

Machine Learning

The Art and Science of Algorithms that Make Sense of Data

Peter A. Flach

Intelligent Systems Laboratory, University of Bristol, United Kingdom

December 29, 2013



PETER FLACH

Machine Learning

The Art and Science of Algorithms
that Make Sense of Data

CAMBRIDGE

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, and advanced material indicated by ★ up to D.

Table of contents I

- 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
 - Two uses of features
 - Feature construction and transformation
- 2 Binary classification and related tasks
 - Classification
 - Assessing classification performance
 - Visualising classification performance

Table of contents II

- Scoring and ranking
 - Assessing and visualising ranking performance
 - Turning rankers into classifiers
- Class probability estimation
 - Assessing class probability estimates
 - Turning rankers into class probability estimators

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

4 Concept learning

- The hypothesis space

Table of contents III

- Least general generalisation
- Internal disjunction
- Paths through the hypothesis space
 - Most general consistent hypotheses
 - Using first-order logic ★
- Learnability ★

5 Tree models

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

6 Rule models

- Learning ordered rule lists

Table of contents IV

- Rule lists for ranking and probability estimation
- Learning unordered rule sets
 - Rule sets for ranking and probability estimation
 - A closer look at rule overlap ★
- Descriptive rule learning
 - Rule learning for subgroup discovery
 - Association rule mining

7 Linear models

- The least-squares method
 - Multivariate linear regression
 - Regularised regression ★
 - Using least-squares regression for classification ★
- The perceptron: a heuristic learning algorithm for linear classifiers
- Support vector machines
 - Soft margin SVM

Table of contents V

- Obtaining probabilities from linear classifiers
- Going beyond linearity with kernel methods ★

8

Distance-based models

- Neighbours and exemplars
- Nearest-neighbour classification
- Distance-based clustering
 - K -means algorithm
 - Clustering around medoids
 - Silhouettes
- Hierarchical clustering
- From kernels to distances ★

9

Probabilistic models

- The normal distribution and its geometric interpretations
- Probabilistic models for categorical data
 - Using a naive Bayes model for classification

Table of contents VI

- Training a naive Bayes model
- Discriminative learning by optimising conditional likelihood ★
- Probabilistic models with hidden variables
 - Expectation-Maximisation ★
 - Gaussian mixture models ★
- Compression-based models ★

10 Features

- Kinds of feature
 - Calculations on features
 - Categorical, ordinal and quantitative features
 - Structured features
- Feature transformations
 - Thresholding and discretisation
 - Normalisation and calibration

11 Model ensembles

Table of contents VII

- Bagging and random forests
- Boosting
 - Bias, variance and margins

- 12 Machine learning experiments
 - What to measure
 - How to measure it
 - How to interpret it
 - Interpretation of results over multiple data sets

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.



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.

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 data.



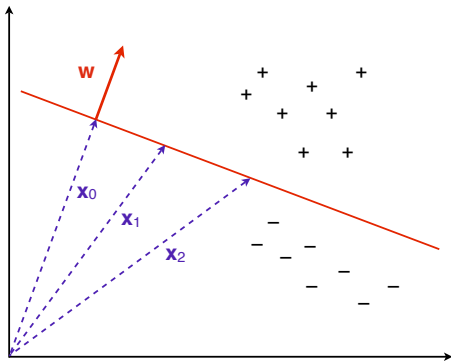
E-mail	x_1	x_2	Spam?	$4x_1 + 4x_2$
1	1	1	1	8
2	0	0	0	0
3	1	0	0	4
4	0	1	0	4

The columns marked x_1 and x_2 indicate the results of two tests on four different e-mails. The fourth column indicates which of the e-mails are spam. The right-most column demonstrates that by thresholding the function $4x_1 + 4x_2$ at 5, we can separate spam from ham.



Figure 1, p.5

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}).



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 $\mathbf{x}^\circ = (1, x_1, \dots, x_n)$ and the extended weight vector is $\mathbf{w}^\circ = (-t, w_1, \dots, w_n)$, 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 $x_0 = 1$.

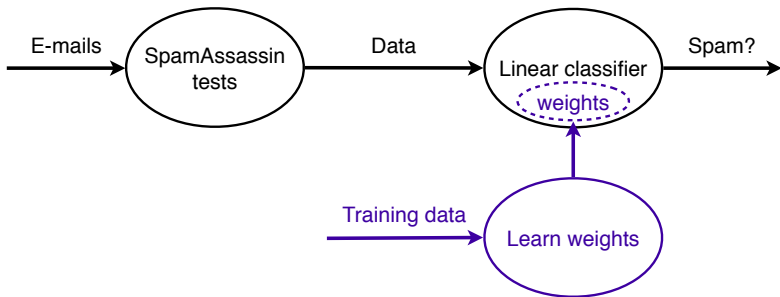
Important point to remember

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



Figure 2, p.5

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.



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.

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.

- ☞ 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

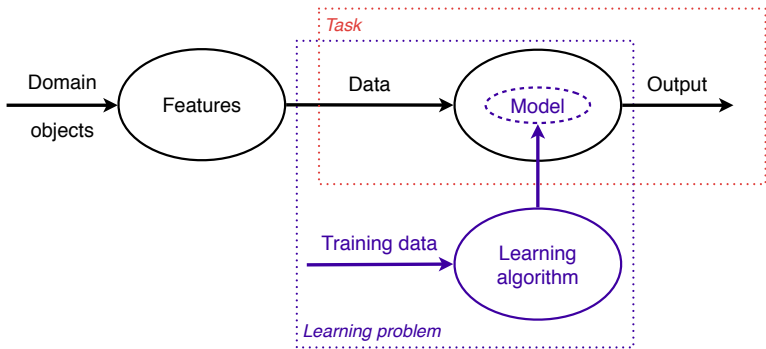
- ☞ if the e-mail contains the word 'Viagra' then estimate the odds of spam as 4:1;
- ☞ otherwise, if it contains the phrase 'blue pill' then estimate the odds of spam as 3:1;
- ☞ otherwise, estimate the odds of spam as 1:6.

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'.



Figure 3, p.11

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
 - Two 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?

- 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**
 - Two uses of features
 - Feature construction and transformation

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.



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 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

	<i>Predictive model</i>	<i>Descriptive model</i>
<i>Supervised learning</i>	classification, regression	subgroup discovery
<i>Unsupervised learning</i>	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?

- 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
 - Two uses of features
 - Feature construction and transformation

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.

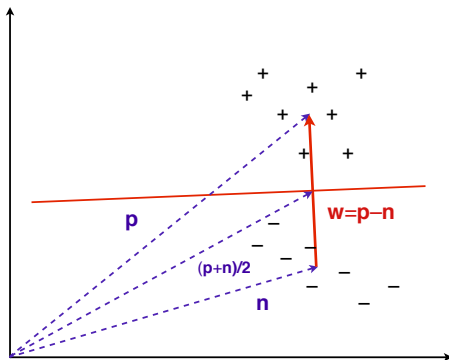
Alternatively, they can be characterised by their *modus operandi*:

- 👉 *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.



Figure 1.1, p.22

Basic linear classifier

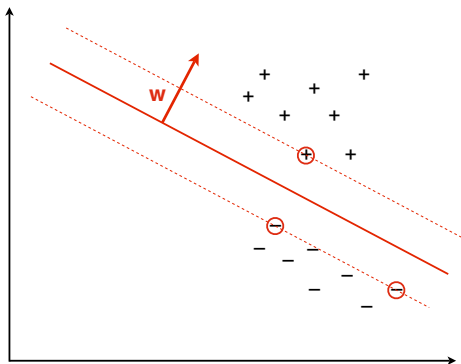


The basic linear classifier constructs a decision boundary by half-way intersecting the line between the positive and negative centres of mass. It is described by the equation $\mathbf{w} \cdot \mathbf{x} = t$, with $\mathbf{w} = \mathbf{p} - \mathbf{n}$; the decision threshold can be found by noting 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.



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 $P(Y = \text{spam}|\text{Viagra}, \text{lottery})$, 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*.



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):

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



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

$$P(Y = \text{spam} | \text{lottery} = 1) = 0.65 \cdot 0.90 + 0.40 \cdot 0.10 = 0.625 \text{ and}$$

$P(Y = \text{ham} | \text{lottery} = 1) = 0.35 \cdot 0.90 + 0.60 \cdot 0.10 = 0.375$. 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: 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	$P(\text{Viagra} = 1 Y)$	$P(\text{Viagra} = 0 Y)$
spam	0.40	0.60
ham	0.12	0.88

Y	$P(\text{lottery} = 1 Y)$	$P(\text{lottery} = 0 Y)$
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):

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

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

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

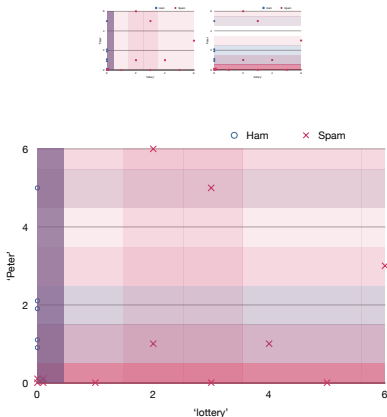
$$\frac{P(\text{Viagra} = 1|Y = \text{spam})}{P(\text{Viagra} = 1|Y = \text{ham})} \frac{P(\text{lottery} = 1|Y = \text{spam})}{P(\text{lottery} = 1|Y = \text{ham})} = \frac{0.40}{0.12} \frac{0.21}{0.13} = 5.4 \quad (0.67)$$

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

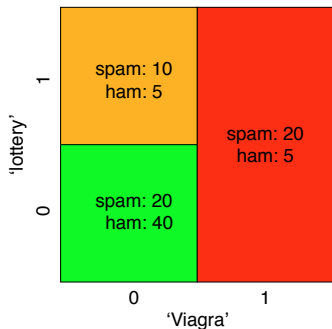
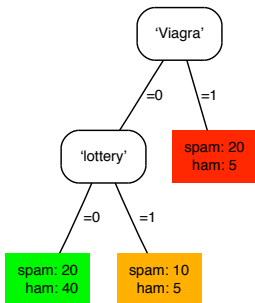


(top) Visualisation of two marginal likelihoods as estimated from a small data set. The colours indicate whether the likelihood points to **spam** or **ham**. **(bottom)** Combining the two marginal likelihoods gives a pattern not unlike that of a Scottish tartan.



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.



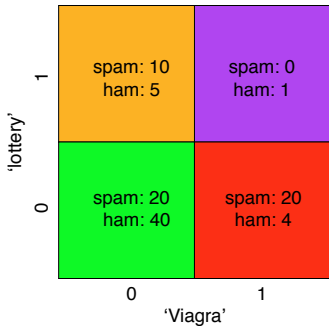
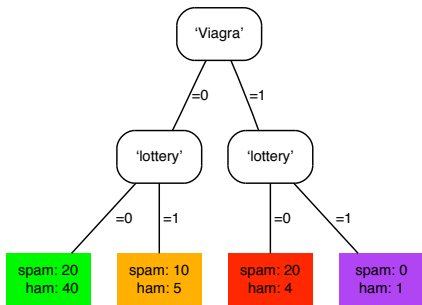
Labelling a feature tree

- 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.



Consider the following rules:

·if lottery = 1 then Class = Y = spam·

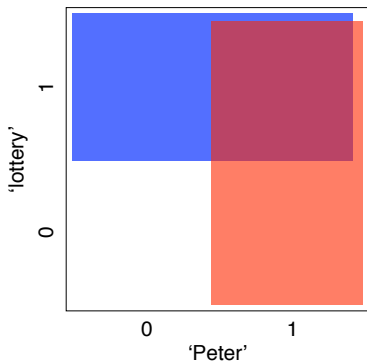
·if Peter = 1 then Class = Y = ham·

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

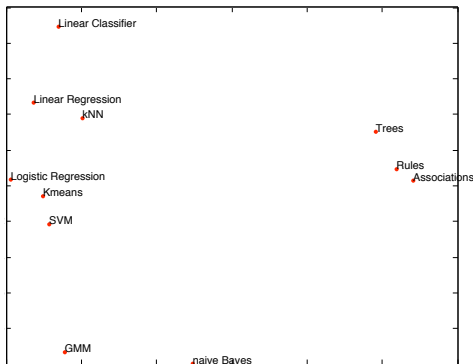


The effect of overlapping rules in instance space. The two rules make contradictory predictions in the top right-hand corner, and no prediction at all in the bottom left-hand corner.



Figure 1.7, p.37

Mapping machine learning models

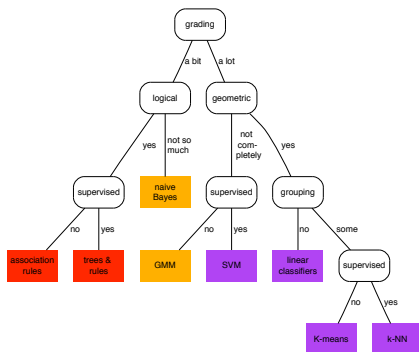


A 'map' of some of the models that will be considered in this book. Models that share characteristics are plotted closer together: logical models to the right, geometric models on the top left and probabilistic models on the bottom left. The horizontal dimension roughly ranges from grading models on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

- 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
 - Two uses of features
 - Feature construction and transformation



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.



Example 1.8, p.41

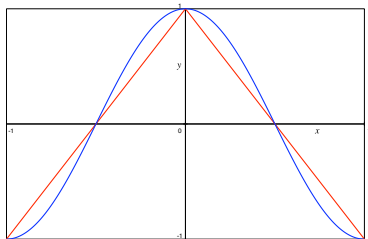
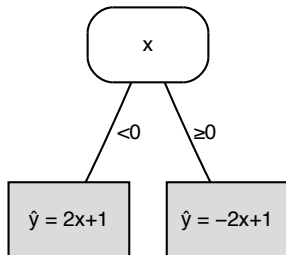
Two uses of features

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



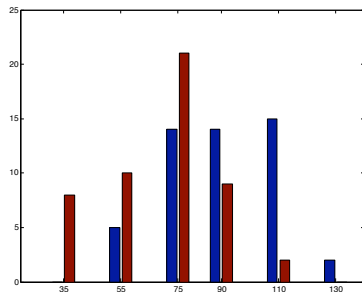
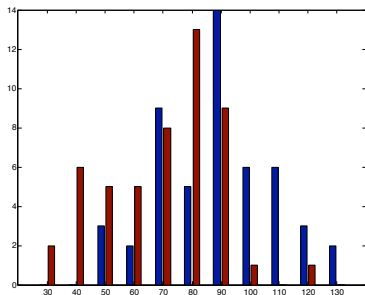
(left) A regression tree combining a one-split feature tree with linear regression models in the leaves. Notice how x is used as both a splitting feature and a regression variable.

(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.



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.



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

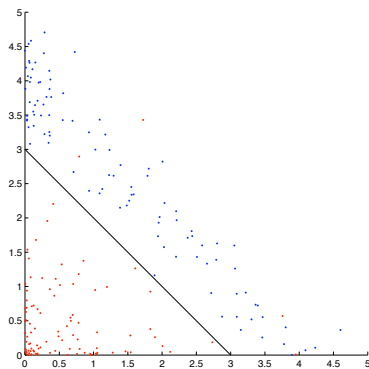
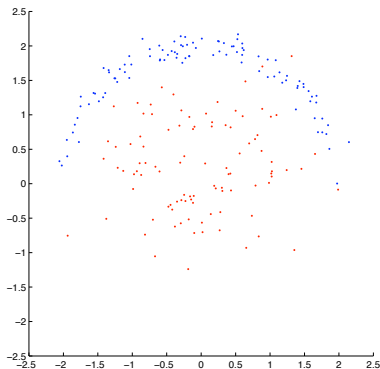
$$\mathbf{x}'_1 \cdot \mathbf{x}'_2 = x_1^2 x_2^2 + y_1^2 y_2^2 + 2x_1 y_1 x_2 y_2 = (x_1 x_2 + y_1 y_2)^2 = (\mathbf{x}_1 \cdot \mathbf{x}_2)^2$$

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?

2 Binary classification and related tasks

- Classification
 - Assessing classification performance
 - Visualising classification performance
- Scoring and ranking
 - Assessing and visualising ranking performance
 - Turning rankers into classifiers
- Class probability estimation
 - Assessing class probability estimates
 - Turning rankers into class probability estimators



Table 2.1, p.52

Predictive machine learning scenarios

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

What's next?

2 Binary classification and related tasks

- **Classification**

- Assessing classification performance
- Visualising classification performance

- **Scoring and ranking**

- Assessing and visualising ranking performance
- Turning rankers into classifiers

- **Class probability estimation**

- Assessing class probability estimates
- Turning rankers into class probability estimators

Classification

A *classifier* is a mapping $\hat{c}: \mathcal{X} \rightarrow \mathcal{C}$, where $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ 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.

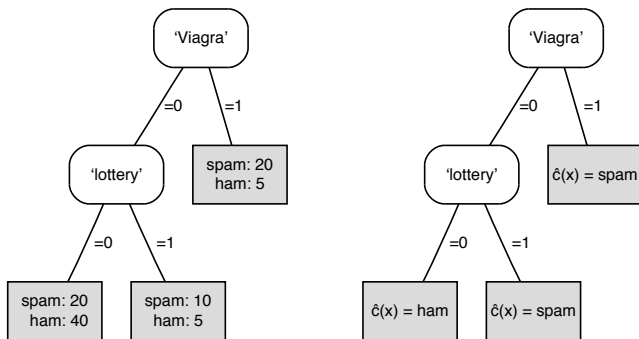
We use the ‘hat’ to indicate that $\hat{c}(x)$ is an estimate of the true but unknown function $c(x)$. Examples for a classifier take the form $(x, c(x))$, where $x \in \mathcal{X}$ is an instance and $c(x)$ is the true class of the instance (sometimes contaminated by noise).

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 \mathcal{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.



Table 2.2, p.54

Contingency table

	<i>Predicted</i> \oplus	<i>Predicted</i> \ominus	
<i>Actual</i> \oplus	30	20	50
<i>Actual</i> \ominus	10	40	50
	40	60	100

	\oplus	\ominus	
\oplus	20	30	50
\ominus	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.



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 \oplus	Predicted \ominus	
Actual \oplus	60	15	75
Actual \ominus	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

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



Table 2.3, p.57

Performance measures I

<i>Measure</i>	<i>Definition</i>	<i>Equal to</i>	<i>Estimates</i>
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

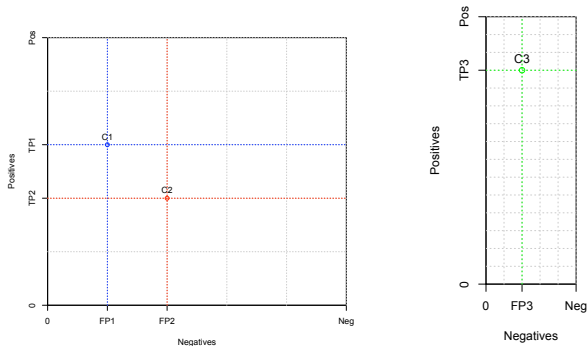
<i>Measure</i>	<i>Definition</i>	<i>Equal to</i>	<i>Estimates</i>
true positive rate, sensitivity, recall	$tpr = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]}{\sum_{x \in Te} I[c(x) = \oplus]}$	<i>TP/Pos</i>	$P(\hat{c}(x) = \oplus c(x) = \oplus)$
true negative rate, specificity	$tnr = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = \ominus]}{\sum_{x \in Te} I[c(x) = \ominus]}$	<i>TN/Neg</i>	$P(\hat{c}(x) = \ominus c(x) = \ominus)$
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.



Figure 2.2, p.58

A coverage plot

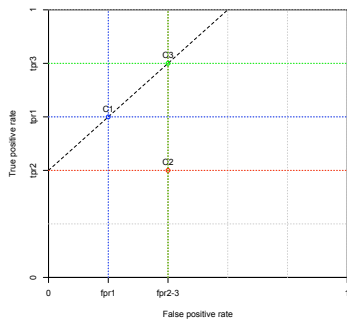
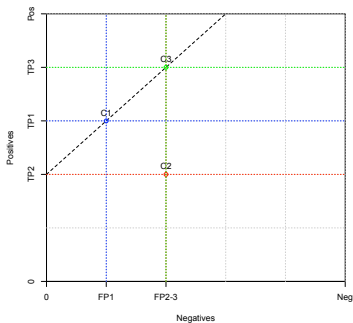


(left) A coverage plot depicting the two contingency tables in Table 2.2. The plot is square because the class distribution is uniform. **(right)** Coverage plot for Example 2.1, with a class ratio $clr = 3$.



Figure 2.3, p.59

An ROC plot



(left) C1 and C3 both dominate C2, but neither dominates the other. The diagonal line indicates that C1 and C3 achieve equal accuracy. **(right)** The same plot with normalised axes. We can interpret this plot as a merger of the two coverage plots in [Figure 2.2](#), employing normalisation to deal with the different class distributions. The diagonal line now indicates that C1 and C3 have the same average recall.

Important points to remember

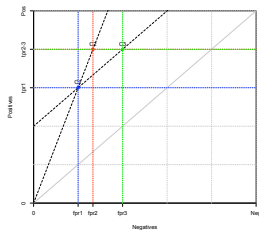
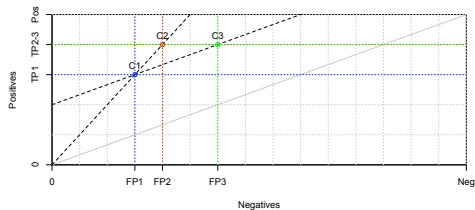
In a coverage plot, classifiers with the same accuracy are connected by line segments with slope 1.

In a ROC plot (i.e., a normalised coverage plot), line segments with slope 1 connect classifiers with the same average recall.



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?

2 Binary classification and related tasks

- Classification
 - Assessing classification performance
 - Visualising classification performance
- Scoring and ranking
 - Assessing and visualising ranking performance
 - Turning rankers into classifiers
- Class probability estimation
 - Assessing class probability estimates
 - Turning rankers into class probability estimators

Scoring classifier

A *scoring classifier* is a mapping $\hat{\mathbf{s}}: \mathcal{X} \rightarrow \mathbb{R}^k$, 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 $\hat{\mathbf{s}}(x) = (\hat{s}_1(x), \dots, \hat{s}_k(x))$ rather than a single number; $\hat{s}_i(x)$ is the score assigned to class C_i for instance x .

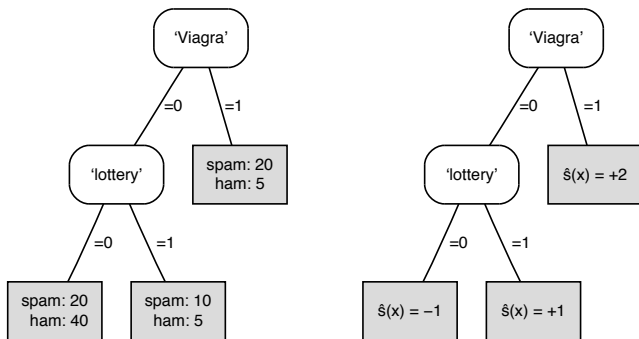
This score indicates how likely it is that class label C_i applies.

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* $L: \mathbb{R} \rightarrow [0, \infty)$ which maps each example's margin $z(x)$ to an associated loss $L(z(x))$.

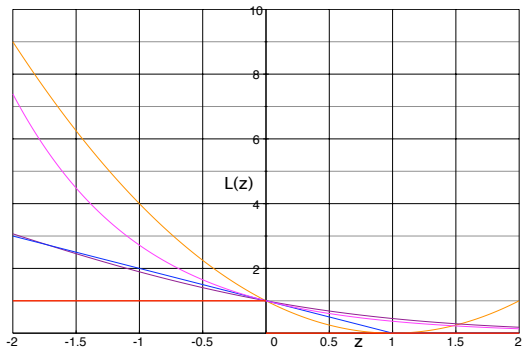
We will assume that $L(0) = 1$, which is the loss incurred by having an example on the decision boundary. We furthermore have $L(z) \geq 1$ for $z < 0$, and usually also $0 \leq L(z) < 1$ for $z > 0$ (Figure 2.6).

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 $L_h(z) = (1 - z)$ if $z \leq 1$, and $L_h(z) = 0$ 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 $L_{\text{sq}}(z) = (1 - z)^2$ (this can be set to 0 for $z > 1$, just like hinge loss).



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 $[...]$ 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 .



The *ranking error rate* is defined as

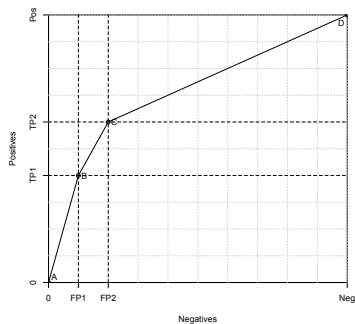
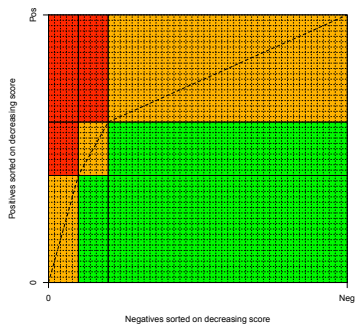
$$\text{rank-err} = \frac{\sum_{x \in Te^{\oplus}, x' \in Te^{\ominus}} I[\hat{s}(x) < \hat{s}(x')] + \frac{1}{2} I[\hat{s}(x) = \hat{s}(x')]}{\text{Pos} \cdot \text{Neg}}$$

- ☞ 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

The area under the ROC curve is the ranking accuracy.

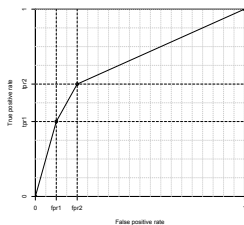
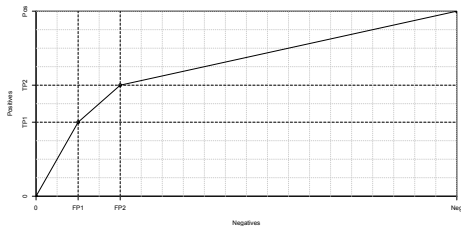


- Suppose we feed the scoring tree in [Figure 2.5](#) an extended test set, with an additional batch of 50 negatives.
- The added negatives happen to be identical to the original ones, so the net effect is that the number of negatives in each leaf doubles.
- As a result the coverage curve changes (because 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.

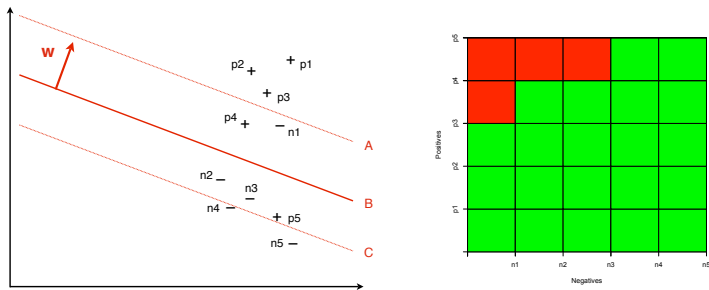
We can derive a score from this linear classifier by taking the distance of an example from the decision boundary; if the example is on the negative side we take the negative distance. This means that the examples are ranked in the following order: $p_1 - p_2 - p_3 - n_1 - p_4 - n_2 - n_3 - p_5 - n_4 - n_5$.

This ranking incurs four ranking errors: n_1 before p_4 , and n_1 , n_2 and n_3 before p_5 . 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).

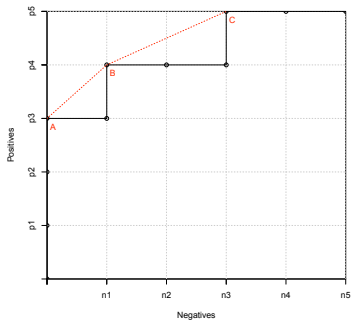
Important point to remember

By decreasing a model's refinement we sometimes achieve better ranking performance.



Figure 2.10, p.70

Coverage curve of grading classifier



The coverage curve of the linear classifier in [Figure 2.9](#). The points labelled A, B and C indicate the classification performance of the corresponding decision boundaries. The dotted lines indicate the improvement that can be obtained by turning the grading classifier into a grouping classifier with four segments.

Important point to remember

Grouping model ROC curves have as many line segments as there are instance space segments in the model; grading models have one line segment for each example in the data set.



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 3 spam against 2 ham is representative, you can select the optimal point on the ROC curve using an isometric with slope $2/3$. 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).



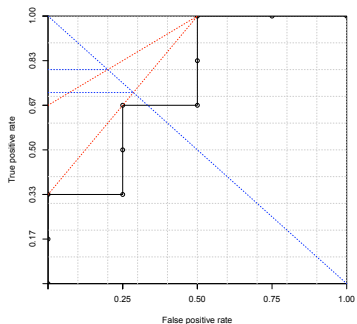
An alternative way of finding the optimal point is to iterate over all possible split points – from before the top ranked e-mail to after the bottom one – and calculate the number of correctly classified examples at each split: 4 – 5 – 6 – 5 – 6 – 7 – 6 – 7 – 8 – 7 – 6. The maximum is achieved at the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

2 Binary classification and related tasks

- Classification
 - Assessing classification performance
 - Visualising classification performance
- Scoring and ranking
 - Assessing and visualising ranking performance
 - Turning rankers into classifiers
- **Class probability estimation**
 - Assessing class probability estimates
 - Turning rankers into class probability estimators

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 $\hat{\mathbf{p}} : \mathcal{X} \rightarrow [0, 1]^k$. We write $\hat{\mathbf{p}}(x) = (\hat{p}_1(x), \dots, \hat{p}_k(x))$, where $\hat{p}_i(x)$ is the probability assigned to class C_i for instance x , and $\sum_{i=1}^k \hat{p}_i(x) = 1$.

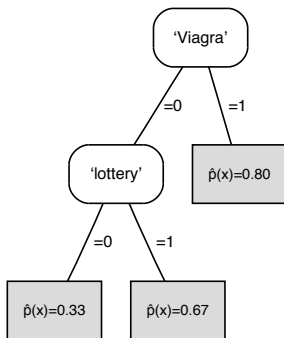
If we have only two classes, the probability associated with one class is 1 minus the probability of the other class; in that case, we use $\hat{p}(x)$ to denote the estimated probability of the positive class for instance x .

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](#).

Mean squared probability error

We can define the *squared error* (*SE*) of the predicted probability vector

$\hat{\mathbf{p}}(x) = (\hat{p}_1(x), \dots, \hat{p}_k(x))$ as

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

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

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

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.



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):

$$SE_1 = 20(0.33 - 1)^2 + 40(0.33 - 0)^2 = 13.33$$

$$SE_2 = 10(0.67 - 1)^2 + 5(0.67 - 0)^2 = 3.33$$

$$SE_3 = 20(0.80 - 1)^2 + 5(0.80 - 0)^2 = 4.00$$

which leads to a mean squared error of $MSE = \frac{1}{100}(SE_1 + SE_2 + SE_3) = 0.21$. 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:

$$SE'_1 = 20(0.40 - 1)^2 + 40(0.40 - 0)^2 = 13.6$$

$$SE''_1 = 20(0.20 - 1)^2 + 40(0.20 - 0)^2 = 14.4$$

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:

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

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

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 $\pi_i = 1/k$.

★ Calibration loss and refinement loss

If all elements of S receive the same predicted probability vector $\hat{\mathbf{p}}(S)$ – which happens if S is a segment of a grouping model – then a similar derivation to the one above allows us to write

$$\begin{aligned} \text{SE}(S) &= \sum_{x \in S} \text{SE}(x) = \sum_{x \in S} \frac{1}{2} \sum_{i=1}^k (\hat{p}_i(x) - I[c(x) = C_i])^2 \\ &= \frac{1}{2} |S| \sum_{i=1}^k (\hat{p}_i(S) - \dot{p}_i(S))^2 + \frac{1}{2} |S| \sum_{i=1}^k (\dot{p}_i(S)(1 - \dot{p}_i(S))) \end{aligned}$$

The **first term** of the final expression is called the *calibration loss*, and measures squared error with respect to the empirical probabilities. It can be reduced to 0 in grouping models where we are free to choose the predicted probabilities for each segment, as in probability estimation trees. Models with low calibration loss are said to be well-calibrated.

The **second term** is called the *refinement loss*; this depends only on the empirical probabilities, and is smaller if they are less uniform.

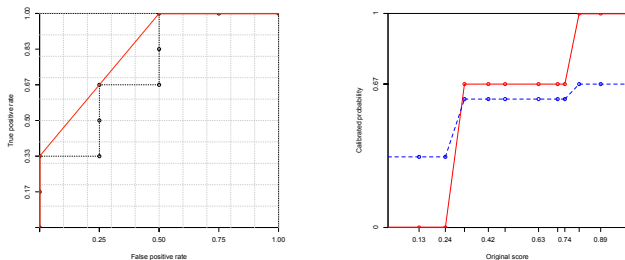
Important point to remember

Concavities in ROC curves can be remedied by combining segments through tied scores.



Figure 2.13, p.77

ROC convex hull



(left) The solid red line is the convex hull of the dotted ROC curve. (right) The corresponding calibration map in red: the plateaus correspond to several examples being mapped to the same segment of the convex hull, and linear interpolation between example scores occurs when we transition from one convex hull segment to the next. A Laplace-corrected calibration map is indicated by the dashed line in blue: Laplace smoothing compresses the range of calibrated probabilities but can sometimes affect the ranking.

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?

- 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



Example 3.1, p.82

Performance of multi-class classifiers I

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

	<i>Predicted</i>			
	15	2	3	20
<i>Actual</i>	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 $45/50 = 0.90$.



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?).



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.



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.



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 $(14, 19, 11)$.



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 $\hat{\mathbf{s}}(x) = (\hat{s}_1(x), \dots, \hat{s}_k(x))$ 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.



- 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 **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.



Reweighting multi-class scores

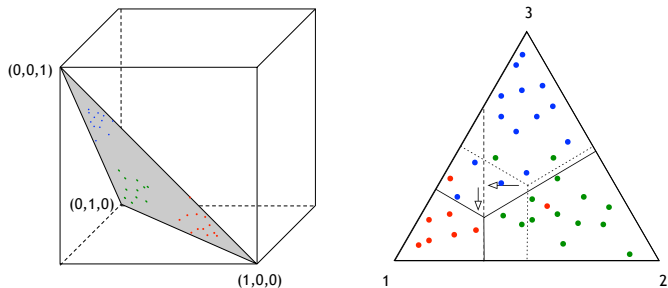
We illustrate the procedure for a three-class probabilistic classifier. The probability vectors $\hat{\mathbf{p}}(x) = (\hat{p}_1(x), \hat{p}_2(x), \hat{p}_3(x))$ can be thought of as points inside the unit cube. Since the probabilities add up to 1, the points lie in an equilateral triangle connecting three corners of the cube (Figure 3.1 (left)). Each corner of this triangle represents one of the classes; the probability assigned to a particular class in a given point is proportional to the distance to the opposite side.

Any decision rule of the form $\arg \max_i w_i \hat{s}_i(x)$ cuts the triangle in three areas using lines perpendicular to the sides. For the unweighted decision rule these lines intersect in the triangle's centre of mass (Figure 3.1 (right)). Optimising the separation between C_2 against C_1 means moving this point along a line parallel to the base of the triangle, moving away from the class that receives greater weight. Once the optimal point on this line is found, we optimise the separation of C_3 against the first two classes by moving in a direction perpendicular to the previous line.



Figure 3.1, p.89

★ Reweighting multi-class scores



(left) Triples of probabilistic scores represented as points in an equilateral triangle connecting three corners of the unit cube. **(right)** The arrows show how the weights are adjusted from the initial equal weights (dotted lines), first by optimising the separation of C_2 against C_1 (dashed line), then by optimising the separation of C_3 against the other two classes (solid lines). The end result is that the weight of C_1 is considerably decreased, to the benefit of the other two classes.



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?

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

Real-valued targets

A *function estimator*, also called a *regressor*, is a mapping $\hat{f}: \mathcal{X} \rightarrow \mathbb{R}$. 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.



Consider the following set of five points:

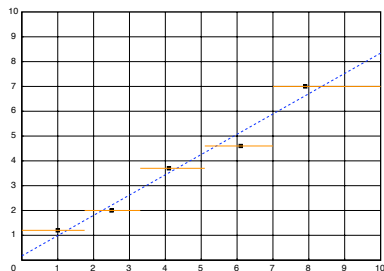
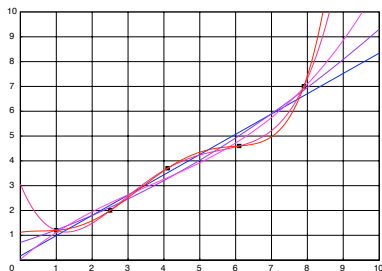
x	y
1.0	1.2
2.5	2.0
4.1	3.7
6.1	4.6
7.9	7.0

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.

★ Bias and variance I

If we underestimate the number of parameters of the model, we will not be able to decrease the loss to zero, regardless of how much training data we have.

On the other hand, with a larger number of parameters the model will be more dependent on the training sample, and small variations in the training sample can result in a considerably different model.

This is sometimes called the *bias–variance dilemma*: a low-complexity model suffers less from variability due to random variations in the training data, but may introduce a systematic bias that even large amounts of training data can't resolve; on the other hand, a high-complexity model eliminates such bias but can suffer non-systematic errors due to variance.

★ Bias and variance II

We can make this a bit more precise by noting that expected squared loss on a training example x can be decomposed as follows:

$$\mathbb{E} \left[(f(x) - \hat{f}(x))^2 \right] = (f(x) - \mathbb{E}[\hat{f}(x)])^2 + \mathbb{E} \left[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2 \right]$$

The expectation is taken over different training sets and hence different function estimators.

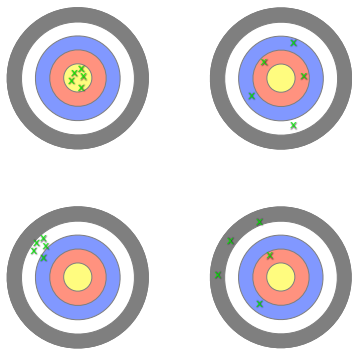
The first term on the right is zero if these function estimators get it right on average; otherwise the learning algorithm exhibits a systematic *bias* of some kind.

The second term quantifies the *variance* in the function estimates $\hat{f}(x)$ as a result of variations in the training set.



Figure 3.3, p.94

★ Bias and variance



A dartboard metaphor illustrating the concepts of bias and variance. Each dartboard corresponds to a different learning algorithm, and each dart signifies a different training sample. The top row learning algorithms exhibit low bias, staying close to the bull's eye (the true function value for a particular x) on average, while the ones on the bottom row have high bias. The left column shows low variance and the right column high variance.

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



Table 3.1, p.95

Unsupervised and descriptive learning

	<i>Predictive model</i>	<i>Descriptive model</i>
<i>Supervised learning</i>	classification, regression	subgroup discovery
<i>Unsupervised learning</i>	predictive clustering	descriptive clustering, association rule discovery

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

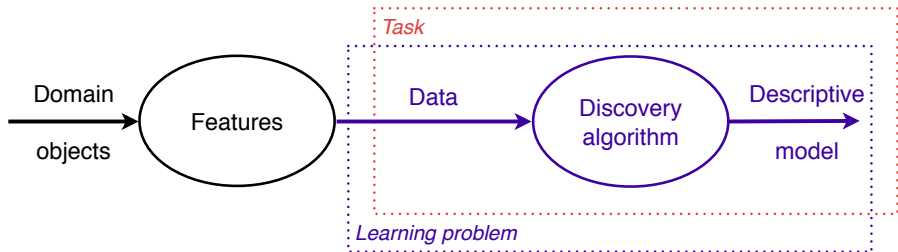
Important point to remember

In descriptive learning the task and learning problem coincide.



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

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 $\hat{q} : \mathcal{X} \rightarrow \mathcal{C}$, where $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ 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 $D \subseteq \mathcal{X}$ would be a mapping $\hat{q} : D \rightarrow \mathcal{C}$ whose domain is D rather than \mathcal{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 $\hat{q} \subseteq \mathcal{X} \times \mathcal{X}$ or $\hat{q} \subseteq D \times D$ or, equivalently, as a partition of \mathcal{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 $D = D_1 \uplus \dots \uplus D_K$ 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:

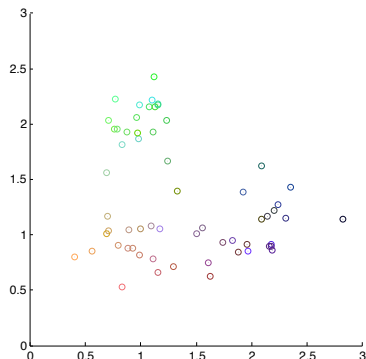
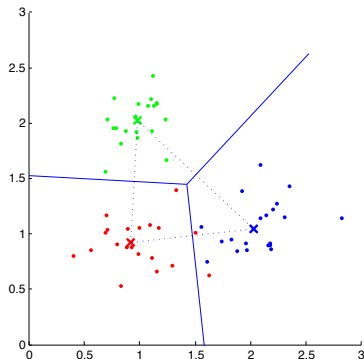
- 👉 by applying a heuristic approach, which finds a ‘good enough’ solution rather than the best possible one;
- 👉 by relaxing the problem into a ‘soft’ clustering problem, by allowing instances a degree of membership in more than one cluster.

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:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ \dots & \dots & \dots \end{pmatrix}$$

$$\begin{pmatrix} 0.40 & 0.30 & 0.30 \\ 0.40 & 0.51 & 0.09 \\ 0.44 & 0.29 & 0.27 \\ 0.35 & 0.08 & 0.57 \\ \dots & \dots & \dots \end{pmatrix}$$



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:

	<i>Are together</i>	<i>Are not together</i>	
<i>Should be together</i>	2	2	4
<i>Should not be together</i>	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*.



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.



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 Concept learning
 - The hypothesis space
 - Least general generalisation
 - Internal disjunction
 - Paths through the hypothesis space
 - Most general consistent hypotheses
 - Using first-order logic ★
 - Learnability ★

What's next?

- 4 Concept learning
 - The hypothesis space
 - Least general generalisation
 - Internal disjunction
 - Paths through the hypothesis space
 - Most general consistent hypotheses
 - Using first-order logic ★
 - Learnability ★



Example 4.1, p.106

Learning conjunctive concepts

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. Using these features, the first animal can be described by the following conjunction:

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

The next one has the same characteristics but is a metre longer, so you drop the length condition and generalise the conjunction to

$$\text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$$

The third animal is again 3 metres long, has a beak, no gills and few teeth, so your description becomes

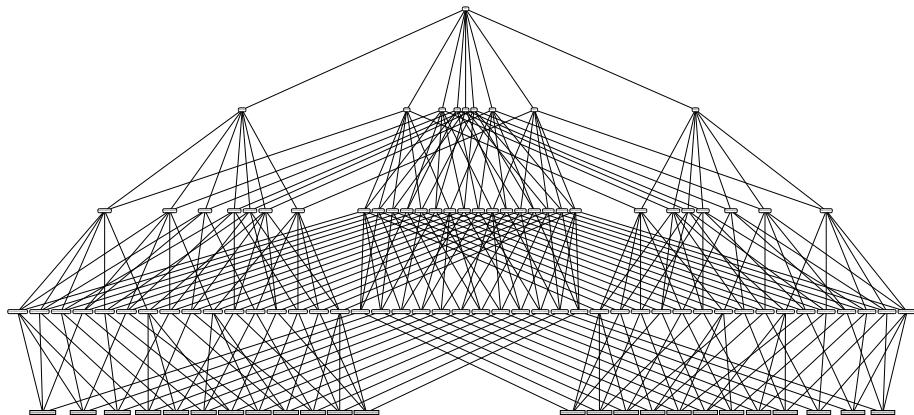
$$\text{Gills} = \text{no} \wedge \text{Beak} = \text{yes}$$

All remaining animals satisfy this conjunction, and you finally decide they are some kind of dolphin.



Figure 4.1, p.107

A hypothesis space

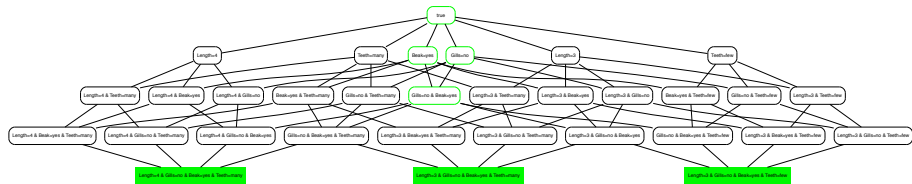


The hypothesis space corresponding to [Example 4.1](#). The bottom row corresponds to the 24 possible instances, which are complete conjunctions with four literals each.



Figure 4.2, p.109

Reducing the hypothesis space



Part of the hypothesis space in Figure 4.1 containing only concepts that are more general than at least one of the three given instances on the bottom row. Only four conjunctions, indicated in green at the top, are more general than all three instances; the least general of these is $Gills = no \wedge Beak = yes$. It can be observed that the two left-most and right-most instances would be sufficient to learn that concept.

Least General Generalisation (LGG)

Intuitively, the LGG of two instances is the nearest concept in the hypothesis space where paths upward from both instances intersect. The fact that this point is unique is a special property of many logical hypothesis spaces, and can be put to good use in learning.

More precisely, such a hypothesis space forms a *lattice*: a partial order in which each two elements have a *least upper bound (lub)* and a *greatest lower bound (glb)*. So, the LGG of a set of instances is exactly the least upper bound of the instances in that lattice.

Furthermore, it is the greatest lower bound of the set of all generalisations of the instances: all possible generalisations are at least as general as the LGG. In this very precise sense, the LGG is the most conservative generalisation that we can learn from the data.



Least general generalisation

Algorithm LGG-Set(D) – find least general generalisation of a set of instances.

Input : data D .

Output : logical expression H .

```
1  $x \leftarrow$  first instance from  $D$ ;  
2  $H \leftarrow x$ ;  
3 while instances left do  
4   |  $x \leftarrow$  next instance from  $D$ ;  
5   |  $H \leftarrow$  LGG( $H, x$ ) ; // e.g., LGG-Conj (Alg. 4.2) or LGG-Conj-ID (Alg. 4.3)  
6 end  
7 return  $H$ 
```



Least general conjunctive generalisation

Algorithm LGG-Conj(x, y) – find least general conjunctive generalisation of two conjunctions.

Input : conjunctions x, y .

Output : conjunction z .

- 1 $z \leftarrow$ conjunction of all literals common to x and y ;
 - 2 **return** z
-



In Example 4.1 we observed the following dolphins:

p1: Length = 3 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

p2: Length = 4 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

p3: Length = 3 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few

Suppose you next observe an animal that clearly doesn't belong to the species – a negative example. It is described by the following conjunction:

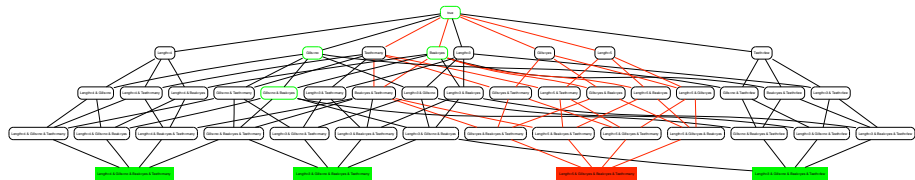
n1: Length = 5 \wedge Gills = yes \wedge Beak = yes \wedge Teeth = many

This negative example rules out some of the generalisations that were hitherto still possible: in particular, it rules out the concept **Beak = yes**, as well as the empty concept which postulates that everything is a dolphin.



Figure 4.3, p.111

Employing negative examples



A negative example can rule out some of the generalisations of the LGG of the positive examples. Every concept which is connected by a **red** path to a negative example covers that negative and is therefore ruled out as a hypothesis. Only two conjunctions cover all positives and no negatives: **Gills = no** \wedge **Beak = yes** and **Gills = no**.



We now enrich our hypothesis language with internal disjunction between values of the same feature. Using the same three positive examples as in [Example 4.1](#), the second and third hypothesis are now

$$\text{Length} = [3, 4] \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$$

and

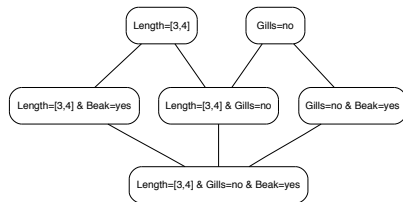
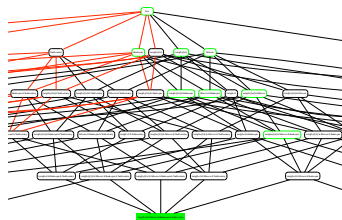
$$\text{Length} = [3, 4] \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes}$$

We can drop any of the three conditions in the latter LGG without covering the negative example from [Example 4.2](#). Generalising further to single conditions, we see that $\text{Length} = [3, 4]$ and $\text{Gills} = \text{no}$ are still OK but $\text{Beak} = \text{yes}$ is not, as it covers the negative example.



Figure 4.4, p.113

Hypothesis space with internal disjunction



(left) A snapshot of the expanded hypothesis space that arises when internal disjunction is used for the 'Length' feature. We now need one more generalisation step to travel upwards from a completely specified example to the empty conjunction. **(right)** The version space consists of one least general hypothesis, two most general hypotheses, and three in between.



Algorithm LGG-Conj-ID(x, y) – find least general conjunctive generalisation of two conjunctions, employing internal disjunction.

Input : conjunctions x, y .

Output : conjunction z .

```

1  $z \leftarrow \text{true}$ ;
2 for each feature  $f$  do
3   | if  $f = v_x$  is a conjunct in  $x$  and  $f = v_y$  is a conjunct in  $y$  then
4   |   | add  $f = \text{Combine-ID}(v_x, v_y)$  to  $z$ ;           // Combine-ID: see text
5   | end
6 end
7 return  $z$ 

```

What's next?

4 Concept learning

- The hypothesis space
 - Least general generalisation
 - Internal disjunction
- Paths through the hypothesis space
 - Most general consistent hypotheses
 - Using first-order logic ★
- Learnability ★

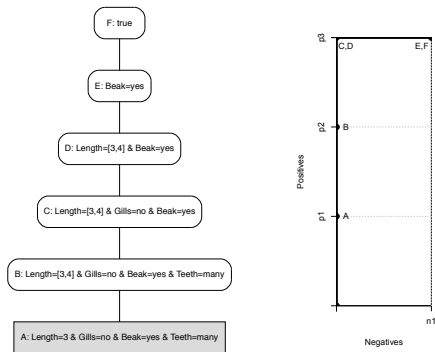


A concept is *complete* if it covers all positive examples. A concept is *consistent* if it covers none of the negative examples. The *version space* is the set of all complete and consistent concepts. This set is convex and is fully defined by its least and most general elements.



Figure 4.5, p.114

Paths through the hypothesis space



(left) A path in the hypothesis space of Figure 4.3 from one of the positive examples (p_1 , see Example 4.2) all the way up to the empty concept. Concept A covers a single example; B covers one additional example; C and D are in the version space, and so cover all three positives; E and F also cover the negative. **(right)** The corresponding coverage curve, with ranking $p_1 - p_2 - p_3 - n_1$.

Important point to remember

An upward path through the hypothesis space corresponds to a coverage curve.



Example 4.4, p.115

Data that is not conjunctively separable I

Suppose we have the following five positive examples (the first three are the same as in [Example 4.1](#)):

p1: Length = 3 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

p2: Length = 4 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

p3: Length = 3 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few

p4: Length = 5 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

p5: Length = 5 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few

and the following negatives (the first one is the same as in [Example 4.2](#)):

n1: Length = 5 \wedge Gills = yes \wedge Beak = yes \wedge Teeth = many

n2: Length = 4 \wedge Gills = yes \wedge Beak = yes \wedge Teeth = many

n3: Length = 5 \wedge Gills = yes \wedge Beak = no \wedge Teeth = many

n4: Length = 4 \wedge Gills = yes \wedge Beak = no \wedge Teeth = many

n5: Length = 4 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few



Example 4.4, p.115

Data that is not conjunctively separable II

The least general complete hypothesis is $\text{Gills} = \text{no} \wedge \text{Beak} = \text{yes}$ as before, but this covers n_5 and hence is inconsistent. There are seven most general consistent hypotheses, none of which are complete:

- $\text{Length} = 3$ (covers p_1 and p_3)
- $\text{Length} = [3, 5] \wedge \text{Gills} = \text{no}$ (covers all positives except p_2)
- $\text{Length} = [3, 5] \wedge \text{Teeth} = \text{few}$ (covers p_3 and p_5)
- $\text{Gills} = \text{no} \wedge \text{Teeth} = \text{many}$ (covers p_1, p_2 and p_4)
- $\text{Gills} = \text{no} \wedge \text{Beak} = \text{no}$
- $\text{Gills} = \text{yes} \wedge \text{Teeth} = \text{few}$
- $\text{Beak} = \text{no} \wedge \text{Teeth} = \text{few}$

The last three of these do not cover any positive examples.



Most general consistent specialisation

Algorithm $\text{MGConsistent}(C, N)$ – find most general consistent specialisations of a concept.

Input : concept C ; negative examples N .

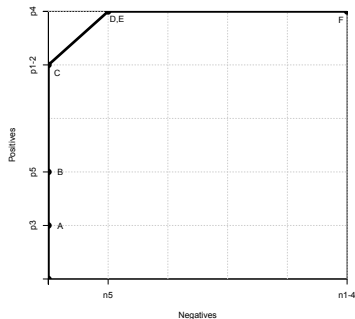
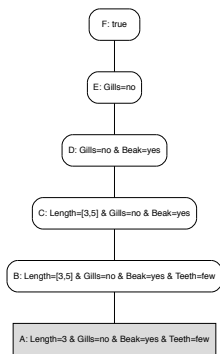
Output : set of concepts S .

```
1 if  $C$  doesn't cover any element from  $N$  then return  $\{C\}$ ;  
2  $S \leftarrow \emptyset$ ;  
3 for each minimal specialisation  $C'$  of  $C$  do  
4   |  $S \leftarrow S \cup \text{MGConsistent}(C', N)$ ;  
5 end  
6 return  $S$ 
```



Figure 4.6, p.117

Another path



(left) A path in the hypothesis space of [Example 4.4](#). Concept A covers a single positive (p_3); B covers one additional positive (p_5); C covers all positives except p_4 ; D is the LGG of all five positive examples, but also covers a negative (n_5), as does E. **(right)** The corresponding coverage curve.

★ Closed concepts

In this example, concepts D and E occupy the same point in coverage space: generalising D into E by dropping $\text{Beak} = \text{yes}$ does not change its coverage. That the data suggests that, in the context of concept E, the condition $\text{Beak} = \text{yes}$ is implicitly understood.

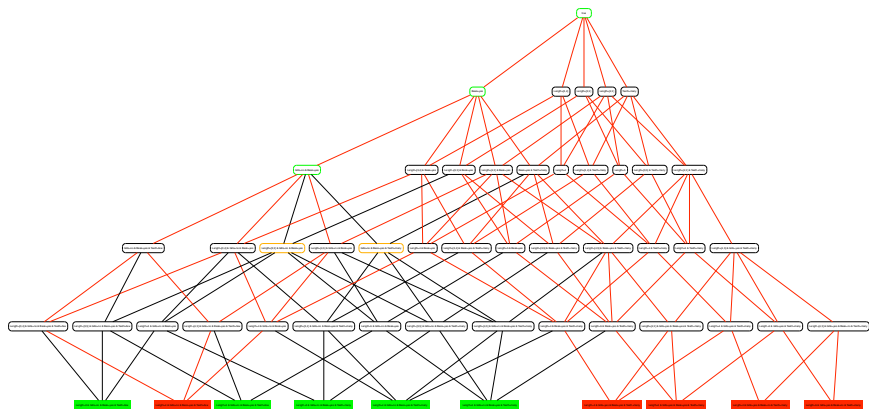
A concept that includes all implicitly understood conditions is called a *closed concept*. Essentially, a closed concept is the LGG of all examples that it covers. For instance, D and E both cover all positives and n5; the LGG of those six examples is $\text{Gills} = \text{no} \wedge \text{Beak} = \text{yes}$, which is D. Mathematically speaking we say that the closure of E is D, which is also its own closure – hence the term ‘closed concept’.

This doesn’t mean that D and E are logically equivalent: there exist instances in \mathcal{X} that are covered by E but not by D. However, none of these ‘witnesses’ are present in the data, and thus, as far as the data is concerned, D and E are indistinguishable.



Figure 4.7, p.118

★ Closed concepts



The hypothesis space is reduced considerably if we restrict attention to closed concepts. There are three, rather than four, complete concepts (in green), and two, rather than seven, most general consistent closed concepts (in orange).



Algorithm 4.5, p.120

★ Learning a conjunction of Horn clauses

Algorithm $\text{Horn}(Mb, Eq)$ – learn a conjunction of Horn clauses from membership and equivalence oracles.

Input : equivalence oracle Eq ; membership oracle Mb .

Output : Horn theory h equivalent to target formula f .

```

1  $h \leftarrow \text{true}$ ; // conjunction of Horn clauses, initially empty
2  $S \leftarrow \emptyset$ ; // a list of negative examples, initially empty
3 while  $Eq(h)$  returns counter-example  $x$  do
4     if  $x$  violates at least one clause of  $h$  then //  $x$  is a false negative
5         | specialise  $h$  by removing every clause that  $x$  violates
6     else //  $x$  is a false positive
7         | find the first negative example  $s \in S$  such that (i)  $z = s \cap x$  has fewer true literals than  $s$ , and
8         | (ii)  $Mb(z)$  labels it as a negative;
9         | if such an example exists then replace  $s$  in  $S$  with  $z$ , else append  $x$  to the end of  $S$ ;
10        |  $h \leftarrow \text{true}$ ;
11        | for all  $s \in S$  do // rebuild  $h$  from  $S$ 
12            |  $p \leftarrow$  the conjunction of literals true in  $s$ ;
13            |  $Q \leftarrow$  the set of literals false in  $s$ ;
14            | for all  $q \in Q$  do  $h \leftarrow h \wedge (p \rightarrow q)$ ;
15        | end
16    end
17 return  $h$ 

```



Example 4.5, p.121

★ Learning a Horn theory I

Suppose the target theory f is

$$(\text{ManyTeeth} \wedge \text{Short} \rightarrow \text{Beak}) \wedge (\text{ManyTeeth} \wedge \text{Gills} \rightarrow \text{Short})$$

This theory has 12 positive examples: eight in which **ManyTeeth** is **false**; another two in which **ManyTeeth** is **true** but both **Gills** and **Short** are **false**; and two more in which **ManyTeeth**, **Short** and **Beak** are **true**. The negative examples, then, are

n1: **ManyTeeth** \wedge **Gills** \wedge **Short** \wedge \neg **Beak**

n2: **ManyTeeth** \wedge **Gills** \wedge \neg **Short** \wedge **Beak**

n3: **ManyTeeth** \wedge **Gills** \wedge \neg **Short** \wedge \neg **Beak**

n4: **ManyTeeth** \wedge \neg **Gills** \wedge **Short** \wedge \neg **Beak**

S is initialised to the empty list and h to the empty conjunction. We call the equivalence oracle which returns a counter-example which has to be a false positive (since every example satisfies our initial hypothesis), say n1 which



Example 4.5, p.121

★ Learning a Horn theory II

violates the first clause in f . There are no negative examples in S yet, so we add n_1 to S (step 8 of Algorithm 4.5). We then generate a new hypothesis from S (steps 9–13): p is **ManyTeeth** \wedge **Gills** \wedge **Short** and Q is **{Beak}**, so h becomes (**ManyTeeth** \wedge **Gills** \wedge **Short** \rightarrow **Beak**). Notice that this clause is implied by our target theory: if **ManyTeeth** and **Gills** are **true** then so is **Short** by the second clause of f ; but then so is **Beak** by f 's first clause. But we need more clauses to exclude all the negatives.

Now, suppose the next counter-example is the false positive n_2 . We form the intersection with n_1 which was already in S to see if we can get a negative example with fewer literals set to **true** (step 7). The result is equal to n_3 so the membership oracle will confirm this as a negative, and we replace n_1 in S with n_3 . We then rebuild h from S which gives (p is **ManyTeeth** \wedge **Gills** and Q is **{Short, Beak}**)

$$(\text{ManyTeeth} \wedge \text{Gills} \rightarrow \text{Short}) \wedge (\text{ManyTeeth} \wedge \text{Gills} \rightarrow \text{Beak})$$



Example 4.5, p.121

★ Learning a Horn theory III

Finally, assume that n_4 is the next false positive returned by the equivalence oracle. The intersection with n_3 on S is actually a positive example, so instead of intersecting with n_3 we append n_4 to S and rebuild h . This gives the previous two clauses from n_3 plus the following two from n_4 :

$$(\text{ManyTeeth} \wedge \text{Short} \rightarrow \text{Gills}) \wedge (\text{ManyTeeth} \wedge \text{Short} \rightarrow \text{Beak})$$

The first of this second pair will subsequently be removed by a false negative from the equivalence oracle, leading to the final theory

$$\begin{aligned} &(\text{ManyTeeth} \wedge \text{Gills} \rightarrow \text{Short}) \wedge \\ &(\text{ManyTeeth} \wedge \text{Gills} \rightarrow \text{Beak}) \wedge \\ &(\text{ManyTeeth} \wedge \text{Short} \rightarrow \text{Beak}) \end{aligned}$$

which is logically equivalent (though not identical) to f .



Example 4.6, p.123

★ Using first-order logic

Consider the following terms:

$\text{BodyPart}(x, \text{PairOf}(\text{Gill}))$

describing the objects that have a pair of gills;

$\text{BodyPart}(\text{Dolphin42}, \text{PairOf}(y))$

describing the body parts that **Dolphin42** has a pair of.

The following two terms are their unification and anti-unification, respectively:

$\text{BodyPart}(\text{Dolphin42}, \text{PairOf}(\text{Gill}))$

describing **Dolphin42** as having a pair of gills;

$\text{BodyPart}(x, \text{PairOf}(y))$

describing the objects that have a pair of unspecified body parts.

What's next?

4 Concept learning

- The hypothesis space
 - Least general generalisation
 - Internal disjunction
- Paths through the hypothesis space
 - Most general consistent hypotheses
 - Using first-order logic ★
- Learnability ★



★ Shattering a set of instances

Consider the following instances:

$m = \text{ManyTeeth} \wedge \neg\text{Gills} \wedge \neg\text{Short} \wedge \neg\text{Beak}$

$g = \neg\text{ManyTeeth} \wedge \text{Gills} \wedge \neg\text{Short} \wedge \neg\text{Beak}$

$s = \neg\text{ManyTeeth} \wedge \neg\text{Gills} \wedge \text{Short} \wedge \neg\text{Beak}$

$b = \neg\text{ManyTeeth} \wedge \neg\text{Gills} \wedge \neg\text{Short} \wedge \text{Beak}$

There are 16 different subsets of the set $\{m, g, s, b\}$. Can each of them be represented by its own conjunctive concept? The answer is yes: for every instance we want to exclude, we add the corresponding negated literal to the conjunction. Thus, $\{m, s\}$ is represented by $\neg\text{Gills} \wedge \neg\text{Beak}$, $\{g, s, b\}$ is represented by $\neg\text{ManyTeeth}$, $\{s\}$ is represented by $\neg\text{ManyTeeth} \wedge \neg\text{Gills} \wedge \neg\text{Beak}$, and so on. We say that this set of four instances is *shattered* by the hypothesis language of conjunctive concepts.

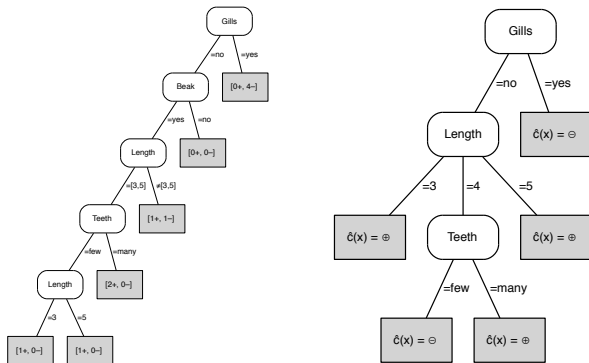
What's next?

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



Figure 5.1, p.130

Paths as trees



(left) The path from Figure 4.6, redrawn in the form of a tree. The coverage numbers in the leaves are obtained from the data in Example 4.4. **(right)** A decision tree learned on the same data. This tree separates the positives and negatives perfectly.

Important point to remember

Decision trees are strictly more expressive than conjunctive concepts.

Important point to remember

One way to avoid overfitting and encourage learning is to deliberately choose a restrictive hypothesis language.



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 $\text{GrowTree}(D, F)$ – grow a feature tree from training data.

Input : data D ; set of features F .

Output : feature tree T with labelled leaves.

```

1 if  $\text{Homogeneous}(D)$  then return  $\text{Label}(D)$ ;
2  $S \leftarrow \text{BestSplit}(D, F)$ ; // 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  $T_i \leftarrow \text{GrowTree}(D_i, F)$ ;
6   | else  $T_i$  is a leaf labelled with  $\text{Label}(D)$ ;
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?

5

Tree models

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



Figure 5.2, p.134

Measuring impurity I

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

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

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

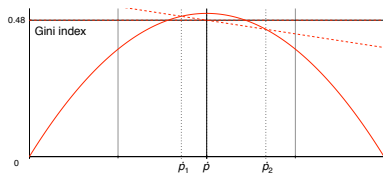
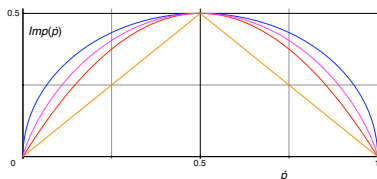
For a binary split there is a nice geometric construction to find $\text{Imp}(\{D_1, D_2\})$:

- ☞ We first find the impurity values $\text{Imp}(D_1)$ and $\text{Imp}(D_2)$ 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, $\min(\hat{p}, 1 - \hat{p})$; the Gini index, $2\hat{p}(1 - \hat{p})$; entropy, $-\hat{p}\log_2 \hat{p} - (1 - \hat{p})\log_2(1 - \hat{p})$ (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, $\sqrt{\hat{p}(1 - \hat{p})}$ – notice that this last function describes a semi-circle. **(right)** Geometric construction to determine the impurity of a split (**Teeth = [many, few]** from **Example 5.1**): \hat{p} is the empirical probability of the parent, and \hat{p}_1 and \hat{p}_2 are the empirical probabilities of the children.



Example 5.1, p.135

Calculating impurity I

Consider again the data in [Example 4.4](#). 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:

Length = [3, 4, 5] [2+, 0-][1+, 3-][2+, 2-]

Gills = [yes, no] [0+, 4-][5+, 1-]

Beak = [yes, no] [5+, 3-][0+, 2-]

Teeth = [many, 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:

$$\text{Gills} \quad 4/10 \cdot 0 + 6/10 \cdot \left(-\frac{5}{6} \log_2(5/6) - \frac{1}{6} \log_2(1/6) \right) = 0.39;$$

$$\text{Beak} \quad 8/10 \cdot \left(-\frac{5}{8} \log_2(5/8) - \frac{3}{8} \log_2(3/8) \right) + 2/10 \cdot 0 = 0.76;$$

$$\text{Teeth} \quad 7/10 \cdot \left(-\frac{3}{7} \log_2(3/7) - \frac{4}{7} \log_2(4/7) \right) \\ + 3/10 \cdot \left(-\frac{2}{3} \log_2(2/3) - \frac{1}{3} \log_2(1/3) \right) = 0.97.$$

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):

$$\text{Length} \quad 2/10 \cdot 2 \cdot (2/2 \cdot 0/2) + 4/10 \cdot 2 \cdot (1/4 \cdot 3/4) + 4/10 \cdot 2 \cdot (2/4 \cdot 2/4) = 0.35;$$

$$\text{Gills} \quad 4/10 \cdot 0 + 6/10 \cdot 2 \cdot (5/6 \cdot 1/6) = 0.17;$$

$$\text{Beak} \quad 8/10 \cdot 2 \cdot (5/8 \cdot 3/8) + 2/10 \cdot 0 = 0.38;$$

$$\text{Teeth} \quad 7/10 \cdot 2 \cdot (3/7 \cdot 4/7) + 3/10 \cdot 2 \cdot (2/3 \cdot 1/3) = 0.48.$$

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'.



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 .

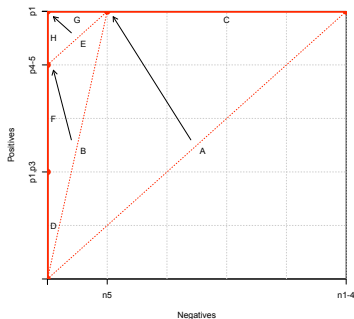
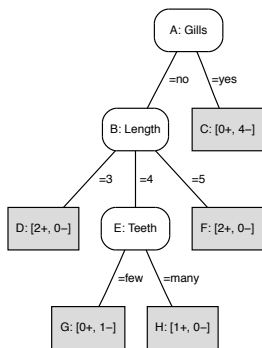
Output : feature f to split on.

```
1  $I_{\min} \leftarrow 1$ ;  
2 for each  $f \in F$  do  
3   | split  $D$  into subsets  $D_1, \dots, D_l$  according to the values  $v_j$  of  $f$ ;  
4   | if  $\text{Imp}(\{D_1, \dots, D_l\}) < I_{\min}$  then  
5   |   |  $I_{\min} \leftarrow \text{Imp}(\{D_1, \dots, D_l\})$ ;  
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 in [Example 4.4](#). **(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?

5

Tree models

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

Important point to remember

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



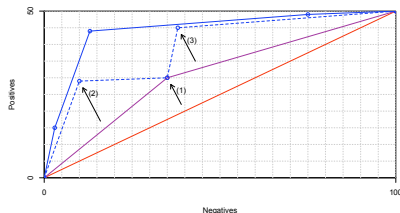
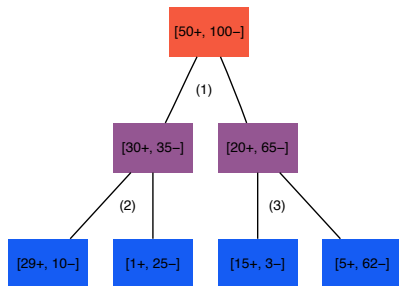
Consider the tree in [Figure 5.4 \(left\)](#). Each node is labelled with the numbers of positive and negative examples covered by it: so, for instance, the root of the tree is labelled with the overall class distribution (50 positives and 100 negatives), resulting in the trivial ranking $[50+, 100-]$.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

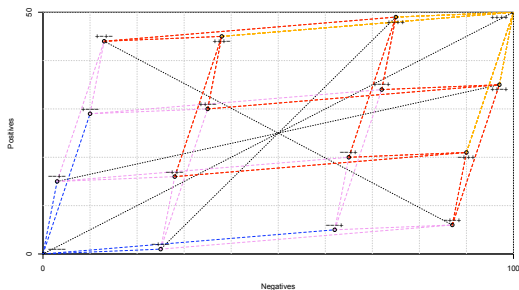


(left) Abstract representation of a tree with numbers of positive and negative examples covered in each node. Binary splits are added to the tree in the order indicated. **(right)** Adding a split to the tree will add new segments to the coverage curve as indicated by the arrows. After a split is added 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

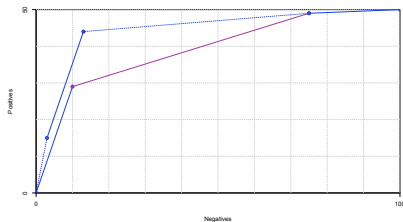
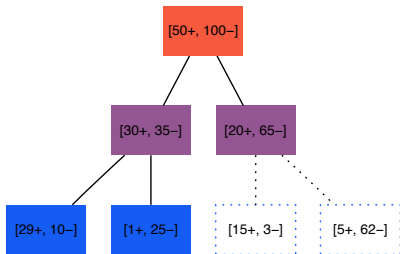
Assume the training set class ratio $clr = 50/100$ is representative. We have a choice of five labellings, depending on the expected cost ratio $c = c_{FN}/c_{FP}$ of misclassifying a positive in proportion to the cost of misclassifying a negative:

- + - + - 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.



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   |  $D_N \leftarrow \{x \in D \mid x \text{ is covered by } N\}$ ;
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  $T$ 

```



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?



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.

Relative impurity

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: $\frac{n_1}{n} \sqrt{\frac{n_1^{\oplus}}{n_1} \frac{n_1^{\ominus}}{n_1}}$;

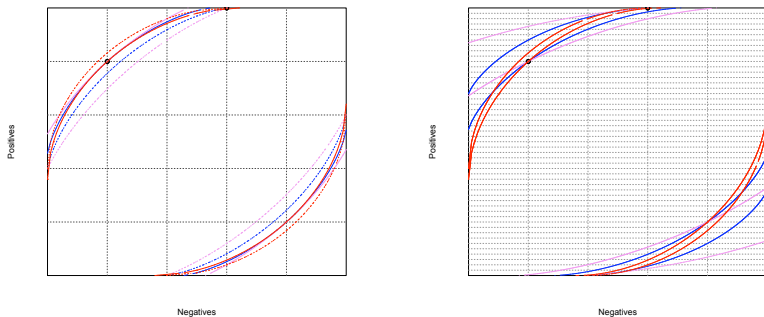
✎ relative impurity: $\sqrt{\frac{n_1^{\oplus} n_1^{\ominus}}{n^{\oplus} n^{\ominus}}}$.

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 **$\sqrt{\text{Gini}}$ in red** through the splits $[8+, 2-][2+, 8-]$ (solid lines) and $[10+, 6-][0+, 4-]$ (dotted lines). Only $\sqrt{\text{Gini}}$ 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 $\sqrt{\text{Gini}}$ 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 $\sqrt{\text{Gini}}$; 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?

5

Tree models

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

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

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

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

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:

Model = [A100, B3, E112, M102, T202]

[1051, 1770, 1900][4513][77][870][99, 270, 625]

Condition = [excellent, good, fair]

[1770, 4513][270, 870, 1051, 1900][77, 99, 625]

Leslie = [yes, no] [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:

Condition = [excellent, good, fair]	[1770][1051, 1900][]
Leslie = [yes, 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:

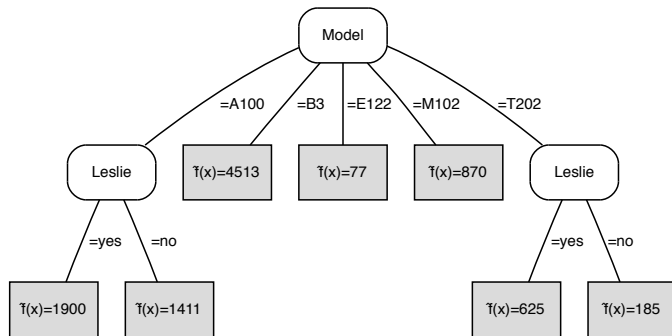
Condition = [excellent, good, fair]	[][270][99, 625]
Leslie = [yes, no]	[625][99, 270]

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](#).



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:

0	11	6	13	10	3	13	3	12
11	0	1	1	1	3	0	4	0
6	1	0	2	1	1	2	2	1
13	1	2	0	0	4	0	4	0
10	1	1	0	0	3	0	2	0
3	3	1	4	3	0	4	1	3
13	0	2	0	0	4	0	4	0
3	4	2	4	2	1	4	0	4
12	0	1	0	0	3	0	4	0

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):

Model = [A100, B3, E112, M102, T202] [3, 6, 8][1][9][5][2, 4, 7]

Condition = [excellent, good, fair] [1, 6][3, 4, 5, 8][2, 7, 9]

Leslie = [yes, 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



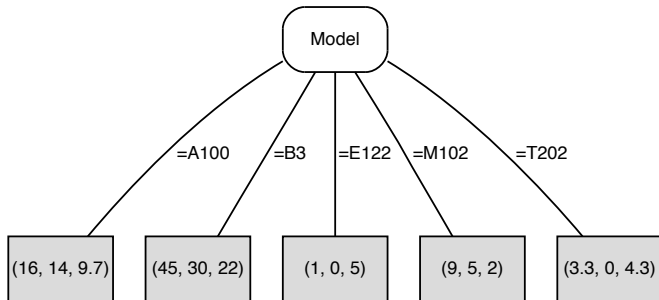
Clustering with Euclidean distance II

- ✎ 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?

6 Rule models

- Learning ordered rule lists
 - Rule lists for ranking and probability estimation
- Learning unordered rule sets
 - Rule sets for ranking and probability estimation
 - A closer look at rule overlap ★
- Descriptive rule learning
 - Rule learning for subgroup discovery
 - Association rule mining

What's next?

6 Rule models

- Learning ordered rule lists
 - Rule lists for ranking and probability estimation
- Learning unordered rule sets
 - Rule sets for ranking and probability estimation
 - A closer look at rule overlap ★
- Descriptive rule learning
 - Rule learning for subgroup discovery
 - Association rule mining



Example 6.1, p.159

Learning a rule list I

Consider again our small dolphins data set with positive examples

p1: Length = 3 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

p2: Length = 4 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

p3: Length = 3 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few

p4: Length = 5 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

p5: Length = 5 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few

and negatives

n1: Length = 5 \wedge Gills = yes \wedge Beak = yes \wedge Teeth = many

n2: Length = 4 \wedge Gills = yes \wedge Beak = yes \wedge Teeth = many

n3: Length = 5 \wedge Gills = yes \wedge Beak = no \wedge Teeth = many

n4: Length = 4 \wedge Gills = yes \wedge Beak = no \wedge Teeth = many

n5: Length = 4 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few



Example 6.1, p.159

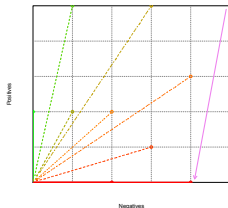
Learning a rule list II

- 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

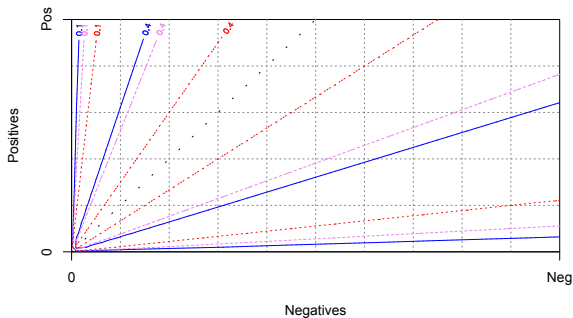


(left) All literals with their coverage counts on the data in [Example 6.1](#). The ones in **green** (**red**) are pure for the positive (negative) class. **(right)** The nine literals plotted as points in coverage space, with their impurity values indicated by impurity isometrics (away from the ascending diagonal is better). Impurity values are colour-coded: towards **green** if $\hat{p} > 1/2$, towards **red** if $\hat{p} < 1/2$, and **orange** if $\hat{p} = 1/2$ (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

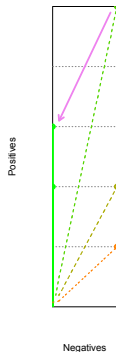
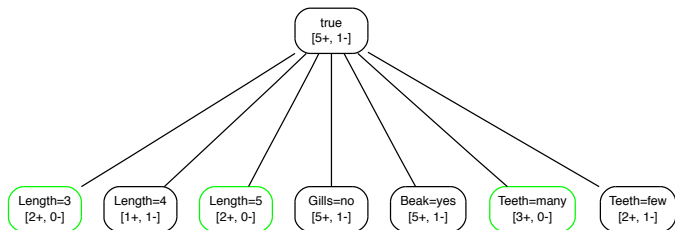


ROC isometrics for **entropy** (rescaled to have a maximum value of $1/2$), **Gini index** and **minority class**. The grey dotted symmetry line is defined by $\hat{p} = 1/2$: each isometric has two parts, one above the symmetry line (where impurity decreases with increasing empirical probability \hat{p})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



Figure 6.3, p.161

Constructing the second rule

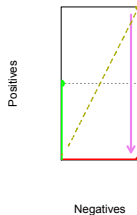
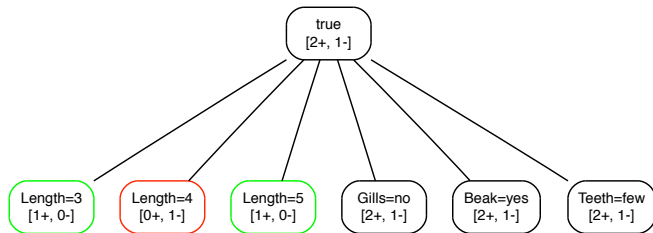


(left) Revised coverage counts after removing the four negative examples covered by the first rule found (literals not covering any examples are omitted). **(right)** We are now operating in the right-most 'slice' of Figure 6.2.



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.



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    $D \leftarrow D \setminus \{x \in D \mid x \text{ is covered by } r\}$ ;  
6 end  
7 return  $R$ 
```



Algorithm LearnRule(D) – learn a single rule.

Input : labelled training data D .

Output : rule r .

```

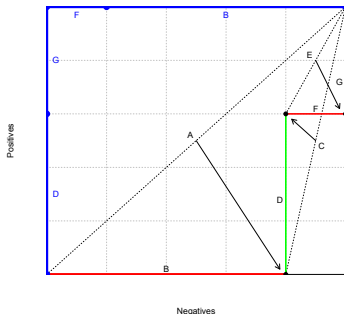
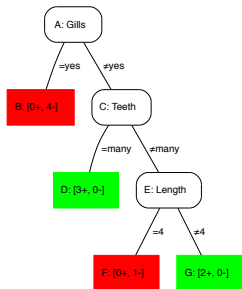
1  $b \leftarrow \text{true}$ ;
2  $L \leftarrow$  set of available literals;
3 while not Homogeneous( $D$ ) do
4    $l \leftarrow \text{BestLiteral}(D, L)$ ; // e.g., highest purity; see text
5    $b \leftarrow b \wedge l$ ;
6    $D \leftarrow \{x \in D \mid x \text{ is covered by } b\}$ ;
7    $L \leftarrow L \setminus \{l' \in L \mid l' \text{ uses same feature as } l\}$ ;
8 end
9  $C \leftarrow \text{Label}(D)$ ; // e.g., majority class
0  $r \leftarrow \cdot \text{if } b \text{ then Class} = C \cdot$ ;
1 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.

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 = \ominus · [1+, 3-]
- else if Beak = yes then Class = \oplus · [4+, 1-]
- else Class = \ominus · [0+, 1-]

The coverage curve of this rule list is given in [Figure 6.6](#).



- 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 $B \setminus A$.
- 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.



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

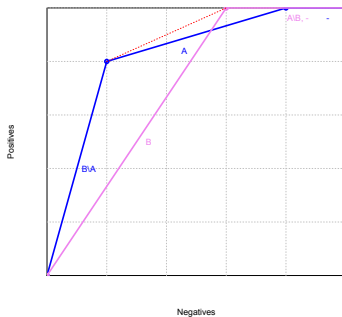
- if Beak = yes then Class = \oplus · [5+, 3-]
- else if Length = 4 then Class = \ominus · [0+, 1-]
- else Class = \ominus · [0+, 1-]

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: $A \setminus B$) 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). $B \setminus A$ 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 $A \wedge B$, 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?

6 Rule models

- Learning ordered rule lists
 - Rule lists for ranking and probability estimation
- **Learning unordered rule sets**
 - Rule sets for ranking and probability estimation
 - A closer look at rule overlap ★
- Descriptive rule learning
 - Rule learning for subgroup discovery
 - Association rule mining

Learning a rule set for class \oplus

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

·if Length = 3 then Class = \oplus ·

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:

·if Gills = no \wedge Length = 5 then Class = \oplus ·

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

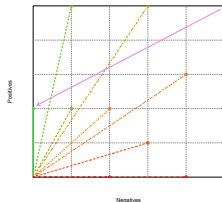
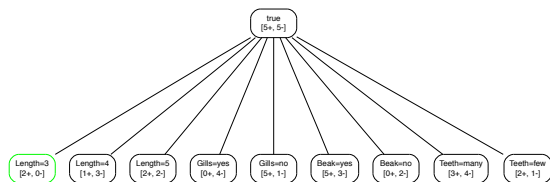
·if Gills = no \wedge Teeth = many then Class = \oplus ·

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

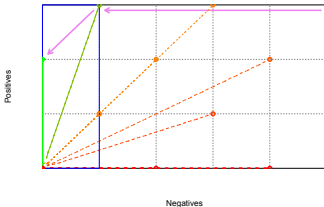
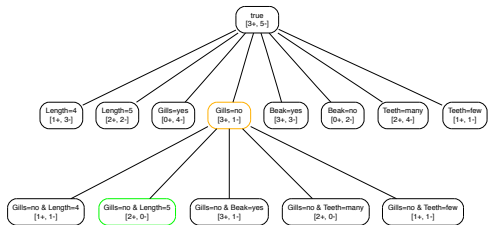


(left) The first rule is learned for the positive class. **(right)** Precision isometrics look identical to impurity isometrics (Figure 6.2); however, the difference is that precision is lowest on the x -axis and highest on the y -axis, while purity is lowest on the ascending diagonal and highest on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



Figure 6.8, p.169

Learning the second rule



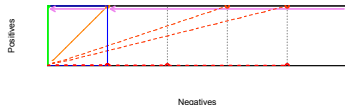
(left) The second rule needs two literals: we use maximum precision to select both.

(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          $r \leftarrow \text{LearnRuleForClass}(D_i, C_i)$ ; // LearnRuleForClass: see Algorithm
6          $R \leftarrow R \cup \{r\}$ ;
7          $D_i \leftarrow D_i \setminus \{x \in C_i \mid x \text{ is covered by } r\}$ ; // remove only positives
8     end
9 end
0 return  $R$ 
  
```



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 .

```

1  $b \leftarrow \text{true}$ ;
2  $L \leftarrow$  set of available literals ;           // can be initialised by seed example
3 while not Homogeneous( $D$ ) do
4    $l \leftarrow$  BestLiteral( $D, L, C_i$ ) ;       // e.g. maximising precision on class  $C_i$ 
5    $b \leftarrow b \wedge l$ ;
6    $D \leftarrow \{x \in D \mid x \text{ is covered by } b\}$ ;
7    $L \leftarrow L \setminus \{l' \in L \mid l' \text{ uses same feature as } l\}$ ;
8 end
9  $r \leftarrow \cdot \text{if } b \text{ then Class} = C_i \cdot$ ;
0 return  $r$ 

```

The need for probability smoothing

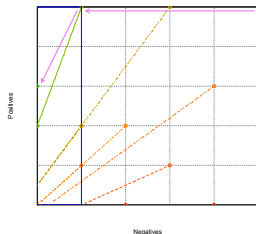
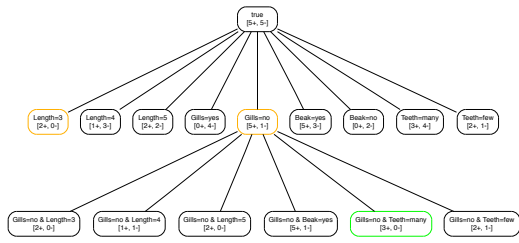
One issue with using precision as search heuristic is that it tends to focus a bit too much on finding pure rules, thereby occasionally missing near-pure rules that can be specialised into a more general pure rule.

- 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.



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

- (A) **·if** Length = 4 **then** Class = \ominus [1+, 3-]
- (B) **·if** Beak = yes **then** Class = \oplus [5+, 3-]
- (C) **·if** Length = 5 **then** Class = \ominus [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 $\hat{p}(\mathbf{A}) = 1/4 = 0.25$, and similarly $\hat{p}(\mathbf{B}) = 5/8 = 0.63$ and $\hat{p}(\mathbf{C}) = 2/4 = 0.50$.
- ☞ 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 $\hat{p}(AB) = 3/6 = 0.50$. Similarly, $\hat{p}(BC) = 3.5/6 = 0.58$.
- 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**:

```

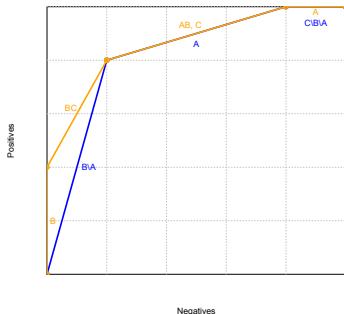
·if Length = 4 then Class = ⊖·      [1+, 3–]
·else if Beak = yes then Class = ⊕·  [4+, 1–]
·else if Length = 5 then Class = ⊖·  [0+, 1–]
  
```

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.



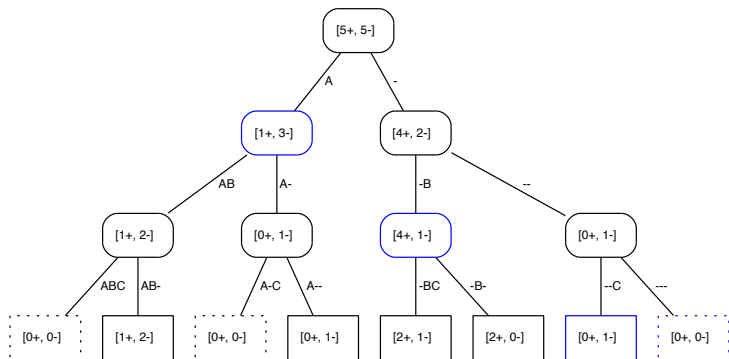
From the rules in [Example 6.4](#) we can construct the rule tree in [Figure 6.12](#). The use of a tree rather than a list allows further splitting of the segments of the rule list.

- ➡ For example, the node labelled **A** is further split into **AB** ($A \wedge B$) and **A-** ($A \wedge \neg B$).
- ➡ As the latter is pure, we obtain a better coverage curve (the **red** line in [Figure 6.13](#)).



Figure 6.12, p.175

★ Rule tree

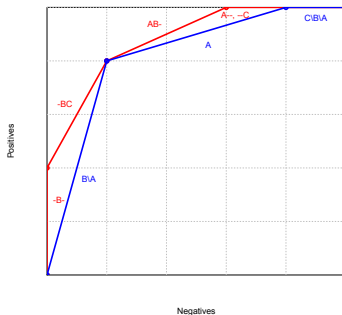


A rule tree constructed from the rules in [Example 6.5](#). Nodes are labelled with their coverage (dotted leaves have empty coverage), and branch labels indicate particular areas in the instance space (e.g., $A-C$ denotes $A \wedge \neg B \wedge C$). The blue nodes are the instance space segments corresponding to the rule list ABC : the rule tree has better performance because it is able to split them further.



Figure 6.13, p.176

★ Rule tree dominates rule list and rule set

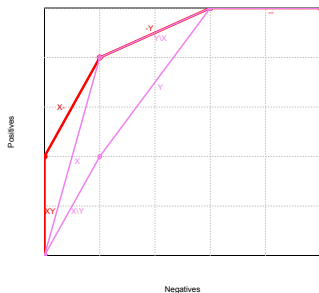
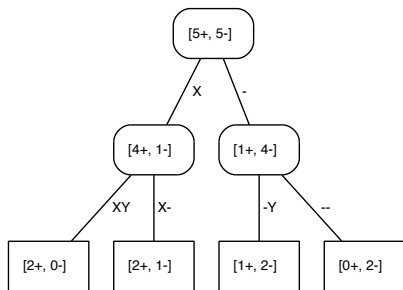


The blue line is the coverage curve of the rule list ABC in Example 6.4. This curve is dominated by the red coverage curve, corresponding to the rule tree in Figure 6.12. The rule tree also improves upon the rule set (orange curve in Figure 6.11), as it has access to exact coverage counts in all segments and thus recognises that $AB-$ goes before $--C$.



Figure 6.14, p.177

★ Rule lists cannot reach some points



(left) A rule tree built on two rules X and Y . **(right)** The rule tree coverage curve strictly dominates the convex hull of the two rule list curves. This means that there is an operating point $[2+, 0-]$ that cannot be achieved by either rule list.

What's next?

6 Rule models

- Learning ordered rule lists
 - Rule lists for ranking and probability estimation
- Learning unordered rule sets
 - Rule sets for ranking and probability estimation
 - A closer look at rule overlap ★
- **Descriptive rule learning**
 - Rule learning for subgroup discovery
 - Association rule mining

Subgroup discovery

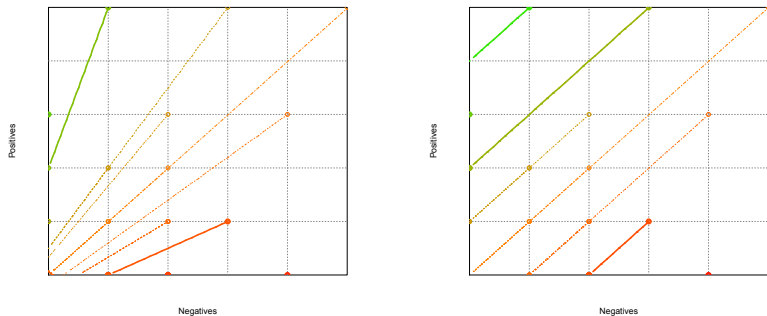
Subgroups are subsets of the instance space – or alternatively, mappings $\hat{g} : \mathcal{X} \rightarrow \{\text{true}, \text{false}\}$ – that are learned from a set of labelled examples $(x_i, l(x_i))$, where $l : \mathcal{X} \rightarrow \mathcal{C}$ is the true labelling function.

- ✎ 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 **Gills = yes**, which covers four negatives and no positives, could be considered as interesting as its complement **Gills = 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 $[0+, 4-]$.



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

<i>Subgroup</i>	<i>Coverage</i>	$prec^L$	<i>Rank</i>	<i>avg-rec</i>	<i>Rank</i>
Gills = yes	[0+,4-]	0.17	1	0.10	1-2
Gills = no \wedge 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 \wedge 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 \wedge Gills = yes	[0+,2-]	0.25	3-9	0.30	4-11
Length = 5 \wedge Gills = no	[2+,0-]	0.75	3-9	0.70	4-11
Length = 5 \wedge 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 $|prec^L - pos|$. Alternatively, we can use average recall to define the quality of a subgroup as $|avg-rec - 0.5|$.



Algorithm `WeightedCovering(D)` – learn overlapping rules by weighting examples.

Input : labelled training data D with instance weights initialised to 1.

Output : rule list R .

```

1  $R \leftarrow \emptyset$ ;
2 while some examples in  $D$  have weight 1 do
3    $r \leftarrow \text{LearnRule}(D)$  ; // LearnRule: see Algorithm 6.2
4   append  $r$  to the end of  $R$ ;
5   decrease the weights of examples covered by  $r$ ;
6 end
7 return  $R$ 

```



Example 6.7, p.180

★ The effect of weighted covering

Suppose the first subgroup found is **Length = 4**, reducing the weight of the one positive and three negatives covered by it to 1/2. Detailed calculations of how this affects the weighted coverage of subgroups are given in [Table 6.2](#).

- ☞ We can see how the coverage space shrinks to the **blue box** in [Figure 6.16](#). It also affects the weighted coverage of the subgroups overlapping with **Length = 4**, as indicated by the arrows.
- ☞ Some subgroups end up closer to the diagonal and hence lose importance: for instance, **Length = 4** itself, which moves from [3+, 1-] to [1.5+, 0.5-].
- ☞ Others move away from the diagonal and hence gain importance: for example **Length = 5 \wedge Gills = yes** at [0+, 2-].



Table 6.2, p.180

★ The effect of weighted covering

<i>Subgroup</i>	<i>Coverage</i>	<i>avg-rec</i>	<i>Wgtd coverage</i>	<i>W-avg-rec</i>	<i>Rank</i>
Gills = yes	[0+, 4-]	0.10	[0+, 3-]	0.07	1-2
Gills = no	[5+, 1-]	0.90	[4.5+ , 0.5-]	0.93	1-2
Gills = no \wedge Teeth = many	[3+, 0-]	0.80	[2.5+ , 0-]	0.78	3
Length = 5 \wedge Gills = yes	[0+, 2-]	0.30	[0+, 2-]	0.21	4
Length = 3	[2+, 0-]	0.70	[2+, 0-]	0.72	5-6
Length = 5 \wedge Gills = no	[2+, 0-]	0.70	[2+, 0-]	0.72	5-6
Beak = no	[0+, 2-]	0.30	[0+, 1.5-]	0.29	7-9
Gills = yes \wedge Beak = yes	[0+, 2-]	0.30	[0+, 1.5-]	0.29	7-9
Beak = yes	[5+, 3-]	0.70	[4.5+ , 2-]	0.71	7-9
Length = 4	[1+, 3-]	0.30	[0.5+ , 1.5-]	0.34	10
Length = 4 \wedge Gills = yes	[0+, 2-]	0.30	[0+, 1-]	0.36	11

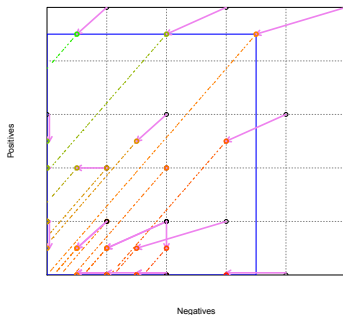
The 'Wgtd coverage' column shows how the weighted coverage of the subgroups is affected if the weights of the examples covered by **Length = 4** are reduced to 1/2.

'W-avg-rec' shows how the *avg-rec* numbers as calculated in [Table 6.1](#) are affected by the weighting, leading to further differentiation between subgroups that were previously considered equivalent.



Figure 6.16, p.181

★ The effect of weighted covering



If the first subgroup found is $\text{Length} = 4$, then this halves the weight of one positive and three negatives, shrinking the coverage space to the blue box. The arrows indicate how this affects the weighted coverage of other subgroups, depending on which of the reduced-weight examples they cover.

Items and transactions

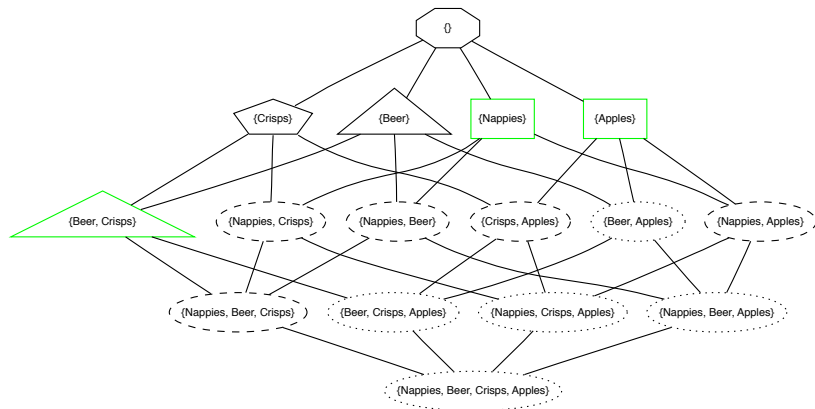
<i>Transaction</i>	<i>Items</i>
1	nappies
2	beer, crisps
3	apples, nappies
4	beer, crisps, nappies
5	apples
6	apples, beer, crisps, nappies
7	apples, crisps
8	crisps

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 {beer, 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 $\text{FrequentItems}(D, f_0)$ – find all maximal item sets exceeding a given support threshold.

Input : data $D \subseteq \mathcal{X}$; support threshold f_0 .

Output : set of maximal frequent item sets M .

```

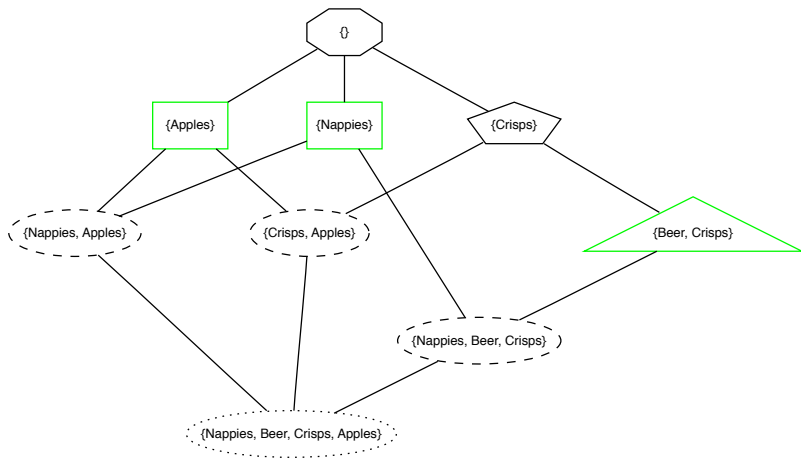
1  $M \leftarrow \emptyset$ ;
2 initialise priority queue  $Q$  to contain the empty item set;
3 while  $Q$  is not empty do
4    $I \leftarrow$  next item set deleted from front of  $Q$ ;
5    $max \leftarrow \text{true}$ ; // flag to indicate whether  $I$  is maximal
6   for each possible extension  $I'$  of  $I$  do
7     if  $\text{Supp}(I') \geq f_0$  then
8        $max \leftarrow \text{false}$ ; // frequent extension found, so  $I$  is not maximal
9       add  $I'$  to back of  $Q$ ;
10    end
11  end
12  if  $max = \text{true}$  then  $M \leftarrow M \cup \{I\}$ ;
13 end
14 return  $M$ 

```



Figure 6.18, p.185

★ Closed item sets



Closed item set lattice corresponding to the item sets in [Figure 6.17](#). This lattice has the property that no two adjacent item sets have the same coverage.

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 $\{\text{beer}\}$ and $\{\text{nappies, beer}\}$. 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 $\{\text{nappies}\}$ and $\{\text{nappies, beer}\}$ 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 \wedge 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 $\text{AssociationRules}(D, f_0, c_0)$ – find all association rules exceeding given support and confidence thresholds.

Input : data $D \subseteq \mathcal{X}$; support threshold f_0 ; confidence threshold c_0 .

Output : set of association rules R .

```

1  $R \leftarrow \emptyset$ ;
2  $M \leftarrow \text{FrequentItems}(D, f_0)$ ;           // 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  $R \leftarrow R \cup \{\cdot \text{if } B \text{ then } H \cdot\}$ ;
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

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

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 $\text{Lift}(\cdot\text{if } B \text{ then } H\cdot) = \text{Lift}(\cdot\text{if } H \text{ then } B\cdot)$.
- ☞ For the third rule we have $\text{Lift}(\cdot\text{if true then crisps}\cdot) = \frac{8 \cdot 5}{8 \cdot 5} = 1$. This holds for any rule with $B = \emptyset$, as

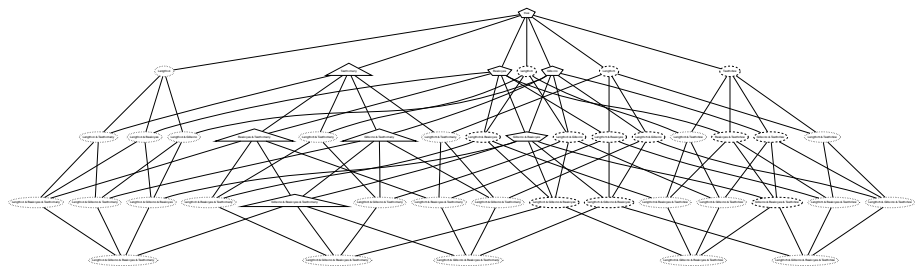
$$\text{Lift}(\cdot\text{if } \emptyset \text{ then } H\cdot) = \frac{n \cdot \text{Supp}(\emptyset \cup H)}{\text{Supp}(\emptyset) \cdot \text{Supp}(H)} = \frac{n \cdot \text{Supp}(H)}{n \cdot \text{Supp}(H)} = 1$$

More generally, a lift of 1 means that $\text{Supp}(B \cup H)$ is entirely determined by the *marginal* frequencies $\text{Supp}(B)$ and $\text{Supp}(H)$ 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



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](#).



Figure 6.20, p.188

★ Closed item sets and dolphins



Closed item set lattice corresponding to the item sets in [Figure 6.19](#).

What's next?

7 Linear models

- The least-squares method
 - Multivariate linear regression
 - Regularised regression ★
 - Using least-squares regression for classification ★
- The perceptron: a heuristic learning algorithm for linear classifiers
- Support vector machines
 - Soft margin SVM
- Obtaining probabilities from linear classifiers
- Going beyond linearity with kernel methods ★

What's next?

7 Linear models

- The least-squares method

- Multivariate linear regression
- Regularised regression ★
- Using least-squares regression for classification ★

- The perceptron: a heuristic learning algorithm for linear classifiers

- Support vector machines

- Soft margin SVM

- Obtaining probabilities from linear classifiers

- Going beyond linearity with kernel methods ★



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 :

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

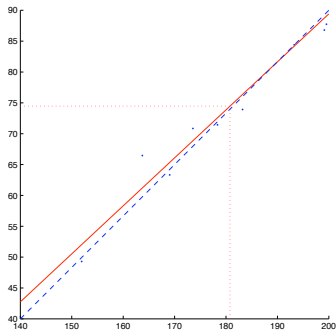
$$\frac{\partial}{\partial b} \sum_{i=1}^n (w_i - (a + bh_i))^2 = -2 \sum_{i=1}^n (w_i - (a + bh_i))h_i = 0 \quad \Rightarrow \hat{b} = \frac{\sum_{i=1}^n (h_i - \bar{h})(w_i - \bar{w})}{\sum_{i=1}^n (h_i - \bar{h})^2}$$

So the solution found by linear regression is $w = \hat{a} + \hat{b}h = \bar{w} + \hat{b}(h - \bar{h})$; 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 $\bar{h} = 181$ and the average weight $\bar{w} = 74.5$; 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 (\bar{x}, \bar{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 \bar{x} , in which case the intercept is equal to \bar{y} .

We could even subtract \bar{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 \bar{x} with $\bar{x}' = \bar{x} / \sigma_{xx}$, then we have that $\hat{b} = \frac{1}{n} \sum_{i=1}^n (x'_i - \bar{x}') (y_i - \bar{y}) = \sigma_{x'y}$.

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:

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

The result follows because $\hat{a} = \bar{y} - \hat{b}\bar{x}$, 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.

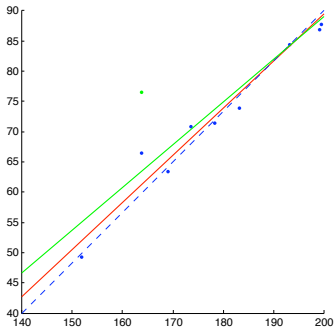


Suppose that, as the result of a transcription error, one of the weight values in [Figure 7.1](#) is increased by 10 kg. [Figure 7.2](#) shows that this has a considerable effect on the least-squares regression line.



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 I

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

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

Assuming for the moment that every feature is zero-centred, we have $\mu_j = 0$ and thus $\mathbf{X}^T \mathbf{y}$ 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

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

Multivariate linear regression II

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

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

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

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

In the general case we cannot make any assumptions about the features, and $(\mathbf{X}^T \mathbf{X})^{-1}$ 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.

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

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

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

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



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 $1 \leq i \leq n$; $\bar{x}_1 = 1$; 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}):

$$(\mathbf{X}^T \mathbf{X})^{-1} = \frac{1}{n\sigma_{xx}} \begin{pmatrix} \sigma_{xx} + \bar{x}^2 & -\bar{x} \\ -\bar{x} & 1 \end{pmatrix}$$

$$\mathbf{X}^T \mathbf{y} = n \begin{pmatrix} \bar{y} \\ \sigma_{xy} + \bar{x} \bar{y} \end{pmatrix}$$

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \frac{1}{\sigma_{xx}} \begin{pmatrix} \sigma_{xx} \bar{y} - \sigma_{xy} \bar{x} \\ \sigma_{xy} \end{pmatrix}$$

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

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

$$\mathbf{X}^T \mathbf{y} = n \begin{pmatrix} \sigma_{1y} \\ \sigma_{2y} \end{pmatrix}$$

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

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 ($\sigma_{12} \neq 0$).

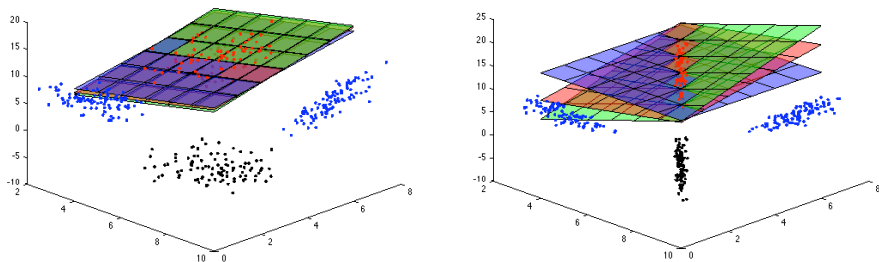
Important point to remember

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.

★ Regularised regression I

Regularisation is a general method to avoid overfitting by applying additional constraints to the weight vector. A common approach is to make sure the weights are, on average, small in magnitude: this is referred to as *shrinkage*.

The least-squares regression problem can be written as an optimisation problem:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

The regularised version of this optimisation is then as follows:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \|\mathbf{w}\|^2$$

where $\|\mathbf{w}\|^2 = \sum_i w_i^2$ is the squared norm of the vector \mathbf{w} , or, equivalently, the dot product $\mathbf{w}^T \mathbf{w}$; λ is a scalar determining the amount of regularisation.

★ Regularised regression II

This regularised problem still has a closed-form solution:

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

where \mathbf{I} denotes the identity matrix. Regularisation amounts to adding λ to the diagonal of $\mathbf{X}^T \mathbf{X}$, a well-known trick to improve the numerical stability of matrix inversion. This form of least-squares regression is known as *ridge regression*.

An interesting alternative form of regularised regression is provided by the *lasso*, which stands for ‘least absolute shrinkage and selection operator’. It replaces the ridge regularisation term $\sum_i w_i^2$ with the sum of absolute weights $\sum_i |w_i|$. The result is that some weights are shrunk, but others are set to 0, and so the lasso regression favours *sparse solutions*.



Example 7.4, p.205

★ Univariate least-squares classifier I

We can use linear regression to learn a binary classifier by encoding the two classes as real numbers.

☞ With $y^{\oplus} = +1$ and $y^{\ominus} = -1$ we have $\mathbf{X}^T \mathbf{y} = Pos \mu^{\oplus} - Neg \mu^{\ominus}$, where μ^{\oplus} and μ^{\ominus} are d -vectors containing each feature's mean values for the positive and negative examples, respectively.

☞ In the univariate case we can rewrite the covariance between x and y as $\sigma_{xy} = 2pos \cdot neg (\mu^{\oplus} - \mu^{\ominus})$, and so the slope of the regression line is

$$\hat{b} = 2pos \cdot neg \frac{\mu^{\oplus} - \mu^{\ominus}}{\sigma_{xx}}$$

☞ This equation shows that the slope of the regression line increases with the separation between the classes (measured as the distance between the class means in proportion to the feature's variance), but also decreases if the class distribution becomes skewed.



Example 7.4, p.205

★ Univariate least-squares classifier II

The regression equation $y = \bar{y} + \hat{b}(x - \bar{x})$ can then be used to obtain a decision boundary.

- ☞ We need to determine the point (x_0, y_0) such that y_0 is half-way between y^{\oplus} and y^{\ominus} (i.e., $y_0 = 0$ in our case). We then have

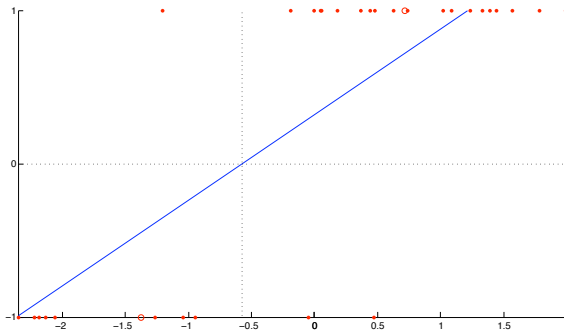
$$x_0 = \bar{x} + \frac{y_0 - \bar{y}}{\hat{b}} = \bar{x} - \frac{\text{pos} - \text{neg}}{2\text{pos} \cdot \text{neg}} \frac{\sigma_{xx}}{\mu^{\oplus} - \mu^{\ominus}}$$

- ☞ That is, if there are equal numbers of positive and negative examples we simply threshold the feature at the feature mean \bar{x} ; in case of unequal class distribution we shift this threshold to the left or right as appropriate (Figure 7.4).



Figure 7.4, p.206

★ Univariate least-squares classifier



Using univariate linear regression to obtain a decision boundary. The 10 negative examples are labelled with $y^{\ominus} = -1$ and the 20 positive examples are labelled $y^{\oplus} = +1$. μ^{\ominus} and μ^{\oplus} are indicated by red circles. The blue line is the linear regression line $y = \bar{y} + \hat{b}(x - \bar{x})$, and the crosshair indicates the decision boundary $x_0 = \bar{x} - \bar{y}/\hat{b}$. This results in three examples being misclassified – notice that this is the best that can be achieved with the given data.

★ Least-squares classifier


In the general case, the *least-squares classifier* learns the decision boundary $\mathbf{w} \cdot \mathbf{x} = t$ with

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} (\text{Pos } \mu^{\oplus} - \text{Neg } \mu^{\ominus})$$

We would hence assign class $\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x} - t)$ to instance \mathbf{x} , where

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Various simplifying assumptions can be made, including zero-centred features, equal-variance features, uncorrelated features and equal class prevalences.

When all these assumptions are made, [Equation 7.7](#) reduces to $\mathbf{w} = c(\mu^{\oplus} - \mu^{\ominus})$ where c is some scalar that can be incorporated in the decision threshold t . We recognise this as the  *basic linear classifier* that was introduced in [the Prologue](#).

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?

7 Linear models

- The least-squares method
 - Multivariate linear regression
 - Regularised regression ★
 - Using least-squares regression for classification ★
- **The perceptron: a heuristic learning algorithm for linear classifiers**
- Support vector machines
 - Soft margin SVM
- Obtaining probabilities from linear classifiers
- Going beyond linearity with kernel methods ★

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 $\text{Perceptron}(D, \eta)$ – train a perceptron for linear classification.

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

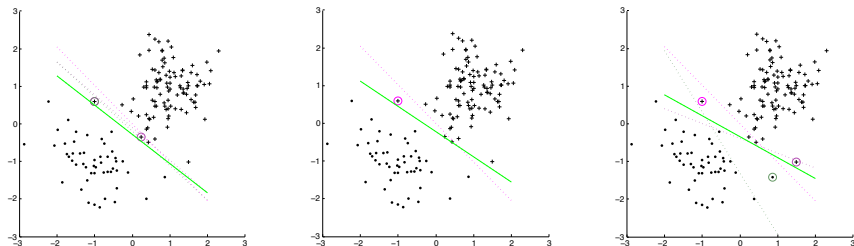
```

1  $\mathbf{w} \leftarrow \mathbf{0}$ ; // Other initialisations of the weight vector are possible
2  $converged \leftarrow false$ ;
3 while  $converged = false$  do
4    $converged \leftarrow true$ ;
5   for  $i = 1$  to  $|D|$  do
6     if  $y_i \mathbf{w} \cdot \mathbf{x}_i \leq 0$  // i.e.,  $\hat{y}_i \neq y_i$ 
7     then
8        $\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$ ;
9        $converged \leftarrow false$ ; // We changed  $\mathbf{w}$  so haven't converged yet
10    end
11  end
12 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

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

- During training, the only information needed about the training data is all pairwise dot products: the n -by- n matrix $\mathbf{G} = \mathbf{X}\mathbf{X}^T$ 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 $\mathbf{w} = \sum_{i=1}^{|D|} \alpha_i y_i \mathbf{x}_i$.

```

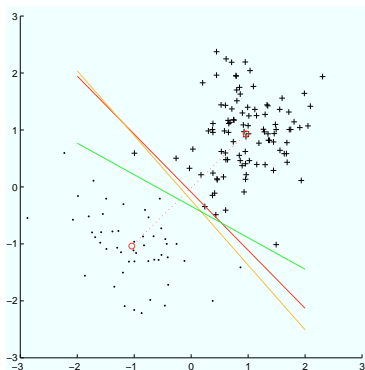
1  $\alpha_i \leftarrow 0$  for  $1 \leq i \leq |D|$ ;
2 converged  $\leftarrow$  false;
3 while converged = false do
4     converged  $\leftarrow$  true;
5     for  $i = 1$  to  $|D|$  do
6         if  $y_i \sum_{j=1}^{|D|} \alpha_j y_j \mathbf{x}_i \cdot \mathbf{x}_j \leq 0$  then
7              $\alpha_i \leftarrow \alpha_i + 1$ ;
8             converged  $\leftarrow$  false;
9         end
10    end
11 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.



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?

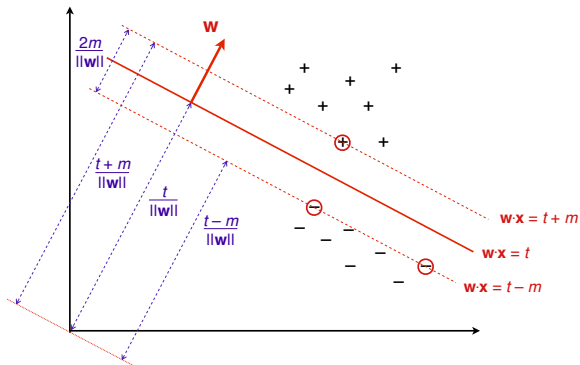
7 Linear models

- The least-squares method
 - Multivariate linear regression
 - Regularised regression ★
 - Using least-squares regression for classification ★
- The perceptron: a heuristic learning algorithm for linear classifiers
- **Support vector machines**
 - Soft margin SVM
- Obtaining probabilities from linear classifiers
- Going beyond linearity with kernel methods ★



Figure 7.7, p.212

Support vector machine



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:

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

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

★ Deriving the dual problem I

Adding the constraints with multipliers α_i for each training example gives the Lagrange function

$$\begin{aligned}
 \Lambda(\mathbf{w}, t, \alpha_1, \dots, \alpha_n) &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i (\mathbf{w} \cdot \mathbf{x}_i - t) - 1) \\
 &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i y_i (\mathbf{w} \cdot \mathbf{x}_i) + \sum_{i=1}^n \alpha_i y_i t + \sum_{i=1}^n \alpha_i \\
 &= \frac{1}{2} \mathbf{w} \cdot \mathbf{w} - \mathbf{w} \cdot \left(\sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \right) + t \left(\sum_{i=1}^n \alpha_i y_i \right) + \sum_{i=1}^n \alpha_i
 \end{aligned}$$

- ☞ By taking the partial derivative of the Lagrange function with respect to t and setting it to 0 we find $\sum_{i=1}^n \alpha_i y_i = 0$.
- ☞ Similarly, by taking the partial derivative of the Lagrange function with respect to \mathbf{w} and setting to 0 we obtain $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$ – the same expression as we derived for the perceptron.

★ Deriving the dual problem II

- For the perceptron, the instance weights α_i are non-negative integers denoting the number of times an example has been misclassified in training. For a support vector machine, the α_i are non-negative reals.
- What they have in common is that, if $\alpha_i = 0$ for a particular example \mathbf{x}_i , that example could be removed from the training set without affecting the learned decision boundary. In the case of support vector machines this means that $\alpha_i > 0$ only for the support vectors: the training examples nearest to the decision boundary.

These expressions allow us to eliminate \mathbf{w} and t and lead to the dual Lagrangian

$$\begin{aligned} \Lambda(\alpha_1, \dots, \alpha_n) &= -\frac{1}{2} \left(\sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \right) + \sum_{i=1}^n \alpha_i \\ &= -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^n \alpha_i \end{aligned}$$

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:

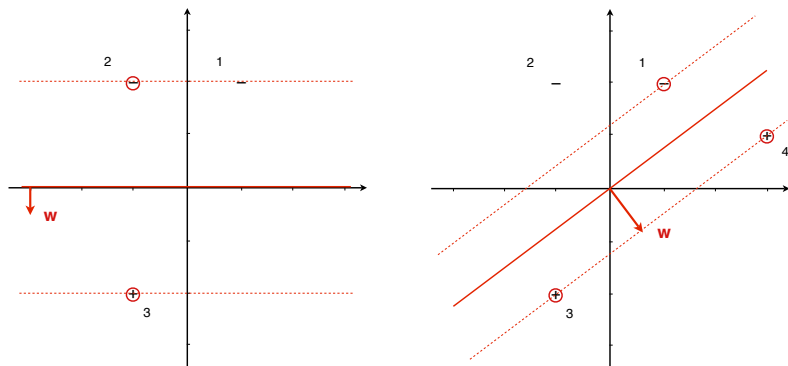
$$\alpha_1^*, \dots, \alpha_n^* = \operatorname{argmax}_{\alpha_1, \dots, \alpha_n} -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^n \alpha_i$$

subject to $\alpha_i \geq 0, 1 \leq i \leq n$ 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

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

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):

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

The dual optimisation problem is thus

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

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

$$\operatorname{argmax}_{\alpha_1, \alpha_2, \alpha_3} -\frac{1}{2} (20\alpha_1^2 + 32\alpha_1\alpha_2 + 16\alpha_2^2) + 2\alpha_1 + 2\alpha_2$$

- 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:

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

- ☞ 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.

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.

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

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

- ☞ 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 :

$$\begin{aligned} \alpha_1^*, \dots, \alpha_n^* &= \arg \max_{\alpha_1, \dots, \alpha_n} -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^n \alpha_i \\ &\text{subject to } 0 \leq \alpha_i \leq C \text{ and } \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

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;

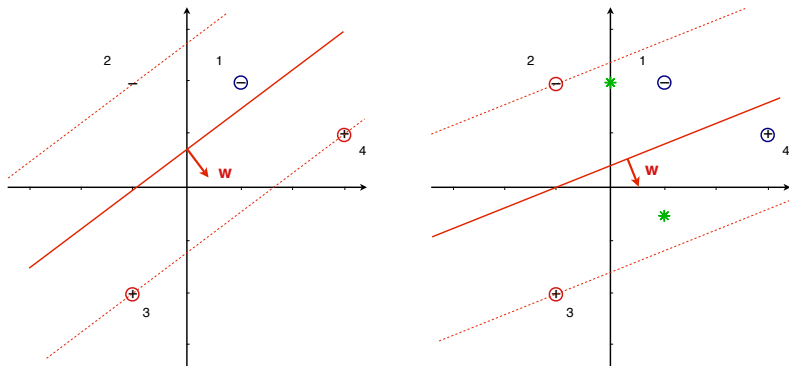
$\alpha_i = C$ these are on or inside the margin.

☞ 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 $\alpha_4 = 1/4$.



- ☞ 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?

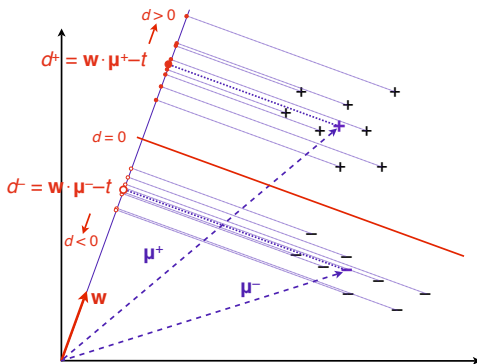
7 Linear models

- The least-squares method
 - Multivariate linear regression
 - Regularised regression ★
 - Using least-squares regression for classification ★
- The perceptron: a heuristic learning algorithm for linear classifiers
- Support vector machines
 - Soft margin SVM
- **Obtaining probabilities from linear classifiers**
- Going beyond linearity with kernel methods ★



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 μ^{\oplus} and μ^{\ominus} , and the corresponding mean distances d^{\oplus} and d^{\ominus} .

★ Deriving the logistic function I

Let \bar{d}^{\oplus} denote the mean distance of the positive examples to the decision boundary: i.e., $\bar{d}^{\oplus} = \mathbf{w} \cdot \boldsymbol{\mu}^{\oplus} - t$, where $\boldsymbol{\mu}^{\oplus}$ is the mean of the positive examples and \mathbf{w} is unit length.

It would not be unreasonable to expect that the distance of positive examples to the decision boundary is normally distributed around this mean. Under this assumption, the probability density function of d is

$$P(d|\oplus) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(d-\bar{d}^{\oplus})^2}{2\sigma^2}\right).$$

Similarly, the distances of negative examples to the decision boundary can be expected to be normally distributed around $\bar{d}^{\ominus} = \mathbf{w} \cdot \boldsymbol{\mu}^{\ominus} - t$, with $\bar{d}^{\ominus} < 0 < \bar{d}^{\oplus}$. We will assume that both normal distributions have the same variance σ^2 .

Suppose we now observe a point \mathbf{x} with distance $d(\mathbf{x})$. We classify this point as positive if $d(\mathbf{x}) > 0$ and as negative if $d(\mathbf{x}) < 0$, but we want to attach a probability $\hat{p}(\mathbf{x}) = P(\oplus|d(\mathbf{x}))$ to these predictions.

★ Deriving the logistic function II

👉 Using Bayes' rule we obtain

$$P(\oplus|d(\mathbf{x})) = \frac{P(d(\mathbf{x})|\oplus)P(\oplus)}{P(d(\mathbf{x})|\oplus)P(\oplus) + P(d(\mathbf{x})|\ominus)P(\ominus)} = \frac{LR}{LR + 1/clr}$$

where LR is the likelihood ratio obtained from the normal score distributions, and clr is the class ratio. We will assume for simplicity that $clr = 1$ in the derivation below.

👉 Furthermore, assume for now that $\sigma^2 = 1$ and $\bar{d}^{\oplus} = -\bar{d}^{\ominus} = 1/2$ (we will relax this in a moment). We then have

$$\begin{aligned} LR &= \frac{P(d(\mathbf{x})|\oplus)}{P(d(\mathbf{x})|\ominus)} = \frac{\exp(-(d(\mathbf{x}) - 1/2)^2/2)}{\exp(-(d(\mathbf{x}) + 1/2)^2/2)} \\ &= \exp(-(d(\mathbf{x}) - 1/2)^2/2 + (d(\mathbf{x}) + 1/2)^2/2) = \exp(d(\mathbf{x})) \end{aligned}$$

and so

$$P(\oplus|d(\mathbf{x})) = \frac{\exp(d(\mathbf{x}))}{\exp(d(\mathbf{x})) + 1} = \frac{\exp(\mathbf{w} \cdot \mathbf{x} - t)}{\exp(\mathbf{w} \cdot \mathbf{x} - t) + 1}$$

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,

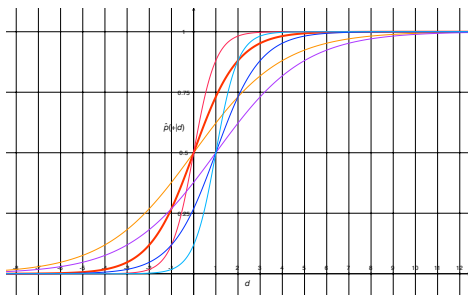
$$d \mapsto \frac{1}{1 + \exp(-d)}$$

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.

★ Logistic calibration: the general case I

Suppose now that $\bar{d}^{\oplus} = -\bar{d}^{\ominus}$ as before, but we do not assume anything about the magnitude of these mean distances or of σ^2 . In this case we have

$$\begin{aligned} LR &= \exp\left(\frac{-(d(\mathbf{x}) - \bar{d}^{\oplus})^2 + (d(\mathbf{x}) - \bar{d}^{\ominus})^2}{2\sigma^2}\right) \\ &= \exp\left(\frac{2\bar{d}^{\oplus} d(\mathbf{x}) - (\bar{d}^{\oplus})^2 - 2\bar{d}^{\ominus} d(\mathbf{x}) + (\bar{d}^{\ominus})^2}{2\sigma^2}\right) = \exp(\gamma d(\mathbf{x})) \end{aligned}$$

with $\gamma = (\bar{d}^{\oplus} - \bar{d}^{\ominus})/\sigma^2$ a scaling factor that rescales the weight vector so that the mean distances per class are one unit of variance apart. In other words, by taking the scaling factor γ into account, we can drop our assumption that \mathbf{w} is a unit vector.

★ Logistic calibration: the general case II

If we also drop the assumption that \bar{d}^{\oplus} and \bar{d}^{\ominus} are symmetric around the decision boundary, then we obtain the most general form

$$LR = \frac{P(d(\mathbf{x})|\oplus)}{P(d(\mathbf{x})|\ominus)} = \exp(\gamma(d(\mathbf{x}) - d_0))$$

$$\gamma = \frac{\bar{d}^{\oplus} - \bar{d}^{\ominus}}{\sigma^2} = \frac{\mathbf{w} \cdot (\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus})}{\sigma^2}, \quad d_0 = \frac{\bar{d}^{\oplus} + \bar{d}^{\ominus}}{2} = \frac{\mathbf{w} \cdot (\boldsymbol{\mu}^{\oplus} + \boldsymbol{\mu}^{\ominus})}{2} - t$$

d_0 has the effect of moving the decision boundary from $\mathbf{w} \cdot \mathbf{x} = t$ to $\mathbf{x} = (\boldsymbol{\mu}^{\oplus} + \boldsymbol{\mu}^{\ominus})/2$, that is, halfway between the two class means. The logistic mapping thus becomes $d \mapsto \frac{1}{1 + \exp(-\gamma(d - d_0))}$.



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

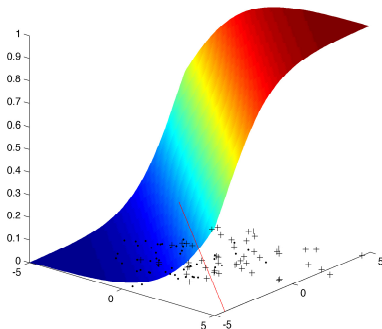
$$\bar{d}^{\oplus} - \bar{d}^{\ominus} = \frac{\mathbf{w} \cdot (\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus})}{\|\mathbf{w}\|} = \frac{\|\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}\|^2}{\|\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}\|} = \|\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}\|$$

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

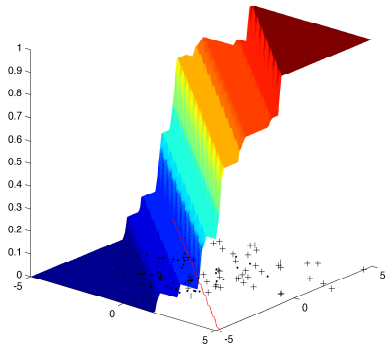
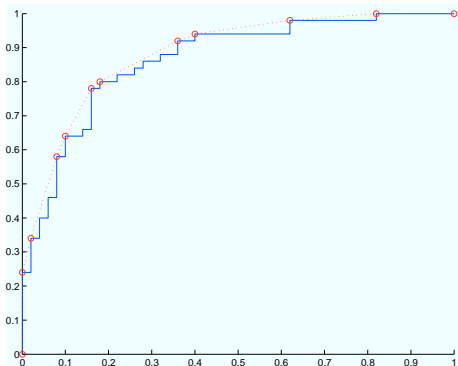


The surface shows the sigmoidal probability estimates resulting from logistic calibration of the basic linear classifier on random data satisfying the assumptions of logistic calibration.



Figure 7.13, p.224

★ Isotonic calibration of a linear classifier



(left) ROC curve and convex hull of the same model and data as in Figure 7.12. **(right)** The convex hull can be used as a non-parametric calibration method. Each segment of the convex hull corresponds to a plateau of the probability surface.

What's next?

7 Linear models

- The least-squares method
 - Multivariate linear regression
 - Regularised regression ★
 - Using least-squares regression for classification ★
- The perceptron: a heuristic learning algorithm for linear classifiers
- Support vector machines
 - Soft margin SVM
- Obtaining probabilities from linear classifiers
- Going beyond linearity with kernel methods ★



Example 7.8, p.225

★ Learning a quadratic decision boundary

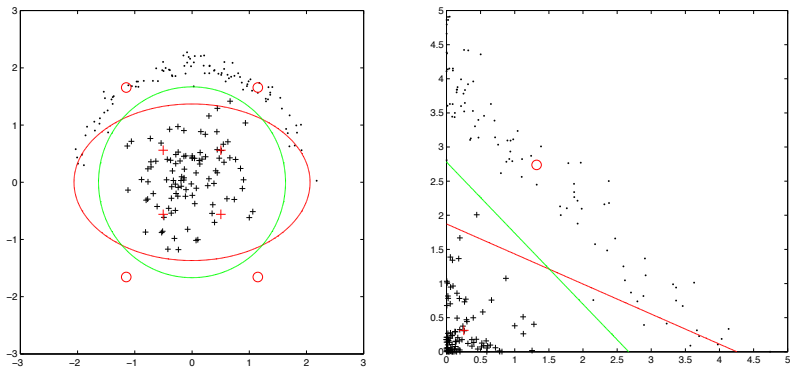
- ☞ The data in [Figure 7.14 \(left\)](#) is not linearly separable, but both classes have a clear circular shape. [Figure 7.14 \(right\)](#) shows the same data with the feature values squared.
- ☞ In this transformed feature space the data has become linearly separable, and the perceptron is able to separate the classes. The resulting decision boundary in the original space is a near-circle.
- ☞ Also shown is the decision boundary learned by the basic linear classifier in the quadratic feature space, corresponding to an ellipse in the original space.

In general, mapping points back from the feature space to the instance space is non-trivial. E.g., in this example each class mean in feature space maps back to four points in the original space, owing to the quadratic mapping.



Figure 7.14, p.225

★ Learning a quadratic decision boundary



(left) Decision boundaries learned by the **basic linear classifier** and the **perceptron** using the square of the features. **(right)** Data and decision boundaries in the transformed feature space.

★ 'Kernelising' the perceptron I

The perceptron algorithm is a simple counting algorithm – the only operation that is somewhat involved is testing whether example \mathbf{x}_i is correctly classified by evaluating $y_i \sum_{j=1}^{|D|} \alpha_j y_j \mathbf{x}_i \cdot \mathbf{x}_j$.

- 👉 The key component of this calculation is the dot product $\mathbf{x}_i \cdot \mathbf{x}_j$.
- 👉 Assuming bivariate examples $\mathbf{x}_i = (x_i, y_i)$ and $\mathbf{x}_j = (x_j, y_j)$ for notational simplicity, the dot product can be written as $\mathbf{x}_i \cdot \mathbf{x}_j = x_i x_j + y_i y_j$.
- 👉 The corresponding instances in the quadratic feature space are (x_i^2, y_i^2) and (x_j^2, y_j^2) , and their dot product is $(x_i^2, y_i^2) \cdot (x_j^2, y_j^2) = x_i^2 x_j^2 + y_i^2 y_j^2$.
- 👉 This is almost equal to $(\mathbf{x}_i \cdot \mathbf{x}_j)^2 = (x_i x_j + y_i y_j)^2 = (x_i x_j)^2 + (y_i y_j)^2 + 2x_i x_j y_i y_j$, but not quite because of the third term of cross-products.

★ 'Kernelising' the perceptron II

- ☞ We can capture this term by extending the feature vector with a third feature $\sqrt{2}x_i y_i$. This gives the following feature space:

$$\begin{aligned}\phi(\mathbf{x}_i) &= (x_i^2, y_i^2, \sqrt{2}x_i y_i) & \phi(\mathbf{x}_j) &= (x_j^2, y_j^2, \sqrt{2}x_j y_j) \\ \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) &= x_i^2 x_j^2 + y_i^2 y_j^2 + 2x_i x_j y_i y_j &= (\mathbf{x}_i \cdot \mathbf{x}_j)^2\end{aligned}$$

- ☞ We now define $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$, and replace $\mathbf{x}_i \cdot \mathbf{x}_j$ with $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ in the dual perceptron algorithm to obtain the *kernel perceptron* (Algorithm 7.4).
- ☞ This would work for many other kernels satisfying certain conditions.



Algorithm 7.4, p.226

★ Kernel perceptron

Algorithm KernelPerceptron(D, κ) – perceptron training algorithm using a kernel.

Input : labelled training data D in homogeneous coordinates;
 kernel function κ .

Output : coefficients α_i defining non-linear decision boundary.

```

1  $\alpha_i \leftarrow 0$  for  $1 \leq i \leq |D|$ ;
2 converged  $\leftarrow$  false;
3 while converged = false do
4   | converged  $\leftarrow$  true;
5   | for  $i = 1$  to  $|D|$  do
6   | | if  $y_i \sum_{j=1}^{|D|} \alpha_j y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \leq 0$  then
7   | | |  $\alpha_i \leftarrow \alpha_i + 1$ ;
8   | | | converged  $\leftarrow$  false;
9   | | end
10  | end
11 end
  
```

★ Other kernels I

We can define a polynomial kernel of any degree p as $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^p$. This transforms a d -dimensional input space into a high-dimensional feature space, such that each new feature is a product of p terms (possibly repeated).

If we include a constant, say $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^p$, we would get all lower-order terms as well. So, for example, in a bivariate input space and setting $p = 2$ the resulting feature space is

$$\phi(\mathbf{x}) = (x^2, y^2, \sqrt{2}xy, \sqrt{2}x, \sqrt{2}y, 1)$$

with linear as well as quadratic features.

★ Other kernels II

An often-used kernel is the *Gaussian kernel*, defined as

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

where σ is a parameter known as the *bandwidth*.

Notice that the soft margin optimisation problem (Equation 7.12) is defined in terms of dot products between training instances and hence the ‘kernel trick’ can be applied to SVMs:

$$\alpha_1^*, \dots, \alpha_n^* = \underset{\alpha_1, \dots, \alpha_n}{\operatorname{argmax}} -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^n \alpha_i$$

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

★ Other kernels III

- The decision boundary learned with a non-linear kernel cannot be represented by a simple weight vector in input space. Thus, in order to classify a new example \mathbf{x} we need to evaluate $y_i \sum_{j=1}^n \alpha_j y_j \kappa(\mathbf{x}, \mathbf{x}_j)$ which is an $O(n)$ computation involving all training examples, or at least the ones with non-zero multipliers α_j .
- This is why support vector machines are a popular choice as a kernel method, since they naturally promote sparsity in the support vectors.
- Although we have restricted attention to numerical features here, kernels can be defined over discrete structures, including trees, graphs, and logical formulae, opening the way to extending geometric models to non-numerical data.

What's next?

- 8 Distance-based models
 - Neighbours and exemplars
 - Nearest-neighbour classification
 - Distance-based clustering
 - K -means algorithm
 - Clustering around medoids
 - Silhouettes
 - Hierarchical clustering
 - From kernels to distances ★



If $\mathcal{X} = \mathbb{R}^d$, the *Minkowski distance* of order $p > 0$ is defined as

$$\text{Dis}_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{j=1}^d |x_j - y_j|^p \right)^{1/p} = \|\mathbf{x} - \mathbf{y}\|_p$$

where $\|\mathbf{z}\|_p = \left(\sum_{j=1}^d |z_j|^p \right)^{1/p}$ is the *p-norm* (sometimes denoted L_p norm) of the vector \mathbf{z} . We will often refer to Dis_p simply as the *p-norm*.

👉 The *2-norm* refers to the familiar *Euclidean distance*

$$\text{Dis}_2(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{j=1}^d (x_j - y_j)^2} = \sqrt{(\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})}$$

which measures distance ‘as the crow flies’.



☞ The *1-norm* denotes *Manhattan distance*, also called *cityblock distance*:

$$\text{Dis}_1(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^d |x_j - y_j|$$

This is the distance if we can only travel along coordinate axes.

☞ If we now let p grow larger, the distance will be more and more dominated by the largest coordinate-wise distance, from which we can infer that $\text{Dis}_\infty(\mathbf{x}, \mathbf{y}) = \max_j |x_j - y_j|$; this is also called *Chebyshev distance*.



- 👉 You will sometimes see references to the *0-norm* (or L_0 norm) which counts the number of non-zero elements in a vector. The corresponding distance then counts the number of positions in which vectors \mathbf{x} and \mathbf{y} differ. This is not strictly a Minkowski distance; however, we can define it as

$$\text{Dis}_0(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^d (x_j - y_j)^0 = \sum_{j=1}^d I[x_j \neq y_j]$$

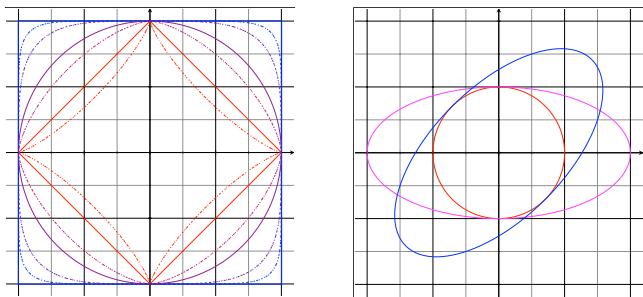
under the understanding that $x^0 = 0$ for $x = 0$ and 1 otherwise.

- 👉 If \mathbf{x} and \mathbf{y} are binary strings, this is also called the *Hamming distance*. Alternatively, we can see the Hamming distance as the number of bits that need to be flipped to change \mathbf{x} into \mathbf{y} .
- 👉 For non-binary strings of unequal length this can be generalised to the notion of *edit distance* or *Levenshtein distance*.



Figure 8.3, p.235

Circles and ellipses



(left) Lines connecting points at order- p Minkowski distance 1 from the origin for (from inside) $p = 0.8$; $p = 1$ (Manhattan distance, the **rotated square in red**); $p = 1.5$; $p = 2$ (Euclidean distance, the **violet circle**); $p = 4$; $p = 8$; and $p = \infty$ (Chebyshev distance, the **blue rectangle**). Notice that for points on the coordinate axes all distances agree. For the other points, our reach increases with p ; however, if we require a rotation-invariant distance metric then Euclidean distance is our only choice. **(right)** The rotated ellipse $\mathbf{x}^T \mathbf{R}^T \mathbf{S}^2 \mathbf{R} \mathbf{x} = 1/4$; the axis-parallel ellipse $\mathbf{x}^T \mathbf{S}^2 \mathbf{x} = 1/4$; and the circle $\mathbf{x}^T \mathbf{x} = 1/4$.



Given an instance space \mathcal{X} , a *distance metric* is a function $\text{Dis} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ such that for any $x, y, z \in \mathcal{X}$:

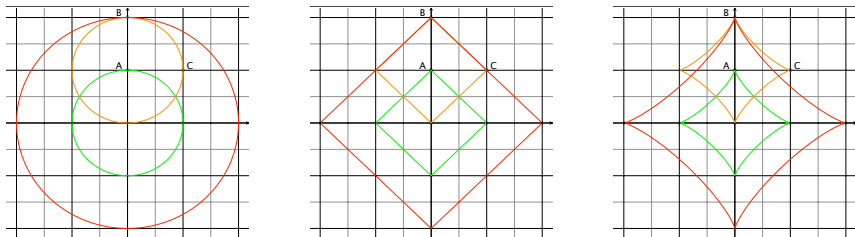
- ☞ distances between a point and itself are zero: $\text{Dis}(x, x) = 0$;
- ☞ all other distances are larger than zero: if $x \neq y$ then $\text{Dis}(x, y) > 0$;
- ☞ distances are symmetric: $\text{Dis}(y, x) = \text{Dis}(x, y)$;
- ☞ detours can not shorten the distance: $\text{Dis}(x, z) \leq \text{Dis}(x, y) + \text{Dis}(y, z)$.

If the second condition is weakened to a non-strict inequality – i.e., $\text{Dis}(x, y)$ may be zero even if $x \neq y$ – the function Dis is called a *pseudo-metric*.



Figure 8.4, p.236

The triangle inequality



(left) The **green circle** connects points the same Euclidean distance (i.e., Minkowski distance of order $p = 2$) away from the origin as A. The **orange circle** shows that B and C are equidistant from A. The **red circle** demonstrates that C is closer to the origin than B, which conforms to the triangle inequality. **(middle)** With Manhattan distance ($p = 1$), B and C are equally close to the origin and also equidistant from A. **(right)** With $p < 1$ (here, $p = 0.8$) C is further away from the origin than B; since both are again equidistant from A, it follows that travelling from the origin to C via A is quicker than going there directly, which violates the triangle inequality.



Consider the following matrices

$$\mathbf{R} = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} \quad \mathbf{S} = \begin{pmatrix} 1/2 & 0 \\ 0 & 1 \end{pmatrix} \quad \mathbf{M} = \begin{pmatrix} 5/8 & -3/8 \\ -3/8 & 5/8 \end{pmatrix}$$

The matrix \mathbf{R} describes a clockwise rotation of 45 degrees, and the diagonal matrix \mathbf{S} scales the x -axis by a factor 1/2. The equation

$$(\mathbf{SRx})^T (\mathbf{SRx}) = \mathbf{x}^T \mathbf{R}^T \mathbf{S}^T \mathbf{SRx} = \mathbf{x}^T \mathbf{R}^T \mathbf{S}^2 \mathbf{Rx} = \mathbf{x}^T \mathbf{Mx} = 1/4$$

describes a shape which, after clockwise rotation of 45 degrees and scaling of the x -axis by a factor 1/2, is a circle with radius 1/2 – i.e., the ‘ascending’ ellipse in [Figure 8.3 \(right\)](#). The ellipse equation is $(5/8)x^2 + (5/8)y^2 - (3/4)xy = 1/2$.

★ Mahalanobis distance

Often, the shape of the ellipse is estimated from data as the inverse of the covariance matrix: $\mathbf{M} = \mathbf{\Sigma}^{-1}$. This leads to the definition of the *Mahalanobis distance*

$$\text{Dis}_M(\mathbf{x}, \mathbf{y} | \mathbf{\Sigma}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mathbf{y})}$$

Using the covariance matrix in this way has the effect of decorrelating and normalising the features.

Clearly, Euclidean distance is a special case of Mahalanobis distance with the identity matrix \mathbf{I} as covariance matrix: $\text{Dis}_2(\mathbf{x}, \mathbf{y}) = \text{Dis}_M(\mathbf{x}, \mathbf{y} | \mathbf{I})$.

What's next?

- 8 Distance-based models
 - Neighbours and exemplars
 - Nearest-neighbour classification
 - Distance-based clustering
 - K -means algorithm
 - Clustering around medoids
 - Silhouettes
 - Hierarchical clustering
 - From kernels to distances ★

Means and distances I

Theorem (The arithmetic mean minimises squared Euclidean distance)

The arithmetic mean μ of a set of data points D in a Euclidean space is the unique point that minimises the sum of squared Euclidean distances to those data points.

Proof.

We will show that $\operatorname{argmin}_{\mathbf{y}} \sum_{\mathbf{x} \in D} \|\mathbf{x} - \mathbf{y}\|^2 = \mu$, where $\|\cdot\|$ denotes the 2-norm. We find this minimum by taking the gradient (the vector of partial derivatives with respect to y_i) of the sum and setting it to the zero vector:

$$\nabla_{\mathbf{y}} \sum_{\mathbf{x} \in D} \|\mathbf{x} - \mathbf{y}\|^2 = -2 \sum_{\mathbf{x} \in D} (\mathbf{x} - \mathbf{y}) = -2 \sum_{\mathbf{x} \in D} \mathbf{x} + 2|D|\mathbf{y} = \mathbf{0}$$

from which we derive $\mathbf{y} = \frac{1}{|D|} \sum_{\mathbf{x} \in D} \mathbf{x} = \mu$. □

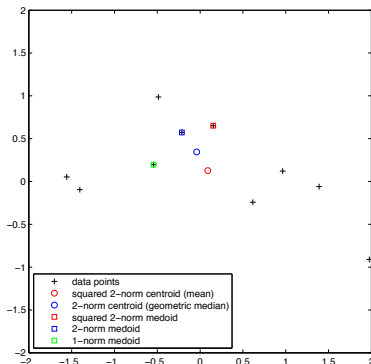
Means and distances II

- 👉 You may wonder what happens if we drop the square here: wouldn't it be more natural to take the point that minimises total Euclidean distance as exemplar?
- 👉 This point is known as the *geometric median*, as for univariate data it corresponds to the median or 'middle value' of a set of numbers. However, for multivariate data there is no closed-form expression for the geometric median, which needs to be calculated by successive approximation.
- 👉 In certain situations it makes sense to restrict an exemplar to be one of the given data points. In that case, we speak of a *medoid*, to distinguish it from a *centroid* which is an exemplar that doesn't have to occur in the data.
- 👉 Finding a medoid requires us to calculate, for each data point, the total distance to all other data points, in order to choose the point that minimises it. Regardless of the distance metric used, this is an $O(n^2)$ operation for n points.



Figure 8.5, p.239

Centroids and medoids



A small data set of 10 points, with circles indicating centroids and squares indicating medoids (the latter must be data points), for different distance metrics. Notice how the outlier on the bottom-right 'pulls' the mean away from the geometric median; as a result the corresponding medoid changes as well.

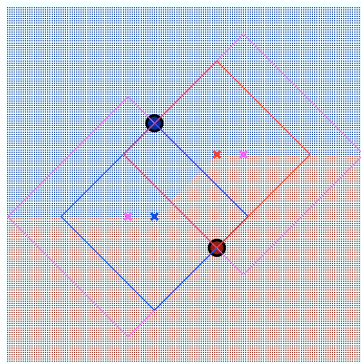
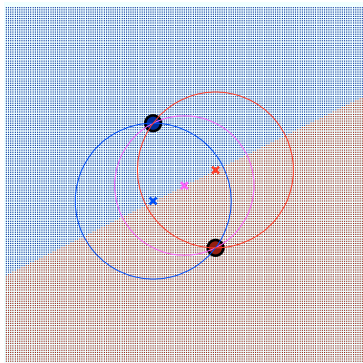
The basic linear classifier is distance-based

- ✎ The basic linear classifier constructs the decision boundary as the perpendicular bisector of the line segment connecting the two exemplars (one for each class).
- ✎ An alternative, distance-based way to classify instances without direct reference to a decision boundary is by the following decision rule: if \mathbf{x} is nearest to μ^+ then classify it as positive, otherwise as negative; or equivalently, classify an instance to the class of the *nearest* exemplar.
- ✎ If we use Euclidean distance as our closeness measure, simple geometry tells us we get exactly the same decision boundary ([Figure 8.6 \(left\)](#)).
- ✎ So the basic linear classifier can be interpreted from a distance-based perspective as constructing exemplars that minimise squared Euclidean distance within each class, and then applying a nearest-exemplar decision rule.



Figure 8.6, p.240

Two-exemplar decision boundaries

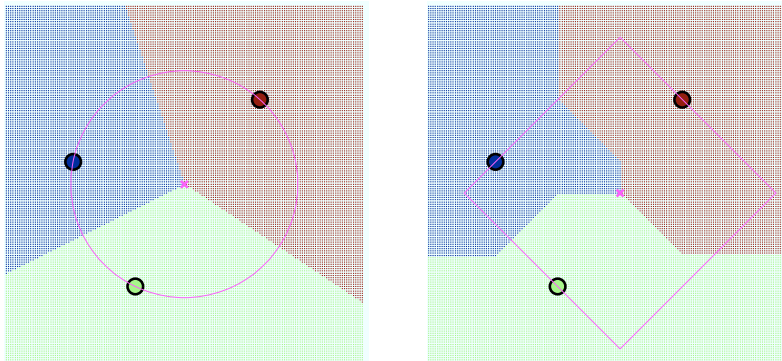


(left) For two exemplars the nearest-exemplar decision rule with Euclidean distance results in a linear decision boundary coinciding with the perpendicular bisector of the line connecting the two exemplars. **(right)** Using Manhattan distance the circles are replaced by diamonds.



Figure 8.7, p.240

Three-exemplar decision boundaries



(left) Decision regions defined by the 2-norm nearest-exemplar decision rule for three exemplars. **(right)** With Manhattan distance the decision regions become non-convex.



Two neighbours know more than one

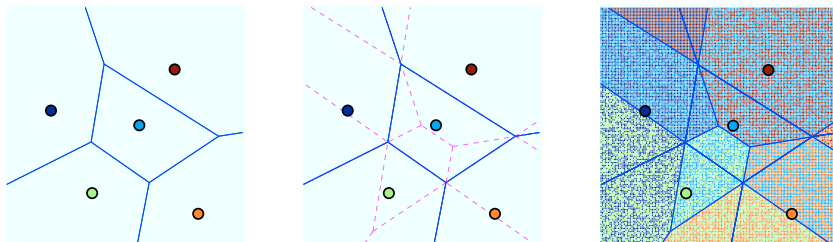
Figure 8.8 (left) gives a Voronoi tessellation for five exemplars. Each line segment is part of the perpendicular bisector of two exemplars. There are $\binom{5}{2} = 10$ pairs of exemplars, but two of these pairs are too far away from each other so we observe only eight line segments in the Voronoi tessellation.

If we now also take the second-nearest exemplars into account, each Voronoi cell is further subdivided: for instance, since the central point has four neighbours, the central cell is divided into four subregions (Figure 8.8 (middle)). You can think of those additional line segments as being part of the Voronoi tessellation that results when the central point is removed. The other exemplars have only three immediate neighbours and so their cells are divided into three subregions. We thus obtain 16 '2-nearest exemplar' decision regions, each of which is defined by a different pair of nearest and second-nearest exemplars. Figure 8.8 (right) shades each of these regions according to the two nearest exemplars spanning it.



Figure 8.8, p.241

One vs two nearest neighbours



(left) Voronoi tessellation for five exemplars. **(middle)** Taking the two nearest exemplars into account leads to a further subdivision of each Voronoi cell. **(right)** The shading indicates which exemplars contribute to which cell.

Distance-based models

To summarise, the main ingredients of distance-based models are

- 👉 distance metrics, which can be Euclidean, Manhattan, Minkowski or Mahalanobis, among many others;
- 👉 exemplars: centroids that find a centre of mass according to a chosen distance metric, or medoids that find the most centrally located data point; and
- 👉 distance-based decision rules, which take a vote among the k nearest exemplars.

In the next subsections these ingredients are combined in various ways to obtain supervised and unsupervised learning algorithms.

What's next?

- 8 Distance-based models
 - Neighbours and exemplars
 - **Nearest-neighbour classification**
 - Distance-based clustering
 - *K*-means algorithm
 - Clustering around medoids
 - Silhouettes
 - Hierarchical clustering
 - From kernels to distances ★

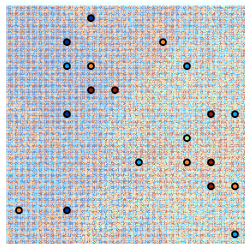
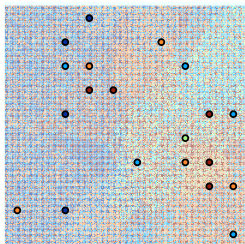
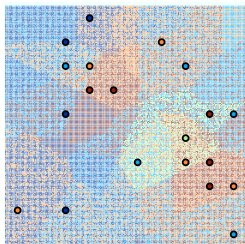
Nearest-neighbour classifier

- 👉 kNN uses the training data as exemplars, so training is $O(n)$ (but prediction is also $O(n)$!)
- 👉 1NN perfectly separates training data, so low bias but high variance
- 👉 By increasing the number of neighbours k we increase bias and decrease variance (what happens when $k = n$?)
- 👉 Easily adapted to real-valued targets, and even to structured objects (nearest-neighbour retrieval). Can also output probabilities when $k > 1$
- 👉 Warning: in high-dimensional spaces everything is far away from everything and so pairwise distances are uninformative (curse of dimensionality)



Figure 8.9, p.244

Three-, five- and seven-nearest neighbour

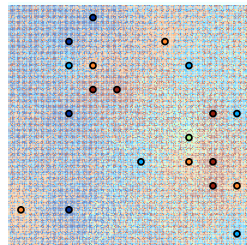
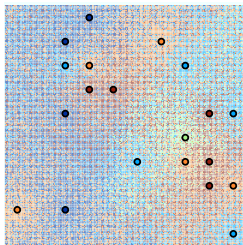
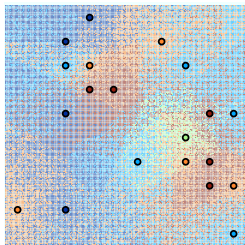


(left) Decision regions of a 3-nearest neighbour classifier; the shading represents the predicted probability distribution over the five classes. **(middle)** 5-nearest neighbour. **(right)** 7-nearest neighbour.



Figure 8.10, p.245

Distance weighting



(left) 3-nearest neighbour with distance weighting on the data from Figure 8.9. **(middle)** 5-nearest neighbour. **(right)** 7-nearest neighbour.

What's next?

- 8 Distance-based models
 - Neighbours and exemplars
 - Nearest-neighbour classification
 - Distance-based clustering
 - K -means algorithm
 - Clustering around medoids
 - Silhouettes
 - Hierarchical clustering
 - From kernels to distances ★



Given a data matrix \mathbf{X} , the *scatter matrix* is the matrix

$$\mathbf{S} = (\mathbf{X} - \mathbf{1}\boldsymbol{\mu})^T (\mathbf{X} - \mathbf{1}\boldsymbol{\mu}) = \sum_{i=1}^n (\mathbf{X}_{i\cdot} - \boldsymbol{\mu})^T (\mathbf{X}_{i\cdot} - \boldsymbol{\mu})$$

where $\boldsymbol{\mu}$ is a row vector containing all column means of \mathbf{X} . The *scatter* of \mathbf{X} is defined as $\text{Scat}(\mathbf{X}) = \sum_{i=1}^n \|\mathbf{X}_{i\cdot} - \boldsymbol{\mu}\|^2$, which is equal to the trace of the scatter matrix (i.e., the sum of its diagonal elements).

Within-cluster and between-cluster scatter

Imagine that we partition data D into K subsets $D_1 \uplus \dots \uplus D_K = D$, and let μ_j denote the mean of D_j . Let \mathbf{S} be the scatter matrix of D , and \mathbf{S}_j be the scatter matrices of D_j .

- ☞ These scatter matrices then have the relationship $\mathbf{S} = \sum_{j=1}^K \mathbf{S}_j + \mathbf{B}$.
- ☞ \mathbf{B} is the *between-cluster scatter matrix* that results by replacing each point in D with the corresponding centroid μ_j : it describes the spread of the centroids.
- ☞ Each \mathbf{S}_j is called a *within-cluster scatter matrix* and describes the compactness of the j -th cluster.
- ☞ It follows that the traces of these matrices can be decomposed similarly:

$$\text{Scat}(D) = \sum_{j=1}^K \text{Scat}(D_j) + \sum_{j=1}^K |D_j| \|\mu_j - \mu\|^2$$

- ☞ The *K-means problem* is to find a partition that minimises the first term (or maximises the second term).



Example 8.3, p.246

Reducing scatter by partitioning data I

Consider the following five points centred around $(0,0)$: $(0,3)$, $(3,3)$, $(3,0)$, $(-2,-4)$ and $(-4,-2)$. The scatter matrix is

$$\mathbf{S} = \begin{pmatrix} 0 & 3 & 3 & -2 & -4 \\ 3 & 3 & 0 & -4 & -2 \end{pmatrix} \begin{pmatrix} 0 & 3 \\ 3 & 3 \\ 3 & 0 \\ -2 & -4 \\ -4 & -2 \end{pmatrix} = \begin{pmatrix} 38 & 25 \\ 25 & 38 \end{pmatrix}$$

with trace $\text{Scat}(D) = 76$. If we cluster the first two points together in one cluster and the remaining three in another, then we obtain cluster means $\boldsymbol{\mu}_1 = (1.5, 3)$ and $\boldsymbol{\mu}_2 = (-1, -2)$ and within-cluster scatter matrices

$$\mathbf{S}_1 = \begin{pmatrix} 0-1.5 & 3-1.5 \\ 3-3 & 3-3 \end{pmatrix} \begin{pmatrix} 0-1.5 & 3-3 \\ 3-1.5 & 3-3 \end{pmatrix} = \begin{pmatrix} 4.5 & 0 \\ 0 & 0 \end{pmatrix}$$

$$\mathbf{S}_2 = \begin{pmatrix} 3-(-1) & -2-(-1) & -4-(-1) \\ 0-(-2) & -4-(-2) & -2-(-2) \end{pmatrix} \begin{pmatrix} 3-(-1) & 0-(-2) \\ -2-(-1) & -4-(-2) \\ -4-(-1) & -2-(-2) \end{pmatrix} = \begin{pmatrix} 26 & 10 \\ 10 & 8 \end{pmatrix}$$

with traces $\text{Scat}(D_1) = 4.5$ and $\text{Scat}(D_2) = 34$.



Example 8.3, p.246

Reducing scatter by partitioning data II

Two copies of μ_1 and three copies of μ_2 have, by definition, the same centre of gravity as the complete data set: $(0,0)$ in this case. We thus calculate the between-cluster scatter matrix as

$$\mathbf{B} = \begin{pmatrix} 1.5 & 1.5 & -1 & -1 & -1 \\ 3 & 3 & -2 & -2 & -2 \end{pmatrix} \begin{pmatrix} 1.5 & 3 \\ 1.5 & 3 \\ -1 & -2 \\ -1 & -2 \\ -1 & -2 \end{pmatrix} = \begin{pmatrix} 7.5 & 15 \\ 15 & 30 \end{pmatrix}$$

with trace 37.5. Alternatively, if we treat the first three points as a cluster and put the other two in a second cluster, then we obtain cluster means $\mu'_1 = (2,2)$ and $\mu'_2 = (-3,-3)$, and within-cluster scatter matrices

$$\mathbf{S}'_1 = \begin{pmatrix} 0-2 & 3-2 & 3-2 \\ 3-2 & 3-2 & 0-2 \end{pmatrix} \begin{pmatrix} 0-2 & 3-2 \\ 3-2 & 3-2 \\ 3-2 & 0-2 \end{pmatrix} = \begin{pmatrix} 6 & -3 \\ -3 & 6 \end{pmatrix}$$

$$\mathbf{S}'_2 = \begin{pmatrix} -2-(-3) & -4-(-3) \\ -4-(-3) & -2-(-3) \end{pmatrix} \begin{pmatrix} -2-(-3) & -4-(-3) \\ -4-(-3) & -2-(-3) \end{pmatrix} = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix}$$



Example 8.3, p.246

Reducing scatter by partitioning data III

with traces $\text{Scat}(D'_1) = 12$ and $\text{Scat}(D'_2) = 4$. The between-cluster scatter matrix is

$$\mathbf{B}' = \begin{pmatrix} 2 & 2 & 2 & -3 & -3 \\ 2 & 2 & 2 & -3 & -3 \end{pmatrix} \begin{pmatrix} 2 & 2 \\ 2 & 2 \\ 2 & 2 \\ -3 & -3 \\ -3 & -3 \end{pmatrix} = \begin{pmatrix} 30 & 30 \\ 30 & 30 \end{pmatrix}$$

with trace 60. Clearly, the second clustering produces tighter clusters whose centroids are further apart.



Algorithm $KMeans(D, K)$ – K -means clustering using Euclidean distance Dis_2 .

Input : data $D \subseteq \mathbb{R}^d$; number of clusters $K \in \mathbb{N}$.

Output : K cluster means $\mu_1, \dots, \mu_K \in \mathbb{R}^d$.

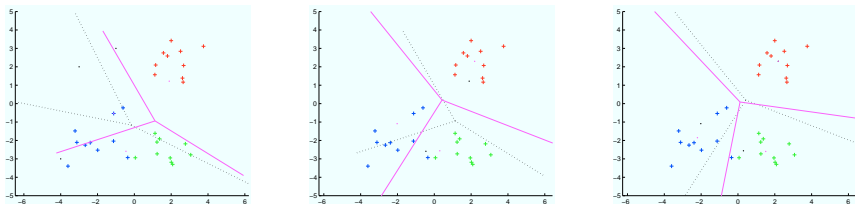
```

1 randomly initialise  $K$  vectors  $\mu_1, \dots, \mu_K \in \mathbb{R}^d$ ;
2 repeat
3   assign each  $\mathbf{x} \in D$  to  $\arg \min_j Dis_2(\mathbf{x}, \mu_j)$ ;
4   for  $j = 1$  to  $K$  do
5      $D_j \leftarrow \{\mathbf{x} \in D \mid \mathbf{x} \text{ assigned to cluster } j\}$ ;
6      $\mu_j = \frac{1}{|D_j|} \sum_{\mathbf{x} \in D_j} \mathbf{x}$ ;
7   end
8 until no change in  $\mu_1, \dots, \mu_K$ ;
9 return  $\mu_1, \dots, \mu_K$ ;
```



Figure 8.11, p.248

K-means clustering



(left) First iteration of 3-means on Gaussian mixture data. The dotted lines are the Voronoi boundaries resulting from randomly initialised centroids; the **violet solid lines** are the result of the recalculated means. **(middle)** Second iteration, taking the previous partition as starting point (dotted line). **(right)** Third iteration with stable clustering.



Refer back to the MLM data set in [Table 1.4](#) (it is also helpful to look at its two-dimensional approximation in [Figure 1.7](#)). When we run K -means on this data with $K = 3$, we obtain the clusters [{Associations, Trees, Rules}](#), [{GMM, naive Bayes}](#), and a larger cluster with the remaining data points. When we run it with $K = 4$, we get that the large cluster splits into two: [{kNN, Linear Classifier, Linear Regression}](#) and [{Kmeans, Logistic Regression, SVM}](#); but also that [GMM](#) gets reallocated to the latter cluster, and [naive Bayes](#) ends up as a singleton.



Stationary points in clustering

Consider the task of dividing the set of numbers $\{8, 44, 50, 58, 84\}$ into two clusters. There are four possible partitions that 2-means can find:

☞ $\{8\}, \{44, 50, 58, 84\}$;

☞ $\{8, 44\}, \{50, 58, 84\}$;

☞ $\{8, 44, 50\}, \{58, 84\}$; and

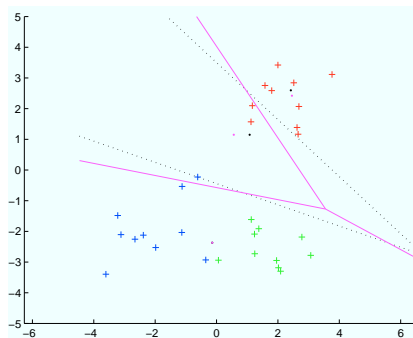
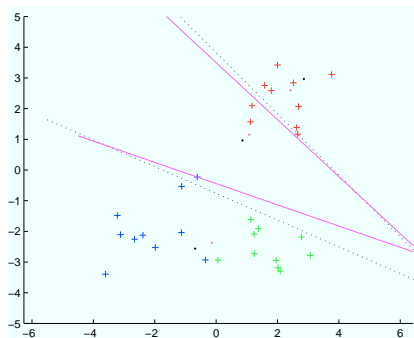
☞ $\{8, 44, 50, 58\}, \{84\}$.

It is easy to verify that each of these establishes a stationary point for 2-means, and hence will be found with a suitable initialisation. Only the first clustering is optimal; i.e., it minimises the total within-cluster scatter.



Figure 8.12, p.249

Sub-optimality of K -means



(left) First iteration of 3-means on the same data as Figure 8.11 with differently initialised centroids. (right) 3-means has converged to a sub-optimal clustering.



Algorithm $KMedoids(D, K, Dis)$ – K -medoids clustering using arbitrary distance metric Dis .

Input : data $D \subseteq \mathcal{X}$; number of clusters $K \in \mathbb{N}$;
 distance metric $Dis: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$.

Output : K medoids $\mu_1, \dots, \mu_K \in D$, representing a predictive clustering of \mathcal{X} .

```

1 randomly pick  $K$  data points  $\mu_1, \dots, \mu_K \in D$ ;
2 repeat
3   assign each  $\mathbf{x} \in D$  to  $\arg \min_j Dis(\mathbf{x}, \mu_j)$ ;
4   for  $j = 1$  to  $k$  do
5      $D_j \leftarrow \{\mathbf{x} \in D \mid \mathbf{x} \text{ assigned to cluster } j\}$ ;
6      $\mu_j = \arg \min_{\mathbf{x} \in D_j} \sum_{\mathbf{x}' \in D_j} Dis(\mathbf{x}, \mathbf{x}')$ ;
7   end
8 until no change in  $\mu_1, \dots, \mu_K$ ;
9 return  $\mu_1, \dots, \mu_K$ ;
```



Algorithm 8.3, p.251

Partitioning around medoids clustering

Algorithm PAM(D, K, Dis) – Partitioning around medoids clustering using arbitrary distance metric Dis .

Input : data $D \subseteq \mathcal{X}$; number of clusters $K \in \mathbb{N}$;
distance metric $\text{Dis} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$.

Output : K medoids $\mu_1, \dots, \mu_K \in D$, representing a predictive clustering of \mathcal{X} .

```

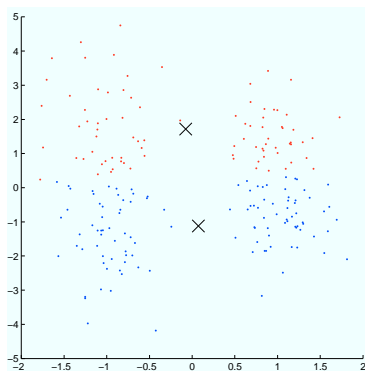
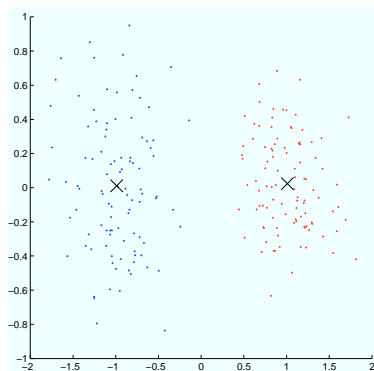
1 randomly pick  $K$  data points  $\mu_1, \dots, \mu_K \in D$ ;
2 repeat
3   assign each  $\mathbf{x} \in D$  to  $\arg \min_j \text{Dis}(\mathbf{x}, \mu_j)$ ;
4   for  $j = 1$  to  $k$  do
5      $D_j \leftarrow \{\mathbf{x} \in D \mid \mathbf{x} \text{ assigned to cluster } j\}$ ;
6   end
7    $Q \leftarrow \sum_j \sum_{\mathbf{x} \in D_j} \text{Dis}(\mathbf{x}, \mu_j)$ ;
8   for each medoid  $\mathbf{m}$  and each non-medoid  $\mathbf{o}$  do
9     calculate the improvement in  $Q$  resulting from swapping  $\mathbf{m}$  with  $\mathbf{o}$ ;
10  end
11  select the pair with maximum improvement and swap;
12 until no further improvement possible;
13 return  $\mu_1, \dots, \mu_K$ ;

```




Figure 8.13, p.251

Scale-sensitivity of K -means



(left) On this data 2-means detects the right clusters. **(right)** After rescaling the y -axis, this configuration has a higher between-cluster scatter than the intended one.

Silhouettes I

- ✎ For any data point \mathbf{x}_i , let $d(\mathbf{x}_i, D_j)$ denote the average distance of \mathbf{x}_i to the data points in cluster D_j , and let $j(i)$ denote the index of the cluster that \mathbf{x}_i belongs to.
- ✎ Furthermore, let $a(\mathbf{x}_i) = d(\mathbf{x}_i, D_{j(i)})$ be the average distance of \mathbf{x}_i to the points in its own cluster $D_{j(i)}$, and let $b(\mathbf{x}_i) = \min_{k \neq j(i)} d(\mathbf{x}_i, D_k)$ be the average distance to the points in its neighbouring cluster.
- ✎ We would expect $a(\mathbf{x}_i)$ to be considerably smaller than $b(\mathbf{x}_i)$, but this cannot be guaranteed.
- ✎ So we can take the difference $b(\mathbf{x}_i) - a(\mathbf{x}_i)$ as an indication of how 'well-clustered' \mathbf{x}_i is, and divide this by $b(\mathbf{x}_i)$ to obtain a number less than or equal to 1.

Silhouettes II

- ✎ It is, however, conceivable that $a(\mathbf{x}_i) > b(\mathbf{x}_i)$, in which case the difference $b(\mathbf{x}_i) - a(\mathbf{x}_i)$ is negative. This describes the situation that, on average, the members of the neighbouring cluster are closer to \mathbf{x}_i than the members of its own cluster.
- ✎ In order to get a normalised value we divide by $a(\mathbf{x}_i)$ in this case. This leads to the following definition:

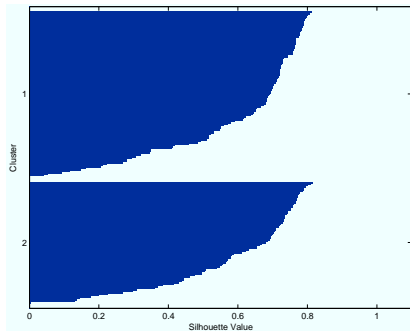
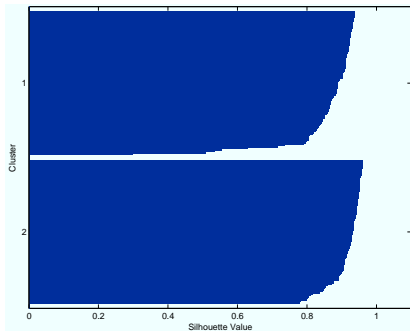
$$s(\mathbf{x}_i) = \frac{b(\mathbf{x}_i) - a(\mathbf{x}_i)}{\max(a(\mathbf{x}_i), b(\mathbf{x}_i))}$$

- ✎ A *silhouette* then sorts and plots $s(\mathbf{x})$ for each instance, grouped by cluster.



Figure 8.14, p.252

Silhouettes



- (left)** Silhouette for the clustering in [Figure 8.13 \(left\)](#), using squared Euclidean distance. Almost all points have a high $s(\mathbf{x})$, which means that they are much closer, on average, to the other members of their cluster than to the members of the neighbouring cluster.
- (right)** The silhouette for the clustering in [Figure 8.13 \(right\)](#) is much less convincing.

What's next?

- 8 Distance-based models
 - Neighbours and exemplars
 - Nearest-neighbour classification
 - Distance-based clustering
 - K -means algorithm
 - Clustering around medoids
 - Silhouettes
 - Hierarchical clustering
 - From kernels to distances ★



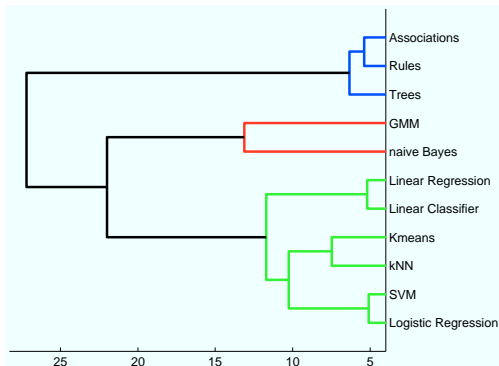
Hierarchical clustering of MLM data

We continue [Example 8.4](#). A hierarchical clustering of the MLM data is given in [Figure 8.15](#). The tree shows that the three logical methods at the top form a strong cluster. If we wanted three clusters, we get the logical cluster, a second small cluster $\{\text{GMM}, \text{naive Bayes}\}$, and the remainder. If we wanted four clusters, we would separate GMM and naive Bayes , as the tree indicates this cluster is the least tight of the three (notice that this is slightly different from the solution found by 4-means). If we wanted five clusters, we would construct $\{\text{Linear Regression}, \text{LinearClassifier}\}$ as a separate cluster. This illustrates the key advantage of hierarchical clustering: it doesn't require fixing the number of clusters in advance.



Figure 8.15, p.253

Hierarchical clustering example



A dendrogram (printed left to right to improve readability) constructed by hierarchical clustering from the data in [Table 1.4](#).



Given a data set D , a *dendrogram* is a binary tree with the elements of D at its leaves.

An internal node of the tree represents the subset of elements in the leaves of the subtree rooted at that node.

The level of a node is the distance between the two clusters represented by the children of the node.

Leaves have level 0.



A *linkage function* $L : 2^{\mathcal{X}} \times 2^{\mathcal{X}} \rightarrow \mathbb{R}$ calculates the distance between arbitrary subsets of the instance space, given a distance metric $\text{Dis} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$.

The most common linkage functions are as follows:

- Single linkage** defines the distance between two clusters as the *smallest* pairwise distance between elements from each cluster.
- Complete linkage** defines the distance between two clusters as the *largest* pointwise distance.
- Average linkage** defines the cluster distance as the *average* pointwise distance.
- Centroid linkage** defines the cluster distance as the point distance between the cluster means.



These linkage functions can be defined mathematically as follows:

$$L_{\text{single}}(A, B) = \min_{x \in A, y \in B} \text{Dis}(x, y)$$

$$L_{\text{complete}}(A, B) = \max_{x \in A, y \in B} \text{Dis}(x, y)$$

$$L_{\text{average}}(A, B) = \frac{\sum_{x \in A, y \in B} \text{Dis}(x, y)}{|A| \cdot |B|}$$

$$L_{\text{centroid}}(A, B) = \text{Dis} \left(\frac{\sum_{x \in A} x}{|A|}, \frac{\sum_{y \in B} y}{|B|} \right)$$

Clearly, all these linkage functions coincide for singleton clusters:

$L(\{x\}, \{y\}) = \text{Dis}(x, y)$. However, for larger clusters they start to diverge.



For example, suppose $\text{Dis}(x, y) < \text{Dis}(x, z)$, then the linkage between $\{x\}$ and $\{y, z\}$ is different in all four cases:

$$L_{\text{single}}(\{x\}, \{y, z\}) = \text{Dis}(x, y)$$

$$L_{\text{complete}}(\{x\}, \{y, z\}) = \text{Dis}(x, z)$$

$$L_{\text{average}}(\{x\}, \{y, z\}) = (\text{Dis}(x, y) + \text{Dis}(x, z)) / 2$$

$$L_{\text{centroid}}(\{x\}, \{y, z\}) = \text{Dis}(x, (y + z) / 2)$$



Hierarchical agglomerative clustering

Algorithm $HAC(D, L)$ – Hierarchical agglomerative clustering.

Input : data $D \subseteq \mathcal{X}$; linkage function $L: 2^{\mathcal{X}} \times 2^{\mathcal{X}} \rightarrow \mathbb{R}$ defined in terms of distance metric.

Output : a dendrogram representing a descriptive clustering of D .

- 1 initialise clusters to singleton data points;
 - 2 create a leaf at level 0 for every singleton cluster;
 - 3 **repeat**
 - 4 find the pair of clusters X, Y with lowest linkage l , and merge;
 - 5 create a parent of X, Y at level l ;
 - 6 **until** all data points are in one cluster;
 - 7 **return** the constructed binary tree with linkage levels;
-



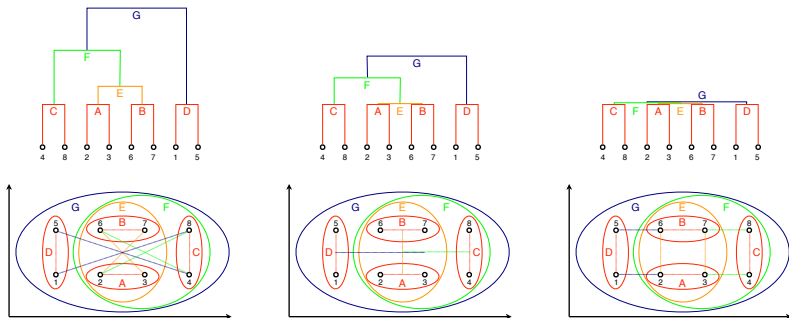
We consider a regular grid of 8 points in two rows of four (Figure 8.16). We assume that ties are broken by small irregularities. Each linkage function merges the same clusters in the same order, but the linkages are quite different in each case.

- ☞ Complete linkage gives the impression that D is far removed from the rest, whereas by moving D very slightly to the right it would have been added to E before C.
- ☞ With centroid linkage we see that E has in fact the same linkage as A and B, which means that A and B are not really discernible as separate clusters, even though they are found first.
- ☞ Single linkage seems preferable in this case, as it most clearly demonstrates that there is no meaningful cluster structure in this set of points.



Figure 8.16, p.256

Linkage matters

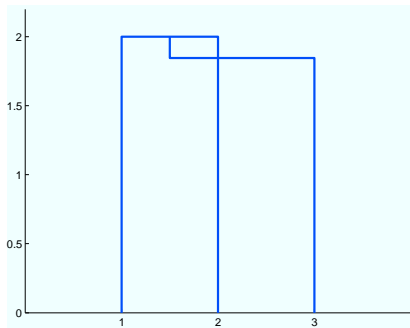
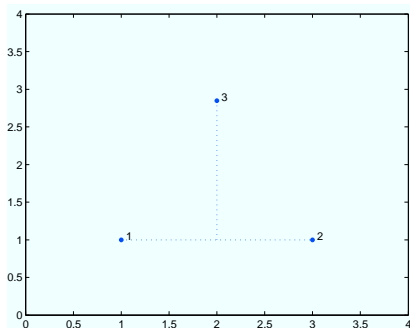


(left) Complete linkage defines cluster distance as the largest pairwise distance between elements from each cluster, indicated by the coloured lines between data points. **(middle)** Centroid linkage defines the distance between clusters as the distance between their means. Notice that E obtains the same linkage as A and B, and so the latter clusters effectively disappear. **(right)** Single linkage defines the distance between clusters as the smallest pairwise distance. The dendrogram all but collapses, which means that no meaningful clusters are found in the given grid configuration.



Figure 8.17, p.257

★ Non-monotonic dendrogram

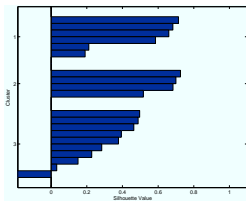
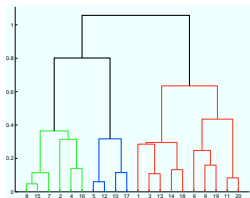
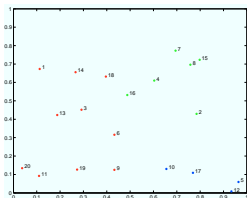


(left) Points 1 and 2 are closer to each other than to point 3. However, the distance between point 3 to the centroid of the other two points is less than any of the pairwise distances. **(right)** This results in a decrease in centroid linkage when adding point 3 to cluster $\{1, 2\}$, and hence a non-monotonic dendrogram. The other three linkage functions are monotonic (the example also serves as an illustration why average linkage and centroid linkage are not the same).



Figure 8.18, p.258

A spurious clustering



(left) 20 data points, generated by uniform random sampling. **(middle)** The dendrogram generated from complete linkage. The three clusters suggested by the dendrogram are spurious as they cannot be observed in the data. **(right)** The rapidly decreasing silhouette values in each cluster confirm the absence of a strong cluster structure. Point 18 has a negative silhouette value as it is on average closer to the **green points** than to the other **red points**.

What's next?

- 8 Distance-based models
 - Neighbours and exemplars
 - Nearest-neighbour classification
 - Distance-based clustering
 - K -means algorithm
 - Clustering around medoids
 - Silhouettes
 - Hierarchical clustering
 - From kernels to distances ★

★ From kernels to distances I

Recall that a kernel is a function $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$ that calculates a dot product in some feature space, but without constructing the feature vectors $\phi(\mathbf{x})$ explicitly. Any learning method that can be defined purely in terms of dot products of data points is amenable to such ‘kernelisation’.

We can apply the same ‘kernel trick’ to many distance-based learning methods. The key insight is that Euclidean distance can be rewritten in terms of dot products:

$$\text{Dis}_2(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{(\mathbf{x} - \mathbf{y}) \cdot (\mathbf{x} - \mathbf{y})} = \sqrt{\mathbf{x} \cdot \mathbf{x} - 2\mathbf{x} \cdot \mathbf{y} + \mathbf{y} \cdot \mathbf{y}}$$

The two terms $\mathbf{x} \cdot \mathbf{x}$ and $\mathbf{y} \cdot \mathbf{y}$ have the effect of making the overall expression translation-invariant (as the dot product isn’t).

★ From kernels to distances II

Replacing the dot product with a kernel function κ , we can construct the following kernelised distance:

$$\text{Dis}_{\kappa}(\mathbf{x}, \mathbf{y}) = \sqrt{\kappa(\mathbf{x}, \mathbf{x}) - 2\kappa(\mathbf{x}, \mathbf{y}) + \kappa(\mathbf{y}, \mathbf{y})}$$

It turns out that Dis_{κ} defines a pseudo-metric (see [Definition 8.2](#)) whenever κ is a positive semi-definite kernel.

(It is only a metric if the feature mapping ϕ is injective: suppose not, then some distinct \mathbf{x} and \mathbf{y} are mapped to the same feature vector $\phi(\mathbf{x}) = \phi(\mathbf{y})$, from which we derive $\kappa(\mathbf{x}, \mathbf{x}) - 2\kappa(\mathbf{x}, \mathbf{y}) + \kappa(\mathbf{y}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}) - 2\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) + \phi(\mathbf{y}) \cdot \phi(\mathbf{y}) = 0$.)



Algorithm 8.5, p.259

★ 'Kernelised' K -means clustering

Algorithm Kernel-KMeans(D, K) – K -means clustering using kernelised distance Dis_κ .

Input : data $D \subseteq \mathcal{X}$; number of clusters $K \in \mathbb{N}$.

Output : K -fold partition $D_1 \uplus \dots \uplus D_K = D$.

```

1 randomly initialise  $K$  clusters  $D_1, \dots, D_K$ ;
2 repeat
3   assign each  $\mathbf{x} \in D$  to  $\arg \min_j \frac{1}{|D_j|} \sum_{\mathbf{y} \in D_j} \text{Dis}_\kappa(\mathbf{x}, \mathbf{y})$ ;
4   for  $j = 1$  to  $K$  do
5      $D_j \leftarrow \{\mathbf{x} \in D \mid \mathbf{x} \text{ assigned to cluster } j\}$ ;
6   end
7 until no change in  $D_1, \dots, D_K$ ;
8 return  $D_1, \dots, D_K$ ;

```

★ Cosine similarity

There is an alternative way to turn dot products into distances. Since the dot product can be written as $\|\mathbf{x}\| \cdot \|\mathbf{y}\| \cos \theta$, where θ is the angle between the vectors \mathbf{x} and \mathbf{y} , we define the *cosine similarity* as

$$\cos \theta = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|} = \frac{\mathbf{x} \cdot \mathbf{y}}{\sqrt{(\mathbf{x} \cdot \mathbf{x})(\mathbf{y} \cdot \mathbf{y})}}$$

Cosine similarity differs from Euclidean distance in that it doesn't depend on the length of the vectors \mathbf{x} and \mathbf{y} .

On the other hand, it is not translation-independent, but assigns special status to the origin: one way to think of it is as a projection onto a unit sphere around the origin, and measuring distance on that sphere. Cosine similarity is usually turned into a distance metric by taking $1 - \cos \theta$.

What's next?

9

Probabilistic models

- The normal distribution and its geometric interpretations
- Probabilistic models for categorical data
 - Using a naive Bayes model for classification
 - Training a naive Bayes model
- Discriminative learning by optimising conditional likelihood ★
- Probabilistic models with hidden variables
 - Expectation-Maximisation ★
 - Gaussian mixture models ★
- Compression-based models ★

Discriminative and generative probabilistic models

- 👉 *Discriminative models* model the posterior probability distribution $P(Y|X)$, where Y is the target variable and X are the features. That is, given X they return a probability distribution over Y .
- 👉 *Generative models* model the joint distribution $P(Y, X)$ of the target Y and the feature vector X . Once we have access to this joint distribution we can derive any conditional or marginal distribution involving the same variables. In particular, since $P(X) = \sum_y P(Y = y, X)$ it follows that the posterior distribution can be obtained as $P(Y|X) = \frac{P(Y, X)}{\sum_y P(Y = y, X)}$.
- 👉 Alternatively, generative models can be described by the likelihood function $P(X|Y)$, since $P(Y, X) = P(X|Y)P(Y)$ and the target or prior distribution (usually abbreviated to ‘prior’) can be easily estimated or postulated.
- 👉 Such models are called ‘generative’ because we can sample from the joint distribution to obtain new data points together with their labels. Alternatively, we can use $P(Y)$ to sample a class and $P(X|Y)$ to sample an instance for that class.



Assessing uncertainty in estimates

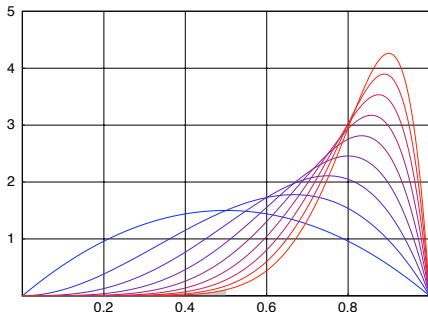
Suppose we want to estimate the probability θ that an arbitrary e-mail is spam, so that we can use the appropriate prior distribution.

- The natural thing to do is to inspect n e-mails, determine the number of spam e-mails d , and set $\hat{\theta} = d/n$; we don't really need any complicated statistics to tell us that.
- However, while this is the most likely estimate of θ – the maximum a posteriori (MAP) estimate – this doesn't mean that other values of θ are completely ruled out.
- We model this by a probability distribution over θ (a Beta distribution in this case) which is updated each time new information comes in. This is further illustrated in [Figure 9.1](#) for a distribution that is more and more skewed towards spam.
- For each curve, its bias towards spam is given by the area under the curve and to the right of $\theta = 1/2$.



Figure 9.1, p.264

Assessing uncertainty in estimates



Each time we inspect an e-mail, we are reducing our uncertainty regarding the prior spam probability θ . After we inspect two e-mails and observe one spam, the possible θ values are characterised by a symmetric distribution around $1/2$. If we inspect a third, fourth, \dots , tenth e-mail and each time (except the first one) it is spam, then this distribution narrows and shifts a little bit to the right each time. The distribution for n e-mails reaches its maximum at $\hat{\theta}_{\text{MAP}} = \frac{n-1}{n}$ (e.g., $\hat{\theta}_{\text{MAP}} = 0.8$ for $n = 5$).

The Bayesian perspective I

Explicitly modelling the posterior distribution over the parameter θ has a number of advantages that are usually associated with the ‘Bayesian’ perspective:

- We can precisely characterise the uncertainty that remains about our estimate by quantifying the spread of the posterior distribution.
- We can obtain a generative model for the parameter by sampling from the posterior distribution, which contains much more information than a summary statistic such as the MAP estimate can convey – so, rather than using a single e-mail with $\theta = \theta_{\text{MAP}}$, our generative model can contain a number of e-mails with θ sampled from the posterior distribution.
- We can quantify the probability of statements such as ‘e-mails are biased towards ham’ (the tiny shaded area in [Figure 9.1](#) demonstrates that after observing one ham and nine spam e-mails this probability is very small, about 0.6%).

The Bayesian perspective II

☞ We can use one of these distributions to encode our prior beliefs: e.g., if we believe that the proportions of spam and ham are typically 50–50, we can take the distribution for $n = 2$ (the lowest, symmetric one in [Figure 9.1](#)) as our prior.

The key point is that probabilities do not have to be interpreted as estimates of relative frequencies, but can carry the more general meaning of (possibly subjective) degrees of belief.

Consequently, we can attach a probability distribution to almost anything: not just features and targets, but also model parameters and even models.

A classifier is *Bayes-optimal* if it always assigns $\operatorname{argmax}_y P^*(Y = y|X = x)$ to an instance x , where P^* denotes the true posterior distribution.

- 👉 For example, we can perform experiments with artificially generated data for which we have chosen the true distribution ourselves: this allows us to experimentally evaluate how close the performance of a model is to being Bayes-optimal.
- 👉 Alternatively, the derivation of a probabilistic learning method usually makes certain assumptions about the true distribution, which allows us to prove theoretically that the model will be Bayes-optimal provided these assumptions are met. For example, later on in this chapter we will state the conditions under which the basic linear classifier is Bayes-optimal.

Model selection I

The choice of a single model, often referred to as *model selection*, does not necessarily lead to Bayes-optimality – even if the model chosen is the one that performs best under the true distribution.

To illustrate this, let m^* be the best probability estimation tree we have learned from a sufficient amount of data. Using m^* we would predict $\operatorname{argmax}_y P(Y = y | M = m^*, X = x)$ for an instance x , where M is a random variable ranging over the model class m^* was chosen from.

However, these predictions are not necessarily Bayes-optimal since

$$\begin{aligned}
 P(Y|X = x) &= \sum_{m \in M} P(Y, M = m | X = x) && \text{by marginalising over } M \\
 &= \sum_{m \in M} P(Y|M = m, X = x)P(M = m|X = x) && \text{by the chain rule} \\
 &= \sum_{m \in M} P(Y|M = m, X = x)P(M = m) && \text{by independence of } M \text{ and } X
 \end{aligned}$$

Model selection II

Here, $P(M)$ can be interpreted as a posterior distribution over models after seeing the training data (the MAP model is therefore $m^* = \operatorname{argmax}_m P(M = m)$).

The final expression in the preceding derivation tells us to average the predictions of all models, weighted by their posterior probabilities.

Clearly, this distribution is only equal to $P(Y|M = m^*, X = x)$ if $P(M)$ is zero for all models other than m^* , i.e., if we have seen sufficient training data to rule out all but one remaining model. This is obviously unrealistic.

What's next?

9

Probabilistic models

- The normal distribution and its geometric interpretations
- Probabilistic models for categorical data
 - Using a naive Bayes model for classification
 - Training a naive Bayes model
- Discriminative learning by optimising conditional likelihood ★
- Probabilistic models with hidden variables
 - Expectation-Maximisation ★
 - Gaussian mixture models ★
- Compression-based models ★

The normal distribution

The univariate normal or Gaussian distribution has the following probability density function:

$$P(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) = \frac{1}{E} \exp\left(-\frac{1}{2} \left[\frac{x-\mu}{\sigma}\right]^2\right) = \frac{1}{E} \exp\left(-z^2/2\right), \quad E = \sqrt{2\pi}\sigma$$

The distribution has two parameters: μ , which is the mean or expected value, as well as the median (i.e., the point where the area under the density function is split in half) and the mode (i.e., the point where the density function reaches its maximum); and σ , which is the standard deviation and determines the width of the bell-shaped curve.

The *multivariate normal distribution* over d -vectors $\mathbf{x} = (x_1, \dots, x_d)^T \in \mathbb{R}^d$ is

$$P(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{E_d} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right), \quad E_d = (2\pi)^{d/2} \sqrt{|\boldsymbol{\Sigma}|}$$

Mixture model I

Suppose the values of $x \in \mathbb{R}$ follow a *mixture model*: i.e., each class has its own probability distribution (a *component* of the mixture model). We will assume a Gaussian mixture model, which means that the components of the mixture are both Gaussians. We thus have

$$P(x|\oplus) = \frac{1}{\sqrt{2\pi}\sigma^{\oplus}} \exp\left(-\frac{1}{2} \left[\frac{x - \mu^{\oplus}}{\sigma^{\oplus}}\right]^2\right) \quad P(x|\ominus) = \frac{1}{\sqrt{2\pi}\sigma^{\ominus}} \exp\left(-\frac{1}{2} \left[\frac{x - \mu^{\ominus}}{\sigma^{\ominus}}\right]^2\right)$$

where μ^{\oplus} and σ^{\oplus} are the mean and standard deviation for the positive class, and μ^{\ominus} and σ^{\ominus} are the mean and standard deviation for the negative class. This gives the following likelihood ratio:

$$\text{LR}(x) = \frac{P(x|\oplus)}{P(x|\ominus)} = \frac{\sigma^{\ominus}}{\sigma^{\oplus}} \exp\left(-\frac{1}{2} \left[\left(\frac{x - \mu^{\oplus}}{\sigma^{\oplus}}\right)^2 - \left(\frac{x - \mu^{\ominus}}{\sigma^{\ominus}}\right)^2 \right]\right)$$

Mixture model II

Let's first consider the case that both components have the same standard deviation, i.e., $\sigma^{\oplus} = \sigma^{\ominus} = \sigma$. We can then simplify the exponent in $\text{LR}(x)$ as follows:

$$\begin{aligned} -\frac{1}{2\sigma^2} [(x - \mu^{\oplus})^2 - (x - \mu^{\ominus})^2] &= -\frac{1}{2\sigma^2} [x^2 - 2\mu^{\oplus}x + \mu^{\oplus 2} - (x^2 - 2\mu^{\ominus}x + \mu^{\ominus 2})] \\ &= -\frac{1}{2\sigma^2} [-2(\mu^{\oplus} - \mu^{\ominus})x + (\mu^{\oplus 2} - \mu^{\ominus 2})] \\ &= \frac{\mu^{\oplus} - \mu^{\ominus}}{\sigma^2} \left[x - \frac{\mu^{\oplus} + \mu^{\ominus}}{2} \right] \end{aligned}$$

The likelihood ratio can thus be written as $\text{LR}(x) = \exp(\gamma(x - \mu))$, with two parameters: $\gamma = (\mu^{\oplus} - \mu^{\ominus})/\sigma^2$ is the difference between the means in proportion to the variance, and $\mu = (\mu^{\oplus} + \mu^{\ominus})/2$ is the midpoint between the two class means. It follows that the maximum-likelihood decision threshold (the value of x such that $\text{LR}(x) = 1$) is $x_{\text{ML}} = \mu$.



Example 9.2, p.268

Mixture model with unequal variances

If $\sigma^{\oplus} \neq \sigma^{\ominus}$, the x^2 terms in $\text{LR}(x)$ do not cancel. This results in two decision boundaries and a non-contiguous decision region for one of the classes.

👉 Suppose $\mu^{\oplus} = 1$, $\mu^{\ominus} = 2$ and $\sigma^{\ominus} = 2\sigma^{\oplus} = 2$, then

$$\text{LR}(x) = 2 \exp\left(-\frac{[(x-1)^2 - (x-2)^2/4]}{2}\right) = 2 \exp(3x^2/8).$$

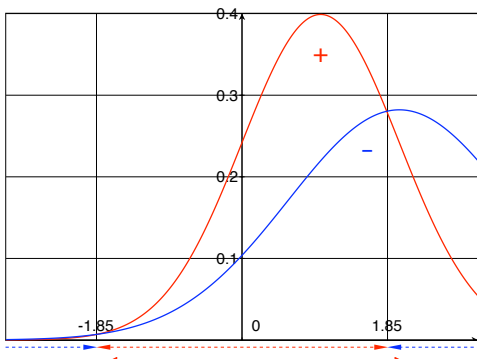
👉 It follows that the ML decision boundaries are $x = \pm(8/3) \ln 2 = \pm 1.85$. As can be observed in [Figure 9.2](#), these are the points where the two Gaussians cross.

👉 In contrast, if $\sigma^{\ominus} = \sigma^{\oplus}$ then we get a single ML decision boundary at $x = 1.5$.



Figure 9.2, p.268

Mixture model with unequal variances



If positive examples are drawn from a Gaussian with mean and standard deviation 1 and negatives from a Gaussian with mean and standard deviation 2, then the two distributions cross at $x = \pm 1.85$. This means that the maximum-likelihood region for positives is the closed interval $[-1.85, 1.85]$, and hence the negative region is non-contiguous.



Example 9.3, p.269

Bivariate Gaussian mixture

Throughout the example we assume $\mu_1^{\oplus} = \mu_2^{\oplus} = 1$ and $\mu_1^{\ominus} = \mu_2^{\ominus} = -1$.

(i) If all variances are 1 and both correlations are 0, then the ML decision boundary is given by

$$(x_1 - 1)^2 + (x_2 - 1)^2 - (x_1 + 1)^2 - (x_2 + 1)^2 = -2x_1 - 2x_2 - 2x_1 - 2x_2 = 0, \text{ i.e., } x_1 + x_2 = 0 \text{ (Figure 9.3 (left)).}$$

(ii) If $\sigma_1^{\oplus} = \sigma_1^{\ominus} = 1$, $\sigma_2^{\oplus} = \sigma_2^{\ominus} = \sqrt{2}$ and $\rho^{\oplus} = \rho^{\ominus} = \sqrt{2}/2$, then the ML decision boundary is $(x_1 - 1)^2 + (x_2 - 1)^2/2 - \sqrt{2}(x_1 - 1)(x_2 - 1)/\sqrt{2} - (x_1 + 1)^2 - (x_2 + 1)^2/2 + \sqrt{2}(x_1 + 1)(x_2 + 1)/\sqrt{2} = -2x_1 = 0$ (Figure 9.3 (middle)).

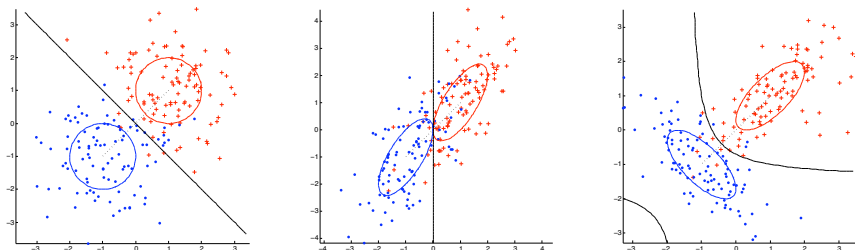
(iii) If all variances are 1 and $\rho^{\oplus} = -\rho^{\ominus} = \rho$, then the ML decision boundary is given by

$$(x_1 - 1)^2 + (x_2 - 1)^2 - 2\rho(x_1 - 1)(x_2 - 1) - (x_1 + 1)^2 - (x_2 + 1)^2 - 2\rho(x_1 + 1)(x_2 + 1) = -4x_1 - 4x_2 - 4\rho x_1 x_2 - 4\rho = 0, \text{ i.e., } x_1 + x_2 + \rho x_1 x_2 + \rho = 0, \text{ which is a hyperbole. Figure 9.3 (right) illustrates this for } \rho = 0.7.$$



Figure 9.3, p.269

Bivariate Gaussian mixture



(left) If the features are uncorrelated and have the same variance, maximum-likelihood classification leads to the basic linear classifier, whose decision boundary is orthogonal to the line connecting the means. **(middle)** As long as the per-class covariance matrices are identical, the Bayes-optimal decision boundary is linear – if we were to decorrelate the features by rotation and scaling, we would again obtain the basic linear classifier. **(right)** Unequal covariance matrices lead to hyperbolic decision boundaries, which means that one of the decision regions is non-contiguous.

When is the basic linear classifier Bayes-optimal? I

The general form of the likelihood ratio can be derived as

$$\text{LR}(\mathbf{x}) = \sqrt{\frac{|\boldsymbol{\Sigma}^{\ominus}|}{|\boldsymbol{\Sigma}^{\oplus}|}} \exp\left(-\frac{1}{2} \left[(\mathbf{x} - \boldsymbol{\mu}^{\oplus})^T (\boldsymbol{\Sigma}^{\oplus})^{-1} (\mathbf{x} - \boldsymbol{\mu}^{\oplus}) - (\mathbf{x} - \boldsymbol{\mu}^{\ominus})^T (\boldsymbol{\Sigma}^{\ominus})^{-1} (\mathbf{x} - \boldsymbol{\mu}^{\ominus}) \right]\right)$$

where $\boldsymbol{\mu}^{\oplus}$ and $\boldsymbol{\mu}^{\ominus}$ are the class means, and $\boldsymbol{\Sigma}^{\oplus}$ and $\boldsymbol{\Sigma}^{\ominus}$ are the covariance matrices for each class.

Assume that $\boldsymbol{\Sigma}^{\oplus} = \boldsymbol{\Sigma}^{\ominus} = \mathbf{I}$ (i.e., in each class the features are uncorrelated and have unit variance), then we have

$$\begin{aligned} \text{LR}(\mathbf{x}) &= \exp\left(-\frac{1}{2} \left[(\mathbf{x} - \boldsymbol{\mu}^{\oplus})^T (\mathbf{x} - \boldsymbol{\mu}^{\oplus}) - (\mathbf{x} - \boldsymbol{\mu}^{\ominus})^T (\mathbf{x} - \boldsymbol{\mu}^{\ominus}) \right]\right) \\ &= \exp\left(-\frac{1}{2} \left[\|\mathbf{x} - \boldsymbol{\mu}^{\oplus}\|^2 - \|\mathbf{x} - \boldsymbol{\mu}^{\ominus}\|^2 \right]\right) \end{aligned}$$

When is the basic linear classifier Bayes-optimal? II

It follows that $\text{LR}(\mathbf{x}) = 1$ for any \mathbf{x} equidistant from μ^{\oplus} and μ^{\ominus} . But this means that the ML decision boundary is a straight line at equal distances from the class means – in which we recognise our old friend, the basic linear classifier! In other words, for uncorrelated, unit-variance Gaussian features, the basic linear classifier is Bayes-optimal.

More generally, as long as the per-class covariance matrices are equal, the ML decision boundary will be linear, intersecting $\mu^{\oplus} - \mu^{\ominus}$ in the middle, but not at right angles if the features are correlated. This means that the basic linear classifier is only Bayes-optimal in this case if we first decorrelate and normalise the features.

With non-equal class covariances the decision boundary will be hyperbolic. So, the three cases in [Figure 9.3](#) generalise to the multivariate case.

★ From distances to probabilities, and back

The multivariate normal distribution essentially translates distances into probabilities ($\text{Dis}_M(\mathbf{x}, \mathbf{y}|\Sigma) = \sqrt{(\mathbf{x} - \mathbf{y})^T \Sigma^{-1} (\mathbf{x} - \mathbf{y})}$ denotes the Mahalanobis distance introduced in [Chapter 8](#)):

$$P(\mathbf{x}|\boldsymbol{\mu}, \Sigma) = \frac{1}{E_d} \exp\left(-\frac{1}{2} (\text{Dis}_M(\mathbf{x}, \boldsymbol{\mu}|\Sigma))^2\right)$$

Conversely, we see that the negative logarithm of the Gaussian likelihood can be interpreted as a squared distance:

$$-\ln P(\mathbf{x}|\boldsymbol{\mu}, \Sigma) = \ln E_d + \frac{1}{2} (\text{Dis}_M(\mathbf{x}, \boldsymbol{\mu}|\Sigma))^2$$

The intuition is that the logarithm transforms the multiplicative probability scale into an additive scale (which, in the case of Gaussian distributions, corresponds to a squared distance).

Since additive scales are often easier to handle, log-likelihoods are a common concept in statistics.

★ Maximum-likelihood estimation

Suppose we want to estimate the mean μ of a multivariate Gaussian distribution with given covariance matrix Σ from a set of data points X . The principle of *maximum-likelihood estimation* states that we should find the value of μ that maximises the joint likelihood of X . Assuming that the elements of X were independently sampled, the joint likelihood decomposes into a product over the individual data points in X , and the maximum-likelihood estimate can be found as follows:

$$\begin{aligned}
 \hat{\mu} &= \operatorname{argmax}_{\mu} \prod_{\mathbf{x} \in X} P(\mathbf{x} | \mu, \Sigma) \\
 &= \operatorname{argmax}_{\mu} \prod_{\mathbf{x} \in X} \frac{1}{E_d} \exp\left(-\frac{1}{2} (\operatorname{Dis}_M(\mathbf{x}, \mu | \Sigma))^2\right) && \text{using Equation 9.4} \\
 &= \operatorname{argmin}_{\mu} \sum_{\mathbf{x} \in X} \left[\ln E_d + \frac{1}{2} (\operatorname{Dis}_M(\mathbf{x}, \mu | \Sigma))^2 \right] && \text{taking negative logarithms} \\
 &= \operatorname{argmin}_{\mu} \sum_{\mathbf{x} \in X} (\operatorname{Dis}_M(\mathbf{x}, \mu | \Sigma))^2 && \text{dropping constant term and factor}
 \end{aligned}$$

We thus find that the maximum-likelihood estimate of the mean of a multivariate distribution is the point that minimises the total squared Mahalanobis distance to all points in X .

What's next?

9

Probabilistic models

- The normal distribution and its geometric interpretations
- **Probabilistic models for categorical data**
 - Using a naive Bayes model for classification
 - Training a naive Bayes model
- Discriminative learning by optimising conditional likelihood ★
- Probabilistic models with hidden variables
 - Expectation-Maximisation ★
 - Gaussian mixture models ★
- Compression-based models ★

Categorical random variables I

Categorical variables or features (also called discrete or nominal) are ubiquitous in machine learning.

- Perhaps the most common form of the Bernoulli distribution models whether or not a word occurs in a document. That is, for the i -th word in our vocabulary we have a random variable X_i governed by a Bernoulli distribution. The joint distribution over the *bit vector* $X = (X_1, \dots, X_k)$ is called a *multivariate Bernoulli distribution*.
- Variables with more than two outcomes are also common: for example, every word position in an e-mail corresponds to a categorical variable with k outcomes, where k is the size of the vocabulary. The multinomial distribution manifests itself as a *count vector*: a histogram of the number of occurrences of all vocabulary words in a document. This establishes an alternative way of modelling text documents that allows the number of occurrences of a word to influence the classification of a document.

Categorical random variables II

Both these document models are in common use. Despite their differences, they both assume independence between word occurrences, generally referred to as the *naive Bayes assumption*.

- 👉 In the multinomial document model, this follows from the very use of the multinomial distribution, which assumes that words at different word positions are drawn independently from the same categorical distribution.
- 👉 In the multivariate Bernoulli model we assume that the bits in a bit vector are statistically independent, which allows us to compute the joint probability of a particular bit vector (x_1, \dots, x_k) as the product of the probabilities of each component $P(X_i = x_i)$.
- 👉 In practice, such word independence assumptions are often not true: if we know that an e-mail contains the word 'Viagra', we can be quite sure that it will also contain the word 'pill'. Violated independence assumptions violate the quality of probability estimates but may still allow good classification performance.

Probabilistic decision rules

We have chosen one of the possible distributions to model our data X as coming from either class.

- ☞ The more different $P(X|Y = \text{spam})$ and $P(X|Y = \text{ham})$ are, the more useful the features X are for classification.
- ☞ Thus, for a specific e-mail x we calculate both $P(X = x|Y = \text{spam})$ and $P(X = x|Y = \text{ham})$, and apply one of several possible decision rules:

maximum likelihood (ML)

– predict $\arg \max_y P(X = x|Y = y)$;

maximum a posteriori (MAP)

– predict $\arg \max_y P(X = x|Y = y)P(Y = y)$;

recalibrated likelihood

– predict $\arg \max_y w_y P(X = x|Y = y)$.

The relation between the first two decision rules is that ML classification is equivalent to MAP classification with a uniform class distribution. The third decision rule generalises the first two in that it replaces the class distribution with a set of weights learned from the data.



Example 9.4, p.276

Prediction using a naive Bayes model I

Suppose our vocabulary contains three words a , b and c , and we use a multivariate Bernoulli model for our e-mails, with parameters

$$\theta^{\oplus} = (0.5, 0.67, 0.33) \qquad \theta^{\ominus} = (0.67, 0.33, 0.33)$$

This means, for example, that the presence of b is twice as likely in spam (+), compared with ham.

The e-mail to be classified contains words a and b but not c , and hence is described by the bit vector $\mathbf{x} = (1, 1, 0)$. We obtain likelihoods

$$P(\mathbf{x}|\oplus) = 0.5 \cdot 0.67 \cdot (1 - 0.33) = 0.222 \qquad P(\mathbf{x}|\ominus) = 0.67 \cdot 0.33 \cdot (1 - 0.33) = 0.148$$

The ML classification of \mathbf{x} is thus spam.



Example 9.4, p.276

Prediction using a naive Bayes model II

In the case of two classes it is often convenient to work with likelihood ratios and odds.

☞ The likelihood ratio can be calculated as

$$\frac{P(\mathbf{x}|\oplus)}{P(\mathbf{x}|\ominus)} = \frac{0.5}{0.67} \frac{0.67}{0.33} \frac{1-0.33}{1-0.33} = 3/2 > 1$$

☞ This means that the MAP classification of \mathbf{x} is also spam if the prior odds are more than $2/3$, but ham if they are less than that.

☞ For example, with 33% spam and 67% ham the prior odds are $\frac{P(\oplus)}{P(\ominus)} = \frac{0.33}{0.67} = 1/2$, resulting in a posterior odds of

$$\frac{P(\oplus|\mathbf{x})}{P(\ominus|\mathbf{x})} = \frac{P(\mathbf{x}|\oplus)}{P(\mathbf{x}|\ominus)} \frac{P(\oplus)}{P(\ominus)} = 3/2 \cdot 1/2 = 3/4 < 1$$

In this case the likelihood ratio for \mathbf{x} is not strong enough to push the decision away from the prior.



Example 9.4, p.276

Prediction using a naive Bayes model III

Alternatively, we can employ a multinomial model. The parameters of a multinomial establish a distribution over the words in the vocabulary, say

$$\theta^{\oplus} = (0.3, 0.5, 0.2) \quad \theta^{\ominus} = (0.6, 0.2, 0.2)$$

The e-mail to be classified contains three occurrences of word a , one single occurrence of word b and no occurrences of word c , and hence is described by the count vector $\mathbf{x} = (3, 1, 0)$. The total number of vocabulary word occurrences is $n = 4$. We obtain likelihoods

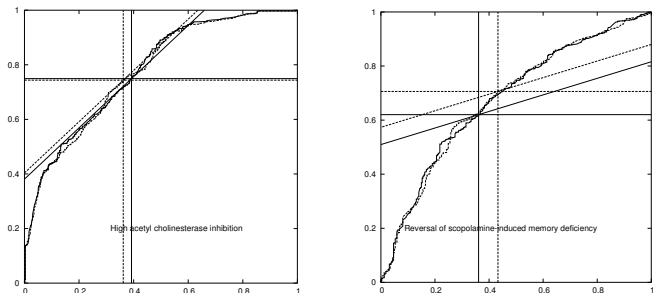
$$P(\mathbf{x}|\oplus) = 4! \frac{0.3^3}{3!} \frac{0.5^1}{1!} \frac{0.2^0}{0!} = 0.054 \quad P(\mathbf{x}|\ominus) = 4! \frac{0.6^3}{3!} \frac{0.2^1}{1!} \frac{0.2^0}{0!} = 0.1728$$

The likelihood ratio is $\left(\frac{0.3}{0.6}\right)^3 \left(\frac{0.5}{0.2}\right)^1 \left(\frac{0.2}{0.2}\right)^0 = 5/16$. The ML classification of \mathbf{x} is thus ham, the opposite of the multivariate Bernoulli model. This is mainly because of the three occurrences of word a , which provide strong evidence for ham.



Figure 9.4, p.278

Uncalibrated threshold of naive Bayes



(left) ROC curves produced by two naive Bayes classifiers. Both models have similar ranking performance and yield almost the same – more or less optimal – MAP decision threshold. **(right)** On a different data set from the same domain, the multinomial model's MAP threshold is slightly better, hinting at somewhat better calibrated probability estimates. But since the slope of the accuracy isometrics indicates that there are about four positives for every negative, the optimal decision rule is in fact to always predict positive.

Important point to remember

An often overlooked consequence of having uncalibrated probability estimates such as those produced by naive Bayes is that both the ML and MAP decision rules become inadequate.



Table 9.1, p.280

Training data for naive Bayes

E-mail	# <i>a</i>	# <i>b</i>	# <i>c</i>	Class
e_1	0	3	0	+
e_2	0	3	3	+
e_3	3	0	0	+
e_4	2	3	0	+
e_5	4	3	0	-
e_6	4	0	3	-
e_7	3	0	0	-
e_8	0	0	0	-

E-mail	<i>a</i> ?	<i>b</i> ?	<i>c</i> ?	Class
e_1	0	1	0	+
e_2	0	1	1	+
e_3	1	0	0	+
e_4	1	1	0	+
e_5	1	1	0	-
e_6	1	0	1	-
e_7	1	0	0	-
e_8	0	0	0	-

(left) A small e-mail data set described by count vectors. **(right)** The same data set described by bit vectors.



Example 9.5, p.279

Training a naive Bayes model I

Consider the following e-mails consisting of five words a , b , c , d , e :

$e_1: b d e b b d e$

$e_2: b c e b b d d e c c$

$e_3: a d a d e a e e$

$e_4: b a d b e d a b$

$e_5: a b a b a b a e d$

$e_6: a c a c a c a e d$

$e_7: e a e d a e a$

$e_8: d e d e d$

We are told that the e-mails on the left are spam and those on the right are ham, and so we use them as a small training set to train our Bayesian classifier.

- ☞ First, we decide that d and e are so-called *stop words* that are too common to convey class information.
- ☞ The remaining words, a , b and c , constitute our vocabulary.



Training a naive Bayes model II

For the multinomial model, we represent each e-mail as a count vector, as in Table 9.1 (left).

- In order to estimate the parameters of the multinomial, we sum up the count vectors for each class, which gives $(5, 9, 3)$ for spam and $(11, 3, 3)$ for ham.
- To smooth these probability estimates we add one pseudo-count for each vocabulary word, which brings the total number of occurrences of vocabulary words to 20 for each class.
- The estimated parameter vectors are thus
$$\hat{\theta}^{\oplus} = (6/20, 10/20, 4/20) = (0.3, 0.5, 0.2) \text{ for spam and}$$
$$\hat{\theta}^{\ominus} = (12/20, 4/20, 4/20) = (0.6, 0.2, 0.2) \text{ for ham.}$$



Training a naive Bayes model III

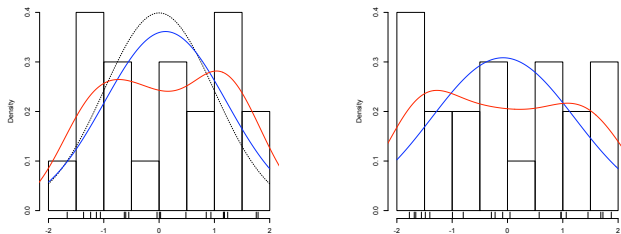
In the multivariate Bernoulli model e-mails are represented by bit vectors, as in Table 9.1 (right).

- Adding the bit vectors for each class results in (2, 3, 1) for spam and (3, 1, 1) for ham.
- Each count is to be divided by the number of documents in a class, in order to get an estimate of the probability of a document containing a particular vocabulary word.
- Probability smoothing now means adding two pseudo-documents, one containing each word and one containing none of them.
- This results in the estimated parameter vectors
 $\hat{\theta}^{\oplus} = (3/6, 4/6, 2/6) = (0.5, 0.67, 0.33)$ for spam and
 $\hat{\theta}^{\ominus} = (4/6, 2/6, 2/6) = (0.67, 0.33, 0.33)$ for ham.



Figure 9.5, p.282

Density estimation



(left) Examples of three density estimators on 20 points sampled from a normal distribution with zero mean and unit variance (dotted line). A histogram is a simple non-parametric method which employs a fixed number of equal-width intervals. A **kernel density estimator (in red)** applies interpolation to obtain a smooth density function. The **solid bell curve (in blue)** is obtained by estimating the sample mean and variance, assuming the true distribution is normal. **(right)** Here, the 20 points are sampled uniformly from $[-2, 2]$, and the non-parametric methods generally do better.

What's next?

9

Probabilistic models

- The normal distribution and its geometric interpretations
- Probabilistic models for categorical data
 - Using a naive Bayes model for classification
 - Training a naive Bayes model
- **Discriminative learning by optimising conditional likelihood ★**
- Probabilistic models with hidden variables
 - Expectation-Maximisation ★
 - Gaussian mixture models ★
- Compression-based models ★

★ Logistic regression I

The logistic regression model is related to the logistically calibrated linear classifier:

$$\hat{p}(\mathbf{x}) = \frac{\exp(\mathbf{w} \cdot \mathbf{x} - t)}{\exp(\mathbf{w} \cdot \mathbf{x} - t) + 1} = \frac{1}{1 + \exp(-(\mathbf{w} \cdot \mathbf{x} - t))}$$

Assuming the class labels are $y = 1$ for positives and $y = 0$ for negatives, this defines a Bernoulli distribution for each training example:

$$P(y_i | \mathbf{x}_i) = \hat{p}(\mathbf{x}_i)^{y_i} (1 - \hat{p}(\mathbf{x}_i))^{(1-y_i)}$$

It is important to note that the parameters of these Bernoulli distributions are linked through \mathbf{w} and t , and consequently there is one parameter for every feature dimension, rather than for every training instance.

★ Logistic regression II

The likelihood function is

$$\mathbf{CL}(\mathbf{w}, t) = \prod_i P(y_i | \mathbf{x}_i) = \prod_i \hat{p}(\mathbf{x}_i)^{y_i} (1 - \hat{p}(\mathbf{x}_i))^{(1-y_i)}$$

This is called *conditional likelihood* to stress that it gives us the *conditional* probability $P(y_i | \mathbf{x}_i)$ rather than $P(\mathbf{x}_i)$ as in a generative model.

- 👉 Notice that our use of the product requires the assumption that the y -values are independent given \mathbf{x} ; but this is not nearly as strong as the naive Bayes assumption of \mathbf{x} being independent within each class.
- 👉 As usual, the logarithm of the likelihood function is easier to work with:

$$\mathbf{LCL}(\mathbf{w}, t) = \sum_i y_i \ln \hat{p}(\mathbf{x}_i) + (1 - y_i) \ln(1 - \hat{p}(\mathbf{x}_i)) = \sum_{\mathbf{x}^{\oplus} \in Tr^{\oplus}} \ln \hat{p}(\mathbf{x}^{\oplus}) + \sum_{\mathbf{x}^{\ominus} \in Tr^{\ominus}} \ln(1 - \hat{p}(\mathbf{x}^{\ominus}))$$

★ Logistic regression III

We want to maximise the log-conditional likelihood with respect to these parameters, which means that all partial derivatives must be zero:

$$\begin{aligned}\nabla_{\mathbf{w}} \text{LCL}(\mathbf{w}, t) &= \mathbf{0} \\ \frac{\partial}{\partial t} \text{LCL}(\mathbf{w}, t) &= 0\end{aligned}$$

It turns out that the partial derivative of **LCL** with respect to t has a simple form:

$$\begin{aligned}\frac{\partial}{\partial t} \text{LCL}(\mathbf{w}, t) &= \sum_{\mathbf{x}^{\oplus} \in Tr^{\oplus}} (\hat{p}(\mathbf{x}) - 1) + \sum_{\mathbf{x}^{\ominus} \in Tr^{\ominus}} \hat{p}(\mathbf{x}^{\ominus}) \\ &= \sum_{\mathbf{x}_i \in Tr} (\hat{p}(\mathbf{x}_i) - y_i)\end{aligned}$$

☞ For the optimal solution this partial derivative is zero. What this means is that, on average, the predicted probability should be equal to the proportion of positives *pos*.

★ Logistic regression IV

- 👉 Notice that grouping models such as probability estimating trees have this property by construction, as they set the predicted probability equal to the empirical probability in a segment.
- 👉 A very similar derivation leads to

$$\frac{\partial}{\partial w_j} \text{LCL}(\mathbf{w}, t) = \sum_{\mathbf{x}_i \in Tr} (y_i - \hat{p}(\mathbf{x}_i)) x_{ij}$$

Setting this partial derivative to zero expresses another, feature-wise calibration property.

★ Training a logistic regression model

In order to train a logistic regression model we need to find

$$\mathbf{w}^*, t^* = \underset{\mathbf{w}, t}{\operatorname{argmax}} \text{CL}(\mathbf{w}, t) = \underset{\mathbf{w}, t}{\operatorname{argmax}} \text{LCL}(\mathbf{w}, t)$$

This can be shown to be a convex optimisation problem, which means that there is only one maximum. A range of optimisation techniques can be applied. One simple approach is inspired by the perceptron algorithm and iterates over examples, using the following update rule:

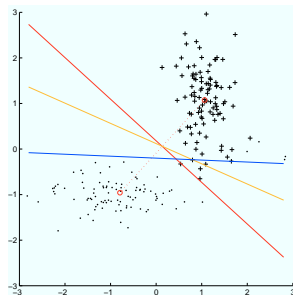
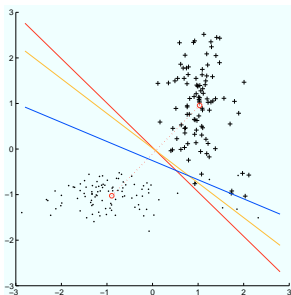
$$\mathbf{w} = \mathbf{w} + \eta(y_i - \hat{p}_i)\mathbf{x}_i$$

where η is the learning rate. Notice the relationship with the partial derivative in [Equation 9.5](#). Essentially, we are using single examples to approximate the direction of steepest ascent.



Figure 9.6, p.283

★ Logistic regression compared



(left) On this data set, **logistic regression (in blue)** outperforms the **basic linear classifier (in red)** and the **least squares classifier (in orange)** because the latter two are more sensitive to the shape of the classes, while logistic regression concentrates on where the classes overlap. **(right)** On this slightly different set of points, **logistic regression** is outperformed by the other two methods because it concentrates too much on tracking the transition from mostly positive to mostly negative.



★ Univariate logistic regression

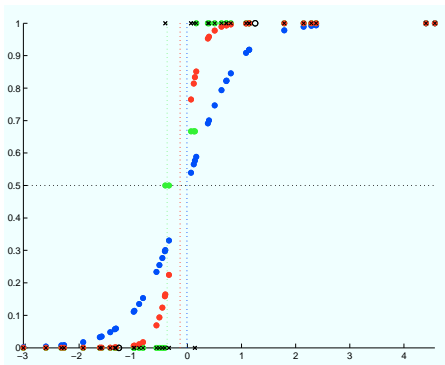
Consider the data in [Figure 9.7](#) with 20 points in each class.

- ➡ Although both classes were generated from normal distributions, class overlap in this particular sample is less than what could be expected on the basis of the class means.
- ➡ Logistic regression is able to take advantage of this and gives a much steeper sigmoid than the basic linear classifier with logistic calibration which is entirely formulated in terms of class means and variance.
- ➡ Also shown are the probability estimates obtained from the convex hull of the ROC curve; this calibration procedure is non-parametric and hence better able to detect the limited class overlap.



Figure 9.7, p.286

★ Univariate logistic regression



Logistic regression (in red) compared with probability estimates obtained by logistic calibration (in blue) and isotonic calibration (in green); the latter two are applied to the basic linear classifier (estimated class means are indicated by circles). The corresponding three decision boundaries are shown as vertical dotted lines.

What's next?

9

Probabilistic models

- The normal distribution and its geometric interpretations
- Probabilistic models for categorical data
 - Using a naive Bayes model for classification
 - Training a naive Bayes model
- Discriminative learning by optimising conditional likelihood ★
- **Probabilistic models with hidden variables**
 - Expectation-Maximisation ★
 - Gaussian mixture models ★
- Compression-based models ★

Hidden variables I

Suppose you are dealing with a four-class classification problem with classes A , B , C and D .

- 👉 If you have a sufficiently large and representative training sample of size n , you can use the relative frequencies in the sample n_A, \dots, n_D to estimate the class prior $\hat{p}_A = n_A/n, \dots, \hat{p}_D = n_D/n$.
- 👉 Conversely, if you know the prior and want to know the most likely class distribution in a random sample of n instances, you would use the prior to calculate expected values $\mathbb{E}[n_A] = p_A \cdot n, \dots, \mathbb{E}[n_D] = p_D \cdot n$.
- 👉 So, complete knowledge of one allows us to estimate or infer the other. However, sometimes we have a bit of knowledge about both.

Hidden variables II

For example, we may know that $p_A = 1/2$ and that C is twice as likely as B , without knowing the complete prior. And we may know that the sample we saw last week was evenly split between $A \cup B$ and $C \cup D$, and that C and D were equally large, but we can't remember the size of A and B separately. What should we do?

- ➡ Formalising what we know about the prior, we have $p_A = 1/2$; $p_B = \beta$, as yet unknown; $p_C = 2\beta$, since it is twice p_B ; and $p_D = 1/2 - 3\beta$, since the four cases need to add up to 1.
- ➡ Furthermore: $n_A + n_B = a + b = s$, $n_C = c$ and $n_D = d$, with s , c and d known. We want to infer a , b and β : however, it seems we are stuck in a chicken-and-egg problem.

Hidden variables III

If we knew β we would have full knowledge about the prior and we could use that to infer expected values for a and b :

$$\frac{\mathbb{E}[a]}{\mathbb{E}[b]} = \frac{1/2}{\beta} \qquad \mathbb{E}[a] + \mathbb{E}[b] = s$$

from which we could derive

$$\mathbb{E}[a] = \frac{1}{1+2\beta} s \qquad \mathbb{E}[b] = \frac{2\beta}{1+2\beta} s$$

So, for example, if $s = 20$ and $\beta = 1/10$, then $\mathbb{E}[a] = 16\frac{2}{3}$ and $\mathbb{E}[b] = 3\frac{1}{3}$.

Hidden variables IV

Conversely, if we knew a and b , then we could estimate β by maximum-likelihood estimation, using a multinomial distribution for a , b , c and d :

$$P(a, b, c, d|\beta) = K(1/2)^a \beta^b (2\beta)^c (1/2 - 3\beta)^d$$

$$\ln P(a, b, c, d|\beta) = \ln K + a \ln(1/2) + b \ln \beta + c \ln(2\beta) + d \ln(1/2 - 3\beta)$$

Here, K is a combinatorial constant that doesn't affect the value of β which maximises the likelihood. Taking the partial derivative with respect to β gives

$$\frac{\partial}{\partial \beta} \ln P(a, b, c, d|\beta) = \frac{b}{\beta} + \frac{2c}{2\beta} - \frac{3d}{1/2 - 3\beta}$$

Setting to 0 and solving for β finally gives

$$\hat{\beta} = \frac{b + c}{6(b + c + d)}$$

So, for example, if $b = 5$ and $c = d = 10$, then $\hat{\beta} = 1/10$.

Hidden variables V

The way out of this chicken-and-egg problem is by *Expectation-Maximisation*, which iterates the following two steps: (i) calculate an expected value of the missing frequencies a and b from an assumed or previously estimated value of the parameter β ; and (ii) calculate a maximum-likelihood estimate of the parameter β from assumed or expected values of the missing frequencies a and b . These two steps are iterated until a stationary configuration is reached.

- ☞ So, if we start with $a = 15$, $b = 5$ and $c = d = 10$, then we have just seen that $\hat{\beta} = 1/10$. Plugging this into the equations for $\mathbb{E}[a]$ and $\mathbb{E}[b]$ gives us $\mathbb{E}[a] = 16\frac{2}{3}$ and $\mathbb{E}[b] = 3\frac{1}{3}$. These give a new maximum-likelihood estimate $\hat{\beta} = 2/21$, which in turn gives $\mathbb{E}[a] = 16.8$ and $\mathbb{E}[b] = 3.2$, and so on.
- ☞ A stationary configuration with $\beta = 0.0948$, $a = 16.813$ and $b = 3.187$ is reached in fewer than 10 iterations.
- ☞ In this simple case this is a global optimum that is reached regardless of the starting point, essentially because the relationship between b and β is monotonic. However, this is not normally the case.

★ The general form of Expectation-Maximisation I

The problem that we have just discussed is an example of a problem with missing data, where the full data Y separates into observed variables X and *hidden variables* Z (also called *latent variables*). We also have model parameter(s) θ , which is β in the example. Denote the estimate of θ in the t -th iteration as θ^t . We have two relevant quantities:

- 👉 the expectation $\mathbb{E}[Z|X, \theta^t]$ of the hidden variables given the observed variables and the current estimate of the parameters (so in the previous example the expectations of a and b depend on s and β);
- 👉 the likelihood $P(Y|\theta)$, which is used to find the maximising value of θ .

In the likelihood function we need values for $Y = X \cup Z$. We obviously use the observed values for X , but we need to use previously calculated expectations for Z . This means that we really want to maximise $P(X \cup \mathbb{E}[Z|X, \theta^t] | \theta)$, or equivalently, the logarithm of that function.

★ The general form of Expectation-Maximisation II

We now make the assumption that the logarithm of the likelihood function is linear in Y . For any linear function f , $f(\mathbb{E}[Z]) = \mathbb{E}[f(Z)]$ and thus we can bring the expectation outside in our objective function:

$$\ln P(X \cup \mathbb{E}[Z|X, \theta^t] | \theta) = \mathbb{E}[\ln P(X \cup Z | \theta) | X, \theta^t] = \mathbb{E}[\ln P(Y | \theta) | X, \theta^t]$$

This last expression is usually denoted as $Q(\theta | \theta^t)$, as it essentially tells us how to calculate the next value of θ from the current one:

$$\theta^{t+1} = \arg \max_{\theta} Q(\theta | \theta^t) = \arg \max_{\theta} \mathbb{E}[\ln P(Y | \theta) | X, \theta^t]$$

Similar to the example above, we iterate over assigning an expected value to the hidden variables given our current estimates of the parameters, and re-estimating the parameters from these updated expectations, until a stationary configuration is reached. We can start the iteration by initialising either the parameters or the hidden variables in some way.

★ EM applied to Gaussian mixtures I

A common application of Expectation-Maximisation is to estimate the parameters of a *Gaussian mixture model* from data.

- ☞ In such a model the data points are generated by K normal distributions, each with their own mean μ_j and covariance matrix Σ_j , and the proportion of points coming from each Gaussian is governed by a prior $\tau = (\tau_1, \dots, \tau_K)$.
- ☞ If each data point in a sample were labelled with the index of the Gaussian it came from this would be a straightforward classification problem, which could be solved easily by estimating each Gaussian's μ_j and Σ_j separately from the data points belonging to class j .
- ☞ However, we are now considering the much harder predictive clustering problem in which the class labels are hidden and need to be reconstructed from the observed feature values.

★ EM applied to Gaussian mixtures II

- A convenient way to model this is to have for each data point \mathbf{x}_i a Boolean vector $\mathbf{z}_i = (z_{i1}, \dots, z_{iK})$ such that exactly one bit z_{ij} is set to 1 and the rest set to 0, signalling that the i -th data point comes from the j -th Gaussian.
- Using this notation we can adapt the expression for the multivariate normal distribution to obtain a general expression for a Gaussian mixture model:

$$P(\mathbf{x}_i, \mathbf{z}_i | \theta) = \sum_{j=1}^K z_{ij} \tau_j \frac{1}{(2\pi)^{d/2} \sqrt{|\Sigma_j|}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_j)^T \Sigma_j^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_j)\right)$$

Here, θ collects all the parameters τ , $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$ and $\Sigma_1, \dots, \Sigma_K$.

- The interpretation as a generative model is as follows: we first randomly select a Gaussian using the prior τ , and then we invoke the corresponding Gaussian using the indicator variables z_{ij} .

★ EM applied to Gaussian mixtures III

In order to apply Expectation-Maximisation we form the Q function:

$$\begin{aligned}
 Q(\theta|\theta^t) &= \mathbb{E}[\ln P(\mathbf{X} \cup \mathbf{Z}|\theta) | \mathbf{X}, \theta^t] = \mathbb{E}\left[\ln \prod_{i=1}^n P(\mathbf{x}_i \cup \mathbf{z}_i|\theta) \middle| \mathbf{X}, \theta^t\right] = \mathbb{E}\left[\sum_{i=1}^n \ln P(\mathbf{x}_i \cup \mathbf{z}_i|\theta) \middle| \mathbf{X}, \theta^t\right] \\
 &= \mathbb{E}\left[\sum_{i=1}^n \ln \sum_{j=1}^K z_{ij} \tau_j \frac{1}{(2\pi)^{d/2} \sqrt{|\Sigma_j|}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mu_j)^T \Sigma_j^{-1} (\mathbf{x}_i - \mu_j)\right) \middle| \mathbf{X}, \theta^t\right] = \dots \\
 &= \sum_{i=1}^n \sum_{j=1}^K \mathbb{E}\left[z_{ij} \middle| \mathbf{X}, \theta^t\right] \left(\ln \tau_j - \frac{d}{2} \ln(2\pi) - \frac{1}{2} \ln|\Sigma_j| - \frac{1}{2} (\mathbf{x}_i - \mu_j)^T \Sigma_j^{-1} (\mathbf{x}_i - \mu_j)\right)
 \end{aligned}$$

The last line shows the Q function in the desired form, involving on the one hand expectations over the hidden variables conditioned on the observable data \mathbf{X} and the previously estimated parameters θ^t , and on the other hand expressions in θ that allow us to find θ^{t+1} by maximisation.

★ EM applied to Gaussian mixtures IV

The Expectation step of the EM algorithm is thus the calculation of the expected values of the indicator variables $\mathbb{E}[z_{ij} | \mathbf{X}, \theta^t]$.

- ☞ The hard cluster assignment of K -means is changed into a soft assignment – one of the ways in which Gaussian mixture models generalise K -means.
- ☞ Now, suppose that $K = 2$ and we expect both clusters to be of equal size and with equal covariances. If a given data point \mathbf{x}_i is equidistant from the two cluster means (or rather, our current estimates of these), then clearly $\mathbb{E}[z_{i1} | \mathbf{X}, \theta^t] = \mathbb{E}[z_{i2} | \mathbf{X}, \theta^t] = 1/2$.
- ☞ In the general case these expectations are apportioned proportionally to the probability mass assigned to the point by each Gaussian:

$$\mathbb{E}[z_{ij} | \mathbf{X}, \theta^t] = \frac{\tau_j^t f(\mathbf{x}_i | \boldsymbol{\mu}_j^t, \boldsymbol{\Sigma}_j^t)}{\sum_{k=1}^K \tau_k^t f(\mathbf{x}_i | \boldsymbol{\mu}_k^t, \boldsymbol{\Sigma}_k^t)}$$

where $f(\mathbf{x} | \boldsymbol{\mu}, \boldsymbol{\Sigma})$ stands for the multivariate Gaussian density function.

★ EM applied to Gaussian mixtures V

For the Maximisation step we optimise the parameters in the Q-function.

- 👉 Notice there is no interaction between the terms containing τ_j and the terms containing the other parameters, and so the prior distribution τ can be optimised separately:

$$\tau^{t+1} = \operatorname{argmax}_{\tau} \sum_{i=1}^n \sum_{j=1}^K \mathbb{E}[z_{ij} | \mathbf{X}, \theta^t] \ln \tau_j \quad \text{under the constraint} \quad \sum_{j=1}^K \tau_j = 1$$

- 👉 The optimal value can be shown to be achieved by

$$\tau_j^{t+1} = \frac{1}{n} \sum_{i=1}^n \mathbb{E}[z_{ij} | \mathbf{X}, \theta^t]$$

★ EM applied to Gaussian mixtures VI

The means and covariance matrices can be optimised for each cluster separately:

$$\begin{aligned}\boldsymbol{\mu}_j^{t+1}, \boldsymbol{\Sigma}_j^{t+1} &= \operatorname{argmax}_{\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j} \sum_{i=1}^n \mathbb{E} \left[z_{ij} \mid \mathbf{X}, \theta^t \right] \left(-\frac{1}{2} \ln |\boldsymbol{\Sigma}_j| - \frac{1}{2} (\mathbf{x}_i - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_j) \right) \\ &= \operatorname{argmin}_{\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j} \sum_{i=1}^n \mathbb{E} \left[z_{ij} \mid \mathbf{X}, \theta^t \right] \left(\frac{1}{2} \ln |\boldsymbol{\Sigma}_j| + \frac{1}{2} (\mathbf{x}_i - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_j) \right)\end{aligned}$$

👉 This describes a generalised version of the problem of finding the point that minimises the sum of squared Euclidean distances to a set of points ([Theorem 8.1](#)). While that problem is solved by the arithmetic mean, here we simply take the *weighted* average over all the points:

$$\boldsymbol{\mu}_j^{t+1} = \frac{\sum_{i=1}^n \mathbb{E} \left[z_{ij} \mid \mathbf{X}, \theta^t \right] \mathbf{x}_i}{\sum_{i=1}^n \mathbb{E} \left[z_{ij} \mid \mathbf{X}, \theta^t \right]}$$

★ EM applied to Gaussian mixtures VII

- ☞ Similarly, the covariance matrix is computed as a weighted average of covariance matrices obtained from each data point, taking into account the newly estimated mean:

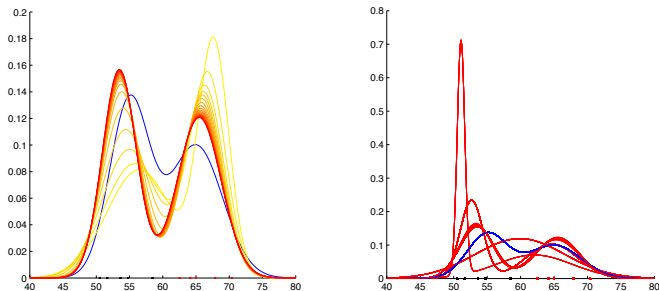
$$\Sigma_j^{t+1} = \frac{\sum_{i=1}^n \mathbb{E}[z_{ij} | \mathbf{X}, \theta^t] (\mathbf{x}_i - \boldsymbol{\mu}_j^{t+1})(\mathbf{x}_i - \boldsymbol{\mu}_j^{t+1})^T}{\sum_{i=1}^n \mathbb{E}[z_{ij} | \mathbf{X}, \theta^t]}$$

- ☞ Notice this allows for clusters of different shapes, unlike the K -means algorithm which assumes that all clusters have the same spherical shape. Consequently, the boundaries between clusters will not be linear, as they are in the clusterings learned by K -means.



Figure 9.8, p.292

EM for univariate Gaussian mixture



(left) The blue line shows the true Gaussian mixture model from which the 10 points on the x -axis were sampled; the colour of the points indicates whether they came from the left or the right Gaussian. The other lines show convergence of Expectation-Maximisation to a stationary configuration from a random initialisation.

(right) This plot shows four stationary configurations for the same set of variables. The EM algorithm was run for 20 iterations; the thickness of one of the lines demonstrates that this configuration takes longer to converge.

What's next?

9

Probabilistic models

- The normal distribution and its geometric interpretations
- Probabilistic models for categorical data
 - Using a naive Bayes model for classification
 - Training a naive Bayes model
- Discriminative learning by optimising conditional likelihood ★
- Probabilistic models with hidden variables
 - Expectation-Maximisation ★
 - Gaussian mixture models ★
- **Compression-based models ★**

★ Compression-based models I

Consider the maximum a posteriori decision rule again:

$$y_{\text{MAP}} = \arg \max_y P(X = x|Y = y)P(Y = y)$$

Taking negative logarithms, we can turn this into an equivalent minimisation:

$$y_{\text{MAP}} = \arg \min_y -\log P(X = x|Y = y) - \log P(Y = y)$$

- 👉 If an event has probability p of happening, the negative logarithm of p quantifies the *information content* of the message that the event has indeed happened.
- 👉 We write $IC(X|Y) = -\log_2 P(X|Y)$ and $IC(Y) = -\log_2 P(Y)$.

★ Compression-based models II

- ☞ The unit of information depends on the base of the logarithm: it is customary to take logarithms to the base 2, in which case information is measured in bits.
- ☞ For example, if you toss a fair coin once and tell me it came up heads, this contains $-\log_2 1/2 = 1$ bit of information; if you roll a fair die once and let me know it came up six, the information content of your message is $-\log_2 1/6 = 2.6$ bits.



Example 9.7, p.293

★ Information-based classification

Y	$P(\text{Viagra} = 1 Y)$	$IC(\text{Viagra} = 1 Y)$	$P(\text{Viagra} = 0 Y)$	$IC(\text{Viagra} = 0 Y)$
spam	0.40	1.32 bits	0.60	0.74 bits
ham	0.12	3.06 bits	0.88	0.18 bits

If Y is uniformly distributed then $IC(Y = \text{spam}) = IC(Y = \text{ham}) = 1$ bit, and

$$\operatorname{argmin}_y (IC(\text{Viagra} = 1|Y = y) + IC(Y = y)) = \text{spam}$$

$$\operatorname{argmin}_y (IC(\text{Viagra} = 0|Y = y) + IC(Y = y)) = \text{ham}$$

If ham is four times as likely as spam then $IC(Y = \text{spam}) = 2.32$ bit and $IC(Y = \text{ham}) = 0.32$ bit, so now

$$\operatorname{argmin}_y (IC(\text{Viagra} = 1|Y = y) + IC(Y = y)) = \text{ham}.$$



★ Minimum description length principle

Let $L(m)$ denote the length in bits of a description of model m , and let $L(D|m)$ denote the length in bits of a description of data D given model m (both measured by some near-optimal code L). According to the minimum description length principle, the preferred model is the one minimising the description length of model and data given model:

$$m_{\text{MDL}} = \underset{m \in M}{\operatorname{argmin}} (L(m) + L(D|m))$$

What encoding to use in order to determine the model complexity $L(m)$ is often not straightforward and to some extent subjective. This is similar to the Bayesian perspective, where we need to define a prior distribution on models. The MDL viewpoint offers a concrete way of defining model priors by means of codes.

What's next?

10 Features

- Kinds of feature
 - Calculations on features
 - Categorical, ordinal and quantitative features
 - Structured features
- Feature transformations
 - Thresholding and discretisation
 - Normalisation and calibration

Features, also called attributes, are defined as mappings $f_i : \mathcal{X} \rightarrow \mathcal{F}_i$ from the instance space \mathcal{X} to the feature domain \mathcal{F}_i .

- ☞ We can distinguish features by their domain: common feature domains include real and integer numbers, but also discrete sets such as colours, the Booleans, and so on.
- ☞ We can also distinguish features by the range of permissible operations. For example, we can calculate a group of people's average age but not their average blood type, so taking the average value is an operation that is permissible on some features but not on others.
- ☞ Although many data sets come with pre-defined features, they can be manipulated in many ways. For example, we can change the domain of a feature by rescaling or discretisation; we can select the best features from a larger set and only work with the selected ones; or we can combine two or more features into a new feature.

What's next?

10 Features

- **Kinds of feature**
 - Calculations on features
 - Categorical, ordinal and quantitative features
 - Structured features
- Feature transformations
 - Thresholding and discretisation
 - Normalisation and calibration

Feature statistics

Three main categories are *statistics of central tendency*, *statistics of dispersion* and *shape statistics*. Each of these can be interpreted either as a theoretical property of an unknown population or a concrete property of a given sample – here we will concentrate on sample statistics.

Starting with statistics of central tendency, the most important ones are

- 👉 the *mean* or average value;
- 👉 the *median*, which is the middle value if we order the instances from lowest to highest feature value; and
- 👉 the *mode*, which is the majority value or values.

Statistics of dispersion

- Two well-known statistics of dispersion are the *variance* or average squared deviation from the (arithmetic) mean, and its square root, the *standard deviation*. Variance is additive, while standard deviation is expressed on the same scale as the feature itself.
- A simpler dispersion statistic is the difference between maximum and minimum value, which is called the *range*. A natural statistic of central tendency to be used with the range is the *midrange point*, which is the mean of the two extreme values.
- Other statistics of dispersion include *percentiles*. The p -th percentile is the value such that p per cent of the instances fall below it. If p is a multiple of 25 the percentiles are also called *quartiles*, and if it is a multiple of 10 the percentiles are also called *deciles*. The general term is *quantiles*. Once we have quantiles we can measure dispersion as the distance between different quantiles. For instance, the *interquartile range* is the difference between the third and first quartile (i.e., the 75th and 25th percentile).



Suppose you are learning a model over an instance space of countries, and one of the features you are considering is the gross domestic product (GDP) per capita. [Figure 10.1](#) shows a so-called *percentile plot* of this feature.

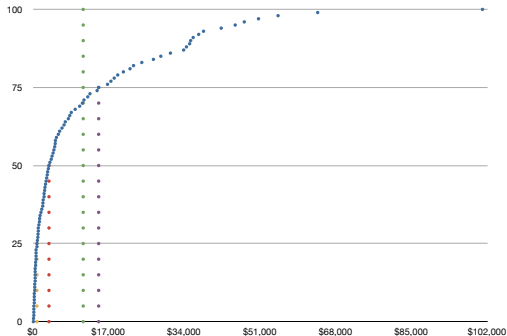
In order to obtain the p -th percentile, you intersect the line $y = p$ with the dotted curve and read off the corresponding percentile on the x -axis. Indicated in the figure are the 25th, 50th and 75th percentile.

Also indicated is the mean (which has to be calculated from the raw data). As you can see, the mean is considerably higher than the median; this is mainly because of a few countries with very high GDP per capita. In other words, the mean is more sensitive to *outliers* than the median, which is why the median is often preferred to the mean for skewed distributions like this one.



Figure 10.1, p.302

Percentile plot



Percentile plot of GDP per capita for 231 countries (data obtained from `wolframalpha.com` by means of the query 'GDP per capita'). The vertical dotted lines indicate, from left to right: the **first quartile** (\$900); the **median** (\$3600); the **mean** (\$11 284); and the **third quartile** (\$14 750). The interquartile range is \$13 850, while the standard deviation is \$16 189.

Cumulative probability distribution

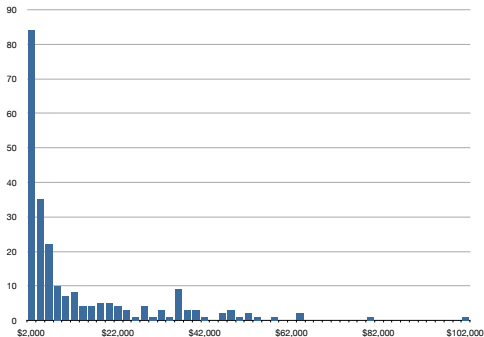
By interpreting the y -axis as probabilities, the plot can be read as a *cumulative probability distribution*: a plot of $P(X \leq x)$ against x for a random variable X . For example, the plot shows that $P(X \leq \mu)$ is approximately 0.70, where $\mu = \$11\,284$ is the mean GDP per capita. In other words, if you choose a random country the probability that its GDP per capita is less than the average is about 0.70.

Since GDP per capita is a real-valued feature, it doesn't necessarily make sense to talk about its mode, since if you measure the feature precisely enough every country will have a different value. We can get around this by means of a *histogram*, which counts the number of feature values in a particular interval or bin.



Figure 10.2, p.303

Histogram



A histogram of the data from [Example 10.1](#). The left-most bin is the mode, with well over a third of the countries having a GDP per capita of not more than \$2000. This demonstrates that the distribution is extremely *right-skewed* (i.e., has a long right tail), resulting in a mean that is considerably higher than the median.

Skewness and kurtosis I

The skew and 'peakedness' of a distribution can be measured by shape statistics such as skewness and kurtosis. The main idea is to calculate the third and fourth *central moment* of the sample.

In general, the k -th central moment of a sample $\{x_i, \dots, x_n\}$ is defined as $m_k = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^k$, where μ is the sample mean.

- 👉 The first central moment is the average deviation from the mean – this is always zero, as the positive and negative deviations cancel each other out.
- 👉 The second central moment is the average squared deviation from the mean, otherwise known as the variance.
- 👉 The third central moment m_3 can again be positive or negative.

Skewness and kurtosis II

Skewness is then defined as m_3/σ^3 , where σ is the sample's standard deviation. A positive value of skewness means that the distribution is right-skewed, which means that the right tail is longer than the left tail. Negative skewness indicates the opposite, left-skewed case.

Kurtosis is defined as m_4/σ^4 . As it can be shown that a normal distribution has kurtosis 3, people often use *excess kurtosis* $m_4/\sigma^4 - 3$ as the statistic of interest. Positive excess kurtosis means that the distribution is more sharply peaked than the normal distribution.

In the GDP per capita example we can calculate skewness as 2.12 and excess kurtosis as 2.53. This confirms that the distribution is heavily right-skewed, and also more sharply peaked than the normal distribution.

Categorical, ordinal and quantitative features

Given these various statistics we can distinguish three main kinds of feature: those with a meaningful numerical scale, those without a scale but with an ordering, and those without either.

- We will call features of the first type *quantitative*; they most often involve a mapping into the reals (another term in common use is ‘continuous’).
- Features with an ordering but without scale are called *ordinal features*. The domain of an ordinal feature is some totally ordered set, such as the set of characters or strings. Another common example are features that express a rank order: first, second, third, and so on. Ordinal features allow the mode and median as central tendency statistics, and quantiles as dispersion statistics.
- Features without ordering or scale are called *categorical features* (or sometimes ‘nominal’ features). They do not allow any statistical summary except the mode. One subspecies of the categorical features is the *Boolean feature*, which maps into the truth values *true* and *false*.



Table 10.1, p.304

Kinds of feature

<i>Kind</i>	<i>Order</i>	<i>Scale</i>	<i>Tendency</i>	<i>Dispersion</i>	<i>Shape</i>
Categorical	×	×	mode	n/a	n/a
Ordinal	✓	×	median	quantiles	n/a
Quantitative	✓	✓	mean	range, interquartile range, variance, standard deviation	skewness, kurtosis

Kinds of feature, their properties and allowable statistics. Each kind inherits the statistics from the kinds above it in the table. For instance, the mode is a statistic of central tendency that can be computed for any kind of feature.

Important point to remember

Tree models ignore the scale of quantitative features, treating them as ordinal.



Suppose an e-mail is represented as a sequence of words. This allows us to define, apart from the usual word frequency features, a host of other features, including:

- ☞ whether the phrase ‘machine learning’ – or any other set of consecutive words – occurs in the e-mail;
- ☞ whether the e-mail contains at least eight consecutive words in a language other than English;
- ☞ whether the e-mail is palindromic, as in ‘Degas, are we not drawn onward, we freer few, drawn onward to new eras?’

Furthermore, we could go beyond properties of single e-mails and express relations such as whether one e-mail is quoted in another e-mail, or whether two e-mails have one or more passages in common.

What's next?

10 Features

- Kinds of feature
 - Calculations on features
 - Categorical, ordinal and quantitative features
 - Structured features
- Feature transformations
 - Thresholding and discretisation
 - Normalisation and calibration



Table 10.2, p.307

Feature transformations

↓ to, from →	<i>Quantitative</i>	<i>Ordinal</i>	<i>Categorical</i>	<i>Boolean</i>
<i>Quantitative</i>	normalisation	calibration	calibration	calibration
<i>Ordinal</i>	discretisation	ordering	ordering	ordering
<i>Categorical</i>	discretisation	unordering	grouping	
<i>Boolean</i>	thresholding	thresholding	binarisation	

An overview of possible feature transformations. **Normalisation and calibration** adapt the scale of quantitative features, or add a scale to features that don't have one.

Ordering adds or adapts the order of feature values without reference to a scale. The other operations abstract away from unnecessary detail, either in a deductive way (**unordering**, **binarisation**) or by introducing new information (**thresholding**, **discretisation**).



Consider the GDP per capita feature plotted in [Figure 10.1](#) again. Without knowing how this feature is to be used in a model, the most sensible thresholds are the statistics of central tendency such as the mean and the median. This is referred to as *unsupervised thresholding*.

In a *supervised* learning setting we can do more. For example, suppose we want to use the GDP per capita as a feature in a decision tree to predict whether a country is one of the 23 countries that use the Euro as their official currency (or as one of their currencies). Using the feature as a ranker, we can construct a coverage curve ([Figure 10.3 \(left\)](#)). We see that for this feature the mean is not the most obvious threshold, as it splits right in the middle of a run of negatives. A better split is obtained at the start of that run of negatives, or at the end of the following run of positives, indicated by the red points at either end of the mean split. More generally, any point on the convex hull of the coverage curve represents a candidate threshold; which one to choose is informed by whether we put more value on picking out positives or negatives.

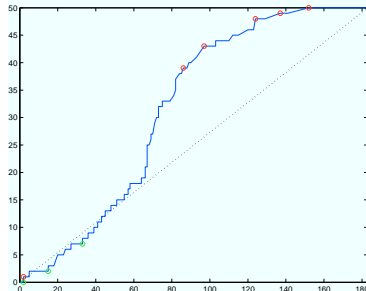
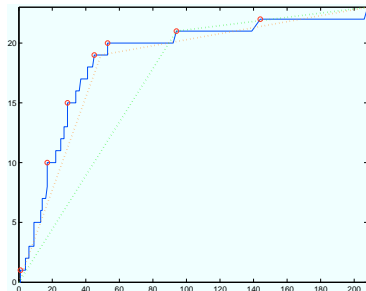


Figure 10.3 (right) shows the same feature with a different target: whether a country is in the Americas. We see that part of the curve is below the ascending diagonal, indicating that, in comparison with the whole data set, the initial segment of the ranking contains a smaller proportion of American countries. This means that potentially useful thresholds can also be found on the *lower convex hull*.



Figure 10.3, p.308

Thresholding



(left) Coverage curve obtained by ranking countries on decreasing GDP per capita, using 23 Euro countries as the positive class. The **orange split** sets the threshold equal to the mean and the **green split** sets the threshold equal to the median. The **red points** are on the convex hull of the coverage curve and indicate potentially optimal splits when the class label is taken into account. **(right)** Coverage curve of the same feature, using 50 countries in the Americas as the positive class.



Example 10.6, p.310

Discretisation by recursive partitioning I

Consider the following feature values, which are ordered on increasing value for convenience.

<i>Instance</i>	<i>Value</i>	<i>Class</i>
e_1	-5.0	⊖
e_2	-3.1	⊕
e_3	-2.7	⊖
e_4	0.0	⊖
e_5	7.0	⊖
e_6	7.1	⊕
e_7	8.5	⊕
e_8	9.0	⊖
e_9	9.0	⊕
e_{10}	13.7	⊖
e_{11}	15.1	⊖
e_{12}	20.1	⊖



Example 10.6, p.310

Discretisation by recursive partitioning II

This feature gives rise to the following ranking: $e_1 \oplus e_2 \oplus e_3 \oplus e_4 \oplus e_5 \oplus e_6 \oplus e_7 \oplus [e_8 \oplus e_9] \oplus e_{10} \oplus e_{11} \oplus e_{12}$, where the square brackets indicate a tie between instances e_8 and e_9 .

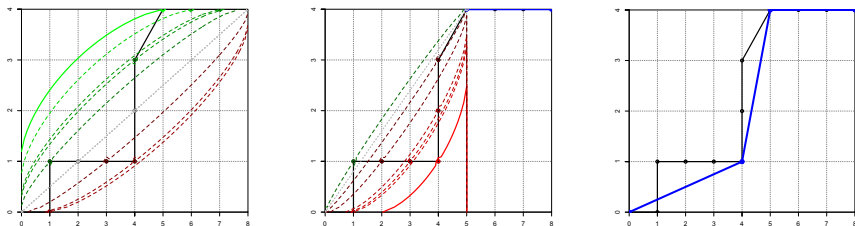
☞ Tracing information gain isometrics through each possible split, we see that the best split is $e_1 \oplus e_2 \oplus e_3 \oplus e_4 \oplus e_5 \oplus e_6 \oplus e_7 \oplus [e_8 \oplus e_9] | e_{10} \oplus e_{11} \oplus e_{12}$.

☞ Repeating the process once more gives the discretisation $e_1 \oplus e_2 \oplus e_3 \oplus e_4 \oplus e_5 \oplus [e_6 \oplus e_7] \oplus [e_8 \oplus e_9] | e_{10} \oplus e_{11} \oplus e_{12}$.



Figure 10.4, p.311

Recursive partitioning



(left) A coverage curve visualising the ranking of four positive and eight negative examples by a feature to be discretised. The curved lines are information gain isometrics through possible split points; the solid isometric indicates the best split $[4+, 5-][0+, 3-]$ according to information gain. **(middle)** Recursive partitioning proceeds to split the segment $[4+, 5-]$ into $[1+, 4-][3+, 1-]$. **(right)** If we stop here, the blue curve visualises the discretised (but still ordinal) feature.



Algorithm $\text{RecPart}(S, f, Q)$ – supervised discretisation by means of recursive partitioning.

Input : set of labelled instances S ranked on feature values $f(x)$; scoring function Q .

Output : sequence of thresholds t_1, \dots, t_{k-1} .

```
1 if stopping criterion applies then return  $\emptyset$ ;  
2 ;  
3 Split  $S$  into  $S_l$  and  $S_r$  using threshold  $t$  that optimises  $Q$  ;  
4  $T_l = \text{RecPart}(S_l, f, Q)$ ;  
5  $T_r = \text{RecPart}(S_r, f, Q)$ ;  
6 return  $T_l \cup \{t\} \cup T_r$ ;
```



Algorithm $\text{AggloMerge}(S, f, Q)$ – supervised discretisation by means of agglomerative merging.

Input : set of labelled instances S ranked on feature values $f(x)$; scoring function Q .

Output : sequence of thresholds.

```

1 initialise bins to data points with the same scores;
2 merge consecutive pure bins ;                               // optional optimisation
3 repeat
4   | evaluate  $Q$  on consecutive bin pairs;
5   | merge the pairs with best  $Q$  (unless they invoke the stopping criterion);
6 until no further merges are possible;
7 return thresholds between bins;

```



Example 10.7, p.312

Agglomerative merging using χ^2 I

Algorithm 10.2 initialises the bins to $\ominus|\oplus|\ominus\ominus\ominus|\oplus\oplus|[\ominus\oplus]|\ominus\ominus\ominus$. We illustrate the calculation of the χ^2 statistic for the last two bins. We construct the following contingency table:

	<i>Left bin</i>	<i>Right bin</i>	
\oplus	1	0	1
\ominus	1	3	4
	2	3	5

At the basis of the χ^2 statistic lies a comparison of these observed frequencies with expected frequencies obtained from the row and column marginals.



Example 10.7, p.312

Agglomerative merging using χ^2 II

- For example, the marginals say that the top row contains 20% of the total mass and the left column 40%; so if rows and columns were statistically independent we would expect 8% of the mass – or 0.4 of the five instances – in the top-left cell.
- Following a clockwise direction, the expected frequencies for the other cells are 0.6, 2.4 and 1.6. If the observed frequencies are close to the expected ones, this suggests that these two bins are candidates for merging since the split appears to have no bearing on the class distribution.
- The χ^2 statistic sums the squared differences between the observed and expected frequencies, each term normalised by the expected frequency:

$$\chi^2 = \frac{(1 - 0.4)^2}{0.4} + \frac{(0 - 0.6)^2}{0.6} + \frac{(3 - 2.4)^2}{2.4} + \frac{(1 - 1.6)^2}{1.6} = 1.88$$




Example 10.7, p.312

Agglomerative merging using χ^2 III

- Going left-to-right through the other pairs of consecutive bins, the χ^2 values are 2, 4, 5 and 1.33. This tells us that the fourth and fifth bin are first to be merged, leading to $\ominus|\oplus|\ominus\ominus\ominus|\oplus\oplus[\ominus\oplus]|\ominus\ominus\ominus$.
- We then recompute the χ^2 values (in fact, only those involving the newly merged bin need to be re-computed), yielding 2, 4, 3.94 and 3.94. We now merge the first two bins, giving the partition $\ominus\oplus|\ominus\ominus\ominus|\oplus\oplus[\ominus\oplus]|\ominus\ominus\ominus$.
- This changes the first χ^2 value to 1.88, so we again merge the first two bins, arriving at $\ominus\oplus\ominus\ominus\ominus|\oplus\oplus[\ominus\oplus]|\ominus\ominus\ominus$ (the same three bins as in [Example 10.6](#)).

Normalisation and calibration I

Feature normalisation is often required to neutralise the effect of different quantitative features being measured on different scales. If the features are approximately normally distributed, we can convert them into  *z-scores* (Background 9.1) by centring on the mean and dividing by the standard deviation. If we don't want to assume normality we can centre on the median and divide by the interquartile range.

Sometimes feature normalisation is understood in the stricter sense of expressing the feature on a $[0, 1]$ scale. If we know the feature's highest and lowest values h and l , then we can simply apply the linear scaling $f \mapsto (f - l)/(h - l)$. We sometimes have to guess the value of h or l , and truncate any value outside $[l, h]$. For example, if the feature measures age in years, we may take $l = 0$ and $h = 100$, and truncate any $f > h$ to 1.

Normalisation and calibration II

Feature calibration is understood as a supervised feature transformation adding a meaningful scale carrying class information to arbitrary features. This has a number of important advantages. For instance, it allows models that require scale, such as linear classifiers, to handle categorical and ordinal features. It also allows the learning algorithm to choose whether to treat a feature as categorical, ordinal or quantitative. We will assume a binary classification context, and so a natural choice for the calibrated feature's scale is the posterior probability of the positive class, conditioned on the feature.

The problem of feature calibration can thus be stated as follows: given a feature $F: \mathcal{X} \rightarrow \mathcal{F}$, construct a calibrated feature $F^c: \mathcal{X} \rightarrow [0, 1]$ such that $F^c(x)$ estimates the probability $F^c(x) = P(\oplus | v)$, where $v = F(x)$ is the value of the original feature for x .



Calibration of categorical features

Suppose we want to predict whether or not someone has diabetes from categorical features including whether the person is obese or not, whether he or she smokes, and so on. We collect some statistics which tell us that 1 in every 18 obese persons has diabetes while among non-obese people this is 1 in 55 (data obtained from `wolframalpha.com` with the query 'diabetes').

If $F(x) = 1$ for person x who is obese and $F(y) = 0$ for person y who isn't, then the calibrated feature values are $F^c(x) = 1/18 = 0.055$ and $F^c(y) = 1/55 = 0.018$.



★ Logistic calibration of two features I

Logistic feature calibration is illustrated in [Figure 10.5](#). I generated two sets of 50 points by sampling bivariate Gaussians with identity covariance matrix, centred at (2,2) and (4,4). I then constructed the basic linear classifier as well as two parallel decision boundaries through the class means. Tracing these three lines in calibrated space will help us understand feature calibration.

In the middle figure we see the transformed data in log-odds space, which is clearly a linear rescaling of the axes. The basic linear classifier is now the line $F_1^d(x) + F_2^d(x) = 0$ through the origin. In other words, for this simple classifier feature calibration has removed the need for further training: instead of fitting a decision boundary to the data, we have fitted the data to a fixed decision boundary!



★ Logistic calibration of two features II

On the right we see the transformed data in probability space, which clearly has a non-linear relationship with the other two feature spaces. The basic linear classifier is still linear in this space, but actually this is no longer true for more than two features. To see this, note that $F_1^c(x) + F_2^c(x) = 1$ can be rewritten as

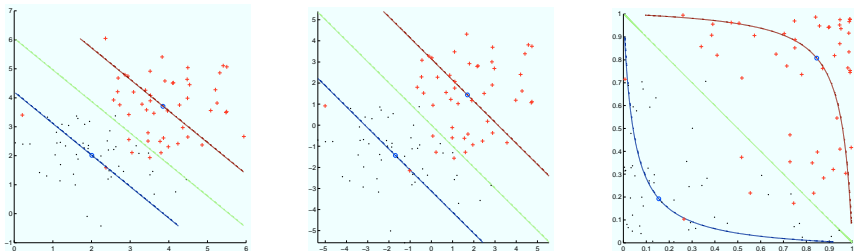
$$\frac{\exp(F_1^d(x))}{1 + \exp(F_1^d(x))} + \frac{\exp(F_2^d(x))}{1 + \exp(F_2^d(x))} = 1$$

which can be simplified to $\exp(F_1^d(x)) \exp(F_2^d(x)) = 1$ and hence to $F_1^d(x) + F_2^d(x) = 0$. However, if we add a third feature not all cross-terms cancel and we obtain a non-linear boundary .



Figure 10.5, p.317

★ Logistic calibration of two features



(left) Two-class Gaussian data. The middle line is the decision boundary learned by the basic linear classifier; the other two are parallel lines through the class means. **(middle)** Logistic calibration to log-odds space is a linear transformation; assuming unit standard deviations, the basic linear classifier is now the fixed line $F_1^d(x) + F_2^d(x) = 0$. **(right)** Logistic calibration to probability space is a non-linear transformation that pushes data away from the decision boundary.

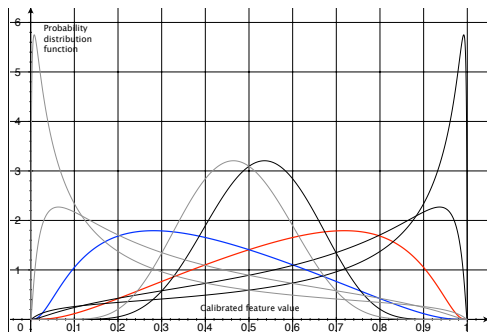
★ Important point to remember

Fitting data to a fixed linear decision boundary in log-odds space by means of feature calibration can be understood as training a naive Bayes model.



Figure 10.6, p.319

★ Calibrated feature densities



Per-class distributions of a logistically calibrated feature for different values of d' , the distance between the uncalibrated class means in proportion to the feature's standard deviation. The red and blue curves depict the distributions for the positive and negative class for a feature whose means are one standard deviation apart ($d' = 1$). The other curves are for $d' \in \{0.5, 1.4, 1.8\}$.



Example 10.10, p.319

Isotonic feature calibration

The following table gives sample values of a weight feature in relation to a diabetes classification problem. [Figure 10.7](#) shows the ROC curve and convex hull of the feature and the calibration map obtained by isotonic calibration.

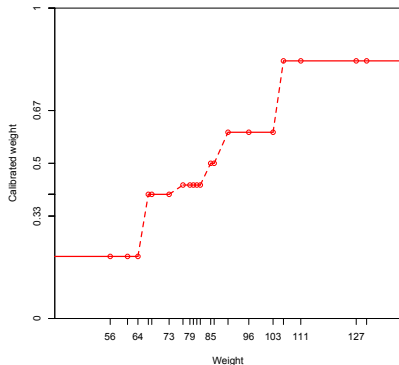
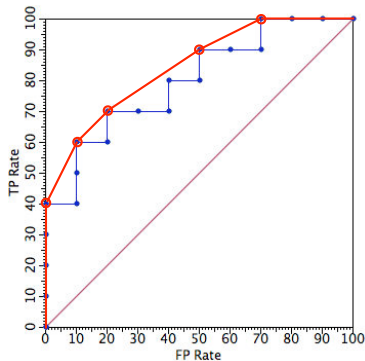
<i>Weight</i>	<i>Diabetes?</i>	<i>Calibrated weight</i>	<i>Weight</i>	<i>Diabetes?</i>	<i>Calibrated weight</i>
130	⊕	0.83	81	⊖	0.43
127	⊕	0.83	80	⊕	0.43
111	⊕	0.83	79	⊖	0.43
106	⊕	0.83	77	⊕	0.43
103	⊖	0.60	73	⊖	0.40
96	⊕	0.60	68	⊖	0.40
90	⊕	0.60	67	⊕	0.40
86	⊖	0.50	64	⊖	0.20
85	⊕	0.50	61	⊖	0.20
82	⊖	0.43	56	⊖	0.20

For example, a weight of 80 kilograms is calibrated to 0.43, meaning that three out of seven people in that weight interval have diabetes (after Laplace correction).



Figure 10.7, p.320

Isotonic feature calibration



(left) ROC curve and convex hull of an uncalibrated feature. Calibrated feature values are obtained from the proportion of positives in each segment of the ROC convex hull.

(right) The corresponding pieewise-constant calibration map, which maps the uncalibrated feature values on the x -axis to the calibrated feature values on the y -axis.



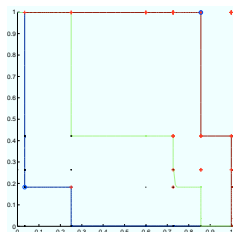
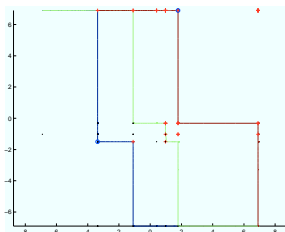
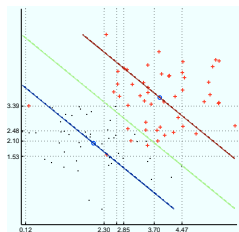
★ Isotonic calibration of two features

Figure 10.8 shows the result of isotonic calibration on the same data as in Example 10.9, both in log-odds space and in probability space. Because of the discrete nature of isotonic calibration, even the transformation to log-odds space is no longer linear: the basic linear classifier becomes a series of axis-parallel line segments. This is also true in the opposite direction: if we imagine a linear decision boundary in log-odds space or in probability space, this maps to a decision boundary following the dotted lines in the original feature space. Effectively, isotonic feature calibration has changed the linear grading model into a grouping model.



Figure 10.8, p.321

★ Isotonic calibration of two features



(left) The data from Figure 10.5, with grid lines indicating the discretisation obtained by isotonic feature calibration. **(middle)** Isotonically calibrated data in log-odds space. **(right)** Isotonically calibrated data in probability space.

What's next?

- 11 Model ensembles
 - Bagging and random forests
 - Boosting
 - Bias, variance and margins

Ensemble methods

In essence, ensemble methods in machine learning have the following two things in common:

- 👉 they construct multiple, diverse predictive models from adapted versions of the training data (most often reweighted or resampled);
- 👉 they combine the predictions of these models in some way, often by simple averaging or voting (possibly weighted).

What's next?

11 Model ensembles

- Bagging and random forests
- Boosting
 - Bias, variance and margins



Algorithm Bagging(D, T, \mathcal{A}) – train an ensemble of models from bootstrap samples.

Input : data set D ; ensemble size T ; learning algorithm \mathcal{A} .

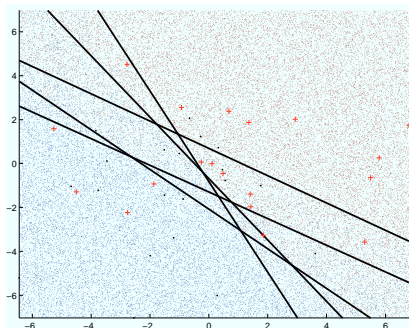
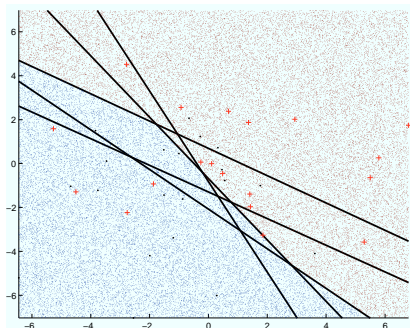
Output : ensemble of models whose predictions are to be combined by voting or averaging.

```
1 for  $t = 1$  to  $T$  do
2   | build a bootstrap sample  $D_t$  from  $D$  by sampling  $|D|$  data points with
   | replacement;
3   | run  $\mathcal{A}$  on  $D_t$  to produce a model  $M_t$ ;
4 end
5 return  $\{M_t | 1 \leq t \leq T\}$ 
```



Figure 11.1, p.332

Bagging



(left) An ensemble of five basic linear classifiers built from bootstrap samples with bagging. The decision rule is majority vote, leading to a piecewise linear decision boundary. **(right)** If we turn the votes into probabilities, we see the ensemble is effectively a grouping model: each instance space segment obtains a slightly different probability.



Algorithm $\text{RandomForest}(D, T, d)$ – train an ensemble of tree models from bootstrap samples and random subspaces.

Input : data set D ; ensemble size T ; subspace dimension d .

Output : ensemble of tree models whose predictions are to be combined by voting or averaging.

```
1 for  $t = 1$  to  $T$  do
2   | build a bootstrap sample  $D_t$  from  $D$  by sampling  $|D|$  data points with
   | replacement;
3   | select  $d$  features at random and reduce dimensionality of  $D_t$  accordingly;
4   | train a tree model  $M_t$  on  $D_t$  without pruning;
5 end
6 return  $\{M_t | 1 \leq t \leq T\}$ 
```

What's next?

11 Model ensembles

- Bagging and random forests
- **Boosting**
 - Bias, variance and margins



Example 11.1, p.334

Weight updates in boosting

- Suppose a linear classifier achieves performance as in the contingency table on the left. The error rate is $\epsilon = (9 + 16)/100 = 0.25$.
- We want to give half the weight to the misclassified examples. The following weight updates achieve this: a factor $1/2\epsilon = 2$ for the misclassified examples and $1/2(1 - \epsilon) = 2/3$ for the correctly classified examples.

	<i>Predicted</i> \oplus		<i>Predicted</i> \ominus		
<i>Actual</i> \oplus	24	16	40		
<i>Actual</i> \ominus	9	51	60		
	33	67	100		

	\oplus	\ominus	
\oplus	16	32	48
\ominus	18	34	52
	34	66	100

- Taking these updated weights into account leads to the contingency table on the right, which has a (weighted) error rate of 0.5.



Algorithm 11.3, p.335

Boosting

Algorithm Boosting(D, T, \mathcal{A}) – train an ensemble of binary classifiers from reweighted training sets.

Input : data set D ; ensemble size T ; learning algorithm \mathcal{A} .

Output : weighted ensemble of models.

```

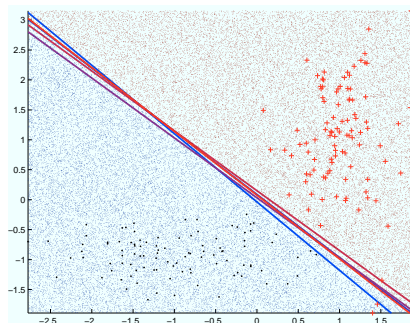
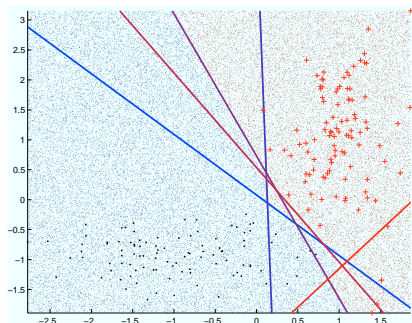
1  $w_{1i} \leftarrow 1/|D|$  for all  $x_i \in D$ ; // start with uniform weights
2 for  $t = 1$  to  $T$  do
3   run  $\mathcal{A}$  on  $D$  with weights  $w_{ti}$  to produce a model  $M_t$ ;
4   calculate weighted error  $\epsilon_t$ ;
5   if  $\epsilon_t \geq 1/2$  then
6     | set  $T \leftarrow t - 1$  and break
7   end
8    $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ ; // confidence for this model
9    $w_{(t+1)i} \leftarrow \frac{w_{ti}}{2\epsilon_t}$  for misclassified instances  $x_i \in D$ ; // increase weight
10   $w_{(t+1)j} \leftarrow \frac{w_{tj}}{2(1-\epsilon_t)}$  for correctly classified instances  $x_j \in D$ ; // decrease
11 end
12 return  $M(x) = \sum_{t=1}^T \alpha_t M_t(x)$ 

```



Figure 11.2, p.336

Boosting



(left) An ensemble of five boosted basic linear classifiers with majority vote. The linear classifiers were learned from blue to red; none of them achieves zero training error, but the ensemble does. **(right)** Applying bagging results in a much more homogeneous ensemble, indicating that there is little diversity in the bootstrap samples.

★ Why those α_t ?

The two weight updates for the misclassified instances and the correctly classified instances can be written as reciprocal terms δ_t and $1/\delta_t$ normalised by some term Z_t :

$$\frac{1}{2\epsilon_t} = \frac{\delta_t}{Z_t} \qquad \frac{1}{2(1-\epsilon_t)} = \frac{1/\delta_t}{Z_t}$$

From this we can derive

$$Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)} \qquad \delta_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = \exp(\alpha_t)$$

So the weight update for misclassified instances is $\exp(\alpha_t)/Z_t$ and for correctly classified instances $\exp(-\alpha_t)/Z_t$. Using the fact that $y_i M_t(x_i) = +1$ for instances correctly classified by model M_t and -1 otherwise, we can write the weight update as

$$w_{(t+1)i} = w_{ti} \frac{\exp(-\alpha_t y_i M_t(x_i))}{Z_t}$$

which is the expression commonly found in the literature.

Important points to remember

Low-bias models tend to have high variance, and vice versa.

Bagging is predominantly a variance-reduction technique, while boosting is primarily a bias-reduction technique.

This explains why bagging is often used in combination with high-variance models such as tree models (👉 *random forests* in [Algorithm 11.2](#)), whereas boosting is typically used with high-bias models such as linear classifiers or univariate decision trees (also called *decision stumps*).

What's next?

- 12 Machine learning experiments
 - What to measure
 - How to measure it
 - How to interpret it
 - Interpretation of results over multiple data sets

Machine learning experiments

Machine learning experiments pose questions about models that we try to answer by means of measurements on data.

The following are common examples of the types of question we are interested in:

- 👉 How does model m perform on data from domain \mathcal{D} ?
- 👉 Which of these models has the best performance on data from domain \mathcal{D} ?
- 👉 How do models produced by learning algorithm \mathcal{A} perform on data from domain \mathcal{D} ?
- 👉 Which of these learning algorithms gives the best model on data from domain \mathcal{D} ?

Assuming we have access to data from domain \mathcal{D} , we perform measurements on our models using this data in order to answer these questions.

What's next?

- 12 Machine learning experiments
 - What to measure
 - How to measure it
 - How to interpret it
 - Interpretation of results over multiple data sets



Example 12.1, p.345

Expected accuracy I

Imagine your classifier achieves the following result on a test data set:

	Predicted \oplus	Predicted \ominus	
Actual \oplus	60	20	80
Actual \ominus	0	20	20
	60	40	100

This gives $tpr = 0.75$, $tnr = 1.00$ and $acc = 0.80$.

However, this is conditioned on having four times as many positives as negatives. If we take the expectation over pos uniformly sampled from the unit interval, expected accuracy increases to $(tpr + tnr)/2 = 0.88 = avg-rec$. This is higher because the test data under-emphasises the classifier's good performance on the negatives.



Example 12.1, p.345

Expected accuracy II

Suppose you have a second classifier achieving the following result on the test set:

	Predicted \oplus	Predicted \ominus	
Actual \oplus	75	5	80
Actual \ominus	10	10	20
	85	15	100

This gives $tpr = 0.94$, $tnr = 0.50$, $acc = 0.85$ and $avg-rec = 0.72$. These experimental results tell you that the second model is better if the class distribution in the test set is representative, but the first model should be chosen if we have no prior information about the class distribution in the operating context.



Example 12.2, p.346

Precision and recall

The F-measure is the harmonic mean of precision and recall, which is 0.91. Now consider the following contingency table:

	<i>Predicted</i> \oplus	<i>Predicted</i> \ominus	
<i>Actual</i> \oplus	75	5	80
<i>Actual</i> \ominus	10	910	920
	85	915	1000

We have a much higher number of true negatives and therefore a much higher true negative rate and accuracy (both rounded to 0.99). On the other hand, recall, precision and F-measure stay exactly the same.

This example demonstrates that the combination of precision and recall, and therefore the F-measure, is insensitive to the number of true negatives and hence implicitly asserts that true negatives are not relevant for your operating context.



Example 12.3, p.347

Expected accuracy and AUC

Suppose a ranker obtains the following ranking on a small test set: $\oplus\oplus\ominus\ominus\oplus\ominus$. This corresponds to two ranking errors out of a possible nine, so has **AUC** = 7/9. There are seven potential split points, corresponding to predicted positive rates of (from left to right) 0, 1/6, ..., 5/6, 1 and corresponding accuracies 3/6, 4/6, 5/6, 4/6, 3/6, 4/6, 3/6. The expected accuracy over all possible split points is $(3 + 4 + 5 + 4 + 3 + 4 + 3)/(6 \cdot 7) = 26/42$. On the other hand, $(2\mathbf{AUC} - 1)/4 = 5/36$ and so $\frac{n}{n+1}(2\mathbf{AUC} - 1)/4 + 1/2 = 5/42 + 1/2 = 26/42$.

What to measure

Your choice of evaluation measures should reflect the assumptions you are making about your experimental objective as well as possible contexts in which your models operate. We have looked at the following cases:

- 👉 Accuracy is a good evaluation measure if the class distribution in your test set is representative for the operating context.
- 👉 Average recall is the evaluation measure of choice if all class distributions are equally likely.
- 👉 Precision and recall shift the focus from classification accuracy to a performance analysis ignoring the true negatives.
- 👉 Predicted positive rate and **AUC** are relevant measures in a ranking context.

What's next?

- 12 Machine learning experiments
 - What to measure
 - How to measure it
 - How to interpret it
 - Interpretation of results over multiple data sets

Measurements as random variables

When we measure something – say, a person's height – several times, we expect some variation to occur from one measurement to the next.

This variation can be modelled by treating our measurement as a random variable characterised by its mean – the value we are trying to measure – and variance σ^2 , both of which are unknown but can be estimated.

A standard trick is to average k measurements, as this gets the variance in your estimate down to σ^2/k . Crucially, this assumes that your measurements are independent: if you are introducing a systematic error by using a faulty tape measure, averaging won't help!

Accuracy etc. as a random variable

Now suppose you are measuring a classifier's accuracy from test data.

The natural model here is that each test instance represents a Bernoulli trial with success probability a , the true but unknown accuracy of the classifier. We estimate a by counting the number of correctly classified test instances A and setting $\hat{a} = A/n$; notice that A has a binomial distribution.

The variance of a single Bernoulli trial is $a(1 - a)$; averaged over n test instances it is $a(1 - a)/n$, assuming the test instances are chosen independently. We can estimate the variance by plugging in our estimate for a , leading to an estimated variance of $\hat{a}(1 - \hat{a})/n$.

Cross-validation I

We can improve our estimate by averaging k independent estimates \hat{a}_i and take their sample variance $\frac{1}{k-1} \sum_{i=1}^k (\hat{a}_i - \bar{a})^2$ instead, with $\bar{a} = \frac{1}{k} \sum_{i=1}^k \hat{a}_i$ the sample mean. (We divide by $k-1$ rather than k in the expression for the sample variance, to account for the uncertainty in our estimate of the sample mean.)

How do we obtain k independent estimates of a ? If we have plenty of data, we can sample k independent test sets of size n and estimate a on each of them. This test data needs to be separate from any training data used to build the model or tune its parameters.

If we don't have a lot of data, the following *cross-validation* procedure is often applied: randomly partition the data in k parts or 'folds', set one fold aside for testing, train a model on the remaining $k-1$ folds and evaluate it on the test fold. This process is repeated k times until each fold has been used for testing once.

Cross-validation II

This may seem curious at first since we are evaluating k models rather than a single one, but this makes sense if we are evaluating a learning algorithm (whose output is a model, so we want to average over models) rather than a single model (whose outputs are instance labels, so we want to average over those).

By averaging over training sets we get a sense of the variance of the learning algorithm (i.e., its dependence on variations in the training data), although it should be noted that the training sets in cross-validation have considerable overlap and are clearly not independent.

Once we are satisfied with the performance of our learning algorithm, we can run it over the entire data set to obtain a single model.

Cross-validation III

Cross-validation is conventionally applied with $k = 10$, although this is somewhat arbitrary. A rule of thumb is that individual folds should contain at least 30 instances, as this allows us to approximate the binomial distribution of the number of correctly classified instances in a fold by a normal distribution. So if we have fewer than 300 instances we need to adjust k accordingly.

Alternatively, we can set $k = n$ and train on all but one test instance, repeated n times: this is known as *leave-one-out* cross-validation (or the jackknife in statistics). This means that in each single-instance ‘fold’ our accuracy estimate is 0 or 1, but by averaging n of those we get an approximately normal distribution by the central limit theorem.

If we expect the learning algorithm to be sensitive to the class distribution we should apply *stratified cross-validation*: this aims at achieving roughly the same class distribution in each fold.



Example 12.4, p.350

Cross-validation

The following table gives a possible result of evaluating three learning algorithms on a data set with 10-fold cross-validation:

<i>Fold</i>	<i>Naive Bayes</i>	<i>Decision tree</i>	<i>Nearest neighbour</i>
1	0.6809	0.7524	0.7164
2	0.7017	0.8964	0.8883
3	0.7012	0.6803	0.8410
4	0.6913	0.9102	0.6825
5	0.6333	0.7758	0.7599
6	0.6415	0.8154	0.8479
7	0.7216	0.6224	0.7012
8	0.7214	0.7585	0.4959
9	0.6578	0.9380	0.9279
10	0.7865	0.7524	0.7455
avg	0.6937	0.7902	0.7606
stdev	0.0448	0.1014	0.1248

The last two lines give the average and standard deviation over all ten folds. Clearly the decision tree achieves the best result, but should we completely discard nearest neighbour?

What's next?

- 12 Machine learning experiments
 - What to measure
 - How to measure it
 - How to interpret it
 - Interpretation of results over multiple data sets

Understanding the uncertainty in your measurements I

Suppose our estimate \hat{a} follows a normal distribution around the true mean a with standard deviation σ . Assuming for the moment that we know these parameters, we can calculate for any interval the likelihood of the estimate falling in the interval, by calculating the area under the normal density function in that interval.

- 👉 For example, the likelihood of obtaining an estimate within ± 1 standard deviation around the mean is 68%. Thus, if we take 100 estimates from independent test sets, we expect 68 of them to be within one standard deviation on either side of the mean – or equivalently, we expect the true mean to fall within one standard deviation on either side of the estimate in 68 cases. This is called the 68% *confidence interval* of the estimate.
- 👉 For two standard deviations the confidence level is 95% – these values can be looked up in probability tables or calculated using statistical packages such as Matlab or R.

Understanding the uncertainty in your measurements II

Confidence intervals for normally distributed estimates are symmetric because the normal distribution is symmetric, but this is not necessarily the case for other distributions: e.g., the binomial distribution is asymmetric (except for $p = 1/2$).

In case of symmetry, we can easily change the interval into a one-sided interval: for example, we expect the mean to be more than one standard deviation *above* the estimate in 16 cases out of 100, which gives a one-sided 84% confidence interval from minus infinity to the mean plus one standard deviation.

Understanding the uncertainty in your measurements III

More generally, in order to construct confidence intervals we need to know (i) the sampling distribution of the estimates, and (ii) the parameters of that distribution.

- 👉 We saw previously that accuracy estimated from a single test set with n instances follows a scaled binomial distribution with variance $\hat{a}(1 - \hat{a})/n$.
- 👉 This would lead to asymmetric confidence intervals, but the skew in the binomial distribution is only really noticeable if $na(1 - a) < 5$: if that is not the case the normal distribution is a good approximation for the binomial one.
- 👉 So we usually use the binomial expression for the variance and use the normal distribution to construct the confidence intervals.



Suppose 80 out of 100 test instances are correctly classified.

We have $\hat{a} = 0.80$ with an estimated variance of $\hat{a}(1 - \hat{a})/n = 0.0016$ or a standard deviation of $\sqrt{\hat{a}(1 - \hat{a})/n} = 0.04$.

Notice $n\hat{a}(1 - \hat{a}) = 16 \geq 5$ so the 68% confidence interval is estimated as $[0.76, 0.84]$ in accordance with the normal distribution, and the 95% interval is $[0.72, 0.88]$.

If we reduce the size of our test sample to 50 and find that 40 test instances are correctly classified, then the standard deviation increases to 0.06 and the 95% confidence interval widens to $[0.68, 0.92]$. If the test sample drops to less than 30 instances we would need to construct an asymmetric confidence interval using tables for the binomial distribution.

A common misunderstanding about confidence intervals

Notice that confidence intervals are statements about estimates rather than statements about the true value of the evaluation measure.

- ✎ The statement ‘assuming the true accuracy a is 0.80, the probability that a measurement m falls in the interval $[0.72, 0.88]$ is 0.95’ is correct, but we cannot reverse this to say ‘assuming a measurement $m = 0.80$, the probability that the true accuracy falls in the interval $[0.72, 0.88]$ is 0.95’.
- ✎ To infer $P(a \in [0.72, 0.88] | m = 0.80)$ from $P(m \in [0.72, 0.88] | a = 0.80)$ we must somehow invoke Bayes’ rule, but this requires meaningful prior distributions over both true accuracies and measurements, which we don’t generally have.

Null hypothesis and p -value

We can, however, use similar reasoning to test a particular *null hypothesis* we have about a .

- For example, suppose our null hypothesis is that the true accuracy is 0.5 and that the standard deviation derived from the binomial distribution is therefore $\sqrt{0.5(1 - 0.5)/100} = 0.05$.
- Given our estimate of 0.80, we then calculate the *p -value*, which is the probability of obtaining a measurement of 0.80 or higher given the null hypothesis.
- The p -value is then compared with a pre-defined significance level, say $\alpha = 0.05$: this corresponds to a confidence of 95%.
- The null hypothesis is rejected if the p -value is smaller than α ; in our case this applies since $p = 1.9732 \cdot 10^{-9}$.

Significance testing in cross-validation: the paired t -test

- ✎ For a pair of algorithms we calculate the difference in accuracy on each fold; this difference is normally distributed if the two accuracies are. Our null hypothesis is that the true difference is 0, so that any differences in performance are attributed to chance. We calculate a p -value using the normal distribution, and reject the null hypothesis if the p -value is below our significance level α .
- ✎ The one complication is that we don't have access to the true standard deviation in the differences, which therefore needs to be estimated. This introduces additional uncertainty into the process, which means that the sampling distribution is bell-shaped like the normal distribution but slightly more heavy-tailed. This distribution is referred to as the *t -distribution*.
- ✎ The extent to which the t -distribution is more heavy-tailed than the normal distribution is regulated by the number of *degrees of freedom*: in our case this is equal to 1 less than the number of folds (since the final fold is completely determined by the other ones).



Example 12.6, p.353

Paired t -test

The numbers show pairwise differences in each fold. The null hypothesis in each case is that the differences come from a normal distribution with mean 0 and unknown standard deviation.

<i>Fold</i>	<i>NB-DT</i>	<i>NB-NN</i>	<i>DT-NN</i>
1	-0.0715	-0.0355	0.0361
2	-0.1947	-0.1866	0.0081
3	0.0209	-0.1398	-0.1607
4	-0.2189	0.0088	0.2277
5	-0.1424	-0.1265	0.0159
6	-0.1739	-0.2065	-0.0325
7	0.0992	0.0204	-0.0788
8	-0.0371	0.2255	0.2626
9	-0.2802	-0.2700	0.0102
10	0.0341	0.0410	0.0069
avg	-0.0965	-0.0669	0.0295
stdev	0.1246	0.1473	0.1278
<i>p</i> -value	0.0369	0.1848	0.4833

The p -value in the last line of the table is calculated by means of the t -distribution with $k - 1 = 9$ degrees of freedom, and only the difference between the naive Bayes and decision tree algorithms is found significant at $\alpha = 0.05$.

Interpretation of results over multiple data sets I

The t -test is not appropriate for multiple data sets because performance measures cannot be compared across data sets (they are not 'commensurate'). In order to compare two learning algorithms over multiple data sets we need to use a test specifically designed for that purpose such as *Wilcoxon's signed-rank test*.

- 👉 The idea is to rank the performance differences in absolute value, from smallest (rank 1) to largest (rank n).
- 👉 We then calculate the sum of ranks for positive and negative differences separately, and take the smaller of these sums as our test statistic.
- 👉 For a large number of data sets (at least 25) this statistic can be converted to one which is approximately normally distributed, but for smaller numbers the *critical value* (the value of the statistic at which the p -value equals α) can be found in a statistical table.

Interpretation of results over multiple data sets II

- ✎ The Wilcoxon signed-rank test assumes that larger performance differences are better than smaller ones, but otherwise makes no assumption about their commensurability – in other words, performance differences are treated as ordinals rather than real-valued.
- ✎ Furthermore, there is no normality assumption regarding the distribution of these differences which means, among other things, that the test is less sensitive to outliers.
- ✎ In statistical terminology the test is ‘non-parametric’ as opposed to a parametric test such as the t -test which assumes a particular distribution. Parametric tests are generally more powerful when that assumed distribution is appropriate but can be misleading when it is not.



Example 12.7, p.354

Wilcoxon's signed-rank test

<i>Data set</i>	<i>NB-DT</i>	<i>Rank</i>
1	-0.0715	4
2	-0.1947	8
3	0.0209	1
4	-0.2189	9
5	-0.1424	6
6	-0.1739	7
7	0.0992	5
8	-0.0371	3
9	-0.2802	10
10	0.0341	2

The sum of ranks for positive differences is $1 + 5 + 2 = 8$ and for negative differences $4 + 8 + 9 + 6 + 7 + 3 + 10 = 47$. The critical value for 10 data sets at the $\alpha = 0.05$ level is 8, which means that if the smallest of the two sums of ranks is less than or equal to 8 the null hypothesis that the ranks are distributed the same for positive and negative differences can be rejected. This applies in this case.

Comparing multiple algorithms over multiple data sets I

If we want to compare k algorithms over n data sets we need to use specialised significance tests to avoid that our confidence level drops with each additional pairwise comparison between algorithms. The *Friedman test* is designed for exactly this situation.

- Like the Wilcoxon signed-rank test it is based on ranked rather than absolute performance, and hence makes no assumption regarding the distribution of the performance measurements. The idea is to rank the performance of all k algorithms per data set, from best performance (rank 1) to worst performance (rank k).
- Let R_{ij} denote the rank of the j -th algorithm on the i -th data set, and let $R_j = (\sum_i R_{ij}) / n$ be the average rank of the j -th algorithm. Under the null hypothesis that all algorithms perform equally these average ranks R_j should be the same.

Comparing multiple algorithms over multiple data sets II

☞ In order to test this we calculate the following quantities:

the average rank $\bar{R} = \frac{1}{nk} \sum_{ij} R_{ij} = \frac{k+1}{2}$;

the sum of squared differences $n \sum_j (R_j - \bar{R})^2$; and

the sum of squared differences $\frac{1}{n(k-1)} \sum_{ij} (R_{ij} - \bar{R})^2$.

☞ There is an analogy with clustering here, in that the second quantity measures the spread between the rank ‘centroids’ – which we want to be large – and the third quantity measures the spread over all ranks. The Friedman statistic is the ratio of the former and latter quantities.



Example 12.8, p.356

Friedman test I

We use the data from [Example 12.4](#), assuming it comes from different data sets rather than cross-validation folds. The following table shows the ranks in brackets:

<i>Data set</i>	<i>Naive Bayes</i>	<i>Decision tree</i>	<i>Nearest neighbour</i>
1	0.6809 (3)	0.7524 (1)	0.7164 (2)
2	0.7017 (3)	0.8964 (1)	0.8883 (2)
3	0.7012 (2)	0.6803 (3)	0.8410 (1)
4	0.6913 (2)	0.9102 (1)	0.6825 (3)
5	0.6333 (3)	0.7758 (1)	0.7599 (2)
6	0.6415 (3)	0.8154 (2)	0.8479 (1)
7	0.7216 (1)	0.6224 (3)	0.7012 (2)
8	0.7214 (2)	0.7585 (1)	0.4959 (3)
9	0.6578 (3)	0.9380 (1)	0.9279 (2)
10	0.7865 (1)	0.7524 (2)	0.7455 (3)
avg rank	2.3	1.6	2.1



Example 12.8, p.356

Friedman test II

We have $\bar{R} = 2$, $n \sum_j (R_j - \bar{R})^2 = 2.6$ and $\frac{1}{n(k-1)} \sum_{ij} (R_{ij} - \bar{R})^2 = 1$, so the Friedman statistic is 2.6.

The critical value for $k = 3$ and $n = 10$ at the $\alpha = 0.05$ level is 7.8, so we cannot reject the null hypothesis that all algorithms perform equally.

In comparison, if the average ranks were 2.7, 1.3 and 2.0, then the null hypothesis would be rejected at that significance level.

Post-hoc tests I

The Friedman test tells us whether the average ranks as a whole display significant differences, but further analysis is needed on a pairwise level. This is achieved by applying a *post-hoc test* once the Friedman test gives significance.

- 👉 The idea is to calculate the *critical difference (CD)* against which the difference in average rank between two algorithms is compared.
- 👉 The *Nemenyi test* calculates the critical difference as follows:

$$CD = q_{\alpha} \sqrt{\frac{k(k+1)}{6n}}$$

where q_{α} depends on the significance level α as well as k : for $\alpha = 0.05$ and $k = 3$ it is 2.343, leading to a critical difference of 1.047 in our example.

- 👉 If the average ranks are 2.7, 1.3 and 2.0, then only the difference between the first and second algorithm exceeds the critical difference.

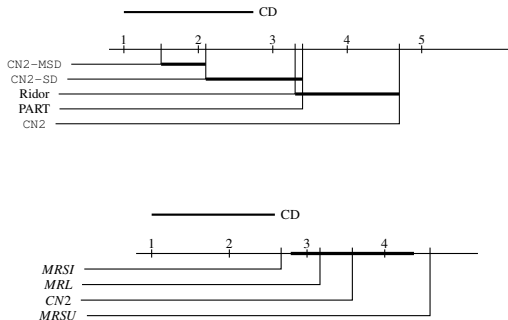
Post-hoc tests II

- A variant of the Nemenyi test called the *Bonferroni–Dunn test* can be applied when we perform pairwise tests only against a control algorithm.
- The calculation of the critical difference is the same, except q_α is adjusted to reflect the fact that we make $k - 1$ pairwise comparisons rather than $k(k - 1)/2$.
- For example, for $\alpha = 0.05$ and $k = 3$ we have $q_\alpha = 2.241$, which is slightly lower than the value used for the Nemenyi test, leading to a tighter critical difference.



Figure 12.1, p.357

Critical difference diagram



(top) Average ranks for each algorithm are plotted on the real axis. The critical difference is shown as a bar above the figure, and any group of consecutively ranked algorithms such that the outermost ones are less than the critical difference apart are connected by a horizontal thick line. **(bottom)** Critical difference diagram for the Bonferroni–Dunn test with **CN2** as control. The critical differences are now drawn symmetrically around the average rank of the control.