CHAPTER 4

COMBINING LABEL OUTPUTS

How do we combine the outputs of the individual classifiers in the ensemble? Numerous theoretical analyses [145,207,235,241,249,260,394], experimental comparisons [109,152,222,382,434,436] and reviews [392,439] look for the answer to this question.

4.1 Types of classifier outputs

Consider a classifier ensemble consisting of L classifiers in the set $\mathcal{D} = \{D_1, \ldots, D_L\}$ and a set of classes $\Omega = \{\omega_1, \ldots, \omega_c\}$. Xu et al. [425] distinguish between three types of classifier outputs

- *Class labels.* (The Abstract level.) Each classifier D_i produces a class label $s_i \in \Omega$, $i = 1, \ldots, L$. Thus, for any object $\mathbf{x} \in \mathbb{R}^n$ to be classified, the L classifier outputs define a vector $\mathbf{s} = [s_1, \ldots, s_L]^T \in \Omega^L$. At the abstract level, there is no information about the certainty of the guessed labels, nor are any alternative labels suggested. By definition, any classifier is capable of producing a label for $\mathbf{x}$, so the abstract level is universal.

By Ludmila I. Kuncheva Copyright © 2014 John Wiley & Sons, Inc.

- *Ranked class labels.* The output of each D_i is a subset of the class labels Ω, ranked in order of plausibility [184, 391]. This type is especially suitable for problems with a large number of classes, such as character, face and speaker recognition.

- *Numerical support for the classes.* (The Measurement level.) Each classifier D_i produces a c-dimensional vector $[d_{i,1}, \ldots, d_{i,c}]^T$. The value $d_{i,j}$ represents the support for the hypothesis that the vector $\mathbf{x}$ submitted for classification comes from class ω_j. The outputs $d_{i,j}$ are functions of the input $\mathbf{x}$, but to simplify the notation we will use just $d_{i,j}$ instead of $d_{i,j}(\mathbf{x})$. Without loss of generality, we can assume that the outputs contain values between 0 and 1, spanning the space $[0, 1]^c$.

We add to this list one more output type:

- *Oracle.* The output of classifier D_i for a given $\mathbf{x}$ is only known to be either *correct* or *wrong*. We deliberately disregard the information as to which class label has been assigned. The oracle output is artificial because we can only apply it to a labeled data set. For a given data set $\mathbf{Z}$, classifier D_i produces an output vector $\mathbf{y}_i$ such that

$$y_{ij} = \begin{cases} 1, & \text{if } D_i \text{ classifies object } \mathbf{z}_j \text{ correctly,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

4.2 A probabilistic framework for combining label outputs

Consider class label outputs $\mathbf{s} = [s_1, \ldots, s_L]^T \in \Omega^L$. We are interested in the probability

$$P(\omega_k \text{ is the true class } |\mathbf{s}), \quad k = 1, \ldots, c,$$

denoted for short $P(\omega_k|\mathbf{s})$. Assume that the classifiers give their decisions independently, conditioned upon the class label. Conditional independence means that[1]

$$P(s_1, s_2, \ldots, s_L|\omega_k) = P(s_1|\omega_k)P(s_2|\omega_k) \ldots P(s_L|\omega_k).$$

Therefore we can write

$$P(\omega_k|\mathbf{s}) \quad = \quad \frac{P(\omega_k)}{P(\mathbf{s})} \prod_{i=1}^{L} P(s_i|\omega_k). \tag{4.2}$$

Split the product into two parts depending on which classifiers suggested ω_k. Denote by I_+^k the set of indices of classifiers which suggested ω_k, and by I_-^k the

[1]However, this assumption precludes unconditional independence, that is,

$$P(s_1, s_2, \ldots, s_L) \neq P(s_1)P(s_2) \ldots P(s_L).$$

set of indices of classifiers which suggested another class label. The probability of interest becomes

$$P(\omega_k|\mathbf{s}) = \frac{P(\omega_k)}{P(\mathbf{s})} \times \prod_{i \in I_+^k} P(s_i|\omega_k) \times \prod_{i \in I_-^k} P(s_i|\omega_k). \tag{4.3}$$

This decomposition allows us to define the optimality conditions for several combination rules [244]. Optimality is understood in a sense that the combiner guarantees the minimum possible classification error.

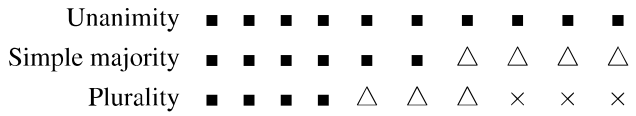
4.3 Majority vote

Never underestimate the power of stupid people in large groups.

- George Carlin

4.3.1 'Democracy' in classifier combination

Dictatorship and majority vote are perhaps the two oldest strategies for decision making [84, 159]. Three consensus patterns: unanimity, simple majority and plurality, are illustrated below. Assume that shapes correspond to class labels, and the decision makers are the individual classifiers in the ensemble. The final label in all three consensus patterns is ■



Assume that the *label* outputs of the classifiers are given as c-dimensional binary vectors $[d_{i,1}, \ldots, d_{i,c}]^T \in \{0,1\}^c$, $i = 1, \ldots, L$, where $d_{i,j} = 1$ if D_i labels $\mathbf{x}$ in ω_j, and 0, otherwise. The *plurality vote* will return class ω_k if

$$\sum_{i=1}^{L} d_{i,k} = \max_{j=1}^{c} \sum_{i=1}^{L} d_{i,j}. \tag{4.4}$$

Ties are resolved arbitrarily. This rule is often called in the literature *the majority vote*. It will indeed coincide with the simple majority (50% of the votes +1) in the case of two classes ($c = 2$). Xu et al. [425] suggest a thresholded plurality vote. They augment the set of class labels Ω with one more class, ω_{c+1}, for all objects for which the ensemble either fails to determine a class label with a sufficient confidence or produces a tie. Thus the decision is

$$\begin{cases} \omega_k, & \text{if } \sum_{i=1}^{L} d_{i,k} \geq \alpha.L, \\ \omega_{c+1}, & \text{otherwise,} \end{cases} \tag{4.5}$$

where $0 < \alpha \leq 1$. For the simple majority, we can pick α to be $\frac{1}{2} + \epsilon$, where ϵ is arbitrarily small and $0 < \epsilon < \frac{1}{L}$. When $\alpha = 1$, (4.5) becomes the *unanimity voting* rule: a decision is made for a class label only if all decision makers agree on that label; otherwise the ensemble refuses to decide and assigns label ω_{c+1} to $\mathbf{x}$. The algorithm of the majority vote combiner is shown in Figure 4.1.

MAJORITY (PLURALITY) VOTE COMBINER (MV)

Training: None

Operation: For each new object

1. Find the class labels $s_1, \ldots, s_L$, assigned to this object by the L base classifiers.
2. Calculate the number of votes for each class ω_k, $k = 1, \ldots, c$.

$$P(k) = \sum_{i=1}^{L} I(s_i, \omega_k),$$

   where $I(a, b) = 1$ if $a = b$ and 0 otherwise.

3. Assign label k^* to the object, where

$$k^* = \arg\max_{k=1}^{c} P(k).$$

Return the ensemble label of the new object.

Figure 4.1 Training and operation algorithm for the Majority Vote combiner.

The plurality vote (4.4), called in a wide sense 'the majority vote', is the most often used rule from the majority vote group [26, 28, 248, 250, 260, 343].

4.3.2 Accuracy of the majority vote

Assume that

- The number of classifiers, L, is odd.
- The probability for each classifier to give the correct class label is p for any $\mathbf{x} \in \mathbb{R}^n$.

- The classifier outputs are independent, that is, for any set of classifiers $A \subseteq \mathcal{D}$, $A = \{D_{i_1}, \ldots, D_{i_K}\}$,

$$\begin{aligned} P(D_{i_1} = s_{i_1}, \quad \ldots \quad , D_{i_K} = s_{i_K}) \\ = \quad P(D_{i_1} = s_{i_1}) \times \ldots \times P(D_{i_K} = s_{i_K}), \end{aligned} \tag{4.6}$$

  where s_{i_j} is the label output of classifier D_{i_j}.

According to (4.4), the majority vote will give an accurate class label if at least $\lfloor L/2 \rfloor + 1$ classifiers give correct answers ($\lfloor a \rfloor$ denotes the 'floor', which is the nearest integer smaller than a).[2] Then the accuracy of the ensemble is

$$P_{maj} = \sum_{m=\lfloor L/2 \rfloor + 1}^{L} \binom{L}{m} p^m (1-p)^{L-m}. \tag{4.7}$$

The probabilities of correct classification of the ensemble for $p = 0.6, 0.7, 0.8$ and 0.9, and $L = 3, 5, 7$ and 9, are displayed in Table 4.1.

Table 4.1 Tabulated values of the majority vote accuracy of L independent classifiers with individual accuracy p.

	$L = 3$	$L = 5$	$L = 7$	$L = 9$
$p = 0.6$	0.6480	0.6826	0.7102	0.7334
$p = 0.7$	0.7840	0.8369	0.8740	0.9012
$p = 0.8$	0.8960	0.9421	0.9667	0.9804
$p = 0.9$	0.9720	0.9914	0.9973	0.9991

The following result is also known as the Condorcet Jury Theorem (1785) [359]

1. If $p > 0.5$, then P_{maj} in (4.7) is monotonically increasing and

$$P_{maj} \to 1 \quad \text{as} \quad L \to \infty. \tag{4.8}$$

2. If $p < 0.5$, then P_{maj} at (4.7) is monotonically decreasing and

$$P_{maj} \to 0 \quad \text{as} \quad L \to \infty. \tag{4.9}$$

3. If $p = 0.5$, then $P_{maj} = 0.5$ for any L.

This result supports the intuition that we can expect improvement over the individual accuracy p only when p is higher than 0.5. Lam and Suen [250] proceed to

[2]Notice that the majority (50 %+1) is necessary and sufficient in the case of two classes, and is sufficient but not necessary for $c > 2$. Thus the accuracy of an ensemble using plurality when $c > 2$ could be greater than the majority vote accuracy.

analyze the case of even L and the effect on the ensemble accuracy of adding or removing classifiers.

Shapley and Grofman [359] note that the result is valid even for unequal p, provided the distribution of the individual accuracies p_i is symmetrical about the mean.

EXAMPLE 4.1 Majority and unanimity in medical diagnostics

An accepted practice in medicine is to confirm the diagnosis by several (supposedly independent) tests. Lachenbruch [245] studies the unanimity and majority rules on a sequence of 3 tests for HIV diagnosis.

Sensitivity and specificity are the two most important characteristics of a medical test. Sensitivity (denoted by U) is the probability that the test procedure declares an affected individual affected (probability of a *true positive*). Specificity (denoted by V) is the probability that the test procedure declares an unaffected individual unaffected (probability of a *true negative*).[3]

Denote by T the event 'positive test result', and by A, the event 'the person is affected by the disease'. Then $U = P(T|A)$ and $V = P(\bar{T}|\bar{A})$, where the over-bar denotes negation. We regard the test as an individual classifier with accuracy $p = U \times P(A) + V \times (1 - P(A))$, where $P(A)$ is the probability for the occurrence of the disease among the examined individuals, or the *prevalence* of the disease. In testing for HIV, a unanimous positive result from 3 tests is required to declare the individual affected [245]. Since the tests are applied one at a time, encountering the first negative result will cease the procedure. Another possible combination is the majority vote which will stop if the first two readings agree or otherwise take the third reading to resolve the tie. Table 4.2 shows the outcomes of the tests and the overall decision for the unanimity and majority rules.

Table 4.2 Unanimity and majority schemes for three independent consecutive tests.

Unanimity sequences for decision $(+)$:	$\langle+++\rangle$		
Unanimity sequences for decision $(-)$:	$\langle-\rangle$	$\langle+-\rangle$	$\langle++-\rangle$
Majority sequences for decision $(+)$:	$\langle++\rangle$	$\langle-++\rangle$	$\langle+-+\rangle$
Majority sequences for decision $(-)$:	$\langle--\rangle$	$\langle-+-\rangle$	$\langle+--\rangle$

Assume that the three tests are applied independently and all have the same sensitivity u and specificity v. Then the sensitivity and the specificity of the

[3] In social sciences, for example, sensitivity translates to 'convicting the guilty' and specificity, to 'freeing the innocent' [359].

procedure with the unanimity vote become

$$\begin{aligned} U_{una} &= u^3, \\ V_{una} &= 1-(1-v)^3. \end{aligned} \tag{4.10}$$

For the majority vote,

$$\begin{aligned} U_{maj} &= u^2+2u^2(1-u) = u^2(3-2u), \\ V_{maj} &= v^2(3-2v). \end{aligned} \tag{4.11}$$

For $0 < u < 1$ and $0 < v < 1$, by simple algebra we obtain

$$U_{una} < u \quad \text{and} \quad V_{una} > v, \tag{4.12}$$

and

$$U_{maj} > u \quad \text{and} \quad V_{maj} > v. \tag{4.13}$$

Thus, there is a certain gain on both sensitivity and specificity if majority vote is applied. Therefore, the combined accuracy $P_{maj} = U \times P(A) + V \times (1-P(A))$ is also higher than the accuracy of a single test $p = u \times P(A) + v \times (1-P(A))$. For the unanimity rule, there is a substantial increase of specificity at the expense of decreased sensitivity. To illustrate this point, consider the ELISA test used for diagnosing HIV. According to Lachenbruch [245], this test has been reported to have sensitivity $u = 0.95$ and specificity $v = 0.99$. Then

$$\begin{aligned} U_{una} &\approx 0.8574 \quad & V_{una} &\approx 1.0000 \\ U_{maj} &\approx 0.9928 \quad & V_{maj} &\approx 0.9997. \end{aligned}$$

The sensitivity of the unanimity scheme is dangerously low. This means that the chance of an affected individual being misdiagnosed as unaffected is above 14%. There are different ways to remedy this. One possibility is to apply a more expensive and more accurate second test in case ELISA gave a positive result, for example, the Western Blot test, for which $u = v = 0.99$ [245].

4.3.3 Limits on the majority vote accuracy: an example

Let $\mathcal{D} = \{D_1, D_2, D_3\}$ be an ensemble of three classifiers with the same individual probability of correct classification $p = 0.6$. Suppose that there are 10 objects in a hypothetical data set, and that each classifier correctly labels exactly 6 of them. Each classifier output is recorded as correct (1) or wrong (0). Given these requirements, *all* possible combinations of distributing 10 elements into the 8 combinations of outputs of the three classifiers are shown in Table 4.3. The penultimate column of Table 4.3 shows the majority vote accuracy of each of the 28 possible combinations. It is obtained as the proportion (out of 10 elements) of the sum of the entries in columns '111', '101', '011' and '110' (two or more correct votes). The rows of the table are

Table 4.3 All possible combinations of correct/incorrect classification of 10 objects by three classifiers so that each classifier recognizes exactly 6 objects. The entries in the table are the number of occurrences of the specific binary output of the three classifiers in the particular combination. The majority vote accuracy P_{maj} and the improvement over the single classifier, $P_{maj} - p$ are also shown. Three characteristic classifier ensembles are marked.

No	111	101	011	001	110	100	010	000	P_{maj}	$P_{maj} - p$
	a	b	c	d	e	f	g	h		
Pattern of success										
1	**0**	**3**	**3**	**0**	**3**	**0**	**0**	**1**	**0.9**	**0.3**
2	2	2	2	0	2	0	0	2	0.8	0.2
3	1	2	2	1	3	0	0	1	0.8	0.2
4	0	2	3	1	3	1	0	0	0.8	0.2
5	0	2	2	2	4	0	0	0	0.8	0.2
6	4	1	1	0	1	0	0	3	0.7	0.1
7	3	1	1	1	2	0	0	2	0.7	0.1
8	2	1	2	1	2	1	0	1	0.7	0.1
9	2	1	1	2	3	0	0	1	0.7	0.1
10	1	2	2	1	2	1	1	0	0.7	0.1
11	1	1	2	2	3	1	0	0	0.7	0.1
12	1	1	1	3	4	0	0	0	0.7	0.1
Identical classifiers										
13	6	0	0	0	0	0	0	4	0.6	0.0
14	5	0	0	1	1	0	0	3	0.6	0.0
15	4	0	1	1	1	1	0	2	0.6	0.0
16	4	0	0	2	2	0	0	2	0.6	0.0
17	3	1	1	1	1	1	1	1	0.6	0.0
18	3	0	1	2	2	1	0	1	0.6	0.0
19	3	0	0	3	3	0	0	1	0.6	0.0
20	2	1	1	2	2	1	1	0	0.6	0.0
21	2	0	2	2	2	2	0	0	0.6	0.0
22	2	0	1	3	3	1	0	0	0.6	0.0
23	2	0	0	4	4	0	0	0	0.6	0.0
24	5	0	0	1	0	1	1	2	0.5	-0.1
25	4	0	0	2	1	1	1	1	0.5	-0.1
26	3	0	1	2	1	2	1	0	0.5	-0.1
27	3	0	0	3	2	1	1	0	0.5	-0.1
Pattern of failure										
28	**4**	**0**	**0**	**2**	**0**	**2**	**2**	**0**	**0.4**	**-0.2**

Table 4.4 The 2×2 relationship table with probabilities.

	D_k correct (1)	D_k wrong (0)
D_i correct (1)	a	b
D_i wrong (0)	c	d

Total, $a + b + c + d = 1$

ordered by the majority vote accuracy. To clarify the entries in Table 4.3, consider as an example the first row. The number 3 in the column under the heading '101', means that exactly 3 objects are correctly recognized by D_1 and D_3 (the first and the third 1's of the heading) and misclassified by D_2 (the zero in the middle).

The table offers a few interesting facts:

- There is a case where the majority vote produces 90% correct classification. Although purely hypothetical, this vote distribution is *possible* and offers a dramatic increase over the individual rate $p = 0.6$.
- On the other hand, the majority vote is not guaranteed to do better than a single member of the ensemble. The combination in the bottom row has a majority vote accuracy of 0.4.

The best and the worst possible cases illustrated above are named 'the pattern of success' and the 'pattern of failure' [241] and detailed next.

4.3.4 Patterns of success and failure

Consider two classifiers D_i and D_k, and a 2×2 table of probabilities that summarizes their combined outputs as in Table 4.4.

The three-classifier problem from the previous section can be visualized using two pairwise tables as in Table 4.5. For this case,

$$a + b + c + d + e + f + g + h = 1. \tag{4.14}$$

Table 4.5 The probabilities in two 2-way tables illustrating a 3-classifier voting ensemble.

D_3 correct (1)

	$D_2 \rightarrow$	
$D_1 \downarrow$	1	0
1	a	b
0	c	d

D_3 wrong (0)

	$D_2 \rightarrow$	
$D_1 \downarrow$	1	0
1	e	f
0	g	h

The probability of correct classification of the majority vote of the three classifiers is (two or more correct)

$$P_{maj} = a + b + c + e. \tag{4.15}$$

All three classifiers have the same individual accuracy p, which brings in the following three equations

$$\begin{aligned} a + b + e + f &= p, \quad D_1 \text{ correct ;} \\ a + c + e + g &= p, \quad D_2 \text{ correct ;} \\ a + b + c + d &= p, \quad D_3 \text{ correct} \end{aligned} \tag{4.16}$$

Maximizing P_{maj} in (4.15) subject to conditions (4.14), (4.16) and a, b, c, d, e, f, g, $h \geq 0$, for $p = 0.6$, we obtain $P_{maj} = 0.9$ with the pattern highlighted in Table 4.3: $a = d = f = g = 0$, $b = c = e = 0.3$, $h = 0.1$. This example, optimal for 3 classifiers, indicates the possible characteristics of the best combination of L classifiers. The 'pattern of success' and 'pattern of failure' defined later follow the same intuition although we do not include a formal proof for their optimality.

Consider the pool $\mathcal{D}$ of L (odd) classifiers, each with accuracy p. For the majority vote to give a correct answer we need $\lfloor L/2 \rfloor + 1$ or more of the classifiers to be correct. Intuitively, the best improvement over the individual accuracy will be achieved when exactly $\lfloor L/2 \rfloor + 1$ votes are correct. Any extra correct vote for the same $\mathbf{x}$ will be wasted because it is not needed to give the correct class label. Correct votes which participate in combinations not leading to a correct overall vote are also wasted. To use the above idea, we make the following definition:

The 'pattern of success' is a distribution of the L classifier outputs such that:

1. The probability of any combination of $\lfloor L/2 \rfloor + 1$ correct and $\lfloor L/2 \rfloor$ incorrect votes is α;

2. The probability of all L votes being incorrect is γ;

3. The probability of any other combination is zero.

For $L = 3$, the 2-table expression of the pattern of success is shown in Table 4.6.

Table 4.6 The *Pattern of Success.*

D_3 correct (1)

$D_1 \downarrow$ \ $D_2 \rightarrow$	1	0
1	0	α
0	α	0

D_3 wrong (0)

$D_1 \downarrow$ \ $D_2 \rightarrow$	1	0
1	α	0
0	0	$\gamma = 1 - 3\alpha$

Here no votes are wasted; the only combinations that occur are where all classifiers are incorrect or exactly $\lfloor L/2 \rfloor + 1$ are correct. To simplify notation, let $l = \lfloor L/2 \rfloor$. The probability of a correct majority vote (P_{maj}) for the pattern of success is the sum of the probabilities of each correct majority vote combination. Each such combination has probability α. There are $\binom{L}{l+1}$ ways of having $l+1$ correct out of L classifiers. Therefore

$$P_{maj} = \binom{L}{l+1} \alpha. \tag{4.17}$$

The pattern of success is only possible when $P_{maj} \leq 1$

$$\alpha \leq \frac{1}{\binom{L}{l+1}}. \tag{4.18}$$

To relate the individual accuracies p to α and P_{maj}, consider the following argument. In the pattern of success, if D_i gives a correct vote, then the remaining $L-1$ classifiers must give l correct votes. There are $\binom{L-1}{l}$ ways in which the remaining $L-1$ classifiers can give l correct votes, each with probability α. So the overall accuracy p of D_i is

$$p = \binom{L-1}{l} \alpha. \tag{4.19}$$

Expressing α from (4.19) and substituting in (4.17) gives

$$P_{maj} = \frac{pL}{l+1} = \frac{2pL}{L+1}. \tag{4.20}$$

Feasible patterns of success have $P_{maj} \leq 1$, so (4.20) requires

$$p \leq \frac{L+1}{2L}. \tag{4.21}$$

If $p > \frac{(L+1)}{2L}$ then $P_{maj} = 1$ can be achieved, but there is an excess of correct votes. The improvement over the individual p will not be as large as for the pattern of success but the majority vote accuracy will be 1 anyway. The final formula for P_{maj} is

$$P_{maj} = \min\left\{1, \frac{2pL}{L+1}\right\}. \tag{4.22}$$

The worst possible behavior of an ensemble of L classifiers each with accuracy p, is described by the pattern of failure.

The 'pattern of failure' is a distribution of the L classifier outputs such that:

1. The probability of any combination of $\lfloor L/2 \rfloor$ correct and $\lfloor L/2 \rfloor + 1$ incorrect votes is β;
2. The probability of all L votes being correct is δ;
3. The probability of any other combination is zero.

For $L = 3$, the 2-table expression of the pattern of failure is shown in Table 4.7.

Table 4.7 The *Pattern of Failure*.

D_3 correct (1)

$D_1 \downarrow$ \ $D_2 \rightarrow$	1	0
1	$\delta = 1 - 3\beta$	0
0	0	β

D_3 wrong (0)

$D_1 \downarrow$ \ $D_2 \rightarrow$	1	0
1	0	β
0	β	0

The worst scenario is when the correct votes are wasted, that is, grouped in combinations of exactly l out of L correct (one short for the majority to be correct). The excess of correct votes needed to make up the individual p are also wasted by all the votes being correct together, while half of them plus one would suffice.

The probability of a correct majority vote (P_{maj}) is δ. As there are $\binom{L}{l}$ ways of having l correct out of L classifiers, each with probability β, then

$$P_{maj} = \delta = 1 - \binom{L}{l} \beta. \tag{4.23}$$

If D_i gives a correct vote then either all the remaining classifiers are correct (probability δ) or exactly $l - 1$ are correct out of the $L - 1$ remaining classifiers. For the second case there are $\binom{L-1}{l-1}$ ways of getting this, each with probability β. To get the overall accuracy p for classifier D_i we sum the probabilities of the two cases

$$p = \delta + \binom{L-1}{l-1} \beta. \tag{4.24}$$

Combining (4.23) and (4.24) gives

$$P_{maj} = \frac{pL - l}{l + 1} = \frac{(2p - 1)L + 1}{L + 1}. \tag{4.25}$$

For values of individual accuracy $p > 0.5$, the pattern of failure is always possible.

Matan [277] gives tight upper and lower bounds of the majority vote accuracy in the case of unequal individual accuracies (see Appendix A.1). Suppose that classifier D_i has accuracy p_i, and $\{D_1, \ldots, D_L\}$ are arranged so that $p_1 \leq p_2 \cdots \leq p_L$. Let $k = l + 1 = (L + 1)/2$. Matan proves that

1. The upper bound of the majority vote accuracy of the ensemble is

$$\max P_{maj} = \min\{1, \Sigma(k), \Sigma(k-1), \ldots, \Sigma(1)\}, \tag{4.26}$$

where

$$\Sigma(m) = \frac{1}{m} \sum_{i=1}^{L-k+m} p_i, \quad m = 1, \ldots, k. \tag{4.27}$$

2. The lower bound of the majority vote accuracy of the ensemble is

$$\min P_{maj} = \max\{0, \xi(k), \xi(k-1), \ldots, \xi(1)\}, \tag{4.28}$$

where

$$\xi(m) = \frac{1}{m} \sum_{i=k-m+1}^{L} p_i - \frac{L-k}{m}, \quad m = 1, \ldots, k. \tag{4.29}$$

EXAMPLE 4.2 Matan's limits on the majority vote accuracy

Let $\mathcal{D} = \{D_1, \ldots, D_5\}$ be a set of classifiers with accuracies (0.56, 0.58, 0.60, 0.60, 0.62), respectively. To find the upper bound of the majority vote accuracy of this ensemble, form the sums $\Sigma(m)$ for $m = 1, 2, 3$

$$\begin{aligned}
\Sigma(1) &= 0.56 + 0.58 + 0.60 = 1.74; \\
\Sigma(2) &= \frac{1}{2}(0.56 + 0.58 + 0.60 + 0.60) = 1.17; \\
\Sigma(3) &= \frac{1}{3}(0.56 + 0.58 + 0.60 + 0.60 + 0.62) = 0.99.
\end{aligned} \tag{4.30}$$

Then

$$\max\ P_{maj} = \min\{1, 1.74, 1.17, 0.99\} = 0.99. \tag{4.31}$$

For the lower bound,

$$\begin{aligned}
\xi(1) &= 0.60 + 0.60 + 0.62 - (5-3) = -0.18; \\
\xi(2) &= \frac{1}{2}(0.58 + 0.60 + 0.60 + 0.62) - \frac{5-3}{2} = 0.20; \\
\xi(3) &= \frac{1}{3}(0.56 + 0.58 + 0.60 + 0.60 + 0.62) - \frac{5-3}{3} = 0.32.
\end{aligned} \tag{4.32}$$

Then

$$\min\ P_{maj} = \max\{0, -0.18, 0.20, 0.32\} = 0.32. \tag{4.33}$$

The range of possible results from the majority vote across $\mathcal{D}$ is wide, so without more knowledge about how the classifiers are related to each other we can only guess within this range. If we assume that the classifier outputs are independent, then $P_{maj} = 0.67$, which indicates that there is much more to be achieved from the majority vote with dependent outputs.

Matan's result leads to the pattern of success and the pattern of failure as the upper and the lower bounds respectively, for $p_1 = \cdots = p_L = p$. Demirekler and Altincay [87] and Ruta and Gabrys [343] give further insights into the behavior of the two limit patterns.

Hierarchical majority voting ensembles have been found to be very promising [277, 343, 359]. There is a potential gain in accuracy but this has only been shown by construction examples.

4.3.5 Optimality of the majority vote combiner

Majority vote (plurality vote for more than two classes) is the optimal combiner when the individual classifier accuracies are equal, the 'leftover probability' is uniformly distributed across the remaining classes, and the prior probabilities for the classes are the same. The following theorem states this result more formally.

Theorem 4.1 *Let $\mathcal{D}$ be an ensemble of L classifiers. Suppose that*

1. *The classifiers give their decisions independently, conditioned upon the class label.*
2. *The individual classification accuracy is $P(s_i = \omega_k|\omega_k) = p$ for any classifier i and class ω_k, and also for any data point in the feature space.*
3. *The probability for incorrect classification is equally distributed among the remaining classes, that is $P(s_i = \omega_j|\omega_k) = \frac{1-p}{c-1}$, for any $i = 1, \ldots, L$, $k, j = 1, \ldots, c$ $j \neq k$.*

Then the majority vote is the optimal combination rule.

Proof. Substituting in the probabilistic framework defined in (4.3),

$$P(\omega_k|\mathbf{s}) = \frac{P(\omega_k)}{P(\mathbf{s})} \times \prod_{i \in I_+^k} p \times \prod_{i \in I_-^k} \frac{1-p}{c-1} \tag{4.34}$$

$$= \frac{P(\omega_k)}{P(\mathbf{s})} \times \prod_{i \in I_+^k} p \times \prod_{i \in I_-^k} \frac{1-p}{c-1} \times \frac{\prod_{i \in I_+^k} \frac{1-p}{c-1}}{\prod_{i \in I_+^k} \frac{1-p}{c-1}} \tag{4.35}$$

$$= \frac{P(\omega_k)}{P(\mathbf{s})} \times \prod_{i \in I_+^k} \frac{p(c-1)}{1-p} \times \prod_{i=1}^{L} \frac{1-p}{c-1} \, . \tag{4.36}$$

Notice that $P(\mathbf{s})$ and the last product term in (4.36) do not depend on the class label. The prior probability, $P(\omega_k)$ does depend on the class label but not on the

votes, so it can be designated as the *class constant*. Rearranging and taking the logarithm,

$$\begin{aligned}\log(P(\omega_k|\mathbf{s})) &= \log\left(\frac{(1-p)^L}{P(\mathbf{s})(c-1)^L}\right) + \log\left(P(\omega_k)\right) \\ &+ \log\left(\frac{p(c-1)}{1-p}\right) \times |I_+^k|, \end{aligned} \tag{4.37}$$

where $|.|$ denotes cardinality. Dividing by $\log\left(\frac{p(c-1)}{1-p}\right)$ and dropping all terms that do not depend on the class label or the vote counts, we can create the following class-support functions for the object $\mathbf{x}$

$$\mu_k(\mathbf{x}) = \underbrace{\frac{\log\left(P(\omega_k)\right)}{\log\left(\frac{p(c-1)}{1-p}\right)}}_{\text{class constant } \zeta(\omega_k)} + |I_+^k|. \tag{4.38}$$

Note that $|I_+^k|$ is the number of votes for ω_k. Choosing the class label corresponding to the largest support function is equivalent to choosing the class most voted for, subject to a constant term. ■

Interestingly, the standard majority vote rule does not include a class constant, and is still one of the most robust and accurate combiners for classifier ensembles. The class constant may sway the vote, especially for highly unbalanced classes and uncertain ensemble decisions where the number of votes for different classes are close. However, including the class constant will make majority vote a trainable combiner, which defeats one of its main assets. To comply with the common interpretation, here we adopt the standard majority vote formulation, whereby the class label is obtained by

$$\omega = \arg\max_k |I_+^k|. \tag{4.39}$$

4.4 Weighted majority vote

The weighted majority vote is among the most intuitive and widely used combiners [204, 261]. It is the designated combination method derived from minimizing a bound on the training error in AdaBoost [118, 133].

If the classifiers in the ensemble are not of identical accuracy, then it is reasonable to attempt to give the more competent classifiers more power in making the final decision. The label outputs can be represented as degrees of support for the classes in the following way

$$d_{i,j} = \begin{cases} 1, & \text{if } D_i \text{ labels } \mathbf{x} \text{ in } \omega_j, \\ 0, & \text{otherwise.} \end{cases} \tag{4.40}$$

The class-support function for class ω_j obtained through weighted voting is

$$\mu_j(\mathbf{x}) = \sum_{i=1}^{L} b_i d_{i,j}, \tag{4.41}$$

where b_i is a coefficient for classifier D_i. Thus, the value of the class-support function (4.41) will be the sum of the weights for those members of the ensemble whose output for $\mathbf{x}$ is ω_j.

4.4.1 Two examples

EXAMPLE 4.3 Assigning weights to the classifiers

Consider an ensemble of three classifiers D_1, D_2, and D_3 with accuracies 0.6, 0.6, and 0.7, respectively, and with independent oracle outputs. An accurate ensemble vote will be obtained if any two classifiers are correct. The ensemble accuracy will be

$$P_{maj} = 0.6^2 \times 0.3 + 2 \times 0.4 \times 0.6 \times 0.7 + 0.6^2 \times 0.7 = 0.6960. \tag{4.42}$$

Clearly, it will be better if we remove D_1 and D_2, and reduce the ensemble to the single and more accurate classifier D_3. We introduce weights or coefficients of importance b_i, $i = 1, 2, 3$, and rewrite (4.4) as: choose class label ω_k if

$$\sum_{i=1}^{L} b_i d_{i,k} = \max_{j=1}^{c} \sum_{i=1}^{L} b_i d_{i,j}. \tag{4.43}$$

For convenience we normalize the weights so that

$$\sum_{i=1}^{c} b_j = 1. \tag{4.44}$$

Assigning $b_1 = b_2 = 0$ and $b_3 = 1$, we get rid of D_1 and D_2, leading to $P_{maj} = p_3 = 0.7$. In fact, any set of weights which makes D_3 the dominant classifier will yield the same P_{maj}, for example, $b_3 > 0.5$ and any b_1 and b_2 satisfying (4.44)

In the above example the weighted voting did not improve on the single best classifier in the ensemble even for independent classifiers. The following example shows that, in theory, the weighting might lead to a result better than both the single best member of the ensemble and the simple majority.

EXAMPLE 4.4 Improving the accuracy by weighting

Consider an ensemble of 5 classifiers $D_1, \ldots, D_5$ with accuracies (0.9, 0.9, 0.6, 0.6, 0.6).[4] If the classifiers are independent, the majority vote accuracy (at least

[4] After [359].

3 out of 5 correct votes) is

$$\begin{aligned} P_{maj} &= 3 \times 0.9^2 \times 0.4 \times 0.6 + 0.6^3 + 6 \times 0.9 \times 0.1 \times 0.6^2 \times 0.4 \\ &\approx 0.877. \end{aligned} \tag{4.45}$$

Assume now that the weights given to the voters are (1/3, 1/3, 1/9, 1/9, 1/9). Then the two more competent classifiers agreeing will be enough to make the decision because the score for the class label they agree upon will become 2/3. If they disagree, that is, one is correct and one is wrong, the vote of the ensemble will be decided by the majority of the remaining three classifiers. Then the accuracy for the weighted voting will be

$$\begin{aligned} P^w_{maj} &= 0.9^2 + 2 \times 3 \times 0.9 \times 0.1 \times 0.6^2 \times 0.4 + 2 \times 0.9 \times 0.1 \times 0.6^3 \\ &\approx 0.927. \end{aligned} \tag{4.46}$$

Again, any set of weights that satisfy (4.44) and make the first two classifiers prevail when they agree, will lead to the same outcome.

4.4.2 Optimality of the weighted majority vote combiner

Here we use the probabilistic framework 4.2 to derive the optimality conditions for the weighted majority vote. This type of combiner follows from relaxing the assumption about equal individual accuracies. Hence the majority vote combiner is a special case of the weighted majority combiner for equal individual accuracies.

Theorem 4.2 *Let $\mathcal{D}$ be an ensemble of L classifiers. Suppose that*

1. *The classifiers give their decisions independently, conditioned upon the class label.*

2. *The individual classification accuracy is $P(s_i = \omega_k|\omega_k) = p_i$ for any class ω_k, and also for any data point in the feature space.*

3. *The probability for incorrect classification is equally distributed among the remaining classes, that is $P(s_i = \omega_j|\omega_k) = \frac{1-p_i}{c-1}$, for any $i = 1, \ldots, L$, $k, j = 1, \ldots, c$ $j \neq k$.*

Then the weighted majority vote is the optimal combination rule with weights

$$w_i = \log\left(\frac{p_i}{1-p_i}\right), \quad 0 < p_i < 1. \tag{4.47}$$

Proof. Following the same derivation path as with the majority vote optimality, equation (4.3) becomes

$$P(\omega_k|\mathbf{s}) = \frac{P(\omega_k)}{P(\mathbf{s})} \times \prod_{i\in I_+^k} p_i \times \prod_{i\in I_-^k} \frac{1-p_i}{c-1} \tag{4.48}$$

$$= \frac{P(\omega_k)}{P(\mathbf{s})} \times \prod_{i\in I_+^k} \frac{p_i(c-1)}{1-p_i} \times \prod_{i=1}^{L} \frac{1-p_i}{c-1} \tag{4.49}$$

$$= \frac{1}{P(\mathbf{s})} \times \prod_{i=1}^{L} \frac{1-p_i}{c-1} \times P(\omega_k) \times \prod_{i\in I_+^k} \frac{p_i(c-1)}{1-p_i}. \tag{4.50}$$

Then

$$\log(P(\omega_k|\mathbf{s})) = \log\left(\frac{\prod_{i=1}^{L}(1-p_i)}{P(\mathbf{s})(c-1)^L}\right) + \log\left(P(\omega_k)\right) + \sum_{i\in|I_+^k|} \log\left(\frac{p_i}{1-p_i}\right) + |I_+^k| \times \log(c-1). \tag{4.51}$$

Dropping the first term, which will not influence the class decision, and expressing the classifier weights as

$$w_i = \log\left(\frac{p_i}{1-p_i}\right), \quad 0 < p_i < 1,$$

will transform equation (4.51) to

$$\mu_k(\mathbf{x}) = \underbrace{\log\left(P(\omega_k)\right)}_{\text{class constant } \zeta(\omega_k)} + \sum_{i\in|I_+^k|} w_i + |I_+^k| \times \log(c-1). \tag{4.52}$$

■

If $p_i = p$ for all $i = 1, \ldots, L$, equation (4.52) reduces to the majority vote equation (4.37).

Similar proofs have been derived independently by several researchers in different fields of science such as democracy studies, pattern recognition and automata theory. The earliest reference according to [26, 359] was Pierce, 1961 [309] .

The algorithm of the weighted majority vote combiner is shown in Figure 4.2. Note that the figure shows the conventional version of the algorithm which does not include the class constant or the last term in Equation (4.52).

4.5 Naïve-Bayes combiner

Exploiting the independence assumption further leads to a combiner called the 'independence model' [385], 'naïve Bayes', 'simple Bayes' [103] and even 'idiot's Bayes'

WEIGHTED MAJORITY VOTE COMBINER (WMV)

Training

1. Obtain an array $E_{(N\times L)}$ with individual outputs of L classifiers for N objects. Entry $e(i,j)$ is the class label assigned by classifier D_j to object i. An array $T_{(N\times 1)}$ with the true labels is also provided.
2. Estimate the accuracy of each base classifier D_i, $i = 1,\ldots,L$, as the proportion of matches between column i of E and the the true labels T. Denote the estimates by $\hat{p}_i$.
3. Calculate the weights for the classifiers

$$v_i = \log\left(\frac{\hat{p}_i}{1-\hat{p}_i}\right), \quad 0 < \hat{p}_i < 1, \quad i = 1\ldots,L.$$

Operation: For each new object

1. Find the class labels $s_1,\ldots,s_L$ assigned to this object by the L base classifiers.
2. Calculate the score for all classes

$$P(k) = \sum_{\mathbf{s}_i=\omega_k} v_i, \quad k = 1,\ldots,c.$$

3. Assign label k^* to the object, where

$$k^* = \arg\max_{k=1}^{c} P(k).$$

Return the ensemble label of the new object.

Figure 4.2 Training and operation algorithm for the Weighted Majority Vote combiner.

[107, 330]. Sometimes the first adjective is skipped and the combination method is called just '*Bayes* combination'.

4.5.1 Optimality of the Naïve Bayes combiner

We can derive this combiner by finally dropping the assumption of equal individual accuracies in the probabilistic framework 4.2.

Theorem 4.3 *Let $\mathcal{D}$ be an ensemble of L classifiers. Suppose that the classifiers give their decisions independently, conditioned upon the class label. Then the Naïve Bayes combiner*

$$\max\left\{P(\omega_k)\prod_{i=1}^{L}P(s_i|\omega_k)\right\}$$

is the optimal combination rule.

Proof. Think of $P(s_i = \omega_j|\omega_k)$ as the (j,k)-th entry in a probabilistic confusion matrix for classifier i. In this case, equation (4.2) can be used directly

$$P(\omega_k|\mathbf{s}) \quad = \quad \frac{P(\omega_k)}{P(\mathbf{s})}\prod_{i=1}^{L}P(s_i|\omega_k)\ . \tag{4.53}$$

Dropping $P(\mathbf{s})$, which does not depend on the class label, the support for class ω_k is

$$\mu_k(\mathbf{x}) = P(\omega_k)\prod_{i=1}^{L}P(s_i|\omega_k)\ . \tag{4.54}$$

■

4.5.2 Implementation of the Naïve Bayes combiner

The implementation of the Naïve Bayes (NB) method on a data set $\mathbf{Z}$ with cardinality N is explained below. For each classifier D_i, a $c \times c$ confusion matrix CM^i is calculated by applying D_i to the training data set. The (k,s)th entry of this matrix, $cm^i_{k,s}$ is the number of elements of the data set whose true class label was ω_k, and were assigned by D_i to class ω_s. Denote by N_k the number of elements of $\mathbf{Z}$ from class ω_k, $k = 1,\ldots,c$. Taking $cm^i_{k,s_i}/N_k$ to be an estimate of the probability $P(s_i|\omega_k)$, and N_k/N to be an estimate of the prior probability for class ω_s, the support for class ω_k in (4.54) can be expressed as

$$\mu_k(\mathbf{x}) = \frac{1}{N_k^{L-1}}\prod_{i=1}^{L}cm^i(k,s_i). \tag{4.55}$$

EXAMPLE 4.5 Naïve Bayes combination

Consider a problem with L=2 classifiers, D_1 and D_2, and $c = 3$ classes. Let the number of training data points be $N = 20$. From these, let 8 be from ω_1, 9 from ω_2 and 3 from ω_3. Suppose that the following confusion matrices have been obtained for the two classifiers

$$CM^1 = \begin{bmatrix} 6 & 2 & 0 \\ 1 & 8 & 0 \\ 1 & 0 & 2 \end{bmatrix}, \quad \text{and} \quad CM^2 = \begin{bmatrix} 4 & 3 & 1 \\ 3 & 5 & 1 \\ 0 & 0 & 3 \end{bmatrix}. \tag{4.56}$$

Assume $D_1(\mathbf{x}) = s_1 = \omega_2$ and $D_2(\mathbf{x}) = s_2 = \omega_1$ for the input $\mathbf{x} \in \mathbb{R}^n$. Using (4.55),

$$\begin{aligned}
\mu_1(\mathbf{x}) &\propto = \frac{1}{8} \times 2 \times 4 = 1; \\
\mu_2(\mathbf{x}) &\propto \frac{1}{9} \times 8 \times 3 = \frac{8}{3} \approx 2.67; \\
\mu_3(\mathbf{x}) &\propto \frac{1}{3} \times 0 \times 0 = 0.
\end{aligned} \tag{4.57}$$

As $\mu_2(\mathbf{x})$ is the highest of the three values, the maximum membership rule will label $\mathbf{x}$ in ω_2.

Notice that a zero as an estimate of $P(s_i|\omega_k)$ automatically nullifies $\mu_k(\mathbf{x})$ regardless of the rest of the estimates. Titterington et al. [385] study the Naïve Bayes classifier for independent categorical features. They discuss several modifications of the estimates to account for the possible zeros. For the Naïve Bayes combination, we can plug in (4.54) the following estimate:

$$\hat{P}(s_i|\omega_k) = \frac{cm^i_{k,s_i} + \frac{1}{c}}{N_k + 1}, \tag{4.58}$$

where N_k is the number of elements in the training set $\mathbf{Z}$ from class ω_k, $k = 1, \ldots, c$. The algorithm for the training and the operation of the NB combiner is shown in Figure 4.3. MATLAB function `nb_combiner` is given in Appendix A.2.

EXAMPLE 4.6 Naïve Bayes combination with a correction for zeros

Take the 20-point data set and the confusion matrices CM^1 and CM^2 from the previous example. The estimates of the class-conditional pmfs for the values $s_1 = \omega_2$ and $s_2 = \omega_1$ are

$$\begin{aligned}
\mu_1(\mathbf{x}) &\propto \frac{N_1}{N} \times \left(\frac{c^1_{1,2} + \frac{1}{3}}{N_1 + 1}\right) \left(\frac{c^2_{1,1} + \frac{1}{3}}{N_1 + 1}\right) \\
&= \frac{8}{20} \times \left(\frac{2 + \frac{1}{3}}{8 + 1}\right) \left(\frac{4 + \frac{1}{3}}{8 + 1}\right) \approx 0.050 \\
\mu_2(\mathbf{x}) &\propto \frac{N_2}{N} \times \left(\frac{c^1_{2,2} + \frac{1}{3}}{N_2 + 1}\right) \left(\frac{c^2_{2,1} + \frac{1}{3}}{N_2 + 1}\right) \\
&= \frac{9}{20} \times \left(\frac{8 + \frac{1}{3}}{9 + 1}\right) \left(\frac{3 + \frac{1}{3}}{9 + 1}\right) \approx 0.125 \\
\mu_3(\mathbf{x}) &\propto \frac{N_3}{N} \times \left(\frac{c^1_{3,2} + \frac{1}{3}}{N_3 + 1}\right) \left(\frac{c^2_{3,1} + \frac{1}{3}}{N_3 + 1}\right) \\
&= \frac{3}{20} \times \left(\frac{0 + \frac{1}{3}}{3 + 1}\right) \left(\frac{0 + \frac{1}{3}}{3 + 1}\right) \approx 0.001.
\end{aligned} \tag{4.59}$$

NAÏVE BAYES COMBINER (NB)

Training

1. Obtain an array $E_{(N\times L)}$ with individual outputs of L classifiers for N objects. Entry $e(i,j)$ is the class label assigned by classifier D_j to object i. An array $T_{(N\times 1)}$ with the true labels is also provided.
2. Find the number of objects in each class within T. Denote these numbers by $N_1, N_2, \ldots, N_c$.
3. For each classifier D_i, $i = 1, \ldots, L$, calculate a bespoke $c \times c$ confusion matrix C_i. The (j_1, j_2)-th entry is

$$C_i(j_1, j_2) = \frac{K(j_1, j_2) + \frac{1}{c}}{N_{j_1} + 1},$$

   where $K(j_1, j_2)$ is the number of objects with true class label j_1, labeled by classifier D_i in class j_2.

Operation: For each new object

1. Find the class labels $s_1, \ldots, s_L$, assigned to this object by the L base classifiers.
2. For each class ω_k, $k = 1, \ldots, c$
   (a) Set $P(k) = \frac{N_k}{N}$.
   (b) For $i = 1 \ldots L$, calculate $P(k) \leftarrow P(k) \times C_i(k, s_i)$.
3. Assign label k^* to the object, where

$$k^* = \arg\max_{k=1}^{c} P(k).$$

Return the ensemble label of the new object.

Figure 4.3 Training and operation algorithm for the NB combiner.

Again, label ω_2 will be assigned to $\mathbf{x}$. Notice that class ω_3 now has a small non-zero support.

Despite the condescending names it has received, the Naïve Bayes combiner has been acclaimed for its rigorous statistical underpinning and robustness. It has been found to be surprisingly accurate and efficient in many experimental studies. The surprise comes from the fact that the combined entities are seldom independent. Thus, the

independence assumption is nearly always violated, sometimes severely. However, it turns out that the classifier performance is quite robust, even in the case of dependence. Furthermore, attempts to amend the Naïve Bayes by including estimates of some dependencies, do not always pay off [103].

4.6 Multinomial methods

In this group of methods we estimate the posterior probabilities $P(\omega_k|\mathbf{s})$ for all $k = 1, \ldots, c$, and every combination of votes $\mathbf{s} \in \Omega^L$. The highest posterior probability determines the class label for $\mathbf{s}$. Then, given an $\mathbf{x} \in \mathbb{R}^n$, first the labels $s_1, \ldots, s_L$ are assigned by the classifiers in the ensemble $\mathcal{D}$, and then the final label is retrieved for $\mathbf{s} = [s_1, \ldots, s_L]^T$.

"Behavior Knowledge Space"(BKS) is a fancy name for the multinomial combination. The label vector $\mathbf{s}$ is regarded as an index to a cell in a look-up table (the BKS table) [190]. The table is designed using a labeled data set $\mathbf{Z}$. Each $\mathbf{z}_j \in \mathbf{Z}$ is placed in the cell indexed by the $\mathbf{s}$ for that object. The number of elements in each cell are tallied, and the most representative class label is selected for this cell. The highest score corresponds to the highest estimated posterior probability $P(\omega_k|\mathbf{s})$. Ties are resolved arbitrarily. The empty cells are labeled in some appropriate way. For example, we can choose a label at random or use the result from a majority vote between the elements of $\mathbf{s}$.

To have a reliable multinomial combiner, the data set should be large. The BKS combination method is often overtrained: it works very well on the training data but poorly on the testing data. Raudys [324,325] carried out a comprehensive analysis of the problems and solutions related to the training of BKS (among other combiners) for large and small sample sizes.

The BKS combiner is the optimal combiner for any dependencies between the classifier outputs. The caveat here is that it is hardly possible to have reliable estimates of the posterior probabilities for all possible c^L output combinations $\mathbf{s}$, even for the most frequently occurring combinations.

EXAMPLE 4.7 BKS combination method

Consider a problem with three classifiers and two classes. Assume that D_1, D_2 and D_3 produce output $(s_1, s_2, s_3) = (\omega_2, \omega_1, \omega_2)$. Suppose that there are 100 objects in $\mathbf{Z}$ for which this combination of labels occurred: 60 having label ω_1, and 40 having label ω_2. Hence the table cell indexed by $(\omega_2, \omega_1, \omega_2)$ will contain label ω_1 no matter that the majority of the classifiers suggest otherwise.

From an implementation point of view, the BKS combiner can be regarded as the nearest neighbor classifier in the space of the ensemble outputs over the training data set. The concept of distance is replaced by exact match. If there are more than one nearest neighbors (exact matches) in the training set, the labels of the matches are tallied, and the label of the largest class representation is assigned. The algorithm is shown in Figure 4.4

BEHAVIOR KNOWLEDGE SPACE (BKS) COMBINER

Training

1. Obtain a reference array $E_{(N\times L)}$ with individual outputs of L classifiers for N objects. Entry $e(i,j)$ is the class label assigned by classifier D_j to object i. An array $T_{(N\times 1)}$ with the true labels is also provided.
2. Find the prevalent class label within T, say ω_p.

Operation: For each new object

1. Find the class labels assigned to this object by the L base classifiers and place them in a vector row $\mathbf{r}$.
2. Compare $\mathbf{r}$ with each row of the reference array. Record in a set S the labels of the objects whose rows match $\mathbf{r}$.
3. If there is no match ($S = \emptyset$), assign label ω_p to the new object. Otherwise, assign the prevalent class label within S. If there is a label tie, choose at random among the tied classes.

Return the ensemble label of the new object.

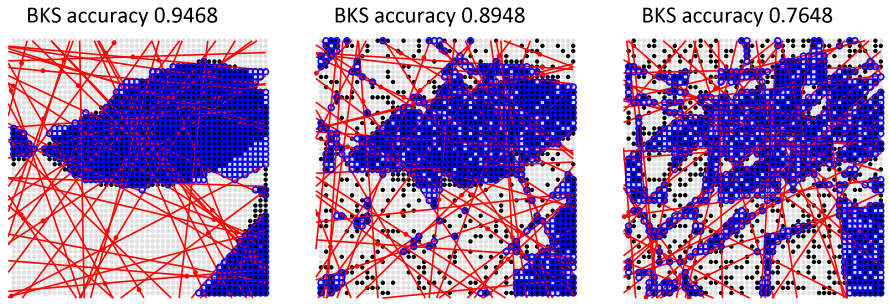
Figure 4.4 Operation algorithm for the BKS combiner for a given reference ensemble with labeled data and a set of new objects to be labeled by the ensemble.

EXAMPLE 4.8 BKS combiner for the fish data

A MATLAB function `bks_combiner` implementing the algorithm in Figure 4.4 is given in Appendix A.2. A MATLAB script which uses the function to label the fish data is also provided. Note that the script needs function `fish_data`, given in Appendix A.1.

Fifty random linear classifiers were generated in the data space. The grid space was scaled to the unit square. To generate a linear classifier, a random point $P(p_1, p_2)$ was selected within the square (not necessarily a node on the grid). Two random numbers were drawn from a standard normal distribution, to be used as coefficients a and b in the line equation $ax + by + c = 0$. Then the constant c was calculated so that P lies on the line: $c = -ap_1 - bp_2$.

BKS has been applied to combine the outputs of the 50 classifiers. Figure 4.5 shows the classifier boundaries and the regions labeled as the fish (class black dots) by the ensemble with three levels of noise: 0%, 20% and 35%. The accuracy displayed as the plot title is calculated with respect to the original (noise-free) class labels.



(a) No label noise (b) Label noise 20% (c) Label noise 35%

Figure 4.5 BKS classifier combiner for ensembles with $L = 50$ random linear classifiers for the fish data, with different amount of label noise.

The accuracy of the BKS combiner is very high, even for large amount of label noise. The real problem with this combiner comes when the testing data evokes ensemble outputs which do not appear in the reference ensemble. If the reference data is sufficiently representative, unmatched outputs will be relatively rare. Our implementation of the BKS combiner does not look beyond the exact match. It is possible to combat the brittleness of the method by considering distances between the (nominal) label vectors.

4.7 Comparison of combination methods for label outputs

Table 4.8 shows the optimality scopes and the number of tunable parameters for each combiner.

In practice, the success of a particular combiner will depend partly on the validity of the assumptions and partly on the availability of sufficient data to make reliable estimates of the parameters.

The optimality of the combiners is asymptotic, and holds for sample size approaching infinity. For finite sample sizes, the accuracy of the estimates of the parameters may be the primary concern. A combiner with fewer tunable parameters may be preferable even though its optimality assumption does not hold.

EXAMPLE 4.9 Label output combiners for the fish data set

Consider the following experiment. Fifty linear classifiers were randomly generated in the grid space of the fish data set. An example of 50 linear boundaries is shown in Figure 4.6 (a).

The labels of the two regions for each linear classifier were assigned randomly. The accuracy of the classifier was evaluated. Note that the accuracy estimate is exact because we have all possible data points (nodes on the grid). If

Table 4.8 Scopes of optimality (denoted by a black square) and the number of tunable parameters of the 4 combiners for a problem with c classes and an ensemble of L classifiers.

Combiner	1	2	3	4	Number of parameters
Majority vote (not trained)	■	–	–	–	none (requires equal priors for the classes)
Weighted majority vote	■	■	–	–	$L + c$
Naïve Bayes	■	■	■	–	$L * c^2 + c$
BKS	■	■	■	■	c^L

Column headings:
1. Equal p
2. Classifier-specific p_i
3. Full confusion matrix
4. Independence is not required

the accuracy was less than 50%, the regions were swapped over. Knowing the exact value of the classification accuracy eliminates the estimation error. Thus the only source of error in the ensemble error estimate came from the assumptions being incorrect.

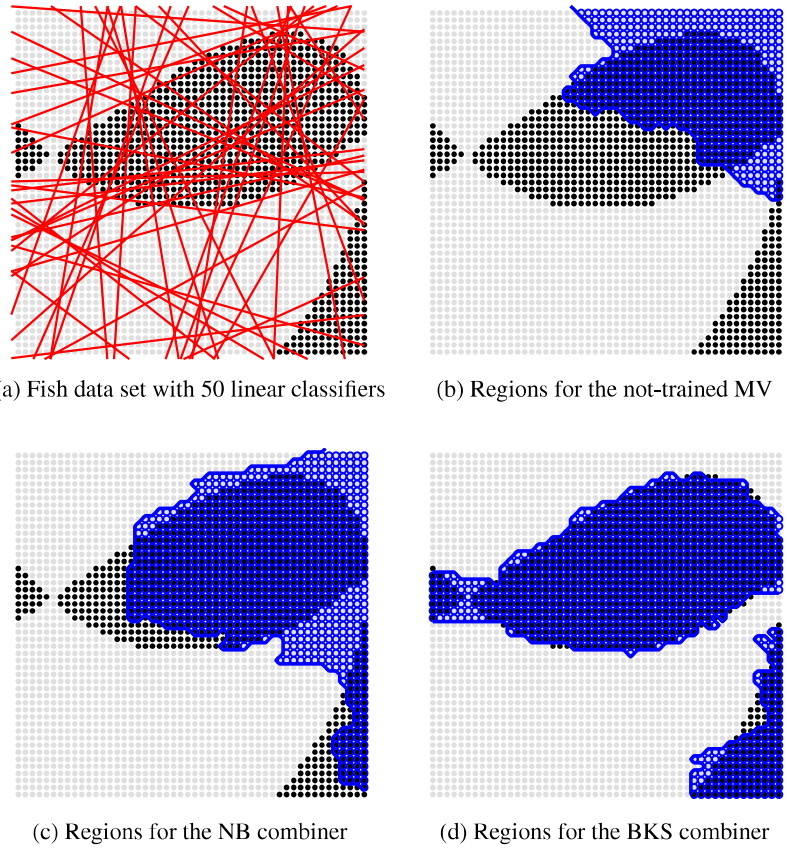
The shaded regions in plots (b), (c) and (d) in Figure 4.6 show the ensemble classification regions for class 'black dots', for three combination rules: the majority vote (4.39), the Naïve Bayes combiner (4.54) with the correction for zeros and for the BKS combiner. The individual and ensemble accuracies are detailed in Table 4.9.

Table 4.9 Classification accuracies in % of the individual and ensemble classifiers for different label combiners

Classifier/Ensemble	Example	Average of 200 runs $\pm\sigma$
Largest prior classifier	64.48	64.48±0.00
Average individual classifier	59.76	59.91±0.81
Majority vote (not trained)	*70.60	68.53±2.69
Weighted majority	68.84	69.63±1.91
Naïve Bayes	*82.92	75.21±2.85
BKS	*95.72	94.42±1.00

Table note: * Plot appears in Figure 4.6.

Two hundred runs were carried out with different random 'bunch of straws' (50 random classifiers) thrown in the unit square. Table 4.9 shows the average accuracies together with the standard deviations. The accuracies are ranked as



(a) Fish data set with 50 linear classifiers

(b) Regions for the not-trained MV

(c) Regions for the NB combiner

(d) Regions for the BKS combiner

Figure 4.6 A random linear ensemble for the fish data set.

expected for the 200-run experiment. Progressively alleviating the assumption of equal individual accuracies pays off. Weighted majority vote is better than the majority vote, and Naïve Bayes is better than both. BKS is always the best combiner because there is no parameter estimation error. However, this ranking is not guaranteed. Violation of the assumptions may affect the ensemble accuracy to various degrees, and disturb the ranking.

Appendix

A.1 Matan's proof for the limits on the majority vote accuracy

Here we give a sketch of the proof as offered in [277].

Theorem A.4 *Given is a classifier ensemble* $\mathcal{D} = \{D_1, \ldots, D_L\}$. *Suppose that classifiers* D_i *have accuracies* p_i, $i = 1, \ldots, L$, *and are arranged so that* $p_1 \leq p_2 \cdots \leq p_L$. *Let* $k = l + 1 = (L + 1)/2$. *Then*

1. The upper bound of the majority vote accuracy of the ensemble is

$$\max P_{maj} = \min\{1, \Sigma(k), \Sigma(k-1), \ldots, \Sigma(1)\}, \tag{A.1}$$

where

$$\Sigma(m) = \frac{1}{m} \sum_{i=1}^{L-k+m} p_i, \quad m = 1, \ldots, k. \tag{A.2}$$

2. The lower bound of the majority vote accuracy of the ensemble is

$$\min P_{maj} = \max\{0, \xi(k), \xi(k-1), \ldots, \xi(1)\}, \tag{A.3}$$

where

$$\xi(m) = \frac{1}{m} \sum_{i=k-m+1}^{L} p_i - \frac{L-k}{m}, \quad m = 1, \ldots, k. \tag{A.4}$$

Proof (sketch)

Upper Bound. The accuracy p_i is the average accuracy of D_i across the whole feature space $\mathbb{R}^n$, and can be written as

$$p_i = \int_{\mathbb{R}^n} \mathcal{I}_i(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}, \tag{A.5}$$

where $\mathcal{I}_i$ is an indicator function for classifier D_i, defined as

$$\mathcal{I}_i(\mathbf{x}) = \begin{cases} 1, & \text{if } D_i \text{ recognizes correctly } \mathbf{x}, \\ 0, & \text{otherwise,} \end{cases} \tag{A.6}$$

and $p(\mathbf{x})$ is the probability density function of $\mathbf{x}$. The majority vote accuracy is the probability of having k or more correct votes, averaged over the feature space $\mathbb{R}^n$.

$$P_{maj} = \int_{\sum \mathcal{I}_i(\mathbf{x}) \geq k} p(\mathbf{x}) d\mathbf{x}. \tag{A.7}$$

First we note that $P_{maj} \leq 1$, and then derive a series of inequalities for P_{maj}. For any $\mathbf{x}$ where the majority vote is correct, at least k of the classifiers are correct. Thus,

$$\begin{aligned} \sum_{i=1}^{L} p_i &= \sum_{i=1}^{L} \int_{\mathbb{R}^n} \mathcal{I}_i(\mathbf{x}) p(\mathbf{x}) \, d\mathbf{x} = \int_{\mathbb{R}^n} \sum_{i=1}^{L} \mathcal{I}_i(\mathbf{x}) p(\mathbf{x}) \, d\mathbf{x} \\ &\geq \int_{\sum \mathcal{I}_i(\mathbf{x}) \geq k} k p(\mathbf{x}) d\mathbf{x} = k P_{maj}. \end{aligned} \tag{A.8}$$

Then

$$P_{maj} \leq \frac{1}{k} \sum_{i=1}^{L} p_i. \tag{A.9}$$

Let us now remove the most accurate member of the ensemble, D_L and consider the remaining $L-1$ classifiers

$$\sum_{i=1}^{L-1} p_i = \sum_{i=1}^{L-1} \int_{\mathbb{R}^n} \mathcal{I}_i(\mathbf{x}) p(\mathbf{x}) \, d\mathbf{x}. \tag{A.10}$$

For each point $\mathbf{x} \in \mathbb{R}^n$ where the majority vote (using the whole ensemble) has been correct, that is, $\sum \mathcal{I}_i(\mathbf{x}) \geq k$, there are now at least $k-1$ correct individual votes. Thus,

$$\begin{aligned} \sum_{i=1}^{L-1} p_i &= \int_{\mathbb{R}^n} \sum_{i=1}^{L-1} \mathcal{I}_i(\mathbf{x}) p(\mathbf{x}) \, d\mathbf{x}, \\ &\geq \int_{\sum_{i=1}^{L} \mathcal{I}_i(\mathbf{x}) \geq k} (k-1) p(\mathbf{x}) \, d\mathbf{x} = (k-1) P_{maj}. \end{aligned} \tag{A.11}$$

Then

$$P_{maj} \leq \frac{1}{k-1} \sum_{i=1}^{L-1} p_i. \tag{A.12}$$

Similarly, by dropping from the remaining set the most accurate classifier at a time, we can derive the series of inequalities (A.1). Note that we can remove *any* classifier from the ensemble at a time, not just the most accurate one, and arrive at a similar inequality. Take for example the step where we remove D_L. The choice of the most accurate classifier is dictated by the fact that the remaining ensemble of $L-1$ classifiers will have the smallest sum of the individual accuracies. So as P_{maj} is less than $\frac{1}{2} \sum_{i=1}^{L-1} p_i$, it will be less than any other sum involving $L-1$ classifiers which includes p_L and excludes a smaller p_i from the summation.

The next step is to show that the upper bound is achievable. Matan suggests to use induction on both L and k for that [277].

Lower Bound. To calculate the lower bound, Matan proposes to invert the concept, and look again for the upper bound but of $(1 - P_{maj})$.

A.2 Selected MATLAB code

```
%--------------------------------------------------------------------%
function oul = nb_combiner(otl,ree,rel)
% --- Naive Bayes (NB) combiner for label outputs
% Input: ----------------------------------------
```

```
%      otl:  outputs to label
%            = array N(objects)-by-L(classifiers)
%            entry (i,j) is the label of object i
%            by classifier j (integer labels)
%      ree:  reference ensemble
%            = array M(objects)-by-L(classifiers)
%            entry (i,j) is the label of object i
%            by classifier j (integer labels)
%      rel:  reference labels
%            = array M(objects)-by-1
%            true labels (integers)
% Output:  -------------------------------------
%     oul:   output labels
%            = array N(objects)-by-1
%            assigned labels (integers)

% Training ------------------------------------
c = max(rel); % number of classes, assuming that the
% class labels are integers 1,2,3,...,c
L = size(ree,2); % number of classifiers

for i = 1:c
    cN(i) = sum(rel == i); % class counts
end

for i = 1:L
    % cross-tabulate the classes to find the
    % confusion matrices
    for j1 = 1:c
        for j2 = 1:c
            CM(i).cm(j1,j2) = (sum(rel == j1 & ree(:,i) ...
                == j2) + 1/c) / (cN(j1) + 1);
            % correction for zeros included
        end
    end
end

% Operation -----------------------------------
N = size(otl,1);
oul = zeros(N,1); % pre-allocate for speed
for i = 1:N
    P = cN/numel(rel);
    for j = 1:c % calculate the score for each class
        for k = 1:L
            P(j) = P(j) * CM(k).cm(j,otl(i,k));
        end
    end
    [~,oul(i)] = max(P);
end
%-------------------------------------------------------------------------%
```

```
%-------------------------------------------------------------------------%
function oul = bks_combiner(otl,ree,rel)
% --- BKS combiner for label outputs
% Input: ----------------------------------------
```

```
%      otl:  outputs to label
%            = array N(objects)-by-L(classifiers)
%            entry (i,j) is the label of object i
%            by classifier j (integer labels)
%      ree:  reference ensemble
%            = array M(objects)-by-L(classifiers)
%            entry (i,j) is the label of object i
%            by classifier j (integer labels)
%      rel:  reference labels
%            = array M(objects)-by-1
%            true labels (integers)
% Output:  ------------------------------------
%     oul:   output labels
%            = array N(objects)-by-1
%            assigned labels (integers)

N = size(otl,1);
M = size(ree,1);
largest_class = mode(rel);
oul = zeros(N,1); % pre-allocate for speed
for i = 1:N
    matches = sum(ree ~= repmat(otl(i,:),M,1),2) == 0;
    if sum(matches)
        oul(i) = mode(rel(matches));
    else % there is no match in the reference
        % ensemble output; use the largest prior
        oul(i) = largest_class;
    end
end
%------------------------------------------------------------------------%
```

An example of using the BKS combiner function is shown below. Note that, with a minor edit of lines 31-35, the the BKS function can be replaced by the Naïve Bayes combiner function or any other combiner function.

```
%------------------------------------------------------------------------%
clear all, close all
clc

% Generate and plot the data
[~,~,labtrue] = fish_data(50,0);
[x,y,lb] = fish_data(50,20); figure, hold on
plot(x(lb == 1),y(lb == 1),'k.','markers',14)
plot(x(lb == 2),y(lb == 2),'k.','markers',14,...
    'color',[0.87, 0.87, 0.87])
axis([0 1 0 1]) % cut the figure to the unit square
axis square off % equalize and remove the axes

% Generate and plot the ensemble of linear classifiers
L = 50; % ensemble size
N = numel(x); % number of data points
ensemble = zeros(N,L); % pre-allocate for speed
for i = 1:L
    p = rand(1,2); % random point in the unit square
    w = randn(1,2); % random normal vector to the line
```

```
    w0 = p * w'; % the free term (neg)
    plot([0 1],[w0, (w0-w(1))]/w(2),'r-',...
        'linewidth',1.4) % plot the linear boundary
    plot(p(1),p(2),'r.','markersize',15)
    pause(0.08)
    t = 2  - ([x y] * w' - w0 > 0);
    if mean(t == lb) < 0.5, t = 3-t; end % revert labels
    ensemble(:,i) = t; % store output of classifier i
end
% Find and plot the BKS combiner output
output_bks = bks_combiner(ensemble,ensemble,lb);
accuracy_bks = mean(output_bks == labtrue);
plot(x(output_bks==1),y(output_bks==1),'bo','linewidth',1.5)
title(['BKS accuracy ',num2str(accuracy_bks)])
%--------------------------------------------------------------------%
```

CHAPTER 5

COMBINING CONTINUOUS-VALUED OUTPUTS

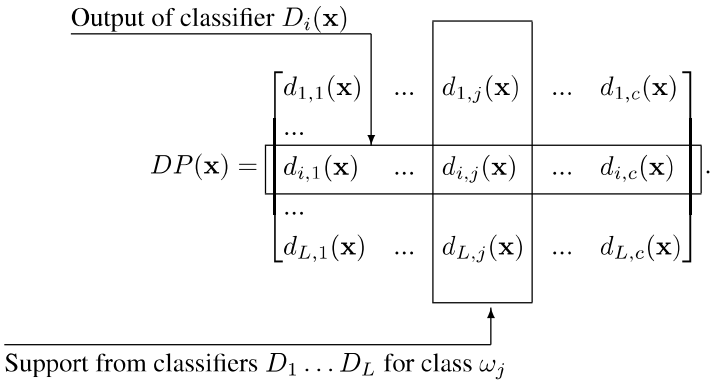
5.1 Decision profile

Consider the canonical model of a classifier illustrated in Figure 1.9. The degrees of support for a given input $\mathbf{x}$ can be interpreted in different ways, the two most common being *confidences* in the suggested labels and *estimates of the posterior probabilities* for the classes.

Let $\mathbf{x} \in \mathbb{R}^n$ be a feature vector and $\Omega = \{\omega_1, \omega_2, \ldots, \omega_c\}$ be the set of class labels. Each classifier D_i in the ensemble $\mathcal{D} = \{D_1, \ldots, D_L\}$ outputs c degrees of support. Without loss of generality we can assume that all c degrees are in the interval $[0, 1]$, that is, $D_i : \mathbb{R}^n \to [0, 1]^c$. Denote by $d_{i,j}(\mathbf{x})$ the support that classifier D_i gives to the hypothesis that $\mathbf{x}$ comes from class ω_j. The larger the support, the more likely the class label ω_j. The L classifier outputs for a particular input $\mathbf{x}$ can be organized in a **decision profile** ($DP(\mathbf{x})$) as the matrix shown in Figure 5.1.

The methods described in this chapter use $DP(\mathbf{x})$ to find the overall support for each class, and subsequently label the input $\mathbf{x}$ in the class with the largest support. There are two general approaches to this task. First, we can use the knowledge that the values in column j are the individual supports for class ω_j, and derive an overall support value for that class. Simple algebraic expressions, such as average

Combining Pattern Classifiers, second edition.
By Ludmila I. Kuncheva Copyright © 2014 John Wiley & Sons, Inc.



(5.1)

Figure 5.1 Decision profile for an input $\mathbf{x}$

or product, can be used for this. Alternatively, we may ignore the context of $DP(\mathbf{x})$ and treat the values $d_{i,j}(\mathbf{x})$ as features in a new feature space, which we call the *intermediate feature space*. The final decision is made by another classifier that takes the intermediate feature space as input, and produces a class label (stacked generalization). The important question is how we train such architectures to make sure that the increased complexity is justified by a corresponding gain in accuracy.

5.2 How do we get probability outputs?

Calibrating the classifiers' outputs is important, especially for heterogeneous ensembles [31]. Some of the base classifiers described in Chapter 2 produce soft labels right away. An example of such outputs is the discriminant scores of the linear discriminant classifier. It is more convenient though if these degrees were in the interval $[0, 1]$, with 0 meaning no support and 1 meaning full support. We can simply normalize the values so that $\mathbb{R}$ is mapped to $[0, 1]$. In addition, to comply with the probability context, we can scale the degrees of support so that their sum is one. The standard solution to this problem is the *softmax* method [107]. Let $g_1(\mathbf{x}), \ldots, g_c(\mathbf{x})$ be the output of classifier D. Then the new support scores $g'_1(\mathbf{x}), \ldots, g'_c(\mathbf{x})$, $g'_j(\mathbf{x}) \in [0, 1]$, $\sum_{j=1}^{c} g'_j(\mathbf{x}) = 1$, are obtained as

$$g'_j(\mathbf{x}) = \frac{\exp\{g_j(\mathbf{x})\}}{\sum_{k=1}^{c} \exp\{g_k(\mathbf{x})\}}. \tag{5.2}$$

It is desirable that $g'_j(\mathbf{x})$ are credible estimates of the probabilities for the classes given the input $\mathbf{x}$. Some ways to obtain continuous outputs as estimates of the posterior probabilities $P(\omega_j|\mathbf{x})$, $j = 1, \ldots, c$, are detailed below.

5.2.1 Probabilities based on discriminant scores

Consider the linear and the quadratic discriminant classifiers. These classifiers are optimal for normal class-conditional densities (a seldom valid but very useful assumption). The discriminant function for class ω_i, denoted $g_i(\mathbf{x})$, is derived from $\log P(\omega_i)p(\mathbf{x}|\omega_i)$ by dropping all the additive terms that do not depend on the class label as shown by equations (2.8) – (2.11).

For the linear discriminant classifier, we arrived at (2.11)

$$g_i(\mathbf{x}) = \log(P(\omega_i)) - \frac{1}{2}\boldsymbol{\mu}_i^T \Sigma^{-1} \boldsymbol{\mu}_i + \boldsymbol{\mu}_i^T \Sigma^{-1} \mathbf{x} = w_{i0} + \mathbf{w}_i^T \ \mathbf{x}. \tag{5.3}$$

Returning the dropped terms into the starting equation, we have

$$\begin{aligned} \log(P(\omega_i)p(\boldsymbol{\mu}|\omega_i)) &= \underbrace{\log(P(\omega_i)) - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)}_{g_i(\mathbf{x})} \\ &\quad -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log(|\Sigma|). \end{aligned} \tag{5.4}$$

Denote by C the constant (possibly depending on $\mathbf{x}$ but not on i) absorbing all dropped additive terms

$$C = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log(|\Sigma|). \tag{5.5}$$

Then

$$P(\omega_j)p(\mathbf{x}|\omega_j) = \exp(C) \ \exp\{g_j(\mathbf{x})\}. \tag{5.6}$$

The posterior probability for class ω_j for the given $\mathbf{x}$ is

$$P(\omega_j|\mathbf{x}) = \frac{P(\omega_j)p(\mathbf{x}|\omega_j)}{p(\mathbf{x})} \tag{5.7}$$

$$= \frac{\exp(C) \times \exp\{g_j(\mathbf{x})\}}{\sum_{k=1}^{c} \exp(C) \times \exp\{g_k(\mathbf{x})\}} = \frac{\exp\{g_j(\mathbf{x})\}}{\sum_{k=1}^{c} \exp\{g_k(\mathbf{x})\}}, \tag{5.8}$$

which is the softmax transform (5.2).

For a two-class problem, instead of comparing $g_1(\mathbf{x})$ with $g_2(\mathbf{x})$, we can form a single discriminant function $g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x})$, which we compare with the threshold 0. In this case,

$$P(\omega_1|\mathbf{x}) = \frac{1}{1+\exp\{-g(\mathbf{x})\}} \tag{5.9}$$

and

$$P(\omega_2|\mathbf{x}) = 1 - P(\omega_1|\mathbf{x}) = \frac{1}{1+\exp\{g(\mathbf{x})\}}. \tag{5.10}$$

This is also called the *logistic link function* [310].

Alternatively, we can think of $g(\mathbf{x})$ as a new feature, and estimate the two class-conditional pdfs, $p(g(\mathbf{x})|\omega_1)$ and $p(g(\mathbf{x})|\omega_2)$ in this new one-dimensional space. The posterior probabilities are calculated from the Bayes formula. The same approach can be extended to a c-class problem by estimating a class-conditional pdf $p(g_j(\mathbf{x})|\omega_j)$ on $g_j(\mathbf{x})$, $j = 1, \ldots, c$, and calculating subsequently $P(\omega_j|\mathbf{x})$.

Next, consider a neural network (NN) with c outputs, each corresponding to a class. Denote the NN output by

$$(y_1, \ldots, y_c) \in \mathbb{R}^c$$

and the target by

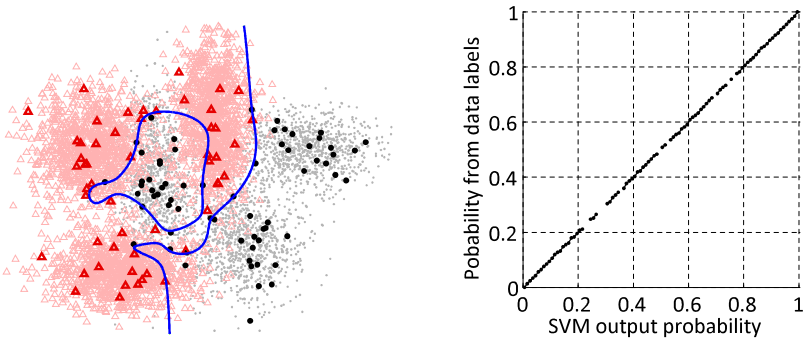
$$(t_1, \ldots, t_c) \in \{0, 1\}^c.$$

The target for an object $\mathbf{z}_j$ from the data set $\mathbf{Z}$ is typically a binary vector with 1 at the position of the class label of $\mathbf{z}_j$ and zeros elsewhere. It is known that, if trained to optimize the squared error between the NN output and the target, in the asymptotic case, the NN output y_j will be an approximation of the posterior probability $P(\omega_j|\mathbf{x})$, $j = 1, \ldots, c$ [40, 330]. Wei et al. [414] argue that the theories about the approximation are based on several assumptions that might be violated in practice: (i) that the network is sufficiently complex to model the posterior distribution accurately, (ii) that there are sufficient training data, and (iii) that the optimization routine is capable of finding the global minimum of the error function. The typical transformation which forms a probability distribution from $(y_1, \ldots, y_L)$ is the sotfmax transformation (5.2) [107]. Wei et al. [414] suggest a histogram-based remapping function. The parameters of this function are tuned separately from the NN training. In the operation phase, the NN output is fed to the remapping function and calibrated to give more adequate posterior probabilities.

EXAMPLE 5.1 SVM output calibration

Figure 5.2 (a) shows a two-dimensional data set with two classes plotted with different markers. Each class contains 3,000 data points. A training set of 120 points was randomly sampled. The training set is marked on the plot with thicker and brighter markers. The SVM classifier was trained using this training data. The classification boundary is shown on the scatterplot.

To examine how accurately the calibrated SVM output matches the posterior probabilities, the whole data set of 6,000 objects was fed to the trained SVM classifier and the outputs were calibrated into posterior probabilities using (5.10). Next, a histogram with 100 bins was created and the 6,000 probability estimates for class 1 were distributed in the respective bins. The true class labels of the objects were recovered and used to calculate the probability for class 1 in each bin. Figure 5.2 (b) shows the nearly perfect correspondence between the SVM probabilities and the probabilities calculated from the data labels.



(a) Data and SVM classifier (b) Correspondence between probabilities

Figure 5.2 Calibrated output for the SVM classifier on a 2-d, 2-class data set.

5.2.2 Probabilities based on counts: Laplace estimator

Consider finding probability estimates from decision trees. Each leaf of the tree defines a set of posterior probabilities. These are assigned to any point that reaches the leaf. Provost and Domingos [314] analyze the reasons for the insufficient capability of the standard decision tree classifiers to provide adequate estimates of the probabilities and conclude that the very heuristics that help us build small and accurate trees are responsible for that. Special amendments were proposed which led to the so called *Probability Estimating Trees* (PETs). These trees still have high classification accuracy but their main purpose is to give more accurate estimates of the posterior probabilities.

We calculate estimates of $P(\omega_j|\mathbf{x})$, $j = 1, \ldots, c$, as the class proportions of the training data points that reached the leaf (the *maximum likelihood* (ML) estimates). Let $k_1, \ldots, k_c$ be the number of training points from classes $\omega_1, \ldots, \omega_c$ respectively, at some leaf node t, and let $K = k_1 + \ldots + k_c$. The ML estimates are

$$\hat{P}(\omega_j|\mathbf{x}) = \frac{k_j}{K}, \quad j = 1, \ldots, c. \tag{5.11}$$

The problem is that when the total number of points, K, is small, the estimates of these probabilities are unreliable. Besides, the tree-growing strategies try to make the leaves as pure as possible. Thus, most probability estimates will be pushed towards 1 and 0 [430].

To remedy this, the *Laplace estimate* or *Laplace correction* can be applied [314, 384, 430]. The idea is to adjust the estimates so that they are less extreme. For c classes, the Laplace estimate used in [314] is

$$\hat{P}(\omega_j|\mathbf{x}) = \frac{k_j + 1}{K + c}. \tag{5.12}$$

Zadrozny and Elkan [430] apply a different version of the Laplace estimate using a parameter m which controls the degree of regularization of the estimate (called m-

estimation). The idea is to smooth the posterior probability towards the (estimate of the) prior probability for the class

$$\hat{P}(\omega_j|\mathbf{x}) = \frac{k_j + m \times \hat{P}(\omega_j)}{K + m}. \tag{5.13}$$

If m is large, then the estimate is close to the prior probability. If $m = 0$, then we have the ML estimates and no regularization. Zadrozny and Elkan suggest that m should be chosen so that $m \times P(\omega_j) \approx 10$ and also point out that practice has shown that the estimate (5.13) is quite robust with respect to the choice of m.

Suppose that ω^* is the majority class at node t. Ting and Witten [384] propose the following version of the Laplace estimator:

$$\hat{P}(\omega_j|\mathbf{x}) = \begin{cases} 1 - \frac{\sum_{l \neq j} k_l + 1}{K+2}, & \text{if } \omega_j = \omega^*, \\ (1 - \hat{P}(\omega^*|\mathbf{x})) \times \frac{k_j}{\sum_{l \neq j} k_l}, & \text{otherwise.} \end{cases} \tag{5.14}$$

The general consensus in the PET studies is that for good estimates of the posterior probabilities, the tree should be grown without pruning, and a form of the Laplace correction should be used for calculating the probabilities.

The same argument can be applied for smoothing the estimates of the k nearest neighbor classifier (k-nn) discussed in Chapter 2. There are many weighted versions of k-nn whereby the posterior probabilities are calculated using distances. While the distance-weighted versions have been found to be asymptotically equivalent to the non-weighted versions in terms of classification accuracy [24], there is no such argument when class ranking is considered. It is possible that the estimates of the soft k-nn versions are more useful for ranking than for labeling. A simple way to derive $\hat{P}(\omega_j|\mathbf{x})$ from k-nn is to average the similarities between $\mathbf{x}$ and its nearest neighbors from class ω_j. Let k be the number of neighbors, $\mathbf{x}^{(i)}$ be the i-th nearest neighbor of $\mathbf{x}$, and $d(\mathbf{x}, \mathbf{x}^{(i)})$ be the distance between $\mathbf{x}$ and $\mathbf{x}^{(i)}$. Then

$$\hat{P}(\omega_j|\mathbf{x}) = \frac{\sum_{\mathbf{x}^{(j)} \in \omega_j} \frac{1}{d(\mathbf{x}, \mathbf{x}^{(j)})}}{\sum_{i=1}^{k} \frac{1}{d(\mathbf{x}, \mathbf{x}^{(i)})}}. \tag{5.15}$$

Albeit intuitive, these estimates are not guaranteed to be good approximations of the posterior probabilities.

EXAMPLE 5.2 Laplace corrections and soft k-nn

Figure 5.3 shows a point in a 2-dimensional feature space (the cross, $\mathbf{x}$) and its 7 nearest neighbors from ω_1 (open circles), ω_2 (bullets) and ω_3 (triangle).

The Euclidean distances between $\mathbf{x}$ and its neighbors are as follows:

$\mathbf{x}$	1	2	3	4	5	6	7
Distance	1	1	$\sqrt{2}$	2	$2\sqrt{2}$	$2\sqrt{2}$	$\sqrt{13}$
Label	ω_2	ω_1	ω_1	ω_3	ω_1	ω_1	ω_2

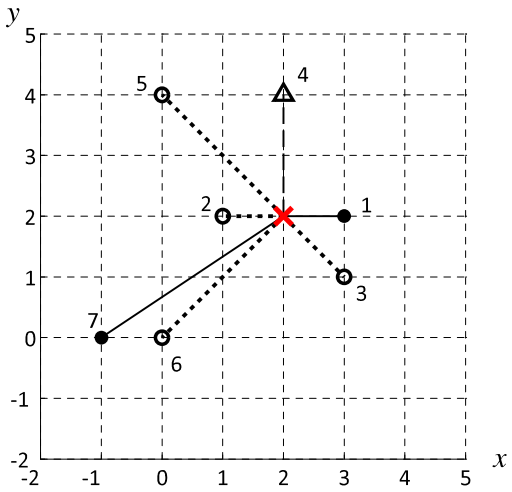


Figure 5.3 A point in a 2-dimensional feature space and its 7 nearest neighbors from ω_1 (open circles), ω_2 (bullets) and ω_3 (triangle).

The probability estimates $\mu_j(\mathbf{x})$, $j = 1, 2, 3$, using the Laplace corrections and the distance-based formula are shown in Table 5.1. As seen in the table, all the corrective modifications of the estimates bring them closer to one another compared to the standard ML estimates, i.e., the modifications smooth the estimates away from the 0/1 bounds.

Table 5.1 Probability estimates for the example in Figure 5.3 using the Laplace corrections and distance-based k-nn

Method	$\mu_1(\mathbf{x})$	$\mu_2(\mathbf{x})$	$\mu_3(\mathbf{x})$
ML	$\frac{4}{7} = 0.571$	$\frac{2}{7} = 0.286$	$\frac{1}{7} = 0.143$
Standard Laplace [314]	$\frac{5}{10} = 0.500$	$\frac{3}{10} = 0.300$	$\frac{2}{10} = 0.200$
m-estimation [430]	$\frac{8}{19} = 0.421$	$\frac{6}{19} = 0.316$	$\frac{5}{19} = 0.263$
($m = 12$, equiprobable classes)			
Ting and Witten [384]	$\frac{12}{27} = 0.444$	$\frac{10}{27} = 0.370$	$\frac{5}{27} = 0.185$
Distance-based	0.576	0.290	0.134

Accurate estimates are a sufficient but not a necessary condition for a high classification accuracy. The final class label will be correctly assigned as long as the degree of support for the correct class label exceeds the degrees for the other classes. Investing effort into refining the probability estimates will be justified in problems with a large number of classes c, where the ranking of the classes by their likelihood is more im-

portant than identifying just one winning label. Examples of such tasks are person identification, text categorization, and fraud detection.

5.3 Non-trainable (fixed) combination rules

5.3.1 A generic formulation

Simple fusion methods are the most obvious choice when constructing a multiple classifier system [207, 237, 382, 394, 399]. A degree of support for class ω_j is calculated from the L entries in the j-th column of $DP(\mathbf{x})$

$$\mu_j(\mathbf{x}) = \mathcal{F}\left(d_{1,j}(\mathbf{x}), \ldots d_{L,j}(\mathbf{x})\right), \tag{5.16}$$

where $\mathcal{F}$ is a combination function. The class label of $\mathbf{x}$ is found as the index of the maximum $\mu_j(\mathbf{x})$. $\mathcal{F}$ can be chosen in many different ways:

- Average (Sum)

$$\mu_j(\mathbf{x}) = \frac{1}{L}\sum_{i=1}^{L} d_{i,j}(\mathbf{x}). \tag{5.17}$$

- Minimum/maximum/median combiner, for example,

$$\mu_j(\mathbf{x}) = \max_i\{d_{i,j}(\mathbf{x})\}. \tag{5.18}$$

- Trimmed mean combiner (competition jury). For a $K\%$ trimmed mean the L degrees of support are sorted and $\frac{K}{2}\%$ of the values are dropped on each side. The overall support $\mu_j(\mathbf{x})$ is found as the mean of the remaining degrees of support.

- Product combiner

$$\mu_j(\mathbf{x}) = \prod_{i=1}^{L} d_{i,j}(\mathbf{x}). \tag{5.19}$$

Represented by the Average combiner, the category of simple non-trainable combiners is described in Figure 5.4, and illustrated diagrammatically in Figure 5.5. These combiners are called non-trainable, because once the individual classifiers are trained, their outputs can be fused to produce an ensemble decision, without any further training.

EXAMPLE 5.3 Simple non-trainable combiners

AVERAGE COMBINER

Training: None

Operation: For each new object

1. Classify the new object $\mathbf{x}$ to find its decision profile $DP(\mathbf{x})$, as in equation (5.1).
2. Calculate the support for each class by

$$P(k) = \frac{1}{L} \sum_{i=1}^{L} d_{j,i}, \quad k = 1, \ldots, c.$$

3. Assign label k^* to the object, where

$$k^* = \arg\max_{k=1}^{c} P(k).$$

Return the ensemble label of the new object.

Figure 5.4 Training and operation algorithm for the Average combiner.

The following example helps to clarify simple combiners. Let $c = 3$ and $L = 5$. Assume that for a certain $\mathbf{x}$

$$DP(\mathbf{x}) = \begin{bmatrix} 0.1 & 0.5 & 0.4 \\ 0.0 & 0.0 & 1.0 \\ 0.4 & 0.3 & 0.4 \\ 0.2 & 0.7 & 0.1 \\ 0.1 & 0.8 & 0.2 \end{bmatrix}. \tag{5.20}$$

Applying the simple combiners column-wise, we obtain

Combiner	$\mu_1(\mathbf{x})$	$\mu_2(\mathbf{x})$	$\mu_3(\mathbf{x})$
Average	0.16	0.46	0.42
Minimum	0.00	0.00	0.10
Maximum	0.40	0.80	1.00
Median	0.10	0.50	0.40
40% trimmed mean	0.13	0.50	0.33
Product	0.00	0.00	0.0032

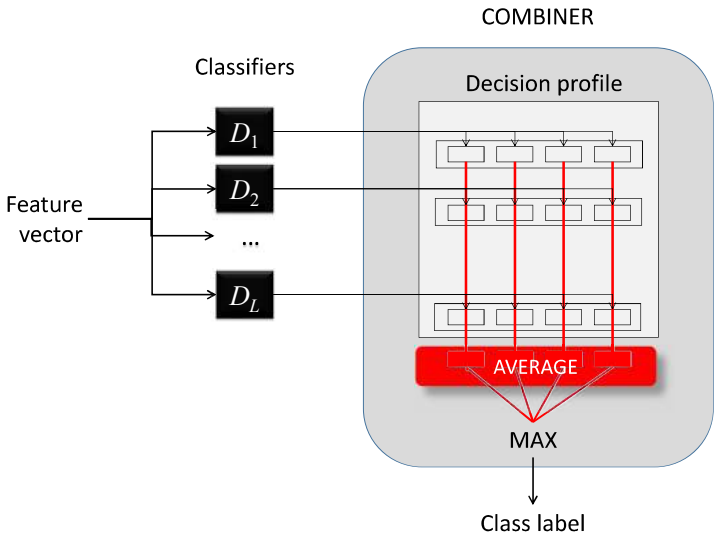


Figure 5.5 Operation of the Average combiner.

Note that we do not require that $d_{i,j}(\mathbf{x})$ for classifier D_i sum up to one. We only assume that they are measured in the same units. If we take the class with the maximum support to be the class label of $\mathbf{x}$, the minimum, maximum, and product will label $\mathbf{x}$ in class ω_3, whereas the average, the median and the trimmed mean will put $\mathbf{x}$ in class ω_2.

5.3.2 Equivalence of simple combination rules

5.3.2.1 Equivalence of MINIMUM and MAXIMUM combiners for two classes

Let $\mathcal{D} = \{D_1, \ldots, D_L\}$ be the classifier ensemble and $\Omega = \{\omega_1, \omega_2\}$ be the set of class labels. The individual outputs are estimates of the posterior probabilities. The output $d_{i,j}$ of classifier D_i (supporting the hypothesis that $\mathbf{x}$ comes from class ω_j) is an estimate of $P(\omega_j|\mathbf{x})$, $j = 1, 2$. Here we prove that the minimum and the maximum combiners are equivalent for $c = 2$ classes and any number of classifiers L, provided the two outputs from each classifier satisfy

$$\hat{P}(\omega_1|\mathbf{x}) + \hat{P}(\omega_2|\mathbf{x}) = 1.$$

This equivalence means that the class label assigned by the minimum and the maximum combiners will be the same. In case of a tie for one of the rules, there will be a tie for the other rule as well, and any of the two class labels could be assigned in both cases.

Proposition. *Let $a_1, \ldots, a_L$ be the L outputs for class ω_1, and $1 - a_1, \ldots, 1 - a_L$ be the L outputs for class ω_2, $a_i \in [0, 1]$. Then the class label assigned to $\mathbf{x}$ by the MINIMUM and MAXIMUM combination rules is the same.*

Proof. Without loss of generality assume that $a_1 = \min_i a_i$, and $a_L = \max_i a_i$. Then the minimum combination rule will pick a_1 and $1 - a_L$ as the support for ω_1 and ω_2 respectively, and the maximum rule will pick a_L and $1 - a_1$. Consider the three possible relationships between a_1 and $1 - a_L$.

(a) If $a_1 > 1 - a_L$ then $a_L > 1 - a_1$, and the selected class is ω_1 *with both methods.*

(b) If $a_1 < 1 - a_L$ then $a_L < 1 - a_1$, and the selected class is ω_2 *with both methods.*

(c) If $a_1 = 1 - a_L$ then $a_L = 1 - a_1$, and we will pick a class at random *with both methods.*

Note: A discrepancy between the error rates of the two combination methods might occur in numerical experiments due to the random tie-break in (c). If we agree to always assign class ω_1 when the support for the two classes is the same (a perfectly justifiable choice), the results for the two methods will coincide.

5.3.2.2 Equivalence of MAJORITY VOTE and MEDIAN combiner for two classes and odd L

Consider again the case of two classes, and L classifiers with outputs for a certain $\mathbf{x}$, $a_1, \ldots, a_L$, for class ω_1, and $1 - a_1, \ldots, 1 - a_L$, for class ω_2, where L is odd.

Proposition. *The class label assigned to $\mathbf{x}$ by the MAJORITY VOTE rule and MEDIAN combination rule is the same.*

Proof. Assume again that $a_1 = \min_i a_i$, and $a_L = \max_i a_i$. Consider the median rule first. The median of the outputs for class ω_1 is $a_{\frac{L+1}{2}}$.

(a) If $a_{\frac{L+1}{2}} > 0.5$, then the median of the outputs for ω_2, $1 - a_{\frac{L+1}{2}} < 0.5$, and class ω_1 will be assigned. The fact that $a_{\frac{L+1}{2}} > 0.5$ means that all $a_{\frac{L+1}{2}+1}, \ldots, a_L$ are strictly greater than 0.5. This makes at least $\frac{L+1}{2}$ posterior probabilities for ω_1 greater than 0.5, which, when 'hardened', will give label ω_1. Then the majority vote rule will assign to $\mathbf{x}$ class label ω_1.

(b) Alternatively, if $a_{\frac{L+1}{2}} < 0.5$, then $1 - a_{\frac{L+1}{2}} > 0.5$, and class ω_2 will be assigned by the median rule. In this case, at least $\frac{L+1}{2}$ posterior probabilities for ω_2 are greater than 0.5, and the majority vote rule will assign label ω_2 as well.

(c) For $a_{\frac{L+1}{2}} = 0.5$ a tie occurs, and any of the two labels can be assigned by

the median rule. The same applies for the majority vote, as all the soft votes at 0.5 (same for both classes) can be 'hardened' to any of the two class labels.

Again, a difference in the estimated errors of the two methods might occur in experiments due to the arbitrary 'hardening' of label 0.5. For example, if we agree to always assign class ω_1 when the posterior probabilities are both 0.5, the results for the two methods will coincide.

5.3.3 Generalized mean combiner

The *generalized mean* [105] is a useful aggregation formula governed by a parameter. Applied in the classifier combination context, the ensemble output for class ω_j is

$$\mu_j(\mathbf{x}, \alpha) = \left(\frac{1}{L} \sum_{i=1}^{L} d_{i,j}(\mathbf{x})^{\alpha} \right)^{\frac{1}{\alpha}}, \tag{5.21}$$

where α is the parameter. Some special cases of the generalized mean are shown in Table 5.2.

Table 5.2 Special cases of the generalized mean

$\alpha \to -\infty$	$\Rightarrow$	$\mu_j(\mathbf{x}, \alpha) = \min_i \{d_{i,j}(\mathbf{x})\}$	minimum
$\alpha = -1$	$\Rightarrow$	$\mu_j(\mathbf{x}, \alpha) = \left(\frac{1}{L} \sum_{i=1}^{L} \frac{1}{d_{i,j}(\mathbf{x})} \right)^{-1}$	harmonic mean
$\alpha \to 0$	$\Rightarrow$	$\mu_j(\mathbf{x}, \alpha) = \left(\prod_{i=1}^{L} d_{i,j}(\mathbf{x}) \right)^{1/L}$	geometric mean
$\alpha = 1$	$\Rightarrow$	$\mu_j(\mathbf{x}, \alpha) = \frac{1}{L} \sum_{i=1}^{L} d_{i,j}(\mathbf{x})$	arithmetic mean
$\alpha \to \infty$	$\Rightarrow$	$\mu_j(\mathbf{x}, \alpha) = \max_i \{d_{i,j}(\mathbf{x})\}$	maximum

Observe that the geometric mean is equivalent to the product combiner. Raising to the power of $\frac{1}{L}$ is a monotonic transformation which does not depend on the class label j, and therefore will not change the order of $\mu_j(\mathbf{x})$s. Hence the winning label obtained from the product combiner will be the same as the winning label from the geometric mean combiner.

As we are considering non-trainable combiners here, we assume that the system designer chooses α beforehand. This parameter can be thought of as the 'level of optimism' of the combiner. The *minimum* combiner ($\alpha \to -\infty$) is the most pessimistic choice. With this combiner, we know that ω_j is supported by *all* members of the ensemble at least as much as $\mu_j(\mathbf{x})$. At the other extreme, *maximum* is the most optimistic combiner. Here we accept an ensemble degree of support $\mu_j(\mathbf{x})$ on the ground that *at least one* member of the team supports ω_j with this degree.. If

we choose to tune α with respect to the ensemble performance, then we should regard the generalized mean combiner as a trainable combiner as discussed later. The generalized mean combiner is detailed in Figure 5.6.

GENERALIZED MEAN COMBINER

Training: Choose the level of optimism α (see Table 5.2).

Operation: For each new object:

1. Classify the new object $\mathbf{x}$ to find its decision profile $DP(\mathbf{x})$, as in equation (5.1).
2. Calculate the support for each class by

$$P(k) = \left(\frac{1}{L} \sum_{i=1}^{L} d_{i,k}(\mathbf{x})^{\alpha} \right)^{\frac{1}{\alpha}}, \quad k = 1, \ldots, c.$$

3. Assign label k^* to the object, where

$$k^* = \arg\max_{k=1}^{c} P(k).$$

Return the ensemble label of the new object.

Figure 5.6 Training and operation algorithm for the Generalized Mean combiner.

EXAMPLE 5.4 Effect of the level of optimism α.

To illustrate the effect of the level of optimism α we used the 2-D rotated checker board data set. Examples of a training and a testing data sets are shown in Figure 5.7 (a) and (b), respectively.

One hundred training/testing sets were generated from a uniform distribution within the unit square. The labels of the points were assigned as in the rotated checker board example. In each experiment, the training set consisted of 200 examples and the testing set consisted of 1,000 examples. Each ensemble was formed by taking 10 bootstrap samples of size 200 from the training data (uniform sampling with replacement) and training a classifier on each sample. We chose SVM as the base ensemble classifier.[1] A Gaussian kernel with spread $\sigma = 0.3$ was applied, with a penalizing constant $C = 50$. The generalized mean formula (5.21) was used, where the level of optimism α was varied from -50 to

[1]The version we used is the SVM implementation within the Bioinformatics toolbox of MATLAB.

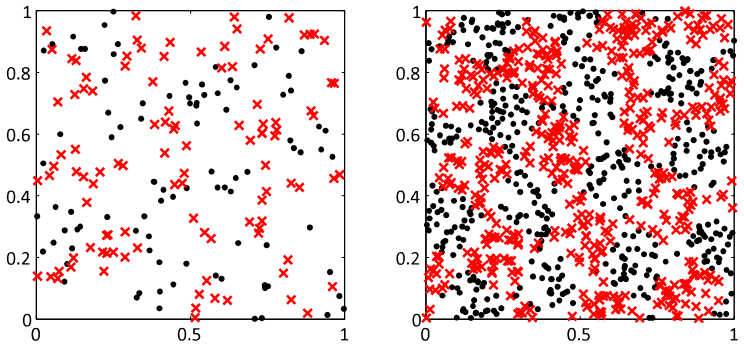
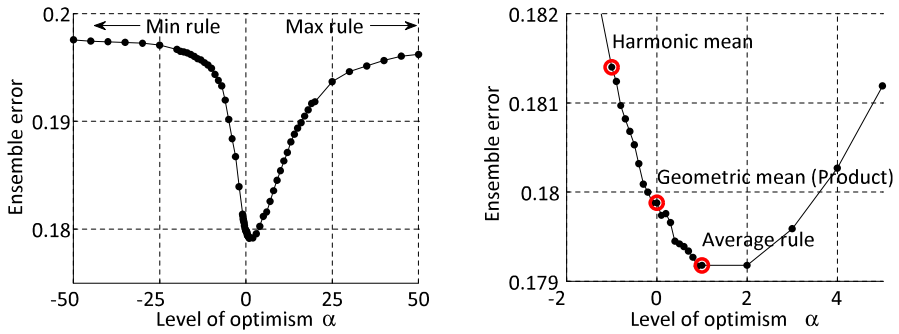


Figure 5.7 An example of a training (a) and testing (b) set for the rotated checker board data. One hundred randomly sampled training/testing sets were used in the experiment.

50 with finer discretization from -1 to 1. The ensemble error, averaged across the 100 runs, is plotted against α in Figure 5.8 (a).



(a) Ensemble error for $-50 \leq \alpha \leq 50$

(b) Zoom for $\alpha \in [-2, 5]$

Figure 5.8 Generalized mean combiner for the checkerboard data.

A zoom window of the ensemble error for $\alpha \in [-2, 5]$ is shown in Figure 5.8 (b). The average, product and harmonic mean combiners are identified on the curve. For this example, the average combiner gave the best result.

The results from the illustration above should not be taken as evidence that the average combiner is always the best. The shape of the curve will depend heavily on the problem and on the base classifier used. The average and the product are the two most popular combiners. Yet, there is no guideline as to which one is better for a specific problem. The current understanding is that the average, in general, might be

less accurate than the product for some problems, but is the more stable of the two [8, 109, 207, 281, 382, 383].

Even though Figure 5.8 shows a clear trend, the actual difference between the ensemble classification errors for the different combiners is negligible. Much more accuracy can be gained (or lost) by changing the width of the Gaussian kernel σ, or the penalizing constant C.

5.3.4 A theoretical comparison of simple combiners

Can we single out a combiner that performs best in a simple scenario? Consider the following set-up [9, 235]:

- There are only two classes, $\Omega = \{\omega_1, \omega_2\}$.
- All classifiers produce soft class labels, $d_{j,i}(\mathbf{x}) \in [0, 1]$, $i = 1, 2$, $j = 1, \ldots, L$, where $d_{j,i}(\mathbf{x})$ is an estimate of the posterior probability $P(\omega_i|\mathbf{x})$ by classifier D_j for an input $\mathbf{x} \in \mathbb{R}^n$. We consider the case where for any $\mathbf{x}$, $d_{j,1}(\mathbf{x}) + d_{j,2}(\mathbf{x}) = 1, j = 1, \ldots, L$.
- Let $\mathbf{x} \in \mathbb{R}^n$ be a data point to classify. Without loss of generality, we assume that the true posterior probability is $P(\omega_1|\mathbf{x}) = p > 0.5$. Thus, the Bayes-optimal class label for $\mathbf{x}$ is ω_1, and a classification error occurs if label ω_2 is assigned.

Assumption. The classifiers commit independent and identically distributed errors in estimating $P(\omega_1|\mathbf{x})$ such that

$$d_{j,1}(\mathbf{x}) = P(\omega_1|\mathbf{x}) + \eta(\mathbf{x}) = p + \eta(\mathbf{x}), \tag{5.22}$$

and respectively

$$d_{j,2}(\mathbf{x}) = 1 - p - \eta(\mathbf{x}), \tag{5.23}$$

where $\eta(\mathbf{x})$ has

(i) a normal distribution with mean 0 and variance σ^2 (we take σ to vary between 0.1 and 1)

(ii) a uniform distribution spanning the interval $[-b, +b]$ (b varies from 0.1 to 1).

Thus $d_{j,1}(\mathbf{x})$ is a random variable with normal or uniform distribution and so is $d_{j,2}(\mathbf{x})$.

The combiners we compare are minimum (same as maximum), average and median combiners [235] (equation (5.16)). As the median and the majority vote combiners are also identical for two classes, the comparison includes majority vote as well. Finally, we include the individual classification rate and an 'oracle' combiner which predicts the correct class label if at least one classifier in the ensemble gives a correct prediction. The performance of the combination rules is expected to be better than that of the individual classifier but worse than that of the oracle.

Table 5.3 shows the analytical expressions of the probability of error for the combiners under the normal distribution assumption, and Table 5.4, for the uniform distribution. The derivations of the expressions are shown in Appendix A.1. As explained in the appendix, there is no closed-form expression for the Min/Max combiner for the normal distribution of the estimation error, so these combiners were taken into the comparison only for the uniform distribution.

Table 5.3 The theoretical error P_e for the single classifier and the six fusion methods for the normal distribution.

Method	Ensemble error rate P_e
Single classifier	$\Phi\left(\frac{0.5-p}{\sigma}\right)$ (Individual error rate)
Min/Max	–
Average (Sum)	$\Phi\left(\frac{\sqrt{L}(0.5-p)}{\sigma}\right)$
Median/Majority	$\sum_{j=\frac{L+1}{2}}^{L} \binom{L}{j} \times \Phi\left(\frac{0.5-p}{\sigma}\right)^{j} \times \left[1-\Phi\left(\frac{0.5-p}{\sigma}\right)\right]^{L-j}$
Oracle	$\Phi\left(\frac{0.5-p}{\sigma}\right)^{L}$

Notes:
L is the number of classifiers in the ensemble;
p is the true posterior probability $P(\omega_1|\mathbf{x})$ for class ω_1 for the given object $\mathbf{x}$;
$\Phi(.)$ is the cumulative distribution function for the standard normal distribution $N(0,1)$.

Figures 5.9 and 5.10 show the ensemble error rate for the normal and uniform distributions, respectively, as a function of two arguments: the true posterior probability $P(\omega_1|\mathbf{x}) = p$ and the parameter of the distribution. For the normal distribution (Figure 5.9), σ took values from 0.1 to 1, and for the uniform distribution, b took values from 0.1 to 1, ensuring that $p - b < 0.5$. The posterior probability p was varied from 0.5 to 1 for both figures. The ensemble size for this example was $L = 5$ classifiers.

The surfaces in both figures are clearly layered beneath one another. Expectedly, the top surface (largest error) is the individual classifier while the bottom layer (smallest error) is the oracle combiner. Further on, when p is close to 0.5, the Bayes error is high, and so is the ensemble error. The ensemble error goes down to 0 for

Table 5.4 The theoretical error P_e for the single classifier and the six fusion methods for the uniform distribution ($p - b < 0.5$).

Method	Ensemble error rate P_e
Single classifier	$\frac{0.5 - p + b}{2b}$ (Individual error rate.)
Min/Max	$\frac{1}{2}\left(\frac{1-2p}{2b} + 1\right)^L$
Average	$\Phi\left(\frac{\sqrt{3L}(0.5-p)}{b}\right)$
Median/Majority	$\sum_{j=\frac{L+1}{2}}^{L} \binom{L}{j} \times \left(\frac{0.5-p+b}{2b}\right)^j \times \left[1 - \frac{0.5-p+b}{2b}\right]^{L-j}$
Oracle	$\left(\frac{0.5-p+b}{2b}\right)^L$

Notes:
L is the number of classifiers in the ensemble;
p is the true posterior probability $P(\omega_1|\mathbf{x})$ for class ω_1 for the given object $\mathbf{x}$;
$\Phi(.)$ is the cumulative distribution function for the standard normal distribution $N(0, 1)$.

a higher p and a lower variability of the estimate (low σ and low b), and does so quicker for the better combiners.

The average and the median/vote methods have a closer performance for normally distributed than for the uniformly distributed η, the average being the better of the two. Finally, for the uniform distribution, the average combiner is outperformed by the minimum/maximum combiner.

Figure 5.11 shows the behavior of the combiners as a function of the ensemble size L. We chose a fairly difficult problem where the true posterior probability is 0.55 (high uncertainty), and the spread parameter of the distributions is large ($\sigma = 0.9$ for the normal distribution, and $b = 0.9$ for the uniform distribution. The figure confirms that the above observations hold for any number of classifiers. It also indicates that larger ensembles secure a smaller classification error, and amplify the performance differences of the combiners. Even though this analysis is based on assumptions and theory, it suggests that checking several combiners for a set of trained classifiers may pay off.

Similar analyses can be carried out for distributions other than normal or uniform. Kittler and Alkoot [206], Chen and Cheng [71] and Cabrera [62] studied the behav-

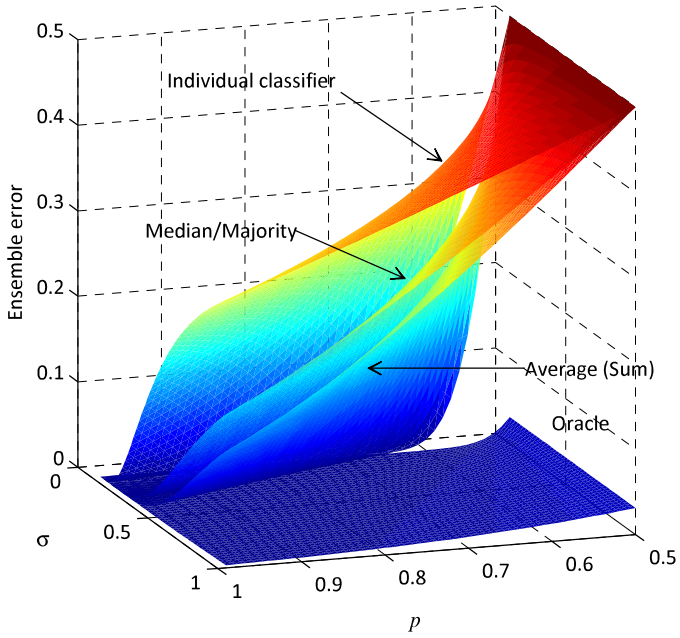


Figure 5.9 Ensemble error rate of the individual classifier and the simple combiners for normal distribution of the estimation error η.

ior of various combiners for non-normal distributions and large ensemble sizes. The conclusion is that for non-symmetrical, bimodal or heavy-tailed distributions, the combiners may have very different performances. Even though the assumptions may not hold in real-life problems, these analyses suggest that choosing a suitable combiner is important.

We should be aware of the following caveat. Although the combiners in this section are considered non-trainable, any comparison for the purpose of picking one among them is, in fact, a form of training. Choosing a combiner is the same as tuning the level of optimism α of the generalized mean combiner. We look at the question "to train or not to train" later in this chapter.

5.3.5 Where do they come from?

5.3.5.1 Intuition and common sense. Many simple combiners come from intuition and common sense. Figure 5.12 shows an example. Suppose that we want to bet on a horse, and have 3 choices. One of the horses will win the race, and the classification task is to predict which horse. The only information we have access to is the opinions of four friends. Each friend offers a guess of the probability for each horse to win the race. The friends are the classifiers in the ensemble, and the

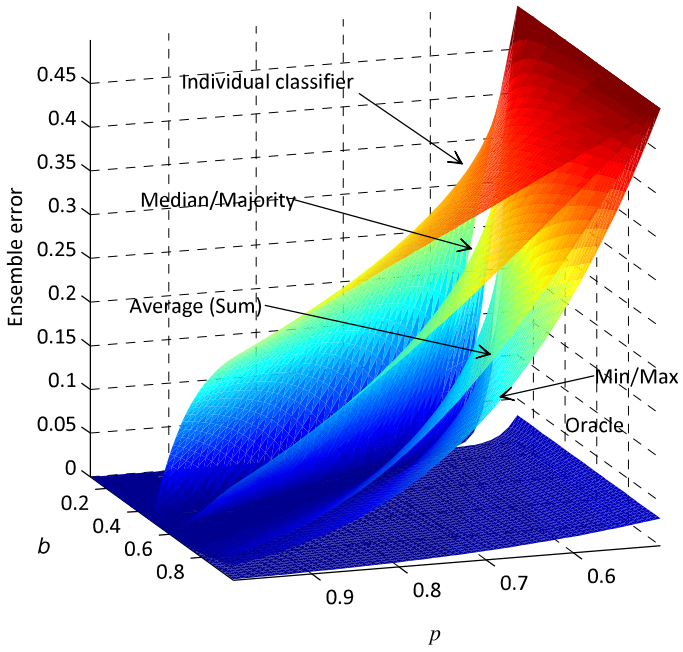


Figure 5.10 Ensemble error rate of the individual classifier and the simple combiners for uniform distribution of the estimation error η.

probabilities they predicted are arranged in a decision profile, as shown in the figure. The decision maker has to decide which combination rule to apply. If there is no further information about how accurate the predictions might be, the decision maker can choose their level of optimism, and pick the respective combiner. Take the over-conservative minimum combiner, for example. Given a horse, the support for this 'class', denoted μ, can be interpreted in the following way. All experts agree to at least a degree μ that the horse will win. It makes sense therefore, to choose the horse with the largest μ. The opposite strategy is to choose the horse that achieved the highest degree of support among all horses and all experts. In this case, there is at least one expert that believes in this horse with a degree this high. Both minimum and maximum combiners disregard the consensus opinion. Conversely, the average (sum), median and the jury combiners measure a central tendency of the support for the classes. The decision maker might reach a different conclusion depending on the combination rule they apply. In absence of a ground truth, we cannot judge whether the decision was right. This example merely demonstrates the flexibility of the simple combiners.

Interestingly, many simple combiners can be derived as the *optimal* combiner under various scenarios and assumptions.

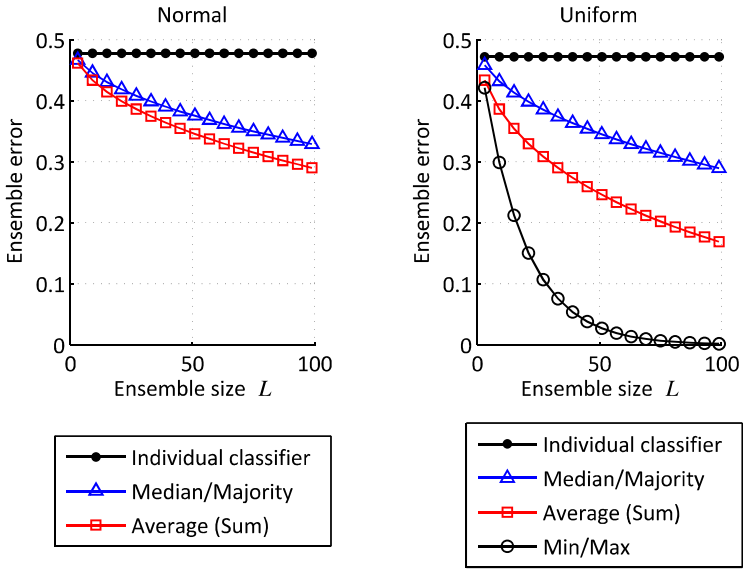


Figure 5.11 Ensemble error for the simple combiners as a function of the ensemble size L

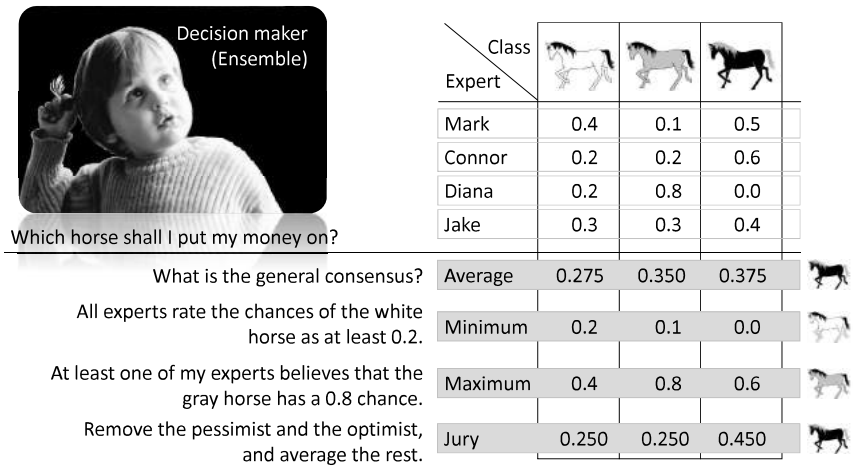


Figure 5.12 Simple combiners come from intuition and common sense.

5.3.5.2 Conditional independence of features. We can regard $d_{i,j}(\mathbf{x})$ as an estimate of the posterior probability $P(\omega_j|\mathbf{x})$ produced by classifier D_i. Finding an optimal (in Bayesian sense) combination of these estimates is not straightforward. Here we give a brief account of some of the theories underpinning the most common simple combiners.

Consider L different conditionally independent feature subsets. Each subset generates a part of the feature vector, $\mathbf{x}^{(i)}$, so that $\mathbf{x} = [\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(L)}]^T$, $\mathbf{x} \in \mathbb{R}^n$. For example, suppose that there are $n = 10$ features altogether and these are grouped in the following $L = 3$ conditionally independent subsets (1,4,8), (2,3,5,6,10) and (7,9). Then

$$\mathbf{x}^{(1)} = [x_1, x_4, x_8]^T, \quad \mathbf{x}^{(2)} = [x_2, x_3, x_4, x_6, x_{10}]^T, \quad \mathbf{x}^{(3)} = [x_7, x_9]^T.$$

From the assumed independence, the class-conditional pdf for class ω_j is a product of the class-conditional pdfs on each feature subset (representation)

$$p(\mathbf{x}|\omega_j) = \prod_{i=1}^{L} p(\mathbf{x}^{(i)}|\omega_j). \tag{5.24}$$

Deriving the product rule weighted by the prior probabilities as the optimal combiner for this case is straightforward [44, 231, 256]. The j-th output of classifier D_i is an estimate of the probability

$$P(\omega_j|\mathbf{x}^{(i)}, D_i) = \frac{P(\omega_j)p(\mathbf{x}^{(i)}|\omega_j)}{p(\mathbf{x}^{(i)})}, \tag{5.25}$$

hence

$$p(\mathbf{x}^{(i)}|\omega_j) = \frac{P(\omega_j|\mathbf{x}^{(i)})p(\mathbf{x}^{(i)})}{P(\omega_j)}. \tag{5.26}$$

The posterior probability using the whole of $\mathbf{x}$ is

$$P(\omega_j|\mathbf{x}) = \frac{P(\omega_j)p(\mathbf{x}|\omega_j)}{p(\mathbf{x})} = \frac{P(\omega_j)}{p(\mathbf{x})} \prod_{i=1}^{L} p(\mathbf{x}^{(i)}|\omega_j). \tag{5.27}$$

Substituting (5.26) into (5.27),

$$P(\omega_j|\mathbf{x}) = P(\omega_j)^{(1-L)} \prod_{i=1}^{L} P(\omega_j|\mathbf{x}^{(i)}) \times \frac{\prod_{i=1}^{L} p(\mathbf{x}^{(i)})}{p(\mathbf{x})}. \tag{5.28}$$

The fraction at the end does not depend on the class label k therefore we can ignore it when calculating the support $\mu_j(\mathbf{x})$ for class ω_j. Taking the classifier output $d_{i,k}(\mathbf{x}^{(i)})$ as the estimate of $P(\omega_j|\mathbf{x}^{(i)})$ and estimating the prior probabilities for the classes from the data, the support for ω_j is calculated as the product combination rule

$$P(\omega_j|\mathbf{x}) \propto P(\omega_j)^{(1-L)} \prod_{i=1}^{L} P(\omega_j|\mathbf{x}^{(i)}) \tag{5.29}$$

$$= \hat{P}(\omega_j)^{(1-L)} \prod_{i=1}^{L} d_{i,k}(\mathbf{x}^{(i)}) = \mu_j(\mathbf{x}). \tag{5.30}$$

Kittler et al. [207] take this formula further to derive the average combiner. They investigate the error sensitivity of the two combiners and show that the average combiner is much more resilient to estimation errors of the posterior probabilities than the product combiner. The product combiner is over sensitive to estimates close to zero. Presence of such estimates has the effect of veto on that particular class regardless of how large some of the estimates of other classifiers might be.

5.3.5.3 Kullback-Leibler divergence. Miller and Yan [281] offer a theoretical framework for the average and product combiners based on the *Kullback-Leibler divergence* (KL). KL divergence measures the distance between two probability distributions, q (prior distribution) and p (posterior distribution). KL divergence is also called 'relative entropy' or 'cross-entropy', denoted by $KL(p \parallel q)$.[2] It can be interpreted as the amount of information necessary to change the prior probability distribution q into posterior probability distribution p. For a discrete x,

$$KL(p \parallel q) = \sum_x p(x) \log_2 \left(\frac{p(x)}{q(x)} \right). \tag{5.31}$$

For identical distributions, the KL divergence is zero. We regard each row of $DP(\mathbf{x})$ as a prior probability distribution on the set of class labels Ω and use $d_{i,j}(\mathbf{x})$ to denote the estimate of the probability $P(\omega_j|\mathbf{x}, D_i)$. Denote by $P_{(i)}$ the probability distribution on Ω provided by classifier D_i, i.e., $P_{(i)} = (d_{i,1}(\mathbf{x}), \ldots, d_{i,c}(\mathbf{x}))$. For example, let $DP(\mathbf{x})$ be

$$DP(\mathbf{x}) = \begin{bmatrix} 0.3 & 0.7 \\ 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix}.$$

Then $P_{(1)} = (0.3, 0.7)$ is the pmf on $\Omega = \{\omega_1, \omega_2\}$ due to classifier D_1.

Given the L sets of probability estimates, one for each classifier, our first hypothesis is that the true values of $P(\omega_i|\mathbf{x})$ (posterior probabilities) are the ones most agreed upon by the ensemble $\mathcal{D} = \{D_1, \ldots, D_L\}$. Denote these agreed values by $P_{ens} = (\mu_1(\mathbf{x}), \ldots, \mu_c(\mathbf{x}))$. Then the averaged KL divergence across the L ensemble members is

$$KL_{av} = \frac{1}{L} \sum_{i=1}^{L} KL(P_{ens} \parallel P_{(i)}). \tag{5.32}$$

We seek P_{ens} which minimizes (5.32). To simplify the notation, we shall drop the $(\mathbf{x})$ from $\mu_j(\mathbf{x})$ and $d_{i,j}(\mathbf{x})$ keeping in mind that we are operating on a specific point $\mathbf{x}$ in the feature space $\mathbb{R}^n$. Take $\partial KL_{av}/\partial \mu_j$, include the term with the Lagrange

[2]It is assumed that for any x, if $q(x) = 0$ then $p(x) = 0$, and also $0 \times \log 0 = 0$.

multiplier to ensure that P_{ens} is a pmf, and set to zero

$$\begin{aligned} & \frac{\partial}{\partial \mu_j}\left[KL_{av}+\lambda\left(1-\sum_{k=1}^{c}\mu_k\right)\right] \\ = & \frac{1}{L}\sum_{i=1}^{L}\frac{\partial}{\partial \mu_j}\left[\sum_{k=1}^{c}\mu_k \log_2\left(\frac{\mu_k}{d_{i,k}}\right)\right]-\lambda \qquad (5.33) \\ = & \frac{1}{L}\sum_{i=1}^{L}\left(\log_2\left(\frac{\mu_j}{d_{i,j}}\right)+C\right)-\lambda=0, \qquad (5.34) \end{aligned}$$

where $C=\frac{1}{\ln(2)}$ Solving for μ_j, we obtain

$$\mu_j = 2^{(\lambda-C)}\prod_{i=1}^{L}(d_{i,j})^{\frac{1}{L}}. \qquad (5.35)$$

Substituting (5.35) in $\sum_{k=1}^{c}\mu_j=1$ and solving for λ we arrive at

$$\lambda = C-\log_2\left(\sum_{k=1}^{c}\prod_{i=1}^{L}(d_{i,k})^{\frac{1}{L}}\right). \qquad (5.36)$$

Substituting λ back in (5.35) yields the final expression for the ensemble probability for class ω_j given the input $\mathbf{x}$ as the normalized geometric mean

$$\mu_j = \frac{\prod_{i=1}^{L}(d_{i,j})^{\frac{1}{L}}}{\sum_{k=1}^{c}\prod_{i=1}^{L}(d_{i,k})^{\frac{1}{L}}}. \qquad (5.37)$$

Notice that the denominator of μ_j does not depend on j. Also, the power $\frac{1}{L}$ in the numerator is only a monotone transformation of the product and will not change the ordering of the discriminant functions obtained through product. Therefore, the ensemble degree of support for class ω_j, $\mu_j(\mathbf{x})$ reduces to the *product combination rule*

$$\mu_j = \prod_{i=1}^{L} d_{i,j}. \qquad (5.38)$$

If we swap the places of the prior and posterior probabilities in (5.32) and again look for a minimum with respect to μ_j, we obtain

$$\begin{aligned} & \frac{\partial}{\partial \mu_j}\left[KL_{av}+\lambda\left(1-\sum_{k=1}^{c}\mu_k\right)\right] \\ = & \frac{1}{L}\sum_{i=1}^{L}\frac{\partial}{\partial \mu_j}\left[\sum_{k=1}^{c} d_{i,k}\log_2\left(\frac{d_{i,k}}{\mu_k}\right)\right]-\lambda \qquad (5.39) \\ = & -\frac{1}{C\,L\,\mu_j}\sum_{i=1}^{L} d_{i,j}-\lambda=0, \qquad (5.40) \end{aligned}$$

where C is again $\frac{1}{\ln(2)}$. Solving for μ_j gives

$$\mu_j = -\frac{1}{\lambda\, C\, L} \sum_{i=1}^{L} d_{i,j}. \tag{5.41}$$

Substituting (5.41) in $\sum_{k=1}^{c} \mu_k = 1$ and solving for λ leads to

$$\lambda = -\frac{1}{C\, L} \sum_{k=1}^{c} \sum_{i=1}^{L} d_{i,k} = -\frac{L}{C\, L} = -\frac{1}{C}. \tag{5.42}$$

The final expression for the ensemble probability for class ω_j, given the input $\mathbf{x}$, is the normalized arithmetic mean

$$\mu_j = \frac{1}{L} \sum_{i=1}^{L} d_{i,j}, \tag{5.43}$$

which is *average combination rule* (the same as average or sum combiner).

The average combiner was derived in the same way as the product combiner under a slightly different initial assumption. We assumed that P_{ens} is some unknown prior pmf which needs to be transformed into the L posterior pmfs suggested by the L ensemble members. Thus, to derive the average rule, we minimized the average information necessary to transform P_{ens} to the individual pmfs.

Miller and Yan go further and propose weights which depend on the 'critic' for each classifier and each $\mathbf{x}$ [281]. The 'critic' estimates the probability that the classifier is correct in labeling $\mathbf{x}$. Miller and Yen derive the product rule with the critic probability as the power of $d_{i,j}$ and the sum rule with the critic probabilities as weights. Their analysis and experimental results demonstrate the advantages of the weighted rules. The authors admit that there is no reason why one set-up should be preferred to another.

5.4 The weighted average (linear combiner)

Given an object $\mathbf{x}$, this combiner aggregates the class supports from the decision profile to arrive at a single support value for each class. Three groups of average combiners can be distinguished based on the respective number of weights:

- *L weights.* In this model there is one weight per classifier. The support for class ω_j is calculated as

$$\mu_j(\mathbf{x}) = \sum_{i=1}^{L} w_i\, d_{i,j}(\mathbf{x}). \tag{5.44}$$

- *$c \times L$ weights.* The weights are class-specific and classifier-specific. The support for class ω_j is calculated as

$$\mu_j(\mathbf{x}) = \sum_{i=1}^{L} w_{ij}\, d_{i,j}(\mathbf{x}). \tag{5.45}$$

Again, only the j-th column of the decision profile is used in the calculation, that is, the support for class ω_j is obtained from the individual supports for ω_j.

- $c \times c \times L$ *weights*. The support for each class is obtained by a linear combination of the entire decision profile $DP(\mathbf{x})$,

$$\mu_j(\mathbf{x}) = \sum_{i=1}^{L} \sum_{k=1}^{c} w_{ikj}\, d_{i,k}(\mathbf{x}), \tag{5.46}$$

where w_{ikj} is the (i,k)-th weight for class ω_j. The whole of the decision profile is used as the intermediate feature space.

We following subsections present different ways to calculate the weights.

5.4.1 Consensus theory.

The weights may be set so as to express the quality of the classifiers. Accurate and robust classifiers should receive larger weights. Such weight assignments may come from subjective estimates or theoretical set-ups.

Berenstein et al. [35] bring to the attention of the Artificial Intelligence community the so called *consensus theory* which has enjoyed a considerable interest in social and management sciences but remained not well known elsewhere. The theory looks into combining expert opinions and in particular combining L probability distributions on Ω (in our case, the rows of the decision profile $DP(\mathbf{x})$) into a single distribution $(\mu_1(\mathbf{x}), \ldots, \mu_c(\mathbf{x}))$. A *consensus rule* defines the way this combination is carried out. Consensus rules are derived to satisfy a set of desirable theoretical properties [34, 44, 289].

Based on an experimental study, Ng and Abramson [289] advocate using simple consensus rules such as the weighted average, called the *linear opinion pool* (5.44), and the weighted product called the *logarithmic opinion pool*. The approach taken to assigning weights in consensus theory is based on the decision maker's knowledge of the importance of the experts (classifiers). The weights are assigned on the basis of some subjective or objective measure of importance of the experts [35].

5.4.2 Added error for the weighted mean combination

Extending the theoretical study of Tumer and Ghosh [393], Fumera and Roli derive the added error for the weighted average combination rule [143, 145].

The ensemble estimate of $P(\omega_j|x)$ is

$$\hat{P}(\omega_j|x) = \sum_{i=1}^{L} w_i\, d_{i,j}, \quad i = 1, \ldots, c, \tag{5.47}$$

where $d_{i,j}$ is the respective entry in the decision profile and w_i are classifier-specific weights such that

$$\sum_{i=1}^{L} w_i = 1, \quad w_i \geq 0. \tag{5.48}$$

Under a fairly large list of assumptions, a set of optimal weights for independent classifiers can be calculated from the *added errors* of the individual classifiers, E^i_{add}, $m = 1, \ldots, L$. The added error is the excess above the Bayes error for the problem for the specific classifier. The weights are

$$w_i = \frac{\frac{1}{E^i_{add}}}{\sum_{k=1}^{L} \frac{1}{E^k_{add}}}, \qquad i = 1, \ldots, L. \tag{5.49}$$

Since we do not have a way to estimate the added error, we can use as a proxy the estimates of the classification errors of the base classifiers.

Despite the appealing theoretical context, this way of calculating the weights was not found to be very successful [144]. This can be due to the unrealistic and restrictive assumptions which define the optimality conditions giving rise to these weights. Fumera and Roli's experiments suggested that for large ensembles, the advantage of weighted averaging over simple averaging disappears. Besides, in weighted averaging we have to estimate the L weights, which is a potential source of error and may cancel the already small advantage.

5.4.3 Linear regression

One way to set the weights is to fit a linear regression to the posterior probabilities. Take $d_{i,j}(\mathbf{x})$, $i = 1, \ldots, L$, to be estimates of the posterior probability $P(\omega_j|\mathbf{x})$. For classification problems, the target output is given only in the form of a class label. So the target values for $P(\omega_j|\mathbf{x})$ are either 1 (in ω_j) or 0 (not in ω_j). Figure 5.13 shows the training and the operation of the regression combiner.

Classifier combination through linear regression has received significant attention. The following questions have been discussed:

- Should the weights be non-negative? If they are, the value of the weight may be interpreted as the importance of a classifier.
- Should the weights be constrained to sum up to one?
- Should there be an intercept term?

It is believed that these choices have only a marginal impact on the final outcome [193, 384]. The important question is what criterion should be optimized. Minimum Squared Error (MSE) is the traditional criterion for regression [175–177, 388]. Different criteria have been examined in the context of classifier combination through linear regression, for both small [120] and large ensembles [327], an example of which is the hinge function, which is responsible for the classification margins [120, 395].

Consider the largest regression model, where the whole decision profile is involved in approximating each posterior probability as in equation (5.46). Given a data set $Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$ with labels $\{y_1, \ldots, y_N\}$, $y_j \in \Omega$, Ergodan and Sen [120]

REGRESSION COMBINER

Training: Given is a set of L trained classifiers and a labeled data set.

1. Find the outputs (decision profiles) of the classifiers for each point in the data set.
2. For each class j, $j = 1, \ldots, c$, train a regression of the type (5.46). We can choose to fit the regression with or without an intercept term.
3. Return the coefficients of the c regressions. The coefficients of the regression for class j are denoted by w_{ikj} as in (5.46). If there was an intercept term, the number of returned coefficients for each regression is $L \times c + 1$.

Operation: For each new object

1. Classify the new object $\mathbf{x}$ and find its decision profile $DP(\mathbf{x})$ as in equation (5.1).
2. Calculate the support for each class $P(j) = \mu_j(\mathbf{x})$ as in equation (5.46).
3. Assign label i^* to the object, where

$$i^* = \arg\max_{j=1}^{c} P(j).$$

Return the ensemble label of the new object.

Figure 5.13 Training and operation algorithm for the linear regression combiner.

formulate the optimization problem as looking for a weight vector $\mathbf{w}$ which minimizes

$$\Psi(\mathbf{w}) = \underbrace{\frac{1}{N}\sum_{j=1}^{N}}_{\text{objects}} \underbrace{\sum_{i=1}^{c}}_{\text{classes}} \mathcal{L}(\mu_i(\mathbf{z}_j), y_j, \omega_i, \mathbf{w}) + R(\mathbf{w}), \tag{5.50}$$

where $\mathcal{L}(\mu_i(\mathbf{z}_j), y_j, \omega_i, \mathbf{w})$ is the loss incurred when labeling object $\mathbf{z}_j \in Z$, with true label y_j, as belonging to class ω_i. $R(\mathbf{w})$ is a regularization term which serves to penalize very large weights.[3] Why is the penalty term needed? Say there are

[3] An intercept term b can be added to the regression in equation (5.46), and included in the weight vector $\mathbf{w}$.

5 classifiers and 4 classes. For this small problem, the regression (5.46) will need $L \times c \times c = 5 \times 4 \times 4 = 80$ weights. The chance of over-training cannot be ignored, hence the need for a regularization term.

To use this optimization set-up, two choices must be made: the type of loss function $\mathcal{L}$ and the regularization function R.

Let us simplify the notation to $\mathcal{L}(a, b)$ where $a \in \{-1, 1\}$ is the true label, and b is the predicted quantity. The classification loss is $\mathcal{L}(a, b) = 0$ if the signs of a and b match and $\mathcal{L}(a, b) = 1$, otherwise. Minimizing this loss is ideal but mathematically awkward, hence Rosasco et al. [339] analyze several alternatives:

- The square loss
$$\mathcal{L}(a, b) = (a - b)^2 = (1 - ab)^2. \tag{5.51}$$
- The hinge loss
$$\mathcal{L}(a, b) = \max\{1 - ab, 0\}. \tag{5.52}$$
This is the criterion function that is minimized for training the SVM classifier.
- The logistic loss
$$\mathcal{L}(a, b) = \frac{1}{\ln 2} \ln(1 + \exp\{-ab\}). \tag{5.53}$$

Based on its theoretical properties, Ergodan and Sen [120] recommend the hinge loss function.

Reid and Grudic [327] study the effect of different regularization functions

- L_2 regularization, which, used with the square loss function (5.51), is called *ridge regression*
$$R(\mathbf{w}) = \lambda \sum_k w_k^2 = \lambda ||\mathbf{w}||_2^2. \tag{5.54}$$
- L_1 (LASSO)[4] regularization
$$R(\mathbf{w}) = \lambda \sum_k |w_k| = \lambda ||\mathbf{w}||_1. \tag{5.55}$$
- The elastic net regularization, which combines the above two. The regularization term is
$$R(\mathbf{w}) = \lambda ||\mathbf{w}||_2^2 + (1 - \lambda) ||\mathbf{w}||_1. \tag{5.56}$$

Ridge regression arrives at dense models (using all classifiers in the ensemble) whereas LASSO produces sparse ensembles. Applying the three penalty terms with the square loss for large ensembles, Reid and Grudic [327] draw the following conclusions. Ridge regression outperforms non-regularized regression, and improves on

[4]LASSO stands for Least Absolute Shrinkage and Selection Operator.

the performance of the single best classifier in the ensemble. LASSO was not as successful as the ridge regression, leading the authors to conclude that dense models were better than the sparse models.

Calculating the solution of the optimization problem with the hinge loss function is not straightforward. However, MATLAB Statistics Toolbox offers a ridge regression code, which we will use for the illustration here.

EXAMPLE 5.5 Ridge regression for posterior probabilities

We used again the letter data set from the UCI Machine Learning Repository [22]. The set consists of $N = 20000$ data points described by $n = 16$ features and labeled into the $c = 26$ classes of the letters of the Latin alphabet. Since the data set is reasonably large, we used the hold-out method for this example. The data set was randomly split into training, validation and testing parts. The training part was used to train $L = 51$ linear classifiers, the validation part, for training the ridge regression with a pre-specified value of the parameter λ, and the testing part was used to estimate the testing error of the ensemble. Each classifier was trained on a bootstrap sample from the training set. The data set was chosen on purpose. The number of classes is large, $c = 26$, which means that the dimensionality of the intermediate space is $L \times c = 51 \times 26 = 1326$. This makes classification in the intermediate space challenging, and sets the scene for demonstrating the advantages of ridge regression. Twenty six sets of coefficients were fitted on the 1,326 features, one regression for each class, and the ensemble outputs were calculated as explained in Figure 5.13.

Table 5.5 shows the ensemble error for a ridge regression on the whole decision profile (5.46), minimizing MSE with L_2 penalty term (5.54).

Table 5.5 Ensemble error for a ridge regression with parameter λ.

	training/validation/testing split in %			
λ	4/16/80	12/48/40	16/64/20	8/72/20
0.01	0.1728	0.1034	0.1012	0.0985
0.02	0.1714	0.1031	0.1007	0.0985
0.50	0.1559	0.1029	0.1012	0.0975
0.80	0.1536	0.1029	0.1014	0.0985
LDC on training+validation	0.3056	0.2913	0.2944	0.3108
Decision Tree on ensemble	0.3925	0.2993	0.2801	0.2834

Along with the ridge regression results, we show the classification error for

1. The linear discriminant classifier (LDC) trained on the training plus validation data, and tested on the testing data.[5]

[5]Function `classify` from the Statistics Toolbox of MATLAB was used for the LDC.

2. Decision tree classifier built on the validation set, using as inputs the classifier outputs. The classifiers were trained on the training data. The decision tree was tested on the classifier outputs for the testing data. Thus, the decision tree is the combiner, trained on unseen data, and tested on another unseen data set.[6]

What does the example show?

(i) *The regression combiner was invariably better than the decision tree combiner.* In all four splits of the data into training/validation/testing, the ensemble errors for the ridge regression were smaller that these for the decision tree combiner.

(ii) The regression combiner was invariably better than the individual LDC. Interestingly, the decision tree combiner failed miserably in comparison with the regression combiner for this problem, and barely managed to improve on the classification error of the individual LDC for the two larger validation sets.

(iii) *Larger validation sets led to smaller ensemble errors.* The training set was kept small on purpose. By doing so we aimed at creating an ensemble of fairly weak but diverse linear classifiers. For such an ensemble, the combiner would be important, and clear differences between the combiners could be expected.

(iv) *The penalty constant λ had a marked effect for the smallest validation set and a little effect for larger sets.* This was also to be expected, as λ is supposed to correct for the instability of the regression trained on a small sample.

This example shows that the regression combiner may work well, especially for problems with a large number of classes, and large ensemble sizes, resulting in a high-dimensional intermediate feature space. Its success will likely depend on the data set, the ensemble size, the way the individual classifiers are trained, and so on.

Regression methods are only one of many possible ways to train the combination weights. Ueda [395] uses a *probabilistic descent* method to derive the weights for combining neural networks as the base classifiers. Some authors consider using *genetic algorithms* for this task [73, 249].

5.5 A classifier as a combiner

Consider the *intermediate feature space* where each point is an expanded version of $DP(\mathbf{x})$ obtained by concatenating its L rows. Any classifier can be applied for labeling this point [189, 384, 395].

5.5.1 The supra Bayesian approach

Jacobs [193] reviews methods for combining experts' probability assessments. *Supra Bayesian methods* consider the experts' estimates as data, as many of the combiners

[6]Function `classregtree` from the Statistics Toolbox of MATLAB was used for the Decision tree classifier.

do. The problem of estimating $\mu_j(\mathbf{x})$ becomes a problem of Bayesian learning in the intermediate feature space where the decision profile $DP(\mathbf{x})$ provides the $L \times c$ features. Loosely speaking, in supra Bayesian approach for our task, we estimate the probabilities $\mu_j(\mathbf{x}) = P(\omega_j|\mathbf{x})$, $j = 1, \ldots, c$, using the L distributions provided by the ensemble members. Since these distributions are organized in a decision profile $DP(\mathbf{x})$, we have

$$\mu_j(\mathbf{x}) = P(\omega_j|\mathbf{x}) \propto p(DP(\mathbf{x})|\omega_j)P(\omega_j), \quad j = 1, \ldots, c, \tag{5.57}$$

where $p(DP(\mathbf{x})|\omega_j)$ is the class-conditional likelihood of the decision profile for the given $\mathbf{x}$ and ω_j. We assume that the only prior knowledge that we have is some estimates of the c prior probabilities $P(\omega_j)$.

When the classifier outputs are class labels, the supra Bayesian approach is the theoretical justification of the multinomial combination method, also called BKS (Chapter 4). For continuous-valued outputs, this approach, albeit theoretically well-motivated, is impractical [193]. The reason is that the pdf $p(DP(\mathbf{x})|\omega_j)$ is difficult to estimate. In principle, the supra Bayesian approach means that we use the intermediate feature space to build a classifier which is as close as possible to the Bayes classifier thereby guaranteeing the minimum possible classification error rate. Viewed in this light, all combiners that treat the classifier outputs in $DP(\mathbf{x})$ as new features are approximations within the supra Bayesian framework.

5.5.2 Decision Templates

The idea of the Decision Templates combiner (DT) is to remember the most typical decision profile for each class ω_j, called the *decision template*, DT_j, and then compare it with the current decision profile $DP(\mathbf{x})$ using some similarity measure $\mathcal{S}$. The closest match will label $\mathbf{x}$. Figures 5.14 and 5.15 describe the training and the operation of the decision templates combiner.

Two typical measures of similarity $\mathcal{S}$ are based upon

- *The squared Euclidean distance.* The ensemble support for ω_j is

$$\mu_j(\mathbf{x}) = 1 - \frac{1}{L \times c} \sum_{i=1}^{L} \sum_{k=1}^{c} \left(DT_j(i,k) - d_{i,k}(\mathbf{x})\right)^2, \tag{5.58}$$

  where $DT_j(i,k)$ is the (i,k)-th entry in decision template DT_j. The outputs μ_j are within the interval [0,1] but this scaling is not necessary for classification purposes. The class with the maximum support would be the same if we use just

$$\mu_j(\mathbf{x}) = -\sum_{i=1}^{L} \sum_{k=1}^{c} \left(DT_j(i,k) - d_{i,k}(\mathbf{x})\right)^2. \tag{5.59}$$

  This calculation is equivalent to applying the nearest mean classifier in the intermediate feature space. While we use only the Euclidean distance in (5.58),

DECISION TEMPLATES COMBINER

Training: For $j = 1, \ldots, c$, calculate the mean of the decision profiles of all members of ω_j from the data set $\mathbf{Z}$. Call this mean *decision template* DT_j

$$DT_j = \frac{1}{N_j} \sum_{\substack{y_k = \omega_j \\ \mathbf{z}_k \in \mathbf{Z}}} DP(\mathbf{z}_k),$$

where N_j is the number of elements of $\mathbf{Z}$ from ω_j.

Operation: Given the input $\mathbf{x} \in \mathbb{R}^n$, construct $DP(\mathbf{x})$. Calculate the similarity $\mathcal{S}$ between $DP(\mathbf{x})$ and each DT_j,

$$\mu_j(\mathbf{x}) = \mathcal{S}(DP(\mathbf{x}), DT_j) \quad j = 1, \ldots, c$$

and label $\mathbf{x}$ to the class with the largest support.

Return the ensemble label of the new object.

Figure 5.14 Training and operation algorithm for the Decision Templates combiner.

there is no reason to stop at this choice. Any distance could be used, for example the Minkowski or the Mahalanobis distances.

- *A symmetric difference* coming from the fuzzy set theory [222, 233]. The support for ω_j is

$$\begin{aligned} \mu_j(\mathbf{x}) &= 1 - \frac{1}{L \times c} \sum_{i=1}^{L} \sum_{k=1}^{c} \max\{\min\{DT_j(i,k), (1 - d_{i,k}(\mathbf{x}))\}, \\ &\quad \min\{(1 - DT_j(i,k)), d_{i,k}(\mathbf{x})\}\}. \end{aligned} \tag{5.60}$$

EXAMPLE 5.6 Decision templates combiner (DT).

Let $c = 3$, $L = 2$, and let the decision templates for ω_1 and ω_2 be respectively

$$DT_1 = \begin{bmatrix} 0.6 & 0.4 \\ 0.8 & 0.2 \\ 0.5 & 0.5 \end{bmatrix} \quad \text{and} \quad DT_2 = \begin{bmatrix} 0.3 & 0.7 \\ 0.4 & 0.6 \\ 0.1 & 0.9 \end{bmatrix}.$$

Assume that for an input $\mathbf{x}$, the following decision profile has been obtained:

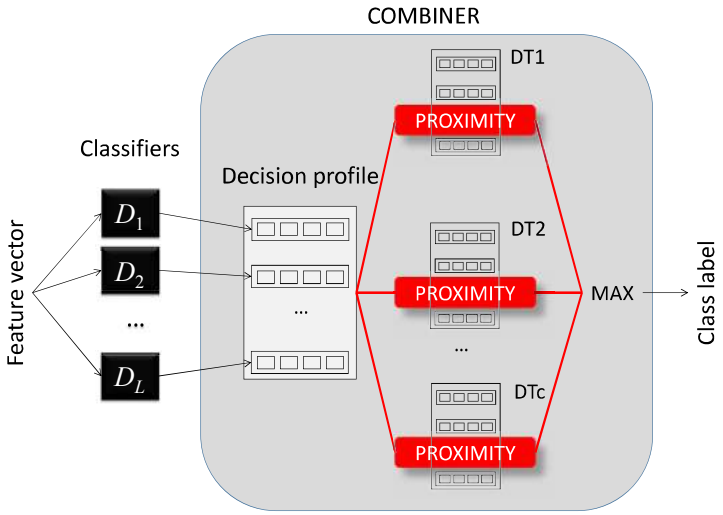


Figure 5.15 Operation of the decision templates combiner.

$$DP(\mathbf{x}) = \begin{bmatrix} 0.3 & 0.7 \\ 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix}.$$

The similarities and the class labels using the Euclidean distance and the symmetric difference are as follows

DT version	$\mu_1(\mathbf{x})$	$\mu_2(\mathbf{x})$	Label
Euclidean distance	0.9567	0.9333	ω_1
Symmetric difference	0.5000	0.5333	ω_2

The difference in the 'opinions' of the two DT versions with respect to the class label is an indication of the flexibility of the combiner.

5.5.3 A linear classifier

The Linear Discriminant Classifier (LDC) seems a good choice for determining the weights of the linear combiner [324,325,434]. It has an advantage over the regression method because it minimizes a function directly related to the classification error while regression methods optimize posterior probability approximations. Better still, we can use the SVM classifier with the linear kernel, which is capable of dealing with

correlated inputs (the classifier outputs) and small training sets [161]. In fact, any classifier can be applied as the combiner, which brings back the rather philosophical issue raised by Tin Ho [183]: Where do we stop growing the hierarchy of classifiers upon classifiers? Do we even have to?

5.6 An example of nine combiners for continuous-valued outputs

Consider again the fish data set, generated with 20% label noise. Seventeen random linear classifiers were generated as the base ensemble classifiers. Their classification boundaries are plotted with lines in each data scatterplot in Figure 5.16.

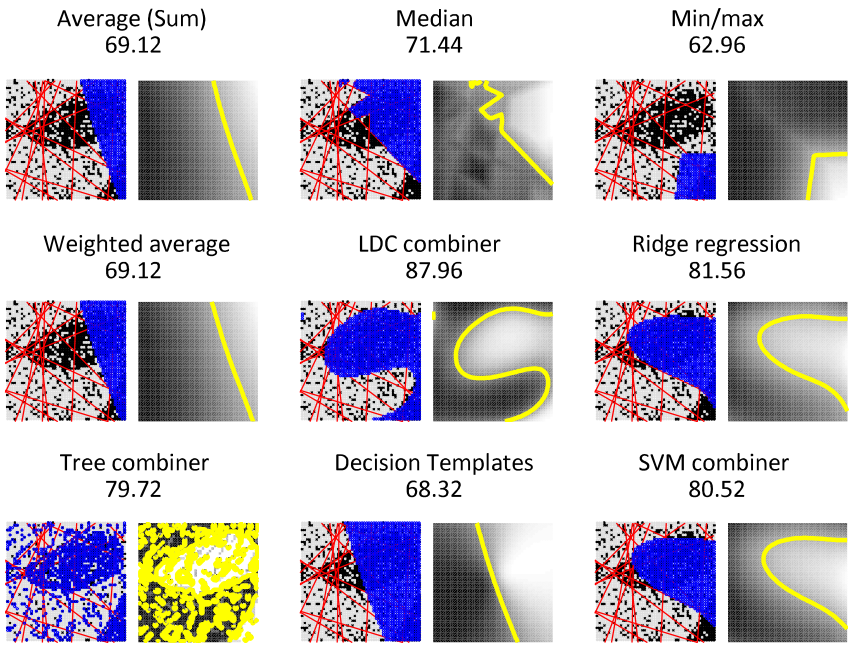


Figure 5.16 Comparison of 9 combiners on the fish data.

The continuous-valued outputs (posterior probability estimates) were obtained using the MATLAB function `classify`. Each of the nine combiners gives rise to two plots. The left plot contains the grid with the noisy fish data. The region labeled as the fish (black dots) by the ensemble is overlaid. The accuracy displayed under the combiner's name is calculated with respect to the original (noise-free) class labels. The right plot is a gray-scale heat map of the ensemble estimate of $P(\text{fish}|\mathbf{x})$. The contour for $P(\text{fish}|\mathbf{x}) = 0.5$, delineating the classification region for class fish, is plotted with a thick line over the heat map.

In this example, the LDC, the Ridge regression ($\lambda = 0.5$, not tuned) and the SVM combiner were the winners, with above 80% correct classification rate, given that

the largest prior classifier would give only 64.48%. The worst combiner happened to be the minimum combiner (equal to the maximum combiner for two classes). The average, weighted average and decision templates combiners were obviously too simplistic for the problem, and gave disappointingly low ensemble accuracies. On the other hand, the 'peppery' right plot for the decision tree combiner demonstrates a great deal of over-training. Nonetheless, this combiner achieved over 79% correct classification, which indicates that, for this problem, even though both alternatives are wrong, over-fitting gives a better pay-off than under-fitting.

As noted before, this example should not be taken to mean that LDC is always the best combiner, and minimum/maximum is the worst. The message is that the choice of a combiner is important, and should not be casually sidelined.

5.7 To train or not to train?

Some combiners do not need training after the classifiers in the ensemble have been trained individually. An example of this type is the majority vote combiner. Other combiners need additional training, for example, the weighted average combiner. A third class of ensembles develop the combiner during the training of the individual classifiers, an example of which is AdaBoost, discussed later. If a large data set is available, training and testing can be done on large, non-intersecting subsets, which allows for precise tuning while guarding against over-fitting. Small data sets, on the other hand, pose a real challenge. Duin [108] points out the crucial role of the training strategy in these cases and gives the following recommendations:

1. If a single training set is used with a *non-trainable combiner*, then make sure that the base classifiers are not overtrained.

2. If a single training set is used with a *trainable combiner*, then leave the base classifiers undertrained and subsequently complete the training of the combiner on the training set. Here it is assumed that the training set has a certain 'training potential'. In order to be able to be train the combiner reasonably, the base classifiers should not use up all the potential.

3. Use separate training sets for the base classifiers and for the combiners. Then the base classifiers can be overtrained on their training set. The bias will be corrected by training the combiner on the separate training set.

Dietrich et al. [93] suggest that the second training set, on which the ensemble should be trained, may be partly overlapping with the first training set used for the individual classifiers. Let R be the first training set, V be the second training set, and T be the testing set. All three sets are obtained from the available labeled set $\mathbf{Z}$, so $R \cup V \cup T = \mathbf{Z}$. If $\mathbf{Z}$ is small, the three sets might become inadequately small thereby leading to badly trained classifiers and ensemble, and unreliable estimates of their accuracies. To remedy this, the two training sets are allowed to have an overlap

controlled by a parameter ρ

$$\rho = \frac{|R \cap V|}{|R|}, \tag{5.61}$$

where $|.|$ denotes cardinality. For $\rho = 0$, R and V are disjoined and for $\rho = 1$, the classifiers and the ensemble are trained on a single set $R = V$. The authors found that better results were obtained for $\rho = 0.5$ compared to the two extreme values. This suggests that a compromise should be sought when the initial data set $\mathbf{Z}$ is relatively small.

Stacked generalization has been defined as a generic methodology for improving generalization in pattern classification [420]. We will present it here through an example, as a protocol for training a classifier ensemble and its combiner.

EXAMPLE 5.7 Stacked generalization

Let $\mathbf{Z}$ be a data set with N objects partitioned into 4 parts of approximately equal sizes, denoted A, B, C and D. Three classifiers, D_1, D_2 and D_3, are trained according to the standard 4-fold cross-validation protocol depicted in Figure 5.17. At the end of this training, there will be four versions of each of the classifiers trained on (ABC), (BCD), (ACD), or (ABD), respectively.

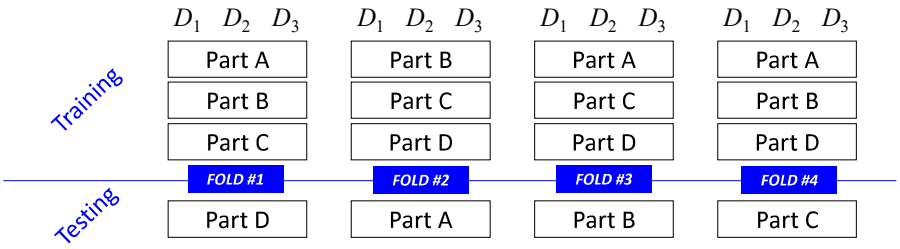


Figure 5.17 Standard 4-fold cross-validation set-up.

The combiner is trained on a data set of size N obtained in the following way. For any data point $\mathbf{z}_j$ in subset A, we take the outputs for that point from the versions of D_1, D_2 and D_3 built on (BCD). In this way subset A has not been seen during the training of the individual classifiers. The three outputs together with the label of $\mathbf{z}_j$ form a data point in the training set for the combiner. All the points from subset B are processed by the versions of the three classifiers built on (ACD) and the outputs added to the training set for the combiner, etc. After the combiner has been trained, the four subsets are pooled again into $\mathbf{Z}$ and D_1, D_2 and D_3 are re-trained, this time on the whole of $\mathbf{Z}$. The new classifiers and the combiner are then ready for operation.

Many authors have studied and compared the performance of ensemble combiners [9, 109, 206, 321, 325, 337, 382, 383, 399, 400, 434]. Most such studies, both em-

pirical and theoretical, do not elect a clear winner. This is to be expected in view of the "no panacea theorem" [187]. The value of such comparative studies is to accumulate knowledge and understanding of the conditions which could guide the choice of a combiner. These conditions may be the type of data or the problem, as well as the ensemble size, homogeneity, diversity and building strategy.

Appendix

A.1 Theoretical classification error for the simple combiners

A.1.1 Set-up and assumptions

We reproduce the scenario and the assumption from the text.

- There are only two classes, $\Omega = \{\omega_1, \omega_2\}$.
- All classifiers produce soft class labels, $d_{j,i}(\mathbf{x}) \in [0,1]$, $i = 1,2$, $j = 1,\ldots,L$, where $d_{j,i}(\mathbf{x})$ is an estimate of the posterior probability $P(\omega_i|\mathbf{x})$ by classifier D_j for an input $\mathbf{x} \in \mathbb{R}^n$. We consider the case where for any $\mathbf{x}$, $d_{j,1}(\mathbf{x}) + d_{j,2}(\mathbf{x}) = 1$, $j = 1,\ldots,L$.
- Let $\mathbf{x} \in \mathbb{R}^n$ be a data point to classify. Without loss of generality, we assume that the true posterior probability is $P(\omega_1|\mathbf{x}) = p > 0.5$. Thus, the Bayes-optimal class label for $\mathbf{x}$ is ω_1, and a classification error occurs if label ω_2 is assigned.

Assumption. The classifiers commit independent and identically distributed errors in estimating $P(\omega_1|\mathbf{x})$ such that

$$d_{j,1}(\mathbf{x}) = P(\omega_1|\mathbf{x}) + \eta(\mathbf{x}) = p + \eta(\mathbf{x}), \tag{A.1}$$

and respectively $d_{j,2}(\mathbf{x}) = 1 - p - \eta(\mathbf{x})$, where $\eta(\mathbf{x})$ has

(i) a normal distribution with mean 0 and variance σ^2 (we take σ to vary between 0.1 and 1)

(ii) a uniform distribution spanning the interval $[-b, +b]$ (b varies from 0.1 to 1).

We derive the theoretical error rate of an ensemble of L classifiers for a given object $\mathbf{x}$ and the following combiners: majority vote, average (sum), minimum, maximum, and median. For comparison, we include in the list the individual classifier error rate and the so called 'oracle' combiner which outputs the correct class label if at least one of the classifiers produces the correct class label.

Recall that, for the majority vote, we first 'harden' the individual decisions by assigning class label ω_1 if $d_{j,1}(\mathbf{x}) > 0.5$, and ω_2 if $d_{j,1}(\mathbf{x}) \leq 0.5$, $j = 1,\ldots,L$. Then the class label most represented among the L (label) outputs is chosen as the final label for $\mathbf{x}$.

Denote by P_j the output of classifier D_j for class ω_1, that is, $P_j = d_{j,1}(\mathbf{x})$, and let

$$\hat{P}_1 = \mathcal{F}(P_1, \ldots, P_L) \tag{A.2}$$

be the fused estimate of $P(\omega_1|\mathbf{x})$. By assumption, the posterior probability estimates for ω_2 are $1 - P_j$, $j = 1, \ldots, L$. The same fusion method $\mathcal{F}$ is used to find the fused estimate of $P(\omega_2|\mathbf{x})$,

$$\hat{P}_2 = \mathcal{F}(1 - P_1, \ldots, 1 - P_L) \tag{A.3}$$

According to the assumptions, we regard the individual estimates P_j as independent, identically distributed random variables, such that $P_j = p + \eta_j$, with probability density functions (pdf) $f(y), y \in \mathbb{R}$ and cumulative distribution functions (cdf) $F(t), t \in \mathbb{R}$. Then $\hat{P}_1$ is a random variable with a pdf $f_{\hat{P}_1}(y)$ and cdf $F_{\hat{P}_1}(t)$.

For a single classifier, the average and the median fusion models will result in $\hat{P}_1 + \hat{P}_2 = 1$. The higher of the two estimates determines the class label. The oracle and the majority vote make decisions on the class label outputs, so $\hat{P}_1 = 1, \hat{P}_2 = 0$ for class ω_1, and $\hat{P}_1 = 0, \hat{P}_2 = 1$ for class ω_2. Thus, it is necessary and sufficient to have $\hat{P}_1 > 0.5$ to label $\mathbf{x}$ in ω_1 (the correct label). The probability of error, given $\mathbf{x}$, denoted P_e, is

$$P_e = P(\text{error}|\mathbf{x}) = P(\hat{P}_1 \leq 0.5) = F_{\hat{P}_1}(0.5) = \int_0^{0.5} f_{\hat{P}_1}(y)dy \tag{A.4}$$

for the single best classifier, average, median, majority vote and the oracle.

For the minimum and the maximum rules, however, the sum of the fused estimates is not necessarily one. The class label is then decided by the maximum of $\hat{P}_1$ and $\hat{P}_2$. Thus, an error will occur if $\hat{P}_1 \leq \hat{P}_2$,[7]

$$P_e = P(\text{error}|\mathbf{x}) = P(\hat{P}_1 \leq \hat{P}_2) \tag{A.5}$$

for the minimum and the maximum.

The two distributions considered are

- Normal distribution, $\hat{P}_1 \sim N(p, \sigma^2)$. We denote by $\Phi(z)$ the cumulative distribution function of $N(0, 1)$. Then

$$F(t) = \Phi\left(\frac{t - p}{\sigma}\right). \tag{A.6}$$

- Uniform distribution within $[p - b, p + b]$, that is,

$$f(y) = \begin{cases} \frac{1}{2b}, & y \in [p - b, p + b]; \\ 0, & \text{elsewhere}, \end{cases} \qquad F(t) = \begin{cases} 0, & t \in (-\infty, p - b); \\ \frac{t - p + b}{2b}, & t \in [p - b, p + b]; \\ 1, & t > p + b. \end{cases} \tag{A.7}$$

[7]We note that since $\hat{P}_1$ and $\hat{P}_2$ are continuous-valued random variables, the inequalities can be written with or without the equal sign, that is, $\hat{P}_1 > 0.5$ is equivalent to $\hat{P}_1 \geq 0.5$, and so on.

Clearly, using these two distributions, the estimates of the probabilities might fall outside the interval [0,1]. We can accept this, and justify our viewpoint by the following argument. Suppose that p is not a probability but *the amount of support* for ω_1. The support for ω_2 will be again $1-p$. In estimating p, we do not have to restrict P_js within the interval $[0,1]$. For example, a neural network (or *any* classifier for that matter) trained by minimizing the squared error between its output and the zero-one (class label) target function produces an estimate of the posterior probability for that class (cf. [40]). Thus, depending on the parameters and the transition functions, a neural network output (that approximates p) might be greater than 1 or even negative. We take the L values (in $\mathbb{R}$) and fuse them by (A.2) and (A.3) to get $\hat{P}_1$ and $\hat{P}_2$. The same rule applies: ω_1 is assigned by the ensemble if $\hat{P}_1 > \hat{P}_2$. Then we calculate the *probability* of error P_e as $P(\hat{P}_1 \leq \hat{P}_2)$. This calculation does not require in any way that P_js are probabilities or are within the unit interval.

A.1.2 Individual error

Since $F_{\hat{P}_1}(t) = F(t)$, the error of a single classifier for the normal distribution is

$$P_e = \Phi\left(\frac{0.5-p}{\sigma}\right), \tag{A.8}$$

and for the uniform distribution,

$$P_e = \frac{0.5-p+b}{2b}. \tag{A.9}$$

A.1.3 Minimum and maximum

These two fusion methods are considered together because, as shown in the text, they are identical for $c = 2$ classes and any number of classifiers L.

Substituting $\mathcal{F} = \max$ in (A.2), the ensemble's support for ω_1 is $\hat{P}_1 = \max_j\{P_j\}$. The support for ω_2 is therefore $\hat{P}_2 = \max_j\{1-P_j\}$. A classification error will occur if

$$\max_j\{P_j\} < \max_j\{1-P_j\}, \tag{A.10}$$

$$p + \max_j\{\eta_j\} < 1 - p - \min_j\{\eta_j\}, \tag{A.11}$$

$$\eta_{\max} + \eta_{\min} < 1 - 2p. \tag{A.12}$$

The probability of error for the minimum and maximum methods is

$$P_e = P\left(\eta_{\max} + \eta_{\min} < 1 - 2p\right) \tag{A.13}$$

$$= F_{\eta_s}(1-2p), \tag{A.14}$$

where $F_{\eta_s}(t)$ is the cdf of the random variable $s = \eta_{\max} + \eta_{\min}$. For the normally distributed P_js, η_j are also normally distributed with mean 0 and variance σ^2.

However, we cannot assume that $\eta_{\max}$ and $\eta_{\min}$ are independent and analyze their sum as another normally distributed variable because these are *order statistics* and $\eta_{\min} \leq \eta_{\max}$. We have not attempted a solution for the normal distribution case.

For the uniform distribution, we follow an example taken from [285] where the pdf of the midrange $(\eta_{\min} + \eta_{\max})/2$ is calculated for L observations. We derive $F_{\eta_s}(t)$ to be

$$F_{\eta_s}(t) = \begin{cases} \frac{1}{2}\left(\frac{t}{2b}+1\right)^L, & t \in [-2b, 0]; \\ 1 - \frac{1}{2}\left(1-\frac{t}{2b}\right)^L, & t \in [0, 2b]. \end{cases} \tag{A.15}$$

Noting that $t = 1 - 2p$ is always negative,

$$P_e = F_{\eta_s}(1-2p) = \frac{1}{2}\left(\frac{1-2p}{2b}+1\right)^L. \tag{A.16}$$

A.1.4 Average (Sum)

The average combiner gives $\hat{P}_1 = \frac{1}{L}\sum_{j=1}^{L} P_j$. If $P_1, \ldots, P_L$ are normally distributed and independent, then $\hat{P} \sim N\left(p, \frac{\sigma^2}{L}\right)$. The probability of error for this case is

$$P_e = P(\hat{P}_1 < 0.5) = \Phi\left(\frac{\sqrt{L}(0.5-p)}{\sigma}\right). \tag{A.17}$$

The calculation of P_e for the case of uniform distribution is not that straightforward. We can assume that the sum of L independent variable will results in a variable of approximately normal distribution. The higher the L, the more accurate the approximation. Knowing that the variance of the uniform distribution for P_j is $\frac{b^2}{3}$, we can assume $\hat{P} \sim N\left(p, \frac{b^2}{3L}\right)$. Then

$$P_e = P(\hat{P}_1 < 0.5) = \Phi\left(\frac{\sqrt{3L}(0.5-p)}{b}\right). \tag{A.18}$$

A.1.5 Median and majority vote

These two fusion methods are pooled because they are identical for the current set-up (see the text). Since only two classes are considered, we restrict our choice of L to odd numbers only. An even L is inconvenient for at least two reasons. First, the majority vote might tie. Second, the theoretical analysis of a median which is calculated as the average of the $(L/2)$ and $(L/2+1)$ order statistics is cumbersome.

For the median fusion method

$$\hat{P}_1 = \text{med}\{P_1, \ldots, P_L\} = p + \text{med}\{\eta_1, \ldots, \eta_L\} = p + \eta_m. \tag{A.19}$$

Then the probability of error is

$$P_e = P(p + \eta_m < 0.5) = P(\eta_m < 0.5 - p) = F_{\eta_m}(0.5-p), \tag{A.20}$$

where F_{η_m} is the cdf of η_m. From the order statistics theory [285],

$$F_{\eta_m}(t) = \sum_{j=\frac{L+1}{2}}^{L} \binom{L}{j} F_\eta(t)^j [1 - F_\eta(t)]^{L-j}, \tag{A.21}$$

where $F_\eta(t)$ is the cdf of η_j, that is, $N(0, \sigma^2)$ or uniform in $[-b, b]$. We can now substitute the two respective cdfs, to obtain P_e

- for the normal distribution

$$P_e = \sum_{j=\frac{L+1}{2}}^{L} \binom{L}{j} \Phi\left(\frac{0.5-p}{\sigma}\right)^j \left[1 - \Phi\left(\frac{0.5-p}{\sigma}\right)\right]^{L-j}. \tag{A.22}$$

- for the uniform distribution

$$P_e = \begin{cases} 0, & p - b > 0.5; \\ \sum_{j=\frac{L+1}{2}}^{L} \binom{L}{j} \left(\frac{0.5-p+b}{2b}\right)^j \left[1 - \frac{0.5-p+b}{2b}\right]^{L-j}, & \text{otherwise.} \end{cases} \tag{A.23}$$

The derivation of these two equations is explained below. The majority vote will assign the wrong class label, ω_2, to $\mathbf{x}$ if at least $\frac{L+1}{2}$ classifiers vote for ω_2. The probability that a single classifier is wrong is given by (A.8) for the normal distribution and (A.9) for the uniform distribution. Denote this probability by P_s. Since the classifiers are independent, the probability that at least $\frac{L+1}{2}$ are wrong is calculated by the binomial formula

$$P_e = \sum_{j=\frac{L+1}{2}}^{L} \binom{L}{j} P_s^j (1 - P_s)^{L-j}. \tag{A.24}$$

By substituting P_s from (A.8) and (A.9), we recover equations (A.22) and (A.23) for the normal and the uniform distribution, respectively.

A.1.6 Oracle

The probability of error for the oracle is

$$P_e = P(\text{all incorrect}) = F(0.5)^L. \tag{A.25}$$

For the normal distribution

$$P_e = \Phi\left(\frac{0.5-p}{\sigma}\right)^L, \tag{A.26}$$

and for the uniform distribution

$$P_e = \begin{cases} 0, & p - b > 0.5; \\ \left(\frac{0.5-p+b}{2b}\right)^L, & \text{otherwise.} \end{cases} \tag{A.27}$$

A.2 Selected MATLAB code

Example of the LDC combiner for the fish data

The code below generates and plots the data and the 50 linear classification boundaries of the random base classifiers. The LDC combiner is trained on the training data, which consists of all points on the grid, with 20% label noise. The posterior probabilities for class ω_1 are calculated by line 35, using the softmax formula. The points labeled by the ensemble as class fish (black dots) are circled. The code needs the function `fish_data` from Chapter 2, and the statistics toolbox of MATLAB for the `classify` function. An example of the output is shown in Figure 5-A.1.

```
%--------------------------------------------------------------------%
clear all, close all
clc

% Generate and plot the data
[~,~,labtrue] = fish_data(50,0);
% Generate labels with 20% noise
[x,y,lb] = fish_data(50,20); figure, hold on
plot(x(lb == 1),y(lb == 1),'k.','markers',14)
plot(x(lb == 2),y(lb == 2),'k.','markers',14,...
    'color',[0.87, 0.87, 0.87])
axis([0 1 0 1]) % cut the figure to the unit square
axis square off % equalize and remove the axes

% Generate and plot the ensemble of linear classifiers
L = 50; % ensemble size
N = numel(x); % number of data points
[ensemble,P1] = deal(zeros(N,L)); % pre-allocate for speed
sc = 1; % scaling constant for the softmax function
for i = 1:L
    p = rand(1,2); % random point in the unit square
    w = randn(1,2); % random normal vector to the line
    w0 = p * w'; % the free term (neg)
    plot([0 1],[w0, (w0-w(1))]/w(2),'r-',...
        'linewidth',1.4) % plot the linear boundary
    plot(p(1),p(2),'r.','markersize',15)
    pause(0.03)
    t = 2  - ([x y] * w' - w0 > 0);
    if mean(t == lb) < 0.5, t = 3-t; end % revert labels

    % Posteriors
    ou = [x y] * w' - w0;
    % Store the estimates of the probability for class 1
    P1(:,i) = 1./(1 + exp(-ou * sc)); % softmax

end

% Find and plot the LDC combiner output
assigned_labels = classify(P1,P1,lb);
% (train with the noisy labels)
accuracy_LDC = mean(assigned_labels == labtrue);
```

```
plot(x(assigned_labels==1),y(assigned_labels==1),...
    'bo','linewidth',1.5)
title(['LDC combiner accuracy ',num2str(accuracy_LDC)])
%--------------------------------------------------------------------------%
```

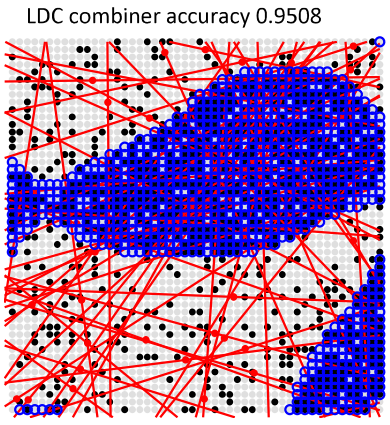


Figure 5-A.1 MATLAB output for the LDC combiner and the fish data.