Dataset splitting

It is standard in ML to split data into training and test sets. The reason for this is very straightforward: if you try and evaluate your system on data you have trained it on, you are doing something unrealistic. The whole point of a machine learning system is to be able to work with unseen data: if you know you are going to see all possible values in your training data, you might as well just use some form of lookup.

However, the two-way split is over-simplistic. Real ML typically involves four phases:

1. Training
2. Development (also known as Validation or Tuning)
3. Testing (aka Evaluation)
4. Use

For supervised learning, the available labelled data is usually quite limited. This data must be split to be used in the first three phases. The fourth phase is usually not explicitly discussed in research, but of course the aim of real ML is to get the best possible performance in actual use, where you do not have ground truth values. This should always be born in mind when we talk about evaluation and statistical methodology: the ultimate aim is to make sure that the performance in the use phase is as high as possible and to avoid fooling oneself into thinking one has a system that performs better than it actually does. Evaluation and statistical methods are used to help us achieve that goal (and to convince others that we've achieved it).

Why is the separate development/validation phase necessary? This is clearest in those varieties of ML where parameters are automatically tuned on some data which is separate from the training data. However, it is actually necessary to make such a split in all cases where one might alter the experimental conditions after the first test. Essentially this means one should always have a development set. The number of times the evaluation can be run on the test data must be strictly limited. Ideally, the test data should only be used once.

To see the problem with repeated tests, imagine a situation where you want to compare the performance of two methods in a binary classification task. You have a balanced dataset with 1000 items. Following the standard advice, you use 900 items for training and 100 for test. You discover that an existing method A achieves 68% accuracy on the test data, while your own method B only achieves 64%. You are sad. But then you realise that you could alter a setting, which might give you better results with method B. You do this and try again. You get 62%. You are even sadder. But you persist, and after 8 experiments, you achieve 77%: a massive improvement. You are happy, stop there, and report your results.

What's wrong with this? Are you fooling yourself? To put this another way, if you are honest about the testing you had done, should a (healthily skeptical) observer be convinced that with these new settings method B will deliver better results on new data than method A? Ideally, you should think about this before reading on.

Consider the possibility that the truth is that both methods can completely reliably classify 40% of the real data, and perform at chance on the other 60%. That is, although each method is deterministic (i.e., it will always give the same result for a given item), for 60% of the data its classification accuracy is no better that flipping a coin would be. Assume that the methods are not identical, in that their 'at chance' prediction is different for different test items.

These assumptions lead to an overall accuracy of 70% for both methods when tested on sufficiently large quantities of data. That is, both methods are really accurate on 40% of the data, and just get lucky half the time on the remaining 60%. However, the results for a sample will vary somewhat from this true accuracy, due to random variation. This also means that the results from method A and B are likely to differ.

Now consider the possibility that the settings you alter in your experiments on method B have the effect of randomly flipping some of the results on the 60% of the data where performance is at chance. The settings do not actually improve the performance, but merely change the results for particular test items. If you alter the settings in this way enough times on the same test set, you will eventually get a run where method B is markedly superior to A. (If you are not convinced, try simulating this situation: see the starred tick suggestion at the end.)

Of course, one does not intentionally change parameters in a way which results in random flipping of the classification function. But as anyone who has worked with ML knows, it is almost impossible to reliably predict the effect of a change in parameters or the use of different features. It is very often the case that changing something in a system will fix one example in the test data and break another. Hence, this thought experiment is not so implausible as it may initially sound.

The conclusion is that the use of a separate development set is recommended because it allows one to experiment without risking fooling oneself (and others) in this way.

Note: This discussion has deliberately avoided talking about statistical methods because this can obscure the essential point. However, we note here that it is unfortunately very common for researchers to run multiple experiments and then do a standard test for statistical significance on the best result. This is invalid (it is known as the 'Multiple Comparisons Problem'). This is not an issue if the test set is only used once, of course.

Starred tick (recommended). Implement a simulation of the scenario described above and investigate what sort of variation you get in the test runs. You should also try simulating a scenario where altering the experimental settings leads to a slight improvement in behaviour and look at the amount of test data you need to reliably show an effect.