Prologue: A machine learning sampler

YOU 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![1] 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
```

[1] 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` – 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 δ, 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 $(w_1, \ldots, w_n)$ and $\mathbf{x}$ for the vector of test results $(x_1, \ldots, x_n)$. 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 $\mathbf{x}^\circ = (1, x_1, \ldots, x_n)$ and the extended weight vector is $\mathbf{w}^\circ = (-t, w_1, \ldots, w_n)$, leading to the decision rule $\mathbf{w}^\circ \cdot \mathbf{x}^\circ > 0$ and the decision boundary $\mathbf{w}^\circ \cdot \mathbf{x}^\circ = 0$. Thanks to these so-called homogeneous coordinates the decision boundary passes through the origin of the extended coordinate system, at the expense of needing an additional dimension (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 or mathworld.wolfram.com – to fully appreciate the rest of the book.

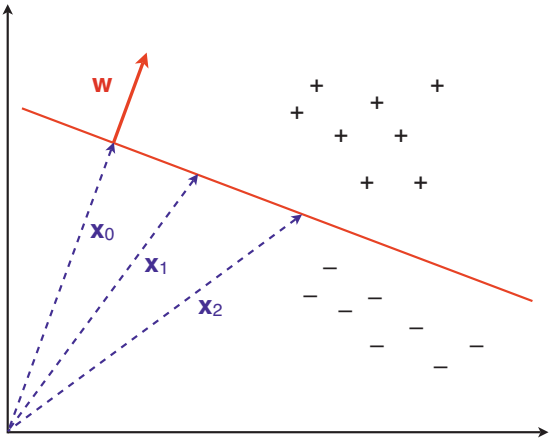

Figure 1. An example of linear classification in two dimensions. The straight line separates the positives from the negatives. It is defined by $\mathbf{w} \cdot \mathbf{x}_i = t$, where $\mathbf{w}$ is a vector perpendicular to the decision boundary and pointing in the direction of the positives, t 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.

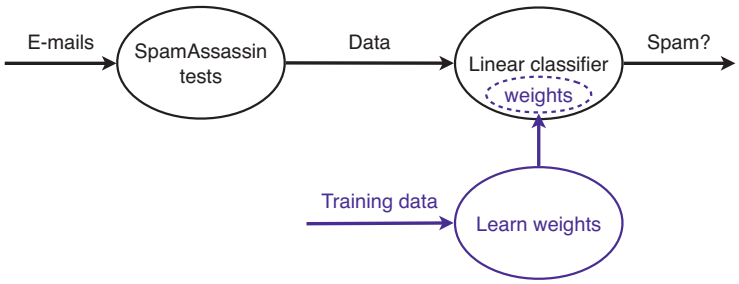

Figure 2. At the top we see how SpamAssassin approaches the spam e-mail classification task: the text of each e-mail is converted into a data point by means of SpamAssassin's built-in tests, and a linear classifier is applied to obtain a 'spam or ham' decision. At the bottom (in blue) we see the bit 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 $P(\text{PM}|\text{woman}) = P(\text{PM},\text{woman})/P(\text{woman})$, where $P(\text{PM},\text{woman})$ 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.[2] 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-

[2] 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 overcounting 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.

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