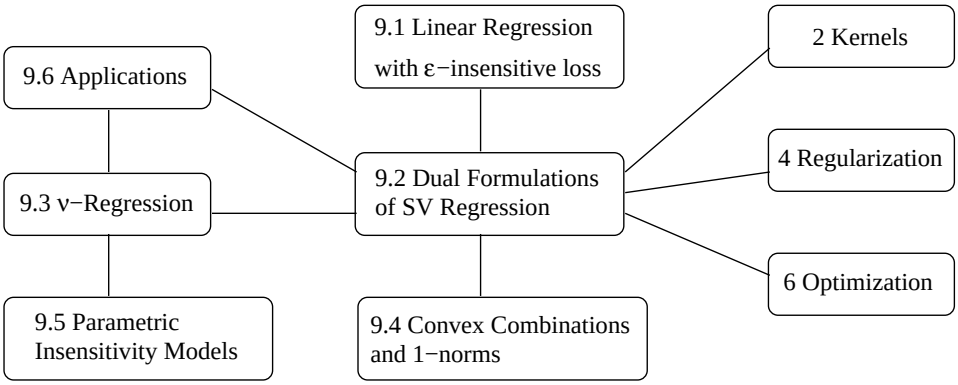

which does not penalize errors below some $\varepsilon \geq 0$, chosen a priori.[1] The rationale behind this choice is the following. In pattern recognition, when measuring the loss incurred for a particular pattern, there is a large area where we accrue zero loss: whenever a pattern is on the correct side of the decision surface, and does not touch the margin, it does not contribute any loss to the objective function (7.35). Correspondingly, it does not carry any information about the position of the decision surface — after all, the latter is computed by minimizing that very objective function. This is the underlying reason why the pattern does not appear in the SV expansion of the solution. A loss function for regression estimation must also have an insensitive zone; hence we use the ε-insensitive loss.

The regression algorithm is then developed in close analogy to the case of pattern recognition. Again, we estimate linear functions, use a $\|\mathbf{w}\|^2$ regularizer, and rewrite everything in terms of dot products to generalize to the nonlinear case. The basic SV regression algorithm, which we will henceforth call ε**-SVR**, seeks to estimate linear functions[2],

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b, \text{ where } \mathbf{w}, \mathbf{x} \in \mathcal{H}, b \in \mathbb{R}, \tag{9.2}$$

based on independent and identically distributed (iid) data,

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m) \in \mathcal{H} \times \mathbb{R}. \tag{9.3}$$

Here, $\mathcal{H}$ is a dot product space in which the (mapped) input patterns live (i.e., the feature space induced by a kernel). The goal of the learning process is to find a

1. The insensitive zone is sometimes referred to as the ε*-tube*. Actually, this term is slightly misleading, as in multi-dimensional problems, the insensitive zone has the shape of a *slab* rather than a tube; in other words, the region between two parallel hyperplanes, differing in their y offset.
2. Strictly speaking, these should be called *affine* functions. We will not indulge in these fine distinctions. The crucial bit is that the part to which we apply the kernel trick is linear.