CHAPTER 12

Machine learning experiments

MACHING LEARNING IS a practical subject as much as a computational one. While we may be able to prove that a particular learning algorithm converges to the theoretically optimal model under certain assumptions, we need actual data to investigate, e.g., the extent to which those assumptions are actually satisfied in the domain under consideration, or whether convergence happens quickly enough to be of practical use. We thus evaluate or run particular models or learning algorithms on one or more data sets, obtain a number of measurements and use these to answer particular questions we might be interested in. This broadly characterises what is known as machine learning *experiments*.

In the natural sciences, an experiment can be seen as a question to nature about a scientific theory. For example, Arthur Eddington's famous 1919 experiment to verify Einstein's theory of general relativity asked the question: Are rays of light bent by gravitational fields produced by large celestial objects such as the Sun? To answer this question, the perceived position of stars was recorded under several conditions including a total solar eclipse. Eddington was able to show that these measurements indeed differed to an extent unexplained by Newtonian physics but consistent with general relativity.

While you don't have to travel to the island of Príncipe to perform machine learning experiments, they bear some similarity to experiments in physics in that *machine*

https://doi.org/10.1017/CBO9780511973000.014 Published online by Cambridge University Press

learning experiments pose questions about models that we try to answer by means of measurements on data. The following are common examples of the types of question we are interested in:

☞ How does model m perform on data from domain $\mathscr{D}$?

☞ Which of these models has the best performance on data from domain $\mathscr{D}$?

☞ How do models produced by learning algorithm $\mathscr{A}$ perform on data from domain $\mathscr{D}$?

☞ Which of these learning algorithms gives the best model on data from domain $\mathscr{D}$?

Assuming we have access to data from domain $\mathscr{D}$, we perform measurements on our models using this data in order to answer these questions.[1] There is a large statistical literature about the technicalities of data experiments, and it is all too easy to mistake the forest for the trees. What I mean is that there is a certain tendency in the machine learning literature to approach experimentation in a formulaic manner, recording a fixed set of measurements and significance tests (the trees) with scant consideration for the question we are aiming to answer (the forest). My aim in this short chapter is not to get buried in technicalities, but rather to give you some appreciation of the importance of choosing measurements that are appropriate for your particular experimental objective (Section 12.1). In Sections 12.2 and 12.3 we take a closer look at how to measure and interpret them.

12.1 What to measure

A good starting point for our measurements are the ☞*evaluation measures* in Table 2.3 on p.57. However, measurements don't have to be scalars: a ROC or coverage curve also counts as a measurement in this context. The appropriateness of any of these for our purposes depends on how we define performance in relation to the question the experiment is designed to answer: let's call it our *experimental objective*. It is important not to confuse performance measures and experimental objectives: the former is something we can measure, while the latter is what we are really interested in. There is often a discrepancy between the two. For example, in psychology our experimental objective may be to quantify a person's intelligence level, and our chosen measurement may be the IQ score achieved on a standardised test – while the IQ score may correlate with intelligence level, it is clear they are not the same thing.

[1] It is much harder to answer questions about a new domain given measurements on different domains, although that is what we are ultimately interested in.

In machine learning the situation is usually more concrete, and our experimental objective – accuracy, say – is something we can measure in principle, or at least estimate (since we're generally interested in accuracy on unseen data). However, there may be unknown factors we have to account for. For example, the model may need to operate in different *operating contexts* with different class distributions. In such a case we can treat accuracy on future data as a random variable and take its expectation, assuming some probability distribution over the proportion of positives *pos*. Since $acc = pos \cdot tpr + (1 - pos) \cdot tnr$, and assuming we can measure true positive and negative rates independently of the class distribution, we have (assuming a uniform distribution over *pos*)

$$\begin{aligned}\mathbb{E}[acc] &= \mathbb{E}\left[pos \cdot tpr + (1 - pos) \cdot tnr\right] = \mathbb{E}\left[pos\right] tpr + \mathbb{E}\left[1 - pos\right] tnr \\ &= tpr/2 + tnr/2 = avg\text{-}rec\end{aligned}$$

In other words, even though estimating accuracy in future contexts is our experimental objective, the fact that we are anticipating the widest possible range of class distributions suggests that the evaluation measure we should use on our test data is not accuracy, but average recall.

Example 12.1 (Expected accuracy for unknown class distributions). Imagine your classifier achieves the following result on a test data set:

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**60**	**20**	80
Actual ⊖	**0**	**20**	20
	60	40	100

This gives $tpr = 0.75$, $tnr = 1.00$ and $acc = 0.80$. However, this is conditioned on having four times as many positives as negatives. If we take the expectation over *pos* uniformly sampled from the unit interval, expected accuracy increases to $(tpr + tnr)/2 = 0.88 = avg\text{-}rec$. This is higher because the test data underemphasises the classifier's good performance on the negatives.

Suppose you have a second classifier achieving the following result on the test set:

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**75**	**5**	80
Actual ⊖	**10**	**10**	20
	85	15	100

This gives *tpr* = 0.94, *tnr* = 0.50, *acc* = 0.85 and *avg-rec* = 0.72. These experimental results tell you that the second model is better if the class distribution in the test set is representative, but the first model should be chosen if we have no prior information about the class distribution in the operating context.

As the example demonstrates, if you choose accuracy as your evaluation measure, you are making an implicit assumption that the class distribution in the test set is representative for the operating context in which the model is going to be deployed. Furthermore, if all you recorded in your experiments is accuracy, you will not be able to switch to average recall later if you realise that you need to incorporate varying class distributions. It is therefore good practice to record sufficient information to be able to reproduce the contingency table if needed. A sufficient set of measurements would be true positive rate, true negative rate (or false positive rate), the class distribution and the size of the test set.

As a second example of how your choice of evaluation measures can carry implicit assumptions we consider the case of precision and recall as often reported in the information retrieval literature.

Example 12.2 (Precision and recall as evaluation measures). In the second contingency table in Example 12.1 we have precision *prec* = 75/85 = 0.88 and recall *rec* = 75/80 = 0.94 (remember that recall and true positive rate are different names for the same measure). The F-measure is the harmonic mean of precision and recall (see Background 10.1), which is 0.91.

Now consider the following contingency table:

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**75**	**5**	80
Actual ⊖	**10**	**910**	920
	85	915	1000

We have a much higher number of true negatives and therefore a much higher true negative rate and accuracy (both rounded to 0.99). On the other hand, true positive rate/recall, precision and F-measure stay exactly the same.

This example demonstrates that *the combination of precision and recall, and therefore the F-measure, is insensitive to the number of true negatives*. This is not a deficiency of the F-measure: quite the contrary, it is very useful in domains where negatives abound, and it would therefore be very easy to achieve high accuracy by always predicting negative. Examples of such domains include search and query engines (most search items are not answers to most queries) and link prediction in networks (most pairs of nodes are not linked). The point is rather to emphasise that if you choose F-measure as your evaluation measure, you are making an implicit assertion that true negatives are not relevant for your operating context.

Finally, I would like to draw attention to a much-neglected evaluation measure that has, nevertheless, practical significance. This is the *predicted positive rate* which is the number of positive predictions (the sum of the left column in the contingency table) in proportion to the number of instances:

$$ppr = \frac{TP + FP}{Pos + Neg} = pos \cdot tpr + (1 - pos) \cdot fpr$$

While the predicted positive rate doesn't tell us much about the classification performance of the classifier, it does tell us what the classifier estimates the class distribution to be. It is also something that is normally under control of a ranker or scoring classifier if the entire test set is given: e.g., a predicted positive rate of 1/2 is simply achieved by setting the threshold such that the ranking splits into equal parts. This suggests a connection between classification accuracy and ranking accuracy: for example, it can be shown that if we split a ranking of n instances at one of the $n+1$ possible split points chosen uniformly at random, the expected accuracy is equal to

$$\mathbb{E}[acc] = \frac{n}{n+1} \frac{2\text{AUC} - 1}{4} + 1/2$$

Example 12.3 (Expected accuracy and AUC). Suppose a ranker obtains the following ranking on a small test set: $\oplus\oplus\ominus\ominus\oplus\ominus$. This corresponds to two ranking errors out of a possible nine, so has AUC $= 7/9$. There are seven potential split points, corresponding to predicted positive rates of (from left to right) $0, 1/6, \ldots, 5/6, 1$ and corresponding accuracies $3/6, 4/6, 5/6, 4/6, 3/6, 4/6, 3/6$. The expected accuracy over all possible split points is $(3+4+5+4+3+4+3)/(6 \cdot 7) = 26/42$. On the other hand, $(2\text{AUC} - 1)/4 = 5/36$ and so $\frac{n}{n+1}(2\text{AUC} - 1)/4 + 1/2 = 5/42 + 1/2 = 26/42$.

This discussion is intended to highlight two things. First, if a model is a good ranker but its probability estimates are not well-calibrated, it might be a good idea to set the decision threshold such that it achieves a particular predicted positive rate (e.g., $ppr = pos$) rather than invoking the MAP decision rule. Secondly, in such situations AUC is a good evaluation measure because it is linearly related to expected accuracy in that scenario.

In summary, your choice of evaluation measures should reflect the assumptions you are making about your experimental objective as well as possible contexts in which your models operate. We have looked at the following cases:

☞ Accuracy is a good evaluation measure if the class distribution in your test set is representative for the operating context.

☞ Average recall is the evaluation measure of choice if all class distributions are equally likely.

☞ Precision and recall shift the focus from classification accuracy to a performance analysis ignoring the true negatives.

☞ Predicted positive rate and AUC are relevant measures in a ranking context.

In the next section we consider how to estimate these evaluation measures from data.

12.2 How to measure it

The evaluation measures we discussed in the previous section are all calculated from a contingency table. The question of 'how to measure it' thus seems to have a very straightforward answer: construct the contingency table from a test set and perform the relevant calculations. However, two issues demand our attention: (*i*) which data to base our measurements on, and (*ii*) how to assess the inevitable uncertainty associated with every measurement. In this section we are concerned with the first issue; the second issue will be discussed in the next section.

When we measure something – say, a person's height – several times, we expect some variation to occur from one measurement to the next. This is inherent to the measurement process: e.g., you may be stretching the tape measure a bit less for the second measurement, or you may be reading it from a slightly different angle.[2] This variation can be modelled by treating our measurement as a random variable characterised by its mean – the value we are trying to measure – and variance σ^2, both of which are unknown but can be estimated. It follows from this model that if you measure a person's height many times, the sample variance in the measured values converges to σ^2. A standard trick is to average k measurements, as this gets the variance in your estimate down to σ^2/k. What this means is that, if you repeated this averaging for many sets of k measurements, the averages have a sample variance σ^2/k. Crucially,

[2]Some variation is also due to the fact that somebody tends to be taller fresh out of bed in the morning than at the end of a day of sitting and standing up, but I ignore that here and assume an unambiguous true value that we are trying to measure.

this assumes that your measurements are independent: if you are introducing a systematic error by using a faulty tape measure, averaging won't help!

Now suppose you are measuring a classifier's accuracy (or its true positive rate, or the predicted positive rate, or any other evaluation measure discussed earlier), rather than a person's height. The natural model here is that each test instance represents a Bernoulli trial with success probability a, the true but unknown accuracy of the classifier. We estimate a by counting the number of correctly classified test instances A and setting $\hat{a} = A/n$; notice that A has a binomial distribution. The variance of a single Bernoulli trial is $a(1-a)$; averaged over n test instances it is $a(1-a)/n$, assuming the test instances are chosen independently. We can estimate the variance by plugging in our estimate for a: this will help us to assess the uncertainty in $\hat{a}$, as we shall see in the next section. Under certain conditions we can improve our estimate by averaging k independent estimates $\hat{a}_i$ and take their sample variance $\frac{1}{k-1}\sum_{i=1}^{k}(\hat{a}_i - \overline{a})^2$ instead, with $\overline{a} = \frac{1}{k}\sum_{i=1}^{k}\hat{a}_i$ the sample mean.[3]

How do we obtain k independent estimates of a? If we have plenty of data, we can sample k independent test sets of size n and estimate a on each of them. Notice that if we are evaluating a learning algorithm rather than a given model we need to set aside training data which needs to be separate from the test data. If we don't have a lot of data, the following *cross-validation* procedure is often applied: randomly partition the data in k parts or 'folds', set one fold aside for testing, train a model on the remaining $k-1$ folds and evaluate it on the test fold. This process is repeated k times until each fold has been used for testing once. This may seem curious at first since we are evaluating k models rather than a single one, but this makes sense if we are evaluating a learning algorithm (whose output is a model, so we want to average over models) rather than a single model (whose outputs are instance labels, so we want to average over those). By averaging over training sets we get a sense of the variance of the learning algorithm (i.e., its dependence on variations in the training data), although it should be noted that the training sets in cross-validation have considerable overlap and are clearly not independent. Once we are satisfied with the performance of our learning algorithm, we can run it over the entire data set to obtain a single model.

Cross-validation is conventionally applied with $k = 10$, although this is somewhat arbitrary. A rule of thumb is that individual folds should contain at least 30 instances, as this allows us to approximate the binomial distribution of the number of correctly classified instances in a fold by a normal distribution. So if we have fewer than 300 instances we need to adjust k accordingly. Alternatively, we can set $k = n$ and train on all but one test instance, repeated n times: this is known as *leave-one-out* cross-validation (or the jackknife in statistics). This means that in each single-instance 'fold'

[3]Notice that we divide by $k-1$ rather than k in the expression for the sample variance, to account for the uncertainty in our estimate of the sample mean.

our accuracy estimate is 0 or 1, but by averaging n of those we get an approximately normal distribution by the central limit theorem. If we expect the learning algorithm to be sensitive to the class distribution we should apply *stratified cross-validation*: this aims at achieving roughly the same class distribution in each fold. Cross-validation runs can be repeated for different random partitions into folds and the results averaged again to further reduce variance in our estimates: this is referred to as, e.g., 10 times 10-fold cross-validation. It should be kept in mind that this leads increasingly to independence assumptions being violated – if we take this too far our accuracy estimate will overfit the given data and not be representative for new data.

Example 12.4 (Cross-validation). The following table gives a possible result of evaluating three learning algorithms on a data set with 10-fold cross-validation:

Fold	*Naive Bayes*	*Decision tree*	*Nearest neighbour*
1	0.6809	0.7524	0.7164
2	0.7017	0.8964	0.8883
3	0.7012	0.6803	0.8410
4	0.6913	0.9102	0.6825
5	0.6333	0.7758	0.7599
6	0.6415	0.8154	0.8479
7	0.7216	0.6224	0.7012
8	0.7214	0.7585	0.4959
9	0.6578	0.9380	0.9279
10	0.7865	0.7524	0.7455
avg	0.6937	0.7902	0.7606
stdev	0.0448	0.1014	0.1248

The last two lines give the average and standard deviation over all ten folds. We can see that nearest neighbour has the highest standard deviation. Clearly the decision tree achieves the best result, but should we completely discard nearest neighbour?

Cross-validation can also be applied to ROC curves obtained from a scoring classifier. This is because every instance participates in exactly one test fold and receives a score from the corresponding model. We can therefore simply merge all test folds which produces a single ranking.

12.3 How to interpret it

Once we have estimates of a relevant evaluation measure for our models or learning algorithms we can use them to select the best one. The fundamental issue here is how to deal with the inherent uncertainty in these estimates. We will discuss two key concepts: confidence intervals and significance tests. An understanding of these concepts is necessary if you want to appreciate current practice in interpreting results from machine learning experiments – however, it is good to realise that current practice is coming under increasing scrutiny. It should also be noted that the methods described here represent only a tiny fraction of the vast spectrum of possibilities.

Suppose our estimate $\hat{a}$ follows a normal distribution around the true mean a with standard deviation σ. Assuming for the moment that we know these parameters, we can calculate for any interval the likelihood of the estimate falling in the interval, by calculating the area under the normal density function in that interval. For example, the likelihood of obtaining an estimate within ± 1 standard deviation around the mean is 68%. Thus, if we take 100 estimates from independent test sets, we expect 68 of them to be within one standard deviation on either side of the mean – or equivalently, we expect the true mean to fall within one standard deviation on either side of the estimate in 68 cases. This is called the 68% *confidence interval* of the estimate. For two standard deviations the confidence level is 95% – these values can be looked up in probability tables or calculated using statistical packages such as Matlab or R. Notice that confidence intervals for normally distributed estimates are symmetric because the normal distribution is symmetric, but this is not generally the case: e.g., the binomial distribution is asymmetric (except for $p = 1/2$). Notice also that, in case of symmetry, we can easily change the interval into a one-sided interval: for example, we expect the mean to be more than one standard deviation *above* the estimate in 16 cases out of 100, which gives a one-sided 84% confidence interval from minus infinity to the mean plus one standard deviation.

More generally, in order to construct confidence intervals we need to know (*i*) the sampling distribution of the estimates, and (*ii*) the parameters of that distribution. We saw previously that accuracy estimated from a single test set with n instances follows a scaled binomial distribution with variance $\hat{a}(1-\hat{a})/n$. This would lead to asymmetric confidence intervals, but the skew in the binomial distribution is only really noticeable if $na(1-a) < 5$: if that is not the case the normal distribution is a good approximation for the binomial one. So we use the binomial expression for the variance and use the normal distribution to construct the confidence intervals.

Example 12.5 (Confidence interval). Suppose 80 out of 100 test instances are correctly classified. We have $\hat{a} = 0.80$ with an estimated variance of $\hat{a}(1-\hat{a})/n = 0.0016$ or a standard deviation of $\sqrt{\hat{a}(1-\hat{a})/n} = 0.04$. Notice $n\hat{a}(1-\hat{a}) = 16 \geq 5$ so the 68% confidence interval is estimated as $[0.76, 0.84]$ in accordance with the normal distribution, and the 95% interval is $[0.72, 0.88]$.

If we reduce the size of our test sample to 50 and find that 40 test instances are correctly classified, then the standard deviation increases to 0.06 and the 95% confidence interval widens to $[0.68, 0.92]$. If the test sample drops to less than 30 instances we would need to construct an asymmetric confidence interval using tables for the binomial distribution.

Notice that *confidence intervals are statements about estimates rather than statements about the true value of the evaluation measure*. The statement 'assuming the true accuracy a is 0.80, the probability that a measurement m falls in the interval $[0.72, 0.88]$ is 0.95' is correct, but we cannot reverse this to say 'assuming a measurement $m = 0.80$, the probability that the true accuracy falls in the interval $[0.72, 0.88]$ is 0.95'. To infer $P(a \in [0.72, 0.88] | m = 0.80)$ from $P(m \in [0.72, 0.88] | a = 0.80)$ we must somehow invoke Bayes' rule, but this requires meaningful prior distributions over both true accuracies and measurements, which we don't generally have.

We can, however, use similar reasoning to test a particular *null hypothesis* we have about a. For example, suppose our null hypothesis is that the true accuracy is 0.5 and that the standard deviation derived from the binomial distribution is therefore $\sqrt{0.5(1-0.5)/100} = 0.05$. Given our estimate of 0.80, we then calculate the *p-value*, which is the probability of obtaining a measurement of 0.80 or higher given the null hypothesis. The p-value is then compared with a pre-defined significance level, say $\alpha = 0.05$: this corresponds to a confidence of 95%. The null hypothesis is rejected if the p-value is smaller than α; in our case this applies since $p = 1.9732 \cdot 10^{-9}$.

This idea of *significance testing* can be extended to learning algorithms evaluated in cross-validation. For a pair of algorithms we calculate the difference in accuracy on each fold. The difference between two normally distributed variables is also normally distributed. Our null hypothesis is that the true difference is 0, so that any differences in performance are attributed to chance. We calculate a p-value using the normal distribution, and reject the null hypothesis if the p-value is below our significance level α. The one complication is that we don't have access to the true standard deviation in the differences, which therefore needs to be estimated. This introduces additional uncertainty into the process, which means that the sampling distribution is bell-shaped like the normal distribution but slightly more heavy-tailed. This distribution is referred to as Student's t-distribution or simply the *t-distribution*.[4] The extent to which the t-distribution is more heavy-tailed than the normal distribution is regulated by the number of *degrees of freedom*: in our case this is equal to 1 less than the number of folds (since the final fold is completely determined by the other ones). The whole procedure is known as the *paired t-test*.

Example 12.6 (Paired t-test). The following table demonstrates the calculation of a paired t-test on the results in Example 12.4. The numbers show pairwise differences in each fold. The null hypothesis in each case is that the differences come from a normal distribution with mean 0 and unknown standard deviation.

Fold	*NB–DT*	*NB–NN*	*DT–NN*
1	-0.0715	-0.0355	0.0361
2	-0.1947	-0.1866	0.0081
3	0.0209	-0.1398	-0.1607
4	-0.2189	0.0088	0.2277
5	-0.1424	-0.1265	0.0159
6	-0.1739	-0.2065	-0.0325
7	0.0992	0.0204	-0.0788
8	-0.0371	0.2255	0.2626
9	-0.2802	-0.2700	0.0102
10	0.0341	0.0410	0.0069
avg	-0.0965	-0.0669	0.0295
stdev	0.1246	0.1473	0.1278
p-value	**0.0369**	**0.1848**	**0.4833**

The p-value in the last line of the table is calculated by means of the t-distribution with $k-1=9$ degrees of freedom, and only the difference between the naive Bayes and decision tree algorithms is found significant at the $\alpha = 0.05$ level.

[4] It was published by William Sealy Gosset in 1908 under the pseudonym 'Student' because his employer, the Guinness brewery in Dublin, did not want the competition to know that they were using statistics.

Interpretation of results over multiple data sets

The t-test can be applied for comparing two learning algorithms over a single data set, typically using results obtained in cross-validation. It is not appropriate for multiple data sets because performance measures cannot be compared across data sets (they are not 'commensurate'). In order to compare two learning algorithms over multiple data sets we need to use a test specifically designed for that purpose such as *Wilcoxon's signed-rank test*. The idea is to rank the performance differences in absolute value, from smallest (rank 1) to largest (rank n). We then calculate the sum of ranks for positive and negative differences separately, and take the smaller of these sums as our test statistic. For a large number of data sets (at least 25) this statistic can be converted to one which is approximately normally distributed, but for smaller numbers the *critical value* (the value of the statistic at which the p-value equals α) can be found in a statistical table.

Example 12.7 (Wilcoxon's signed-rank test). We use the performance differences between naive Bayes and decision tree as in the previous example, but now assume for the sake of argument that they come from 10 different data sets.

Data set	*NB–DT*	*Rank*
1	-0.0715	**4**
2	-0.1947	**8**
3	0.0209	**1**
4	-0.2189	**9**
5	-0.1424	**6**
6	-0.1739	**7**
7	0.0992	**5**
8	-0.0371	**3**
9	-0.2802	**10**
10	0.0341	**2**

The sum of ranks for positive differences is $1+5+2=8$ and for negative differences $4+8+9+6+7+3+10=47$. The critical value for 10 data sets at the $\alpha = 0.05$ level is 8, which means that if the smallest of the two sums of ranks is less than or equal to 8 the null hypothesis that the ranks are distributed the same for positive and negative differences can be rejected. This applies in this case, so we conclude that the performance difference between naive Bayes and decision trees is significant according to Wilcoxon's signed-rank test (as it was for the paired t-test in Example 12.6).

The Wilcoxon signed-rank test assumes that larger performance differences are better than smaller ones, but otherwise makes no assumption about their commensurability – in other words, performance differences are treated as ordinals rather than real-valued. Furthermore, there is no normality assumption regarding the distribution of these differences[5] which means, among other things, that the test is less sensitive to outliers.

If we want to compare k algorithms over n data sets we need to use specialised significance tests to avoid that our confidence level drops with each additional pairwise comparison between algorithms. The *Friedman test* is designed for exactly this situation. Like the Wilcoxon signed-rank test it is based on ranked rather than absolute performance, and hence makes no assumption regarding the distribution of the performance measurements.[6] The idea is to rank the performance of all k algorithms per data set, from best performance (rank 1) to worst performance (rank k). Let R_{ij} denote the rank of the j-th algorithm on the i-th data set, and let $R_j = \left(\sum_i R_{ij}\right)/n$ be the average rank of the j-th algorithm. Under the null hypothesis that all algorithms perform equally these average ranks R_j should be the same. In order to test this we calculate the following quantities:

1. the average rank $\overline{R} = \dfrac{1}{nk}\sum_{ij} R_{ij} = \dfrac{k+1}{2}$;
2. the sum of squared differences $n\sum_j (R_j - \overline{R})^2$; and
3. the sum of squared differences $\dfrac{1}{n(k-1)}\sum_{ij}(R_{ij} - \overline{R})^2$.

There is an analogy with clustering here, in that the second quantity measures the spread between the rank 'centroids' – which we want to be large – and the third quantity measures the spread over all ranks. The Friedman statistic is the ratio of the former and latter quantities.

---

[5]In statistical terminology the test is 'non-parametric' as opposed to a parametric test such as the t-test which assumes a particular distribution. Parametric tests are generally more powerful when that assumed distribution is appropriate but can be misleading when it is not.

[6]A well-known parametric alternative to the Friedman test is *analysis of variance* (ANOVA).

Example 12.8 (Friedman test). We use the data from Example 12.4, assuming it comes from different data sets rather than cross-validation folds. The following table shows the ranks in brackets:

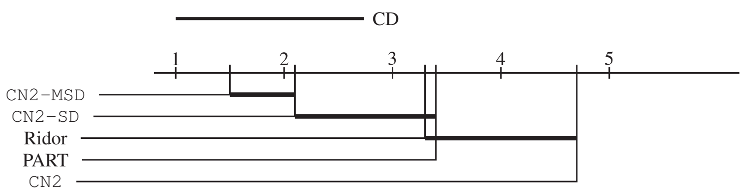
Data set	*Naive Bayes*	*Decision tree*	*Nearest neighbour*
1	0.6809 (3)	0.7524 (1)	0.7164 (2)
2	0.7017 (3)	0.8964 (1)	0.8883 (2)
3	0.7012 (2)	0.6803 (3)	0.8410 (1)
4	0.6913 (2)	0.9102 (1)	0.6825 (3)
5	0.6333 (3)	0.7758 (1)	0.7599 (2)
6	0.6415 (3)	0.8154 (2)	0.8479 (1)
7	0.7216 (1)	0.6224 (3)	0.7012 (2)
8	0.7214 (2)	0.7585 (1)	0.4959 (3)
9	0.6578 (3)	0.9380 (1)	0.9279 (2)
10	0.7865 (1)	0.7524 (2)	0.7455 (3)
avg rank	2.3	1.6	2.1

We have $\overline{R} = 2$, $n\sum_j (R_j - \overline{R})^2 = 2.6$ and $\frac{1}{n(k-1)} \sum_{ij} (R_{ij} - \overline{R})^2 = 1$, so the Friedman statistic is 2.6. The critical value for $k = 3$ and $n = 10$ at the $\alpha = 0.05$ level is 7.8, so we cannot reject the null hypothesis that all algorithms perform equally. In comparison, if the average ranks were 2.7, 1.3 and 2.0, then the null hypothesis would be rejected at that significance level.

The Friedman test tells us whether the average ranks as a whole display significant differences, but further analysis is needed on a pairwise level. This is achieved by applying a *post-hoc test* once the Friedman test gives significance. The idea is to calculate the *critical difference* (*CD*) against which the difference in average rank between two algorithms is compared. The *Nemenyi test* calculates the critical difference as follows:

$$\mathrm{CD} = q_\alpha \sqrt{\frac{k(k+1)}{6n}} \tag{12.1}$$

where q_α depends on the significance level α as well as k: for $\alpha = 0.05$ and $k = 3$ it is 2.343, leading to a critical difference of 1.047 in our example. If the average ranks are 2.7, 1.3 and 2.0, then only the difference between the first and second algorithm exceeds the critical difference. Figure 12.1 (top) demonstrates a useful visual representation of the results of the Nemenyi post-hoc test.



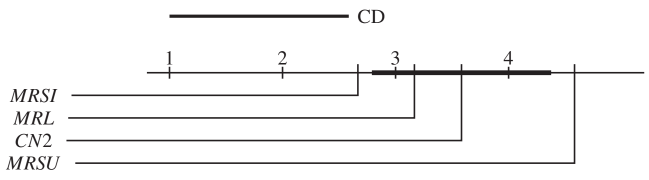


Figure 12.1. (top) Critical difference diagram for the pairwise Nemenyi test. Average ranks for each algorithm are plotted on the real axis. The critical difference is shown as a bar above the figure, and any group of consecutively ranked algorithms such that the outermost ones are less than the critical difference apart are connected by a horizontal thick line. The diagram shows, e.g., that the performance of the top ranked algorithm is significantly better than the bottom three. **(bottom)** Critical difference diagram for the Bonferroni–Dunn test with CN2 as control. The critical differences are now drawn symmetrically around the average rank of the control. The top ranked algorithm is significantly better than the control, and the bottom ranked one is significantly worse. (Figures courtesy of Tarek Abudawood (2011)).

A variant of the Nemenyi test called the *Bonferroni–Dunn test* can be applied when we perform pairwise tests only against a control algorithm. The calculation of the critical difference is the same, except q_α is adjusted to reflect the fact that we make $k-1$ pairwise comparisons rather than $k(k-1)/2$. For example, for $\alpha = 0.05$ and $k = 3$ we have $q_\alpha = 2.241$, which is slightly lower than the value used for the Nemenyi test, leading to a tighter critical difference. Figure 12.1 (bottom) shows a graphical representation of the Bonferroni–Dunn post-hoc test.

12.4 Machine learning experiments: Summary and further reading

In this chapter we have taken a look at how we can use data to answer questions about the performance of models and learning algorithms. A ‘machine learning experimenter’ needs to address three questions: (*i*) what to measure, (*ii*) how to measure it, and (*iii*) how to interpret it. An excellent source – particularly for the last two questions – is Japkowicz and Shah (2011).

☞ In order to decide what to measure, we first need to explicate our experimental objective. We also need to consider the operating context: performance aspects that might change when using the model. For example, the operating context might be given by the class distribution, but we may have no prior knowledge telling us that certain distributions are more likely than others. Example 12.1 on p.345 demonstrated that in such a case average recall would be more appropriate as a performance measure, even if the experimental objective is accuracy. We also looked at the difference between a precision–recall analysis which ignores the true negatives, and a true/false positive rate analysis which takes them into account; a fuller analysis is provided by Davis and Goadrich (2006). The relation between accuracy as experimental objective and AUC as performance measure is studied by Hernández-Orallo *et al.* (2011).

☞ Once we decided what to measure, we need to establish a measuring protocol. The most common approach is k-fold cross-validation, which divides the data into k folds, repeatedly trains on $k-1$ of those and tests on the remaining one. It is of paramount importance that there be no information leak between the training data used to learn the model and the test data used to evaluate it. A common mistake is to use cross-validation to find the best setting of one or more parameters of a learning algorithm, say the complexity parameter of a support vector machine. This is methodologically wrong as parameter tuning should be carried out as part of the training process, without any access to the test data. A methodologically sound option is to use *internal cross-validation* by setting aside a validation fold in each cross-validation run for parameter tuning. Experimental studies regarding cross-validation are carried out by Dietterich (1998) and Bouckaert and Frank (2004): the former recommends five times two-fold cross-validation and the latter ten times ten-fold. ROC curves can be drawn in cross-validation as each instance appears in a test fold exactly once, and so we can collect the scores on all test folds. Fawcett (2006) considers alternatives including horizontal and vertical averaging.

☞ In the context of interpreting experimental results we looked at confidence intervals and significance tests. Confidence intervals have a clear statistical interpretation: they quantify the likelihood of a measurement falling in a particular interval, assuming a particular true value. Significance tests extend this to reasoning about a particular null hypothesis, such as 'these learning algorithms do not perform differently on these data sets'. Significance tests are designed for particular protocols: the t-test can be used for evaluating two learning algorithms on two data sets, Wilcoxon's signed-rank test is applicable for comparing two algorithms over multiple data sets, and Friedman's test (or analysis of variance) compares multiple algorithms over multiple data sets. An excellent discussion of these and related tests is provided by Demšar (2006).

☞ It should be mentioned that there is much discussion on the use of significance tests in machine learning, and on the wider issue regarding machine learning as an experimental science. The importance of experiments in machine learning was stressed early on by Pat Langley in two influential papers (Langley, 1988; Kibler and Langley, 1988); however, more recently he expressed criticism at the way experimental methodology in machine learning has become rather inflexible (Langley, 2011). Other authors critical of current practice include Drummond (2006) and Demšar (2008).