1. Introduction

The following machine learning algorithms are considered in the current course: naive Bayes decision rule, discriminant analysis, nearest neighbor searching, discriminant analysis, clustering, decision trees, ensemble models with random forests, support vector machines, and the basic genetic algorithm.

We also treat testing or validation techniques always required when machine learning methods are used for actual data. If one did not know them well, troubles would arise. Machine learning algorithms are applied to build computational *models*. Testing or validation techniques are used to test how well and accurately those models function. Sometimes they do not.

1.1 About machine learning: Example 1

Let us begin to make acquainted with simple examples. Linear classification task is binary at its simplest. We are interested in how a spam filter could detect spam messages. A training set is required to show which messages are spam and which are not. Now the latter are called here as ’non-spam’. All training cases are hand-labeled spam or non-spam. Thus, we know the results of all the tests for each of these emails. We want to determine a weight for each so that if a weight is above score 5, it is said to be spam, but otherwise non-spam. Suppose that there are (unrealistically few) only two tests and four emails, one of which is spam (Table 1.1). Both tests succeeded for the spam email. For non-spam either both tests or one of them did not succeed.

Table 1.1 A small fictive training set for the spam filter. The columns with x_1 and x_2 indicate the results of two tests on four different emails. The fourth column indicated which of the emails are spam. The right-most column shows that by thresholding the function $4x_1+4x_2$ at 5 we can discern spam from non-spam.

Email	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

By assigning both tests a weight of 4 correctly ”classifies” four emails into spam or non-spam. To express it precisely we can define a simple classifier as follows:

$$4x_1+4x_2 > 5 \quad \text{or} \quad (4,4)\cdot(x_1,x_2) > 5$$

In fact, any weight or coefficient between 2.5 and 5 will ensure that the threshold of 5 is only exceeded when both tests succeed. We could even assign different weights to the tests as long as each weight is less than 5 and their sum exceeds 5. It is hard to see how this could be justified by the training data. Perhaps we could know about variables $x_1$and x_2 something in advance.

What does this have to do with learning? Collecting training data that include recognised spam emails and counter-examples it is possible to develop better computational models, binary classifiers here. The notion of performance improving with experience is central to most, if not all, sorts of machine learning. Let us look at the general definition:

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

In the spam filter, the ”experience” it learns from is some correctly labeled training data, and ”performance” denotes its ability to detect spam email.

(Nor always spam filters are ”intelligent”. For example, once the author sent an email as reply for eight people including himself. The system marked that message to the sender himself as a possible spam.)

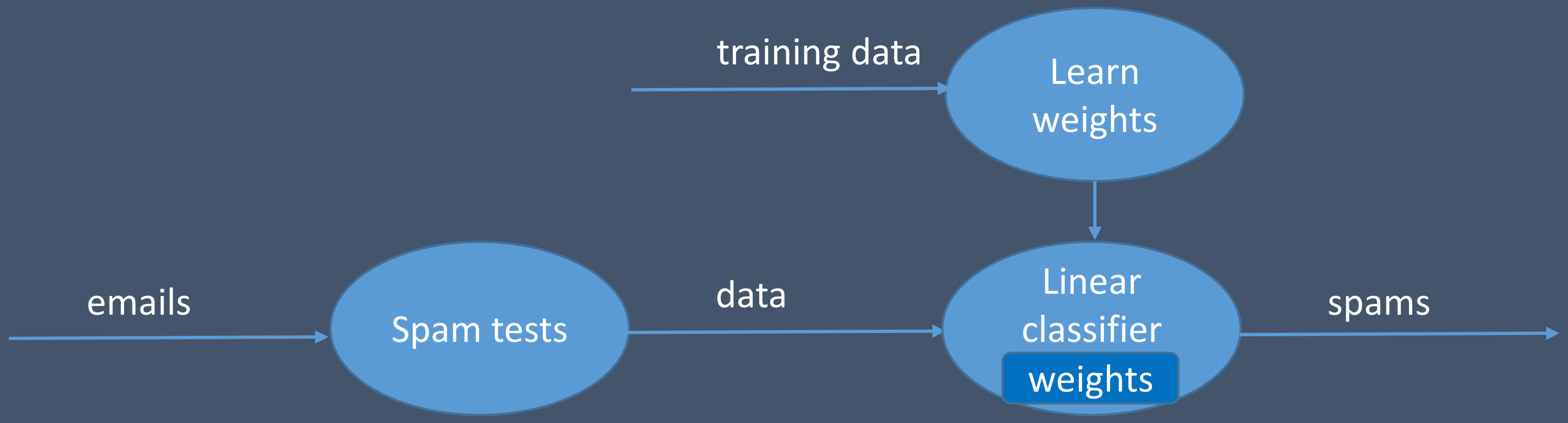


Fig. 1.1 This presents a schematic view how machine learning feeds into the spam email classification task. In another machine learning problem, experience may take a different form, such as corrections of mistakes, rewards when a certain goal is reached, among others.

There are various ways how to express the linear classifier considered in mathematical notation, i.e., in a general way. If we denote the result of the *i*th test for a given email as x_i, where $x_i = 1$ if the test succeeds and 0 otherwise, and we denote the weight of the *i*th test w_i, then the total score of an email can be as follows:

$$\sum_{i=1}^{p} w_i x_i$$

Using t for the threshold above which an email is classified as spam, the ”decision rule” can be written as the following inequality, and also as dot product (see also Fig. 1.2) for p variables:

$$\sum_{i=1}^{p} w_i x_i = \mathbf{w} \cdot \mathbf{x} > t$$

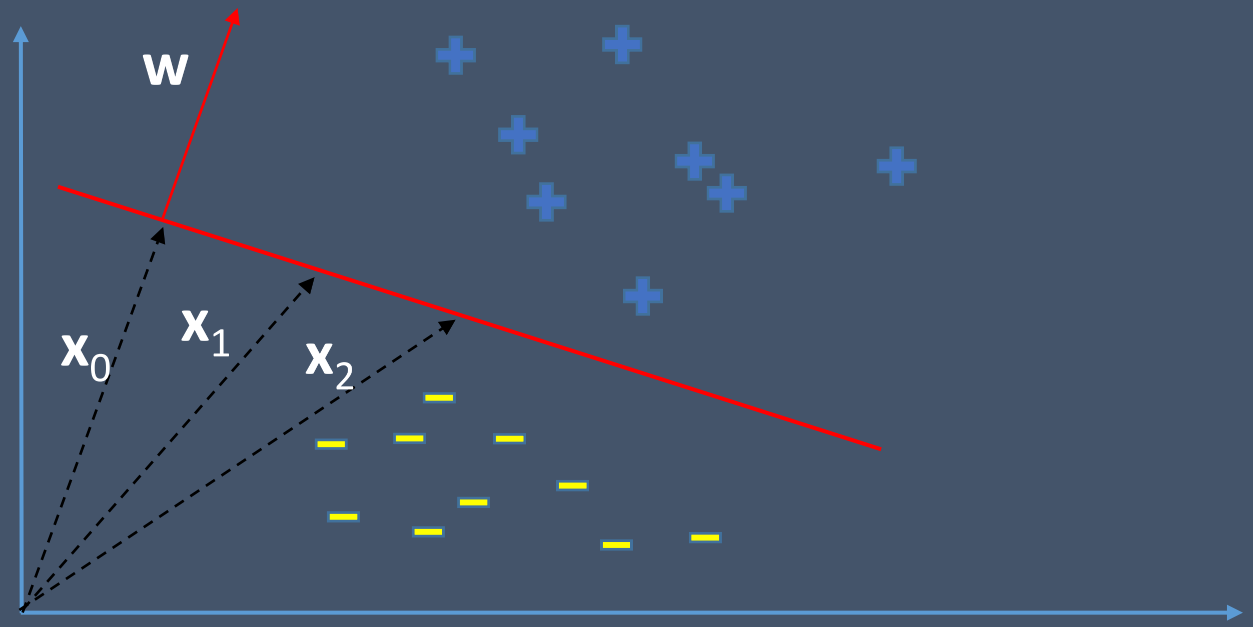


Fig. 1.2 An example of linear classification in two dimensions. The straight line (decision boundary) defined by $\mathbf{w}\cdot\mathbf{x}_i>t$ separates the positives from the negatives, and is defined by dot product (inner product) $\mathbf{w}\cdot\mathbf{x}=t$ of weight vector **w** and input vector $\mathbf{x}_i$, where **w** is perpendicular to the *decision boundary* and $\mathbf{x}_i$ points to a point on the decision boundary. Since $\mathbf{x}_0$ points in the same direction as **w**, it is that $\mathbf{w}\cdot(\mathbf{x}-\mathbf{x}_0)=0$ also describes the decision boundary.

Example 2: Overfitting

Imagine that one is preparing for an exam. Dr. Jekyll has made previous exam papers and their worked answers available online. One begins by attempting to answer the questions from previous papers and comparing one's answers with the model answers provided. Unfortunately, one gets carried away and spends all the time on memorizing the model answers to all past questions. Now, if the upcoming exam completely consists of past questions, one is certain to do very well. If the new exam contains different questions about the same material, one would be ill-prepared and get a far lower mark than with a more extensive preparation. In this case, we could say that one was *overfitting*, *overlearning* or *overtraining* the past exam papers and that the knowledge gained did not *generalize* to future exam questions.

Generalization is probably the most fundamental concept in machine learning. If the knowledge that the previous spam filter has gleaned from its training data carries over, in other words, generalizes, to one’s emails, one is happy, but otherwise one starts looking for a better spam filter. Yet, overfitting is not the only possible reason for poor performance on new data. It may be that the training data applied by the spam filter programmers to set its weights is not repsentative for a kind of emails one gets. This problem does have a solution: one uses different training data that exhibits the same characteristics , if possible actual spam and non-spam emails that one has personally received. Machine learning is an efficient technology for adapting the behavior of software to one’s own personal circumstances. Many spam filters allow the use of one’s own training data.

Example 3: Repsentative training and test sets

An important property of a *training set* is that it represents statistically well enough the data source from which it is originated. Naturally, a *test set* to be used to validate the model formed has to come from the same data source.

For example, when the author of the current text acquired a laptop computer several years ago, a machine including biometric fingerprint authentication was ordered. We studied biometric verification and identification and, thus, it was interesting to see how fingerprint authentication functions in this computer. At the beginning, it was somewhat surprising that four fingerprint images only were measured to start the training set. It is probably too small number, particularly, since for fingerprint images dozens of features (variables) or more are determined to compute authentication. *Biometric authentication or verification* means binary classification: either the one logging in is the correct user or is not.

Perhaps, the designer of the system has thought that a new user, while starting to apply biometric authentication, is too lazy to input, for instance, 10 or 20 fingerprints for his or her personal training set.

What happened when the author started to use the system? It required quite many times before the biometric authentication started to function tolerably. See Fig. 1.3. After some 20 successful logins it learnt. The author did not wonder why some other (not knowing machine learning) who acquired the corresponding authentication system got tired of attempting so many times and gave up.

Later, verification functioned quite well, say, required 1-4 trials before an accepted one. The system has to adapt the model from time to time by computing a fresh model from its newest data.

Note! We cannot conclude on the basis of one laptop a decision that its biometric authentication could have been implemented in a better way. There should be 20-30 similar machines and systems tested to enable a more general conclusion.

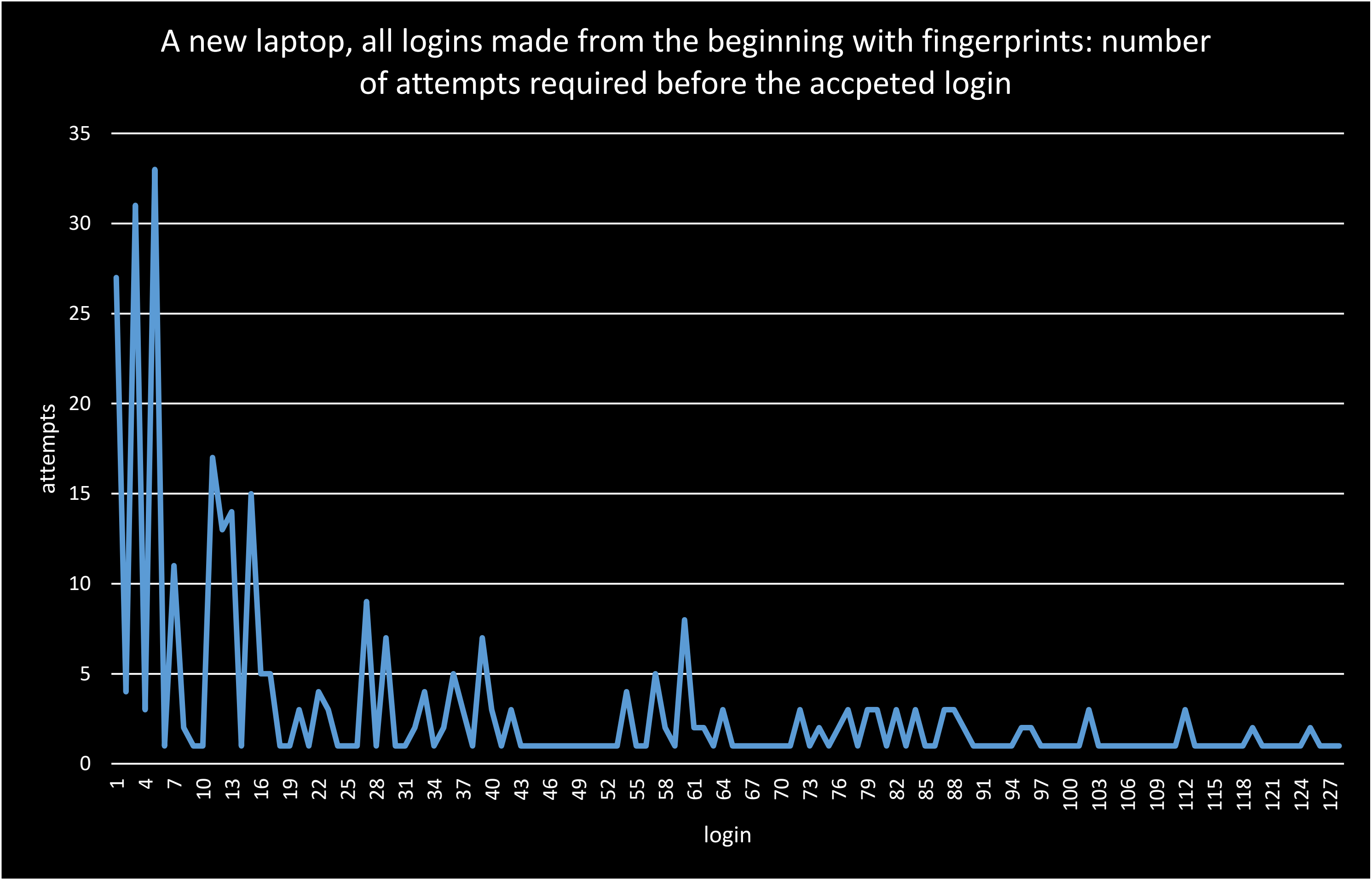


Fig. 1.3 Numbers of trials required up to the accepted biometric fingerprint authentication after the first time using the author’s current laptop. The ”test” included 128 turning on of the machine and beginning to use it.

Brief overview of the use of machine learning

We have seen a few simple, but practical examples of machine learning. These are of the binary classification type as they involve assigning objects to one of two classes. Such a classification task is, of course, the simplest one as to the number of classes. The data of variables, features or attributes have to be collected. The question is how to use the features to distinguish one class from the other. We have to figure out a connection between the features and the class. In machine learning it is called a *model* which is constructed by analysing a training set already *labeled* with the correct class. Fig. 1.4 illustrates the process at its highest level.

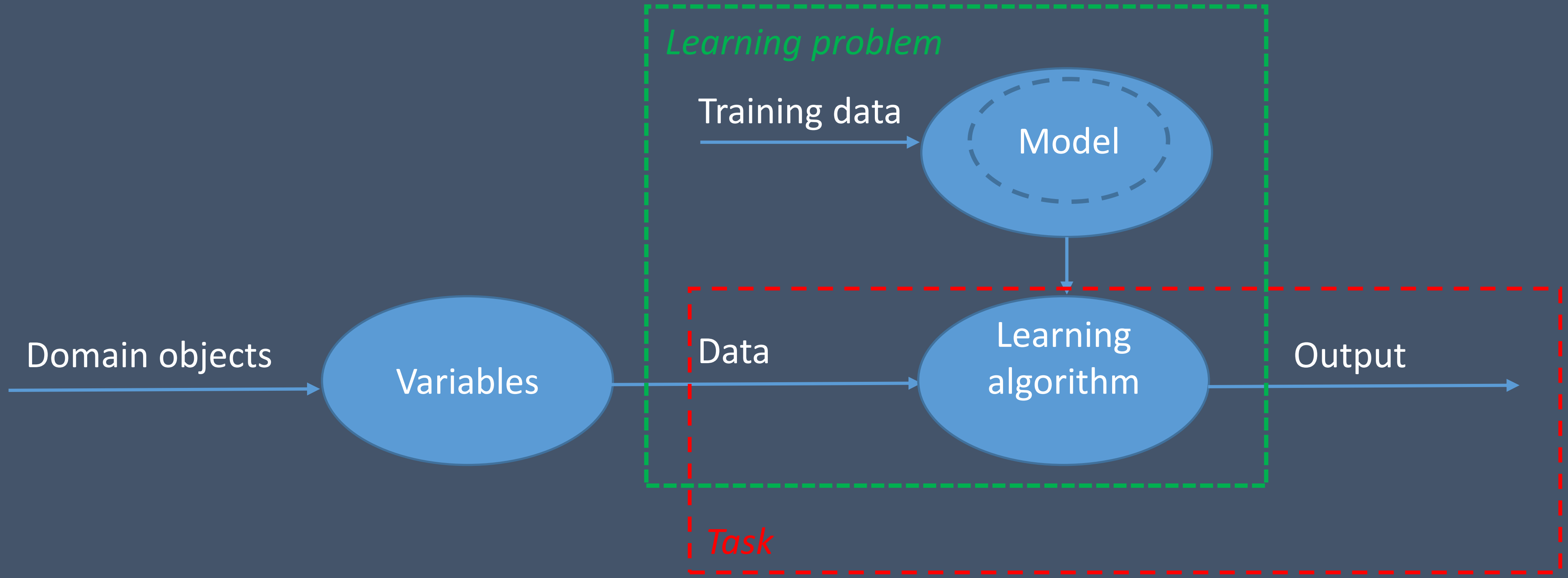


Fig. 1.4 An overview of how machine learning is employed to address a given task. A task requires an appropriate mapping – a model – from data described by variables to outputs. Obtaining a mapping on the basis of training data is what constitutes a learning problem.

1.2 Tasks: about problems that can be solved with machine learning

Up to now, only binary or 2-class classification such as spam filtering has been viewed. It is often to deal with *multiclass classification* of *C* classes where *C*>2 (rarely *C*>>2). This may not seem a big deal since, after all, we still need to train a model to connect the class to the variables. On the other hand, in this more general setting some concepts will need a bit more complicated rethinking and then processing. For example, the notion of a decision boundary (Fig. 1.2.) is less obvious when there are more than two classes.

Sometimes it is more natural and necessary to abandon the notion of discrete classes and instead *predict* a real number. Perhaps it might be useful to have an assessment of an incoming email’s urgency on a sliding scale. This task is called *regression*, which involves learning a real-valued function from training cases labeled with known, true function values. For example, one might construct a training set by randomly selecting a number of emails from the inbox labeling them with an urgency score on real scale [0,10] where 0 is equal to ’ignore’ and 10 ’immediate action required’. This typically works by choosing a class of functions, e.g., in which the function value depends linearly on some numerical variables and constructing a function which minimizes the difference between the predicted and true function values.

Both classification and regression presume the availability of a training set of cases (examples or instances) labeled with true classes or function values. Providing the true labels is often laborious and expensive. Can we train to distinguish spam from non-spam without a labeled 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 unlabeled data is called *unsupervised learning* and is quite distinct from *supervised learning*, which requires labeled training data.

A typical clustering works by assessing the similarity or dissimilarity (distance is quite the same notion) between cases (observations, instances or items).

Example 4: Measuring binary similarity

If emails are described by word-occurrence variables as usual in the text classification, the similarity of emails would be measured in terms of the words they have in common. For example, we could take the number of common words in two emails and divide it by the number of words occurring in either one email. This is called *Jaccard coefficient*. Say that one email contains 42 different words and another contains 111 different words, and the two emails have 23 words in common. Then their Jaccard similarity would be the following.

$$\frac{23}{42+111-23}=\frac{23}{130}\approx 0.18$$

We can then cluster emails into groups such that the average similarity of an email to the other email in its group is much larger than the average similarity to emails from other groups. While it is hardly realistic to expect that this would result in two nicely separated clusters corresponding to spam and non-spam, the clusters may reveal some interesting and useful structure in the data, like a particular kind of spam.

Association rules

Association rules are a computational method used in an unsupervised way and are popular in marketing applications, e.g., for recommendation purposes. For instance, when looking for text books for the current course with term ’machine learning’, one is told that ”customers who bought a certain machine learning item also bought some other books listed”. Such associations are found by algorithms that look for items that frequently occur together.

(We do not learn association rule methods in the current course, since they are involved in others.)

Looking for structure

Like all machine learning models, patterns are a manifestation of underlying structure in the data. Sometimes such takes the form of a single *hidden* or *latent* variable. Let us look at a 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 that these represent ratings by six people (in rows) on a scale of 0 to 3, of four different movies, e.g., The Shawshank Redemption, The Usual Suspects, The Godfather and The Big Lebowski (in columns, from left to right). The Godfather and The Big Lebowski seem to be the most popular with an average rating of 1.5, and The Shawshank Redemption is the least appreciated with 0.5. Can we see any structure in the matrix?

If one thinks ’no’, it is better to look at columns or rows that are combinations of other columns or rows. The third column turns out to be the sum of the first and second columns. The fourth row is the sum of the first and second rows. This means that the fourth person combines the ratings of the first person and second person. The Godfather’s ratings are the sum of the first two movies. To be more explicit see:

$$\text{persons}\left\{\begin{pmatrix}1&0&1&0\\0&2&2&2\\0&0&0&1\\1&2&3&2\\1&0&1&1\\0&2&2&3\end{pmatrix}\right. = \underbrace{\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\\1&1&0\\1&0&1\\0&1&1\end{pmatrix}}_{\text{genres}}\begin{pmatrix}1&0&0\\0&2&0\\0&0&1\end{pmatrix}\underbrace{\begin{pmatrix}1&0&1&0\\0&1&1&1\\0&0&0&1\end{pmatrix}}_{\text{movies}}\left.\vphantom{\begin{pmatrix}1\\0\\0\end{pmatrix}}\right\}\text{genres}$$

The original matrix could be decomposed into the product of three separate matrices. Notice that the first and third matrix on the right are Boolean or binary and the middle one is diagonal, all off-diagonal entries are zero. These matrices have a natural interpretation in terms of movie genres. The rightmost matrix associates movies (in columns) with genres (in rows): The Shawhank 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 movie and also introduces a new genre, comedy.

The 6×3 matrix expresses people’s preferences in terms of genres: the first, fourth and fifth person like drama, the second, fourth and sixth person like crime movies, and the third, fifth and sixth person like comedies. 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 movie genres here come into their own when the number of values of the hidden variable (the number of genres) is far smaller than the number of rows and columns of the original matrix. For instance, www.imdb.com lists hundreds of thousands of movies in 27 categories. In reality, genre boundaries are diffuse. Thus, such ratings cannot entirely be based on genres.

Previously, we have mentioned the distinction between supervised learning from labeled data and unsupervised learning from unlabeled data. Similarly, we can 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 four settings summarised in Table 1.2.

(1) The most common setting is supervised learning of predictive models. Typical tasks are classification and regression.

(2) Using labeled training data to build a descriptive model identifies, say, subsets of data that behave differently with respect to the target variable. This example of supervised learning of a descriptive model is called subgroup discovery. Association rule mining is a typical task.

Table 1.2 An overview of different machine learning settings.

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

(3) Descriptive models can be learned in an unsupervised setting. This is like those previous: clustering, association rules and matrix decomposition.

(4) An example of unsupervised learning of a predictive model occurs when we cluster data with the intention to apply the clusters to assign class labels to new data.

Evaluating performance on a task

An important matter to keep in mind as to machine learning problems is that they do not have an ”entirely correct” answer. This is different from many other ”yes/no problems” in computer science. For instance, if one sorted names alphabetically on last name, there would only be one correct result unless two or more people have the same last name, in which case one can use some other field as tie-breaker, such as first name, age or address.

Things are different in machine learning. We can assume that the perfect spam email filter does not exist. Often the data is noisy, e.g., cases or examples may include more or less erroneous variable or feature values. In some situations the variables used to describe the data only give an indication of what their class might be, but do not contain enough information to predict the class perfectly. For these and other reasons, machine learners take (classification) performance evaluation very seriously.

How well does our newly trained spam filter perform? We can count the number of correctly classified emails, both spam and non-spam, and divide that by the total number of cases to obtain the proportion called *accuracy* of the classifier. This does not, however, indicate whether overfitting is occurring. A better way is to apply a technique, where the data is divided into a *training set* of, say, 90%, and the remaining 10% into a *test set*. Although we select the test cases *randomly* from the data, every once in a while we may get lucky, if most of the test cases are similar to training cases, or unlucky, if the test cases happen to be very atypical or noisy.

In practice the train-test split is repeated in a process called crossvalidation. We randomly divide the data in 10 parts of (almost) equal size, and use nine parts, each 10% of the data, for training and one part for testing. We do this 10 times, using each part once for testing. (Typically, even more tests than 10 are run in practice as will be explained later on.) At the end, we calculate the average test performance and usually also its standard deviation, which is useful to determine whether small differences in average performance of different learning algorithms are meaningful. Crossvalidation is applied to supervised learning tasks, but unsupervised methods typically need to be evaluated differently. There are also other testing techniques.

1.3 Variables in machine learning

In addition to the material presented in the course of Data Mining, we consider briefly some aspects of variables (features or attributes) that determine much of the success of a machine learning application, because a model is only as good as its variables.

A variable is a kind of measurement that can be easily performed in any instance. Mathematically, they are functions that map from the instance or case space to some set of variable values called the *domain* of the variable. Since measurements are often numerical, the most common variable domain is the set of real numbers. They can also be integers, for example when the variable counts something, such as the number of occurrences of a particular words, or binary (Boolean), if the variable is a statement that can be true or false for a particular case, and arbitrary finite sets, such as a set of colors or shapes.

The use of variables

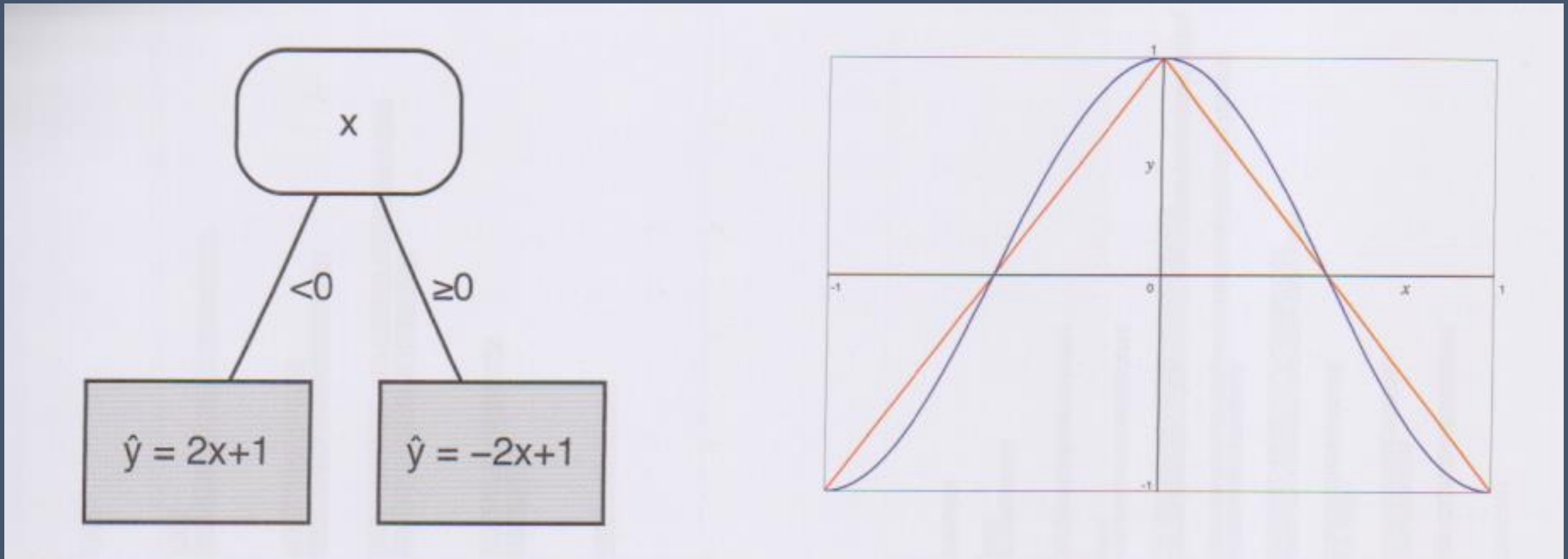
It is worth noting that variables and models are intimately connected, not just because models are defined in terms of variables, but because a single variable can be turned into what is sometimes called a *univariate* model. Such is often used as a part of the larger whole. For example, let *f* be variable counting the number of occurrences of the word ’inheritance’ in an email and let *x* be an arbitrary email. Then the condition $f(x)=0$ selects emails that do not comprise ’inheritance’, $f(x)\neq0$ or $f(x)>0$ selects emails that do, $f(x)\geq2$ selects emails containing that word at least twice, and so on. Such conditions are called *binary splits* that are important for decision trees among others. Naturally, a condition may be more complicated and include several parts that give, e.g., a four-way split of the instance space.

Another use of variables arises particularly in supervised learning. A linear classifier employs a decision rule of the form

$$\sum_{i=1}^{p} w_i x_i > t$$

where is *x_i* a numerical variable and *w_i* its weight or coefficient expressing its importance. For instance, if some *w_i*≈0, its influence is negligible. Thus, the variable makes a precise and measureable contribution to the final prediction.

Fig. 1.5 depicts an example of two uses of variables.



(a) (b)

Fig. 1.5 (a) A regression tree combining a one-split variable tree with linear regression models in the leaves. Here *x* is used both a splitting variable and a regression variable (ŷ means an estimate given by regression analysis). (b) Function *y* = cos *πx* on the interval [-1,1], and the piecewise linear approximation achieved by the regression tree.

Variable construction and transformation

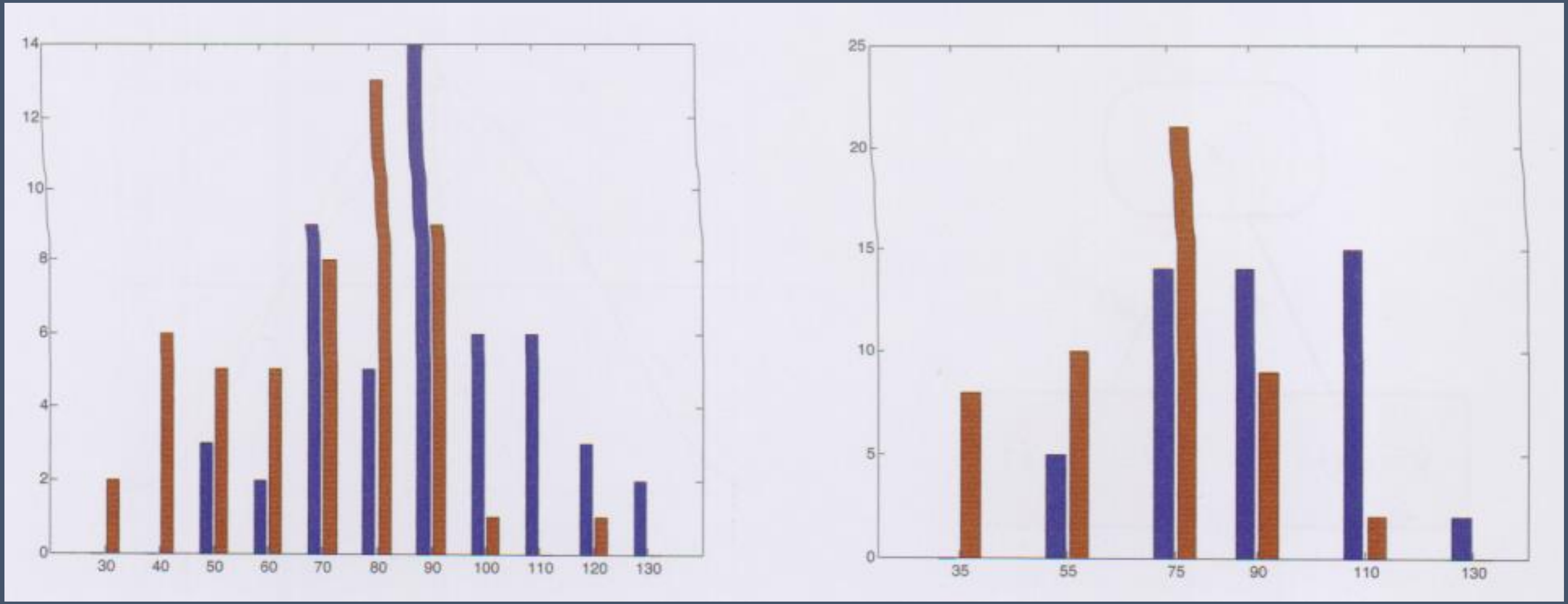
The *variable construction* process is crucial for the success of a machine learning application.

Indexing an email by the words that occur in it, called a bag of words as it disregards the order of words in the email text, is a carefully engineered representation that manages to amplify the ”signal” and to attennuate the ”noise” in spam email filtering and related text analysis tasks.

Nevertheless, it is easy to conceive of problems where this would be the wrong way to do: if we aim to train a classifier to distinguish between grammatical and ungrammatical sentences, word order is clearly signal rather than noise, and a different representation is called for.

It is often simple and natural to form a model in terms of of variables measured. Still, we may change the variable to a more appropriate form. For instance, a real-valued variable sometimes contain an ”unnecessary” detail that can be removed by *discretization*. A more important cause is that after that preprocessing we can apply such computational methods that require discrete values.

Fig. 1.6 shows an example where artificial body weight measurements of people with and without diabetes are depicted, first as quite fine graded and second the preceding values discretized.

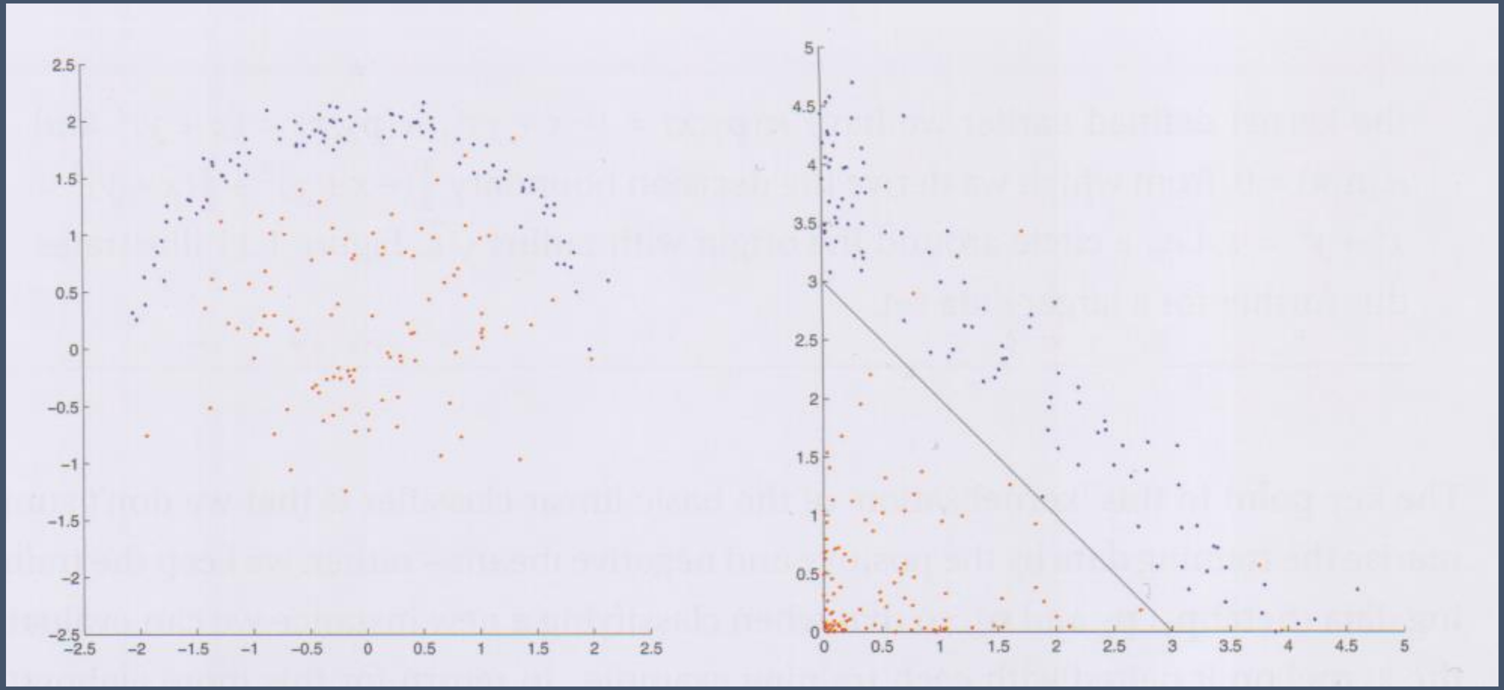


(a)

(b)

Fig. 1.6 (a) Artificial data depicting a histogram of body weight measurements of people with (blue, left bar) and without (right bar) diabetes, with 11 intervals of 10 kg each. (b) By joining the first and second, third and fourth, fifth and sixth, and eight, ninth and tenth intervals, such discretization is obtained that the proportion of diabetes cases increases from left to right. This makes the variable more useful in predicting diabetes.

The previous example illustrates how, for particular task as classification, we can improve the signal-to-noise ratio of a variable. In more extreme situations of variable construction, we transform the entire instance (variable) space. In Fig. 1.7 (a) the data is clearly not linearly separable, but by mapping the space into a new ’variable space’ consisting of the squares of the original variables we see that the data becomes almost linearly separable. As a matter of fact, by taking a third variable we perform a remarkable trick: it is possible to build a classifier without actually constructing a new variable space.



(a) (b)

Fig. 1.7 (a) A linear classifier would perform poorly on the current data (the curved subset of the blue above the round subset of the red). (b) By transforming 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 (the red mostly below the line and the blue above it). In the original space this corresponds to a circle with radius $3^{1/2}$ around the origin.

Example: the kernel trick

Let $\mathbf{x}_1=(x_1,y_1)$ and $\mathbf{x}_2=(x_2,y_2)$ be two data points (vectors), and consider the mapping

$$(x,y)\rightarrow(x^2,y^2,\sqrt{2}xy)$$

to a three-dimensional variable space. The points in variable space corresponding to vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ are as follows:

$$\mathbf{x}'_1=(x_1^2,y_1^2,\sqrt{2}x_1y_1)\ \text{and}\ \mathbf{x}'_2=(x_2^2,y_2^2,\sqrt{2}x_2y_2)$$

The dot product of these two variable vectors is as follows:

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

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