# 19 *Design and Analysis of Machine Learning Experiments*

*We discuss the design of machine learning experiments to assess and compare the performances of learning algorithms in practice and the statistical tests to analyze the results of these experiments.*

## 19.1 Introduction

IN PREVIOUS chapters, we discussed several learning algorithms and saw that, given a certain application, more than one is applicable. Now, we are concerned with two questions:

1. How can we assess the expected error of a learning algorithm on a problem? That is, for example, having used a classification algorithm to train a classifier on a dataset drawn from some application, can we say with enough confidence that later on when it is used in real life, its expected error rate will be less than, for example, 2 percent?

2. Given two learning algorithms, how can we say one has less error than the other one, for a given application? The algorithms compared can be different, for example, parametric versus nonparametric, or they can use different hyperparameter settings. For example, given a multilayer perceptron (chapter 11) with four hidden units and another one with eight hidden units, we would like to be able to say which one has less expected error. Or with the $k$-nearest neighbor classifier (chapter 8), we would like to find the best value of $k$.

We cannot look at the training set errors and decide based on those. The error rate on the training set, by definition, is always smaller than the error rate on a test set containing instances unseen during training.

Similarly, training errors cannot be used to compare two algorithms. This is because over the training set, the more complex model having more parameters will almost always give fewer errors than the simple one.

So as we have repeatedly discussed, we need a validation set that is different from the training set. Even over a validation set though, just one run may not be enough. There are two reasons for this: First, the training and validation sets may be small and may contain exceptional instances, like noise and outliers, which may mislead us. Second, the learning method may depend on other random factors affecting generalization. For example, with a multilayer perceptron trained using backpropagation, because gradient descent converges to the nearest local minimum, the initial weights affect the final weights, and given the exact same architecture and training set, starting from different initial weights, there may be multiple possible final classifiers having different error rates on the same validation set. We thus would like to have several runs to average over such sources of randomness. If we train and validate only once, we cannot test for the effect of such factors; this is only admissible if the learning method is so costly that it can be trained and validated only once.

We use a *learning algorithm* on a dataset and generate a *learner*. If we do the training once, we have one learner and one validation error. To average over randomness (in training data, initial weights, etc.), we use the same algorithm and generate multiple learners. We test them on multiple validation sets and record a sample of validation errors. (Of course, all the training and validation sets should be drawn from the same application.) We base our evaluation of the learning algorithm on the *distribution* of these validation errors. We can use this distribution for assessing the

EXPECTED ERROR

*expected error* of the learning algorithm for that problem, or compare it with the error rate distribution of some other learning algorithm.

Before proceeding to how this is done, it is important to stress a number of points:

1. We should keep in mind that whatever conclusion we draw from our analysis is conditioned on the dataset we are given. We are not comparing learning algorithms in a domain independent way but on some particular application. We are not saying anything about the expected error of a learning algorithm, or comparing one learning algorithm with another algorithm, in general. Any result we have is only true for the particular application, and only insofar as that application is rep-

NO FREE LUNCH THEOREM

resented in the sample we have. And anyway, as stated by the *No Free Lunch Theorem* (Wolpert 1995), there is no such thing as the "best" learning algorithm. For any learning algorithm, there is a dataset where it is very accurate and another dataset where it is very poor. When we say that a learning algorithm is good, we only quantify how well its inductive bias matches the properties of the data.

2. The division of a given dataset into a number of training and validation set pairs is only for testing purposes. Once all the tests are complete and we have made our decision as to the final method or hyperparameters, to train the final learner, we can use all the labeled data that we have previously used for training or validation.

3. Because we also use the validation set(s) for testing purposes, for example, for choosing the better of two learning algorithms, or to decide where to stop learning, it effectively becomes part of the data we use. When after all such tests, we decide on a particular algorithm and want to report its expected error, we should use a separate *test set* for this purpose, unused during training this final system. This data should have never been used before for training or validation and should be large for the error estimate to be meaningful. So, given a dataset, we should first leave some part of it aside as the test set and use the rest for training and validation. Typically, we can leave one-third of the sample as the test set, then use two-thirds for cross-validation to generate multiple training/validation set pairs, as we will see shortly. So, the training set is used to optimize the parameters, given a particular learning algorithm and model structure; the validation set is used to optimize the hyperparameters of the learning algorithm or the model structure; and the test set is used at the end, once both these have been optimized. For example, with an MLP, the training set is used to optimize the weights, the validation set is used to decide on the number of hidden units, how long to train, the learning rate, and so forth. Once the best MLP configuration is chosen, its final error is calculated on the test set. With $k$-NN, the training set is stored as the lookup table; we optimize the distance measure and $k$ on the validation set and test finally on the test set.

4. In general, we compare learning algorithms by their error rates, but it should be kept in mind that in real life, error is only one of the criteria that affect our decision. Some other criteria are (Turney 2000):

- risks when errors are generalized using loss functions, instead of 0/1 loss (section 3.3),
- training time and space complexity,
- testing time and space complexity,
- interpretability, namely, whether the method allows knowledge extraction which can be checked and validated by experts, and
- easy programmability.

The relative importances of these factors change depending on the application. For example, if the training is to be done once in the factory, then training time and space complexity are not important; if adaptability during use is required, then they do become important. Most of the learning algorithms use 0/1 loss and take error as the single criterion to be minimized; recently, *cost-sensitive learning* variants of these algorithms have also been proposed to take other cost criteria into account.

COST-SENSITIVE LEARNING

When we train a learner on a dataset using a training set and test its accuracy on some validation set and try to draw conclusions, what we are doing is experimentation. Statistics defines a methodology to design experiments correctly and analyze the collected data in a manner so as to be able to extract significant conclusions (Montgomery 2005). In this chapter, we will see how this methodology can be used in the context of machine learning.

## 19.2 Factors, Response, and Strategy of Experimentation

As in other branches of science and engineering, in machine learning too, we do experiments to get information about the process under scrutiny. In our case, this is a learner, which, having been trained on a dataset, generates an output for a given input. An *experiment* is a test or a series of tests where we play with the *factors* that affect the output. These factors may be the algorithm used, the training set, input features, and so on, and we observe the changes in the *response* to be able to extract information. The aim may be to identify the most important factors, screen the unimportant ones, or find the configuration of the factors that optimizes the response—for example, classification accuracy on a given test set.

EXPERIMENT

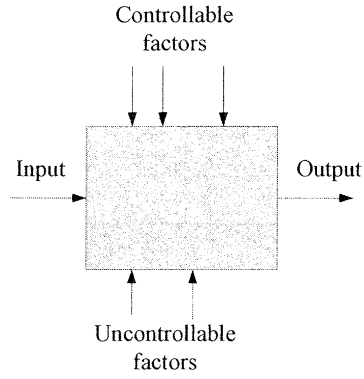



**Figure 19.1** The process generates an output given an input and is affected by controllable and uncontrollable factors.

Our aim is to plan and conduct machine learning experiments and analyze the data resulting from the experiments, to be able to eliminate the effect of chance and obtain conclusions which we can consider *statistically significant.* In machine learning, we target a learner having the highest generalization accuracy and the minimal complexity (so that its implementation is cheap in time and space) and is robust, that is, minimally affected by external sources of variability.

A trained learner can be shown as in figure 19.1; it gives an output, for example, a class code for a test input, and this depends on two type of factors: The *controllable factors*, as the name suggests, are those we have control on. The most basic is the learning algorithm used. There are also the hyperparameters of the algorithm, for example, the number of hidden units for a multilayer perceptron, $k$ for $k$-nearest neighbor, $C$ for support vector machines, and so on. The dataset used and the input representation, that is, how the input is coded as a vector, are other controllable factors.

There are also *uncontrollable factors* over which we have no control, adding undesired variability to the process, which we do not want to affect our decisions. Among these are the noise in the data, the particular training subset if we are resampling from a large set, randomness in the optimization process, for example, the initial state in gradient descent with multilayer perceptrons, and so on.

We use the output to generate the *response* variable—for example, av-

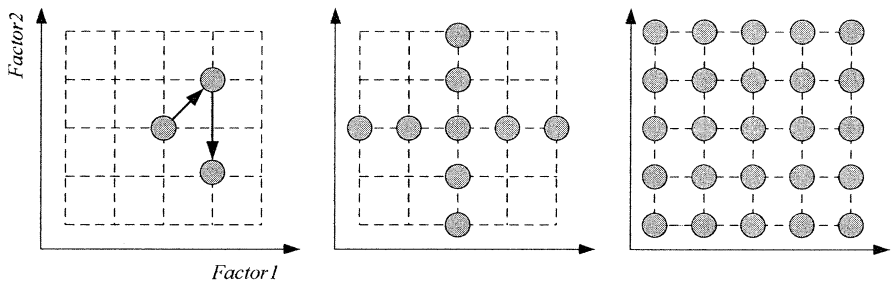



**Figure 19.2** Different strategies of experimentation with two factors and five levels each.

erage classification error on a test set, or the expected risk using a loss function, or some other measure, such as precision and recall, as we will discuss shortly.

Given several factors, we need to find the best setting for best response, or in the general case, determine their effect on the response variable. For example, we may be using principal components analyzer (PCA) to reduce dimensionality to $d$ before a $k$-nearest neighbor ($k$-NN) classifier. The two factors are $d$ and $k$, and the question is to decide which combination of $d$ and $k$ leads to highest performance. Or, we may be using a support vector machine classifier with Gaussian kernel, and we have the regularization parameter $C$ and the spread of the Gaussian $s^2$ to fine-tune together.

STRATEGIES OF EXPERIMENTATION

There are several *strategies of experimentation*, as shown in figure 19.2. In the *best guess* approach, we start at some setting of the factors that we believe is a good configuration. We test the response there and we fiddle with the factors one (or very few) at a time, testing each combination until we get to a state that we consider is good enough. If the experimenter has a good intuition of the process, this may work well; but note that there is no systematic approach to modify the factors and when we stop, we have no guarantee of finding the best configuration.

Another strategy is to modify *one factor at a time* where we decide on a baseline (default) value for all factors, and then we try different levels for one factor while keeping all other factors at their baseline. The major disadvantage of this is that it assumes that there is no *interaction* between the factors, which may not always be true. In the PCA/$k$-NN cascade we discussed earlier, each choice for $d$ defines a different input

space for $k$-NN where a different $k$ value may be appropriate.

FACTORIAL DESIGN

The correct approach is to use a *factorial design* where factors are varied together, instead of one at a time; this is colloquially called *grid search*. With $F$ factors at $L$ levels each, searching one factor at a time takes $\mathcal{O}(L \cdot F)$ time, whereas a factorial experiment takes $\mathcal{O}(L^F)$ time.

## 19.3 Response Surface Design

To decrease the number of runs necessary, one possibility is to run a fractional factorial design where we run only a subset, another is to try to use knowledge gathered from previous runs to estimate configurations that seem likely to have high response. In searching one factor at a time, if we can assume that the response is typically quadratic (with a single maximum, assuming we are maximizing a response value, such as the test accuracy), then instead of trying all values, we can have an iterative procedure where starting from some initial runs, we fit a quadratic, find its maximum analytically, take that as the next estimate, run an experiment there, add the resulting data to the sample, and then continue fitting and sampling, until we get no further improvement.

RESPONSE SURFACE DESIGN

With many factors, this is generalized as the *response surface design* method where we try to fit a parametric response function to the factors as

$$r = g(f_1, f_2, \ldots, f_F | \phi)$$

where $r$ is the response and $f_i, i = 1, \ldots, F$ are the factors. This fitted parametric function defined given the parameters $\phi$ is our empirical model estimating the response for a particular configuration of the (controllable) factors; the effect of uncontrollable factors is modeled as noise. $g(\cdot)$ is a (typically quadratic) regression model and after a small number of runs around some baseline (as defined by a so-called *design matrix*), one can have enough data to fit $g(\cdot)$ on. Then, we can analytically calculate the values of $f_i$ where the fitted $g$ is maximum, which we take as our next guess, run an experiment there, get a data instance, add it to the sample, fit $g$ once more, and so on, until there is convergence. Whether this approach will work well or not depends on whether the response can indeed be written as a quadratic function of the factors with a single maximum.

## 19.4 Randomization, Replication, and Blocking

Let us now talk about the three basic principles of experimental design.

RANDOMIZATION

- *Randomization* requires that the order in which the runs are carried out should be randomly determined so that the results are independent. This is typically a problem in real-world experiments involving physical objects; for example, machines require some time to warm up until they operate in their normal range so tests should be done in random order for time not to bias the results. Ordering generally is not a problem in software experiments.

REPLICATION

- *Replication* implies that for the same configuration of (controllable) factors, the experiment should be run a number of times to average over the effect of uncontrollable factors. In machine learning, this is typically done by running the same algorithm on a number of resampled versions of the same dataset; this is known as *cross-validation*, which we will discuss in section 19.6. How the response varies on these different replications of the same experiment allows us to obtain an estimate of the experimental error (the effect of uncontrollable factors), which we can in turn use to determine how large differences should be to be deemed *statistically significant*.

BLOCKING

- *Blocking* is used to reduce or eliminate the variability due to *nuisance factors* that influence the response but in which we are not interested. For example, defects produced in a factory may also depend on the different batches of raw material, and this effect should be isolated from the controllable factors in the factory, such as the equipment, personnel, and so on. In machine learning experimentation, when we use resampling and use different subsets of the data for different replicates, we need to make sure that for example if we are comparing learning algorithms, they should all use the same set of resampled subsets, otherwise the differences in accuracies would depend not only on the algorithms but also on the different subsets—to be able to measure the difference due to algorithms only, the different training sets in replicated runs should be identical; this is what we mean by blocking.

PAIRING

In statistics, if there are two populations, this is called *pairing* and is used in *paired testing*.

## 19.5 Guidelines for Machine Learning Experiments

Before we start experimentation, we need to have a good idea about what it is we are studying, how the data is to be collected, and how we are planning to analyze it. The steps in machine learning are the same as for any type of experimentation (Montgomery 2005). Note that at this point, it is not important whether the task is classification or regression, or whether it is an unsupervised or a reinforcement learning application. The same overall discussion applies; the difference is only in the sampling distribution of the response data that is collected.

### A. Aim of the Study

We need to start by stating the problem clearly, defining what the objectives are. In machine learning, there may be several possibilities. As we discussed before, we may be interested in assessing the expected error (or some other response measure) of a learning algorithm on a particular problem and check that, for example, the error is lower than a certain acceptable level.

Given two learning algorithms and a particular problem as defined by a dataset, we may want to determine which one has less generalization error. These can be two different algorithms, or one can be a proposed improvement of the other, for example, by using a better feature extractor.

In the general case, we may have more than two learning algorithms, and we may want to choose the one with the least error, or order them in terms of error, for a given dataset.

In an even more general setting, instead of on a single dataset, we may want to compare two or more algorithms on two or more datasets.

### B. Selection of the Response Variable

We need to decide on what we should use as the quality measure. Most frequently, error is used that is the misclassification error for classification and mean square error for regression. We may also use some variant; for example, generalizing from 0/1 to an arbitrary loss, we may use a risk measure. In information retrieval, we use measures such as precision and recall; we will discuss such measures in section 19.7. In a cost-sensitive

setting, not only the output but also system parameters, for example, its complexity, are taken into account.

### C. Choice of Factors and Levels

What the factors are depend on the aim of the study. If we fix an algorithm and want to find the best hyperparameters, then those are the factors. If we are comparing algorithms, the learning algorithm is a factor. If we have different datasets, they also become a factor.

The levels of a factor should be carefully chosen so as not to miss a good configuration and avoid doing unnecessary experimentation. It is always good to try to normalize factor levels. For example, in optimizing $k$ of $k$-nearest neighbor, one can try values such as 1, 3, 5, and so on, but in optimizing the spread $h$ of Parzen windows, we should not try absolute values such as 1.0, 2.0, and so on, because that depends on the scale of the input; it is better to find some statistic that is an indicator of scale—for example, the average distance between an instance and its nearest neighbor—and try $h$ as different multiples of that statistic.

Though previous expertise is a plus in general, it is also important to investigate all factors and factor levels that may be of importance and not be overly influenced by past experience.

### D. Choice of Experimental Design

It is always better to do a factorial design unless we are sure that the factors do not interact, because mostly they do. Replication number depends on the dataset size; it can be kept small when the dataset is large; we will discuss this in the next section when we talk about resampling. However, too few replicates generate few data and this will make comparing distributions difficult; in the particular case of parametric tests, the assumptions of Gaussianity may not be tenable.

Generally, given some dataset, we leave some part as the test set and use the rest for training and validation, probably many times by resampling. How this division is done is important. In practice, using small datasets leads to responses with high variance, and the differences will not be significant and results will not be conclusive.

It is also important to avoid as much as possible toy, synthetic data and use datasets that are collected from real-world under real-life circumstances. Didactic one- or two-dimensional datasets may help provide

intuition, but the behavior of the algorithms may be completely different in high-dimensional spaces.

### E. Performing the Experiment

Before running a large factorial experiment with many factors and levels, it is best if one does a few trial runs for some random settings to check that all is as expected. In a large experiment, it is always a good idea to save intermediate results (or seeds of the random number generator), so that a part of the whole experiment can be rerun when desired. All the results should be reproducable. In running a large experiment with many factors and factor levels, one should be aware of the possible negative effects of software aging.

It is important that an experimenter be unbiased during experimentation. In comparing one's favorite algorithm with a competitor, both should be investigated equally diligently. In large-scale studies, it may even be envisaged that testers be different from developers.

One should avoid the temptation to write one's own "library" and instead, as much as possible, use code from reliable sources; such code would have been better tested and optimized.

As in any software development study, the advantages of good documentation cannot be underestimated, especially when working in groups. All the methods developed for high-quality software engineering should also be used in machine learning experiments.

### F. Statistical Analysis of the Data

This corresponds to analyzing data in a way so that whatever conclusion we get is not subjective or due to chance. We cast the questions that we want to answer in a hypothesis testing framework and check whether the sample supports the hypothesis. For example, the question "Is $A$ a more accurate algorithm than $B$?" becomes the hypothesis "Can we say that the average error of learners trained by $A$ is significantly lower than the average error of learners trained by $B$?"

As always, visual analysis is helpful, and we can use histograms of error distributions, whisker-and-box plots, range plots, and so on.

### G. Conclusions and Recommendations

Once all data is collected and analyzed, we can draw objective conclusions. One frequently encountered conclusion is the need for further experimentation. Most statistical, and hence machine learning or data mining, studies are iterative. It is for this reason that we never start with all the experimentation. It is suggested that no more than 25 percent of the available resources should be invested in the first experiment (Montgomery 2005). The first runs are for investigation only. That is also why it is a good idea not to start with high expectations, or promises to one's boss or thesis advisor.

We should always remember that statistical testing never tells us if the hypothesis is correct or false, but how much the sample seems to concur with the hypothesis. There is always a risk that we do not have a conclusive result or that our conclusions be wrong, especially if the data is small and noisy.

When our expectations are not met, it is most helpful to investigate why they are not. For example, in checking why our favorite algorithm $A$ has worked awfully bad on some cases, we can get a splendid idea for some improved version of $A$. All improvements are due to the deficiencies of the previous version; finding a deficiency is but a helpful hint that there is an improvement we can make!

But we should not go to the next step of testing the improved version before we are sure that we have completely analyzed the current data and learned all we could learn from it. Ideas are cheap, and useless unless tested, which is costly.

## 19.6 Cross-Validation and Resampling Methods

For replication purposes, our first need is to get a number of training and validation set pairs from a dataset $\mathcal{X}$ (after having left out some part as the test set). To get them, if the sample $\mathcal{X}$ is large enough, we can randomly divide it into $K$ parts, then randomly divide each part into two and use one half for training and the other half for validation. $K$ is typically 10 or 30. Unfortunately, datasets are never large enough to do this. So we should do our best with small datasets. This is done by repeated use of the same data split differently; this is called *cross-validation*. The catch is that this makes the error percentages dependent as these different sets share data.

CROSS-VALIDATION

So, given a dataset $\mathcal{X}$, we would like to generate $K$ training/validation set pairs, $\{\mathcal{T}_i, \mathcal{V}_i\}_{i=1}^K$, from this dataset. We would like to keep the training and validation sets as large as possible so that the error estimates are robust, and at the same time, we would like to keep the overlap between different sets as small as possible. We also need to make sure that classes are represented in the right proportions when subsets of data are held out, not to disturb the class prior probabilities; this is called *stratification.* If a class has 20 percent examples in the whole dataset, in all samples drawn from the dataset, it should also have approximately 20 percent examples.

STRATIFICATION

### 19.6.1 K-Fold Cross-Validation

K-FOLD CROSS-VALIDATION

In K-*fold cross-validation,* the dataset $\mathcal{X}$ is divided randomly into $K$ equal-sized parts, $\mathcal{X}_i, i = 1, \ldots, K$. To generate each pair, we keep one of the $K$ parts out as the validation set and combine the remaining $K - 1$ parts to form the training set. Doing this $K$ times, each time leaving out another one of the $K$ parts out, we get $K$ pairs:

$$\begin{aligned}
\mathcal{V}_1 &= \mathcal{X}_1 \quad \mathcal{T}_1 = \mathcal{X}_2 \cup \mathcal{X}_3 \cup \cdots \cup \mathcal{X}_K \\
\mathcal{V}_2 &= \mathcal{X}_2 \quad \mathcal{T}_2 = \mathcal{X}_1 \cup \mathcal{X}_3 \cup \cdots \cup \mathcal{X}_K \\
&\vdots \\
\mathcal{V}_K &= \mathcal{X}_K \quad \mathcal{T}_K = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \cdots \cup \mathcal{X}_{K-1}
\end{aligned}$$

There are two problems with this. First, to keep the training set large, we allow validation sets that are small. Second, the training sets overlap considerably, namely, any two training sets share $K - 2$ parts.

$K$ is typically 10 or 30. As $K$ increases, the percentage of training instances increases and we get more robust estimators, but the validation set becomes smaller. Furthermore, there is the cost of training the classifier $K$ times, which increases as $K$ is increased. As $N$ increases, $K$ can be smaller; if $N$ is small, $K$ should be large to allow large enough training sets. One extreme case of $K$-fold cross-validation is *leave-one-out* where given a dataset of $N$ instances, only one instance is left out as the validation set (instance) and training uses the $N - 1$ instances. We then get $N$ separate pairs by leaving out a different instance at each iteration. This is typically used in applications such as medical diagnosis, where labeled data is hard to find. Leave-one-out does not permit stratification.

LEAVE-ONE-OUT

Recently, with computation getting cheaper, it has also become possible to have multiple runs of $K$-fold cross-validation, for example, $10 \times 10$-

fold, and use average over averages to get more reliable error estimates (Bouckaert 2003).

### 19.6.2 5×2 Cross-Validation

5 × 2 CROSS-VALIDATION

Dietterich (1998) proposed the $5 \times 2$ *cross-validation*, which uses training and validation sets of equal size. We divide the dataset $\mathcal{X}$ randomly into two parts, $\mathcal{X}_1^{(1)}$ and $\mathcal{X}_1^{(2)}$, which gives our first pair of training and validation sets, $\mathcal{T}_1 = \mathcal{X}_1^{(1)}$ and $\mathcal{V}_1 = \mathcal{X}_1^{(2)}$. Then we swap the role of the two halves and get the second pair: $\mathcal{T}_2 = \mathcal{X}_1^{(2)}$ and $\mathcal{V}_2 = \mathcal{X}_1^{(1)}$. This is the first fold; $\mathcal{X}_i^{(j)}$ denotes half $j$ of fold $i$.

To get the second fold, we shuffle $\mathcal{X}$ randomly and divide this new fold into two, $\mathcal{X}_2^{(1)}$ and $\mathcal{X}_2^{(2)}$. This can be implemented by drawing these from $\mathcal{X}$ randomly without replacement, namely, $\mathcal{X}_1^{(1)} \cup \mathcal{X}_1^{(2)} = \mathcal{X}_2^{(1)} \cup \mathcal{X}_2^{(2)} = \mathcal{X}$. We then swap these two halves to get another pair. We do this for three more folds and because from each fold, we get two pairs, doing five folds, we get ten training and validation sets:

$$
\begin{array}{ll}
\mathcal{T}_1 = \mathcal{X}_1^{(1)} & \mathcal{V}_1 = \mathcal{X}_1^{(2)} \\
\mathcal{T}_2 = \mathcal{X}_1^{(2)} & \mathcal{V}_2 = \mathcal{X}_1^{(1)} \\
\mathcal{T}_3 = \mathcal{X}_2^{(1)} & \mathcal{V}_3 = \mathcal{X}_2^{(2)} \\
\mathcal{T}_4 = \mathcal{X}_2^{(2)} & \mathcal{V}_4 = \mathcal{X}_2^{(1)} \\
\quad \vdots & \\
\mathcal{T}_9 = \mathcal{X}_5^{(1)} & \mathcal{V}_9 = \mathcal{X}_5^{(2)} \\
\mathcal{T}_{10} = \mathcal{X}_5^{(2)} & \mathcal{V}_{10} = \mathcal{X}_5^{(1)}
\end{array}
$$

Of course, we can do this for more than five folds and get more training/validation sets, but Dietterich (1998) points out that after five folds, the sets share many instances and overlap so much that the statistics calculated from these sets, namely, validation error rates, become too dependent and do not add new information. Even with five folds, the sets overlap and the statistics are dependent, but we can get away with this until five folds. On the other hand, if we do have fewer than five folds, we get less data (fewer than ten sets) and will not have a large enough sample to fit a distribution to and test our hypothesis on.

**Table 19.1** Confusion matrix for two classes.

| | Predicted class | | |
|---|---|---|---|
| True Class | Positive | Negative | Total |
| Positive | $tp$ : true positive | $fn$ : false negative | $p$ |
| Negative | $fp$ : false positive | $tn$ : true negative | $n$ |
| Total | $p'$ | $n'$ | $N$ |

### 19.6.3 Bootstrapping

BOOTSTRAP

To generate multiple samples from a single sample, an alternative to cross-validation is the *bootstrap* that generates new samples by drawing instances from the original sample *with* replacement. We saw the use of bootstrapping in section 17.6 to generate training sets for different learners in bagging. The bootstrap samples may overlap more than cross-validation samples and hence their estimates are more dependent; but is considered the best way to do resampling for very small datasets.

In the bootstrap, we sample $N$ instances from a dataset of size $N$ with replacement. The original dataset is used as the validation set. The probability that we pick an instance is $1/N$; the probability that we do not pick it is $1 - 1/N$. The probability that we do not pick it after $N$ draws is

$$\left(1 - \frac{1}{N}\right)^N \approx e^{-1} = 0.368$$

This means that the training data contains approximately 63.2 percent of the instances; that is, the system will not have been trained on 36.8 percent of the data, and the error estimate will be pessimistic. The solution is replication, that is, to repeat the process many times and look at the average behavior.

## 19.7 Measuring Classifier Performance

For classification, especially for two-class problems, a variety of measures has been proposed. There are four possible cases, as shown in table 19.1. For a positive example, if the prediction is also positive, this is a *true positive*; if our prediction is negative for a positive example, this is a *false negative*. For a negative example, if the prediction is also negative, we

**Table 19.2** Performance measures used in two-class problems.

| Name | Formula |
|---|---|
| error | $(fp + fn)/N$ |
| accuracy | $(tp + tn)/N = 1 - \text{error}$ |
| tp-rate | $tp/p$ |
| fp-rate | $fp/n$ |
| precision | $tp/p'$ |
| recall | $tp/p = \text{tp-rate}$ |
| sensitivity | $tp/p = \text{tp-rate}$ |
| specificity | $tn/n = 1 - \text{fp-rate}$ |

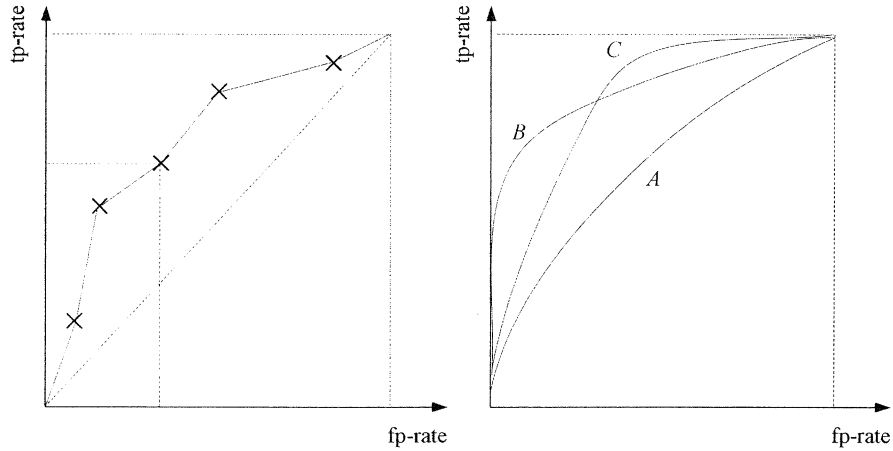
have a *true negative*, and we have a *false positive* if we predict a negative example as positive.

In some two-class problems, we make a distinction between the two classes and hence the two type of errors, false positives and false negatives. Different measures appropriate in different settings are given in table 19.2. Let us envisage an authentication application where, for example, users log on to their accounts by voice. A false positive is wrongly logging on an impostor and a false negative is refusing a valid user. It is clear that the two type of errors are not equally bad; the former is much worse. True positive rate, *tp-rate*, also known as *hit rate*, measures what proportion of valid users we authenticate and false positive rate, *fp-rate*, also known as *false alarm rate*, is the proportion of impostors we wrongly accept.

Let us say the system returns $\hat{P}(C_1|x)$, the probability of the positive class, and for the negative class, we have $\hat{P}(C_2|x) = 1 - \hat{P}(C_1|x)$, and we choose "positive" if $\hat{P}(C_1|x) > \theta$. If $\theta$ is close to 1, we hardly choose the positive class; that is, we will have no false positives but also few true positives. As we decrease $\theta$ to increase the number of true positives, we risk introducing false positives.

RECEIVER OPERATING CHARACTERISTICS

For different values of $\theta$, we can get a number of pairs of (tp-rate, fp-rate) values and by connecting them we get the *receiver operating characteristics* (ROC) curve, as shown in figure 19.3a. Note that different values of $\theta$ correspond to different loss matrices for the two types of error and the ROC curve can also be seen as the behavior of a classifier




(a) Example ROC curve

(b) Different ROC curves for different classifiers

**Figure 19.3** (a) Typical ROC curve. Each classifier has a threshold that allows us to move over this curve, and we decide on a point, based on the relative importance of hits versus false alarms, namely, true positives and false positives. The area below the ROC curve is called AUC. (b) A classifier is preferred if its ROC curve is closer to the upper-left corner (larger AUC). *B* and *C* are preferred over *A*; *B* and *C* are preferred under different loss matrices.

under different loss matrices (see exercise 1).

Ideally, a classifier has a tp-rate of 1 and a fp-rate of 0, and hence a classifier is better the more it gets closer to the upper-left corner. On the diagonal, we make as many true decisions as false ones, and this is the worst one can do (any classifier that is below the diagonal can be improved by flipping its decision). Given two classifiers, we can say one is better than the other one if it is above the other one; if two ROC curves intersect, we can say that the two classifiers are better under different loss conditions, as seen in figure 19.3b.

AREA UNDER THE CURVE

ROC allows a visual analysis; if we want to reduce the curve to a single number we can do this by calculating the *area under the curve* (AUC) . A classifier ideally has an AUC of 1 and AUC values of different classifiers can be compared to give us a general performance averaged over different loss conditions.

INFORMATION RETRIEVAL

In *information retrieval*, there is a database of records; we make a

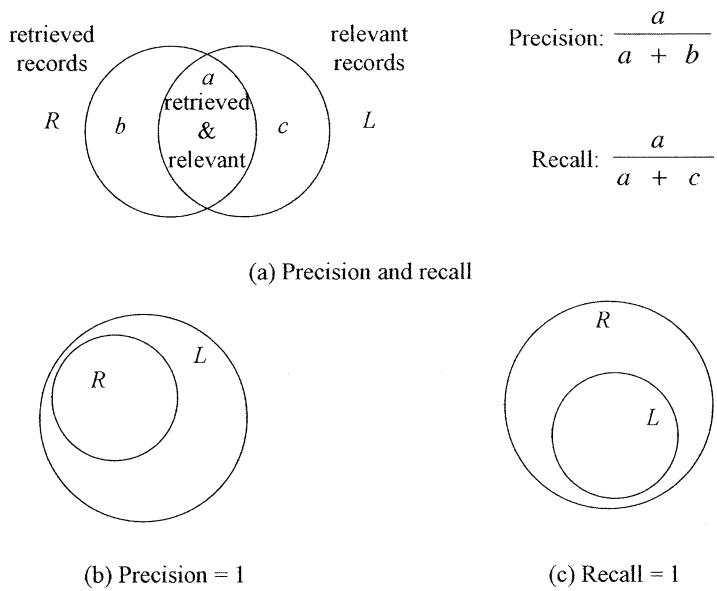



**Figure 19.4** (a) Definition of precision and recall using Venn diagrams. (b) Precision is 1; all the retrieved records are relevant but there may be relevant ones not retrieved. (c) Recall is 1; all the relevant records are retrieved but there may also be irrelevant records that are retrieved.

query, for example, by using some keywords, and a system (basically a two-class classifier) returns a number of records. In the database, there are relevant records and for a query, the system may retrieve some of them (true positives) but probably not all (false negatives); it may also wrongly retrieve records that are not relevant (false positives). The set of relevant and retrieved records can be visualized using a Venn diagram, as shown in figure 19.4a. *Precision* is the number of retrieved and relevant records divided by the total number of retrieved records; if precision is 1, all the retrieved records may be relevant but there may still be records that are relevant but not retrieved. *Recall* is the number of retrieved relevant records divided by the total number of relevant records; even if recall is 1, all the relevant records may be retrieved but there may also be irrelevant records that are retrieved, as shown in figure19.4c. As in the ROC curve, for different threshold values, one can draw a curve for precision vs. recall.

PRECISION

RECALL

SENSITIVITY
SPECIFICITY

From another perspective but with the same aim, there are the two measures of *sensitivity* and *specificity*. Sensitivity is the same as tp-rate and recall. Specificity is how well we detect the negatives, which is the number of true negatives divided by the total number of negatives; this is equal to 1 minus the false alarm rate. One can also draw a sensitivity vs. specificity curve using different thresholds.

CLASS CONFUSION MATRIX

In the case of $K > 2$ classes, if we are using 0/1 error, the *class confusion matrix* is a $K \times K$ matrix whose entry $(i, j)$ contains the number of instances that belong to $C_i$ but are assigned to $C_j$. Ideally, all off-diagonals should be 0, for no misclassification. The class confusion matrix allows us to pinpoint what types of misclassification occur, namely, if there are two classes that are frequently confused. Or, one can define $K$ separate two-class problems, each one separating one class from the other $K - 1$.

## 19.8 Interval Estimation

INTERVAL ESTIMATION

Let us now do a quick review of *interval estimation* that we will use in hypothesis testing. A point estimator, for example, the maximum likelihood estimator, specifies a value for a parameter $\theta$. In interval estimation, we specify an interval within which $\theta$ lies with a certain degree of confidence. To obtain such an interval estimator, we make use of the probability distribution of the point estimator.

For example, let us say we are trying to estimate the mean $\mu$ of a normal density from a sample $\mathcal{X} = \{x^t\}_{t=1}^N$. $m = \sum_t x^t/N$ is the sample average and is the point estimator to the mean. $m$ is the sum of normals and therefore is also normal, $m \sim \mathcal{N}(\mu, \sigma^2/N)$. We define the statistic with a *unit normal distribution*:

UNIT NORMAL DISTRIBUTION

$$\frac{(m-\mu)}{\sigma/\sqrt{N}} \sim \mathcal{Z} \tag{19.1}$$

We know that 95 percent of $\mathcal{Z}$ lies in $(-1.96, 1.96)$, namely, $P\{-1.96 < \mathcal{Z} < 1.96\} = 0.95$, and we can write (see figure 19.5)

$$P\left\{-1.96 < \sqrt{N}\frac{(m-\mu)}{\sigma} < 1.96\right\} = 0.95$$

or equivalently

$$P\left\{m - 1.96\frac{\sigma}{\sqrt{N}} < \mu < m + 1.96\frac{\sigma}{\sqrt{N}}\right\} = 0.95$$

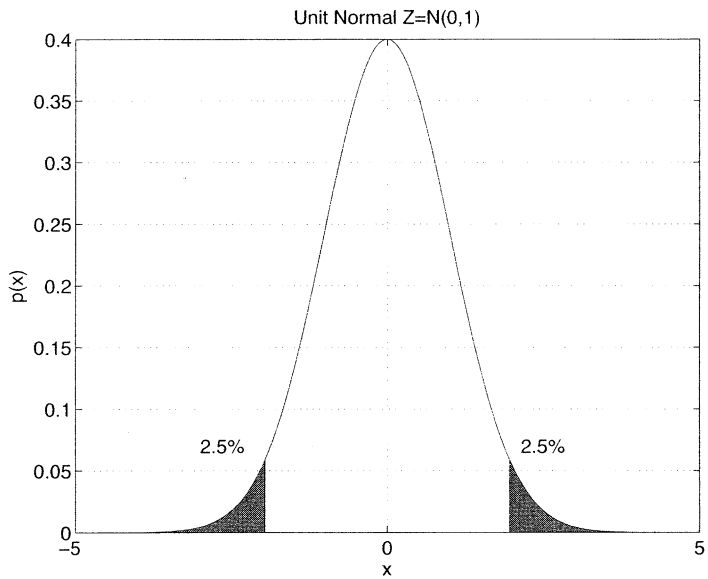



**Figure 19.5** 95 percent of the unit normal distribution lies between −1.96 and 1.96.

TWO-SIDED CONFIDENCE INTERVAL

That is "with 95 percent confidence," $\mu$ will lie within $1.96\sigma/\sqrt{N}$ units of the sample average. This is a *two-sided confidence interval.* With 99 percent confidence, $\mu$ will lie in $(m - 2.58\sigma/\sqrt{N}, m + 2.58\sigma/\sqrt{N})$; that is, if we want more confidence, the interval gets larger. The interval gets smaller as $N$, the sample size, increases.

This can be generalized for any required confidence as follows. Let us denote $z_\alpha$ such that

$$P\{Z > z_\alpha\} = \alpha, \ 0 < \alpha < 1$$

Because $Z$ is symmetric around the mean, $z_{1-\alpha/2} = -z_{\alpha/2}$, and $P\{X < -z_{\alpha/2}\} = P\{X > z_{\alpha/2}\} = \alpha/2$. Hence for any specified level of confidence $1 - \alpha$, we have

$$P\{-z_{\alpha/2} < Z < z_{\alpha/2}\} = 1 - \alpha$$

and

$$P\left\{-z_{\alpha/2} < \sqrt{N}\frac{(m-\mu)}{\sigma} < z_{\alpha/2}\right\} = 1 - \alpha$$

or

$$P\left\{m - z_{\alpha/2}\frac{\sigma}{\sqrt{N}} < \mu < m + z_{\alpha/2}\frac{\sigma}{\sqrt{N}}\right\} = 1 - \alpha \tag{19.2}$$

Hence a $100(1 - \alpha)$ percent two-sided confidence interval for $\mu$ can be computed for any $\alpha$.

Similarly, knowing that $P\{Z < 1.64\} = 0.95$, we have (see figure 19.6)

$$P\left\{\sqrt{N}\frac{(m - \mu)}{\sigma} < 1.64\right\} = 0.95$$

or

$$P\left\{m - 1.64\frac{\sigma}{\sqrt{N}} < \mu\right\} = 0.95$$

ONE-SIDED CONFIDENCE INTERVAL

and $(m - 1.64\sigma/\sqrt{N}, \infty)$ is a 95 percent *one-sided upper confidence interval* for $\mu$, which defines a lower bound. Generalizing, a $100(1 - \alpha)$ percent one-sided confidence interval for $\mu$ can be computed from

$$P\left\{m - z_{\alpha}\frac{\sigma}{\sqrt{N}} < \mu\right\} = 1 - \alpha \tag{19.3}$$

Similarly, the one-sided lower confidence interval that defines an upper bound can also be calculated.

In the previous intervals, we used $\sigma$; that is, we assumed that the variance is known. If it is not, one can plug the sample variance

$$S^2 = \sum_t (x^t - m)^2/(N - 1)$$

instead of $\sigma^2$. We know that when $x^t \sim \mathcal{N}(\mu, \sigma^2)$, $(N - 1)S^2/\sigma^2$ is chi-square with $N - 1$ degrees of freedom. We also know that $m$ and $S^2$ are independent. Then, $\sqrt{N}(m - \mu)/S$ is $t$-distributed with $N - 1$ degrees of freedom (section A.3.7), denoted as

$$\frac{\sqrt{N}(m - \mu)}{S} \sim t_{N-1} \tag{19.4}$$

Hence for any $\alpha \in (0, 1/2)$, we can define an interval, using the values specified by the t *distribution*, instead of the unit normal $Z$

*t* DISTRIBUTION

$$P\left\{t_{1-\alpha/2,N-1} < \sqrt{N}\frac{(m - \mu)}{S} < t_{\alpha/2,N-1}\right\} = 1 - \alpha$$

or using $t_{1-\alpha/2,N-1} = -t_{\alpha/2,N-1}$, we can write

$$P\left\{m - t_{\alpha/2,N-1}\frac{S}{\sqrt{N}} < \mu < m + t_{\alpha/2,N-1}\frac{S}{\sqrt{N}}\right\} = 1 - \alpha$$

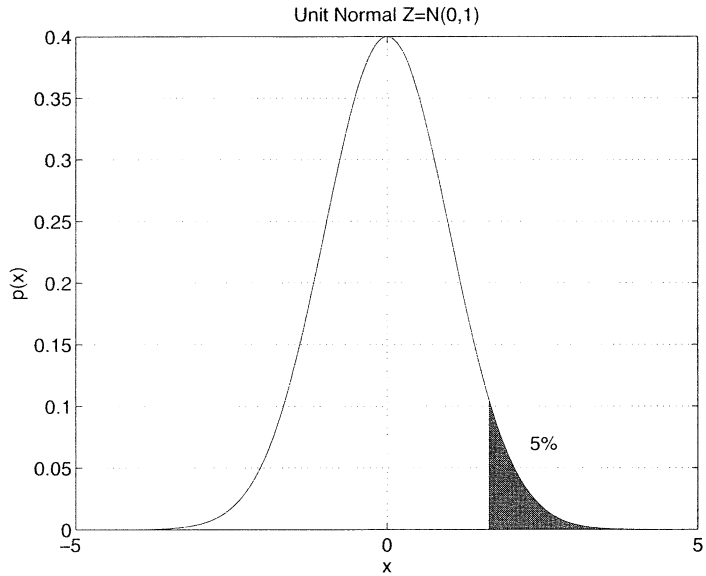



**Figure 19.6** 95 percent of the unit normal distribution lies before 1.64.

Similarly, one-sided confidence intervals can be defined. The $t$ distribution has larger spread (longer tails) than the unit normal distribution, and generally the interval given by the $t$ is larger; this should be expected since additional uncertainty exists due to the unknown variance.

## 19.9 Hypothesis Testing

Instead of explicitly estimating some parameters, in certain applications we may want to use the sample to test some particular hypothesis concerning the parameters. For example, instead of estimating the mean, we may want to test whether the mean is less than 0.02. If the random sample is consistent with the hypothesis under consideration, we "fail to reject" the hypothesis; otherwise, we say that it is "rejected." But when we make such a decision, we are not really saying that it is true or false but rather that the sample data appears to be consistent with it to a given degree of confidence or not.

HYPOTHESIS TESTING

In *hypothesis testing*, the approach is as follows. We define a statistic

**Table 19.3** Type I error, type II error, and power of a test.

| | Decision | |
|---|---|---|
| Truth | Fail to reject | Reject |
| True | Correct | Type I error |
| False | Type II error | Correct (power) |

that obeys a certain distribution if the hypothesis is correct. If the statistic calculated from the sample has very low probability of being drawn from this distribution, then we reject the hypothesis; otherwise, we fail to reject it.

Let us say we have a sample from a normal distribution with unknown mean $\mu$ and known variance $\sigma^2$, and we want to test a specific hypothesis about $\mu$, for example, whether it is equal to a specified constant $\mu_0$. It is denoted as $H_0$ and is called the *null hypothesis*

NULL HYPOTHESIS

$$H_0 : \mu = \mu_0$$

against the alternative hypothesis

$$H_1 : \mu \neq \mu_0$$

$m$ is the point estimate of $\mu$, and it is reasonable to reject $H_0$ if $m$ is too far from $\mu_0$. This is where the interval estimate is used. We fail to reject the hypothesis with *level of significance* $\alpha$ if $\mu_0$ lies in the $100(1-\alpha)$ percent confidence interval, namely, if

LEVEL OF SIGNIFICANCE

$$\frac{\sqrt{N}(m-\mu_0)}{\sigma} \in (-z_{\alpha/2}, z_{\alpha/2}) \tag{19.5}$$

We reject the null hypothesis if it falls outside, on either side. This is a *two-sided test.*

TWO-SIDED TEST

If we reject when the hypothesis is correct, this is a *type I error* and thus $\alpha$, set before the test, defines how much type I error we can tolerate, typical values being $\alpha = 0.1, 0.05, 0.01$ (see table 19.3). A *type II error* is if we fail to reject the null hypothesis when the true mean $\mu$ is unequal to $\mu_0$. The probability that $H_0$ is not rejected when the true mean is $\mu$ is a function of $\mu$ and is given as

TYPE I ERROR

TYPE II ERROR

$$\beta(\mu) = P_\mu\left\{-z_{\alpha/2} \leq \frac{m-\mu_0}{\sigma/\sqrt{N}} \leq z_{\alpha/2}\right\} \tag{19.6}$$

POWER FUNCTION

$1 - \beta(\mu)$ is called the *power function* of the test and is equal to the probability of rejection when $\mu$ is the true value. Type II error probability increases as $\mu$ and $\mu_0$ gets closer, and we can calculate how large a sample we need for us to be able to detect a difference $\delta = |\mu - \mu_0|$ with sufficient power.

ONE-SIDED TEST

One can also have a *one-sided test* of the form

$$H_0 : \mu \leq \mu_0 \text{ vs } H_1 : \mu > \mu_0$$

as opposed to the two-sided test when the alternative hypothesis is $\mu \neq \mu_0$. The one-sided test with $\alpha$ level of significance defines the $100(1 - \alpha)$ confidence interval bounded on one side in which $m$ should lie for the hypothesis not to be rejected. We fail to reject if

$$\frac{\sqrt{N}}{\sigma}(m - \mu_0) \in (-\infty, z_\alpha) \tag{19.7}$$

and reject outside. Note that the null hypothesis $H_0$ also allows equality, which means that we get ordering information only if the test rejects. This tells us which of the two one-sided tests we should use. Whatever claim we have should be in $H_1$ so that rejection of the test would support our claim.

If the variance is unknown, just as we did in the interval estimates, we use the sample variance instead of the population variance and the fact that

$$\frac{\sqrt{N}(m - \mu_0)}{S} \sim t_{N-1} \tag{19.8}$$

For example, for $H_0 : \mu = \mu_0$ vs $H_1 : \mu \neq \mu_0$, we fail to reject at significance level $\alpha$ if

$$\frac{\sqrt{N}(m - \mu_0)}{S} \in (-t_{\alpha/2,N-1}, t_{\alpha/2,N-1}) \tag{19.9}$$

$t$ TEST

which is known as the *two-sided* t *test.* A one-sided $t$ test can be defined similarly.

## 19.10 Assessing a Classification Algorithm's Performance

Now that we have reviewed hypothesis testing, we are ready to see how it is used in testing error rates. We will discuss the case of classification error, but the same methodology applies for squared error in regression, log likelihoods in unsupervised learning, expected reward in

reinforcement learning, and so on, as long as we can write the appropriate parametric form for the sampling distribution. We will also discuss nonparametric tests when no such parametric form can be found.

We now start with error rate assessment, and, in the next section, we discuss error rate comparison.

### 19.10.1 Binomial Test

Let us start with the case where we have a single training set $\mathcal{T}$ and a single validation set $\mathcal{V}$. We train our classifier on $\mathcal{T}$ and test it on $\mathcal{V}$. We denote by $p$ the probability that the classifier makes a misclassification error. We do not know $p$; it is what we would like to estimate or test a hypothesis about. On the instance with index $t$ from the validation set $\mathcal{V}$, let us say $x^t$ denotes the correctness of the classifier's decision: $x^t$ is a 0/1 Bernoulli random variable that takes the value 1 when the classifier commits an error and 0 when the classifier is correct. The binomial random variable $X$ denotes the total number of errors:

$$X = \sum_{t=1}^{N} x^t$$

We would like to test whether the error probability $p$ is less than or equal to some value $p_0$ we specify:

$$H_0 : p \leq p_0 \text{ vs. } H_1 : p > p_0$$

If the probability of error is $p$, the probability that the classifier commits $j$ errors out of $N$ is

$$P\{X = j\} = \binom{N}{j} p^j (1-p)^{N-j}$$

BINOMIAL TEST

It is reasonable to reject $p \leq p_0$ if in such a case, the probability that we see $X = e$ errors or more is very unlikely. That is, the *binomial test* rejects the hypothesis if

$$P\{X \geq e\} = \sum_{x=e}^{N} \binom{N}{x} p_0^x (1-p_0)^{N-x} < \alpha \tag{19.10}$$

where $\alpha$ is the significance, for example, 0.05.

### 19.10.2 Approximate Normal Test

If $p$ is the probability of error, our point estimate is $\hat{p} = X/N$. Then, it is reasonable to reject the null hypothesis if $\hat{p}$ is much larger than $p_0$. How large is large enough is given by the sampling distribution of $\hat{p}$ and the significance $\alpha$.

Because $X$ is the sum of independent random variables from the same distribution, the central limit theorem states that for large $N$, $X/N$ is approximately normal with mean $p_0$ and variance $p_0(1 - p_0)$. Then

$$\frac{X/N - p_0}{\sqrt{p_0(1 - p_0)}} \dot{\sim} Z \tag{19.11}$$

where $\dot{\sim}$ denotes "approximately distributed." Then, using equation 19.7, the *approximate normal test* rejects the null hypothesis if this value for $X = e$ is greater than $z_\alpha$. $z_{0.05}$ is 1.64. This approximation will work well as long as $N$ is not too small and $p$ is not very close to 0 or 1; as a rule of thumb, we require $Np \geq 5$ and $N(1 - p) \geq 5$.

APPROXIMATE NORMAL TEST

### 19.10.3 $t$ Test

The two tests we discussed earlier use a single validation set. If we run the algorithm $K$ times, on $K$ training/validation set pairs, we get $K$ error percentages, $p_i, i = 1, \ldots, K$ on the $K$ validation sets. Let $x_i^t$ be 1 if the classifier trained on $\mathcal{T}_i$ makes a misclassification error on instance $t$ of $\mathcal{V}_i$; $x_i^t$ is 0 otherwise. Then

$$p_i = \frac{\sum_{t=1}^{N} x_i^t}{N}$$

Given that

$$m = \frac{\sum_{i=1}^{K} p_i}{K}, \quad S^2 = \frac{\sum_{i=1}^{K} (p_i - m)^2}{K - 1}$$

from equation 19.8, we know that we have

$$\frac{\sqrt{K}(m - p_0)}{S} \sim t_{K-1} \tag{19.12}$$

and the $t$ test rejects the null hypothesis that the classification algorithm has $p_0$ or less error percentage at significance level $\alpha$ if this value is greater than $t_{\alpha,K-1}$. Typically, $K$ is taken as 10 or 30. $t_{0.05,9} = 1.83$ and $t_{0.05,29} = 1.70$.

## 19.11 Comparing Two Classification Algorithms

Given two learning algorithms, we want to compare and test whether they construct classifiers that have the same expected error rate.

### 19.11.1 McNemar's Test

CONTINGENCY TABLE

Given a training set and a validation set, we use two algorithms to train two classifiers on the training set and test them on the validation set and compute their errors. A *contingency table*, like the one shown here, is an array of natural numbers in matrix form representing counts, or frequencies:

| | |
|---|---|
| $e_{00}$: Number of examples misclassified by both | $e_{01}$: Number of examples misclassified by 1 but not 2 |
| $e_{10}$: Number of examples misclassified by 2 but not 1 | $e_{11}$: Number of examples correctly classified by both |

Under the null hypothesis that the classification algorithms have the same error rate, we expect $e_{01} = e_{10}$ and these to be equal to $(e_{01}+e_{10})/2$. We have the chi-square statistic with one degree of freedom

$$\frac{(|e_{01} - e_{10}| - 1)^2}{e_{01} + e_{10}} \sim \chi_1^2 \tag{19.13}$$

MCNEMAR'S TEST

and *McNemar's test* rejects the hypothesis that the two classification algorithms have the same error rate at significance level $\alpha$ if this value is greater than $\chi^2_{\alpha,1}$. For $\alpha = 0.05$, $\chi^2_{0.05,1} = 3.84$.

### 19.11.2 *K*-Fold Cross-Validated Paired *t* Test

This set uses $K$-fold cross-validation to get $K$ training/validation set pairs. We use the two classification algorithms to train on the training sets $\mathcal{T}_i, i = 1, \ldots, K$, and test on the validation sets $\mathcal{V}_i$. The error percentages of the classifiers on the validation sets are recorded as $p_i^1$ and $p_i^2$.

If the two classification algorithms have the same error rate, then we expect them to have the same mean, or equivalently, that the difference of their means is 0. The difference in error rates on fold $i$ is $p_i = p_i^1 - p_i^2$. This is a *paired test*; that is, for each $i$, both algorithms see the same training and validation sets. When this is done $K$ times, we have a distribution of $p_i$ containing $K$ points. Given that $p_i^1$ and $p_i^2$ are both (approximately)

PAIRED TEST

normal, their difference $p_i$ is also normal. The null hypothesis is that this distribution has 0 mean:

$$H_0 : \mu = 0 \text{ vs. } H_1 : \mu \neq 0$$

We define

$$m = \frac{\sum_{i=1}^{K} p_i}{K}, \quad S^2 = \frac{\sum_{i=1}^{K} (p_i - m)^2}{K - 1}$$

Under the null hypothesis that $\mu = 0$, we have a statistic that is $t$-distributed with $K - 1$ degrees of freedom:

$$\frac{\sqrt{K}(m - 0)}{S} = \frac{\sqrt{K} \cdot m}{S} \sim t_{K-1} \tag{19.14}$$

K-FOLD CV PAIRED $t$ TEST

Thus the K-*fold cv paired* t *test* rejects the hypothesis that two classification algorithms have the same error rate at significance level $\alpha$ if this value is outside the interval $(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1})$. $t_{0.025,9} = 2.26$ and $t_{0.025,29} = 2.05$.

If we want to test whether the first algorithm has less error than the second, we need a one-sided hypothesis and use a one-tailed test:

$$H_0 : \mu \geq 0 \text{ vs. } H_1 : \mu < 0$$

If the test rejects, our claim that the first one has significantly less error is supported.

### 19.11.3 5 × 2 cv Paired $t$ Test

In the 5 × 2 cv $t$ test, proposed by Dietterich (1998), we perform five replications of twofold cross-validation. In each replication, the dataset is divided into two equal-sized sets. $p_i^{(j)}$ is the difference between the error rates of the two classifiers on fold $j = 1, 2$ of replication $i = 1, \ldots, 5$. The average on replication $i$ is $\overline{p}_i = (p_i^{(1)} + p_i^{(2)})/2$, and the estimated variance is $s_i^2 = (p_i^{(1)} - \overline{p}_i)^2 + (p_i^{(2)} - \overline{p}_i)^2$.

Under the null hypothesis that the two classification algorithms have the same error rate, $p_i^{(j)}$ is the difference of two identically distributed proportions, and ignoring the fact that these proportions are not independent, $p_i^{(j)}$ can be treated as approximately normal distributed with 0 mean and unknown variance $\sigma^2$. Then $p_i^{(j)}/\sigma$ is approximately unit normal. If we assume $p_i^{(1)}$ and $p_i^{(2)}$ are independent normals (which is not strictly true because their training and test sets are not drawn independently of each other), then $s_i^2/\sigma^2$ has a chi-square distribution with

one degree of freedom. If each of the $s_i^2$ are assumed to be independent (which is not true because they are all computed from the same set of available data), then their sum is chi-square with five degrees of freedom:

$$M = \frac{\sum_{i=1}^{5} s_i^2}{\sigma^2} \sim \chi_5^2$$

and

$$t = \frac{p_1^{(1)}/\sigma}{\sqrt{M/5}} = \frac{p_1^{(1)}}{\sqrt{\sum_{i=1}^{5} s_i^2/5}} \sim t_5 \tag{19.15}$$

5 × 2 CV PAIRED $t$ TEST

giving us a $t$ statistic with five degrees of freedom. The $5 \times 2$ *cv paired* t *test* rejects the hypothesis that the two classification algorithms have the same error rate at significance level $\alpha$ if this value is outside the interval $(-t_{\alpha/2,5}, t_{\alpha/2,5})$. $t_{0.025,5} = 2.57$.

### 19.11.4 5 × 2 cv Paired $F$ Test

We note that the numerator in equation 19.15, $p_1^{(1)}$, is arbitrary; actually, ten different values can be placed in the numerator, namely, $p_i^{(j)}, j = 1, 2, i = 1, \ldots, 5$, leading to ten possible statistics:

$$t_i^{(j)} = \frac{p_i^{(j)}}{\sqrt{\sum_{i=1}^{5} s_i^2/5}} \tag{19.16}$$

Alpaydın (1999) proposed an extension to the $5 \times 2$ cv $t$ test that combines the results of the ten possible statistics. If $p_i^{(j)}/\sigma \sim \mathcal{Z}$, then $\left(p_i^{(j)}\right)^2/\sigma^2 \sim \chi_1^2$ and their sum is chi-square with ten degrees of freedom:

$$N = \frac{\sum_{i=1}^{5}\sum_{j=1}^{2}\left(p_i^{(j)}\right)^2}{\sigma^2} \sim \chi_{10}^2$$

Placing this in the numerator of equation 19.15, we get a statistic that is the ratio of two chi-square distributed random variables. Two such variables divided by their respective degrees of freedom is $F$-distributed with ten and five degrees of freedom (section A.3.8):

$$f = \frac{N/10}{M/5} = \frac{\sum_{i=1}^{5}\sum_{j=1}^{2}\left(p_i^{(j)}\right)^2}{2\sum_{i=1}^{5} s_i^2} \sim F_{10,5} \tag{19.17}$$

5 × 2 CV PAIRED $F$ TEST

$5 \times 2$ *cv paired* F *test* rejects the hypothesis that the classification algorithms have the same error rate at significance level $\alpha$ if this value is greater than $F_{\alpha,10,5}$. $F_{0.05,10,5} = 4.74$.

## 19.12 Comparing Multiple Algorithms: Analysis of Variance

In many cases, we have more than two algorithms, and we would like to compare their expected error. Given $L$ algorithms, we train them on $K$ training sets, induce $K$ classifiers with each algorithm, and then test them on $K$ validation sets and record their error rates. This gives us $L$ groups of $K$ values. The problem then is the comparison of these $L$ samples for statistically significant difference. This is an experiment with a single factor with $L$ levels, the learning algorithms, and there are $K$ replications for each level.

ANALYSIS OF VARIANCE

In *analysis of variance* (ANOVA), we consider $L$ independent samples, each of size $K$, composed of normal random variables of unknown mean $\mu_j$ and unknown common variance $\sigma^2$:

$$X_{ij} \sim \mathcal{N}(\mu_j, \sigma^2), j = 1, \ldots, L, \; i = 1, \ldots, K,$$

We are interested in testing the hypothesis $H_0$ that all means are equal:

$$H_0 : \mu_1 = \mu_2 = \cdots = \mu_L \text{ vs. } H_1 : \mu_r \neq \mu_s, \text{for at least one pair } (r, s)$$

The comparison of error rates of multiple classification algorithms fits this scheme. We have $L$ classification algorithms, and we have their error rates on $K$ validation folds. $X_{ij}$ is the number of validation errors made by the classifier, which is trained by classification algorithm $j$ on fold $i$. Each $X_{ij}$ is binomial and approximately normal. If $H_0$ is not rejected, we fail to find a significant error difference among the error rates of the $L$ classification algorithms. This is therefore a generalization of the tests we saw in section 19.11 that compared the error rates of two classification algorithms. The $L$ classification algorithms may be different or may use different hyperparameters, for example, number of hidden units in a multilayer perceptron, number of neighbors in $k$-nn, and so forth.

The approach in ANOVA is to derive two estimators of $\sigma^2$. One estimator is designed such that it is true only when $H_0$ is true, and the second is always a valid estimator, regardless of whether $H_0$ is true or not. ANOVA then rejects $H_0$, namely, that the $L$ samples are drawn from the same population, if the two estimators differ significantly.

Our first estimator to $\sigma^2$ is valid only if the hypothesis is true, namely, $\mu_j = \mu, j = 1, \ldots, L$. If $X_{ij} \sim \mathcal{N}(\mu, \sigma^2)$, then the group average

$$m_j = \sum_{i=1}^{K} \frac{X_{ij}}{K}$$

is also normal with mean $\mu$ and variance $\sigma^2/K$. If the hypothesis is true, then $m_j, j = 1, \ldots, L$ are $L$ instances drawn from $\mathcal{N}(\mu, \sigma^2/K)$. Then *their* mean and variance are

$$m = \frac{\sum_{j=1}^{L} m_j}{L}, \quad S^2 = \frac{\sum_j (m_j - m)^2}{L - 1}$$

Thus an estimator of $\sigma^2$ is $K \cdot S^2$, namely,

$$\hat{\sigma}_b^2 = K \sum_{j=1}^{L} \frac{(m_j - m)^2}{L - 1} \tag{19.18}$$

Each of $m_j$ is normal and $(L-1)S^2/(\sigma^2/K)$ is chi-square with $(L-1)$ degrees of freedom. Then, we have

$$\sum_j \frac{(m_j - m)^2}{\sigma^2/K} \sim \chi^2_{L-1} \tag{19.19}$$

We define $SS_b$, the between-group sum of squares, as

$$SS_b \equiv K \sum_j (m_j - m)^2$$

So, when $H_0$ is true, we have

$$\frac{SS_b}{\sigma^2} \sim \chi^2_{L-1} \tag{19.20}$$

Our second estimator of $\sigma^2$ is the average of group variances, $S_j^2$, defined as

$$S_j^2 = \frac{\sum_{i=1}^{K} (X_{ij} - m_j)^2}{K - 1}$$

and their average is

$$\hat{\sigma}_w^2 = \sum_{j=1}^{L} \frac{S_j^2}{L} = \sum_j \sum_i \frac{(X_{ij} - m_j)^2}{L(K-1)} \tag{19.21}$$

We define $SS_w$, the within-group sum of squares:

$$SS_w \equiv \sum_j \sum_i (X_{ij} - m_j)^2$$

Remembering that for a normal sample, we have

$$(K-1)\frac{S_j^2}{\sigma^2} \sim \chi^2_{K-1}$$

and that the sum of chi-squares is also a chi-square, we have

$$(K-1)\sum_{j=1}^{L}\frac{S_j^2}{\sigma^2} \sim \chi^2_{L(K-1)}$$

So

$$\frac{SS_w}{\sigma^2} \sim \chi^2_{L(K-1)} \tag{19.22}$$

Then we have the task of comparing two variances for equality, which we can do by checking whether their ratio is close to 1. The ratio of two independent chi-square random variables divided by their respective degrees of freedom is a random variable that is $F$-distributed, and hence when $H_0$ is true, we have

$$F_0 = \left(\frac{SS_b/\sigma^2}{L-1}\right) \Big/ \left(\frac{SS_w/\sigma^2}{L(K-1)}\right) = \frac{SS_b/(L-1)}{SS_w/(L(K-1))} = \frac{\hat{\sigma}_b^2}{\hat{\sigma}_w^2} \sim F_{L-1,L(K-1)} \tag{19.23}$$

For any given significance value $\alpha$, the hypothesis that the $L$ classification algorithms have the same expected error rate is rejected if this statistic is greater than $F_{\alpha,L-1,L(K-1)}$.

Note that we are rejecting if the two estimators disagree significantly. If $H_0$ is not true, then the variance of $m_j$ around $m$ will be larger than what we would normally have if $H_0$ were true, and hence if $H_0$ is not true, the first estimator $\hat{\sigma}_b^2$ will overestimate $\sigma^2$, and the ratio will be greater than 1. For $\alpha = 0.05$, $L = 5$ and $K = 10$, $F_{0.05,4,45} = 2.6$. If $X_{ij}$ vary around $m$ with a variance of $\sigma^2$, then if $H_0$ is true, $m_j$ vary around $m$ by $\sigma^2/K$. If it seems as if they vary more, then $H_0$ should be rejected because the displacement of $m_j$ around $m$ is more than what can be explained by some constant added noise.

The name *analysis of variance* is derived from a partitioning of the total variability in the data into its components.

$$SS_T \equiv \sum_j \sum_i (X_{ij} - m)^2 \tag{19.24}$$

$SS_T$ divided by its degree of freedom, namely, $K \cdot L - 1$ (there are $K \cdot L$ data points, and we lose one degree of freedom because $m$ is fixed), gives us the sample variance of $X_{ij}$. It can be shown that (exercise 5) the total sum of squares can be split into between-group sum of squares and within-group sum of squares

$$SS_T = SS_b + SS_w \tag{19.25}$$

**Table 19.4** The analysis of variance (ANOVA) table for a single factor model.

| Source of variation | Sum of squares | Degrees of freedom | Mean square | $F_0$ |
|---|---|---|---|---|
| Between groups | $SS_b \equiv K\sum_j (m_j - m)^2$ | $L-1$ | $MS_b = \frac{SS_b}{L-1}$ | $\frac{MS_b}{MS_w}$ |
| Within groups | $SS_w \equiv \sum_j \sum_i (X_{ij} - m_j)^2$ | $L(K-1)$ | $MS_w = \frac{SS_w}{L(K-1)}$ | |
| Total | $SS_T \equiv \sum_j \sum_i (X_{ij} - m)^2$ | $L \cdot K - 1$ | | |

Results of ANOVA are reported in an ANOVA table as shown in table 19.4. This is the basic *one-way* analysis of variance where there is a single factor, for example, learning algorithm. We may consider experiments with multiple factors, for example, we can have one factor for classification algorithms and another factor for feature extraction algorithms used before, and this will be a *two-factor experiment with interaction.*

POSTHOC TESTING

If the hypothesis is rejected, we only know that there is some difference between the $L$ groups but we do not know where. For this, we do *posthoc testing*, that is, an additional set of tests involving subsets of groups, for example, pairs.

LEAST SQUARE DIFFERENCE TEST

Fisher's *least square difference test* (LSD) compares groups in a pairwise manner. For each group, we have $m_i \sim \mathcal{N}(\mu_i, \sigma_w^2 = MS_w/K)$ and $m_i - m_j \sim \mathcal{N}(\mu_i - \mu_j, 2\sigma_w^2)$. Then, under the null hypothesis that $H_0 : \mu_i = \mu_j$, we have

$$t = \frac{m_i - m_j}{\sqrt{2}\sigma_w} \sim t_{L(K-1)}$$

We reject $H_0$ in favor of the alternative hypothesis $H_1 : \mu_1 \neq \mu_2$ if $|t| > t_{\alpha/2,L(K-1)}$. Similarly, one-sided tests can be defined to find pairwise orderings.

MULTIPLE COMPARISONS

When we do a number of tests to draw one conclusion, this is called *multiple comparisons*, and we need to keep in mind that if $T$ hypotheses are to be tested, each at significance level $\alpha$, then the probability that at least one hypothesis is incorrectly rejected is at most $T\alpha$. For example,

the probability that six confidence intervals, each calculated at 95 percent individual confidence intervals, will simultaneously be correct is at least 70 percent. Thus to ensure that the overall confidence interval is at least $100(1 - \alpha)$, each confidence interval should be set at $100(1 - \alpha/T)$. This is called a *Bonferroni correction.*

BONFERRONI CORRECTION

Sometimes it may be the case that ANOVA rejects and none of the posthoc pairwise tests find a significant difference. In such a case, our conclusion is that there is a difference between the means but that we need more data to be able to pinpoint the source of the difference.

Note that the main cost is the training and testing of $L$ classification algorithms on $K$ training/validation sets. Once this is done and the values are stored in a $K \times L$ table, calculating the ANOVA or pairwise comparison test statistics from those is very cheap in comparison.

## 19.13 Comparison over Multiple Datasets

Let us say we want to compare two or more algorithms on several datasets and not one. What makes this different is that an algorithm depending on how well its inductive bias matches the problem will behave differently on different datasets, and these error values on different datasets cannot be said to be normally distributed around some mean accuracy. This implies that the parametric tests that we discussed in the previous sections based on binomials being approximately normal are no longer applicable and we need to resort to *nonparametric tests.* The advantage of having such tests is that we can also use them for comparing other statistics that are not normal, for example, training times, number of free parameters, and so on.

NONPARAMETRIC TESTS

Parametric tests are generally robust to slight departures from normality, especially if the sample is large. Nonparametric tests are distribution free but are less efficient; that is, if both are applicable, a parametric test should be preferred. The corresponding nonparametric test will require a larger sample to achieve the same power. Nonparametric tests assume no knowledge about the distribution of the underlying population but only that the values can be compared or ordered, and, as we will see, such tests make use of this order information.

When we have an algorithm trained on a number of different datasets, the average of its errors on these datasets is not a meaningful value, and, for example, we cannot use such averages to compare two algorithms $A$

and $B$. To compare two algorithms, the only piece of information we can use is if on any dataset, $A$ is more accurate than $B$; we can then count the number of times $A$ is more accurate than $B$ and check whether this could have been by chance if they indeed were equally accurate. With more than two algorithms, we will look at the average *ranks* of the learners trained by different algorithms. Nonparametric tests basically use this rank data and not the absolute values.

Before proceeding with the details of these tests, it should be stressed that it does not make sense to compare error rates of algorithms on a whole variety of applications. Because there is no such thing as the "best learning algorithm," such tests would not be conclusive. However, we can compare algorithms on a number of datasets, or versions, of the same application. For example, we may have a number of different datasets for face recognition but with different properties (resolution, lighting, number of subjects, and so on), and we may use a nonparametric test to compare algorithms on those; different properties of the datasets would make it impossible for us to lump images from different datasets together in a single set, but we can train algorithms separately on different datasets, obtain ranks separately, and then combine these to get an overall decision.

### 19.13.1 Comparing Two Algorithms

Let us say we want to compare two algorithms. We both train and validate them on $i = 1, \ldots, N$ different datasets in a paired manner—that is, all the conditions except the different algorithms should be identical. We get results $e_i^1$ and $e_i^2$ and if we use $K$-fold cross-validation on each dataset, these are averages or medians of the $K$ values. The *sign test* is based on the idea that if the two algorithms have equal error, on each dataset, there should be $1/2$ probability that the first has less error than the second, and thus we expect the first to win on $N/2$ datasets. Let us define

SIGN TEST

$$X_i = \begin{cases} 1 & \text{if } e_i^1 < e_i^2 \\ 0 & \text{otherwise} \end{cases} \quad \text{and } X = \sum_{i=1}^{N} X_i$$

Let us say we want to test

$$H_0 : \mu_1 \geq \mu_2 \text{ vs. } H_1 : \mu_1 < \mu_2$$

If the null hypothesis is correct, $X$ is binomial in $N$ trials with $p = 1/2$. Let us say that we saw that the first one wins on $X = e$ datasets. Then, the probability that we have $e$ or less wins when indeed $p = 1/2$ is

$$P\{X \leq e\} = \sum_{x=0}^{e} \binom{N}{x} \left(\frac{1}{2}\right)^{x} \left(\frac{1}{2}\right)^{N-x}$$

and we reject if this probability is too small, that is, less than $\alpha$. If there are ties, we divide them equally to both sides; that is, if there are $t$ ties, we add $t/2$ to $e$ (if $t$ is odd, we ignore the odd one and decrease $N$ by 1).

In testing

$$H_0 : \mu_1 \leq \mu_2 \text{ vs. } H_1 : \mu_1 > \mu_2$$

we reject if $P\{X \geq e\} < \alpha$.

For the two-sided test

$$H_0 : \mu_1 = \mu_2 \text{ vs. } H_1 : \mu_1 \neq \mu_2$$

we reject if $e$ is too small or too large. If $e < N/2$, we reject if $2P\{X \leq e\} < \alpha$; if $e > N/2$, we reject if $2P\{X \geq e\} < \alpha$—we need to find the corresponding tail, and we multiply it by 2 because it is a two-tailed test.

As we discussed before, nonparametric tests can be used to compare any measurements, for example, training times. In such a case, we see the advantage of a nonparametric test that uses order rather than averages of absolute values. Let us say we compare two algorithms on ten datasets, nine of which are small and have training times for both algorithms on the order of minutes, and one that is very large and whose training time is on the order of a day. If we use a parametric test and take the average of training times, the single large dataset will dominate the decision, but when we use the nonparameric test and compare values separately on each dataset, using the order will have the effect of normalizing separately for each dataset and hence will help us make a robust decision.

We can also use the sign test as a one sample test, for example, to check if the average error on all datasets is less than two percent, by comparing $\mu_1$ not by the mean of a second population but by a constant $\mu_0$. We can do this simply by plugging the constant $\mu_0$ in place of all observations from a second sample and using the procedure used earlier; that is, we will count how many times we get more or less than 0.02 and check if this is too unlikely under the null hypothesis. For large $N$, normal

approximation to the binomial can be used (exercise 6), but in practice, the number of datasets may be smaller than 20. Note that the sign test is a test on the median of a population, which is equal to the mean if the distribution is symmetric.

The sign test only uses the sign of the difference and not its magnitude, but we may envisage a case where the first algorithm, when it wins, always wins by a large margin whereas the second algorithm, when it wins, always wins barely. The *Wilcoxon signed rank test* uses both the sign and the magniture of differences, as follows:

WILCOXON SIGNED RANK TEST

Let us say, additional to the sign of differences, we also calculate $m_i = |e_i^1 - e_i^2|$ and then we order them so that the smallest, $\min_i m_i$, is assigned rank 1, the next smallest is assigned rank 2, and so on. If there are ties, their ranks are given the average value that they would receive if they differed slightly. For example, if the magnitudes are $2, 1, 2, 4$, the ranks are $2.5, 1, 2.5, 4$. We then calculate $w_+$ as the sum of all ranks whose signs are positive and $w_-$ as the sum of all ranks whose signs are negative.

The null hypothesis $\mu_1 \leq \mu_2$ can be rejected in favor of the alternative $\mu_1 > \mu_2$ only if $w_+$ is much smaller than $w_-$. Similarly, the two-sided hypothesis $\mu_1 = \mu_2$ can be rejected in favor of the alternative $\mu_1 \neq \mu_2$ only if either $w_+$ or $w_-$, that is, $w = \min(w_+, w_-)$, is very small. The critical values for the Wilcoxon signed rank test are tabulated and for $N > 20$, normal approximations can be used.

### 19.13.2 Multiple Algorithms

KRUSKAL-WALLIS TEST

The *Kruskal-Wallis test* is the nonparametric version of ANOVA and is a multiple sample generalization of a rank test. Given the $M = L \cdot N$ observations, for example, error rates, of $L$ algorithms on $N$ datasets, $X_{ij}, i = 1, \ldots, L, j = 1, \ldots, N$, we rank them from the smallest to the largest and assign them ranks, $R_{ij}$, between 1 and $M$, again taking averages in case of ties. If the null hypothesis

$$H_0 : \mu_1 = \mu_2 = \cdots = \mu_L$$

is true, then the average of ranks of algorithm $i$ should be approximately halfway between 1 and $M$, that is, $(M + 1)/2$. We denote the sample average rank of algorithm $i$ by $\overline{R}_{i\bullet}$ and we reject the hypothesis if the average ranks seem to differ from halfway. The test statistic

$$H = \frac{12}{(M+1)L} \sum_{i=1}^{L} \left( \overline{R}_{i\bullet} - \frac{M+1}{2} \right)$$

is approximately chi-square distributed with $L - 1$ degrees of freedom and we reject the null hypothesis if the statistic exceeds $\chi_{\alpha,L-1}$.

Just like the parametric ANOVA, if the null hypothesis is rejected, we can do posthoc testing to check for pairwise comparison of ranks. One method for this is *Tukey's test*, which makes use of the *studentized range statistic*

TUKEY'S TEST

$$q = \frac{\overline{R}_{max} - \overline{R}_{min}}{\sigma_w}$$

where $\overline{R}_{max}$ and $\overline{R}_{min}$ are the largest and smallest means (of ranks), respectively, out of the $L$ means, and $\sigma_w^2$ is the average variance of ranks around group rank averages. We reject that groups $i$ and $j$ have the same ranks in favor of the alternative hypothesis that they are different if

$$|\overline{R}_{i\bullet} - \overline{R}_{j\bullet}| > q_\alpha(L, L(K-1))\sigma_w$$

where $q_\alpha(L, L(K-1))$ are tabulated. One-sided tests can also be defined to order algorithms in terms of average rank.

Demsar (2006) proposes to use CD (critical difference) diagrams for visualization. On a scale of 1 to $L$, we mark the averages, $\overline{R}_{i\bullet}$, and draw lines of length given by the critical difference, $q_\alpha(L, L(K-1))\sigma_w$, between groups, so that lines connect groups that are not statistically significantly different.

## 19.14 Notes

The material related to experiment design follows the discussion from (Montgomery 2005), which here is adapted for machine learning. A more detailed discussion of interval estimation, hypothesis testing, and analysis of variance can be found in any introductory statistics book, for example, Ross 1987.

Dietterich (1998) discusses statistical tests and compares them on a number of applications using different classification algorithms. A review of ROC use and AUC calculation is given in Fawcett 2006. Demsar (2006) reviews statistical tests for comparing classifiers over multiple datasets.

When we compare two or more algorithms, if the null hypothesis that they have the same error rate is not rejected, we choose the simpler one, namely, the one with less space or time complexity. That is, we use our prior preference if the data does not prefer one in terms of error rate. For example, if we compare a linear model and a nonlinear model and

if the test does not reject that they have the same expected error rate, we should go for the simpler linear model. Even if the test rejects, in choosing one algorithm over another, error rate is only one of the criteria. Other criteria like training (space/time) complexity, testing complexity, and interpretability may override in practical applications.

This is how the posthoc test results are used in the MultiTest algorithm (Yıldız and Alpaydın 2006) to generate a full ordering. We do $L(L-1)/2$ one-sided pairwise tests to order the $L$ algorithms, but it is very likely that the tests will not give a full ordering but only a partial order. The missing links are filled in using the prior complexity information to get a full order. A topological sort gives an ordering of algorithms using both types of information, error and complexity.

There are also tests to allow checking for *contrasts*. Let us say 1 and 2 are neural network methods and 3 and 4 are fuzzy logic methods. We can then test whether the average of 1 and 2 differs from the average of 3 and 4, thereby allowing us to compare methods in general.

Another important point to note is that we are only assessing or comparing misclassifications. This implies that from our point of view, all misclassifications have the same cost. When this is not the case, our tests should be based on risks taking a suitable loss function into account. Not much work has been done in this area. Similarly, these tests should be generalized from classification to regression, so as to be able to assess the mean square errors of regression algorithms, or to be able to compare the errors of two regression algorithms.

In comparing two classification algorithms, note that we are testing only whether they have the same expected error rate. If they do, this does not mean that they make the same errors. This is an idea that we used in chapter 17; we can combine multiple models to improve accuracy if different classifiers make different errors.

## 19.15 Exercises

1. In a two-class problem, let us say we have the loss matrix where $\lambda_{11} = \lambda_{22} = 0$, $\lambda_{21} = 1$ and $\lambda_{12} = \alpha$. Determine the threshold of decision as a function of $\alpha$.
2. We can simulate a classifier with error probability $p$ by drawing samples from a Bernoulli distribution. Doing this, implement the binomial, approximate, and $t$ tests for $p_0 \in (0, 1)$. Repeat these tests at least 1,000 times for several values of $p$ and calculate the probability of rejecting the null hypothesis. What do you expect the probability of reject to be when $p_0 = p$?

3. Assume $x^t \sim \mathcal{N}(\mu, \sigma^2)$ where $\sigma^2$ is known. How can we test for $H_0 : \mu \geq \mu_0$ vs. $H_1 : \mu < \mu_0$?
4. The $K$-fold cross-validated $t$ test only tests for the equality of error rates. If the test rejects, we do not know which classification algorithm has the lower error rate. How can we test whether the first classification algorithm does not have higher error rate than the second one? Hint: We have to test $H_0 : \mu \leq 0$ vs. $H_1 : \mu > 0$.
5. Show that the total sum of squares can be split into between-group sum of squares and within-group sum of squares as $SS_T = SS_b + SS_w$.
6. Use the normal approximation to the binomial for the sign test.
7. Let us say we have three classification algorithms. How can we order these three from best to worst?
8. If we have two variants of algorithm $A$ and three variants of algorithm $B$, how can we compare the overall accuracies of $A$ and $B$ taking all their variants into account?
9. Propose a suitable test to compare the errors of two regression algorithms.
10. Propose a suitable test to compare the expected rewards of two reinforcement learning algorithms.

## 19.16 References

Alpaydın, E. 1999. "Combined $5 \times 2$ cv $F$ Test for Comparing Supervised Classification Learning Algorithms." *Neural Computation* 11: 1885–1892.

Bouckaert, R. R. 2003. "Choosing between Two Learning Algorithms based on Calibrated Tests." In *Twentieth International Conference on Machine Learning*, ed. T. Fawcett and N. Mishra, 51–58. Menlo Park, CA: AAAI Press.

Demsar, J. 2006. "Statistical Comparison of Classifiers over Multiple Data Sets." *Journal of Machine Learning Research* 7: 1–30.

Dietterich, T. G. 1998. "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms." *Neural Computation* 10: 1895–1923.

Fawcett, T. 2006. "An Introduction to ROC Analysis." *Pattern Recognition Letters* 27: 861–874.

Montgomery, D. C. 2005. *Design and Analysis of Experiments.* 6th ed., New York: Wiley.

Ross, S. M. 1987. *Introduction to Probability and Statistics for Engineers and Scientists.* New York: Wiley.

Turney, P. 2000. "Types of Cost in Inductive Concept Learning." Paper presented at Workshop on Cost-Sensitive Learning at the Seventeenth International Conference on Machine Learning, Stanford University, Stanford, CA, July 2.

Wolpert, D. H. 1995. "The Relationship between PAC, the Statistical Physics Framework, the Bayesian Framework, and the VC Framework." In *The Mathematics of Generalization*, ed. D. H. Wolpert, 117–214. Reading, MA: Addison-Wesley.

Yıldız, O. T., and E. Alpaydın. 2006. "Ordering and Finding the Best of $K > 2$ Supervised Learning Algorithms." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28: 392–402.