Table 7.2 Overview of different types of SVs. In each case, the condition on the Lagrange multipliers α_i (corresponding to an SV x_i) is given. In the table, $\alpha_{\max}$ stands for the upper bound in the optimization problem; for instance, $\alpha_{\max} = \frac{C}{m}$ in (7.38) and $\alpha_{\max} = \frac{1}{m}$ in (7.50).

Type of SV	Definition	Properties
(standard) SV	$0 < \alpha_i$	lies on or in margin
in-bound SV	$0 < \alpha_i < \alpha_{\max}$	lies on margin
bound SV	$\alpha_i = \alpha_{\max}$	usually lies in margin ("margin error")
essential SV	appears in all possible expansions of solution	becomes margin error when left out (Section 7.3)

7.6 Multi-Class Classification

So far, we have talked about binary classification, where the class labels can only take two values: ± 1. Many real-world problems, however, have more than two classes — an example being the widely studied optical character recognition (OCR) problem. We will now review some methods for dealing with this issue.

7.6.1 One Versus the Rest

To get *M-class classifiers*, it is common to construct a set of binary classifiers $f^1, \ldots, f^M$, each trained to separate one class from the rest, and combine them by doing the multi-class classification according to the maximal output before applying the sgn function; that is, by taking

$$\operatorname*{argmax}_{j=1,\ldots,M} g^j(x), \quad \text{where } g^j(x) = \sum_{i=1}^{m} y_i \alpha_i^j k(x, x_i) + b^j \tag{7.58}$$

(note that $f^j(x) = \operatorname{sgn}(g^j(x))$, cf. (7.25)).

Reject Decisions

The values $g^j(x)$ can also be used for *reject decisions*. To see this, we consider the difference between the two largest $g^j(x)$ as a measure of confidence in the classification of x. If that measure falls short of a threshold θ, the classifier rejects the pattern and does not assign it to a class (it might instead be passed on to a human expert). This has the consequence that on the remaining patterns, a lower error rate can be achieved. Some benchmark comparisons report a quantity referred to as the *punt error*, which denotes the fraction of test patterns that must be rejected in order to achieve a certain accuracy (say 1% error) on the remaining test samples. To compute it, the value of θ is adjusted on the *test* set [64].

The main shortcoming of (7.58), sometimes called the *winner-takes-all* approach, is that it is somewhat heuristic. The binary classifiers used are obtained by training on different binary classification problems, and thus it is unclear whether their

real-valued outputs (before thresholding) are on comparable scales.[13] This can be a problem, since situations often arise where *several* binary classifiers assign the pattern to their respective class (or where *none* does); in this case, *one* class must be chosen by comparing the real-valued outputs.

In addition, binary one-versus-the-rest classifiers have been criticized for dealing with rather asymmetric problems. For instance, in digit recognition, the classifier trained to recognize class '7' is usually trained on many more negative than positive examples. We can deal with these asymmetries by using values of the regularization constant C which differ for the respective classes (see Problem 7.10). It has nonetheless been argued that the following approach, which is more symmetric from the outset, can be advantageous.

7.6.2 Pairwise Classification

In pairwise classification, we train a classifier for each possible pair of classes [178, 463, 233, 311]. For M classes, this results in $(M-1)M/2$ binary classifiers. This number is usually larger than the number of one-versus-the-rest classifiers; for instance, if $M = 10$, we need to train 45 binary classifiers rather than 10 as in the method above. Although this suggests large training times, the individual problems that we need to train on are significantly smaller, and if the training algorithm scales superlinearly with the training set size, it is actually possible to save time.

Similar considerations apply to the runtime execution speed. When we try to classify a test pattern, we evaluate all 45 binary classifiers, and classify according to which of the classes gets the highest number of votes. A vote for a given class is defined as a classifier putting the pattern into that class.[14] The individual classifiers, however, are usually smaller in size (they have fewer SVs) than they would be in the one-versus-the-rest approach. This is for two reasons: First, the training sets are smaller, and second, the problems to be learned are usually easier, since the classes have less overlap.

Nevertheless, if M is large, and we evaluate the $(M-1)M/2$ classifiers, then the resulting system may be slower than the corresponding one-versus-the-rest SVM. To illustrate this weakness, consider the following hypothetical situation: Suppose, in a digit recognition task, that after evaluating the first few binary classifiers, both digit 7 and digit 8 seem extremely unlikely (they already "lost" on several classifiers). In that case, it would seem pointless to evaluate the 7-vs-8 classifier. This idea can be cast into a precise framework by embedding the binary classifiers into a directed acyclic graph. Each classification run then corresponds to a directed traversal of that graph, and classification can be much faster [411].

---

13. Note, however, that some effort has gone into developing methods for transforming the real-valued outputs into class probabilities [521, 486, 410].
14. Some care has to be exercised in tie-breaking. For further detail, see [311].

7.6.3 Error-Correcting Output Coding

The method of error-correcting output codes was developed in [142], and later adapted to the case of SVMs [5]. In a nutshell, the idea is as follows. Just as we can generate a binary problem from a multiclass problem by separating one class from the rest — digit 0 from digits 1 through 9, say — we can generate a large number of further binary problems by splitting the original set of classes into two subsets. For instance, we could separate the even digits from the odd ones, or we could separate digits 0 through 4 from 5 through 9. It is clear that if we design a set of binary classifiers $f^1, \ldots, f^L$ in the right way, then the binary responses will completely determine the class of a test patterns. Each class corresponds to a unique vector in $\{\pm 1\}^L$; for M classes, we thus get a so-called *decoding matrix* $M \in \{\pm 1\}^{M \times L}$. What happens if the binary responses are inconsistent with each other; if, for instance, the problem is noisy, or the training sample is too small to estimate the binary classifiers reliably? Formally, this means that we will obtain a vector of responses $f^1(x), \ldots, f^L(x)$ which does not occur in the matrix M. To deal with these cases, [142] proposed designing a clever set of binary problems, which yields robustness against some errors. Here, the closest match between the vector of responses and the rows of the matrix is determined using the Hamming distance (the number of entries where the two vectors differ; essentially, the L_∞ distance). Now imagine a situation where the code is such that the minimal Hamming distance is three. In this case, we can *guarantee* that we will correctly classify all test examples which lead to at most one error amongst the binary classifiers.

This method produces very good results in multi-class tasks; nevertheless, it has been pointed out that it does not make use of a crucial quantity in classifiers: the margin. Recently [5], a version was developed that replaces the Hamming-based decoding with a more sophisticated scheme that takes margins into account. Recommendations are also made regarding how to design good codes for margin classifiers, such as SVMs.

7.6.4 Multi-Class Objective Functions

Arguably the most elegant multi-class algorithm, and certainly the method most closely aligned with Vapnik's principle of always trying to solve problems *directly*, entails modifying the SVM objective function in such a way that it simultaneously allows the computation of a multi-class classifier. For instance [593, 58], we can modify (7.35) and use the following quadratic program:

$$\underset{\mathbf{w}_r \in \mathcal{H}, \boldsymbol{\xi}^r \in \mathbb{R}^m, b_r \in \mathbb{R}}{\text{minimize}} \ \frac{1}{2} \sum_{r=1}^{M} \|\mathbf{w}_r\|^2 + \frac{C}{m} \sum_{i=1}^{m} \sum_{r \neq y_i} \xi_i^r, \tag{7.59}$$

$$\text{subject to } \langle \mathbf{w}_{y_i}, \mathbf{x}_i \rangle + b_{y_i} \geq \langle \mathbf{w}_r, \mathbf{x}_i \rangle + b_r + 2 - \xi_i^r,$$

$$\xi_i^r \geq 0, \tag{7.60}$$

where $m \in \{1, \ldots, M\} \setminus y_i$, and $y_i \in \{1, \ldots, M\}$ is the multi-class label of the pattern $\mathbf{x}_i$ (cf. Problem 7.17).

In terms of accuracy, the results obtained with this approach are comparable to those obtained with the widely used one-versus-the-rest approach. Unfortunately, the optimization problem is such that it has to deal with *all* SVs at the same time. In the other approaches, the individual binary classifiers usually have much smaller SV sets, with beneficial effects on the training time. For further multiclass approaches, see [160, 323]. Generalizations to *multi-label* problems, where patterns are allowed to belong to several classes at the same time, are discussed in [162].

Overall, it is fair to say that there is probably no multi-class approach that generally outperforms the others. For practical problems, the choice of approach will depend on constraints at hand. Relevant factors include the required accuracy, the time available for development and training, and the nature of the classification problem (e.g., for a problem with very many classes, it would not be wise to use (7.59)). That said, a simple one-against-the-rest approach often produces acceptable results.

7.7 Variations on a Theme

There are a number of variations of the standard SV classification algorithm, such as the elegant *leave-one-out machine* [589, 592] (see also Section 12.2.2 below), the idea of *Bayes point machines* [451, 239, 453, 545, 392], and extensions to *feature selection* [70, 224, 590]. Due to lack of space, we only describe one of the variations; namely, *linear programming machines*.

Linear Programming Machines

As we have seen above, the SVM approach automatically leads to a decision function of the form (7.25). Let us rewrite it as $f(x) = \operatorname{sgn}(g(x))$, with

$$g(x) = \sum_{i=1}^{m} v_i k(x, x_i) + b. \tag{7.61}$$

In Chapter 4, we showed that this form of the solution is essentially a consequence of the form of the regularizer $\|\mathbf{w}\|^2$ (Theorem 4.2). The idea of linear programming (LP) machines is to use the kernel expansion as an ansatz for the solution, but to use a different regularizer, namely the ℓ_1 norm of the coefficient vector [343, 344, 74, 184, 352, 37, 591, 593, 39]. The main motivation for this is that this regularizer is known to induce sparse expansions (see Chapter 4).

ℓ_1 Regularizer

This amounts to the objective function

$$R_{\text{reg}}[g] := \frac{1}{m}\|\boldsymbol{v}\|_1 + C\, R_{\text{emp}}[g], \tag{7.62}$$

where $\|\boldsymbol{v}\|_1 = \sum_{i=1}^{m} |v_i|$ denotes the ℓ_1 norm in coefficient space, using the soft margin empirical risk,

$$R_{\text{emp}}[g] = \frac{1}{m} \sum_i \xi_i, \tag{7.63}$$