Universiteit Leiden
The Netherlands

Massively collaborative machine learning

Rijn, J.N. van

2 Machine Learning

As argued in Chapter 1, there is a great amount of data being generated every day. Collecting data in itself is only useful if we are able to make sense out of it. A great challenge lies in analysing and modelling the collected data.

2.1 Introduction

Model building is a time-consuming task that has already been practised for a long time by many scientists. In the 16th century, Nicolaus Copernicus described a model that considered the sun to be the centre of the universe, rather than the Earth. The data that he based this model on were the observations he made of the sun and its orbiting planets. Although his model was not entirely accurate, it is widely accepted that his ideas led to great scientific breakthroughs in his time and thereafter. Many examples of scientific models based on data can be found, where data can be literally anything, ranging from measurements obtained through a microscope to observations obtained through a telescope.

Machine Learning is the field of research that focuses on the *automatic* building of *predictive* models from *collected data*, that can be evaluated using an *objective* performance measure. As the term 'automatic' indicates, it uses algorithms, aiming to keep the human expert out of the loop. Of course, there are also Machine Learning techniques that deliberately use domain experts' knowledge [157]. However, even those techniques are focused on automating the process, effectively taking over the workload and suggesting interesting patterns that would be too complex to distinguish otherwise. As the term 'predictive' indicates, the models should be capable of making predictions for yet unseen data. It is easy to perform well on the already seen data

Table 2.1: Random sample from the 'iris' dataset, as provided by [44].

sepal length	sepal width	petal length	petal width	class
5.1	3.7	1.5	0.4	setosa
5.1	3.3	1.7	0.5	setosa
6.2	2.8	4.8	1.8	virginica
6.9	3.1	5.4	2.1	virginica
6.1	3.0	4.6	1.4	versicolor
4.7	3.2	1.3	0.2	setosa
4.4	3.2	1.3	0.2	setosa
6.5	2.8	4.6	1.5	versicolor
6.8	3.0	5.5	2.1	virginica
6.3	3.4	5.6	2.4	virginica
5.1	3.5	1.4	0.3	setosa
6.3	3.3	6.0	2.5	virginica
5.0	3.2	1.2	0.2	setosa
5.1	3.8	1.9	0.4	setosa
5.7	4.4	1.5	0.4	setosa

sepal length	sepal width	petal length	petal width	class
5.5	4.2	1.4	0.2	setosa
6.4	3.2	4.5	1.5	versicolor
7.1	3.0	5.9	2.1	virginica
5.1	3.5	1.4	0.2	setosa
5.5	3.5	1.3	0.2	setosa
5.6	2.8	4.9	2.0	virginica
6.3	2.5	4.9	1.5	versicolor
5.8	4.0	1.2	0.2	setosa
6.0	2.2	5.0	1.5	virginica
5.0	3.4	1.5	0.2	setosa
4.8	3.0	1.4	0.3	setosa
6.8	3.2	5.9	2.3	virginica
5.8	2.6	4.0	1.2	versicolor
5.7	2.9	4.2	1.3	versicolor
7.4	2.8	6.1	1.9	virginica

(just memorize it); Machine Learning is about *generalizing* beyond this. Typically, the models will be evaluated based on predictions made for unseen data. An evaluation criterion can be the percentage of correct predictions, but in some cases more subtle measures are required. This is the objective performance measure.

The main concepts of Machine Learning are *data*, *tasks*, *models*, *algorithms* and *evaluation measures*. Chapter 2.2 gives examples of common types of data(sets), Chapter 2.3 overviews some common tasks that will recur in this thesis. In Chapter 2.4, we review the most common model types and mention the algorithms used to build them. Chapter 2.5 discusses the various ways of evaluating such models. Chapter 2.6 concludes with a discussion about *meta-learning*.

2.2 Data

In this chapter we will explore some common datasets that can be modelled using Machine Learning techniques.

2.2.1 Iris

Iris is a species of flowering plants, named after a Greek mythological goddess who rides the rainbows. Many iris flowers exist, e.g., iris unguicularis, iris latifolia and iris tectorum, to name a few. Some of these are easy to distinguish, others are harder. The English statistician and biologist Fisher created a dataset, containing measurable features about three types of iris flowers [44].

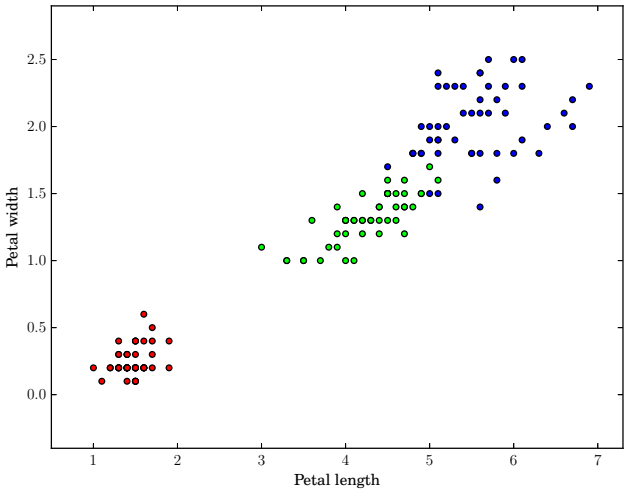


Figure 2.1: Scatter plot of the 'iris' dataset, as provided by [44].

Table 2.1 shows a random sample from the dataset, in the tabular form that data often is represented in. Each row in this dataset represents an iris flower. We call these rows *instances* or *observations*. Because of limited space, we only show a few. The dataset describes for each iris flower some perceptible *features*, such as sepal length, sepal width and petal length (all measured in centimetres). These we call the *attributes* of a dataset. Based on these features, flower experts can determine to which type of iris plants a certain instance belongs (the attribute 'class'). This dataset was created with the purpose of automatically modelling whether an iris flower belongs to the type setosa, versicolor or virginica. That makes this attribute somewhat special, hence we call it the *class* or the *target* attribute.

This is a *numeric* dataset. All attributes (except for the class) contain only numbers, making it easy to plot the data. This is done in Figure 2.1. The x-axis shows the petal length, the y-axis shows the petal width. The attributes sepal length and sepal width are omitted. Each dot represents an iris flower (150 in total), and the colour and shape of a dot shows to which class that flower belongs.

From this plot, we already see that it is quite easy to distinguish the setosa flowers from the others, just by looking at the petal size. We can draw a straight line that separates the setosa flowers from the others. This class is *linearly separable*. However, it is harder to distinguish the versicolor from the virginica. In general, the versicolor flowers have smaller petals than the virginica. But when presented with an uncategor-

ised iris flower with a petal length of 5 cm and a petal width of 1.5 cm, it becomes hard to classify it. One way of solving this problem would be also looking at the sepal length and sepal width, but we can only plot a limited number of variables.

Having only four numeric attributes and a total of 150 instances, from a computational point of view the iris dataset is considered to be an easy dataset to model. Despite the challenges for human experts, machine learning techniques are quite successful at modelling this dataset.

2.2.2 Mushroom

Mushrooms are popular for their edible, medicinal and psychoactive properties. As some mushrooms are very poisonous, knowing which mushrooms are edible and poisonous is quite important. Table 2.2 shows the 'mushroom' dataset. Each row in this dataset represents a mushroom. To keep this table simple, we have left some attributes out. The dataset describes for each mushroom some perceptible features, such as cap colour, odour and gill size. Based on these features, mushroom experts can determine whether a mushroom is edible or poisonous. This dataset was created with the purpose of modelling which properties makes a mushroom edible.

This is a *nominal* dataset. Each attribute can have certain values, for example, the attribute 'stalk surface above ring' can be either silky, smooth or sometimes even fibrous. There is no particular order in the values, making it hard to plot them meaningfully.

Many things can be observed by just looking at this sample. First, we notice some properties about distributions. Although mushrooms come in many colours, it appears that most of them have a red, grey or brown cap. Many mushrooms have either no odour or a foul odour, although some can smell spicy, musty or fishy. The gill colour is more equally distributed amongst the mushrooms, and can be chocolate, pink, buff or something completely different. Moreover, we can already see some correlations between mushroom features and the target. It appears that whenever a mushroom has a foul odour, it is not advisable to eat it. Conversely, from the data sample it seems that when a mushroom has no odour at all, it is edible. However, great care is still advised when eating an unknown mushroom without odour. As this data sample only covers part of the existing mushrooms, in reality many mushrooms have no odour and yet are poisonous. Sometimes, the person collecting the data makes a mistake. A poisonous mushroom can be recorded as edible, or the other way around. Erroneous recorded feature values or class values are called *noise*, which is a common problem data modellers should deal with.

The full dataset contains more than $8{,}000$ instances, and a total of 22 attributes (not including the target). This makes it hard to do a full analysis by hand. In compar-

Table 2.2: Random sample from the 'mushroom' dataset, as provided by [136].

cap colour	odour	gill spacing	gill size	gill colour	stalk surface above ring	stalk surface below ring	stalk colour below ring	spore print colour	population	habitat	...	class
gray	foul	close	broad	chocolate	silky	silky	pink	chocolate	solitary	grasses		poisonous
gray	foul	close	broad	gray	silky	silky	buff	chocolate	several	grasses		poisonous
buff	none	close	broad	white	smooth	smooth	white	white	clustered	waste		edible
gray	none	crowded	broad	pink	smooth	silky	white	white	scattered	grasses		edible
white	none	crowded	broad	gray	smooth	silky	white	white	numerous	grasses		edible
green	none	close	narrow	chocolate	smooth	fibrous	white	chocolate	solitary	woods		edible
gray	foul	close	broad	gray	silky	silky	buff	chocolate	several	paths		poisonous
brown	spicy	close	narrow	buff	silky	silky	pink	white	several	woods		poisonous
brown	fishy	close	narrow	buff	smooth	smooth	white	white	several	woods		poisonous
buff	foul	close	broad	pink	smooth	smooth	white	chocolate	several	grasses		poisonous
white	none	crowded	broad	pink	smooth	silky	white	white	scattered	grasses		edible
red	foul	close	narrow	buff	smooth	silky	white	white	several	paths		poisonous
white	creosote	close	narrow	brown	smooth	smooth	white	brown	several	woods		poisonous
yellow	almond	close	broad	brown	smooth	smooth	white	brown	scattered	grasses		edible
yellow	foul	close	broad	pink	silky	silky	brown	chocolate	several	paths		poisonous
yellow	anise	crowded	narrow	pink	smooth	smooth	white	purple	several	woods		edible
brown	none	crowded	broad	chocolate	fibrous	fibrous	white	brown	abundant	grasses		edible
gray	none	crowded	broad	chocolate	smooth	smooth	white	black	scattered	grasses		edible
red	foul	close	narrow	buff	silky	silky	pink	white	several	woods		poisonous
red	spicy	close	narrow	buff	smooth	smooth	pink	white	several	paths		poisonous
red	none	close	broad	brown	smooth	smooth	gray	black	solitary	woods		edible
red	foul	close	narrow	buff	silky	smooth	white	white	several	woods		poisonous
red	foul	close	narrow	buff	silky	silky	pink	white	several	leaves		poisonous
buff	none	close	broad	red	smooth	smooth	red	white	clustered	waste		edible
gray	none	crowded	broad	black	fibrous	smooth	white	black	abundant	grasses		edible
gray	foul	close	broad	chocolate	smooth	fibrous	white	chocolate	scattered	grasses		poisonous
yellow	anise	close	broad	white	smooth	smooth	white	black	scattered	meadows		edible
white	almond	close	broad	gray	smooth	smooth	white	brown	numerous	grasses		edible
cinnamon	musty	close	broad	yellow	silky	scaly	cinnamon	white	clustered	woods		poisonous
white	pungent	close	narrow	black	smooth	smooth	white	brown	scattered	urban		poisonous
brown	none	close	broad	brown	smooth	smooth	pink	brown	solitary	woods		edible
gray	foul	close	broad	gray	silky	silky	pink	chocolate	several	paths		poisonous
brown	spicy	close	narrow	buff	smooth	silky	white	white	several	woods		poisonous
gray	foul	close	broad	chocolate	silky	silky	buff	chocolate	solitary	woods		poisonous
yellow	foul	close	broad	gray	silky	silky	brown	chocolate	solitary	woods		poisonous

ison to other datasets, such as astronomical telescope data, this is only a very small dataset. We need more sophisticated models and techniques to analyse data sources of increasing size adequately.

2.3 Tasks

As the previous chapter argued, a common machine learning task is to model the available data, such that predictions for new, yet unseen, instances can be made. The task of modelling the 'mushroom' dataset is what we call *binary classification*, as there are only two classes to choose from (the data is *dichotomous*). In the case of the 'iris' dataset, there are already three possible classes. Whenever a dataset has more than two classes, we call the appropriate modelling task *multi-class classification*.

Many real world applications of machine learning are actually multi-class classification tasks, with a rather high number of classes. For example, face recognition programs typically get raw image data as input, and have to determine which person is displayed. In fact, in this case each person in the database is a unique class. Inconveniently, many Machine Learning models are only capable of solving the binary classification task. In order to overcome this limitation, a binary model can be turned into a multi-class model by a divide and conquer strategy called *One-versus-All*. Consider a dataset that consists of n classes, we build $n - 1$ models, such that each separates one class from all the other classes. Allwein et al. [5] propose a framework that consists of various strategies that make binary models useful for multi-class classification. Sometimes, we are not interested in modelling the whole dataset, but in finding an interesting subgroup of the data. The resulting model then describes the data only partially, the other data is out of the scope. This is called *subgroup discovery*.

The previous described datasets contained a nominal target. It is also possible to model datasets with a numeric target. This task type is known as *regression*.

Furthermore, the shown data did not have any concept of time or predefined order. We call it *stationary* data. This is different when modelling, for example, the stock price of a company. Observations obey a certain order, big changes in the economic landscape can have a huge impact on the performance of the model, therefore it needs to be updated constantly. This task type is called *data stream classification* or *data stream regression*. Chapter 5 covers this in detail.

Many other tasks exists, such as *clustering*, *pattern mining* and *association rule discovery*, but as these are out of the scope of this thesis we will not cover them here.

```
1 if (odour = foul) then class=poisonous (2160)
2 if (gill size = narrow) and (gill colour = buff) then class=poisonous (1152)
3 if (gill size = narrow) and (odour = pungent) then class=poisonous (256)
4 if (odour = creosote) then class=poisonous (192)
5 if (spore print colour = green) then class=poisonous (72)
6 if (stalk surface above ring = silky) and (gill spacing = close)
    then class=poisonous (68)
7 if (habitat = leaves) and (cap colour = white) then class=poisonous (8)
8 if (stalk colour above ring = yellow) then class=poisonous (8)
9 otherwise class=edible (4208)
```

Figure 2.2: Decision rule model of the 'mushroom' dataset. Between brackets is the number of instances that are captured by each rule.

2.4 Models

In this chapter, we will describe common Machine Learning methods. Domingos [38] describes a machine learning algorithm as a combination of *representation*, *evaluation* and *optimization*. The representation is the resulting model, the optimization is the procedure that creates that model and the evaluation is the way that model is evaluated (i.e., how good it fits the data). In this chapter we focus on commonly used model types and how they work, rather than on how these are constructed. For each model type, many optimization algorithms exist that are capable of creating them. For simplicity, these algorithms can be seen as black boxes that take data as input and produce a model.

2.4.1 Decision rules

Rule-based models are amongst the most intuitive types of models. Recall that by first inspecting the 'mushroom' dataset, we already came up with some *decision rules*: if the odour is foul then the mushroom is poisonous and if it has no odour then the mushroom is edible.

The rules depicted above are an example of the famous 'One Rule' model [69]. As it uses only one feature, the name is chosen accordingly. It highly simplifies the concept underlying the data, but is already quite accurate. The model gives great insight in what an important property of the problem is. Although the One Rule model adequately captures more than two-thirds of the data sample in this case, typically, a conjunction of many rules is needed to accurately describe the whole concept. Decision rules are commonly used for classification and subgroup discovery.

Some technicalities arise when using decision rule models for classification. For

example, what happens to an instance that is covered by multiple rules, or what happens to an instance that is not covered by any of the rules. Figure 2.2 shows an example of a *decision rule list*. Here, the rules have a certain order. If an instance is captured by multiple rules, the first rule that it complies to is used. Furthermore, it uses the concept of a default rule. All instances that are not captured by any rule are in this case classified as edible.

One of the nice properties about decision rules is that they are typically quite interpretable; human experts can verify and learn from them. Also, in most cases they are quite accurate in modelling the underlying concept.

Many rule induction algorithms exist, e.g., Repeated Incremental Pruning to Produce Error Reduction (RIPPER) [32], Fuzzy Unordered Rule Induction Algorithm (FURIA) [72] and Ripple-Down Rules (RIDOR) [33].

2.4.2 Decision trees

Decision trees are very similar to decision rules. Figure 2.3 shows an example of a decision tree, built upon the mushroom dataset. A decision tree consists of various *nodes*. Tree nodes are drawn round and contain an attribute name; the attribute that is being checked. Leaf nodes are drawn rectangular and contain a class value, in this case whether a mushroom is edible or not. The number between brackets denotes the number of instances that end up in that leaf node. For each observation, we start at the root node and traverse the tree in the direction that the test indicates. For example, for mushrooms that have an almond or anise odour, we traverse the left edge; these appear to be edible. For mushrooms that have no odour, we traverse the middle edge, and end up in the tree node where we will check the spore print colour. This process continues until we end in a leaf node.

When creating a tree, an important question is: which attribute should be used at some point as the splitting criterion. In the seminal paper by Quinlan, the attribute that obtains the highest *information gain* is used [115]. At each node in the tree, we can calculate the entropy $H(X)$ as follows:

$$H(X) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n} \tag{2.1}$$

where p is the number of instances at that tree node belonging to one class, X is the sub-sample of the dataset that is considered in the node (the full dataset at the root, but it is gradually getting smaller at lower levels) and n is the number of instances belonging to the other class. When all the instances belong to the same class, the entropy is 0. When exactly half of the instances belong to both classes, the entropy is 1. In all other cases, entropy is somewhere between the two.

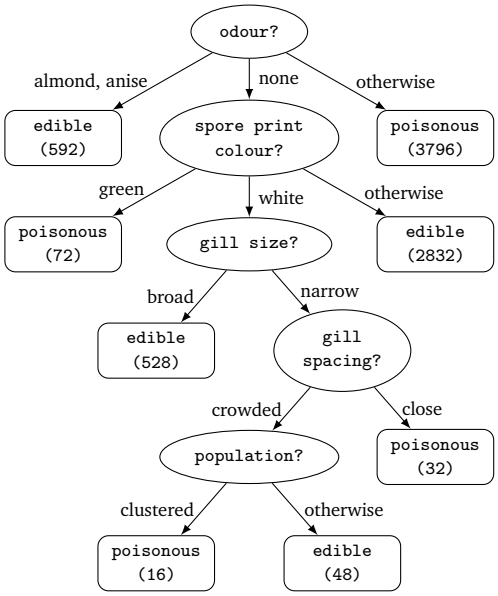


Figure 2.3: Decision tree model of the 'mushroom' dataset.

Suppose we consider an attribute a having v distinct values, and p_i instances of one class and p_n instances of the other class for each $1 \leq i \leq v$. We can determine the entropy after the split in the following way:

$$H'(X) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} \left\{ -\frac{p_i}{p_i + n_i} \log_2 \frac{p_i}{p_i + n_i} - \frac{n_i}{p_i + n_i} \log_2 \frac{n_i}{p_i + n_i} \right\} \tag{2.2}$$

The attribute that minimizes $H'(X)$ is chosen as splitting criterion. This process is repeated at each node, until a certain stopping criterion is met, e.g., all instances in a node belong to one class.

When using the described procedure, the decision tree is built in a greedy way; sometimes a splitting criterion is chosen that turns out to be suboptimal. As it is proven that building an optimal decision tree is NP-complete [76], we have to resort to such greedy procedures when working with large amounts of data.

As with the decision rule models, decision trees are easy to interpret, and therefore commonly used in practice. Many tree induction algorithms exist, e.g., Classification and Regression Tree (CART) [25], C4.5 [116] and Hoeffding Trees [39].

2.4.3 Probabilistic reasoning

Another way to model the mushroom dataset is by *probabilistic modelling*. In the data sample from Table 2.2, out of the 35 mushrooms, 20 mushrooms are poisonous and 15 mushrooms are edible. We say that the *prior probability* of a mushroom being poisonous $P(\text{class} = \text{poisonous}) = 20/35 \approx 0.57$ and the prior probability of a mushroom being edible $P(\text{class} = \text{edible}) = 15/35 \approx 0.43$. This observation gives us reason to believe that in general a mushroom is more likely to be poisonous than edible. However, for each individual mushroom, we can adapt this probability based on the observed features. For example, we observe that the mushroom has a smooth stalk surface above the ring. We immediately observe that the prior probability of a mushroom having a smooth stalk surface (above the ring) $P(\text{stalk surface} = \text{smooth}) = 21/35 = 0.6$. We want to know the probability of a mushroom being poisonous, given the fact that is has a smooth stalk surface. Conveniently, Bayes' theorem states that given a hypothesis and evidence that bears on that hypothesis:

$$P(\text{hypothesis}|\text{evidence}) = \frac{P(\text{evidence}|\text{hypothesis}) \cdot P(\text{hypothesis})}{P(\text{evidence})} \tag{2.3}$$

In this case, the evidence is a smooth stalk surface and the hypothesis is that the mushroom is poisonous. In fact, the hypothesis could as well be that the mushroom is edible, as this is the exact opposite of the previous hypothesis.

Of all the 20 observed poisonous mushrooms, 8 had a smooth stalk surface above the ring. The *likelihood* of a smooth stalk surface above the ring given that the mushroom is poisonous $P(\text{stalk surface} = \text{smooth}|\text{class} = \text{poisonous}) = 8/20 = 0.4$. Likewise, of the 15 edible mushrooms, 13 had a smooth stalk surface above the ring. Thus, the likelihood of a smooth stalk surface above the ring given that the mushroom is edible $P(\text{stalk surface} = \text{smooth}|\text{class} = \text{edible}) = 13/15 \approx 0.87$.

Plugging these numbers in Eq. 2.3 results in a probability of a mushroom being poisonous given that the stalk surface (above the ring) is smooth:

$$P(\text{class} = \text{poisonous}|\text{stalk surface} = \text{smooth}) \approx \frac{0.4 \cdot 0.57}{0.6} \approx 0.38 \tag{2.4}$$

Likewise, the probability of a mushroom being edible given that the stalk surface is smooth:

$$P(\text{class} = \text{edible}|\text{stalk surface} = \text{smooth}) \approx \frac{0.87 \cdot 0.43}{0.6} \approx 0.62 \tag{2.5}$$

This model scales trivially to multiple attributes; when classifying a mushroom, we should not only take into account the stalk surface (as we did in the previous example), but all other attributes that seem to be of influence. In the case of the mushrooms, we should also use odour, gill size, and probably even more. With probabilistic

modelling, the main challenge lies in identifying which attributes are important for classifying an instance.

There are many algorithms capable of building a probabilistic model. One of the most well-known is Naive Bayes, which is built upon the 'naive' assumption that all attributes are independent from each other, i.e., they do not interact. More sophisticated models can be built by means of Bayesian Networks. Creating an optimal Bayesian model is NP-complete [31, 34]. This implies that we have to rely on greedy or heuristic techniques when modelling large datasets.

2.4.4 Nearest Neighbour models

The main idea of *Nearest Neighbour models* is to identify and store some *key* instances from the dataset. Whenever presented with a new instance, find among the remembered instances the ones that are most similar to this new instance [35].

Two important issues arise. First, how do we define which are the key instances, and second, how do we determine which of the instances are most similar.

In the case of the 'iris' and 'mushroom' dataset, the issue of the key instances can be easily resolved. As both datasets contains only a small number of instances, all can be maintained in memory. However, with bigger datasets this becomes a serious issue that needs to be addressed, for example by sampling random instances.

One of the most commonly used measures of similarity is the Euclidean distance. Formally:

$$dist(x, x') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + \ldots + (x_k - x'_k)^2} \tag{2.6}$$

Here, x is the new instance, x' is one of the key instances, and x_i and x'_i denote the value of a given attribute, with $i = (1, 2, \ldots, k)$.

Having a similarity measure, it is easy to find the instances that have the lowest distance to x. Many other distance functions can be used as well. This technique is called *k-Nearest Neighbours*.

When creating a nearest neighbour model such as the one described, the data needs to be prepared with care. When dealing with attributes that are on different scales, the attribute with the biggest scale often dominates the Euclidean distance. Therefore, the data is often normalized to get the values of all attributes within the same interval. Moreover, all distance-based models suffer from a concept that is called the *curse of dimensionality* [50]. As high-dimensional datasets tend to be extremely sparse, the data points are often far away from each other. As such, instance-based models are also vulnerable to irrelevant attributes. It is recommended to use these models in combination with a feature selection technique [111].

2.4.5 Logistic Regression

Regression is a commonly used technique to model the relationship between numeric input variables and a numeric target. However, it can also be used for classification. In that case, *Logistic Regression* is used. Logistic Regression is a technique that models the probability of a newly observed instance belonging to a certain class. One interesting property is that it gives a degree of certainty that an instance belongs to a certain class. For example, consider the 'iris' dataset (Table 2.1). When classifying the third example, we would be much less certain that it actually belongs to the class virginica than when classifying, e.g., the fourth example. This is because the third example is much closer to the so-called *decision boundary*.

A typical regression model usually has the form:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_k x_k \tag{2.7}$$

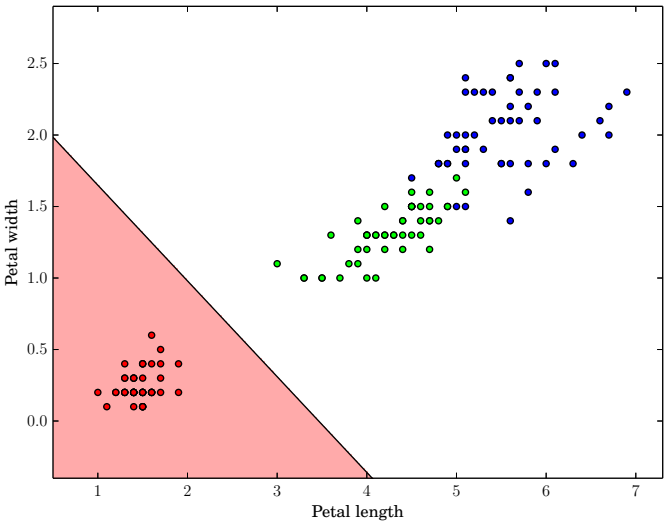
where x_i are the attribute values, w is a weight vector and y is the target.

By setting $y = 0$, we get a line or (hyper)plane that represents the model. In the case of normal regression, this line or (hyper)plane aims to fit the data points. In the case of Logistic Regression, this line or (hyper)plane separates the various classes, and is therefore called the *linear discriminant*. Figure 2.4 shows a Logistic Regression model built upon the petal width and petal length attributes of the 'iris' dataset.

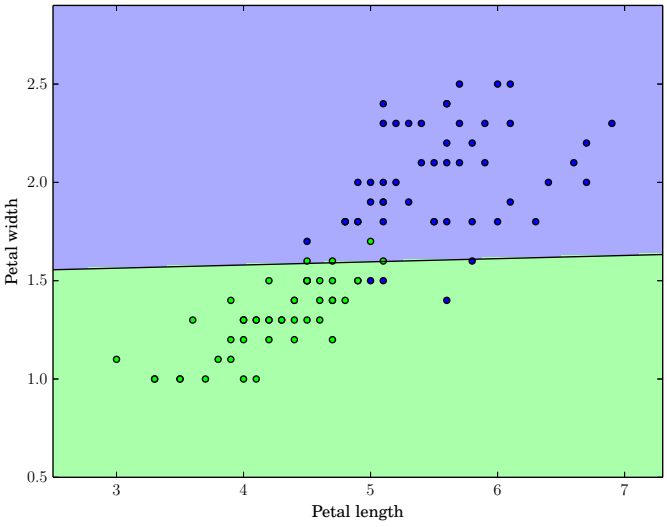
As the 'iris' dataset is a multi-class classification task, we can not separate them using one linear discriminant. Instead, we use the One-versus-All strategy to first separate instances of the class setosa from the others ($y = 3 + -1 \cdot \text{length} + -1 \cdot \text{width}$, see Figure 2.4a). If an instance does not belong to the setosa class, we can further establish whether it belongs to the versicolor class ($y = 4.3 + -0.55 \cdot \text{length} + -1 \cdot \text{width}$, see Figure 2.4b). If it does not belong to that class either, it will be classified as virginica.

The values of any new instance can be plugged into the respective formulas, resulting in a value ranging from $[-\infty, \infty]$. If the outcome is bigger than 0, it means it belongs to the specified class. If it is smaller than 0, it belongs to the other class (or set of classes). When the value is exactly 0, the model does not know to which class it belongs, and will have to guess. Conveniently, we can use the *logistic function* to map these back to the interval $[0, 1]$, in order to obtain proper probability estimates.

Logistic Regression requires a dataset to be linearly separable to perfectly fit the training data. For the 'iris' dataset this is not the case, as can be seen from Figure 2.4b. The resulting model therefore classifies some of the instances wrongly.



(a) Setosa vs. the rest



(b) Versicolor vs. Virginica

Figure 2.4: Logistic Regression model of the 'iris' dataset.

2.4.6 Support Vector Machines

Similar to Logistic Regression models, a *Support Vector Machine* (SVM) is a (hyper)plane-based model that separates the classes. Typically, there are many lines or (hyper)planes that do so. Support Vector Machines maximize the margin around the linear discriminant. The data points that lie closest to the decision surface are typically the most difficult to classify. These are called the *support vectors*. Based on these, the separating hyperplane is calculated. The distance between the support vectors and the separating hyperplane is maximized. This distance is called the *margin*.

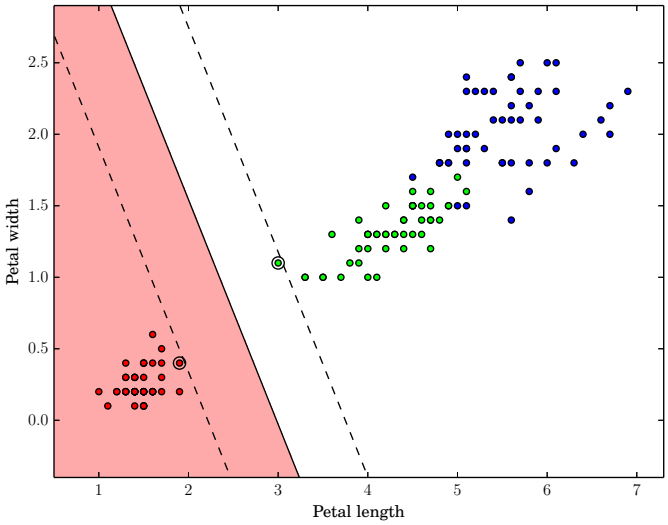
Figure 2.5 shows an example of a Support Vector Machine built upon the 'iris' dataset. Support Vector Machines are suitable for binary classification tasks. Therefore, in this case it uses the One-versus-All strategy to first separate the setosa class from the others (Figure 2.5a), after which it separates the Versicolor from the Virginica (Figure 2.5b). As with Logistic Regression, it requires linear separability. Otherwise, misclassifications already occur in the training set. In Figure 2.5b we see 5 misclassified instances, and even 8 instances that fall within the separation margin.

Often it occurs that the data is not linearly separable in the original representation, but is linearly separable when you represent it differently. Support Vectors Machines are often used in combination with a *kernel*. Kernels map the data in a higher dimension, effectively resulting in more attributes. For the task of finding the separating hyperplane and classifying new instances, we only need the dot product of the original features, saving us from additional memory usage. Popular kernels for classification purposes are the *Radial Basis Function kernel* (RBF), *Polynomial kernel* and *Sigmoid kernel*. Mapping the data to a higher input space should be done with great care, due to the curse of dimensionality. Increasing the number of variables, exponentially increases the number of possible solutions, yielding many sub-optimal solutions [98].

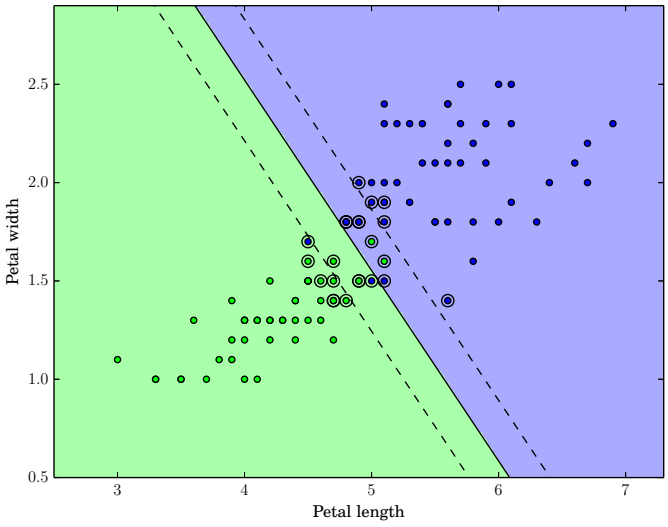
A Support Vector Machine model is based on a small amount of data points, making it a rather stable model. Adding or removing data points does not affect the model, unless that data point is in fact one of the support vectors. Support Vector Machines are typically not affected by local minima. The name of Support Vector Machines can be misleading. It is not a machine, it is a model. When using the right kernel, Support Vector Machines are very powerful models, building upon a solid theoretical foundation.

2.4.7 Neural Networks

An alternative modelling approach is based on the human brain. The human brain is a collection of billions of *neurons*, that are ordered in a certain graph structure. Each neuron has certain inputs and outputs. A neuron outputs a signal if the combined

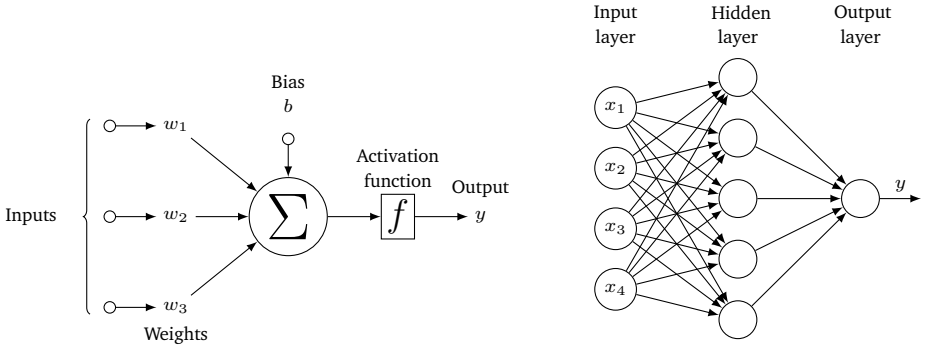


(a) Setosa vs. the rest



(b) Versicolor vs. Virginica

Figure 2.5: Support Vector Machine model built upon the 'iris' dataset. The solid line is the separating hyperplane, the circled data points are the support vectors. The striped line indicates the margin of the separated hyperplane.



(a) A single neuron

(b) Structure of a network with 1 hidden layer that contains 5 neurons

Figure 2.6: Example of a neural network.

input signals exceed a certain threshold. Inspired by the success of the human brain, machine learners imitated this paradigm by means of *Neural Networks*.

Figure 2.6 shows an example of a neural network. Figure 2.6a shows an abstraction of a neuron. This neuron has 3 preceding input neurons, which are all associated with a certain weight. Each neuron has a certain threshold value, here displayed with b. The output value y of each neuron is determined by adding all the weighted inputs together; the result is put in the *activation function*. The activation function f has two purposes. First, it determines whether the threshold value is exceeded or not, and second, it maps the resulting output of the neuron in the desired domain, typically $[0, 1]$.

This determines the outcome of the neuron. Typically, neural networks are built as acyclic graphs, consisting of distinguishable layers. These networks are called *feed forward networks*. Figure 2.6b shows an example of the network structure. It contains an input layer, one hidden layer and an output layer. Typically, the feature values of a given instance are fed into the input layer; all the other layers are the neurons as described before.

Indeed, a network that contains no hidden layers (and thus only has one real neuron, i.e., the output node) has a model corresponding to Linear Regression: each input variable is associated with a certain weight, and the bias of the neuron corresponds to w_0 of the Logistic Regression model. These neural networks are called *perceptrons*.

The real power of neural networks lies in the hidden layers. Effectively, these are capable of simulating derivatives of feature combinations, that are not manifested in

the original input space, giving it a tremendous expressive power. Great care should be taken with noisy data. Due to the high expressive power, it can easily learn a wrong concept from the noise. The biggest challenges in neural networks is finding a good network structure and the right weights for each neuron. Various algorithms are proposed for this, most commonly back-propagation. Training a 3-node neural network is already NP-complete [17].

2.5 Evaluation

For a created model, an important issue that needs to be addressed is, how well does it perform. In order to do so, we need an *evaluation measure*, which quantitatively defines performance. We could give the model some instances of which we already know the correct class label, and see in what percentage of cases it classifies them correctly. These instances are called the *test set*. This measure is often called *predictive accuracy*, and one of the most natural ways of measuring the performance. Predictive accuracy is a measure that is calculated over the whole test set. It is important to measure the *generalization* capabilities of a classifier, i.e., how good the predictions are for instances that it has not been trained on.

Sometimes, there is an asymmetry in the importance of misclassifications. For example, it is worse to classify a poisonous mushroom as edible than the other way around. Likewise, banks determine whether a person is eligible for receiving credit based on social-economic attributes such as employment status, age and credit history. From their perspective, it is worse to give credit to someone that is not creditworthy, than the other way around. The 'German credit' dataset contains past records of credit applications; the goal is to model when someone is eligible for receiving credit. For this purpose, we could use class-specific evaluation measures, that only address the performance on a given class. The *sensitivity* or *true positive rate* (TPR) is the proportion of instances that belong to a certain class that are correctly classified as such. The *specificity* or *true negative rate* (TNR) is the proportion of instances that do belong to a certain class and that are correctly classified as such. Typically, there is a trade-off between the sensitivity and specificity. For example, when a classifier always predicts a given class, it will have a perfect true positive rate, but a true negative rate of 0, and vice versa. A similar trade-off between the true positive rate and the false positive rate is depicted in the *receiver operating characteristic curve* (ROC-curve), a common graphical evaluation measure.

Figure 2.7 shows the ROC curve of three classifiers that model whether someone is credible or not. Banking companies that want to model future credit applications can choose between Naive Bayes and the Nearest Neighbour approach. Although Naive

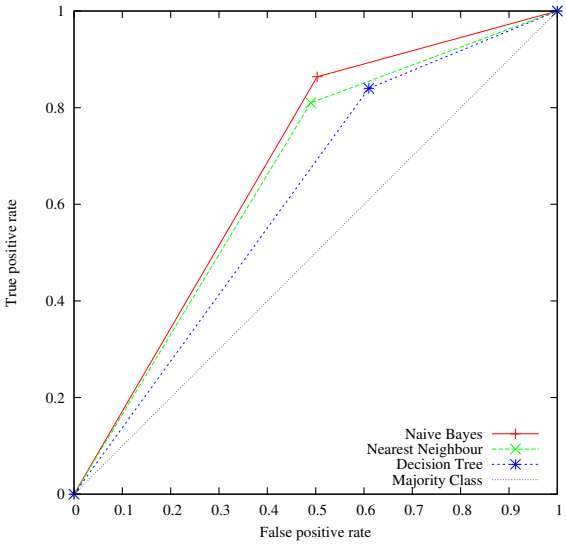


Figure 2.7: ROC curves of three classifiers on the 'German credit' dataset. The grey line shows a virtual classifier that always predicts the same class.

Bayes has a higher accuracy (this can not be deduced from the image), the nearest neighbour method might be selected based on the ROC curve; not credit-worthy persons will be classified as credit-worthy less often. The decision tree model is *dominated* by the Naive Bayes model, it obtains a lower true positive rate and a higher false positive rate.

It is important to always evaluate the performance of a model based on data that the model is not trained on. It is easy to see that by memorization it is easy to perform well on the full training set. What we are interested in is generalization beyond this. For this reason, typically a *holdout set* is used: the model is trained based on a percentage of the data, and then evaluated on the other part of the data.

There are two drawbacks to this approach. First, the model is not tested on all instances. When the instances that are hard to classify all end up in either the training set or the test set, this will give respectively an optimistic or pessimistic assessment of the model. Second, as the model is only evaluated once, it does not give a stable assessment of performance.

In order to overcome these problems, n-fold *cross-validation* is usually preferred. The data set is split in split in n equal subsets, and n different models are trained using $n-1$ of these subsets and tested on the remaining one. This way, it tests the model using each instance exactly once. Cross-validation is widely considered a reliable way

of evaluating a model on a single dataset.

However, from a Machine Learning point of view, it is often desirable to make more general conclusions, based on evaluations over many datasets. To this end, statistical tests are used. This allows making statements about the performance of classifiers over multiple datasets or cross-validation runs. Demšar [36] reviews various statistical tests appropriate for assessing the quality of Machine Learning models and algorithms.

2.6 Discussion

This chapter showed some examples of common Machine Learning data, tasks and models. We have seen various models, all leading to a different representation and therefore a different expressive power and bias towards certain kinds of data. Having more expressive power does not guarantee better classification performance. Sometimes highly complex models over-fit on an irrelevant concept and perform unexpectedly mediocre.

One aspect that is nearly not covered are algorithms. These are sets of rules to be followed in order to create the models. For each model type, there are many algorithms that are able to create such models. Each of these algorithms in turn contains various parameters to be determined by the user; these parameters enable small or big nuances in the resulting model. This makes it hard for anyone interested in modelling a dataset to select the appropriate model, algorithm and parameter settings. One question that arises from this is, can we learn from previous machine learning experiments what kind of algorithm should be used to model a new dataset; this is called *meta-learning*. As machine learning aims to model a certain dataset, meta-learning aims to model what kind of data should be modelled by what kind of technique. In the next chapter, we will review common techniques of doing so.