

CHAPTER 2

Assessment of Class Predictions

- The Confusion Matrix
- ROC (Receiver Operating Characteristic) Curves
- Optimizing ROC-Based Statistics
- Confidence in Classification Decisions
- Confidence Intervals for Future Performance

The previous chapter focused on models that make numeric predictions. This chapter deals with models whose goal is classification. It must be understood that the distinction is not always clear. In particular, almost no models can be considered to be pure classifiers. Most classification models make a numeric prediction (of a scalar or a vector) and then use this numeric prediction to define a classification decision. Thus, the real distinction is not in the nature of the model but in the nature of the ultimate goal. The implications of this fact will resound throughout the chapter.

Also because of this relationship, you should understand that much of the material in Chapter 1 applies to this chapter as well. Most classification models are just prediction models taken one step further to suit the application. Thus, concepts such as consistency of error magnitude and selection bias apply to most classification problems as well as they do to prediction problems. You should see this chapter as an extension of Chapter 1, not a totally different topic.

The Confusion Matrix

The most basic measure of classification quality is the *confusion matrix*. This matrix contains as many rows and columns as there are classes, plus one additional column if the model allows the possibility of a *reject* category. Some applications require that the model have the option of indicating that a trial case most likely does not belong to any of the trained classes. This is the reject category. The confusion matrix entry at row i and column j contains the number of cases that truly belong to class i and have been classified as class j . Thus, we see that the confusion matrix for a perfect model contains positive entries only on the diagonal, with the off-diagonal elements being zero. Entries off the diagonal represent misclassifications.

The basic confusion matrix consists of case counts. There are two modifications to the basic version that can be useful. If the entries for each row are divided by the total of the entries for the row, each resulting number is the fraction of the cases in this row's class that were classified into the column's class. Sometimes it may be helpful to divide each entry by the total number of cases so as to assess the confusion relative to the entire population. In both cases, it may be advisable to multiply the fractions by 100 to express them as percents.

Expected Gain/Loss

The contents of an entire confusion matrix can sometimes be distilled into a single number. This is possible when each type of misclassification has its own well-defined cost. If the data collection on which the confusion matrix is based is meaningful in terms of its size and the proportion of cases in each class, the total cost incurred by the model is easily computed by multiplying the case count for each type of misclassification by the cost of that misclassification, and summing.

However, things are not usually this simple. There are two common problems with this naive approach. First, the proportion of cases in each class for the dataset that produced the confusion matrix may not equal the proportion to be expected in actual use. Second, the total number of cases is almost never meaningful, having been arbitrarily chosen on the basis of convenience. A measure that is almost always more meaningful is the expected cost per case. If we wish, we can then multiply this figure by the number of cases in a batch to be processed later in order to estimate the cost of processing that batch.

The expected cost per case is usually best computed in two steps. First, compute the expected cost per case for each class. This is done one row (true class) at a time. Divide each entry in the row by the row's total. This gives the fraction of the true class that lies in each classified class. Multiply each of these fractions by the cost associated with that entry, and sum across the row. The sum is the expected cost for a member of this class. Find this for all rows. Second, multiply these costs by the expected proportion of cases in each class and sum. This sum is the expected cost per case, regardless of class.

Here is an example of this computation. Table 2-1 shows the results for a three-class problem. Each of the $3 \times 3 = 9$ entries in the confusion matrix consists of three subentries. The upper one is the actual confusion count. The center entry is the fraction of each row, computed as already described. The bottom entry is the cost associated with that entry. Observe that, as is usually the case, the cost of a correct classification is zero.

The total number of cases in the first row is ten. Thus, the proportions of the first class in each classified class are 0.5, 0.2, and 0.3, respectively. These quantities, as well as the corresponding quantities for the other two rows, are shown in the table. The expected cost for a case in the first class is $0 \cdot 0.5 + 4 \cdot 0.2 + 8 \cdot 0.3 = 3.2$. Similar calculations show the costs for the other two classes as 0.7 and 2.8, respectively. Suppose we expect 0.7 that in the future, 0.3 of the cases will be in class 1, 0.3 in class 2, and 0.4 in class 3. The expected cost per case is computed as $0.3 \cdot 3.2 + 0.3 \cdot 0.7 + 0.4 \cdot 2.8 = 2.29$.

Table 2-1. *A Confusion Matrix with Associated Costs*

5	2	3
0.5	0.2	0.3
0	4	8
2	16	2
0.1	0.8	0.1
3	0	4
4	6	10
0.2	0.3	0.5
5	6	0

There is a potential danger in using expected cost as a performance indicator. This is particularly true if this quantity is optimized for training. The danger is that the expected cost can be sensitive to the specified costs and prior probabilities of the classes. In an industrial situation, where both the expected probability of each class and the cost of misclassifications may be known precisely, the expected cost can be an excellent performance measure. But suppose we have a medical application in which costs must be assigned to errors such as misdiagnosing a malignant tumor as benign. The cost will presumably be much higher than that for the opposite error. But if two different researchers assign different relative costs, their trained models may turn out to be quite different. Be wary of using expected costs as a performance indicator when either the cost or the probability of each class is subject to variation.

ROC (Receiver Operating Characteristic) Curves

This section focuses on a special but common classification situation. Many applications involve only two classes that are mutually exclusive (no case can belong to both classes) and exhaustive (each case must belong to one of the classes). The most common version of this situation is when there actually is only one class, called the *target class*, and each case either belongs to this class or does not. For example, a radar blip is a tank or it is something else (about which we do not care). A credit card transaction is either fraudulent or it is not.

Because this version of the problem is so common, it will form the basis of further discussions. Rather than treating each case as belonging to one of two classes, each case will be considered to be a member of the target class or not a member of that class. This naming convention has obvious military connotations. However, we may just as well call a malignancy a target, or we may call a successful financial trade a target, or we may call a fraudulent credit card transaction a target. The central idea is that there exists a dichotomy between a particular class of interest and everything else.

Hits, False Alarms, and Related Measures

There are several important definitions in this problem. It is obvious that the confusion matrix is two by two. Let us label the quantities in each of the four cells as follows:

True positive (TP): The number of targets correctly classified as targets

False negative (FN): The number of targets incorrectly classified as nontargets

True negative (TN): The number of nontargets correctly classified as nontargets

False positive (FP): The number of nontargets incorrectly classified as targets

The following definitions are commonly employed:

- The *hit rate* is the fraction of the targets (members of the target class) correctly classified as being in the target class. This is $TP / (TP + FN)$. It is also sometimes called *sensitivity*.
- The *false alarm rate* is the fraction of nontargets erroneously classified as being in the target class. Statisticians may call this the *Type I error rate*. This is $FP / (TN + FP)$.
- The *miss rate* is the fraction of the targets erroneously classified as not being in the target class. Statisticians may call this the *Type II error rate*. This is $FN / (TP + FN)$.
- The *specificity* is the fraction of nontargets correctly classified as not being in the target class. This quantity is $TN / (TN + FP)$.

Observe that the hit rate and the miss rate sum to one. Similarly, the false alarm rate and the specificity sum to one. Most military targeting applications employ the hit rate and false alarm rate. Sensitivity and specificity are commonly used in medical applications. The choice depends entirely on which measures happen to be more meaningful to the application at hand.

There are a few other performance measures that are not as commonly employed but that can be immensely valuable in some circumstances. These include the following:

- The usual goal is to detect targets. But sometimes we do not need to detect a large fraction of the targets. Instead, we need to be as certain as possible when a decision of “target” is made. The classic example involves trading financial markets. It does not matter if many large market moves are missed (a low hit rate). But when a large market move is predicted and a position is taken, the model had better be

sure of itself. In this case, the *precision* is important. This is $TP / (TP + FP)$. When this measure is employed, we should also know what fraction of the population is represented by the denominator. This is $(TP + FP) / (TP + FP + FN + TN)$, which is the fraction of the cases that the model decided were targets.

- Occasionally, the converse goal is important. If a decision is made that the case is not a target, this decision must be correct with high probability. This arises, for example, in medical diagnosis. If a case is classified as not being diseased, this diagnosis definitely needs to be correct. In this case, the *negative precision*, sometimes also called the *null precision*, is $TN / (TN + FN)$. We then also need to consider how often this benign diagnosis occurs: $(TN + FN) / (TP + FP + FN + TN)$.

Computing the ROC Curve

Nearly all target/nontarget classification models make a numeric prediction and then compare the prediction to a predefined threshold. If the prediction meets or exceeds the threshold, the case is classified as a target. Otherwise, it is classified as a nontarget. It should be apparent that all of the performance measures just discussed are affected by the threshold. If the threshold is set at or below the minimum possible prediction, all cases will be classified as targets. This means that the hit rate will be a perfect 100 percent, but the false alarm rate will be an abysmal 100 percent. If the threshold is set above the maximum possible prediction, all cases will be classified as nontargets. This means that the hit rate will be zero, and the false alarm rate will be a perfect zero as well. The sweet spot lies somewhere in the middle. By sweeping the threshold from one extreme to the other, the hit rate and false alarm rate also sweep across their range from zero to 100 percent, one improving as the other deteriorates. The curve defined by these two rates (with the threshold serving only as an invisible parameter) is called the *Receiver Operating Characteristic (ROC) curve*.

When the ROC curve is plotted as a graph, the convention is to let the horizontal axis be the false alarm rate and let the vertical axis be the hit rate. Suppose the classification model is worthless, generating numeric predictions that are random, having nothing to do with the true class of a case. It is apparent that the hit rate and false alarm rate will be, on average, equal for all thresholds. The ROC curve will be an approximately straight line connecting the lower-left corner of the graph to the upper-right corner.

Now suppose the model is perfect. In other words, suppose there exists a threshold such that all targets produce a numeric prediction that equals or exceeds the magic threshold, while all nontargets generate a prediction that is less than the threshold. As the plot-generating parametric threshold moves from its minimum to the threshold of perfection, the false alarm rate will move from 100 percent to zero, with the hit rate remaining at its perfect value of 100 percent. As the parametric threshold continues on to its maximum, the hit rate will deteriorate from 100 percent to zero, while the false alarm rate remains at zero. The ROC curve will be a line running from the upper-right corner of the graph to the upper-left corner, then continuing on to the lower-left corner. Models that are good but less than perfect will have a ROC curve that lies somewhere between the two extremes of hugging the upper-left corner and a diagonal line. The more the curve is pulled away from the diagonal toward the upper-left corner, the better the model is performing. In fact, we will see later that the area under the ROC curve is a useful performance measure, since it considers the effectiveness of the model at all possible classification thresholds.

It is not often that simple visual examination of a graphical plot of a ROC curve proves useful. Much more information can be had by studying a table of the relevant performance measures. Table 2-2 is a sample ROC curve for a moderately effective classification model. It shows the values of the performance measures defined earlier, computed across the full range of thresholds.

Like many classification models, this one has been trained to produce an output of -1.0 for nontargets and 1.0 for targets. The ROC curve threshold (the rightmost column) accordingly covers this range. As expected, the hit rate and false alarm rate drop from 100 percent to zero as the threshold increases. Because this model has some predictive capability, the false alarm rate falls faster than the hit rate. In fact, the false alarm rate drops to 38.5 percent while the hit rate remains at a perfect 100 percent. The false alarm rate reaches a perfect zero while the hit rate is still at 60.0 percent. Unless the relative importance of the two types of error is extremely unbalanced, the ideal threshold will lie somewhere between these two points. Note that this chart also shows the specificity, the complement of the false alarm rate.

Table 2-2. ROC Curve for a Moderately Effective Model

Hit (35)	FA/Spec (26)	Mean Err	Target Prec.	Null Prec.	Threshold
100.0	100.0/0.0	50.0	57.4 (100.0)	0.0 (0.0)	-1.000
100.0	88.5/11.5	44.2	60.3 (95.1)	100.0 (4.9)	-0.900
100.0	84.6/15.4	42.3	61.4 (93.4)	100.0 (6.6)	-0.800
100.0	76.9/23.1	38.5	63.6 (90.2)	100.0 (9.8)	-0.700
100.0	76.9/23.1	38.5	63.6 (90.2)	100.0 (9.8)	-0.600
100.0	65.4/34.6	32.7	67.3 (85.2)	100.0 (14.8)	-0.500
100.0	61.5/38.5	30.8	68.6 (83.6)	100.0 (16.4)	-0.400
100.0	46.2/53.8	23.1	74.5 (77.0)	100.0 (23.0)	-0.300
100.0	38.5/61.5	19.2	77.8 (73.8)	100.0 (26.2)	-0.200
91.4	26.9/73.1	17.7	82.1 (63.9)	86.4 (36.1)	-0.100
85.7	26.9/73.1	20.6	81.1 (60.7)	79.2 (39.3)	0.000
82.9	23.1/76.9	20.1	82.9 (57.4)	76.9 (42.6)	0.100
74.3	11.5/88.5	18.6	89.7 (47.5)	71.9 (52.5)	0.200
68.6	3.8/96.2	17.6	96.0 (41.0)	69.4 (59.0)	0.300
60.0	0.0/100.0	20.0	100.0 (34.4)	65.0 (65.6)	0.400
51.4	0.0/100.0	24.3	100.0 (29.5)	60.5 (70.5)	0.500
42.9	0.0/100.0	28.6	100.0 (24.6)	56.5 (75.4)	0.600
34.3	0.0/100.0	32.9	100.0 (19.7)	53.1 (80.3)	0.700
22.9	0.0/100.0	38.6	100.0 (13.1)	49.1 (86.9)	0.800
11.4	0.0/100.0	44.3	100.0 (6.6)	45.6 (93.4)	0.900
0.0	0.0/100.0	50.0	0.0 (0.0)	42.6 (100.0)	1.000

ROC area = 0.9275

Normalized ROC area with sensitivity ≥ 0.9 = 0.6648

In many cases, the ideal threshold can be determined by simple inspection of the first two columns. Interpretation is helped if the number of targets and nontargets is also printed in the heading of the table. In this case, the numbers are 35 and 26, respectively.

If the cost of a miss and a false alarm are about the same, the mean error is a good single-number indicator of performance. This column is computed by finding the average of the false alarm rate and the miss rate (100 minus the hit rate). Of course, if misses and false alarms have significantly different cost, the average error loses much of its relevance.

In some applications, the precision (page 49) is of great importance. Usually we are most interested in the precision in regard to detecting targets. Occasionally we are interested in the precision in regard to confirming that a case is not a target. Table 2-2 provides both types of information. When the goal is correctly detecting targets, the column labeled *target precision* is most useful. Conversely, if the goal is to precisely identify nontargets, the column labeled *null precision* (or *negative precision* in some circles) is most useful. In each of these columns, the percent of cases classified as target or nontarget, respectively, is given in parentheses. We see, for example, that at a threshold of 0.1, 42.6 percent of the cases are classified as nontargets with an obtained (null or negative) precision of 76.9 percent. Similarly, at a threshold of 0.4, perfect target precision of 100 percent is obtained, although only 34.4 percent of the cases are classified as targets. Finally, the area under the ROC curve for this model is computed as a respectable 0.9275. This quantity is discussed in the next section.

Before leaving computation of the ROC curve, let us examine some code snippets that demonstrate an efficient way to perform the computations. In most cases, the user will specify a range of thresholds over which the ROC curve is to be listed, and the number of intervals in which the range is to be divided for the table. A sensible approach is to divide the range into the specified number of intervals, printing a line for each, plus one extra line at the end to cover all thresholds above the user's upper limit. There is almost never any interest in information below the lower limit. So, for example, Table 2-2 encompasses a range of thresholds from -1.0 to 1.0 divided into 20 intervals. The lowest interval contains thresholds from -1.0 to -0.9. The next handles -0.9 to -0.8, etc., until the 20th interval is for 0.9 to 1.0. The last line (number 21) counts cases whose threshold

CHAPTER 2 ASSESSMENT OF CLASS PREDICTIONS

exceeds 1.0. It is customary to label each line with the lower bound of its threshold, since that is where the counting for each bin begins. Here is a code fragment for computing the information required to prepare this table:

```
for (i=0; i<nthresh; i++)
    hit[i] = fa[i] = 0.0;
hit_below = hit_above = fa_below = fa_above = 0.0;

factor = (double) nthresh / (stop - start);

for (icase=0; icase<n; icase++) {
    j = (int) (factor * (ind[icase] - start));

    if (cls[icase] > 0.0) {                // Definition of target
        if (ind[icase] < start)            // First bin includes all thresholds below start
            hit_below += 1.0;
        else if (j >= nthresh)            // Last bin includes all thresholds above stop
            hit_above += 1.0;
        else if (j >= 0)                  // Should always be true, but play it safe
            hit[j] += 1.0;
    }
    else {                                // Non-target
        if (ind[icase] < start)            // First bin includes all thresholds below start
            fa_below += 1.0;
        else if (j >= nthresh)            // Last bin includes all thresholds above stop
            fa_above += 1.0;
        else if (j >= 0)                  // Should always be true, but play it safe
            fa[j] += 1.0;
    }
}

/*
The bin counts are computed. Now cumulate.
*/

t = hit_above;
for (j=nthresh-1; j>=0; j--) {
    t += hit[j];
```

```

    hit[j] = t;
}

n_targets = (int) (hit_below + t);
for (j=0; j<nthresh; j++)
    hit[j] /= (n_targets + 1.e-30);

hit_above /= (n_targets + 1.e-30);

t = fa_above;
for (j=nthresh-1; j>=0; j--) {
    t += fa[j];
    fa[j] = t;
}

n_nontargets = (int) (fa_below + t);
for (j=0; j<nthresh; j++)
    fa[j] /= (n_nontargets + 1.e-30);
fa_above /= (n_nontargets + 1.e-30);

```

In this code, `hit` and `fa` are real arrays that will hold the hit rates and false alarm rates for the ROC curve. The `_above` and `_below` versions will cumulate the corresponding information for thresholds above and below the specified range, respectively. The user requested `nthresh` intervals ranging from start to stop. For each case, `cls` defines the class membership of the case, with values greater than zero indicating a target, and values less than or equal to zero being a nontarget. Also for each case, `ind` is the value of the indicator variable, which will be compared with the threshold in order to make a classification decision. Values of `ind` greater than or equal to the threshold will be classified as being a target.

Operation of the code is straightforward. It passes through all cases. For each, it checks the true class membership. The bin corresponding to the indicator variable is computed, and that bin count is incremented. If the indicator lies outside the user's specified range, the appropriate outer bin is incremented.

After the bins are cumulated, the algorithm sums downward. Remember that each bin covers cases for that bin and all above it (cases whose indicator exceeds the threshold). To get the total number of targets we add those in the below-the-limit bin.

Finally, divide all bin counts, as well as the extra bin on top, by the number of targets in order to convert them into fractions. The *1.e-30* ensures that we do not divide by zero in degenerate situations.

We can print or store the table by using code similar to that shown next. Refer to page 48 for definitions of the various ROC parameters that can be computed from these basic ingredients.

```
for (j=0; j<=nthresh; j++) {
  if (j < nthresh) {
    hj = hit[j];
    faj = fa[j];
  }
  else {
    hj = hit_above;
    faj = fa_above;
  }

  tp = hj * n_targets;
  fn = (1.0 - hj) * n_targets;
  tn = (1.0 - faj) * n_nontargets;
  fp = faj * n_nontargets;
}
```

Area Under the ROC Curve

One disadvantage of the ROC curve table as a performance “measure” is that it contains a tremendous amount of superfluous detail. Sometimes a single number is all that is needed. We have already noted that a worthless model has a ROC curve that is a diagonal line, and a perfect ROC curve hugs the upper-left corner of the graph. It is apparent that a good model will have more area under its ROC curve than a poor model. A worthless model will have an area of 0.5 (because the graph runs from 0.0 to 1.0 in both axes). A perfect model will have an area of 1.0.

It is not difficult to compute the area under a ROC curve. The randomly sampled cases in the test set are themselves subject to statistical variation, so there is no need to attempt to achieve high accuracy in the numerical integration algorithm. Simple summation is generally sufficient. We’ll jump right in with a code fragment and explain its operation later.

```

for (icase=0; icase<n; icase++) {
    indicator[icase] = ind[icase];           // model's output
    is_target[icase] = (cls[icase] > 0.0) ? 1 : 0; // Target vs. non-target
}

qsortdsi (0, n-1, indicator, is_target); // Sort outputs ascending, moving class

tp_count = 0;           // Counts true positives
fp_count = 0;           // Counts false positives
j = n-1;                // Will count down as icase counts up
area = 0;                // Will cumulate (unnormalized) ROC area here
ROC90 = -1.0;           // Will cumulate ROC area for hit rate >= .9

for (icase=0; icase<n; icase++) {
    if (is_target[j--]) { // If this is truly a target
        ++tp_count;      // Then it is a true positive at this threshold
        area += fp_count; // Count this thin rectangle

        if (tp_count / (double) n_targets >= 0.9) {
            if (ROC90 < 0.0) // Start with partial rectangle
                ROC90 = (tp_count / (double) n_targets - 0.9) /
                    (1.0 / (double) n_targets) * (n_nontargets - fp_count);
            else
                ROC90 += n_nontargets - fp_count; // Area to right of curve
        }
    }
    else // But if it is not a target
        ++fp_count; // Then it is a false positive
}

ROC_area = 1.0 - (double) area / ((double) n_targets * (double) n_nontargets);
ROC90area = ROC90 / ((1.0 - 0.9) * (double) n_targets * (double) n_nontargets);

```

The first step is to copy the indicator and class (target versus nontarget) information to work vectors so we can rearrange them without annoying the calling routine. The `qsortdsi` routine sorts the `n` cases in ascending order of the indicator variable, simultaneously rearranging the class vector. Note that other than this sorting operation, the actual values of the indicator variable are not needed. The order in which the classes appear (ideally with the nontargets near the beginning, and the targets near the end after sorting) is all

that is needed to compute the area under the ROC curve. Also note that if there are ties in the indicator variable, the algorithm does not employ any special tie-breaking procedures. Most applications do not produce ties, so this should not be a problem. But if ties are possible, it might be wise to call the algorithm several hundred times, using random tie breaking, and average the results. I am not aware of an algorithm for explicitly handling ties in a rigorous fashion (although such an algorithm probably does exist).

This code loops through the sorted data, working down from the highest indicator to the lowest. At each implied threshold, the class of the case is checked. If it is a target, the *true positive* counter is incremented. Otherwise, the *false positive* counter is incremented.

It helps to visualize the ROC curve as lying in a grid consisting of as many rows as targets, and as many columns as nontargets. The loop starts at the lower-left corner of the grid and advances upward and to the right as the implied threshold decreases. Every time the *true positive* counter is incremented, it moves up one row. When this happens, a new rectangle of unit height is defined. The width of this rectangle (on the left side of the ROC curve) is the current false-positive count, so we add this to area. In other words, we are summing the number of grid squares to the left of the ROC curve. After traversing the entire curve, we divide this sum by the total number of grid points and subtract from one to get the area below the ROC curve.

Another interesting area measurement is computed as we traverse the curve. In many applications, a model is useful only if it can obtain a very high hit rate. For example, if we are detecting incoming missiles, a perfect or nearly perfect hit rate is mandatory. If we are deciding whether a tissue sample is malignant, we'd better have an extremely high probability of identifying malignant specimens. In such applications, most of the ROC curve is irrelevant. The only important part is that which lies above a specified hit rate.

The code just shown computes that area under the ROC curve that is above a hit rate of 0.9. It does this by cumulating the area to the right of the ROC curve as soon as the hit rate reaches 0.9. The first rectangle is tricky, though, because it will in general be partial, with the 0.9 line splitting it. Thus, we initialize ROC90 to a negative number and use this as a flag. For the first rectangle we interpolate by using a height less than one unit. When the summation is complete, we compute the area by dividing by the total number of unit squares and normalize for the fraction of the total possible area covered.

Cost and the ROC Curve

The expected cost (gain or loss) of a classification model was discussed on page 46. Let us now specialize to the binary classification scenario in which each case is either a target or a nontarget. Suppose the prior probability that a case is a target is q . Let p_1 be the probability of a type I error, a false alarm, and let c_1 be the cost of this error. Let p_2 be the probability of a type II error, a miss, and let c_2 be the cost of this error. In most applications, q will be known from theory or experience, and the costs will be known at least approximately if not exactly. Therefore, these three quantities can be considered fixed constants. The two error probabilities, type I and II, are determined by the threshold. The expected cost is given by Equation (2.1). Algebraic manipulation of this equation leads to Equation (2.2).

$$COST = (1 - q)p_1c_1 + qp_2c_2 \quad (2.1)$$

$$p_1 = \frac{COST}{(1 - q)c_1} - \frac{qc_2}{(1 - q)c_1}p_2 \quad (2.2)$$

Equation (2.2) makes it clear that for any fixed cost, the relation between p_1 and p_2 is linear. If this line is plotted on the ROC curve, its slope is determined by the prior probability that a case is a target and by the ratio of the costs of the two types of errors. As the expected cost is varied, a family of parallel lines is defined, with lines down and to the right representing higher cost. If the line intersects the ROC curve at two points, these intersection points represent two thresholds that provide equal but excessive cost. Lines that lie entirely above and to the left of the ROC curve represent performance that is unachievable by the model. The line that is precisely tangent to the ROC curve represents the best obtainable cost.

The preceding discussion should make it clear that the area under the ROC curve is not necessarily a good indicator of how well a classification model actually performs. Certainly the ROC area is a very good overall quality indicator. But when actual cost of operation is considered, ROC area can be deceptive. Look at Figure 2-1. This shows ROC curves for two models. They have identical areas under the curve. The prior probability and relative error costs in this example favor a low false alarm rate, even at the expense of a quite high miss rate. It is apparent that one model is very superior to the other in terms of cost, despite having the same ROC area. It should also be obvious that if the cost

situation were reversed, the other model would be favored. Therefore, do not be afraid to judge the power of a model based on the area under its ROC curve. But also do not fail to consider the actual cost of the model.

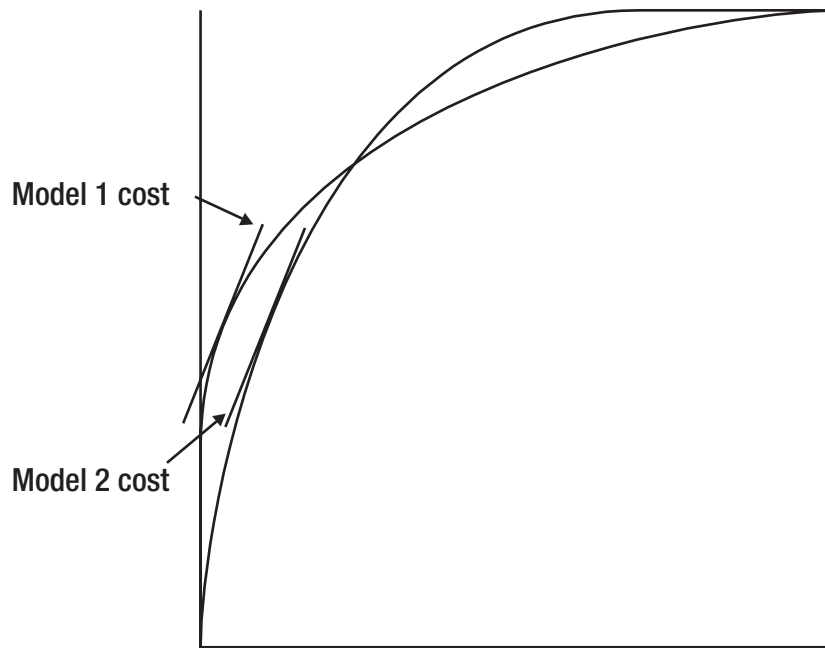


Figure 2-1. *The ROC area can be deceptive*

Optimizing ROC-Based Statistics

In many or most target/nontarget situations, the best possible results are obtained by optimizing a performance criterion that is directly based on some aspect of the ROC curve. The principal disadvantage of this approach is that such criteria almost never lend themselves to traditional optimization techniques. They are difficult or impossible to differentiate, eliminating any training methods that rely on gradients. They can exhibit troublesome discontinuities that may confound naive algorithms. In general, stochastic methods such as genetic algorithms or simulated annealing are the only practical training methods. Nevertheless, ROC-based statistics can be such effective optimization criteria that it is well worth any trouble involved.

A big hint that ROC-based statistics are useful comes from examining Figure 2-1. The two ROC curves shown in this figure have identical areas. It is likely that the models' performance as indicated by other traditional measures, such as MSE or R-squared, would

also be similar or identical. Yet the difference in their actual costs is large. It obviously makes sense to design a training scheme that directly optimizes the cost. This section discusses several ROC-based optimization criteria that have widespread applicability.

Optimizing the Threshold: Now or Later?

ROC-based statistics depend not only on designing a good prediction model but also on finding an optimal threshold for defining the target/nontarget decision. How is this threshold obtained? There are three primary methods for doing so.

- The old-fashioned method is to totally ignore the threshold while training the model using some other optimization criterion. After the model is trained, compute the optimal threshold. If the result is satisfactory, stop. Otherwise, try a different model. This silly approach was needed when computer power and sophisticated optimization algorithms were scarce. It's barely excusable today.
- Treat the threshold as just one more optimizable parameter in the model. Append it onto the end of the parameter vector that is fed to the optimization algorithm. This method can work well. However, the addition of another parameter will significantly slow most optimization algorithms. This method should be avoided if possible unless computer time is freely available.
- Slightly increase the complexity of the criterion-evaluating function by including threshold optimization in this function. The main optimization algorithm supplies the criterion function with trial values of the model's parameters. After this trial parameter set is tested with all of the training cases, the threshold that optimizes the training criterion is computed, and the corresponding best performance measure is returned to the main optimizing algorithm. In other words, threshold optimization is performed with every function evaluation by the model optimizer. The main optimization algorithm optimizes only the model parameters, not the threshold. But for every trial parameter set, the function evaluating algorithm includes threshold optimization as part of its operation. This is usually the preferred method as it is relatively easy to implement, yet it effectively optimizes the threshold simultaneously with the model parameters.

The difference in training time using the second and third methods just described is sometimes great and sometimes insignificant. Most often it is great. Here is why. Suppose, for example, that the prediction model has three optimizable parameters. If we were to use the second method, the training algorithm would be effectively dealing with four optimizable parameters. Anyone who has worked in the field of numerical optimization knows that the speed difference between three and four parameters can be huge. Of course, the third method is not a free lunch. The price paid for allowing the main (complex and slow) optimizer to be burdened with only three parameters instead of four is that now the criterion function that it calls to evaluate performance must itself become a mini-optimizer. After setting the three trial parameters and evaluating this trial model for all training cases, the performance evaluator must traverse the ROC curve to find the optimal threshold. The key point is that in nearly all practical applications, applying the model to the training set is the slow part of a function evaluation. The time taken to traverse the ROC curve is usually insignificant. A very small price is paid to obtain a very large return.

Here is a sample code fragment that illustrates how the function-evaluating routine called by the main optimizer can compute the threshold that produces the best cost. This code also shows how to compute an effective optimization criterion. Simply optimizing the cost itself may not be the best approach. An absolute cost figure, while immensely informative in one sense, can be worthless for judging whether the model is really doing anything useful. If the two types of errors have very disparate costs, a naive model might just set the threshold at whichever extreme forces all misclassifications to be the cheaper type. Thus, a more informative performance criterion would compare the model's attained cost with that obtainable with a naive model. Finally, this code illustrates the fact that when the model is validated, it would not be fair to optimize the threshold. The previously trained threshold must be employed. This facility is demonstrated. In this code, the predicted values are in the *pred* array, and the true classifications are in the *trueval* array. In this array, true targets are flagged with 1.0, and nontargets are flagged with -1.0.

```
// A naive model would set the threshold at whichever extreme results
// in nothing but the cheaper error. Compute this naive cost so that
// we can scale the actual cost with it.

factor1 = factor2 = 0.0;      // Will be cost at each extreme threshold
for (i=0; i<n; i++) {        // Check each case
    if (trueval[i] < 0.0)      // If this is a non-target
        factor1 += fa_cost;   // It would be a false alarm at min thresh
```

```

if (trueval[i] > 0.0)           // If this is a target
    factor2 += miss_cost;      // It would be a miss at max thresh
} // For all cases

if (optimize_threshold) {      // If we are training (versus testing)
    sort (n, pred, trueval);    // Sort predicted values, keeping classes with them
    thr = pred[0];              // Begin by setting trial threshold tiny
    n_missed = n_fa = 0;        // None are missed
    for (i=0; i<n; i++) {        // But pass through entire set (Redundant from above)
        if (trueval[i] < 0.0)    // And find all cases not a target
            ++n_fa;              // These are false alarms
    } // This loop could be avoided by counting n_fa with factor1 above. Shown for clarity.

    best_cost = miss_cost * n_missed + fa_cost * n_fa; // Also redundant: factor1!
                                                // Because n_missed is still zero

    for (i=1; i<n; i++) {        // Try all other possible thresholds
        if (trueval[i-1] < 0.0)  // If we just passed a non-target
            --n_fa;              // We now have one less false alarm
        if (trueval[i-1] > 0.0)  // If we just passed a target
            ++n_missed;          // We now have one more miss
        if (pred[i-1] == pred[i]) // If we have a block of ties
            continue;           // Only the first gives a valid cost
        test_cost = miss_cost * n_missed + fa_cost * n_fa;

        if (test_cost < best_cost) { // If it is the best so far
            best_cost = test_cost;    // Update the best cost
            thr = pred[i];           // And keep track of this threshold
        }
    } // For all possible thresholds beyond the lowest

    // There is one more possibility: Try a threshold beyond the max,
    // which results in nothing but misses.
    if (factor2 < best_cost) {
        best_cost = factor2;
        thr = 1.00000001 * pred[n-1];
    }
} // If set_threshold

```

CHAPTER 2 ASSESSMENT OF CLASS PREDICTIONS

```
else {                                     // Validating, so do not set threshold
    n_missed = n_fa = 0;
    for (i=0; i<n; i++) {                 // Count all false alarms and misses
        if ((pred[i] >= thr) && (trueval[i] < 0.0))
            ++n_fa;
        if ((pred[i] < thr) && (trueval[i] > 0.0)) // Could make this logic slightly faster
            ++n_missed;                     // But this is more clear
    }

    best_cost = miss_cost * n_missed + fa_cost * n_fa;
} // Validating, so do not compute threshold

if (factor1 < factor2)                   // Which naive model is cheaper?
    criterion = best_cost / factor1;     // Don't forget to prevent division by zero here!
else
    criterion = best_cost / factor2;
```

Note that the preceding code contains several redundant calculations. If speed were critical, they could be eliminated as noted, at the cost of a bit of clarity. Also note that the final division should protect against the pathological condition that the factor in the denominator is zero. This is avoided here for clarity.

Finally, this example considers only the cost of errors, and it compares this cost to that of a naive model. A gain-based method of threshold optimization is presented on page 273 as an alternative.

Maximizing Precision

This section has focused on the hit rate and the false alarm rate, largely because these two quantities are intimately related to the expected cost of using the model. But in many instances, the *precision* (page 49) is of paramount importance. Consider a stock-picking application that examines hundreds or thousands of equities, looking for a few good ones to purchase. The miss rate is of no great importance, because we do not particularly care if quite a few bargains are missed. Getting them all would be nice, but this is obviously an unrealistic expectation. The false alarm rate is of somewhat more importance, because we will certainly care to some degree what percentage of the losers are mistakenly chosen by the model for purchase. But even this figure can be misleading. What we really care about is how well the stocks that are picked by the model ultimately

perform. In other words, of those that the model instructs us to purchase, what fraction actually does turn out to be good choices? This is defined as the precision of the model. Any good training algorithm should be prepared to maximize this quantity.

Sometimes it is not the target precision that is important, but rather the nontarget precision. Consider a medical model that classifies a tissue sample as benign or malignant. If one judges only the miss rate and the false alarm rate, the former is obviously more important than the latter. But what is of greatest importance is the answer to the question, “What fraction of those classified as benign truly are benign?” This is the nontarget precision (assuming that malignant is the target). Be prepared to optimize this quantity also.

Generalized Targets

So far, we have assumed that a target comes in exactly one form, and a nontarget comes in exactly one form. But in many applications, some target hits are better than others, and some misses or false alarms are worse than others. Understand that we are not talking about multiple-class problems. We are not explicitly trying to classify a case into one of three or more categories. We are still making a binary decision about whether a case is or is not the target being sought. It is just that there are different degrees of success and failure. The classification model may be able to use this information to increase its probability of hitting the most important targets and/or avoiding the worst false alarms.

One example of this situation is if the model examines passive infrared photographs of battle terrain, seeking to locate powered vehicles in a natural background. Missing an enemy Jeep would be bad, but missing an enemy tank with its gun pointed at you would be disastrous. Another example comes from automated market trading. Suppose the model defines a target as the morning of a day in which a market will rise substantially. Some false alarms will be characterized by the market closing about where it opened, producing no profit. Other false alarms will be characterized by the market plunging, producing a huge loss. The latter is clearly worse than the former. In situations like these, it may make sense to train the model to distinguish between errors of different severity. Examples of appropriate criteria are given in the next few sections. Note that these examples are geared toward maximizing a gain. They could easily be restructured to minimize a cost. This choice is purely due to common convention in this field.

Maximizing Total Gain

Assume that we have a model that makes a binary target/nontarget decision. When this model is applied to a case and the decision is made, the result will be a measured gain, which may be negative to indicate a loss. The model is applied to the entire training set, one case at a time, and the individual gains are tallied. Probably the most obvious performance figure to optimize is the total gain achieved on the training set. This is often a terrible choice.

The total gain ignores the variation in performance among the cases in the training set. In the previous chapter we saw that uniform performance helps generalization (page 3) and evolutionary stability (page 6). Optimizing total gain flies in the face of these extremely important considerations.

The situation becomes even worse for applications in which precision is more important than the hit rate or false alarm rate. Many applications have a gain of zero associated with declaring a case to be a nontarget, regardless of whether the decision is correct or incorrect. The chips go down only when a decision of *target* is made. A possibly large gain will be achieved if the decision proves to be correct, and a possibly large loss will be suffered if the decision is wrong. Since the decision threshold is probably being optimized along with the model parameters, different threshold values can generate significantly different numbers of target decisions. A parameter trial involving a large threshold may produce few target decisions, while another trial having a low threshold may produce many target decisions. This can have a profound impact on the total gain. Suppose one trial parameter set produces a large number of very large gains and a moderate number of very large losses. Then suppose another trial produces a decent number of moderate gains and no losses. In almost all situations, the second parameter set should be treated as preferable to the first, even though the first will have greater total gain. Models that generate consistently good results are nearly always better than models that generate wild performance swings ranging from fantastic to painful. Just ask any financial market trader. In fact, in the finance sector, consistent performance is particularly important because a prediction model that has extremely low probability of disaster can be leveraged to generate large returns, something that cannot be done if the prospect of a ruinous trade always looms nearby. Optimizing total gain is almost always problematic.

Maximizing Mean Gain

Assume that we are in the common situation of incurring a gain or loss on a case if and only if the case is classified as a target. This is nearly always true for automated market trading, because we open a trade if and only if the model detects the predefined target condition. If we do not open a trade, there is no gain or loss. Dividing the total gain by the number of target decisions provides the mean gain per decision. If the gain is one for a hit and zero for a miss, the mean gain is the precision (page 49) of the model.

Like the total gain, the mean gain ignores the consistency of the model's performance on individual cases. For this reason, mean gain shares most of the problems of total gain and is generally best avoided. However, mean gain can be good in some situations. For example, if a unique gain is always obtained for a hit and a smaller (or zero) unique gain is always obtained for a miss, the concept of consistency becomes irrelevant. Maximizing mean gain is equivalent to maximizing precision, which may be just what is desired. Even in this situation, care must be taken. The threshold will often be optimized along with the model's parameters. It will often happen that a very high threshold will provide just a few target decisions, all of which are correct. In fact, the threshold might be so high that just one target decision is made for the entire training set. If this happens to be a correct decision, which is likely, the precision will be a perfect 1.0, and the optimizer will be pleased. The users will not be nearly so pleased. This serious problem can be avoided by imposing a constraint that at least a specified minimum number of target decisions must be made. This prevents the trained threshold from being set unrealistically high by the optimizer.

Maximizing the Standardized Mean Gain

Two simple modifications can transform the mean gain from a poor optimization criterion to an excellent one. The worst problem with mean gain is that it ignores consistency. This problem is eliminated by computing the standard deviation of the gains and dividing the mean gain by this quantity. Inconsistent models will have a large standard deviation, which will lower this criterion accordingly. The other modification is to multiply by the square root of the number of target decisions. This rewards models that achieve their good performance by means of a large number of hits. Conversely, models that obtain good average performance based on very few good cases are penalized. This criterion, called a *t-score* by statisticians, is shown in Equation (2.3). In this equation, there are n target decisions, with the i 'th target decision producing a gain of g_i . The mean gain is shown in Equation (2.4).

$$t = \frac{\sqrt{n}\bar{g}}{\sqrt{\frac{1}{(n-1)}\sum_{i=1}^n (g_i - \bar{g})^2}} \quad (2.3)$$

$$\bar{g} = \frac{1}{n} \sum_{i=1}^n g_i \quad (2.4)$$

This criterion is based on much more than intuition, although its mathematical properties are mostly beyond the scope of this text. Here are a few of the useful properties of this criterion:

- It does a good job of encouraging consistency. If a good mean gain is obtained by virtue of one or very few extremely large individual gains nestled amid mediocrity, this will inflate the denominator of Equation (2.3) as much as the numerator, nullifying the deception. But beware of models producing extremely consistent results. This can produce inflated values of the criterion, or even cause division by zero.
- Scaling according to the number of target decisions encourages repeatability. Suppose two competing good models have identical means and standard deviations, but one had two target decisions and the other had hundreds of target decisions. It should be obvious that the latter is to be preferred, because the former could have easily appeared as the result of pure luck. The scaling reflects this fact.
- The choice to scale by the *square root* of the number of target decisions has some compelling mathematical justifications. Under fairly general and realistic conditions, the criterion will have a standard deviation of approximately one when this scaling factor is used. This gives meaning to the criterion in absolute terms. Of course, a specifically optimized criterion loses all statistical meaning as far as probability statements are concerned. But it still retains its absolute meaning. And if the criterion is computed from an independent test set whose distribution is not terribly bizarre, meaningful statistical statements may be possible.

This criterion may often be legitimately applied to an independent test set to assess whether the mean gain of cases classified as targets is significantly greater than zero (or perhaps some other quantity). This information is often valuable. Consult any elementary statistics text for detailed information about how to use the *one-sample t-test* to decide if good results obtained on a test set are really good or if instead they might be nothing more than the product of luck.

Here is a summary of the important concepts of this test. A fundamental assumption that is theoretically necessary in order to make probability statements is that the gains must have a normal distribution. In practice, this may not be terribly restrictive. Even binary data, far from normal, generally gives acceptable results as long as there are a decent number of data points. The only cause for serious worry is if the distribution of the gains has one or two heavy tails. For example, if most gains range from -5 to 5, but a few gains of plus or minus 100 may occasionally appear, the t-test should not be used to make probability statements. We will discuss robust alternatives later in this text.

In many applications, a worthless model would be expected to obtain a mean gain (among cases classified as targets) of zero or less. If a positive mean gain is obtained, we would like to know the probability that a gain at least this good might have been the result of nothing more than luck. This probability may be found by consulting a table in nearly any statistics text. As a rough rule of thumb, obtaining a *t*-score in excess of 2.0 is quite good. If there are at least 60 cases classified as targets, the probability of this occurring by luck is about 0.025. A score over 3.0 is outstanding.

Finally, if there is a need to assess the model on a test set but the distribution of gains is too non-normal to trust the t-test, there is an alternative called the *bootstrap* test. See page 135 for details.

Confidence in Classification Decisions

It is often useful, or perhaps even vital, to have the ability to supplement a classification decision with a probability statement regarding confidence in that decision. This can be a surprisingly difficult task. The data collection process becomes significantly more stringent in that more and better data is needed. Several crucial assumptions must be satisfied. And even then, confidence statements may themselves be subject to troubling statistical variation due to random sampling error. This is a dangerous topic that must be approached with great respect. That said, let us begin.

Hypothesis Testing

The most straightforward approach to confidence calculation comes from standard statistical practice. Traditional nomenclature will be used here. Assume, as we have been doing, that the task is to classify a case as a target or as a nontarget. The model makes a numeric prediction, and the classification decision is based on the value of the prediction relative to a fixed threshold. If the model is effective, its predictions will usually be larger for target cases than for nontarget cases. Models are often trained to predict a value of 1.0 for targets and a value of -1.0 for nontargets. In most cases, the model's actual predictions will lie somewhere between these two extremes. Generalized targets, involving many possible levels, are also possible, though less common.

Traditional statisticians postulate an entity called the *null hypothesis*, which asserts that the condition we are seeking is not present. In the current context, we are most likely interested in being able to accompany a decision that a case is a target with a statement of confidence that this is so. Later, we will discuss the converse situation. So, right now our null hypothesis is that the case is a nontarget. The technique presented in this section will attempt to answer the question, "If this case truly is a nontarget (i.e., the null hypothesis is true), what is the probability that a prediction at least this large might have arisen from nothing more than random chance?" If this probability turns out to be quite small, we would be inclined to reject the null hypothesis in favor of the alternative that the case is a target. The reasoning in this situation goes like this:

- 1) The case is either a target or a nontarget.
- 2) If it is a nontarget, there is only a small probability that the model's prediction would be this high.
- 3) Therefore, it is probably a target.

There are two aspects of this confidence technique that must be fully understood. The first is rather subtle but vitally important. We do not start with an attained predicted value and then assert the probability that the case is a target or nontarget. The exact opposite occurs. We hypothesize the true class of the case and then compute the probability of this attained prediction under that hypothesis. In practice, most people will jump right to step 3 and assert the probability that the case is a target (the alternative hypothesis), based on the prediction. This is not always a deadly sin, but it can be. A thorough understanding of the route to the alternative hypothesis is needed.

Understanding the second aspect of this confidence technique is utterly crucial. It is a major source of error among careless practitioners, and the error is extremely serious. If one finds that there is a very small probability under the null hypothesis of obtaining a predicted value as large as that obtained, it is legitimate to conclude that there is a high likelihood that the null hypothesis is false, and hence that the alternative is true. However, the converse conclusion absolutely cannot be drawn.

I once more remind you that, as is common practice, we have trained the model with nontargets having a small (probably -1) value and targets having a large (probably $+1$) value. Suppose the prediction is at the low end of the scale, where most nontargets lie. We will naturally find that under the null hypothesis, a prediction of this magnitude is quite likely. (Computation of this probability will be discussed soon.) It is tempting to conclude that the null hypothesis is true. This is most emphatically *not* allowable.

In particular, suppose we have a means of computing probabilities associated with predictions made with nontarget cases. This is called the null hypothesis distribution. The model is presented with an unknown case, and its prediction is solidly in the nontarget territory (small). As much as we would like to conclude that there is a high confidence that this case is a nontarget, we *cannot* do so. For all we know, a large fraction of target cases may, to our annoyance, also produce predictions down here at the low end. The only way we can make statements about the confidence that a case is a *nontarget* is if we have information about the model's behavior for *targets*.

To summarize this vital and often overlooked concept, if the probability of an obtained prediction is small under the null hypothesis, we can legitimately have high confidence in the alternative. However, if the probability of an obtained prediction is large under the null hypothesis, we most certainly cannot conclude that the null hypothesis is true.

Now let's discuss the converse situation. When a case is presented to the model and the prediction is below the threshold, we classify the case as a nontarget. We may want to associate a probability with this decision. This is done with essentially the same technique as that just described, but in reverse. Now, the null hypothesis is that the case is truly a target. We must know the distribution of predictions for targets. We compute the probability of obtaining a prediction this small (or smaller) if the case really is a target. If the probability of a prediction so small is tiny, we can legitimately conclude that the case probably is a nontarget.

Once again, for this reverse situation, we see that the ability to make a confidence statement about one class hinges on our knowledge of the model's predictions for the *other* class. And once again, be warned that the opposite reasoning cannot be employed.

If we know the distribution of predictions for targets and we obtain a large prediction that lies solidly in the realm of target predictions, we cannot conclude that the case is a target. To do so would be a grave error. Our capabilities are limited to *rejecting* a hypothesis in favor of the alternative. We can never accept a hypothesis based on a distribution under that hypothesis.

The entire preceding discussion has hinged on our ability to know (or at least approximate) the distribution of the model's predictions under a hypothesis (the so-called null hypothesis, whether it be nontarget or target). In practice we will virtually never have direct theoretical knowledge of this distribution. It must be estimated by testing the model on a dataset. This dataset must satisfy several important requirements. In many applications, these requirements can often be difficult to meet. Failure to adequately satisfy these requirements can lead to erroneous confidence calculations. Be warned. Here are the requirements:

- The dataset must be totally independent of the data on which the model was trained. If the quality of a trained model was judged by its performance on a test set, with the idea that poor performance would lead to rejection of the model, this test set cannot be used to compute confidences. The reason is subtle but important: because the model was to be kept if and only if it performed well on the test set, there is a built-in prejudice toward good performance on this dataset. A completely new dataset must be collected.
- A large number of cases, typically at least several hundred, must be employed. This subject is discussed in more depth on page 76.
- All possible examples of the hypothesis population must be represented in the dataset, and they must appear in quantities proportional to their expected appearance rates in real life. In practice, this is the most difficult requirement to satisfy. If a case appears later, when the model is used, and this case was not represented in the confidence dataset, false rejection of the hypothesis may occur. For example, suppose the model will discriminate between benign and malignant skin lesions, and the null hypothesis is that the lesion is benign. Then the confidence set must include samples of all possible sorts of benign lesions that a doctor may excise, and they must be represented in at least approximately the same proportions that they would be encountered in real life. This is hard.

Once the confidence dataset is in hand, it is used to compute the function that converts a prediction to a confidence. This is most easily done by sorting the predictions and counting cases in the appropriate tail. The confidence in the alternative (which is *not* a probability, just a deliberately vague term) is estimated as one minus the fraction of cases in the tail.

For example, suppose we have a large and representative collection of nontarget cases. Our model has been trained to make small predictions (such as -1) for nontargets and large predictions (such as $+1$) for targets. To prepare for confidence evaluation, this trained model is invoked for each case in our nontarget collection. We preserve the predictions made for these nontarget cases, as they will form the basis of confidence calculations.

Later, an unknown case is tested, and it produces a prediction greater than the threshold. Therefore, we call it a target, and in addition to this classification decision, we would like a confidence measure. We now turn to the collection of predictions made for our nontarget confidence set, as described in the previous paragraph. Suppose that only a small fraction, say 0.03, of the nontarget collection has predictions this large or larger. Under the (stringent!) assumption that the confidence collection truly is representative of nontargets, we conclude that there is only a 3 percent chance of obtaining a prediction this large if the case is a nontarget. This lets us assert that we are 97 percent confident that the case is a target. Note, by the way, that we are most definitely *not* saying that there is a 97 percent *probability* that the case is a target. We are restricted to using the deliberately vague term *confidence*. This subject is discussed in more detail on page 82.

Alternatively, we might have a confidence collection containing a large number of representative target cases. At some point in the future, an unknown case is presented to the model, and the resulting prediction is less than the threshold. Therefore, we call this case a nontarget. Examining the confidence collection of targets, we may find that 0.4 of the cases in this collection have predictions this low or lower. We conclude that only 60 percent confidence in our nontarget decision is justified.

The best way to compute a hypothesis-test confidence is to break the process into two steps. First, the predictions in the confidence collection are sorted, and tail fractions are computed and stored. Second, as new unknown cases are encountered, a fast binary search is used to locate the correct position in the sorted array, and interpolate to return a probability. Naturally, this process is done separately for the target and nontarget collections. Targets and nontargets are not pooled into a single collection.

CHAPTER 2 ASSESSMENT OF CLASS PREDICTIONS

Here is a code fragment demonstrating the first step. We start with *ncases* predictions in the *x* array. This array is sorted, duplicates are removed, and the algorithm yields *n*, the number of unique values. Both tail probabilities (left and right) are computed here. In practice, only the right tail is used for computing target confidence from the nontarget collection, and only the left tail is used for computing nontarget confidences from the target collection.

```
sort (ncases, x);           // Sort ascending
i = 0;                     // Will index original predictions
n = 0;                     // Will count unique values

while (i < ncases) {       // Check all predictions in confidence collection
    x[n] = x[i];           // Copy first occurrence of this prediction
    rprob[n] = (double) (ncases - i) / ncases; // Fraction greater or equal to x[n]
    while ((++i < ncases) && (x[i] == x[n])); // Bypass repeats
    lprob[n++] = (double) i / ncases; // Fraction less or equal to x[n]
}
```

Once the confidence mapping function has been found by the algorithm just shown, all that remains is to perform a simple table lookup and interpolation to assign a confidence to an observed prediction. Here is code to do this for the left tail. This would be used when we have obtained a low prediction and want to compute confidence in our classification of the case as a nontarget. The confidence sample contained targets.

```
if (observed > x[n-1])     // If the prediction is huge (rare!)
    return 0.0;            // We surely have no confidence

if (observed <= x[0])      // If the prediction is tiny
    return 1.0 - lprob[0]; // Inspires greatest reasonable confidence

lo = 0;                   // Will always keep x[lo] < observed
hi = n - 1;               // And x[hi] >= observed

for (;;) {                // Binary search loop
    mid = (lo + hi) / 2;   // Center of remaining search interval
    if (mid == low)        // Are hi and lo adjacent?
        break;           // If so, we are there
    if (x[mid] < observed) // Replace appropriate end point with mid
        lo = mid;
```

```

else
    hi = mid;
}

tail = lprob[hi-1] + (observed - x[hi-1]) / (x[hi] - x[hi-1]) * (lprob[hi] - lprob[hi-1]);
return 1.0 - tail;

```

The code for processing the right tail (confidence in a target decision based on the nontarget samples) is practically identical to that just shown. The binary search and interpolation in `rprob` are exactly the same. The only difference is in the initial checks for boundary conditions. This is done as follows:

```

if (observed >= x[n-1])          // If the prediction is large
    return 1.0 - rprob[n-1];      // Inspires greatest reasonable confidence
if (observed < x[0])             // If the prediction is tiny (rare!)
    return 0.0;                  // No confidence at all

```

Note that computing the tail probability via interpolation is overkill in many or most applications. It was shown here simply for completeness to cover those unfortunate (and potentially dangerous!) situations in which the data is highly granular. I generally employ the more conservative approach of using whichever of the endpoints of the interval is closer to the center of the distribution. This increases the tail area and hence decreases the confidence in the decision, though typically by only a tiny amount. The next section will assume that this more conservative method is used, although interpolation will not usually change the conclusions much at all.

Confidence in the Confidence

The quality of the confidence computations is inhibited by two sources of error. The most worrisome source of error is incomplete sampling. It is vitally important that all possible exemplars appear in the confidence set. Suppose, for example, that the goal is to detect a tank in an infrared image. A criminally careless designer might procure numerous examples of side views only. If such a collection is used for the training and target confidence sets, there will be problems. When a tank appears pointing directly at the sensor, it may well be classified as a nontarget if no representative examples appeared in the training set. When the low prediction is compared to the predictions in the target's confidence set, it may well lie far out in the left tail. This adds insult to injury, as there will be high confidence that this is not a tank. Such extreme negligence is not common, but lesser forms of inferior collection techniques are common, sometimes by accident and sometimes by laziness. Beware.

The second source of error in confidence calculations is unavoidable, though it can usually be controlled. This is random sampling error. Even if the designer is careful to give equal opportunity to all possible exemplars, bad luck may still intervene. Unknown to the designer, an inordinately large group of cases having unusually large or small predictions may find its way into the confidence set. This phenomenon can be difficult or even impossible to detect. Such an event will distort subsequent confidence calculations. The only way to protect against this situation is to collect a large enough confidence collection that any such anomalies will be diluted. How much error in the confidence might we expect from this uncontrollable random sampling error? It can be quantified with either of two simple mathematical tests.

The first method for estimating the degree of potential error in a confidence calculation is similar to a technique that we've seen before. On page 37 we learned how to compute confidence bounds for quantiles. Review that section if it is not familiar. In that section, we were interested in bounding confidence intervals for prediction errors. The same technique can be used to bound confidence in classifications obtained by thresholding predictions.

Suppose for the moment that we have collected a confidence set containing n cases from the target class. We choose a small probability p and compute $m=np$. We then take the m 'th smallest prediction in the confidence set and use it as our classification threshold. Subsequent cases whose prediction is less than this value will be classified as nontargets, with the understanding that if the case truly is a target, there is only probability p that a prediction this small, which naturally results in misclassification, will be obtained. Our fear is that bad luck in collecting the confidence set results in the actual misclassification probability being some value q that exceeds p . We need to quantify the extent of this potential problem.

In particular, we need to answer the following question: What is the probability that the m 'th smallest prediction in the target confidence set, which we have chosen for our classification threshold, is really the q quantile of the distribution, where q is disturbingly larger than the p we desire? We hope that this probability is small, because the event of q exceeding p is a serious problem in that it implies that the likelihood of misclassification is worse than we believe. This probability question is answered by the *incomplete beta distribution*. In particular, in a collection of n cases, the probability that the m 'th smallest will exceed the quantile of order q is given by $1 - I_q(m, n-m+1)$.

As discussed in the “Confidence Bounds for Quantiles” section beginning on page 37, the bounds are symmetric, meaning that exactly the same method can be used for examining pessimistic quantile bounds for high thresholds such as would be used in a nontarget confidence set. Also in that section we saw three subroutines on my web site that can be useful for these calculations. These are `ibeta` (`param1`, `param2`, `p`) for the incomplete beta distribution, `orderstat_tail` (`n`, `q`, `m`) for the tail probability, and `quantile_conf` (`n`, `m`, `conf`) for solving the inverse problem. See that section for more details.

The technique just described is useful for quantifying potential problems with a classification threshold. However, much of this chapter has been concerned with the more general situation of computing a confidence in a decision, an actual number that, while not a probability of class membership, does nevertheless express confidence in a decision. As was the case for a classification threshold, this confidence calculation is based on predictions obtained in a sample called the confidence set. How sure can we be that our computed confidence accurately reflects the probability that an observed case arose from a hypothesized class? We now explore this subject.

The population (target or nontarget) from which the confidence collection is drawn has a *cumulative distribution function* (CDF) that we will call $F(x)$. By definition, when a randomly selected case from this population is presented to the model, the probability of getting a prediction less than or equal to any specified value x is $F(x)$. We do not know this function, but we approximate it by collecting a representative sample and using its CDF as shown in a previous section.

Let $S_n(x)$ be the proportion of the collection of n cases that are less than or equal to x . Suppose we let x traverse its entire range, and we compute the maximum difference between the true CDF and that defined by the collection. This worst error is shown in Equation (2.5).

$$D_n = \max_x |S_n(x) - F(x)| \quad (2.5)$$

It is remarkable that the distribution of D_n does not depend on the nature of either the true or the sampled CDF. This facilitates a popular statistical test called the *Kolmogorov-Smirnov test*. For some typically small probability α , there exists a constant d_α such that there is a probability of only α that the worst error in the sample CDF exceeds d_α . This is expressed in Equation (2.6).

$$P\{D_n = \max_x |S_n(x) - F(x)| > d_\alpha\} = \alpha \quad (2.6)$$

The key point is that Equation (2.6) holds no matter what the true CDF is. We can rearrange this equation to cast it in a form that is immediately useful to the task at hand. Equation (2.7) shows how, given an observed sample CDF, we can find a probabilistic bound on the true CDF.

$$P\{S_n(x) - d_\alpha \leq F(x) \leq S_n(x) + d_\alpha\} = 1 - \alpha \quad (2.7)$$

Note that this equation provides confidence bounds for the *entire* distribution. Once we have acquired the confidence collection, we can specify a satisfactorily small α and then find the corresponding error limit.

Exact calculation of d_α from α is very complex, but a good approximation exists. This is shown in Equation (2.8).

$$d_\alpha \approx \sqrt{\frac{-\ln\left(\frac{\alpha}{2}\right)}{2n}} \quad (2.8)$$

In the tails, which is where we operate, this approximation is very good if n exceeds about 35, and it is almost exact if n exceeds 100. Since practical confidence sets will always be at least this large, and preferably much larger, there should be no problem using this equation.

Here is an example of this technique. Suppose we collect 100 cases for the confidence set. We decide that we will be satisfied if there is a 95 percent probability that the true confidence is within the computed bounds. This implies that α is 0.05. Equation (2.8) gives an error limit of 0.136. So we can be 95 percent sure that no matter what a computed confidence is, the actual confidence is within 0.136 of the computed value. Now suppose that an unknown case arrives and we use the hypothesis-testing technique to compute a confidence of, say, 80 percent. We can be 95 percent certain that the correct confidence figure, which we would know if we knew the true CDF, is no less than 66.4 percent, and no greater than 93.6 percent. This relatively wide range should inspire us to acquire a confidence collection that is as large as possible. Quadrupling the size of the collection will halve the confidence range.

One possible disadvantage of the Kolmogorov-Smirnov method is that absolute deviations between the empirical CDF and the true CDF usually reach their maximum near the center of the distribution. This test employs the same width for confidence bands across the entire range of the predicted value. Thus, the confidence interval for

confidences is dominated by behavior near central values of the predictions, while many users are mostly interested in extreme predictions. This can even lead to anomalous behavior. For example, suppose the deviation computed by Equation (2.8) is 0.13, and our tail p-value is 0.04, meaning that our confidence in rejecting the null hypothesis is 0.96, plus or minus 0.13. It's fine to talk about a lower bound of $0.96 - 0.13 = 0.83$, or 83 percent. But an upper bound on the confidence of $0.96 + 0.13 = 1.09$ makes no sense, since a confidence cannot exceed 1.0. Despite this problem, Kolmogorov-Smirnov bounds are useful and should be computed in any critical application.

In many situations, we are not particularly interested in an upper bound on the confidence. As was the case for a pessimistic bound for the quantile associated with a decision threshold, we may be interested in only a lower bound on the confidence, but unlike the case for a single decision threshold, we want a universal lower bound, one that is valid for the entire range of possible predictions. In this case, one would double the tail p-value and use it in Equation (2.8). For example, let's again consider the situation of 100 cases in the confidence set and a desired confidence interval of 95 percent. A few paragraphs ago we found that $\alpha = 0.05$ used in Equation (2.8) gave an error limit of plus or minus 0.136. But if we consider only errors in one direction, we set $\alpha = 2 * 0.05 = 0.1$ and use this in Equation (2.8). This gives a distance of 0.122. So we can be 95 percent sure that no matter what a computed confidence is, the actual confidence is no more than 0.122 below the computed value. Observe that the bound is slightly tighter than the two-sided bound, which is always nice.

We end this section with a discussion of a small program supplied on my web site. The source code can be found in the file `CONFCONF.CPP`, and the statistical routines that it calls are in `STATS.CPP`. The program demonstrates the two confidence bound algorithms just discussed. It is called as follows:

```
ConfConf  ncases  pval  conf  nreps

ncases - Number of cases in the sample
pval - Probability value (<0.5) for quantile test
conf - Desired confidence value (<0.5) for both tests
nreps - Number of replications
```

To obtain the numbers used in the examples a few paragraphs ago, as well as a few other numbers, we could call the program as follows:

```
ConfConf  100  0.1  0.05  100000
```

CHAPTER 2 ASSESSMENT OF CLASS PREDICTIONS

This call specifies that the confidence set contains 100 cases. The rejection threshold is defined as that which provides a 0.1 probability of erroneous classification. We also specify a probability of 0.05 for both the pessimistic bound on the threshold and the Kolmogorov-Smirnov universal bound. Correct operation is verified by performing 100,000 simulations. The following output is produced:

If the dataset represents the null hypothesis, the threshold for rejecting the null at $p=0.1000$ is given by the 91'th order statistic.

This is a conservative estimate of the 0.9000 quantile There is only a 0.0500 chance that it will really be the 0.8482 quantile or worse.

If the dataset represents the alternative hypothesis, the threshold for rejecting the alt at $p=0.1000$ is given by the 10'th order statistic.

This is a conservative estimate of the 0.1000 quantile

There is only a 0.0500 chance that it will really be the 0.1518 quantile or worse.

KS thresholds: two-tailed KS = 0.1358 one-tailed KS = 0.1224

Point failure (expected=0.0500) Lower=0.0502 Upper=0.0498

KS failure: two-tailed = 0.0464 NULL = 0.0476 ALT = 0.0459

We see from this output that the pessimistic bound for rejecting the null hypothesis is 0.8482 rather than the assumed 0.9, with this happening at the specified probability of 0.05. In other words, if we use the 91th order statistic as our rejection threshold, which would on average give us a false rejection rate of about 0.1, there is a 5 percent chance that the actual false rejection rate could be as high as $1.0-0.8482=0.1518$ or worse. The symmetric result is obtained for rejecting the alternative hypothesis (which we often called the target set).

For the Kolmogorov-Smirnov test, this output shows that there is a 95 percent chance (1.0 minus our specified 0.05) that the correct confidence figure obtained from a test case will be the computed confidence plus or minus 0.1358. Alternatively, there is a 95 percent chance that the actual confidence will be at least the computed confidence minus 0.1224.

Finally, 100,000 replications of a simulated sample showed that the failure rates for the pessimistic bounds are almost exactly the expected value of 0.05. Failure rates for the three Kolmogorov-Smirnov possibilities are all slightly less than the 0.05 that we specified. This small inaccuracy is because of the combination of the facts that the $m=np$ order statistic is slightly biased in the conservative direction, and Equation (2.8) is only an approximation.

Bayesian Methods

When confidence collections are available for both the target and nontarget classes, it may seem silly to compute and examine *two* confidence figures. This is especially true when we observe that the two computed confidences do not generally sum to one. What does it mean to say that there is a 10 percent confidence that this case is a target and a 15 percent confidence that it is a nontarget? (Actually, it means a lot, and this topic is pursued on page 88.) It would seem to make more sense to compute a single confidence number that handles the other alternative by implication. In other words, we want to be able to say that there is an 80 percent chance that the case is a target, with this figure implying that there is a 20 percent chance that it is a nontarget. This ability is often useful. However, it will be seen that some disturbingly arbitrary and possibly dangerous assumptions are involved. Also, some important information is sacrificed. With these caveats, we now explore how to use Bayesian methods to compute a single confidence figure for a target/nontarget classification.

Even though this section focuses on binary classification, we will start by stating the multiple-class version of Bayes' theorem. This version will be referenced later. Suppose we have K mutually exclusive and exhaustive classes, and they are labeled $\{H_k, k=1, \dots, K\}$. When a case is randomly sampled from a specified class and the model's prediction is made, this prediction is a random variable. Let the likelihood function of this prediction for class k be written $L(x|H_k)$. If the distribution happens to be discrete, the likelihood is simply the probability of x occurring. In the more usual situation that the distribution is continuous, the likelihood is the density function. Finally, assume that the prior probability of class H_k is p_k , remembering that $\sum p_k = 1$. Given a case sampled from an unknown class, the probability that the case is from class H_k is given by Equation (2.9).

$$P(H_k|x) = \frac{p_k L(x|H_k)}{\sum_i (p_i L(x|H_i))} \quad (2.9)$$

We need two pieces of information for each class in order to be able to use Equation (2.9) to compute the confidence (which here is the probability that the specified class is the source of the unknown case). We need the prior probabilities, and we need the likelihood functions. They are both problematic.

It is obvious from Equation (2.9) that the prior probabilities affect the computed confidence as much as the likelihoods. This is troubling because, in many applications, the prior probabilities are subjective at best and capricious at worst. What do we do when we have no objective evidence for assigning priors? The answer to this extremely important question was given by Bayes himself in what is known as *Bayes' Postulate* (sometimes also called the *Principle of Equidistribution of Ignorance*). He states that the prior probabilities should be assumed equal unless there is clear evidence to the contrary. Ignore this advice from the ultimate expert at your own peril.

Estimating the likelihood function is much more problematic than dealing with the priors. It may rarely be the case that the model's input variables can take on only a very few discrete values, and hence the predictions will also be discrete. In this situation, the likelihoods can be computed from the confidence sets by counting the number of occurrences of each discrete value and dividing this by the total number of cases. But it will almost never happen in real life that direct probabilities can be used this way. In most cases, the inputs are either continuous or they have so many possible discrete values that they might as well be continuous. It is necessary to estimate the density function of each class. This is not a trivial undertaking.

Probably the best method for estimating a density function from a collection of observations (model predictions here) is known as *Parzen's method*. Details can be found in many statistics texts, as well as in [Masters, 1993]. Only a summary of the technique is given here. Let the collection of n predictions from a class be $\{x_i, i=1, \dots, n\}$. The density function of the population from which this collection was drawn can be estimated by Equation (2.10).

$$f(x) = \frac{c}{n\sigma} \sum_i e^{-\left(\frac{x-x_i}{\sigma}\right)^2} \quad (2.10)$$

In this equation, c is a constant that is needed only to ensure that the density estimator integrates to unity. It does not concern us here because it appears in both the numerator and the denominator of Equation (2.9), and so it will cancel. We will ignore it. The problem is σ (sigma), the smoothing constant. Suppose we choose to let σ have a very tiny value. The sum in Equation (2.10) will be dominated by whichever x_i happens to be closest to x . All other summands will be negligibly small. The density estimate will consist of nothing more than a series of peaks, each corresponding to a case in the collection. This is clearly not satisfactory. Suppose instead that we choose to let σ have an extremely large value, many times greater than the variation within the collection. The density

estimator will now have about the same relatively large value for any x near the collection, dropping off toward zero only when x is very far from any case in the collection. The contributions of the individual cases in the collection are largely ignored. This, too, is obviously not satisfactory. It is necessary that σ be some moderate value, large enough that no individual case dominates, but small enough that individual cases do make significant contributions to the density across the domain.

How is a good value of σ chosen? The goal is that for every possible x , some cases in the collection that are nearest to x make a significant contribution, a few more distant cases make a modest contribution, and the remainder of the collection is essentially ignored. This admittedly vague goal can usually be achieved by letting σ be somewhere between 0.05 times the standard deviation of the collection, up to perhaps 0.4 times the standard deviation. If the collection is very large (n is many hundreds of cases) and reasonably normal, the smaller limit is appropriate because it will squeeze maximum information from the individual cases. If the collection is not this large or if the distribution contains outliers, a large value of σ will provide better smoothing and more conservative behavior in the tails. In general, values of σ greater than about 0.4 times the standard deviation will blur the density excessively, while values much less than 0.05 times the standard deviation will produce unnatural peaks in the density estimator. Smaller values are appropriate only if the collection contains thousands of cases.

If the collection happens to contain a few relatively isolated predictions (not good!), it may be useful to manually compute the contribution of individual cases in advance. More importantly, it is informative to see how far away cases must be in order to reduce their contribution to a specific degree. Let d be the difference between the point x at which the density is being estimated and a case x_i in the collection. The contribution of this case to the sum in Equation (2.10) is given by Equation (2.11). Solving this for d gives Equation (2.12).

$$w = e^{-\left(\frac{d}{\sigma}\right)^2} \quad (2.11)$$

$$d = \sigma \sqrt{-\ln(w)} \quad (2.12)$$

A case exactly equal to the point x at which the density is being estimated will provide the maximum possible contribution to the sum, 1.0. Suppose we let $w=0.5$, implying that we are interested in seeing how far away from x a sample case must be in order for its contribution to drop to one half of that of a case exactly at x . This, of course, depends on σ . Assume that we have chosen σ to be 5.0 (a reasonable choice

if the standard deviation happens to be from 50 to 100 or so). Equation (2.12) tells us that $d=4.16$. So for any given x , collection cases that happen to be at $x-4.16$ and $x+4.16$ will have half the contribution of a case that happens to be exactly at x . To take it a bit further, let $w=0.1$. We find that $d=7.59$. Thus, cases more distant from x than 7.59 will make an almost negligibly small contribution to the density estimate. This sort of information can be useful because it allows us to confirm that there are no “holes” in the collection. If, for example, the collection contained no cases with predictions from, say, 10 through 30, density estimates around $x=20$ would be extremely poor (near zero!). Sigma should be at least doubled, changing from 5 to 10. Values even larger would not be unreasonable.

There is a more rigorous and objective method for computing an optimum σ (or pair of sigmas if separate values are desired for the target and nontarget collections). This method may be overkill, and it may be dangerous to completely remove the experimenter from the selection process. Nevertheless, automation is handy, and the method appears to work well.

This method is based on defining a performance criterion that rates the effectiveness of any given σ (or pair). Once this criterion is available, any reasonable numeric optimization algorithm may be used to find the best sigma weight(s). An excellent way to rate this performance is to use cross validation. Here is the algorithm:

- 1) Remove the first case from the nontarget collection. Temporarily consider it an unknown.
- 2) Use Equation (2.10) to estimate the density at this “unknown” case in the nontarget collection and also in the target collection.
- 3) Use Equation (2.9) to compute the confidence that this case is a nontarget. It truly is a nontarget, so we ideally want the confidence to be exactly 1.0. It will generally be smaller than this. The difference is the error attributable to this case. Square this error and cumulate it into a sum.
- 4) Replace this case and now remove the second case in the nontarget collection. Use the two previous steps to compute its error and cumulate the total squared error.
- 5) Repeat this process for all cases in both the target and nontarget collections. Remember that for cases in the target collection, we want the target confidence to be 1.0, so compute the error accordingly.

The mean squared error across all cases appears to be an excellent indicator of the quality of the σ weight(s) used in Equation (2.10). In most practical applications, this function has a delightful bowl shape across a reasonable range of σ weights, so optimization is typically straightforward. Multiple broad local minima do sometimes appear, so a rough initial global search is needed to find the vicinity of the global minimum. However, local behavior of the error function is usually excellent.

In summary, here is how Bayesian confidences can be computed if we have large representative collections of cases from both the target and nontarget populations: The first step is to determine a reasonable sigma parameter for each population. The same value does not have to be used for both, although in practice they will usually be similar if not identical. If possible, make a graphical plot of the computed density functions. These plots may be interesting and informative. When an unknown case arrives, use the model to compute the prediction for this case. Apply Equation (2.10) to both the target and nontarget confidence collections to find the density estimate for each. Finally, plug these two values into Equation (2.9) to get the confidence figures. Note that this equation guarantees that the target and nontarget confidences will sum to one.

Do not be surprised if sometimes the predicted value relative to a preordained threshold gives a classification that has *lower* confidence than the other class. This can happen when the prediction is not clear cut because of bumpy density functions, especially if the relative error costs and/or prior probabilities are disparate. We'll see an example of this in Figure 2-4 on page 90.

Multiple Classes

So far, the subject of confidences has been limited to the simple case of target versus nontarget classification. This is by far the most common application. Nevertheless, it is sometimes necessary to be able to predict membership from among three or more classes. Confidence measurements become correspondingly more difficult in this case. The two methods already discussed can be generalized to more than two classes. However, issues become complex rapidly.

For the hypothesis testing method, we need a confidence collection for each class for which we want to be able to reject membership. An immediate problem is that rejection of one class no longer automatically implies confidence in the other class. For example, suppose we want to classify a tissue sample as benign, malignant, or atypical. If we have

a good collection of atypical cases, we may be able to reject membership in the atypical class. But that alone tells us nothing about benign versus malignant. We need to judge this probability in the context of the other probabilities of rejection.

The problem gets even worse when one considers the fact that effective multiple-class classification almost always requires separate predictions for each class. The usual method is to train each prediction to be high for exactly one class, and low for all others. This means that we have the option of considering rejection probabilities for more than one distribution. The general consensus is that when more than two classes are involved, hypothesis testing is not a good choice for confidence computation.

There is an excellent way to use Bayes' method to compute confidences in multiple-class predictions. Train a single model with multiple outputs or a set of models, each making a prediction for a single class. The dependent variable is a large value for cases in that class, and a low value for cases in all other classes. Thus, the confidence sets will consist not of scalar predictions, but of vector predictions, with each vector having a length equal to the number of classes. Then, use these confidence sets as a training set to train a *probabilistic neural network* (PNN) to do the classification. A PNN intrinsically has confidences as its outputs, so it is ideal for this application. When an unknown case arrives, apply the prediction models in order to procure a vector of class predictions. Then submit this prediction vector as an input to the trained PNN. The outputs of the PNN are the computed class confidences. This technique has proven itself to be effective in many applications, and it is probably the most straightforward way of handling the multiple-class confidence problem. Probabilistic neural networks are beyond the scope of this text. They are introduced in [Specht, 1990a] and covered in depth in [Masters, 1995a]. A full discussion of confidence computation using PNNs can be found in [Masters, 1993].

Hypothesis Testing vs. Bayes' Method

Two very different techniques for computing classification confidences have been presented: hypothesis testing and Bayes' method. These confidences have dramatically different theoretical foundations, and they provide very different types of information. A comparison is in order.

First, it must be understood that we sometimes do not have the luxury of choosing between the two methods. The Bayesian method is much more stringent in its requirements. For hypothesis testing, a representative collection of cases needs to be available only for the hypothesis that we will attempt to reject in favor of the alternative. If one of the classes is not able to be adequately represented, we can still proceed with

what we have. For Bayes' method, both the target and nontarget classes must be fully represented. Moreover, the quality of the representation is more critical for Bayes' method than for hypothesis testing. Suppose a collection has an empty area in its distribution, a range of predicted values that, due to sampling error, contains few or no cases. We will be faced with an uncomfortable choice. Either σ will need to be chosen large enough to bridge the gap, causing excessive blurring and distortion everywhere else, or the computed density will be incorrectly tiny in this region, leading to incorrect confidence decisions later. Hypothesis testing uses no local information, only case counts from interior areas out to the end of the sorted collection. This makes it more tolerant of sampling problems.

Assuming that all necessary criteria have been met, let us now compare the meaning of confidence computed with hypothesis testing versus that computed using Bayes method. Figure 2-2 shows the estimated densities for a moderately good model. The left curve is the density function for the nontarget class, and the right curve is that for the target class. Where the two curves intersect at about 0.1, the Bayesian confidence in both classes is the same, 0.5 (assuming equal prior probabilities). Moreover, the area under the left curve to the right of this point is roughly equal to the area under the right curve to the left of this point. Thus, the hypothesis-testing confidences in both classes are also approximately equal, moderately large but nothing to get excited about. If a prediction less than about -0.75 appears, the Bayesian confidence in the nontarget class will clearly be essentially 1.0 because the target density is insignificant there. The area under the target curve to the left of this point is also just about zero, implying that the hypothesis-testing confidence is also nearly 1.0. It should be clear that no surprises lurk in this example.

Now look at Figure 2-3. This illustrates the density curves for a model with almost impossibly good performance. Predictions less than about -0.1 or greater than about 0.1 result in both confidence methods giving identical perfect confidence estimates. Still no surprises. But what if a prediction comes in just above zero, where the two density curves cross? The Bayesian confidences will be 0.5 for both classes. However, the hypothesis-testing confidences will be practically 1.0 for *both* classes! This is because *both* the target and nontarget hypotheses are rejected. The probability of a prediction of this value is nearly zero in both the target and nontarget populations. This conflict has great importance, as we will soon see.

And it gets worse. Problems like this do not occur only at sparse intersection points. Suppose a case produces a prediction of -0.1. The target density here is nearly zero, so Bayes' method will produce a confidence of virtually 1.0 that this case is a nontarget. But the area under the nontarget density to the right of this point is probably less than 0.01,

meaning that hypothesis testing based on the nontarget collection will give a confidence in excess of 0.99 that this case is a target! The two methods are (definitively!) saying exactly opposite things. What's going on?

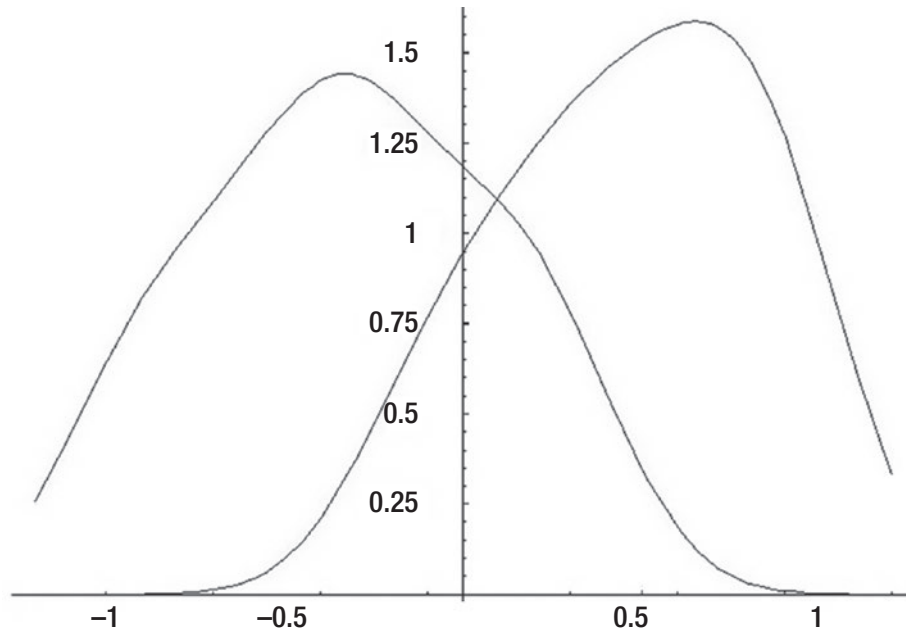


Figure 2-2. Parzen density estimates for a fairly good model

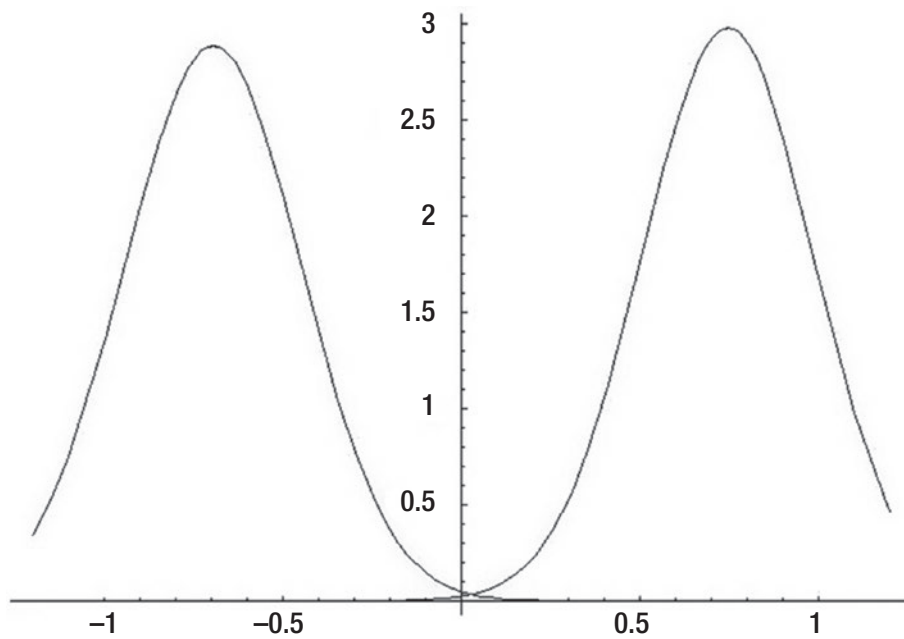


Figure 2-3. Parzen density estimates for a fantasy model

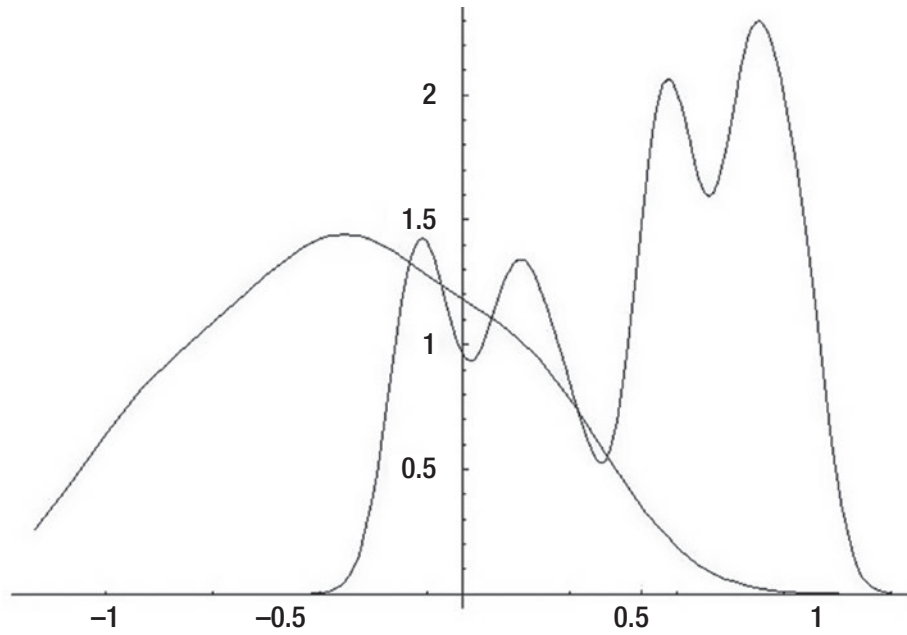


Figure 2-4. Parzen density estimates with poor targets

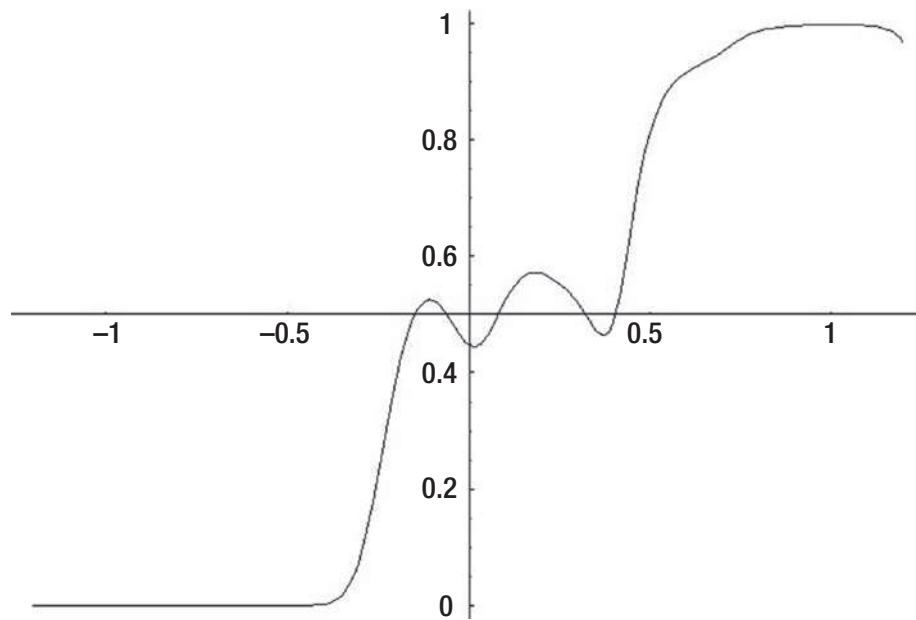


Figure 2-5. Bayes confidence when target definition is poor

To understand the root of this conflict, it is necessary to understand the assumptions implicit in each method. Hypothesis testing and Bayes' method treat the source of the case and the prediction associated with the case as occurring in opposite orders. Bayes' method treats the identity of the class that produced the case as a random event. Either the source was a target or it was a nontarget, and the occurrence was random. The prediction is a given. Based on this prediction, probabilities are computed for each class.

Hypothesis testing is based on the opposite order. Instead of the prediction producing a probability of class membership, the presumed class membership generates a probability for the obtained prediction. Hypothesis testing assumes that one of the two classes is the source of the case. There are no probabilities of class membership involved. The class membership, though unknown, is fixed. Under the assumption of fixed class membership, hypothesis testing looks at the probability of attaining the prediction in question. Other than taking part indirectly through Equation (2.9), this probability is ignored by Bayes' method. This is the key point, and this is why Bayes' method relies so heavily on the quality of the confidence sets. *Only hypothesis testing will detect the occurrence of an event that the confidence sets say is rare in both classes.*

If the experimenter is absolutely positive that both confidence sets are thoroughly representative of all possible cases, Bayes' method is probably the best choice because it provides a single number that completely describes the confidence situation. Also, it is usually legitimate to refer to Bayesian confidences as true probabilities, a powerful assertion, while this is never possible with hypothesis testing. However, if there is even the smallest possibility that an important subpopulation has been inadvertently omitted from one or both confidence sets, hypothesis testing may be important.

A good compromise approach is to provide the user with *two* confidence figures. The most important figure would be the Bayes confidence in the classification decision. The second figure would be the hypothesis-testing confidence in the *other* class. In other words, examine the attained prediction in the context of the class that was chosen. If it happens that the prediction is very far out in the tail, indicating low probability of obtaining this prediction from the chosen class, and hence high confidence in the *other* class, the one not chosen, be suspicious. Something has gone wrong in the data collection, and it will be necessary to study the confidence sets carefully.

Bayes confidence estimates occasionally suffer from a potentially confusing anomaly that is not shared by hypothesis testing. Recall that most models are trained to predict a large value (typically +1) for targets, and a small value (typically -1) for nontargets. This implies that we expect a monotonic mapping from the prediction to confidences: larger predictions map to larger confidences in the target class and smaller confidences

in the nontarget class. This is precisely what happens in hypothesis testing. As the prediction increases, fewer cases exceed it in the nontarget distribution, leading to higher confidence in the target class. Simultaneously, more cases are smaller than it in the target distribution, leading to lower confidence in the nontarget class.

Unfortunately, this is not guaranteed to happen with Bayes confidence. Look at Figure 2-4 on page 89. This illustrates a target population density that is multimodal. Densities like this can arise when the target population contains several subclasses or when the sigma parameter for the Parzen estimator is too small. Observe how the target and nontarget densities cross repeatedly. Figure 2-5 shows the associated Bayes confidence as computed with Equation (2.9) on page 81. The horizontal axis in this plot is the model's prediction, and the vertical axis is the confidence in the target class, with 0.5 (equal confidence in the two classes) used as the horizontal axis. We would hope that this curve would steadily rise, indicating increasing confidence in the target class, as the prediction increases. But it doesn't. We are faced with the peculiar situation that at some places the confidence can flip-flop, with an increasing prediction causing the nontarget class to be favored over the target class!

Is this a serious problem? Specifically, should we base our classification decision on a threshold for the prediction, or should we base it on the Bayes confidence? They may not always give the same results. There is no easy answer. If the multiple modes in one or both densities are real and meaningful, then we should probably use the Bayes confidences to decide class membership, as they reflect the fact that certain regions are expected to be sparse. But before making this rather serious decision, you had better be positive that the multiple modes are truly representative of the nature of the underlying population. If instead they arise from multiple classes within what you are treating as a single class, you should investigate closely and consider a different modeling scheme such as vector prediction trained on each class. If multiple classes are ruled out (or must be ignored for technical reasons) then a larger smoothing parameter for the Parzen window should be used.

Final Thoughts on Hypothesis Testing

Hypothesis testing as a means of computing confidence in a classification decision seems straightforward. However, its indirect method of assertion makes it very susceptible to accidental abuse. For this reason, a final lecture on some of the subtleties of the logic is in order. We begin by repeating the logic by which one asserts confidence in a target classification when the model's prediction is high and we have information

about the distribution of predictions in the nontarget population. This same logic could be inverted to assert confidence in a nontarget classification when we get a low prediction and we have information about the target distribution.

- 1) Our environment is mutually exclusive and exhaustive, so the case in question either is a target or is a nontarget. There are no other possibilities.
- 2) If we were to assume that the case is a nontarget, we would be faced with the fact that the probability of obtaining a prediction as large as or larger than that obtained is very small. In other words, if the case really were a nontarget, we would have just observed a highly unlikely event.
- 3) We thus conclude that the case is probably a target.

In many situations we can compute a quite accurate value for the probability of having observed such a large prediction from a nontarget case. The algorithm presented on page 75 provides decent estimates of this probability as long as the confidence set satisfies the required assumptions. We compute confidence in a target classification by subtracting the nontarget probability from one. So, for example, suppose we find that there is a probability of only 0.01 that a nontarget would produce a prediction this high. Subtracting this probability from one gives a confidence of 0.99 that the case is a target.

A common and potentially serious error is made if we call this 0.99 a probability. It is tempting, though wrong, to conclude that 99 percent of the time that we observe a prediction whose nontarget probability is 0.01 or less we will be correct in concluding that the case is a target. Looked at another way, consider, say, 100 instances in which we observe a case whose nontarget probability is 0.01 or less. These may be, for example, a batch of tissue samples that arrive at a pathology laboratory. Or they may be appearances of blips on a fighter aircraft radar screen. We are tempted to conclude that of these 100 occurrences of 0.01 or less nontarget probability, about 99 of them will in truth be targets. *This is terribly wrong!* The faulty logic goes like this: We just observed a case that is either a target or a nontarget. It had a high prediction, one that is very unlikely to have arisen from a nontarget. Hence, we classify it as a target. Then...

- 1) Either we are right (to call the case a target) or we are wrong. There are no other possibilities.

- 2) If we are wrong (the case really is a nontarget), we will observe a prediction this large only 1 percent of the time (0.01 probability).
- 3) Therefore there is a 99 percent probability (0.99) that we are right.

The flaw in this logic is that it has subtly reversed the order in which events occur. The correct order is that first we assume the fact that the case is a nontarget, and then we observe a prediction. We can legitimately compute a probability associated with the prediction. The phony logic just described says that we observed a prediction and then we try to compute a probability that the case is a target or nontarget. This probability can only be computed with Bayes' method, in which we have knowledge of the prediction distribution for both targets and nontargets and in which we are willing to assume prior probabilities for each class. Neither of these pieces of information is available to a hypothesis test.

If we want to subtract a probability of 0.01 from one and get a complementary probability of 0.99, this is legitimate. But we need to be careful to correctly define the meaning of this probability. It (0.99 in this example) is the probability that *when the case truly is a nontarget, we will observe a prediction less than that obtained for this case*.

To carry this further, suppose our task is to design a model that examines evidence regarding a credit card transaction that a customer is attempting to make at a distant store. If our model decides that the transaction is likely fraudulent, the store is instructed to refuse the purchase. Erroneous refusals will surely produce angry customers, so we decide to be extremely cautious. Suppose we have no information about the distribution of model predictions for the *fraudulent* class. But we do have lots of information about predictions for *nonfraudulent* transactions. In the interest of erring on the liberal side, we choose to reject a transaction only if the model's prediction is so high that its probability under the assumption of the nonfraudulent class is less than or equal to 0.001. This implies a confidence of 0.999 or more that the transaction is fraudulent. By choosing a threshold this high, we can legitimately make two statements about probabilities:

- 1) Only 0.1 percent of truly nonfraudulent transactions will be erroneously flagged as fraudulent.
- 2) 99.9 percent of all nonfraudulent transactions will be allowed to proceed.

CHAPTER 2 ASSESSMENT OF CLASS PREDICTIONS

There are two probability statements that we *cannot* make with a hypothesis test, no matter how much we would like to do so:

- 1) When we obtain a prediction so high that it reaches the 0.001 probability threshold, and hence we decide to forbid the transaction, we will be wrong to do so only 0.1 percent of the time.
- 2) When the model's prediction does not reach the 0.001 threshold and hence we allow the transaction to proceed, we will be wrong only 0.1 percent of the time.

The preceding two assertions reverse the order in which the probability is computed: a decision is made, and then a probability associated with the decision is asserted. This is incorrect. Most people would never make the mistake of believing the second incorrect statement. But an amazing number of people believe the first. The appeal of this illegal logic is strong; a clear-cut decision to reject the null hypothesis has just been made, and now you want to know the probability that the decision is correct. So, the loosely defined “confidence” in the decision is treated as if it's a probability. It happens all the time, and it's just plain wrong.

If you are comfortable with the preceding discussion and feel you solidly understand hypothesis testing, you may safely skip the remainder of this section. But if you remain confused, we will dig deeper still. The main thrust so far has been to explore the use of hypothesis tests to compute confidence figures for classification decisions and to vigorously proclaim that these confidences must not be interpreted as probabilities. But hypothesis testing does allow some probabilities to be asserted for classification decisions. The remainder of this section discusses the legality and illegality of such assertions.

We begin with a presentation of the official version of hypothesis testing as a means of indirectly making a claim. In practice, this may involve testing a weight-loss drug to decide if it is effective, or testing the profitability of a proposed trading system to see if it can make money. In the context of this chapter, hypothesis testing involves using a model's numeric prediction to make a claim about membership or nonmembership in a class. We recognize that random variation precludes all such claims from being definitive. Probabilities are involved. Our goal is to make as many probability claims as possible and to revert to weaker claims when necessary. Here is the official route of a hypothesis test:

- 1) We define a *null hypothesis*, which is often a “status quo” situation, though it need not be. We also define an *alternative hypothesis*, which is often a special situation on which we focus. In pharmaceutical research, the null hypothesis may be that a new drug is worthless, and the alternative hypothesis is that the drug works. In market trading the null hypothesis may be that there is no compelling reason to open a position now, while the alternative is that now is a good time to do so. Or the null hypothesis may be that the set of trades in a historical sample of a trading rule’s activity represents the results of a worthless rule, while the alternative hypothesis is that these trades arose from a good rule. Often, the null hypothesis is simple and completely specified, implying that the distribution of test statistics (model predictions in the current context) is relatively easy to obtain. In contrast, the alternative hypothesis is often incompletely specified, making the alternative distribution difficult or impossible to obtain.
- 2) We agree that the null and alternative hypotheses are mutually exclusive and exhaustive, meaning that they cannot be true simultaneously and that together they cover all possible situations.
- 3) We choose in advance a small significance level usually called *alpha*. This will be, by definition, the probability that, IF the null hypothesis is true (emphasis on IF), we will erroneously reject it. Since we know the distribution of our statistic (the model’s prediction here) under the null hypothesis, choice of alpha implies a threshold for our statistic. So we agree to reject the null hypothesis in favor of the alternative if the probability that we would observe a test statistic as extreme as what we actually observed is less than or equal to alpha. Equivalently, we reject if our test statistic is at or beyond the threshold. Note that one minus alpha is the probability that IF the null hypothesis is true, we will correctly fail to reject it. We saw how to estimate this probability on page 75. Also note that many people do not choose alpha in advance. Instead, they act as if the obtained probability had been exactly chosen in advance. The implications of this minor form of cheating are beyond the scope of this text.

So, for example, suppose we are willing to live with erroneously rejecting five percent of all null-hypothesis cases. In other words, we are willing to erroneously classify into the alternative-hypothesis class 5 percent of all cases that truly belong to the null-hypothesis class. If our confidence set consists of 100 cases (too small!), we would choose the fifth-largest prediction in this set of null-hypothesis cases. This would be our threshold for future classification decisions. In the language of this chapter, we classify a case as a member of the alternative class if and only if we have at least a 95 percent confidence in the decision. Subject to the sources of random variability already discussed, this would result in us correctly classifying about 95 percent of all future cases *that truly belong to the null-hypothesis class*. Only 5 percent of the members of this class would be incorrectly classified.

However, it must be strongly emphasized that we cannot make the converse assertion. Suppose a case elicits a prediction that meets the 95 percent confidence threshold. After assigning this case to the alternative-hypothesis class, we *cannot* assert that there is a 95 percent probability that we were correct to do so. That $100 - 5 = 95$ percent probability of making a correct decision is the probability that we will correctly classify members of the null-hypothesis class. We know *nothing* about the behavior of the model under the alternative hypothesis, and hence we cannot make any general assertions about our ability to classify arbitrary cases. Our probability assertions are limited to members of the null-hypothesis class. If we do have knowledge of the model's behavior under the alternative hypothesis, we should probably be using Bayes' method as our primary decision maker. In the absence of this knowledge, the best we can do is say that we have a 95 percent *confidence* in our decision, a deliberately vague claim.

At the risk of being pedantic, let's make one last attempt to clarify this frequently confusing issue by means of a final example of hypothesis testing. Suppose we work in the quality-control department of a manufacturing facility. A particular gearbox is crucial to operation of the factory. Periodic maintenance is needed to keep this gearbox from failing catastrophically. This maintenance requires that the entire assembly line be shut down, so we want to put it off for as long as possible. Unexpected failure is troubling but not a disaster. A more serious problem is unneeded shutdowns for premature maintenance. Until you were hired, management simply let it run until the gearbox failed and then did the repairs. But you were hired to try to improve the situation. After some experimenting, you discover that the gearbox begins to emit abnormal noises when failure is immanent. So, you rig a sensor to the gearbox and feed its signal to a model whose output is trained to produce a large value when any of several unusual sounds appear. Normal sounds cause the model to make a small prediction.

Two things are immediately apparent. First, you will have no problem amassing a huge collection of model outputs for normal conditions, while it will be difficult to collect more than a few examples of the rare and varied abnormal noises. You barely have enough exemplars for your training and test sets. You certainly don't have enough for any kind of confidence set for the abnormal state.

Second, you see that failure to detect an impending breakdown is not a terribly serious problem. After all, this error simply brings the factory back to how it has operated all along: waiting for failure and then fixing the gearbox. But the opposite error, predicting failure too soon and hence causing premature shutdown of the factory, will be frowned upon by management. You wisely decide to be conservative in your decision.

With this in mind, you collect 1,000 samples of normal sounds, present each sample to your trained model, sort the model's predictions, and choose the tenth-largest prediction as your decision threshold. This means that you will call for shutdown and maintenance when the model provides an output so large that only 1 percent of normal samples exceed it. In the language of this chapter, you will flag immanent failure when your confidence in this decision is at least 99 percent. If you perform the test every morning, then one out of every 100 mornings in which the gearbox is normal you will incorrectly flag impending failure.

So far this example has been straightforward. Here's the monkey wrench. Suppose you are in charge of two such gearboxes. The new foreman in charge of one of them is a real cheapskate. When he replaces parts, he uses junk that typically fails in as few as ten days or so. The foreman in charge of the other gearbox goes to the other extreme. He just discovered a new titanium gear that is guaranteed for 50 years, and he put it in at the last repair. You, of course, haven't been made aware of any of this. The normal sound of both gearboxes is still the same. In fact, the failure sounds will also be about the same. Only the life of the gears is different. Very, very different.

Consider the probability of a correct decision when your model produces a prediction sufficiently high to reach the 99 percent confidence threshold and you therefore choose to sound the alarm. In the case of the cheapskate's gearbox, you will virtually always be correct to do so. If your model incorrectly flags impending failure only once out of every 100 or so good-condition days, but the gearbox fails every ten days, those (frequent!) times in which you flag failure will almost always be correct decisions. In fact, the probability of this being a correct decision will most likely exceed the confidence level of 99 percent! But the gearbox with the titanium gears will, for all practical purposes, never fail. So when your model unavoidably flags impending failure, it will always be wrong. You will have an essentially zero probability of having made the correct decision, despite the fact that you used the same correctly trained model on both gearboxes.

This example ideally makes clear the fact that a confidence figure for rejecting a null hypothesis cannot be interpreted as being the probability of having made a correct decision. It can be wildly inaccurate because of its dependence on the distribution of the prediction under the alternative hypothesis as well as prior probabilities. It's the widely disparate prior probabilities that caused problems in this example. Be warned.

Confidence Intervals for Future Performance

Once we have obtained a classification model that performs well, we will almost surely be interested in estimating how well it will do in the future. To estimate this future performance, we need a large collection of cases that satisfy the usual conditions for a confidence set. These conditions were set forth on page 73. Of course, all classes must be represented in the proportion in which they are expected in the future.

On page 35 we saw how empirical quantiles could be used as confidence intervals for future errors in numerical prediction. The situation is often more complex for classification, because performance for individual cases is often more granular than performance for numerical predictions. It may occasionally be that class decisions have continuous or nearly continuous results, such as returns for a market trading system. But more often, classification performance is binary: the predicted class is correct or incorrect. There may be numerical cost (or reward) values associated with decisions, but it is often the case that these costs are fixed, with one cost for an error and another for a correct decision. It makes no sense to talk about confidence intervals for individual case errors in this situation.

The granularity problem can be somewhat solved by grouping cases into sets whose mean performance (or other measure) is computed. For example, suppose we have a manufacturing process that produces two different grades of a product, with the grade of each sample being more or less random, depending on humidity, the mood of the control operator, and so forth. So there is a sensor monitoring the end of the line, and this sensor drives a model that grades each emerging sample. A correct grade incurs no cost. Misclassifying a truly Grade A product as Grade B incurs one cost, while the opposite error incurs another cost. There are three possible values associated with each test of a sample of the product: zero for a correct classification, and two different error values. It obviously makes no sense to talk about a confidence interval for a single decision by the grading model.

However, we could look at sets of ten samples and compute the mean cost incurred by the model per set of ten. If necessary, we could even bundle them in groups of 100 or more. This provides many more possible cost figures. Now we could use the methods presented earlier to compute confidence bounds for the mean cost per set of samples.

In particular, when we have such a collection of group performances, the technique of using empirical quantiles as confidence intervals, discussed on page 35, may be used to find confidence bounds for future costs or gains. Pessimistic bounds on the confidence bounds may also be computed. Even tolerance intervals may be computed, although they are less commonly useful. The main use of empirical quantiles in this context is to compute an approximate lower confidence bound for future gains or an upper bound for future loss.

One important use for empirical confidence intervals for future performance is in regard to ongoing verification of the model's correct operation. Suppose, for example, that we have determined that there is only a 10 percent probability that future gains will be less than -1000 (presumably a serious loss). Now suppose a string of actual losses worse than -1000 occurs. We should immediately suspect that the model is deteriorating. This is especially important when one is operating on a nonstationary time series.