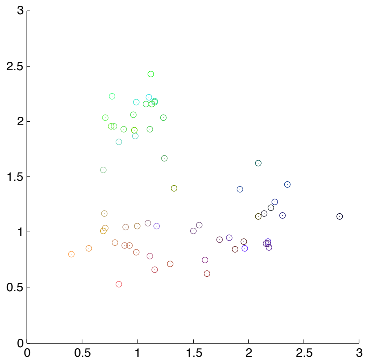
- by applying a heuristic approach, which finds a 'good enough' solution rather than the best possible one;
- by relaxing the problem into a 'soft' clustering problem, by allowing instances a degree of membership in more than one cluster.

Notice that a soft clustering generalises the notion of a partition, in the same way that a probability estimator generalises a classifier.

Figure 3.5, p.98

Predictive clustering





(left) An example of a predictive clustering. The coloured dots were sampled from three bivariate Gaussians centred at $(1,1)$, $(1,2)$ and $(2,1)$. The crosses and solid lines are the cluster exemplars and cluster boundaries found by 3-means. **(right)** A soft clustering of the same data found by matrix decomposition.

Example 3.9, p.98

Representing clusterings

The predictive cluster exemplars in Figure 3.5 (left) can be given as a c-by-2 matrix:

$$\begin{pmatrix} 0.92 & 0.93 \\ 0.98 & 2.02 \\ 2.03 & 1.04 \end{pmatrix}$$

The following n-by-c matrices represent descriptive clusterings of given data points:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ \cdots & \cdots & \cdots \end{pmatrix} \qquad \begin{pmatrix} 0.40 & 0.30 & 0.30 \\ 0.40 & 0.51 & 0.09 \\ 0.44 & 0.29 & 0.27 \\ 0.35 & 0.08 & 0.57 \\ \cdots & \cdots & \cdots \end{pmatrix}$$

Example 3.10, p.99

Evaluating clusterings

Suppose we have five test instances that we think should be clustered as $\{e1, e2\}, \{e3, e4, e5\}$. So out of the $5 \cdot 4 = 20$ possible pairs, 4 are considered 'must-link' pairs and the other 16 as 'must-not-link' pairs. The clustering to be evaluated clusters these as $\{e1, e2, e3\}, \{e4, e5\}$ – so two of the must-link pairs are indeed clustered together ($e1$–$e2$, $e4$–$e5$), the other two are not ($e3$–$e4$, $e3$–$e5$), and so on.

We can tabulate this as follows:

	Are together	*Are not together*	
Should be together	**2**	**2**	4
Should not be together	**2**	**14**	16
	4	16	20

We can now treat this as a two-by-two contingency table, and evaluate it accordingly. For instance, we can take the proportion of pairs on the 'good' diagonal, which is $16/20 = 0.8$. In classification we would call this accuracy, but in the clustering context this is known as the *Rand index*.

Other descriptive models

Example 3.11, p.100

Subgroup discovery

Imagine you want to market the new version of a successful product. You have a database of people who have been sent information about the previous version, containing all kinds of demographic, economic and social information about those people, as well as whether or not they purchased the product.

- ☞ If you were to build a classifier or ranker to find the most likely customers for your product, it is unlikely to outperform the majority class classifier (typically, relatively few people will have bought the product).
- ☞ However, what you are really interested in is finding reasonably sized subsets of people with a proportion of customers that is significantly higher than in the overall population. You can then target those people in your marketing campaign, ignoring the rest of your database.

Example 3.12, p.101

Association rule discovery

Associations are things that usually occur together. For example, in market basket analysis we are interested in items frequently bought together. An example of an association rule is ·**if** beer **then** crisps·, stating that customers who buy beer tend to also buy crisps.

☞ In a motorway service station most clients will buy petrol. This means that there will be many frequent item sets involving petrol, such as {newspaper, petrol}.

☞ This might suggest the construction of an association rule ·**if** newspaper **then** petrol· – however, this is predictable given that {petrol} is already a frequent item set (and clearly at least as frequent as {newspaper, petrol}).

☞ Of more interest would be the converse rule ·**if** petrol **then** newspaper· which expresses that a considerable proportion of the people buying petrol also buy a newspaper.

What's next?

4 Tree models

- Decision trees
- Ranking and probability estimation trees
  - Sensitivity to skewed class distributions
- Tree learning as variance reduction
  - Regression trees
  - Clustering trees

Tree model

- ☞ A tree model is a hierarchical structure of conditions, where leafs contain tree outcome.
- ☞ They represent recusive divide-and-conquer strategies.
- ☞ Tree models are among the most popular models in machine learning, because they are easy to understand and interpret:
- ☞ E.g., Kinect uses them to detect character pose.

Decision tree example

Suppose you come across a number of sea animals that you suspect belong to the same species. You observe their length in metres, whether they have gills, whether they have a prominent beak, and whether they have few or many teeth. Let the following be dolphins (positive class):

p1: $\text{Length} = 3 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$
p2: $\text{Length} = 4 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$
p3: $\text{Length} = 3 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{few}$
p4: $\text{Length} = 5 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$
p5: $\text{Length} = 5 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{few}$

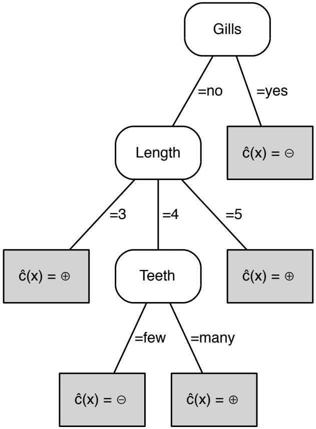
and the following be not dolphins (negative class):

n1: $\text{Length} = 5 \wedge \text{Gills} = \text{yes} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$
n2: $\text{Length} = 4 \wedge \text{Gills} = \text{yes} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$
n3: $\text{Length} = 5 \wedge \text{Gills} = \text{yes} \wedge \text{Beak} = \text{no} \wedge \text{Teeth} = \text{many}$
n4: $\text{Length} = 4 \wedge \text{Gills} = \text{yes} \wedge \text{Beak} = \text{no} \wedge \text{Teeth} = \text{many}$
n5: $\text{Length} = 4 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{few}$

Figure 5.1, p.130

Decision tree example

A decision tree learned on this data separates the positives and negatives perfectly.



Feature tree

Tree models are not limited to classification and can be employed to solve almost all machine learning tasks, including ranking, probability estimation, regression and clustering. A common structure to all those models is *feature tree*.

Definition 5.1, p.132

Feature tree

A *feature tree* is a tree such that each internal node (the nodes that are not leaves) is labelled with a feature, and each edge emanating from an internal node is labelled with a literal.

The set of literals at a node is called a *split*.

Each leaf of the tree represents a logical expression, which is the conjunction of literals encountered on the path from the root of the tree to the leaf. The extension of that conjunction (the set of instances covered by it) is called the *instance space segment* associated with the leaf.

Algorithm 5.1, p.132

Growing a feature tree

Algorithm GrowTree(D, F) – grow a feature tree from training data.

```
Input  : data $D$; set of features $F$.
Output : feature tree $T$ with labelled leaves.
1 if Homogeneous($D$) then return Label($D$);
2 $S \leftarrow$BestSplit($D, F$) ;          // e.g., BestSplit-Class (Algorithm 5.2)
3 split $D$ into subsets $D_i$ according to the literals in $S$;
4 for each $i$ do
5     if $D_i \neq \emptyset$ then $T_i \leftarrow$GrowTree($D_i, F$) ;
6     else $T_i$ is a leaf labelled with Label($D$);
7 end
8 return a tree whose root is labelled with $S$ and whose children are $T_i$
```

Growing a feature tree

Algorithm 5.1 gives the generic learning procedure common to most tree learners. It assumes that the following three functions are defined:

Homogeneous(D) returns true if the instances in D are homogeneous enough to be labelled with a single label, and false otherwise;

Label(D) returns the most appropriate label for a set of instances D;

BestSplit(D, F) returns the best set of literals to be put at the root of the tree.

These functions depend on the task at hand: for instance, for classification tasks a set of instances is homogeneous if they are (mostly) of a single class, and the most appropriate label would be the majority class. For clustering tasks a set of instances is homogenous if they are close together, and the most appropriate label would be some exemplar such as the mean.

What's next?

Decision tree

How to define BestSplit(D, F) for classification?

Assume that we have binary features and two classes only. Let $D^{\oplus}$ denote set of instances from positive class and $D^{\ominus}$ from negative class, $D = D^{\oplus} \cup D^{\ominus}$.

Let split D into D_1 and D_2 using an attribute. The best situation is where $D_1^{\oplus} = D^{\oplus}$ and $D_1^{\ominus} = \emptyset$ or $D_1^{\oplus} = \emptyset$ and $D_1^{\ominus} = D^{\ominus}$. In this cases the child node is said to be *pure*.

This, however, is unlikely in practice, thus we have to measure *impurity* of children nodes somehow.

Requirements for impurity measure

- Impurity should depend on relative magnitude of $n^{\oplus}$ and $n^{\ominus}$, thus can be defined in terms of probability of positive class: $\dot{p} = n^{\oplus}/(n^{\oplus} + n^{\ominus})$.
- Impurity should not change if we swap positive and negative class, i.e., if we replace $\dot{p}$ with $1 - \dot{p}$.
- Impurity should be 0 if $\dot{p} = 0$ or $\dot{p} = 1$, and reach maximum at $\dot{p} = 0.5$.

Impurity measures

- ☞ Minority class $min(\dot{p}, 1-\dot{p})$ – proportion of misclassified examples if labeling leaf using majority class
- ☞ Gini index $2\dot{p}(1-\dot{p})$ – expected error if labeling examples in leaf randomly with prorability $\dot{p}$for positive class and $1-\dot{p}$ for negative class
- ☞ Entropy $-\dot{p}\log_2\dot{p}-(1-\dot{p})\log_2(1-\dot{p})$ – expected amount of information in bits required to classify an example in leaf

Figure 5.2, p.134

Measuring impurity I

Indicating the impurity of a single leaf D_j as $\mathrm{Imp}(D_j)$, the impurity of a set of mutually exclusive leaves $\{D_1, \ldots, D_l\}$ is defined as a weighted average

$$\mathrm{Imp}(\{D_1, \ldots, D_l\}) = \sum_{j=1}^{l} \frac{|D_j|}{|D|} \mathrm{Imp}(D_j)$$

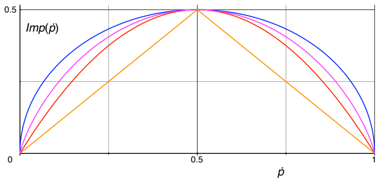
where $D = D_1 \cup \ldots \cup D_l$.

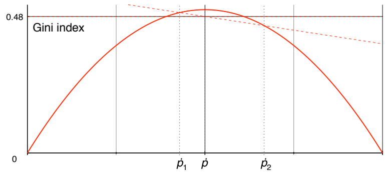
For a binary split there is a nice geometric construction to find $\mathrm{Imp}(\{D_1, D_2\})$:

- ☞ We first find the impurity values $\mathrm{Imp}(D_1)$ and $\mathrm{Imp}(D_2)$ of the two children on the impurity curve (here the Gini index).
- ☞ We then connect these two values by a straight line, as any weighted average of the two must be on that line.
- ☞ Since the empirical probability of the parent is also a weighted average of the empirical probabilities of the children, with the same weights (i.e., $\dot{p} = \frac{|D_1|}{|D|}\dot{p}_1 + \frac{|D_2|}{|D|}\dot{p}_2$), $\dot{p}$ gives us the correct interpolation point.

Figure 5.2, p.134

Measuring impurity II





(left) Impurity functions plotted against the empirical probability of the positive class. From the bottom: the relative size of the minority class, $\min(\dot{p}, 1-\dot{p})$; the Gini index, $2\dot{p}(1-\dot{p})$; entropy, $-\dot{p}\log_2\dot{p} - (1-\dot{p})\log_2(1-\dot{p})$ (divided by 2 so that it reaches its maximum in the same point as the others); and the (rescaled) square root of the Gini index, $\sqrt{\dot{p}(1-\dot{p})}$ – notice that this last function describes a semi-circle. **(right)** Geometric construction to determine the impurity of a split ($\text{Teeth} = [\text{many}, \text{few}]$): $\dot{p}$ is the empirical probability of the parent, and $\dot{p}_1$ and $\dot{p}_2$ are the empirical probabilities of the children.

Example 5.1, p.135

Calculating impurity I

Consider again the data in water animals. We want to find the best feature to put at the root of the decision tree. The four features available result in the following splits:

$\text{Length} = [3,4,5]$	$[2+,0-][1+,3-][2+,2-]$
$\text{Gills} = [\text{yes},\text{no}]$	$[0+,4-][5+,1-]$
$\text{Beak} = [\text{yes},\text{no}]$	$[5+,3-][0+,2-]$
$\text{Teeth} = [\text{many},\text{few}]$	$[3+,4-][2+,1-]$

Let's calculate the impurity of the first split. We have three segments: the first one is pure and so has entropy 0;
the second one has entropy
$-(1/4)\log_2(1/4) - (3/4)\log_2(3/4) = 0.5 + 0.31 = 0.81$;
the third one has entropy 1.
The total entropy is then the weighted average of these, which is
$2/10 \cdot 0 + 4/10 \cdot 0.81 + 4/10 \cdot 1 = 0.72$.

Example 5.1, p.135

Calculating impurity II

Similar calculations for the other three features give the following entropies:

$$\begin{array}{ll}
\text{Gills} & 4/10\cdot 0+6/10\cdot\big(-(5/6)\log_2(5/6)-(1/6)\log_2(1/6)\big)=0.39;\\
\text{Beak} & 8/10\cdot\big(-(5/8)\log_2(5/8)-(3/8)\log_2(3/8)\big)+2/10\cdot 0=0.76;\\
\text{Teeth} & 7/10\cdot\big(-(3/7)\log_2(3/7)-(4/7)\log_2(4/7)\big)
\end{array}$$

$$+3/10\cdot\big(-(2/3)\log_2(2/3)-(1/3)\log_2(1/3)\big)=0.97.$$

We thus clearly see that ‘Gills’ is an excellent feature to split on; ‘Teeth’ is poor; and the other two are somewhere in between.

The calculations for the Gini index are as follows (notice that these are on a scale from 0 to 0.5):

$$\begin{array}{ll}
\text{Length} & 2/10\cdot 2\cdot(2/2\cdot 0/2)+4/10\cdot 2\cdot(1/4\cdot 3/4)+4/10\cdot 2\cdot(2/4\cdot 2/4)=0.35;\\
\text{Gills} & 4/10\cdot 0+6/10\cdot 2\cdot(5/6\cdot 1/6)=0.17;\\
\text{Beak} & 8/10\cdot 2\cdot(5/8\cdot 3/8)+2/10\cdot 0=0.38;\\
\text{Teeth} & 7/10\cdot 2\cdot(3/7\cdot 4/7)+3/10\cdot 2\cdot(2/3\cdot 1/3)=0.48.
\end{array}$$

As expected, the two impurity measures are in close agreement. See Figure 5.2 (right) for a geometric illustration of the last calculation concerning ‘Teeth’.

Algorithm 5.2, p.137

Finding the best split for a decision tree

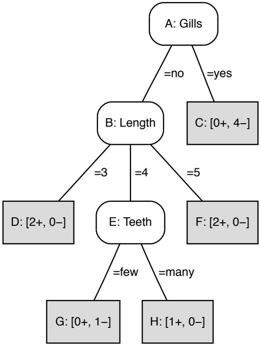
Algorithm BestSplit-Class(D, F) – find the best split for a decision tree.

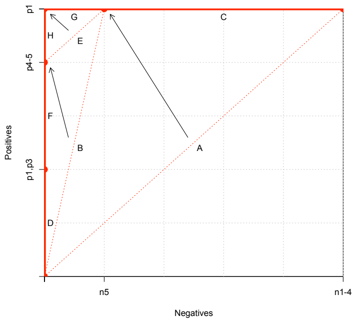
Input : data D; set of features F.
Output : feature f to split on.

```
1 $I_{\min} \leftarrow 1$;
2 for each $f \in F$ do
3     split $D$ into subsets $D_1, \ldots, D_l$ according to the values $v_j$ of $f$;
4     if $\mathrm{Imp}(\{D_1, \ldots, D_l\}) < I_{\min}$ then
5         $I_{\min} \leftarrow \mathrm{Imp}(\{D_1, \ldots, D_l\})$;
6         $f_{\text{best}} \leftarrow f$;
7     end
8 end
9 return $f_{\text{best}}$
```

Figure 5.3, p.137

Decision tree for dolphins





(left) Decision tree learned from the data on water animals. **(right)** Each internal and leaf node of the tree corresponds to a line segment in coverage space: vertical segments for pure positive nodes, horizontal segments for pure negative nodes, and diagonal segments for impure nodes.

What's next?

Important point to remember

Decision trees divide the instance space into segments, by learning ordering on those segments the decision trees can be turned into rankers.

Thanks to access to class distribution in each leaf the optimal orderdering for the training data can be obtained from empirical probabilities $\dot{p}$ (of positive class).

The ranking obtained from the empirical probabilities in the leaves of a decision tree yields a convex ROC curve on the training data.

Example 5.2, p.139

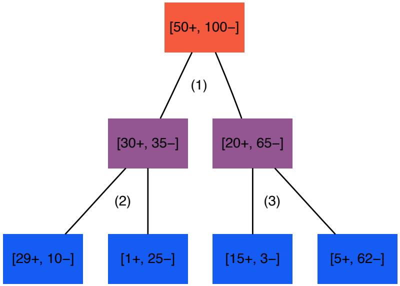
Growing a tree

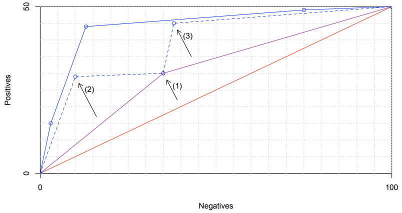
Consider the tree in Figure 5.4 (left). Each node is labelled with the numbers of positive and negative examples covered by it: so, for instance, the root of the tree is labelled with the overall class distribution (50 positives and 100 negatives), resulting in the trivial ranking $[50+, 100-]$. The corresponding one-segment coverage curve is the ascending diagonal (Figure 5.4 (right)).

- Adding split (1) refines this ranking into $[30+, 35-][20+, 65-]$, resulting in a two-segment curve.
- Adding splits (2) and (3) again breaks up the segment corresponding to the parent into two segments corresponding to the children.
- However, the ranking produced by the full tree – $[15+, 3-][29+, 10-][5+, 62-][1+, 25-]$ – is different from the left-to-right ordering of its leaves, hence we need to reorder the segments of the coverage curve, leading to the top-most, solid curve. This reordering always leads to a convex coverage curve

Figure 5.4, p.140

Growing a tree

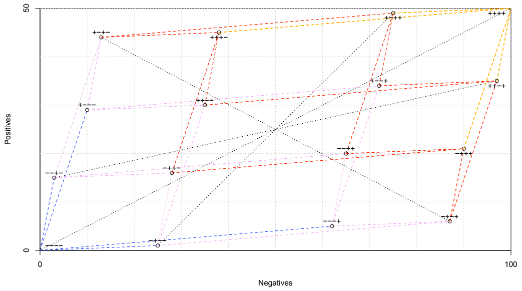




(left) Abstract representation of a tree with numbers of positive and negative examples covered in each node. Binary splits are added to the tree in the order indicated. **(right)** Adding a split to the tree will add new segments to the coverage curve as indicated by the arrows. After a split is added the segments may need reordering, and so only the solid lines represent actual coverage curves.

Figure 5.5, p.141

Labelling a tree



Graphical depiction of all possible labellings and all possible rankings that can be obtained with the four-leaf decision tree in Figure 5.4. There are $2^4 = 16$ possible leaf labellings; e.g., '$+-+-$' denotes labelling the first and third leaf from the left as $+$ and the second and fourth leaf as $-$. There are $4! = 24$ possible blue-violet-red-orange paths through these points which start in $----$ and switch each leaf to $+$ in some order; these represent all possible four-segment coverage curves or rankings.

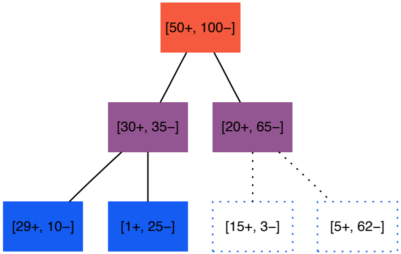
Choosing a labelling based on costs

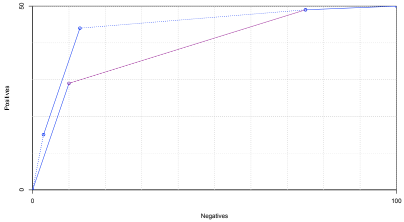
Assume the training set class ratio $clr = 50/100$ is representative. We have a choice of five labellings, depending on the expected cost ratio $c = c_{\mathrm{FN}}/c_{\mathrm{FP}}$ of misclassifying a positive in proportion to the cost of misclassifying a negative:

$+-+-$	would be the labelling of choice if $c = 1$, or more generally if $10/29 < c < 62/5$;
$+-++$	would be chosen if $62/5 < c < 25/1$;
$++++$	would be chosen if $25/1 < c$; i.e., we would always predict positive if false negatives are more than 25 times as costly as false positives, because then even predicting positive in the second leaf would reduce cost;
$--+-$	would be chosen if $3/15 < c < 10/29$;
$----$	would be chosen if $c < 3/15$; i.e., we would always predict negative if false positives are more than 5 times as costly as false negatives, because then even predicting negative in the third leaf would reduce cost.

Figure 5.6, p.143

Pruning a tree





(left) To achieve the labelling + – ++ we don't need the right-most split, which can therefore be pruned away. **(right)** Pruning doesn't affect the chosen operating point, but it does decrease the ranking performance of the tree.

Prunning a tree

- Prunning must not improve classification accuracy on training set
- However may improve generalization accuracy on test set
- A popular algorithm for pruning decision trees is *reduced-error pruning* that employs a separate *prunning set* of labelled data not seen during training.

Algorithm 5.3, p.144

Reduced-error pruning

Algorithm PruneTree(T, D) – reduced-error pruning of a decision tree.

```
Input  : decision tree $T$; labelled data $D$.
Output : pruned tree $T'$.
1 for every internal node $N$ of $T$, starting from the bottom do
2     $T_N \leftarrow$ subtree of $T$ rooted at $N$;
3     $D_N \leftarrow \{x \in D | x$ is covered by $N\}$;
4     if accuracy of $T_N$ over $D_N$ is worse than majority class in $D_N$ then
5         replace $T_N$ in $T$ by a leaf labelled with the majority class in $D_N$;
6     end
7 end
8 return pruned version of $T$
```

Sensitivity to skewed class distributions

Example 5.3, p.144

Skew sensitivity of splitting criteria I

Suppose you have 10 positives and 10 negatives, and you need to choose between the two splits $[8+,2-][2+,8-]$ and $[10+,6-][0+,4-]$.

- ☞ You duly calculate the weighted average entropy of both splits and conclude that the first split is the better one.
- ☞ Just to be sure, you also calculate the average Gini index, and again the first split wins.
- ☞ You then remember somebody telling you that the square root of the Gini index was a better impurity measure, so you decide to check that one out as well. Lo and behold, it favours the second split...! What to do?

Example 5.3, p.144

Skew sensitivity of splitting criteria II

You then remember that mistakes on the positives are about ten times as costly as mistakes on the negatives.

- ☞ You're not quite sure how to work out the maths, and so you decide to simply have ten copies of every positive: the splits are now $[80+,2-][20+,8-]$ and $[100+,6-][0+,4-]$.
- ☞ You recalculate the three splitting criteria and now all three favour the second split.
- ☞ Even though you're slightly bemused by all this, you settle for the second split since all three splitting criteria are now unanimous in their recommendation.

Why splitting point changed for Gini index and entropy?

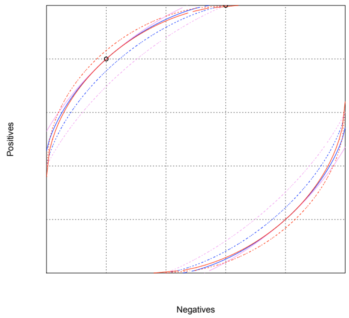
The Gini index of the parent is $2\frac{n^{\oplus}}{n}\frac{n^{\ominus}}{n}$, and the weighted Gini index of one of the children is $\frac{n_1}{n}2\frac{n_1^{\oplus}}{n_1}\frac{n_1^{\ominus}}{n_1}$. So the weighted impurity of the child *in proportion* to the parent's impurity is $\frac{n_1^{\oplus}n_1^{\ominus}/n_1}{n^{\oplus}n^{\ominus}/n}$; let's call this *relative impurity*.
The same calculations for $\sqrt{\text{Gini}}$ give

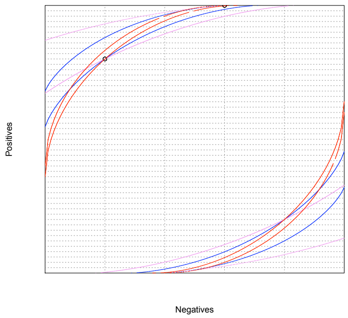
- ☞ impurity of the parent: $\sqrt{\frac{n^{\oplus}}{n}\frac{n^{\ominus}}{n}}$;
- ☞ weighted impurity of the child: $\frac{n_1}{n}\sqrt{\frac{n_1^{\oplus}}{n_1}\frac{n_1^{\ominus}}{n_1}}$;
- ☞ relative impurity: $\sqrt{\frac{n_1^{\oplus}n_1^{\ominus}}{n^{\oplus}n^{\ominus}}}$.

This last ratio doesn't change if we multiply all numbers involving positives with a factor c. That is, a splitting criterion using $\sqrt{\text{Gini}}$ as impurity measure is insensitive to changes in class distribution – unlike Gini index and entropy.

Figure 5.7, p.146

Skew sensitivity of splitting criteria





(left) ROC isometrics for entropy in blue, Gini index in violet and $\sqrt{\mathrm{Gini}}$ in red through the splits $[8+,2-][2+,8-]$ (solid lines) and $[10+,6-][0+,4-]$ (dotted lines). Only $\sqrt{\mathrm{Gini}}$ prefers the second split. **(right)** The same isometrics after inflating the positives with a factor 10. All splitting criteria now favour the second split; the $\sqrt{\mathrm{Gini}}$ isometrics are the only ones that haven't moved.

Important point to remember

Entropy and Gini index are sensitive to fluctuations in the class distribution, $\sqrt{\text{Gini}}$ isn't.

Peter's recipe for decision tree learning

- ☞ First and foremost, I would concentrate on getting good ranking behaviour, because from a good ranker I can get good classification and probability estimation, but not necessarily the other way round.
- ☞ I would therefore try to use an impurity measure that is distribution-insensitive, such as $\sqrt{\mathrm{Gini}}$; if that isn't available and I can't hack the code, I would resort to oversampling the minority class to achieve a balanced class distribution.
- ☞ I would disable pruning and smooth the probability estimates by means of the Laplace correction (or the m-estimate).
- ☞ Once I know the deployment operation conditions, I would use these to select the best operating point on the ROC curve (i.e., a threshold on the predicted probabilities, or a labelling of the tree).
- ☞ (optional) Finally, I would prune away any subtree whose leaves all have the same label.

What's next?

We will now consider how to adapt decision trees to regression and clustering tasks.

Regression trees

Tree learning as variance reduction

- ☞ The variance of a Boolean (i.e., Bernoulli) variable with success probability $\dot{p}$ is $\dot{p}(1-\dot{p})$, which is half the Gini index. So we could interpret the goal of tree learning as minimising the class variance (or standard deviation, in case of $\sqrt{\text{Gini}}$) in the leaves.
- ☞ In regression problems we can define the variance in the usual way:

$$\text{Var}(Y) = \frac{1}{|Y|}\sum_{y \in Y}(y-\overline{y})^2$$

If a split partitions the set of target values Y into mutually exclusive sets $\{Y_1,\ldots,Y_l\}$, the weighted average variance is then

$$\text{Var}(\{Y_1,\ldots,Y_l\}) = \sum_{j=1}^{l}\frac{|Y_j|}{|Y|}\text{Var}(Y_j) = \ldots = \frac{1}{|Y|}\sum_{y \in Y} y^2 - \sum_{j=1}^{l}\frac{|Y_j|}{|Y|}\overline{y}_j^2$$

The first term is constant for a given set Y and so we want to maximise the weighted average of squared means in the children.

Example 5.4, p.150

Learning a regression tree I

Imagine you are a collector of vintage Hammond tonewheel organs. You have been monitoring an online auction site, from which you collected some data about interesting transactions:

#	Model	Condition	Leslie	Price
1.	B3	excellent	no	4513
2.	T202	fair	yes	625
3.	A100	good	no	1051
4.	T202	good	no	270
5.	M102	good	yes	870
6.	A100	excellent	no	1770
7.	T202	fair	no	99
8.	A100	good	yes	1900
9.	E112	fair	no	77

Example 5.4, p.150

Learning a regression tree II

From this data, you want to construct a regression tree that will help you determine a reasonable price for your next purchase.
There are three features, hence three possible splits:

$$\mathrm{Model} = [\mathrm{A100}, \mathrm{B3}, \mathrm{E112}, \mathrm{M102}, \mathrm{T202}]$$
$$[1051, 1770, 1900][4513][77][870][99, 270, 625]$$
$$\mathrm{Condition} = [\mathrm{excellent}, \mathrm{good}, \mathrm{fair}]$$
$$[1770, 4513][270, 870, 1051, 1900][77, 99, 625]$$
$$\mathrm{Leslie} = [\mathrm{yes}, \mathrm{no}] \quad [625, 870, 1900][77, 99, 270, 1051, 1770, 4513]$$

The means of the first split are 1574, 4513, 77, 870 and 331, and the weighted average of squared means is $3.21 \cdot 10^6$.
The means of the second split are 3142, 1023 and 267, with weighted average of squared means $2.68 \cdot 10^6$;
for the third split the means are 1132 and 1297, with weighted average of squared means $1.55 \cdot 10^6$.
We therefore branch on Model at the top level. This gives us three single-instance leaves, as well as three A100s and three T202s.

Example 5.4, p.150

Learning a regression tree III

For the A100s we obtain the following splits:

$\text{Condition} = [\text{excellent}, \text{good}, \text{fair}]$	$[1770][1051, 1900][]$
$\text{Leslie} = [\text{yes}, \text{no}]$	$[1900][1051, 1770]$

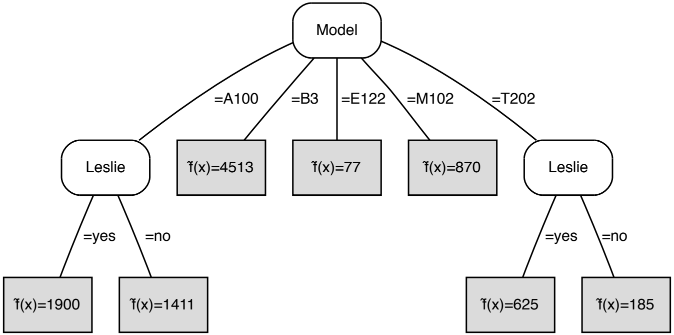
Without going through the calculations we can see that the second split results in less variance (to handle the empty child, it is customary to set its variance equal to that of the parent). For the T202s the splits are as follows:

$\text{Condition} = [\text{excellent}, \text{good}, \text{fair}]$	$[][270][99, 625]$
$\text{Leslie} = [\text{yes}, \text{no}]$	$[625][99, 270]$

Again we see that splitting on Leslie gives tighter clusters of values. The learned regression tree is depicted in Figure 5.8.

Figure 5.8, p.150

A regression tree



A regression tree learned from the data in Example 5.4.

Clustering trees

Dissimilarity measure

Let Dis: $\mathscr{X} \times \mathscr{X} \to \mathbb{R}$ be an abstract function that measures *dissimularity* of any two instances $x, x' \in \mathscr{X}$, such that the higher $\text{Dis}(x, x')$ is, the less similar x and x' are. The *cluster dissimilarity* of a set of instances D is:

$$\text{Dis(D)} = \frac{1}{|D|^2} \sum_{x \in D} \sum_{x' \in D} \text{Dis}(x, x')$$

Example 5.5, p.152 Learning a clustering tree I

Assessing the nine transactions on the online auction site from Example 5.4, using some additional features such as reserve price and number of bids, you come up with the following dissimilarity matrix:

$$\begin{array}{rrrrrrrrr}
0 & 11 & 6 & 13 & 10 & 3 & 13 & 3 & 12 \\
11 & \mathbf{0} & 1 & \mathbf{1} & 1 & 3 & \mathbf{0} & 4 & 0 \\
6 & 1 & \mathbf{0} & 2 & 1 & \mathbf{1} & 2 & \mathbf{2} & 1 \\
13 & \mathbf{1} & 2 & \mathbf{0} & 0 & 4 & \mathbf{0} & 4 & 0 \\
10 & 1 & 1 & 0 & 0 & 3 & 0 & 2 & 0 \\
3 & 3 & \mathbf{1} & 4 & 3 & \mathbf{0} & 4 & \mathbf{1} & 3 \\
13 & \mathbf{0} & 2 & \mathbf{0} & 0 & 4 & \mathbf{0} & 4 & 0 \\
3 & 4 & \mathbf{2} & 4 & 2 & \mathbf{1} & 4 & \mathbf{0} & 4 \\
12 & 0 & 1 & 0 & 0 & 3 & 0 & 4 & 0
\end{array}$$

This shows, for instance, that the first transaction is very different from the other eight. The average pairwise dissimilarity over all nine transactions is 2.94.

Example 5.5, p.152

Learning a clustering tree II

Using the same features from Example 5.4, the three possible splits are (now with transaction number rather than price):

$\text{Model} = [\text{A100}, \text{B3}, \text{E112}, \text{M102}, \text{T202}]$	$[3,6,8][1][9][5][2,4,7]$
$\text{Condition} = [\text{excellent}, \text{good}, \text{fair}]$	$[1,6][3,4,5,8][2,7,9]$
$\text{Leslie} = [\text{yes}, \text{no}]$	$[2,5,8][1,3,4,6,7,9]$

The cluster dissimilarity among transactions 3, 6 and 8 is $\frac{1}{3^2}(\mathbf{0}+\mathbf{1}+\mathbf{2}+\mathbf{1}+\mathbf{0}+\mathbf{1}+\mathbf{2}+\mathbf{1}+\mathbf{0}) = 0.89$; and among transactions 2, 4 and 7 it is $\frac{1}{3^2}(\mathbf{0}+\mathbf{1}+\mathbf{0}+\mathbf{1}+\mathbf{0}+\mathbf{0}+\mathbf{0}+\mathbf{0}+\mathbf{0}) = 0.22$. The other three children of the first split contain only a single element and so have zero cluster dissimilarity. The weighted average cluster dissimilarity of the split is then $3/9 \cdot 0.89 + 1/9 \cdot 0 + 1/9 \cdot 0 + 1/9 \cdot 0 + 3/9 \cdot 0.22 = 0.37$. For the second split, similar calculations result in a split dissimilarity of $2/9 \cdot 1.5 + 4/9 \cdot 1.19 + 3/9 \cdot 0 = 0.86$, and the third split yields $3/9 \cdot 1.56 + 6/9 \cdot 3.56 = 2.89$. The Model feature thus captures most of the given dissimilarities, while the Leslie feature is virtually unrelated.

Example 5.6, p.154

Clustering with Euclidean distance I

We extend our Hammond organ data with two new numerical features, one indicating the reserve price and the other the number of bids made in the auction.

Model	Condition	Leslie	Price	Reserve	Bids
B3	excellent	no	45	30	22
T202	fair	yes	6	0	9
A100	good	no	11	8	13
T202	good	no	3	0	1
M102	good	yes	9	5	2
A100	excellent	no	18	15	15
T202	fair	no	1	0	3
A100	good	yes	19	19	1
E112	fair	no	1	0	5

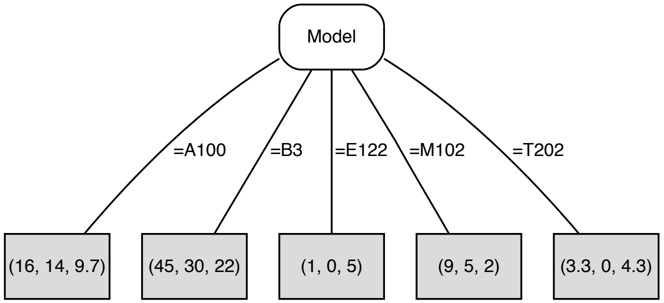
Example 5.6, p.154

Clustering with Euclidean distance II

- The means of the three numerical features are $(13.3, 8.6, 7.9)$ and their variances are $(158, 101.8, 48.8)$. The average squared Euclidean distance to the mean is then the sum of these variances, which is 308.6.
- For the A100 cluster these vectors are $(16, 14, 9.7)$ and $(12.7, 20.7, 38.2)$, with average squared distance to the mean 71.6; for the T202 cluster they are $(3.3, 0, 4.3)$ and $(4.2, 0, 11.6)$, with average squared distance 15.8.
- Using this split we can construct a clustering tree whose leaves are labelled with the mean vectors (Figure 5.9).

Figure 5.9, p.154

A clustering tree



A clustering tree learned from the data in Example 5.6 using Euclidean distance on the numerical features.

What's next?

5. Rule models
   - Learning ordered rule lists
     - Rule lists for ranking and probability estimation
   - Learning unordered rule sets
     - Rule sets for ranking and probability estimation
   - Descriptive rule learning
     - Rule learning for subgroup discovery
     - Association rule mining

What's next?

Example 6.1, p.159

Learning a rule list I

Consider again our small dolphins data set with positive examples

p1: $\text{Length} = 3 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$

p2: $\text{Length} = 4 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$

p3: $\text{Length} = 3 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{few}$

p4: $\text{Length} = 5 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$

p5: $\text{Length} = 5 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{few}$

and negatives

n1: $\text{Length} = 5 \wedge \text{Gills} = \text{yes} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$

n2: $\text{Length} = 4 \wedge \text{Gills} = \text{yes} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$

n3: $\text{Length} = 5 \wedge \text{Gills} = \text{yes} \wedge \text{Beak} = \text{no} \wedge \text{Teeth} = \text{many}$

n4: $\text{Length} = 4 \wedge \text{Gills} = \text{yes} \wedge \text{Beak} = \text{no} \wedge \text{Teeth} = \text{many}$

n5: $\text{Length} = 4 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{few}$

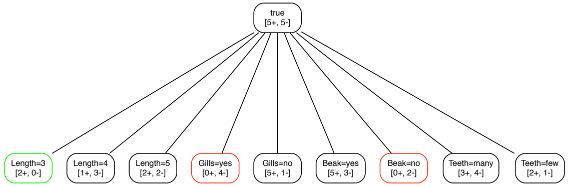
Example 6.1, p.159

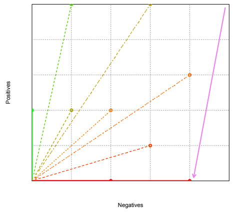
Learning a rule list II

- The nine possible literals are shown with their coverage counts in Figure 6.2 (left).
- Three of these are pure; in the impurity isometrics plot in Figure 6.2 (right) they end up on the x-axis and y-axis.
- One of the literals covers two positives and two negatives, and therefore has the same impurity as the overall data set; this literal ends up on the ascending diagonal in the coverage plot.

Figure 6.2, p.160

Searching for literals

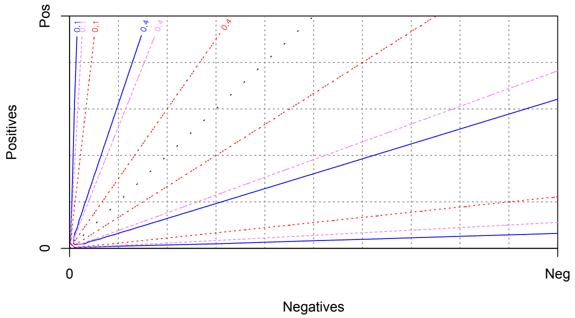




(left) All literals with their coverage counts on the data in Example 6.1. The ones in green (red) are pure for the positive (negative) class. **(right)** The nine literals plotted as points in coverage space, with their impurity values indicated by impurity isometrics (away from the ascending diagonal is better). Impurity values are colour-coded: towards green if $\dot{p} > 1/2$, towards red if $\dot{p} < 1/2$, and orange if $\dot{p} = 1/2$ (on a 45 degree isometric). The violet arrow indicates the selected literal, which excludes all five positives and one negative.

Figure 6.1, p.158

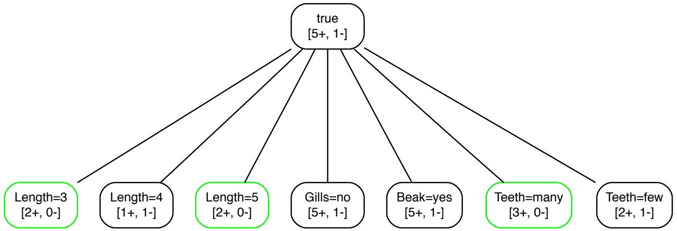
Equivalence of search heuristics

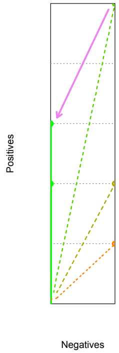


ROC isometrics for entropy (rescaled to have a maximum value of 1/2), Gini index and minority class. The grey dotted symmetry line is defined by $\dot{p} = 1/2$: each isometric has two parts, one above the symmetry line (where impurity decreases with increasing empirical probability $\dot{p}$) and its mirror image below the symmetry line (where impurity is proportional to $\dot{p}$).

Figure 6.3, p.161

Constructing the second rule

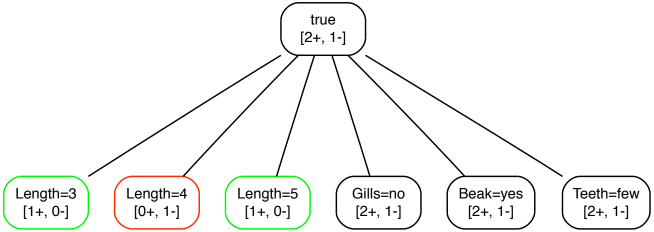


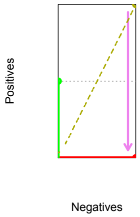


(left) Revised coverage counts after removing the four negative examples covered by the first rule found (literals not covering any examples are omitted). **(right)** We are now operating in the right-most 'slice' of Figure 6.2.

Figure 6.4, p.162

Constructing the third rule





(left) The third rule covers the one remaining negative example, so that the remaining positives can be swept up by a default rule. **(right)** This will collapse the coverage space.

Algorithm 6.1, p.163

Learning an ordered list of rules

Algorithm LearnRuleList(D) – learn an ordered list of rules.

Input : labelled training data D.
Output : rule list R.

```
1 $R \leftarrow \emptyset$;
2 while $D \neq \emptyset$ do
3     $r \leftarrow$LearnRule($D$) ;                // LearnRule: see Algorithm 6.2
4     append $r$ to the end of $R$;
5     $D \leftarrow D \setminus \{x \in D | x$ is covered by $r\}$;
6 end
7 return $R$
```

Algorithm 6.2, p.164

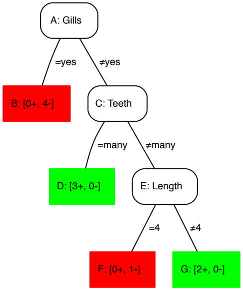
Learning a single rule

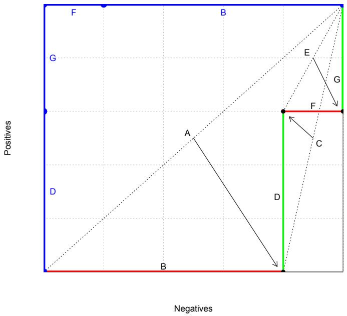
Algorithm LearnRule(D) – learn a single rule.

```
Input    : labelled training data $D$.
Output   : rule $r$.
1  $b \leftarrow$ true;
2  $L \leftarrow$ set of available literals;
3  while not Homogeneous($D$) do
4      $l \leftarrow$ BestLiteral($D, L$) ;                // e.g., highest purity; see text
5      $b \leftarrow b \wedge l$;
6      $D \leftarrow \{x \in D | x$ is covered by $b\}$;
7      $L \leftarrow L \setminus \{l' \in L | l'$ uses same feature as $l\}$;
8  end
9  $C \leftarrow$ Label($D$) ;                              // e.g., majority class
10 $r \leftarrow$ ·if $b$ then Class $= C$·;
11 return $r$
```

Figure 6.5, p.164

Rule list as a tree





(left) A right-branching feature tree corresponding to a list of single-literal rules. **(right)** The construction of this feature tree depicted in coverage space. The leaves of the tree are either purely positive (in green) or purely negative (in red). Reordering these leaves on their empirical probability results in the blue coverage curve. As the rule list separates the classes this is a perfect coverage curve.

Rule lists for ranking and probability estimation

Important point to remember

Rule lists inherit the property of decision trees that their training set coverage curve is always convex.

Example 6.2, p.165

Rule lists as rankers I

Consider the following two concepts:

(A)	Length = 4	p2	n2, n4–5
(B)	Beak = yes	p1–5	n1–2, n5

Indicated on the right is each concept's coverage over the whole training set. Using these concepts as rule bodies, we can construct the rule list AB:

·**if** Length = 4 **then** Class = ⊖·	[1+, 3−]
·**else if** Beak = yes **then** Class = ⊕·	[4+, 1−]
·**else** Class = ⊖·	[0+, 1−]

The coverage curve of this rule list is given in Figure 6.6.

Example 6.2, p.165

Rule lists as rankers II

- The first segment of the curve corresponds to all instances which are covered by B but not by A, which is why we use the set-theoretical notation B \ A.
- Notice that while this segment corresponds to the second rule in the rule list, it comes first in the coverage curve because it has the highest proportion of positives.
- The second coverage segment corresponds to rule A, and the third coverage segment denoted '-' corresponds to the default rule.
- This segment comes last, not because it represents the last rule, but because it happens to cover no positives.

Example 6.2, p.165

Rule lists as rankers III

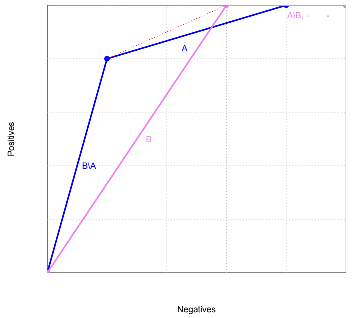
We can also construct a rule list in the opposite order, BA:

·**if** Beak = yes **then** Class = ⊕·	$[5+, 3-]$
·**else if** Length = 4 **then** Class = ⊖·	$[0+, 1-]$
·**else** Class = ⊖·	$[0+, 1-]$

The coverage curve of this rule list is also depicted in Figure 6.6. This time, the first segment corresponds to the first segment in the rule list (B), and the second and third segment are tied between rule A (after the instances covered by B are taken away: A \ B) and the default rule.

Figure 6.6, p.166

Rule lists as rankers



Coverage curves of two rule lists consisting of the rules from Example 6.2, in different order (AB in blue and BA in violet). B \ A corresponds to the coverage of rule B once the coverage of rule A is taken away, and '-' denotes the default rule. The dotted segment in red connecting the two curves corresponds to the overlap of the two rules A ∧ B, which is not accessible by either rule list.

Important point to remember

Rule lists are similar to decision trees in that the empirical probabilities associated with each rule yield convex ROC and coverage curves on the training data.

What's next?

Example 6.3, p.167

Learning a rule set for class ⊕

Figure 6.7 shows that the first rule learned for the positive class is

$$\cdot\textbf{if } \text{Length} = 3 \textbf{ then } \text{Class} = \oplus\cdot$$

The two examples covered by this rule are removed, and a new rule is learned. We now encounter a new situation, as none of the candidates is pure (Figure 6.8). We thus start a second-level search, from which the following pure rule emerges:

$$\cdot\textbf{if } \text{Gills} = \text{no} \wedge \text{Length} = 5 \textbf{ then } \text{Class} = \oplus\cdot$$

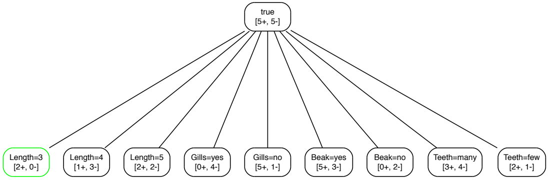
To cover the remaining positive, we again need a rule with two conditions (Figure 6.9):

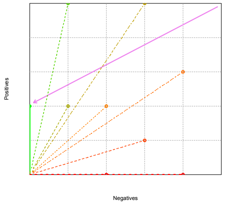
$$\cdot\textbf{if } \text{Gills} = \text{no} \wedge \text{Teeth} = \text{many} \textbf{ then } \text{Class} = \oplus\cdot$$

Notice that, even though these rules are overlapping, their overlap only covers positive examples (since each of them is pure) and so there is no need to organise them in an if-then-else list.

Figure 6.7, p.168

Learning a rule set

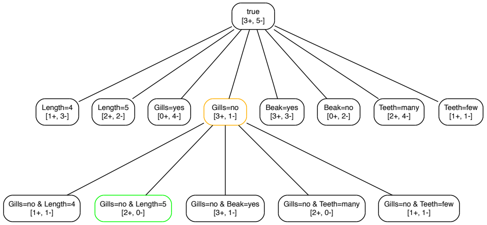


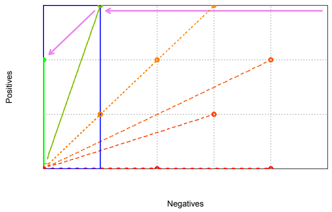


(left) The first rule is learned for the positive class. **(right)** Precision isometrics look identical to impurity isometrics (Figure 6.2); however, the difference is that precision is lowest on the x-axis and highest on the y-axis, while purity is lowest on the ascending diagonal and highest on both the x-axis and the y-axis.

Figure 6.8, p.169

Learning the second rule



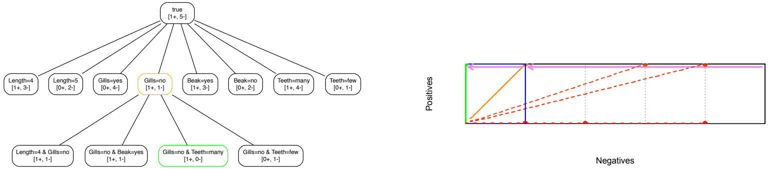


(left) The second rule needs two literals: we use maximum precision to select both.

(right) The coverage space is smaller because the two positives covered by the first rule are removed. The blue box on the left indicates an even smaller coverage space in which the search for the second literal is carried out, after the condition Gills = no filters out four negatives. Inside the blue box precision isometrics overlap with those in the outer box (this is not necessarily the case with search heuristics other than precision).

Figure 6.9, p.170

Learning the third rule



(left) The third and final rule again needs two literals. **(right)** The first literal excludes four negatives, the second excludes the one remaining negative.

Algorithm 6.3, p.171

Learning an unordered set of rules

Algorithm LearnRuleSet(D) – learn an unordered set of rules.

```
Input  : labelled training data $D$.
Output : rule set $R$.
1  $R \leftarrow \emptyset$;
2  for every class $C_i$ do
3      $D_i \leftarrow D$;
4      while $D_i$ contains examples of class $C_i$ do
5          $r \leftarrow$ LearnRuleForClass($D_i, C_i$) ;  // LearnRuleForClass: see Algorithm 6.4
6          $R \leftarrow R \cup \{r\}$;
7          $D_i \leftarrow D_i \setminus \{x \in C_i | x$ is covered by $r\}$ ;    // remove only positives
8      end
9  end
10 return $R$
```

Algorithm 6.4, p.171

Learning a single rule for a given class

Algorithm LearnRuleForClass(D, C_i) – learn a single rule for a given class.

```
Input  : labelled training data $D$; class $C_i$.
Output : rule $r$.
$b \leftarrow$ true;
$L \leftarrow$ set of available literals ;          // can be initialised by seed example
while not Homogeneous($D$) do
    $l \leftarrow$ BestLiteral($D, L, C_i$) ;        // e.g. maximising precision on class $C_i$
    $b \leftarrow b \wedge l$;
    $D \leftarrow \{x \in D | x$ is covered by $b\}$;
    $L \leftarrow L \setminus \{l' \in L | l'$ uses same feature as $l\}$;
end
$r \leftarrow \cdot$if $b$ then Class $= C_i\cdot$;
return $r$
```

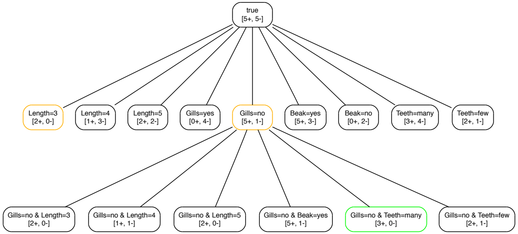
The need for probability smoothing

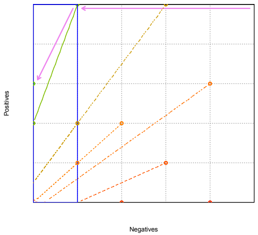
One issue with using precision as search heuristic is that it tends to focus a bit too much on finding pure rules, thereby occasionally missing near-pure rules that can be specialised into a more general pure rule.

- ☞ Consider Figure 6.10 (left): precision favours the rule ·**if** Length = 3 **then** Class = ⊕·, even though the near-pure literal Gills = no leads to the pure rule ·**if** Gills = no ∧ Teeth = many **then** Class = ⊕·.
- ☞ A convenient way to deal with this ‘myopia’ of precision is the Laplace correction, which ensures that $[5+, 1-]$ is ‘corrected’ to $[6+, 2-]$ and thus considered to be of the same quality as $[2+, 0-]$ aka $[3+, 1-]$ (Figure 6.10 (right)).

Figure 6.10, p.172

Using the Laplace correction





(left) Using Laplace-corrected precision allows learning a better rule in the first iteration.
(right) Laplace correction adds one positive and one negative pseudo-count, which means that the isometrics now rotate around $(-1, -1)$ in coverage space, resulting in a preference for more general rules.

Rule sets for ranking and probability estimation

Example 6.4, p.173

Rule sets as rankers I

Consider the following rule set (the first two rules were also used in Example 6.2):

(A)	·**if** Length = 4 **then** Class = ⊖·	$[1+,3-]$
(B)	·**if** Beak = yes **then** Class = ⊕·	$[5+,3-]$
(C)	·**if** Length = 5 **then** Class = ⊖·	$[2+,2-]$

- The figures on the right indicate coverage of each rule over the whole training set. For instances covered by single rules we can use these coverage counts to calculate probability estimates: e.g., an instance covered only by rule A would receive probability $\hat{p}(A) = 1/4 = 0.25$, and similarly $\hat{p}(B) = 5/8 = 0.63$ and $\hat{p}(C) = 2/4 = 0.50$.
- Clearly A and C are mutually exclusive, so the only overlaps we need to take into account are AB and BC.

Example 6.4, p.173

Rule sets as rankers II

- A simple trick that is often applied is to average the coverage of the rules involved: for example, the coverage of AB is estimated as $[3+,3-]$ yielding $\hat{p}(\text{AB}) = 3/6 = 0.50$. Similarly, $\hat{p}(\text{BC}) = 3.5/6 = 0.58$.
- The corresponding ranking is thus B – BC – [AB, C] – A, resulting in the orange training set coverage curve in Figure 6.11.

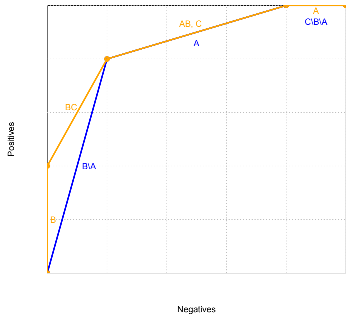
Let us now compare this rule set with the following rule list ABC:

·**if** Length = 4 **then** Class = ⊖·	$[1+,3-]$
·**else if** Beak = yes **then** Class = ⊕·	$[4+,1-]$
·**else if** Length = 5 **then** Class = ⊖·	$[0+,1-]$

The coverage curve of this rule list is indicated in Figure 6.11 as the blue line. We see that the rule set outperforms the rule list, by virtue of being able to distinguish between examples covered by B only and those covered by both B and C.

Figure 6.11, p.174

Rule set vs rule list



Coverage curves of the rule set in Example 6.4 (in orange) and the rule list ABC (in blue). The rule set partitions the instance space in smaller segments, which in this case lead to better ranking performance.

What's next?

Rule learning for subgroup discovery

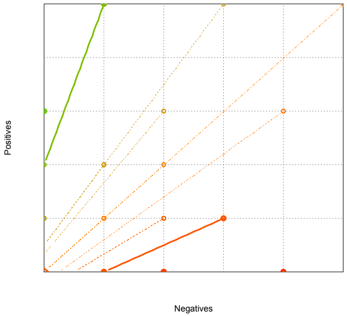
Subgroup discovery

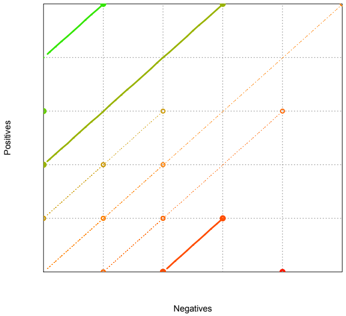
Subgroups are subsets of the instance space – or alternatively, mappings $\hat{g}:\mathscr{X}\rightarrow\{\text{true},\text{false}\}$ – that are learned from a set of labelled examples $(x_i, l(x_i))$, where $l:\mathscr{X}\rightarrow\mathscr{C}$ is the true labelling function.

- ☞ A good subgroup is one whose class distribution is significantly different from the overall population. This is by definition true for pure subgroups, but these are not the only interesting ones.
- ☞ For instance, one could argue that the complement of a subgroup is as interesting as the subgroup itself: in our dolphin example, the concept $\text{Gills} = \text{yes}$, which covers four negatives and no positives, could be considered as interesting as its complement $\text{Gills} = \text{no}$, which covers one negative and all positives.
- ☞ This means that we need to move away from impurity-based evaluation measures.

Figure 6.15, p.179

Evaluating subgroups





(left) Subgroups and their isometrics according to Laplace-corrected precision. The solid, outermost isometrics indicate the best subgroups. **(right)** The ranking changes if we order the subgroups on average recall. For example, $[5+,1-]$ is now better than $[3+,0-]$ and as good as $[0+,4-]$.

Example 6.6, p.180

Evaluating subgroups

Table 6.1 ranks ten subgroups in the dolphin example in terms of Laplace-corrected precision and average recall.

☞ One difference is that $\text{Gills} = \text{no} \ \wedge \text{Teeth} = \text{many}$ with coverage $[3+, 0-]$ is better than $\text{Gills} = \text{no}$ with coverage $[5+, 1-]$ in terms of Laplace-corrected precision, but worse in terms of average recall, as the latter ranks it equally with its complement $\text{Gills} = \text{yes}$.

Table 6.1, p.179

Evaluating subgroups

Subgroup	*Coverage*	$prec^{\mathrm{L}}$	*Rank*	$avg\text{-}rec$	*Rank*
Gills = yes	$[0+,4-]$	0.17	1	0.10	1–2
Gills = no ∧ Teeth = many	$[3+,0-]$	0.80	2	0.80	3
Gills = no	$[5+,1-]$	0.75	3–9	0.90	1–2
Beak = no	$[0+,2-]$	0.25	3–9	0.30	4–11
Gills = yes ∧ Beak = yes	$[0+,2-]$	0.25	3–9	0.30	4–11
Length = 3	$[2+,0-]$	0.75	3–9	0.70	4–11
Length = 4 ∧ Gills = yes	$[0+,2-]$	0.25	3–9	0.30	4–11
Length = 5 ∧ Gills = no	$[2+,0-]$	0.75	3–9	0.70	4–11
Length = 5 ∧ Gills = yes	$[0+,2-]$	0.25	3–9	0.30	4–11
Length = 4	$[1+,3-]$	0.33	10	0.30	4–11
Beak = yes	$[5+,3-]$	0.60	11	0.70	4–11

Using Laplace-corrected precision we can evaluate the quality of a subgroup as $|prec^{\mathrm{L}} - pos|$. Alternatively, we can use average recall to define the quality of a subgroup as $|avg\text{-}rec - 0.5|$.

Association rule mining

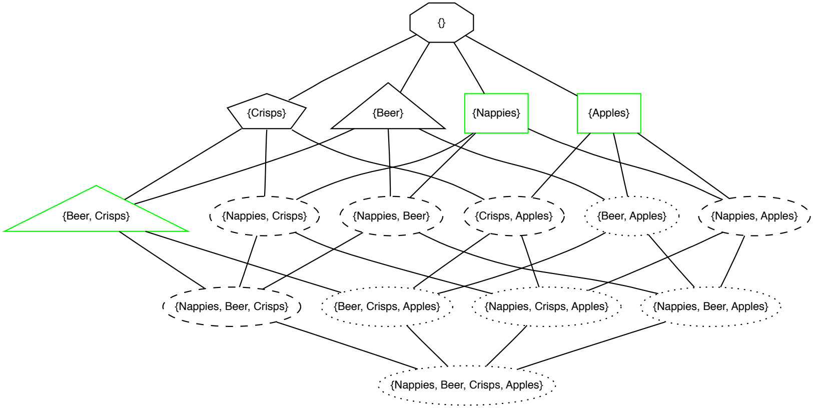
Items and transactions

Transaction	*Items*
1	nappies
2	beer, crisps
3	apples, nappies
4	beer, crisps, nappies
5	apples
6	apples, beer, crisps, nappies
7	apples, crisps
8	crisps

Each *transaction* in this table involves a set of *items*; conversely, for each item we can list the transactions in which it was involved: transactions 1, 3, 4 and 6 for nappies, transactions 3, 5, 6 and 7 for apples, and so on. We can also do this for sets of items: e.g., beer and crisps were bought together in transactions 2, 4 and 6; we say that item set $\{\text{beer}, \text{crisps}\}$ *covers* transaction set $\{2, 4, 6\}$.

Figure 6.17, p.183

An item set lattice



Item sets in dotted ovals cover a single transaction; in dashed ovals, two transactions; in triangles, three transactions; and in polygons with n sides, n transactions. The maximal item sets with support 3 or more are indicated in green.

Algorithm 6.6, p.184

Maximal item sets

Algorithm FrequentItems(D, f_0) – find all maximal item sets exceeding a given support threshold.

```
Input  : data $D \subseteq \mathscr{X}$; support threshold $f_0$.
Output : set of maximal frequent item sets $M$.
$M \leftarrow \emptyset$;
initialise priority queue $Q$ to contain the empty item set;
while $Q$ is not empty do
    $I \leftarrow$ next item set deleted from front of $Q$;
    $max \leftarrow$ true ;                 // flag to indicate whether $I$ is maximal
    for each possible extension $I'$ of $I$ do
        if Supp$(I') \geq f_0$ then
            $max \leftarrow$ false ;        // frequent extension found, so $I$ is not maximal
            add $I'$ to back of $Q$;
        end
    end
    if $max =$ true then $M \leftarrow M \cup \{I\}$;
end
return $M$
```

Association rules I

Frequent item sets can be used to build *association rules*, which are rules of the form ·**if** B **then** H· where both body B and head H are item sets that frequently appear in transactions together.

- ☞ Pick any edge in Figure 6.17, say the edge between {beer} and {nappies, beer}. We know that the support of the former is 3 and of the latter, 2: that is, three transactions involve beer and two of those involve nappies as well. We say that the *confidence* of the association rule ·**if** beer **then** nappies· is 2/3.
- ☞ Likewise, the edge between {nappies} and {nappies, beer} demonstrates that the confidence of the rule ·**if** nappies **then** beer· is 2/4.
- ☞ There are also rules with confidence 1, such as ·**if** beer **then** crisps·; and rules with empty bodies, such as ·**if** true **then** crisps·, which has confidence 5/8 (i.e., five out of eight transactions involve crisps).

Association rules II

But we only want to construct association rules that involve frequent items.

- ☞ The rule ·**if** beer ∧ apples **then** crisps· has confidence 1, but there is only one transaction involving all three and so this rule is not strongly supported by the data.
- ☞ So we first use Algorithm 6.6 to mine for frequent item sets; we then select bodies B and heads H from each frequent set m, discarding rules whose confidence is below a given confidence threshold.
- ☞ Notice that we are free to discard some of the items in the maximal frequent sets (i.e., $H \cup B$ may be a proper subset of m), because any subset of a frequent item set is frequent as well.

Algorithm 6.7, p.185

Association rule mining

Algorithm AssociationRules(D, f_0, c_0) – find all association rules exceeding given support and confidence thresholds.

```
Input  : data $D \subseteq \mathscr{X}$; support threshold $f_0$; confidence threshold $c_0$.
Output : set of association rules $R$.
1 $R \leftarrow \emptyset$;
2 $M \leftarrow$ FrequentItems$(D, f_0)$ ;                // FrequentItems: see Algorithm 6.6
3 for each $m \in M$ do
4     for each $H \subseteq m$ and $B \subseteq m$ such that $H \cap B = \emptyset$ do
5         if $\text{Supp}(B \cup H)/\text{Supp}(B) \geq c_0$ then $R \leftarrow R \cup \{\cdot$if $B$ then $H\cdot\}$;
6     end
7 end
8 return $R$
```

Association rule example

A run of the algorithm with support threshold 3 and confidence threshold 0.6 gives the following association rules:

·**if** beer **then** crisps·	support 3, confidence 3/3
·**if** crisps **then** beer·	support 3, confidence 3/5
·**if** true **then** crisps·	support 5, confidence 5/8

Association rule mining often includes a *post-processing* stage in which superfluous rules are filtered out, e.g., special cases which don't have higher confidence than the general case.

Post-processing

One quantity that is often used in post-processing is *lift*, defined as

$$\mathrm{Lift}(\cdot\textbf{if } B \textbf{ then } H\cdot) = \frac{n \cdot \mathrm{Supp}(B \cup H)}{\mathrm{Supp}(B) \cdot \mathrm{Supp}(H)}$$

where n is the number of transactions.

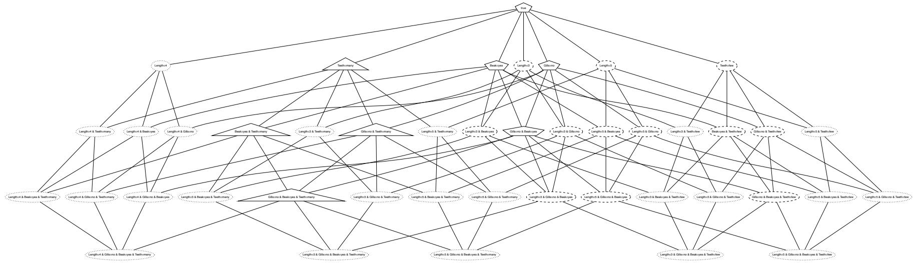
- ☞ For example, for the the first two association rules above we would have lifts of $\frac{8\cdot 3}{3\cdot 5} = 1.6$, as $\mathrm{Lift}(\cdot\textbf{if } B \textbf{ then } H\cdot) = \mathrm{Lift}(\cdot\textbf{if } H \textbf{ then } B\cdot)$.
- ☞ For the third rule we have $\mathrm{Lift}(\cdot\textbf{if } \mathrm{true} \textbf{ then } \mathrm{crisps}\cdot) = \frac{8\cdot 5}{8\cdot 5} = 1$. This holds for any rule with $B = \emptyset$, as

$$\mathrm{Lift}(\cdot\textbf{if } \emptyset \textbf{ then } H\cdot) = \frac{n \cdot \mathrm{Supp}(\emptyset \cup H)}{\mathrm{Supp}(\emptyset) \cdot \mathrm{Supp}(H)} = \frac{n \cdot \mathrm{Supp}(H)}{n \cdot \mathrm{Supp}(H)} = 1$$

More generally, a lift of 1 means that $\mathrm{Supp}(B \cup H)$ is entirely determined by the *marginal* frequencies $\mathrm{Supp}(B)$ and $\mathrm{Supp}(H)$ and is not the result of any meaningful interaction between B and H. Only association rules with lift larger than 1 are of interest.

Figure 6.19, p.187

Item sets and dolphins


true
Length=4
Teeth=many
Beak=yes
Length=3
Gills=no
Length=5
Teeth=few
Length=4 & Teeth=many
Length=4 & Beak=yes
Length=4 & Gills=no
Beak=yes & Teeth=many
Length=3 & Teeth=many
Gills=no & Teeth=many
Length=5 & Teeth=many
Length=3 & Beak=yes
Gills=no & Beak=yes
Length=3 & Gills=no
Length=5 & Beak=yes
Length=5 & Gills=no
Length=3 & Teeth=few
Beak=yes & Teeth=few
Gills=no & Teeth=few
Length=5 & Teeth=few
Length=4 & Beak=yes & Teeth=many
Length=4 & Gills=no & Teeth=many
Length=4 & Gills=no & Beak=yes
Length=3 & Beak=yes & Teeth=many
Gills=no & Beak=yes & Teeth=many
Length=3 & Gills=no & Teeth=many
Length=5 & Beak=yes & Teeth=many
Length=5 & Gills=no & Teeth=many
Length=3 & Gills=no & Beak=yes
Length=5 & Gills=no & Beak=yes
Length=3 & Beak=yes & Teeth=few
Length=3 & Gills=no & Teeth=few
Gills=no & Beak=yes & Teeth=few
Length=5 & Beak=yes & Teeth=few
Length=5 & Gills=no & Teeth=few
Length=4 & Gills=no & Beak=yes & Teeth=many
Length=3 & Gills=no & Beak=yes & Teeth=many
Length=5 & Gills=no & Beak=yes & Teeth=many
Length=3 & Gills=no & Beak=yes & Teeth=few
Length=5 & Gills=no & Beak=yes & Teeth=few

The item set lattice corresponding to the positive examples of the dolphin example in Example 4.4. Each 'item' is a literal Feature = Value; each feature can occur at most once in an item set. The resulting structure is exactly the same as what was called the hypothesis space in Chapter 4.

What's next?

What's next?

Example 7.1, p.197

Univariate linear regression

Suppose we want to investigate the relationship between people's height and weight. We collect n height and weight measurements $(h_i, w_i), 1 \leq i \leq n$.

Univariate linear regression assumes a linear equation $w = a + bh$, with parameters a and b chosen such that the sum of squared residuals $\sum_{i=1}^{n}(w_i - (a + bh_i))^2$ is minimised.

In order to find the parameters we take partial derivatives of this expression, set the partial derivatives to 0 and solve for a and b:

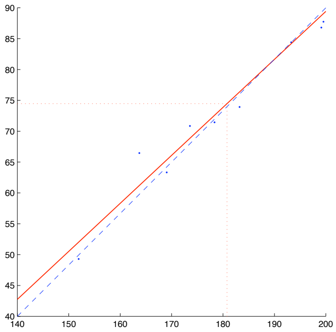
$$\frac{\partial}{\partial a}\sum_{i=1}^{n}(w_i - (a+bh_i))^2 = -2\sum_{i=1}^{n}(w_i - (a+bh_i)) = 0 \qquad \Rightarrow \hat{a} = \overline{w} - \hat{b}\overline{h}$$

$$\frac{\partial}{\partial b}\sum_{i=1}^{n}(w_i - (a+bh_i))^2 = -2\sum_{i=1}^{n}(w_i - (a+bh_i))h_i = 0 \qquad \Rightarrow \hat{b} = \frac{\sum_{i=1}^{n}(h_i - \overline{h})(w_i - \overline{w})}{\sum_{i=1}^{n}(h_i - \overline{h})^2}$$

So the solution found by linear regression is $w = \hat{a} + \hat{b}h = \overline{w} + \hat{b}(h - \overline{h})$; see Figure 7.1 for an example.

Figure 7.1, p.197

Univariate linear regression



The red solid line indicates the result of applying linear regression to 10 measurements of body weight (on the y-axis, in kilograms) against body height (on the x-axis, in centimetres). The orange dotted lines indicate the average height $\overline{h} = 181$ and the average weight $\overline{w} = 74.5$; the regression coefficient $\hat{b} = 0.78$. The measurements were simulated by adding normally distributed noise with mean 0 and variance 5 to the true model indicated by the blue dashed line ($b = 0.83$).

Linear regression: intuitions I

For a feature x and a target variable y, the regression coefficient is the covariance between x and y in proportion to the variance of x:

$$\hat{b} = \frac{\sigma_{xy}}{\sigma_{xx}}$$

(Here I use σ_{xx} as an alternative notation for σ_x^2).

This can be understood by noting that the covariance is measured in units of x times units of y (e.g., metres times kilograms in Example 7.1) and the variance in units of x squared (e.g., metres squared), so their quotient is measured in units of y per unit of x (e.g., kilograms per metre).

Linear regression: intuitions II

The intercept $\hat{a}$ is such that the regression line goes through $(\overline{x}, \overline{y})$.

Adding a constant to all x-values (a translation) will affect only the intercept but not the regression coefficient (since it is defined in terms of deviations from the mean, which are unaffected by a translation).

So we could *zero-centre* the x-values by subtracting $\overline{x}$, in which case the intercept is equal to $\overline{y}$.

We could even subtract $\overline{y}$ from all y-values to achieve a zero intercept, without changing the problem in an essential way.

Linear regression: intuitions III

Suppose we replace x_i with $x'_i = x_i/\sigma_{xx}$ and likewise $\overline{x}$ with $\overline{x'} = \overline{x}/\sigma_{xx}$, then we have that $\hat{b} = \frac{1}{n}\sum_{i=1}^{n}(x'_i - \overline{x'})(y_i - \overline{y}) = \sigma_{x'y}$.

In other words, if we *normalise* x by dividing all its values by x's variance, we can take the covariance between the normalised feature and the target variable as regression coefficient.

This demonstrates that univariate linear regression can be understood as consisting of two steps:

- ☞ normalisation of the feature by dividing its values by the feature's variance;
- ☞ calculating the covariance of the target variable and the normalised feature.

We will see below how these two steps change when dealing with more than one feature.

Linear regression: intuitions IV

Another important point to note is that the sum of the residuals of the least-squares solution is zero:

$$\sum_{i=1}^{n}(y_i-(\hat{a}+\hat{b}x_i))=n(\overline{y}-\hat{a}-\hat{b}\overline{x})=0$$

The result follows because $\hat{a}=\overline{y}-\hat{b}\overline{x}$, as derived in Example 7.1.

While this property is intuitively appealing, it is worth keeping in mind that it also makes linear regression susceptible to *outliers*: points that are far removed from the regression line, often because of measurement errors.

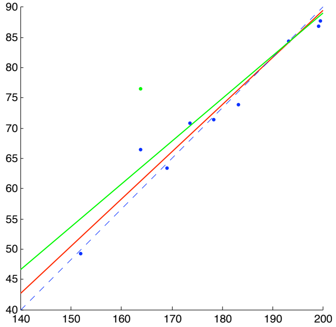
Example 7.2, p.199

The effect of outliers

Suppose that, as the result of a transcription error, one of the weight values in Figure 7.1 is increased by 10 kg. Figure 7.2 shows that this has a considerable effect on the least-squares regression line.

Figure 7.2, p.199

The effect of outliers



One of the blue points got moved up 10 units to the green point, changing the red regression line to the green line.

Multivariate linear regression

Multivariate linear regression I

First, we need the covariances between every feature and the target variable:

$$(\mathbf{X}^{\mathrm{T}}\mathbf{y})_j = \sum_{i=1}^{n} x_{ij} y_i = \sum_{i=1}^{n} (x_{ij} - \mu_j)(y_i - \overline{y}) + n\mu_j\,\overline{y} = n(\sigma_{jy} + \mu_j\,\overline{y})$$

Assuming for the moment that every feature is zero-centred, we have $\mu_j = 0$ and thus $\mathbf{X}^{\mathrm{T}}\mathbf{y}$ is an n-vector holding all the required covariances (times n).

We can normalise the features by means of a d-by-d scaling matrix: a diagonal matrix with diagonal entries $1/n\sigma_{jj}$. If $\mathbf{S}$ is a diagonal matrix with diagonal entries $n\sigma_{jj}$, we can get the required scaling matrix by simply inverting $\mathbf{S}$.

So our first stab at a solution for the *multivariate regression* problem is

$$\hat{\mathbf{w}} = \mathbf{S}^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

Multivariate linear regression II

The general case requires a more elaborate matrix instead of $\mathbf{S}$:

$$\hat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

Let us try to understand the term $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ a bit better.

- Assuming the features are uncorrelated, the covariance matrix $\mathbf{\Sigma}$ is diagonal with entries σ_{jj}.
- Assuming the features are zero-centred, $\mathbf{X}^{\mathrm{T}}\mathbf{X} = n\mathbf{\Sigma}$ is also diagonal with entries $n\sigma_{jj}$.
- In other words, assuming zero-centred and uncorrelated features, $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ reduces to our scaling matrix $\mathbf{S}^{-1}$.

In the general case we cannot make any assumptions about the features, and $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ acts as a transformation that decorrelates, centres and normalises the features.

Example 7.3, p.202

Bivariate linear regression I

First, we derive the basic expressions.

$$\mathbf{X}^{\mathrm{T}}\mathbf{X} = \begin{pmatrix} x_{11} & \cdots & x_{n1} \\ x_{12} & \cdots & x_{n2} \end{pmatrix} \begin{pmatrix} x_{11} & x_{12} \\ \vdots & \vdots \\ x_{n1} & x_{n2} \end{pmatrix} = n \begin{pmatrix} \sigma_{11} + \overline{x_1}^2 & \sigma_{12} + \overline{x_1}\,\overline{x_2} \\ \sigma_{12} + \overline{x_1}\,\overline{x_2} & \sigma_{22} + \overline{x_2}^2 \end{pmatrix}$$

$$(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1} = \frac{1}{nD} \begin{pmatrix} \sigma_{22} + \overline{x_2}^2 & -\sigma_{12} - \overline{x_1}\,\overline{x_2} \\ -\sigma_{12} - \overline{x_1}\,\overline{x_2} & \sigma_{11} + \overline{x_1}^2 \end{pmatrix}$$

$$D = (\sigma_{11} + \overline{x_1}^2)(\sigma_{22} + \overline{x_2}^2) - (\sigma_{12} + \overline{x_1}\,\overline{x_2})^2$$

$$\mathbf{X}^{\mathrm{T}}\mathbf{y} = \begin{pmatrix} x_{11} & \cdots & x_{n1} \\ x_{12} & \cdots & x_{n2} \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = n \begin{pmatrix} \sigma_{1y} + \overline{x_1}\,\overline{y} \\ \sigma_{2y} + \overline{x_2}\,\overline{y} \end{pmatrix}$$

Example 7.3, p.202

Bivariate linear regression II

We now consider two special cases. The first is that $\mathbf{X}$ is in homogeneous coordinates, i.e., we are really dealing with a univariate problem. In that case we have $x_{i1} = 1$ for $1 \leq i \leq n$; $\overline{x_1} = 1$; and $\sigma_{11} = \sigma_{12} = \sigma_{1y} = 0$. We then obtain (we write x instead of x_2, σ_{xx} instead of σ_{22} and σ_{xy} instead of σ_{2y}):

$$
\begin{aligned}
(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1} &= \frac{1}{n\sigma_{xx}} \begin{pmatrix} \sigma_{xx} + \overline{x}^2 & -\overline{x} \\ -\overline{x} & 1 \end{pmatrix} \\
\mathbf{X}^{\mathrm{T}}\mathbf{y} &= n \begin{pmatrix} \overline{y} \\ \sigma_{xy} + \overline{x}\,\overline{y} \end{pmatrix} \\
\hat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} &= \frac{1}{\sigma_{xx}} \begin{pmatrix} \sigma_{xx}\overline{y} - \sigma_{xy}\overline{x} \\ \sigma_{xy} \end{pmatrix}
\end{aligned}
$$

This is the same result as obtained in Example 7.1.

Example 7.3, p.202

Bivariate linear regression III

The second special case we consider is where we assume x_1, x_2 and y to be zero-centred, which means that the intercept is zero and $\mathbf{w}$ contains the two regression coefficients. In this case we obtain

$$\begin{aligned}
(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1} &= \frac{1}{n(\sigma_{11}\sigma_{22}-\sigma_{12}^2)}\begin{pmatrix}\sigma_{22} & -\sigma_{12}\\ -\sigma_{12} & \sigma_{11}\end{pmatrix}\\
\mathbf{X}^{\mathrm{T}}\mathbf{y} &= n\begin{pmatrix}\sigma_{1y}\\ \sigma_{2y}\end{pmatrix}\\
\hat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} &= \frac{1}{(\sigma_{11}\sigma_{22}-\sigma_{12}^2)}\begin{pmatrix}\sigma_{22}\sigma_{1y}-\sigma_{12}\sigma_{2y}\\ \sigma_{11}\sigma_{2y}-\sigma_{12}\sigma_{1y}\end{pmatrix}
\end{aligned}$$

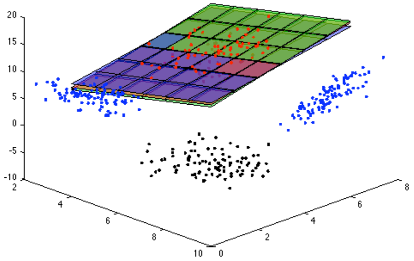
The last expression shows, e.g., that the regression coefficient for x_1 may be non-zero even if x_1 doesn't correlate with the target variable ($\sigma_{1y}=0$), on account of the correlation between x_1 and x_2 ($\sigma_{12}\neq 0$).

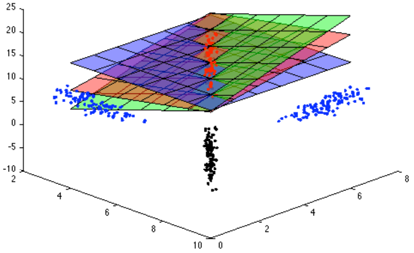
Important point to remember

Assuming uncorrelated features effectively decomposes a multivariate regression problem into d univariate problems.

Figure 7.3, p.204

Feature correlation





(left) Regression functions learned by linear regression. The true function is $y = x_1 + x_2$ (red plane). The red points are noisy samples of this function; the black points show them projected onto the (x_1, x_2)-plane. The green plane indicates the function learned by linear regression; the blue plane is the result of decomposing the problem into two univariate regression problems (blue points). Both are good approximations of the true function. **(right)** The same function, but now x_1 and x_2 are highly (negatively) correlated. The samples now give much less information about the true function: indeed, from the univariate decomposition it appears that the function is constant.

Important point to remember

A general way of constructing a linear classifier with decision boundary $\mathbf{w} \cdot \mathbf{x} = t$ is by constructing $\mathbf{w}$ as $\mathbf{M}^{-1}(n^{\oplus}\boldsymbol{\mu}^{\oplus} - n^{\ominus}\boldsymbol{\mu}^{\ominus})$, with different possible choices of $\mathbf{M}$, $n^{\oplus}$ and $n^{\ominus}$.

What's next?

The perceptron

A linear classifier that will achieve perfect separation on linearly separable data is the *perceptron*, originally proposed as a simple neural network. The perceptron iterates over the training set, updating the weight vector every time it encounters an incorrectly classified example.

☞ For example, let $\mathbf{x}_i$ be a misclassified positive example, then we have $y_i = +1$ and $\mathbf{w} \cdot \mathbf{x}_i < t$. We therefore want to find $\mathbf{w}'$ such that $\mathbf{w}' \cdot \mathbf{x}_i > \mathbf{w} \cdot \mathbf{x}_i$, which moves the decision boundary towards and hopefully past x_i.

☞ This can be achieved by calculating the new weight vector as $\mathbf{w}' = \mathbf{w} + \eta \mathbf{x}_i$, where $0 < \eta \leq 1$ is the *learning rate* (often set to 1). We then have $\mathbf{w}' \cdot \mathbf{x}_i = \mathbf{w} \cdot \mathbf{x}_i + \eta \mathbf{x}_i \cdot \mathbf{x}_i > \mathbf{w} \cdot \mathbf{x}_i$ as required.

☞ Similarly, if $\mathbf{x}_j$ is a misclassified negative example, then we have $y_j = -1$ and $\mathbf{w} \cdot \mathbf{x}_j > t$. In this case we calculate the new weight vector as $\mathbf{w}' = \mathbf{w} - \eta \mathbf{x}_j$, and thus $\mathbf{w}' \cdot \mathbf{x}_j = \mathbf{w} \cdot \mathbf{x}_j - \eta \mathbf{x}_j \cdot \mathbf{x}_j < \mathbf{w} \cdot \mathbf{x}_j$.

☞ The two cases can be combined in a single update rule:

$$\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i$$

Algorithm 7.1, p.208

Perceptron

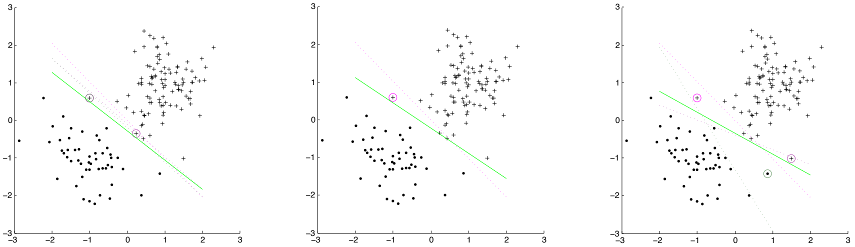
Algorithm Perceptron(D, η) – train a perceptron for linear classification.

Input : labelled training data D in homogeneous coordinates; learning rate η.
Output : weight vector $\mathbf{w}$ defining classifier $\hat{y} = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x})$.

```
$\mathbf{w} \leftarrow \mathbf{0}$ ;                // Other initialisations of the weight vector are possible
$converged \leftarrow$ false;
while $converged =$ false do
    $converged \leftarrow$ true;
    for $i = 1$ to $|D|$ do
        if $y_i \mathbf{w} \cdot \mathbf{x}_i \leq 0$          // i.e., $\hat{y}_i \neq y_i$
         then
            $\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$;
            $converged \leftarrow$ false;       // We changed $\mathbf{w}$ so haven't converged yet
        end
    end
end
```

Figure 7.5, p.209

Varying the learning rate



(left) A perceptron trained with a small learning rate ($\eta = 0.2$). The circled examples are the ones that trigger the weight update. **(middle)** Increasing the learning rate to $\eta = 0.5$ leads in this case to a rapid convergence. **(right)** Increasing the learning rate further to $\eta = 1$ may lead to too aggressive weight updating, which harms convergence. The starting point in all three cases was the basic linear classifier.

Linear classifiers in dual form

Every time an example $\mathbf{x}_i$ is misclassified, we add $y_i\mathbf{x}_i$ to the weight vector.

- ☞ After training has completed, each example has been misclassified zero or more times. Denoting this number as α_i for example $\mathbf{x}_i$, the weight vector can be expressed as

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

- ☞ In the dual, instance-based view of linear classification we are learning instance weights α_i rather than feature weights w_j. An instance $\mathbf{x}$ is classified as

$$\hat{y} = \mathrm{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x}\right)$$

- ☞ During training, the only information needed about the training data is all pairwise dot products: the n-by-n matrix $\mathbf{G} = \mathbf{X}\mathbf{X}^{\mathrm{T}}$ containing these dot products is called the *Gram matrix*.

Algorithm 7.2, p.209

Perceptron training in dual form

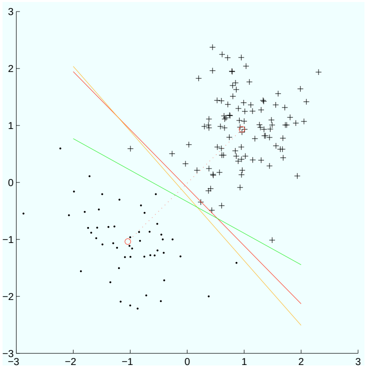
Algorithm DualPerceptron(D) – perceptron training in dual form.

Input : labelled training data D in homogeneous coordinates.
Output : coefficients α_i defining weight vector $\mathbf{w} = \sum_{i=1}^{|D|} \alpha_i y_i \mathbf{x}_i$.

```
$\alpha_i \leftarrow 0$ for $1 \le i \le |D|$;
$converged \leftarrow$ false;
while $converged =$ false do
    $converged \leftarrow$ true;
    for $i = 1$ to $|D|$ do
        if $y_i \sum_{j=1}^{|D|} \alpha_j y_j \mathbf{x}_i \cdot \mathbf{x}_j \le 0$ then
            $\alpha_i \leftarrow \alpha_i + 1$;
            $converged \leftarrow$ false;
        end
    end
end
```

Figure 7.6, p.210

Comparing linear classifiers



Three differently trained linear classifiers on a data set of 100 positives (top-right) and 50 negatives (bottom-left): the basic linear classifier in red, the least-squares classifier in orange and the perceptron in green. Notice that the perceptron perfectly separates the training data, but its heuristic approach may lead to overfitting in certain situations.

Algorithm 7.3, p.211

Training a perceptron for regression

Algorithm PerceptronRegression(D, T) – train a perceptron for regression.

Input : labelled training data D in homogeneous coordinates;
maximum number of training epochs T.

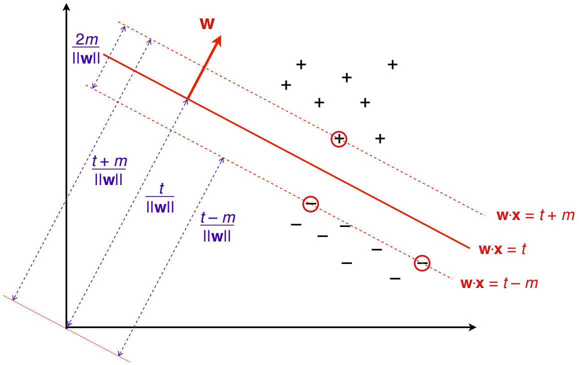
Output : weight vector $\mathbf{w}$ defining function approximator $\hat{y} = \mathbf{w} \cdot \mathbf{x}$.

```
1 $\mathbf{w} \leftarrow \mathbf{0}$; $t \leftarrow 0$;
2 while $t < T$ do
3     for $i = 1$ to $|D|$ do
4         $\mathbf{w} \leftarrow \mathbf{w} + (y_i - \hat{y}_i)^2 \mathbf{x}_i$;
5     end
6     $t \leftarrow t + 1$;
7 end
```

What's next?

Figure 7.7, p.212

Support vector machine



w
$\frac{2m}{||\mathbf{w}||}$
$\frac{t+m}{||\mathbf{w}||}$
$\frac{t}{||\mathbf{w}||}$
$\frac{t-m}{||\mathbf{w}||}$
$\mathbf{w}{\cdot}\mathbf{x} = t+m$
$\mathbf{w}{\cdot}\mathbf{x} = t$
$\mathbf{w}{\cdot}\mathbf{x} = t-m$

The geometry of a support vector classifier. The circled data points are the support vectors, which are the training examples nearest to the decision boundary. The support vector machine finds the decision boundary that maximises the margin $m/||\mathbf{w}||$.

Maximising the margin

Since we are free to rescale t, $||\mathbf{w}||$ and m, it is customary to choose $m = 1$. Maximising the margin then corresponds to minimising $||\mathbf{w}||$ or, more conveniently, $\frac{1}{2}||\mathbf{w}||^2$, provided of course that none of the training points fall inside the margin.

This leads to a quadratic, constrained optimisation problem:

$$\mathbf{w}^*, t^* = \operatorname*{arg\,min}_{\mathbf{w},t} \frac{1}{2}||\mathbf{w}||^2 \qquad \text{subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i - t) \geq 1, 1 \leq i \leq n$$

Using the method of Lagrange multipliers, the dual form of this problem can be derived (see Background 7.3).

SVM in dual form

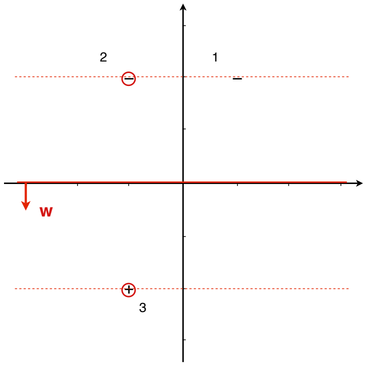
The dual optimisation problem for support vector machines is to maximise the dual Lagrangian under positivity constraints and one equality constraint:

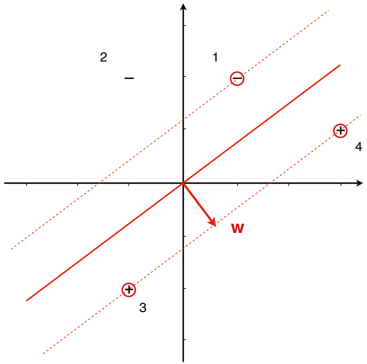
$$\alpha_1^*, \ldots, \alpha_n^* = \operatorname*{arg\,max}_{\alpha_1, \ldots, \alpha_n} -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i$$

$$\text{subject to } \alpha_i \geq 0, 1 \leq i \leq n \text{ and } \sum_{i=1}^{n} \alpha_i y_i = 0$$

Figure 7.8, p.215

Two maximum-margin classifiers





(left) A maximum-margin classifier built from three examples, with $\mathbf{w} = (0, -1/2)$ and margin 2. The circled examples are the support vectors: they receive non-zero Lagrange multipliers and define the decision boundary. **(right)** By adding a second positive the decision boundary is rotated to $\mathbf{w} = (3/5, -4/5)$ and the margin decreases to 1.

Example 7.5, p.215

Two maximum-margin classifiers I

$$\mathbf{X} = \begin{pmatrix} 1 & 2 \\ -1 & 2 \\ -1 & -2 \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} -1 \\ -1 \\ +1 \end{pmatrix} \qquad \mathbf{X}' = \begin{pmatrix} -1 & -2 \\ 1 & -2 \\ -1 & -2 \end{pmatrix}$$

The matrix $\mathbf{X}'$ on the right incorporates the class labels; i.e., the rows are $y_i\mathbf{x}_i$. The Gram matrix is (without and with class labels):

$$\mathbf{X}\mathbf{X}^{\mathrm{T}} = \begin{pmatrix} 5 & 3 & -5 \\ 3 & 5 & -3 \\ -5 & -3 & 5 \end{pmatrix} \qquad \mathbf{X}'\mathbf{X}'^{\mathrm{T}} = \begin{pmatrix} 5 & 3 & 5 \\ 3 & 5 & 3 \\ 5 & 3 & 5 \end{pmatrix}$$

The dual optimisation problem is thus

$$\begin{aligned} &\mathop{\arg\max}_{\alpha_1,\alpha_2,\alpha_3} -\frac{1}{2}\left(5\alpha_1^2 + 3\alpha_1\alpha_2 + 5\alpha_1\alpha_3 + 3\alpha_2\alpha_1 + 5\alpha_2^2 + 3\alpha_2\alpha_3 + 5\alpha_3\alpha_1 + 3\alpha_3\alpha_2 + 5\alpha_3^2\right) + \alpha_1 + \alpha_2 + \alpha_3 \\ = &\mathop{\arg\max}_{\alpha_1,\alpha_2,\alpha_3} -\frac{1}{2}\left(5\alpha_1^2 + 6\alpha_1\alpha_2 + 10\alpha_1\alpha_3 + 5\alpha_2^2 + 6\alpha_2\alpha_3 + 5\alpha_3^2\right) + \alpha_1 + \alpha_2 + \alpha_3 \end{aligned}$$

subject to $\alpha_1 \geq 0, \alpha_2 \geq 0, \alpha_3 \geq 0$ and $-\alpha_1 - \alpha_2 + \alpha_3 = 0$.

Example 7.5, p.215

Two maximum-margin classifiers II

- Using the equality constraint we can eliminate one of the variables, say α_3, and simplify the objective function to

$$\mathop{\arg\max}_{\alpha_1,\alpha_2,\alpha_3} -\frac{1}{2}\left(20\alpha_1^2+32\alpha_1\alpha_2+16\alpha_2^2\right)+2\alpha_1+2\alpha_2$$

- Setting partial derivatives to 0 we obtain $-20\alpha_1-16\alpha_2+2=0$ and $-16\alpha_1-16\alpha_2+2=0$ (notice that, because the objective function is quadratic, these equations are guaranteed to be linear).
- We therefore obtain the solution $\alpha_1=0$ and $\alpha_2=\alpha_3=1/8$. We then have $\mathbf{w}=1/8(\mathbf{x}_3-\mathbf{x}_2)=\begin{pmatrix}0\\-1/2\end{pmatrix}$, resulting in a margin of $1/||\mathbf{w}||=2$.
- Finally, t can be obtained from any support vector, say $\mathbf{x}_2$, since $y_2(\mathbf{w}\cdot\mathbf{x}_2-t)=1$; this gives $-1\cdot(-1-t)=1$, hence $t=0$.

Example 7.5, p.215

Two maximum-margin classifiers III

We now add an additional positive at $(3,1)$. This gives the following data matrices:

$$\mathbf{X}' = \begin{pmatrix} -1 & -2 \\ 1 & -2 \\ -1 & -2 \\ 3 & 1 \end{pmatrix} \qquad \mathbf{X}'\mathbf{X}'^{\mathrm{T}} = \begin{pmatrix} 5 & 3 & 5 & -5 \\ 3 & 5 & 3 & 1 \\ 5 & 3 & 5 & -5 \\ -5 & 1 & -5 & 10 \end{pmatrix}$$

- ☞ It can be verified by similar calculations to those above that the margin decreases to 1 and the decision boundary rotates to $\mathbf{w} = \begin{pmatrix} 3/5 \\ -4/5 \end{pmatrix}$.
- ☞ The Lagrange multipliers now are $\alpha_1 = 1/2$, $\alpha_2 = 0$, $\alpha_3 = 1/10$ and $\alpha_4 = 2/5$. Thus, only $\mathbf{x}_3$ is a support vector in both the original and the extended data set.

Soft margin SVM

Allowing margin errors I

The idea is to introduce *slack variables* ξ_i, one for each example, which allow some of them to be inside the margin or even at the wrong side of the decision boundary.

$$\mathbf{w}^*, t^*, \xi_i^* = \operatorname*{arg\,min}_{\mathbf{w}, t, \xi_i} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i$$

$$\text{subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i - t) \geq 1 - \xi_i \text{ and } \xi_i \geq 0, 1 \leq i \leq n$$

- ☞ C is a user-defined parameter trading off margin maximisation against slack variable minimisation: a high value of C means that margin errors incur a high penalty, while a low value permits more margin errors (possibly including misclassifications) in order to achieve a large margin.
- ☞ If we allow more margin errors we need fewer support vectors, hence C controls to some extent the 'complexity' of the SVM and hence is often referred to as the *complexity parameter*.

Allowing margin errors II

The Lagrange function is then as follows:

$$\begin{aligned}
\Lambda(\mathbf{w}, t, \xi_i, \alpha_i, \beta_i) &= \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i - t) - (1-\xi_i)) - \sum_{i=1}^{n}\beta_i\xi_i \\
&= \Lambda(\mathbf{w}, t, \alpha_i) + \sum_{i=1}^{n}(C - \alpha_i - \beta_i)\xi_i
\end{aligned}$$

- ☞ For an optimal solution every partial derivative with respect to ξ_i should be 0, from which it follows that the added term vanishes from the dual problem.
- ☞ Furthermore, since both α_i and β_i are positive, this means that α_i cannot be larger than C:

$$\alpha_1^*, \dots, \alpha_n^* = \operatorname*{arg\,max}_{\alpha_1,\dots,\alpha_n} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i\cdot\mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$

$$\text{subject to } 0 \leq \alpha_i \leq C \text{ and } \sum_{i=1}^{n}\alpha_i y_i = 0$$

Three cases for the training instances

What is the significance of the upper bound C on the α_i multipliers?

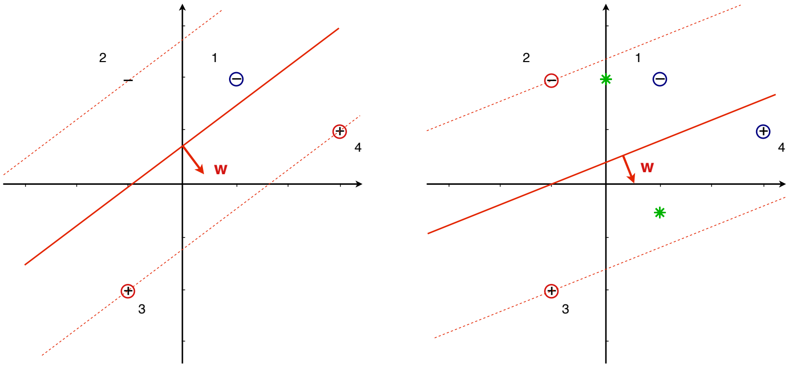
- Since $C - \alpha_i - \beta_i = 0$ for all i, $\alpha_i = C$ implies $\beta_i = 0$. The β_i multipliers come from the $\xi_i \geq 0$ constraint, and a multiplier of 0 means that the lower bound is not reached, i.e., $\xi_i > 0$ (analogous to the fact that $\alpha_j = 0$ means that $\mathbf{x}_j$ is not a support vector and hence $\mathbf{w} \cdot \mathbf{x}_j - t > 1$).
- In other words, a solution to the soft margin optimisation problem in dual form divides the training examples into three cases:

$\alpha_i = 0$	these are outside or on the margin;
$0 < \alpha_i < C$	these are the support vectors on the margin;
$\alpha_i = C$	these are on or inside the margin.

- Notice that we still have $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$, and so both second and third case examples participate in spanning the decision boundary.

Figure 7.9, p.218

Soft margins



(left) The soft margin classifier learned with $C = 5/16$, at which point $\mathbf{x}_2$ is about to become a support vector. **(right)** The soft margin classifier learned with $C = 1/10$: all examples contribute equally to the weight vector. The asterisks denote the class means, and the decision boundary is parallel to the one learned by the basic linear classifier.

Example 7.6, p.218

Soft margins I

- Recall that the Lagrange multipliers for the classifier in Figure 7.8 (right) are $\alpha_1 = 1/2$, $\alpha_2 = 0$, $\alpha_3 = 1/10$ and $\alpha_4 = 2/5$. So α_1 is the largest multiplier, and as long as $C > \alpha_1 = 1/2$ no margin errors are tolerated.
- For $C = 1/2$ we have $\alpha_1 = C$, and hence for $C < 1/2$ we have that $\mathbf{x}_1$ becomes a margin error and the optimal classifier is a soft margin classifier.
- The upper margin reaches $\mathbf{x}_2$ for $C = 5/16$ (Figure 7.9 (left)), at which point we have $\mathbf{w} = \begin{pmatrix} 3/8 \\ -1/2 \end{pmatrix}$, $t = 3/8$ and the margin has increased to 1.6. Furthermore, we have $\xi_1 = 6/8, \alpha_1 = C = 5/16, \alpha_2 = 0, \alpha_3 = 1/16$ and $\alpha_4 = 1/4$.

Example 7.6, p.218

Soft margins II

- If we now decrease C further, the decision boundary starts to rotate clockwise, so that $\mathbf{x}_4$ becomes a margin error as well, and only $\mathbf{x}_2$ and $\mathbf{x}_3$ are support vectors. The boundary rotates until $C = 1/10$, at which point we have $\mathbf{w} = \begin{pmatrix} 1/5 \\ -1/2 \end{pmatrix}$, $t = 1/5$ and the margin has increased to 1.86. Furthermore, we have $\xi_1 = 4/10$ and $\xi_4 = 7/10$, and all multipliers have become equal to C (Figure 7.9 (right)).
- Finally, when C decreases further the decision boundary stays where it is, but the norm of the weight vector gradually decreases and all points become margin errors.

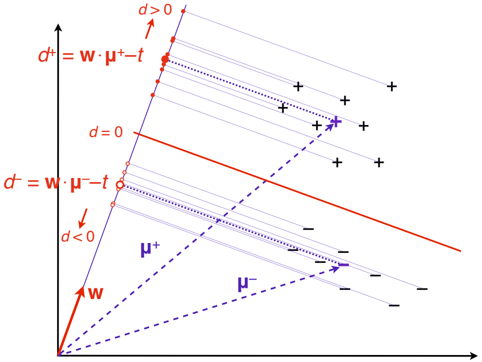
Important point to remember

A minimal-complexity soft margin classifier summarises the classes by their class means in a way very similar to the basic linear classifier.

What's next?

Figure 7.10, p.220

Scores from a linear classifier



We can think of a linear classifier as a projection onto the direction given by $\mathbf{w}$, here assumed to be a unit vector. $\mathbf{w} \cdot \mathbf{x} - t$ gives the signed distance from the decision boundary on the projection line. Also indicated are the class means $\boldsymbol{\mu}^{\oplus}$ and $\boldsymbol{\mu}^{\ominus}$, and the corresponding mean distances $d^{\oplus}$ and $d^{\ominus}$.

Logistic calibration

In order to obtain probability estimates from a linear classifier outputting distance scores d, we convert d into a probability by means of the mapping

$$d \mapsto \frac{\exp(d)}{\exp(d) + 1}$$

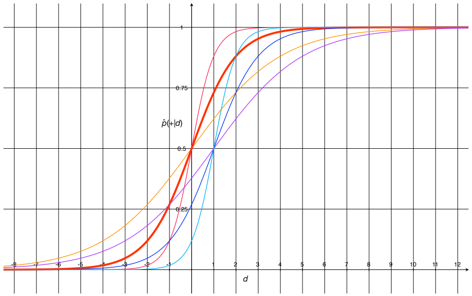
or, equivalently,

$$d \mapsto \frac{1}{1 + \exp(-d)}$$

This S-shaped or *sigmoid* function is called the *logistic function*; it finds applications in a wide range of areas (Figure 7.11).

Figure 7.11, p.222

The logistic function



The **fat** red line indicates the standard logistic function $\hat{p}(d) = \frac{1}{1+\exp(-d)}$; this function can be used to obtain probability estimates if the two classes are equally prevalent and the class means are equidistant from the decision boundary and one unit of variance apart. The steeper and flatter red lines show how the function changes if the class means are 2 and 1/2 units of variance apart, respectively. The three blue lines show how these curves change if $d_0 = 1$, which means that the positives are on average further away from the decision boundary.

Example 7.7, p.222

Logistic calibration of a linear classifier

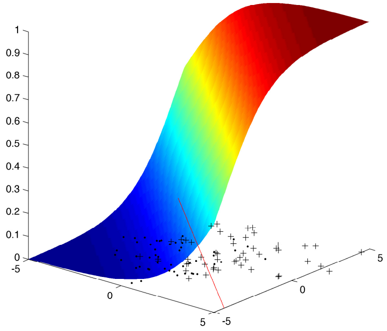
Logistic calibration has a particularly simple form for the basic linear classifier, which has $\mathbf{w} = \boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}$. It follows that

$$\overline{d}^{\oplus} - \overline{d}^{\ominus} = \frac{\mathbf{w} \cdot (\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus})}{||\mathbf{w}||} = \frac{||\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}||^2}{||\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}||} = ||\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}||$$

and hence $\gamma = ||\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}||/\sigma^2$. Furthermore, $d_0 = 0$ as $(\boldsymbol{\mu}^{\oplus} + \boldsymbol{\mu}^{\ominus})/2$ is already on the decision boundary. So in this case logistic calibration does not move the decision boundary, and only adjusts the steepness of the sigmoid according to the separation of the classes. Figure 7.12 illustrates this for some data sampled from two normal distributions with the same diagonal covariance matrix.

Figure 7.12, p.223

Logistic calibration of a linear classifier



The surface shows the sigmoidal probability estimates resulting from logistic calibration of the basic linear classifier on random data satisfying the assumptions of logistic calibration.