

---

## Prologue: A machine learning sampler

---

**Y**OU MAY NOT be aware of it, but chances are that you are already a regular user of machine learning technology. Most current e-mail clients incorporate algorithms to identify and filter out spam e-mail, also known as junk e-mail or unsolicited bulk e-mail. Early spam filters relied on hand-coded pattern matching techniques such as regular expressions, but it soon became apparent that this is hard to maintain and offers insufficient flexibility – after all, one person's spam is another person's ham!<sup>1</sup> Additional adaptivity and flexibility is achieved by employing machine learning techniques.

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. Here is an example report for an e-mail I received:

```
-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
```

---

<sup>1</sup>Spam, a contraction of ‘spiced ham’, is the name of a meat product that achieved notoriety by being ridiculed in a 1970 episode of *Monty Python’s Flying Circus*.

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. As it happens, this particular e-mail was a notification from an intermediate server that another message – which had a whopping score of 14.6 – was rejected as spam. This ‘bounce’ message included the original message and therefore inherited some of its characteristics, such as a low text-to-image ratio, which pushed the score over the threshold of 5.

Here is another example, this time of an important e-mail I had been expecting for some time, only for it to be found languishing in my spam folder:

2.5 URI_NOVOWEL	URI: URI hostname has long non-vowel sequence
3.1 FROM_DOMAIN_NOVOWEL	From: domain has series of non-vowel letters

The e-mail in question concerned a paper that one of the members of my group and I had submitted to the European Conference on Machine Learning (ECML) and the European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), which have been jointly organised since 2001. The 2008 instalment of these conferences used the internet domain [www.ecmlpkdd2008.org](http://www.ecmlpkdd2008.org) – a perfectly respectable one, as machine learning researchers know, but also one with eleven ‘non-vowels’ in succession – enough to raise SpamAssassin’s suspicion! The example demonstrates that the importance of a SpamAssassin test can be different for different users. Machine learning is an excellent way of creating software that adapts to the user.

### §

How does SpamAssassin determine the scores or ‘weights’ for each of the dozens of tests it applies? This is where machine learning comes in. Suppose we have a large ‘training set’ of e-mails which have been hand-labelled spam or ham, and we know the results of all the tests for each of these e-mails. The goal is now to come up with a weight for every test, such that all spam e-mails receive a score above 5, and all ham e-mails get less than 5. As we will discuss later in the book, there are a number of machine learning techniques that solve exactly this problem. For the moment, a simple example will illustrate the main idea.

---

**Example 1 (Linear classification).** Suppose we have only two tests and four training e-mails, one of which is spam (see Table 1). Both tests succeed for the

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

**Table 1.** A small training set for SpamAssassin. 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.

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.

But what does this have to do with learning, I hear you ask? It is just a mathematical problem, after all. That may be true, but it does not appear unreasonable to say that SpamAssassin learns to recognise spam e-mail from examples and counter-examples. Moreover, the more training data is made available, the better SpamAssassin will become at this task. The notion of performance improving with experience is central to most, if not all, forms of machine learning. We will use the following general definition: *Machine learning is the systematic study of algorithms and systems that improve their knowledge or performance with experience*. In the case of SpamAssassin, the ‘experience’ it learns from is some correctly labelled training data, and ‘performance’ refers to its ability to recognise spam e-mail. A schematic view of how machine learning feeds into the spam e-mail classification task is given in [Figure 2](#). In other machine learning problems experience may take a different form, such as corrections of mistakes, rewards when a certain goal is reached, among many others. Also note that, just as is the case with human learning, machine learning is not always directed at improving performance on a certain task, but may more generally result in improved knowledge.

There are a number of useful ways in which we can express the SpamAssassin classifier in mathematical notation. If we denote the result of the  $i$ -th test for a given e-mail as  $x_i$ , where  $x_i = 1$  if the test succeeds and 0 otherwise, and we denote the weight of the  $i$ -th test as  $w_i$ , then the total score of an e-mail can be expressed as  $\sum_{i=1}^n w_i x_i$ , making use of the fact that  $w_i$  contributes to the sum only if  $x_i = 1$ , i.e., if the test succeeds for the e-mail. Using  $t$  for the threshold above which an e-mail is classified as spam (5 in our example), the ‘decision rule’ can be written as  $\sum_{i=1}^n w_i x_i > t$ .

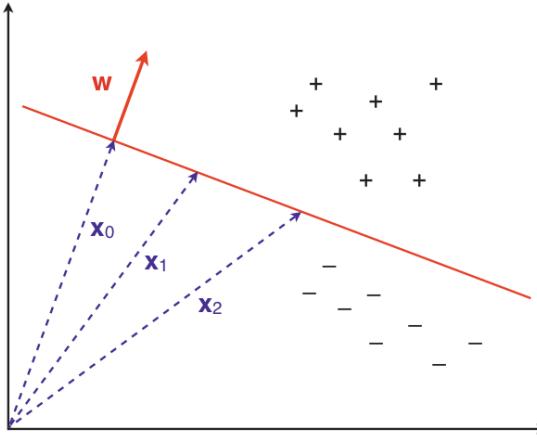
Notice that the left-hand side of this inequality is linear in the  $x_i$  variables, which essentially means that increasing one of the  $x_i$  by a certain amount, say  $\delta$ , will change the sum by an amount ( $w_i \delta$ ) that is independent of the value of  $x_i$ . This wouldn’t be true if  $x_i$  appeared squared in the sum, or with any exponent other than 1.

The notation can be simplified by means of linear algebra, writing  $\mathbf{w}$  for the vector of weights  $(w_1, \dots, w_n)$  and  $\mathbf{x}$  for the vector of test results  $(x_1, \dots, x_n)$ . The above inequality can then be written using a dot product:  $\mathbf{w} \cdot \mathbf{x} > t$ . Changing the inequality to an equality  $\mathbf{w} \cdot \mathbf{x} = t$ , we obtain the ‘decision boundary’, separating spam from ham. The decision boundary is a plane (a ‘straight’ surface) in the space spanned by the  $x_i$  variables because of the linearity of the left-hand side. The vector  $\mathbf{w}$  is perpendicular to this plane and points in the direction of spam. [Figure 1](#) visualises this for two variables.

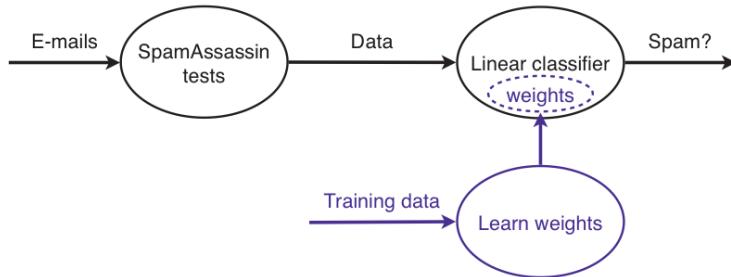
It is sometimes convenient to simplify notation further by introducing an extra constant ‘variable’  $x_0 = 1$ , the weight of which is fixed to  $w_0 = -t$ . The extended data point is then  $\mathbf{x}^\circ = (1, x_1, \dots, x_n)$  and the extended weight vector is  $\mathbf{w}^\circ = (-t, w_1, \dots, w_n)$ , 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 (but note that this doesn’t really affect the data, as all data points and the ‘real’ decision boundary live in the plane  $x_0 = 1$ ).

---

**Background 1.** SpamAssassin in mathematical notation. In boxes such as these, I will briefly remind you of useful concepts and notation. If some of these are unfamiliar, you will need to spend some time reviewing them – using other books or online resources such as [www.wikipedia.org](http://www.wikipedia.org) or [mathworld.wolfram.com](http://mathworld.wolfram.com) – to fully appreciate the rest of the book.



**Figure 1.** An example of 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  $\mathbf{x}$ ). The decision boundary can therefore equivalently be described by  $\mathbf{w} \cdot (\mathbf{x} - \mathbf{x}_0) = 0$ , which is sometimes more convenient. In particular, this notation makes it clear that it is the orientation but not the length of  $\mathbf{w}$  that determines the location of the decision boundary.



**Figure 2.** 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.

We have already seen that a machine learning problem may have several solutions, even a problem as simple as the one from [Example 1](#). This raises the question of how we choose among these solutions. One way to think about this is to realise that we don't really care that much about performance on training data – we already know which of

those e-mails are spam! What we care about is whether *future* e-mails are going to be classified correctly. While this appears to lead into a vicious circle – in order to know whether an e-mail is classified correctly I need to know its true class, but as soon as I know its true class I don't need the classifier anymore – it is important to keep in mind that good performance on training data is only a means to an end, not a goal in itself. In fact, trying too hard to achieve good performance on the training data can easily lead to a fascinating but potentially damaging phenomenon called *overfitting*.

---

**Example 2 (Overfitting).** Imagine you are preparing for your *Machine Learning 101* exam. Helpfully, Professor Flach has made previous exam papers and their worked answers available online. You begin by trying to answer the questions from previous papers and comparing your answers with the model answers provided. Unfortunately, you get carried away and spend all your time on memorising the model answers to all past questions. Now, if the upcoming exam completely consists of past questions, you are certain to do very well. But if the new exam asks different questions about the same material, you would be ill-prepared and get a much lower mark than with a more traditional preparation. In this case, one could say that you were *overfitting* the past exam papers and that the knowledge gained didn't *generalise* to future exam questions.

---

*Generalisation* is probably the most fundamental concept in machine learning. If the knowledge that SpamAssassin has gleaned from its training data carries over – generalises – to your e-mails, you are happy; if not, you start looking for a better spam filter. However, overfitting is not the only possible reason for poor performance on new data. It may just be that the training data used by the SpamAssassin programmers to set its weights is not representative for the kind of e-mails you get. Luckily, this problem does have a solution: use different training data that exhibits the same characteristics, if possible actual spam and ham e-mails that you have personally received. Machine learning is a great technology for adapting the behaviour of software to your own personal circumstances, and many spam e-mail filters allow the use of your own training data.

So, if there are several possible solutions, care must be taken to select one that doesn't overfit the data. We will discuss several ways of doing that in this book. What about the opposite situation, if there isn't a solution that perfectly classifies the training data? For instance, imagine that e-mail 2 in [Example 1](#), the one for which both tests failed, was spam rather than ham – in that case, there isn't a single straight line separating spam from ham (you may want to convince yourself of this by plotting the four

e-mails as points in a grid, with  $x_1$  on one axis and  $x_2$  on the other). There are several possible approaches to this situation. One is to ignore it: that e-mail may be atypical, or it may be mis-labelled (so-called *noise*). Another possibility is to switch to a more expressive type of classifier. For instance, we may introduce a second decision rule for spam: in addition to  $4x_1 + 4x_2 > 5$  we could alternatively have  $4x_1 + 4x_2 < 1$ . Notice that this involves learning a different threshold, and possibly a different weight vector as well. This is only really an option if there is enough training data available to reliably learn those additional parameters.

### §

Linear classification, SpamAssassin-style, may serve as a useful introduction, but this book would have been a lot shorter if that was the only type of machine learning. What about learning not just the weights for the tests, but also the tests themselves? How do we decide if the text-to-image ratio is a good test? Indeed, how do we come up with such a test in the first place? This is an area where machine learning has a lot to offer.

One thing that may have occurred to you is that the SpamAssassin tests considered so far don't appear to take much notice of the *contents* of the e-mail. Surely words and phrases like 'Viagra', 'free iPod' or 'confirm your account details' are good spam indicators, while others – for instance, a particular nickname that only your friends use – point in the direction of ham. For this reason, many spam e-mail filters employ text classification techniques. Broadly speaking, such techniques maintain a vocabulary of words and phrases that are potential spam or ham indicators. For each of those words and phrases, 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 (see [Background 2](#) for some basic notions of probability theory).

The situation is slightly more subtle than you might realise because we have to take into account the prevalence of spam. Suppose, for the sake of argument, that I receive on average one spam e-mail for every six ham e-mails (I wish!). This means that I would estimate the odds of the next e-mail coming in 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 somehow 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 other words, despite the occurrence of the word 'Viagra', the safest bet is still that the e-mail is ham. That doesn't make sense, or does it?

Probabilities involve ‘random variables’ that describe outcomes of ‘events’. These events are often hypothetical and therefore probabilities have to be estimated. For example, consider the statement ‘42% of the UK population approves of the current Prime Minister’. The only way to know this for certain is to ask everyone in the UK, which is of course unfeasible. Instead, a (hopefully representative) sample is queried, and a more correct statement would then be ‘42% of a sample drawn from the UK population approves of the current Prime Minister’, or ‘the proportion of the UK population approving of the current Prime Minister is estimated at 42%’. Notice that these statements are formulated in terms of proportions or ‘relative frequencies’; a corresponding statement expressed in terms of probabilities would be ‘the probability that a person uniformly drawn from the UK population approves of the current Prime Minister is estimated at 0.42’. The event here is ‘this random person approves of the PM’.

The ‘conditional probability’  $P(A|B)$  is the probability of event  $A$  happening given that event  $B$  happened. For instance, the approval rate of the Prime Minister may differ for men and women. Writing  $P(\text{PM})$  for the probability that a random person approves of the Prime Minister and  $P(\text{PM}|\text{woman})$  for the probability that a random woman approves of the Prime Minister, we then have that  $P(\text{PM}|\text{woman}) = P(\text{PM, woman})/P(\text{woman})$ , where  $P(\text{PM, woman})$  is the probability of the ‘joint event’ that a random person both approves of the PM and is a woman, and  $P(\text{woman})$  is the probability that a random person is a woman (i.e., the proportion of women in the UK population).

Other useful equations include  $P(A,B) = P(A|B)P(B) = P(B|A)P(A)$  and  $P(A|B) = P(B|A)P(A)/P(B)$ . The latter is known as ‘Bayes’ rule’ and will play an important role in this book. Notice that many of these equations can be extended to more than two random variables, e.g. the ‘chain rule of probability’:  $P(A,B,C,D) = P(A|B,C,D)P(B|C,D)P(C|D)P(D)$ .

Two events  $A$  and  $B$  are independent if  $P(A|B) = P(A)$ , i.e., if knowing that  $B$  happened doesn’t change the probability of  $A$  happening. An equivalent formulation is  $P(A,B) = P(A)P(B)$ . In general, multiplying probabilities involves the assumption that the corresponding events are independent.

The ‘odds’ of an event is the ratio of the probability that the event happens and the probability that it doesn’t happen. That is, if the probability of a particular event happening is  $p$ , then the corresponding odds are  $o = p/(1-p)$ . Conversely, we have that  $p = o/(o+1)$ . So, for example, a probability of 0.8 corresponds to odds of 4:1, the opposite odds of 1:4 give probability 0.2, and if the event is as likely to occur as not then the probability is 0.5 and the odds are 1:1. While we will most often use the probability scale, odds are sometimes more convenient because they are expressed on a multiplicative scale.

---

**Background 2.** The basics of probability.

The way to make sense of this is to realise that you are combining two independent pieces of evidence, one concerning the prevalence of spam, and the other concerning the occurrence of the word ‘Viagra’. These two pieces of evidence pull in opposite directions, which means that it is important to assess their relative strength. What the numbers tell you is that, in order to overrule the fact that spam is relatively rare, you need odds of at least 6:1. ‘Viagra’ on its own is estimated at 4:1, and therefore doesn’t pull hard enough in the spam direction to warrant the conclusion that the e-mail is in fact spam. What it does do is make the conclusion ‘this e-mail is ham’ a lot less certain, as its probability drops from  $6/7 = 0.86$  to  $6/10 = 0.60$ .

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 (i.e., there are three times more spam e-mails containing the phrase than there are ham e-mails), 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’).

The advantage of not having to estimate and manipulate joint probabilities is that we can handle large numbers of variables. Indeed, the vocabulary of a typical Bayesian spam filter or text classifier may contain some 10 000 terms.<sup>2</sup> So, instead of manually crafting a small set of ‘features’ deemed relevant or predictive by an expert, we include a much larger set and let the classifier figure out which features are important, and in what combinations.

### §

It should be noted that by multiplying the odds associated with ‘Viagra’ and ‘blue pill’, we are implicitly assuming that they are independent pieces of information. This is obviously not true: if we know that an e-mail contains the phrase ‘blue pill’, we are not really surprised to find out that it also contains the word ‘Viagra’. In probabilistic terms:

- ☞ the probability  $P(\text{Viagra}|\text{blue pill})$  will be close to 1;
- ☞ hence the joint probability  $P(\text{Viagra}, \text{blue pill})$  will be close to  $P(\text{blue pill})$ ;
- ☞ hence the odds of spam associated with the two phrases ‘Viagra’ and ‘blue pill’ will not differ much from the odds associated with ‘blue pill’ on its own.

Put differently, by multiplying the two odds we are counting what is essentially one piece of information twice. The product odds of 12:1 is almost certainly an overesti-

---

<sup>2</sup>In fact, phrases consisting of multiple words are usually decomposed into their constituent words, such that  $P(\text{blue pill})$  is estimated as  $P(\text{blue})P(\text{pill})$ .

mate, and the real joint odds may be not more than, say, 5:1.

We appear to have painted ourselves into a corner here. In order to avoid over-counting we need to take joint occurrences of phrases into account; but this is only feasible computationally if we define the problem away by assuming them to be independent. What we want seems to be closer to a rule-based model such as the following:

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

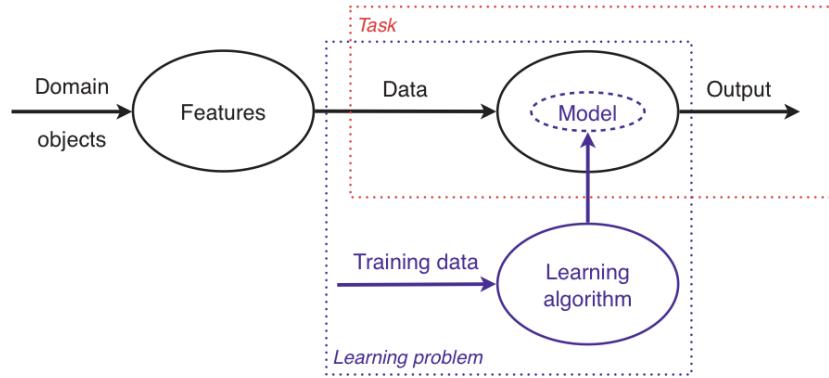
The essence of such rule-based classifiers is that they don’t treat all e-mails in the same way but work on a case-by-case basis. In each case they only invoke the most relevant features. Cases can be defined by several nested features:

1. Does the e-mail contain the word ‘Viagra’?
  - (a) If so: Does the e-mail contain the word ‘blue pill’?
    - i. If so: estimate the odds of spam as 5:1.
    - ii. If not: estimate the odds of spam as 4:1.
  - (b) If not: Does the e-mail contain the word ‘lottery’?
    - i. If so: estimate the odds of spam as 3:1.
    - ii. If not: estimate the odds of spam as 1:6.

These four cases are characterised by logical conditions such as ‘the e-mail contains the word “Viagra” but not the phrase “blue pill”’. Effective and efficient algorithms exist for identifying the most predictive feature combinations and organise them as rules or trees, as we shall see later.

## 5

We have now seen three practical examples of machine learning in spam e-mail recognition. Machine learners call such a task *binary classification*, as it involves assigning objects (e-mails) to one of two classes: spam or ham. This task is achieved by describing each e-mail in terms of a number of variables or features. In the SpamAssassin



**Figure 3.** 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).

example these features were handcrafted by an expert in spam filtering, while in the Bayesian text classification example we employed a large vocabulary of words. The question is then how to use the features to distinguish spam from ham. We have to somehow figure out a connection between the features and the class – machine learners call such a connection a *model* – by analysing a *training set* of e-mails already labelled with the correct class.

- ☞ In the SpamAssassin example we came up with a linear equation of the form  $\sum_{i=1}^n w_i x_i > t$ , where the  $x_i$  denote the 0–1 valued or ‘Boolean’ features indicating whether the  $i$ -th test succeeded for the e-mail,  $w_i$  are the feature weights learned from the training set, and  $t$  is the threshold above which e-mails are classified as spam.
- ☞ In the Bayesian example we used a decision rule that can be written as  $\prod_{i=0}^n o_i > 1$ , where  $o_i = P(\text{spam}|x_i)/P(\text{ham}|x_i)$ ,  $1 \leq i \leq n$ , are the odds of spam associated with each word  $x_i$  in the vocabulary and  $o_0 = P(\text{spam})/P(\text{ham})$  are the prior odds, all of which are estimated from the training set.
- ☞ In the rule-based example we built logical conditions that identify subsets of the data that are sufficiently similar to be labelled in a particular way.

Here we have, then, the main ingredients of machine learning: tasks, models and features. Figure 3 shows how these ingredients relate. If you compare this figure with Figure 2, you’ll see how the model has taken centre stage, rather than merely being a set of parameters of a classifier otherwise defined by the features. We need this flexibility to incorporate the very wide range of models in use in machine learning. It is worth

emphasising the distinction between tasks and learning problems: *tasks are addressed by models, whereas learning problems are solved by learning algorithms that produce models*. While the distinction is widely recognised, terminology may vary: for instance, you may find that other authors use the term ‘learning task’ for what we call a learning problem.

In summary, one could say that *machine learning is concerned with using the right features to build the right models that achieve the right tasks*. I call these ‘ingredients’ to emphasise that they come in many different forms, and need to be chosen and combined carefully to create a successful ‘meal’: what machine learners call an *application* (the construction of a model that solves a practical task, by means of machine learning methods, using data from the task domain). Nobody can be a good chef without a thorough understanding of the ingredients at his or her disposal, and the same holds for a machine learning expert. Our main ingredients of tasks, models and features will be investigated in full detail from [Chapter 2](#) onwards; first we will enjoy a little ‘taster menu’ when I serve up a range of examples in the next chapter to give you some more appreciation of these ingredients.



# CHAPTER 1

---

## The ingredients of machine learning

---

MACHINE LEARNING IS ALL ABOUT using the right features to build the right models that achieve the right tasks – this is the slogan, visualised in [Figure 3](#) on [p.11](#), with which we ended the [Prologue](#). In essence, *features* define a ‘language’ in which we describe the relevant objects in our domain, be they e-mails or complex organic molecules. We should not normally have to go back to the domain objects themselves once we have a suitable feature representation, which is why features play such an important role in machine learning. We will take a closer look at them in [Section 1.3](#). A *task* is an abstract representation of a problem we want to solve regarding those domain objects: the most common form of these is classifying them into two or more classes, but we shall encounter other tasks throughout the book. Many of these tasks can be represented as a mapping from data points to outputs. This mapping or *model* is itself produced as the output of a machine learning algorithm applied to training data; there is a wide variety of models to choose from, as we shall see in [Section 1.2](#).

We start this chapter by discussing tasks, the problems that can be solved with machine learning. No matter what variety of machine learning models you may encounter, you will find that they are designed to solve one of only a small number of tasks and use only a few different types of features. One could say that *models lend the machine learning field diversity, but tasks and features give it unity*.

## 1.1 Tasks: the problems that can be solved with machine learning

Spam e-mail recognition was described in the [Prologue](#). It constitutes a binary classification task, which is easily the most common task in machine learning which figures heavily throughout the book. One obvious variation is to consider classification problems with more than two classes. For instance, we may want to distinguish different kinds of ham e-mails, e.g., work-related e-mails and private messages. We could approach this as a combination of two binary classification tasks: the first task is to distinguish between spam and ham, and the second task is, among ham e-mails, to distinguish between work-related and private ones. However, some potentially useful information may get lost this way, as some spam e-mails tend to look like private rather than work-related messages. For this reason, it is often beneficial to view [\*multi-class classification\*](#) as a machine learning task in its own right. This may not seem a big deal: after all, we still need to learn a model to connect the class to the features. However, in this more general setting some concepts will need a bit of rethinking: for instance, the notion of a decision boundary is less obvious when there are more than two classes.

Sometimes it is more natural to abandon the notion of discrete classes altogether and instead predict a real number. Perhaps it might be useful to have an assessment of an incoming e-mail's urgency on a sliding scale. This task is called [\*regression\*](#), and essentially involves learning a real-valued function from training examples labelled with true function values. For example, I might construct such a training set by randomly selecting a number of e-mails from my inbox and labelling them with an urgency score on a scale of 0 (ignore) to 10 (immediate action required). This typically works by choosing a class of functions (e.g., functions in which the function value depends linearly on some numerical features) and constructing a function which minimises the difference between the predicted and true function values. Notice that this is subtly different from SpamAssassin learning a real-valued spam score, where the training data are labelled with classes rather than 'true' spam scores. This means that SpamAssassin has less information to go on, but it also allows us to interpret SpamAssassin's score as an assessment of how far it thinks an e-mail is removed from the decision boundary, and therefore as a measure of confidence in its own prediction. In a regression task the notion of a decision boundary has no meaning, and so we have to find other ways to express a model's confidence in its real-valued predictions.

Both classification and regression assume the availability of a training set of examples labelled with true classes or function values. Providing the true labels for a data set is often labour-intensive and expensive. Can we learn to distinguish spam from ham, or work e-mails from private messages, without a labelled training set? The answer is: yes, up to a point. The task of grouping data without prior information on the groups is called [\*clustering\*](#). Learning from unlabelled data is called [\*unsupervised learning\*](#) and is quite distinct from [\*supervised learning\*](#), which requires labelled training data. A typical

clustering algorithm works by assessing the similarity between instances (the things we're trying to cluster, e.g., e-mails) and putting similar instances in the same cluster and 'dissimilar' instances in different clusters.

---

**Example 1.1 (Measuring similarity).** If our e-mails are described by word-occurrence features as in the text classification example, the similarity of e-mails would be measured in terms of the words they have in common. For instance, we could take the number of common words in two e-mails and divide it by the number of words occurring in either e-mail (this measure is called the *Jaccard coefficient*). Suppose that one e-mail contains 42 (different) words and another contains 112 words, and the two e-mails have 23 words in common, then their similarity would be  $\frac{23}{42+112-23} = \frac{23}{130} = 0.18$ . We can then cluster our e-mails into groups, such that the average similarity of an e-mail to the other e-mails in its group is much larger than the average similarity to e-mails from other groups. While it wouldn't be realistic to expect that this would result in two nicely separated clusters corresponding to spam and ham – there's no magic here – the clusters may reveal some interesting and useful structure in the data. It may be possible to identify a particular kind of spam in this way, if that subgroup uses a vocabulary, or language, not found in other messages.

---

There are many other patterns that can be learned from data in an unsupervised way. *Association rules* are a kind of pattern that are popular in marketing applications, and the result of such patterns can often be found on online shopping web sites. For instance, when I looked up the book *Kernel Methods for Pattern Analysis* by John Shawe-Taylor and Nello Cristianini on [www.amazon.co.uk](http://www.amazon.co.uk), I was told that 'Customers Who Bought This Item Also Bought' –

- ☞ *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods* by Nello Cristianini and John Shawe-Taylor;
- ☞ *Pattern Recognition and Machine Learning* by Christopher Bishop;
- ☞ *The Elements of Statistical Learning: Data Mining, Inference and Prediction* by Trevor Hastie, Robert Tibshirani and Jerome Friedman;
- ☞ *Pattern Classification* by Richard Duda, Peter Hart and David Stork;

and 34 more suggestions. Such associations are found by data mining algorithms that zoom in on items that frequently occur together. These algorithms typically work by

only considering items that occur a minimum number of times (because you wouldn't want your suggestions to be based on a single customer that happened to buy these 39 books together!). More interesting associations could be found by considering multiple items in your shopping basket. There exist many other types of associations that can be learned and exploited, such as correlations between real-valued variables.

### *Looking for structure*

Like all other machine learning models, patterns are a manifestation of underlying structure in the data. Sometimes this structure takes the form of a single *hidden* or *latent variable*, much like unobservable but nevertheless explanatory quantities in physics, such as energy. 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?

If you are inclined to say no, try to look for columns or rows that are combinations of other columns or rows. For instance, the third column turns out to be the sum of the first and second columns. Similarly, the fourth row is the sum of the first and second rows. What this means is that the fourth person combines the ratings of the first and second person. Similarly, *The Godfather*'s ratings are the sum of the ratings of the first two films. This is made more explicit by writing the matrix as the following product:

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

You might think I just made matters worse – instead of one matrix we now have three! However, notice that the first and third matrix on the right-hand side are now Boolean,

and the middle one is diagonal (all off-diagonal entries are zero). Moreover, these matrices have a very natural interpretation in terms of film *genres*. 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: the first, fourth and fifth person like drama, the second, fourth and fifth person like crime films, and the third, fifth and sixth person like comedies. 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.

Methods for discovering hidden variables such as film genres really come into their own when the number of values of the hidden variable (here: the number of genres) is much smaller than the number of rows and columns of the original matrix. For instance, at the time of writing [www.imdb.com](http://www.imdb.com) lists about 630 000 rated films with 4 million people voting, but only 27 film categories (including the ones above). While it would be naive to assume that film ratings can be completely broken down by genres – genre boundaries are often diffuse, and someone may only like comedies made by the Coen brothers – this kind of *matrix decomposition* can often reveal useful hidden structure. It will be further examined in Chapter 10.

This is a good moment to summarise some terminology that we will be using. We have already seen the distinction between supervised learning from labelled data and unsupervised learning from unlabelled data. We can similarly draw a distinction between whether the model output involves the target variable or not: we call it a *predictive model* if it does, and a *descriptive model* if it does not. This leads to the four different machine learning settings summarised in Table 1.1.

- ☞ The most common setting is supervised learning of predictive models – in fact, this is what people commonly mean when they refer to supervised learning. Typical tasks are classification and regression.
- ☞ It is also possible to use labelled training data to build a descriptive model that is not primarily intended to predict the target variable, but instead identifies, say, subsets of the data that behave differently with respect to the target variable. This example of supervised learning of a descriptive model is called *subgroup discovery*; we will take a closer look at it in Section 6.3.
- ☞ Descriptive models can naturally be learned in an unsupervised setting, and we have just seen a few examples of that (clustering, association rule discovery and matrix decomposition). This is often the implied setting when people talk about unsupervised learning.
- ☞ A typical example of unsupervised learning of a predictive model occurs when

	<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

**Table 1.1.** An overview of different machine learning settings. 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.

we cluster data with the intention of using the clusters to assign class labels to new data. We will call this *predictive clustering* to distinguish it from the previous, *descriptive* form of clustering.

Although we will not cover it in this book, it is worth pointing out a fifth setting of *semi-supervised learning* of predictive models. In many problem domains data is cheap, but labelled data is expensive. For example, in web page classification you have the whole world-wide web at your disposal, but constructing a labelled training set is a painstaking process. One possible approach in semi-supervised learning is to use a small labelled training set to build an initial model, which is then refined using the unlabelled data. For example, we could use the initial model to make predictions on the unlabelled data, and use the most confident predictions as new training data, after which we retrain the model on this enlarged training set.

### Evaluating performance on a task

An important thing to keep in mind with all these machine learning problems is that they don't have a 'correct' answer. This is different from many other problems in computer science that you might be familiar with. For instance, if you sort the entries in your address book alphabetically on last name, there is only one correct result (unless two people have the same last name, in which case you can use some other field as tie-breaker, such as first name or age). This is not to say that there is only one way of achieving that result – on the contrary, there is a wide range of sorting algorithms available: insertion sort, bubblesort, quicksort, to name but a few. If we were to compare the performance of these algorithms, it would be in terms of how fast they are, and how much data they could handle: e.g., we could test this experimentally on real data, or analyse it using computational complexity theory. However, what we wouldn't do is compare different algorithms with respect to the correctness of the result, because an algorithm that isn't guaranteed to produce a sorted list every time is useless as a sorting algorithm.

Things are different in machine learning (and not just in machine learning: see

Background 1.1). We can safely assume that the perfect spam e-mail filter doesn't exist – if it did, spammers would immediately 'reverse engineer' it to find out ways to trick the spam filter into thinking a spam e-mail is actually ham. In many cases the data is 'noisy' – examples may be mislabelled, or features may contain errors – in which case it would be detrimental to try too hard to find a model that correctly classifies the training data, because it would lead to overfitting, and hence wouldn't generalise to new data. In some cases the features used to describe the data only give an indication of what their class might be, but don't contain enough 'signal' to predict the class perfectly. For these and other reasons, machine learners take performance evaluation of learning algorithms very seriously, which is why it will play a prominent role in this book. We need to have some idea of how well an algorithm is expected to perform on new data, not in terms of runtime or memory usage – although this can be an issue too – but in terms of classification performance (if our task is a classification task).

Suppose we want to find out how well our newly trained spam filter does. One thing we can do is count the number of correctly classified e-mails, both spam and ham, and divide that by the total number of examples to get a proportion which is called the *accuracy* of the classifier. However, this doesn't indicate whether overfitting is occurring. A better idea would be to use only 90% (say) of the data for training, and the remaining 10% as a *test set*. If overfitting occurs, the test set performance will be considerably lower than the training set performance. However, even if we select the test instances randomly from the data, every once in a while we may get lucky, if most of the test instances are similar to training instances – or unlucky, if the test instances happen to be very non-typical or noisy. In practice this train–test split is therefore repeated in a process called *cross-validation*, further discussed in Chapter 12. This works as follows: we randomly divide the data in ten parts of equal size, and use nine parts for training and one part for testing. We do this ten times, using each part once for testing. At the end, we compute the average test set performance (and usually also its standard deviation, which is useful to determine whether small differences in average performance of different learning algorithms are meaningful). Cross-validation can also be applied to other supervised learning problems, but unsupervised learning methods typically need to be evaluated differently.

In Chapters 2 and 3 we will take a much closer look at the various tasks that can be approached using machine learning methods. In each case we will define the task and look at different variants. We will pay particular attention to evaluating performance of models learned to solve those tasks, because this will give us considerable additional insight into the nature of the tasks.

---

Long before machine learning came into existence, philosophers knew that generalising from particular cases to general rules is not a well-posed problem with well-defined solutions. Such inference by generalisation is called *induction* and is to be contrasted with *deduction*, which is the kind of reasoning that applies to problems with well-defined correct solutions. There are many versions of this so-called *problem of induction*. One version is due to the eighteenth-century Scottish philosopher David Hume, who claimed that the only justification for induction is itself inductive: since it appears to work for certain inductive problems, it is expected to work for all inductive problems. This doesn't just say that induction cannot be deductively justified but that its justification is circular, which is much worse.

A related problem is stated by the *no free lunch theorem*, which states that no learning algorithm can outperform another when evaluated over all possible classification problems, and thus the performance of any learning algorithm, over the set of all possible learning problems, is no better than random guessing. Consider, for example, the 'guess the next number' questions popular in psychological tests: what comes after 1, 2, 4, 8, ...? If all number sequences are equally likely, then there is no hope that we can improve – on average – on random guessing (I personally always answer '42' to such questions). Of course, some sequences are very much more likely than others, at least in the world of psychological tests. Likewise, the distribution of learning problems in the real world is highly non-uniform. The way to escape the curse of the no free lunch theorem is to find out more about this distribution and exploit this knowledge in our choice of learning algorithm.

---

**Background 1.1.** Problems of induction and free lunches.

## 1.2 Models: the output of machine learning

Models form the central concept in machine learning as they are what is being learned from the data, in order to solve a given task. There is a considerable – not to say bewildering – range of machine learning models to choose from. One reason for this is the ubiquity of the tasks that machine learning aims to solve: classification, regression, clustering, association discovery, to name but a few. Examples of each of these tasks can be found in virtually every branch of science and engineering. Mathematicians, engineers, psychologists, computer scientists and many others have discovered – and sometimes rediscovered – ways to solve these tasks. They have all brought their