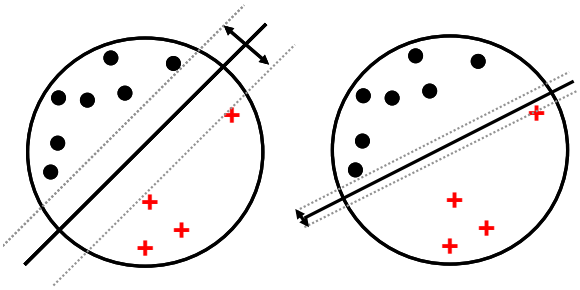


Figure 17.12: Illustration of the large margin principle. Left: a separating hyper-plane with large margin. Right: a separating hyper-plane with small margin.

17.3 Support vector machines (SVMs)

In this section, we discuss a form of (non-probabilistic) predictors for classification and regression problems which have the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}_i) \tag{17.63}$$

By adding suitable constraints, we can ensure that many of the α_i coefficients are 0, so that predictions at test time only depend on a subset of the training points, known as "**support vectors**". Hence the resulting model is called a **support vector machine** or **SVM**. We give a brief summary below. More details, can be found in e.g., [VGS97; SS01].

17.3.1 Large margin classifiers

Consider a binary classifier of the form $h(\boldsymbol{x}) = \text{sign}(f(\boldsymbol{x}))$, where the decision boundary is given by the following linear function:

$$f(\boldsymbol{x}) = \boldsymbol{w}^\mathsf{T}\boldsymbol{x} + w_0 \tag{17.64}$$

(In the SVM literature, it is common to assume the class labels are -1 and $+1$, rather than 0 and 1. To avoid confusion, we denote such target labels by $\tilde{y}$ rather than y.) There may be many lines that separate the data. However, intuitively we would like to pick the one that has maximum **margin**, which is the distance of the closest point to the decision boundary, since this will give us the most robust solution. This idea is illustrated in Figure 17.12: the solution on the left has larger margin than the one on the right, so it will be less sensitive to perturbations of the data.

How can we compute such a **large margin classifier**? First we need to derive an expression for the distance of a point to the decision boundary. Referring to Figure 17.13(a), we see that

$$\boldsymbol{x} = \boldsymbol{x}_\perp + r\frac{\boldsymbol{w}}{||\boldsymbol{w}||} \tag{17.65}$$

where r is the distance of $\boldsymbol{x}$ from the decision boundary whose normal vector is $\boldsymbol{w}$, and $\boldsymbol{x}_\perp$ is the orthogonal projection of $\boldsymbol{x}$ onto this boundary.

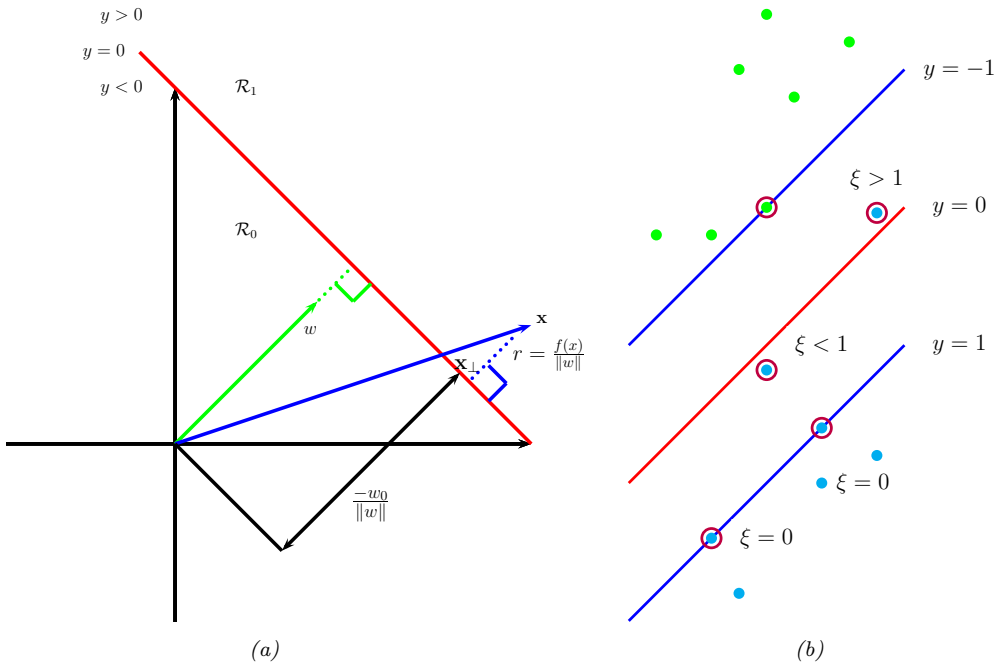


Figure 17.13: (a) Illustration of the geometry of a linear decision boundary in 2d. A point $\boldsymbol{x}$ is classified as belonging in decision region $\mathcal{R}_1$ if $f(\boldsymbol{x}) > 0$, otherwise it belongs in decision region $\mathcal{R}_0$; $\boldsymbol{w}$ is a vector which is perpendicular to the decision boundary. The term w_0 controls the distance of the decision boundary from the origin. $\boldsymbol{x}_\perp$ is the orthogonal projection of $\boldsymbol{x}$ onto the boundary. The signed distance of $\boldsymbol{x}$ from the boundary is given by $f(\boldsymbol{x})/||\boldsymbol{w}||$. Adapted from Figure 4.1 of [Bis06]. (b) Points with circles around them are support vectors, and have dual variables $\alpha_n > 0$. In the soft margin case, we associate a slack variable ξ_n with each example. If $0 < \xi_n < 1$, the point is inside the margin, but on the correct side of the decision boundary. If $\xi_n > 1$, the point is on the wrong side of the boundary. Adapted from Figure 7.3 of [Bis06].

We would like to maximize r, so we need to express it as a function of $\boldsymbol{w}$. First, note that

$$f(\boldsymbol{x}) = \boldsymbol{w}^\mathsf{T}\boldsymbol{x} + w_0 = (\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_\perp + w_0) + r\frac{\boldsymbol{w}^\mathsf{T}\boldsymbol{w}}{||\boldsymbol{w}||} = (\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_\perp + w_0) + r||\boldsymbol{w}|| \tag{17.66}$$

Since $0 = f(\boldsymbol{x}_\perp) = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_\perp + w_0$, we have $f(\boldsymbol{x}) = r||\boldsymbol{w}||$ and hence $r = \frac{f(\boldsymbol{x})}{||\boldsymbol{w}||}$.

Since we want to ensure each point is on the correct side of the boundary, we also require $f(\boldsymbol{x}_n)\tilde{y}_n > 0$. We want to maximize the distance of the closest point, so our final objective becomes

$$\max_{\boldsymbol{w},w_0} \frac{1}{||\boldsymbol{w}||} \min_{n=1}^{N} \left[\tilde{y}_n(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n + w_0)\right] \tag{17.67}$$

Note that by rescaling the parameters using $\boldsymbol{w} \to k\boldsymbol{w}$ and $w_0 \to kw_0$, we do not change the distance of any point to the boundary, since the k factor cancels out when we divide by $||\boldsymbol{w}||$. Therefore let us define the scale factor such that $\tilde{y}_n f_n = 1$ for the point that is closest to the decision boundary.

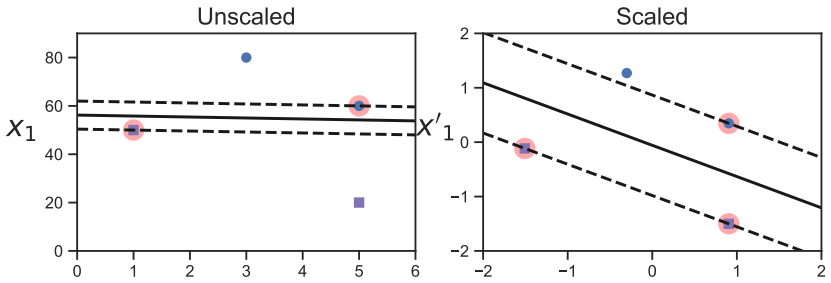


Figure 17.14: Illustration of the benefits of scaling the input features before computing a max margin classifier. Adapted from Figure 5.2 of [Gér19]. Generated by svm_classifier_feature_scaling.ipynb.

Hence we require $\tilde{y}_n f_n \geq 1$ for all n. Finally, note that maximizing $1/||\boldsymbol{w}||$ is equivalent to minimizing $||\boldsymbol{w}||^2$. Thus we get the new objective

$$\min_{\boldsymbol{w}, w_0} \frac{1}{2}||\boldsymbol{w}||^2 \quad \text{s.t.} \quad \tilde{y}_n(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n + w_0) \geq 1, n = 1:N \tag{17.68}$$

(The factor of $\frac{1}{2}$ is added for convenience and doesn't affect the optimal parameters.) The constraint says that we want all points to be on the correct side of the decision boundary with a margin of at least 1.

Note that it is important to scale the input variables before using an SVM, otherwise the margin measures distance of a point to the boundary using all input dimensions equally. See Figure 17.14 for an illustration.

17.3.2 The dual problem

The objective in Equation (17.68) is a standard quadratic programming problem (Section 8.5.4), since we have a quadratic objective subject to linear constraints. This has $N + D + 1$ variables subject to N constraints, and is known as a **primal problem**.

In convex optimization, for every primal problem we can derive a **dual problem**. Let $\boldsymbol{\alpha} \in \mathbb{R}^N$ be the dual variables, corresponding to Lagrange multipliers that enforce the N inequality constraints. The generalized Lagrangian is given below (see Section 8.5.2 for relevant background information on constrained optimization):

$$\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{w}^\mathsf{T}\boldsymbol{w} - \sum_{n=1}^{N} \alpha_n(\tilde{y}_n(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n + w_0) - 1) \tag{17.69}$$

To optimize this, we must find a stationary point that satisfies

$$(\hat{\boldsymbol{w}}, \hat{w}_0, \hat{\boldsymbol{\alpha}}) = \min_{\boldsymbol{w}, w_0} \max_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\alpha}) \tag{17.70}$$

We can do this by computing the partial derivatives wrt $\boldsymbol{w}$ and w_0 and setting to zero. We have

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\alpha}) = \boldsymbol{w} - \sum_{n=1}^{N} \alpha_n \tilde{y}_n \boldsymbol{x}_n \tag{17.71}$$

$$\frac{\partial}{\partial w_0} \mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\alpha}) = -\sum_{n=1}^{N} \alpha_n \tilde{y}_n \tag{17.72}$$

and hence

$$\hat{\boldsymbol{w}} = \sum_{n=1}^{N} \hat{\alpha}_n \tilde{y}_n \boldsymbol{x}_n \tag{17.73}$$

$$0 = \sum_{n=1}^{N} \hat{\alpha}_n \tilde{y}_n \tag{17.74}$$

Plugging these into the Lagrangian yields the following

$$\mathcal{L}(\hat{\boldsymbol{w}}, \hat{w}_0, \boldsymbol{\alpha}) = \frac{1}{2} \hat{\boldsymbol{w}}^{\mathsf{T}} \hat{\boldsymbol{w}} - \sum_{n=1}^{N} \alpha_n \tilde{y}_n \hat{\boldsymbol{w}}^{\mathsf{T}} \boldsymbol{x}_n - \sum_{n=1}^{N} \alpha_n \tilde{y}_n w_0 + \sum_{n=1}^{N} \alpha_n \tag{17.75}$$

$$= \frac{1}{2} \hat{\boldsymbol{w}}^{\mathsf{T}} \hat{\boldsymbol{w}} - \hat{\boldsymbol{w}}^{\mathsf{T}} \hat{\boldsymbol{w}} - 0 + \sum_{n=1}^{N} \alpha_n \tag{17.76}$$

$$= -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j \boldsymbol{x}_i^{\mathsf{T}} \boldsymbol{x}_j + \sum_{n=1}^{N} \alpha_n \tag{17.77}$$

This is called the **dual form** of the objective. We want to maximize this wrt $\boldsymbol{\alpha}$ subject to the constraints that $\sum_{n=1}^{N} \alpha_n \tilde{y}_n = 0$ and $0 \leq \alpha_n$ for $n = 1 : N$.

The above objective is a quadratic problem in N variables. Standard QP solvers take $O(N^3)$ time. However, specialized algorithms, which avoid the use of generic QP solvers, have been developed for this problem, such as the **sequential minimal optimization** or **SMO** algorithm [Pla98], which takes $O(N)$ to $O(N^2)$ time.

Since this is a convex objective, the solution must satisfy the KKT conditions (Section 8.5.2), which tell us that the following properties hold:

$$\alpha_n \geq 0 \tag{17.78}$$

$$\tilde{y}_n f(\boldsymbol{x}_n) - 1 \geq 0 \tag{17.79}$$

$$\alpha_n (\tilde{y}_n f(\boldsymbol{x}_n) - 1) = 0 \tag{17.80}$$

Hence either $\alpha_n = 0$ (in which case example n is ignored when computing $\hat{\boldsymbol{w}}$) or the constraint $\tilde{y}_n(\hat{\boldsymbol{w}}^{\mathsf{T}} \boldsymbol{x}_n + \hat{w}_0) = 1$ is active. This latter condition means that example n lies on the decision boundary; these points are known as the **support vectors**, as shown in Figure 17.13(b). We denote the set of support vectors by $\mathcal{S}$.

To perform prediction, we use

$$f(\boldsymbol{x}; \hat{\boldsymbol{w}}, \hat{w}_0) = \hat{\boldsymbol{w}}^\mathsf{T}\boldsymbol{x} + \hat{w}_0 = \sum_{n \in \mathcal{S}} \alpha_n \tilde{y}_n \boldsymbol{x}_n^\mathsf{T}\boldsymbol{x} + \hat{w}_0 \tag{17.81}$$

To solve for $\hat{w}_0$ we can use the fact that for any support vector, we have $\tilde{y}_n f(\boldsymbol{x}; \hat{\boldsymbol{w}}, \hat{w}_0) = 1$. Multiplying both sides by $\tilde{y}_n$, and exploiting the fact that $\tilde{y}_n^2 = 1$, we get $\hat{w}_0 = \tilde{y}_n - \hat{\boldsymbol{w}}^\mathsf{T}\boldsymbol{x}_n$. In practice we get better results by averaging over all the support vectors to get

$$\hat{w}_0 = \frac{1}{|\mathcal{S}|} \sum_{n \in \mathcal{S}} (\tilde{y}_n - \hat{\boldsymbol{w}}^\mathsf{T}\boldsymbol{x}_n) = \frac{1}{|\mathcal{S}|} \sum_{n \in \mathcal{S}} (\tilde{y}_n - \sum_{m \in \mathcal{S}} \alpha_m \tilde{y}_m \boldsymbol{x}_m^\mathsf{T}\boldsymbol{x}_n) \tag{17.82}$$

17.3.3 Soft margin classifiers

If the data is not linearly separable, there will be no feasible solution in which $\tilde{y}_n f_n \geq 1$ for all n. We therefore introduce **slack variables** $\xi_n \geq 0$ and replace the hard constraints that $\tilde{y}_n f_n \geq 0$ with the **soft margin constraints** that $\tilde{y}_n f_n \geq 1 - \xi_n$. The new objective becomes

$$\min_{\boldsymbol{w}, w_0, \boldsymbol{\xi}} \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{n=1}^{N} \xi_n \quad \text{s.t.} \quad \xi_n \geq 0, \quad \tilde{y}_n(\boldsymbol{x}_n^\mathsf{T}\boldsymbol{w} + w_0) \geq 1 - \xi_n \tag{17.83}$$

where $C \geq 0$ is a hyper parameter controlling how many points we allow to violate the margin constraint. (If $C = \infty$, we recover the unregularized, hard-margin classifier.)

The corresponding Lagrangian for the soft margin classifier becomes

$$\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\alpha}, \boldsymbol{\xi}, \boldsymbol{\mu}) = \frac{1}{2}\boldsymbol{w}^\mathsf{T}\boldsymbol{w} + C \sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} \alpha_n (\tilde{y}_n(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n + w_0) - 1 + \xi_n) - \sum_{n=1}^{N} \mu_n \xi_n \tag{17.84}$$

where $\alpha_n \geq 0$ and $\mu_n \geq 0$ are the Lagrange multipliers. Optimizing out $\boldsymbol{w}$, w_0 and $\boldsymbol{\xi}$ gives the dual form

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j \boldsymbol{x}_i^\mathsf{T}\boldsymbol{x}_j \tag{17.85}$$

This is identical to the hard margin case; however, the constraints are different. In particular, the KKT conditions imply

$$0 \leq \alpha_n \leq C \tag{17.86}$$

$$\sum_{n=1}^{N} \alpha_n \tilde{y}_n = 0 \tag{17.87}$$

If $\alpha_n = 0$, the point is ignored. If $0 < \alpha_n < C$ then $\xi_n = 0$, so the point lies on the margin. If $\alpha_n = C$, the point can lie inside the margin, and can either be correctly classified if $\xi_n \leq 1$, or misclassified if $\xi_n > 1$. See Figure 17.13(b) for an illustration. Hence $\sum_n \xi_n$ is an upper bound on the number of misclassified points.

As before, the bias term can be computed using

$$\hat{w}_0 = \frac{1}{|\mathcal{M}|} \sum_{n \in \mathcal{M}} (\tilde{y}_n - \sum_{m \in \mathcal{S}} \alpha_m \tilde{y}_m \boldsymbol{x}_m^\mathsf{T} \boldsymbol{x}_n) \tag{17.88}$$

where $\mathcal{M}$ is the set of points having $0 < \alpha_n < C$.

There is an alternative formulation of the soft margin SVM known as the **ν-SVM classifier** [Sch+00]. This involves maximizing

$$\mathcal{L}(\boldsymbol{\alpha}) = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j \boldsymbol{x}_i^\mathsf{T} \boldsymbol{x}_j \tag{17.89}$$

subject to the constraints that

$$0 \leq \alpha_n \leq 1/N \tag{17.90}$$

$$\sum_{n=1}^{N} \alpha_n \tilde{y}_n = 0 \tag{17.91}$$

$$\sum_{n=1}^{M} \alpha_n \geq \nu \tag{17.92}$$

This has the advantage that the parameter ν, which replaces C, can be interpreted as an upper bound on the fraction of **margin errors** (points for which $\xi_n > 0$), as well as a lower bound on the number of support vectors.

17.3.4 The kernel trick

So far we have converted the large margin binary classification problem into a dual problem in N unknowns ($\boldsymbol{\alpha}$) which (in general) takes $O(N^3)$ time to solve, which can be slow. However, the principal benefit of the dual problem is that we can replace all inner product operations $\boldsymbol{x}^\mathsf{T} \boldsymbol{x}'$ with a call to a positive definite (Mercer) kernel function, $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}')$. This is called the **kernel trick**.

In particular, we can rewrite the prediction function in Equation (17.81) as follows:

$$f(\boldsymbol{x}) = \hat{\boldsymbol{w}}^\mathsf{T} \boldsymbol{x} + \hat{w}_0 = \sum_{n \in \mathcal{S}} \alpha_n \tilde{y}_n \boldsymbol{x}_n^\mathsf{T} \boldsymbol{x} + \hat{w}_0 = \sum_{n \in \mathcal{S}} \alpha_n \tilde{y}_n \mathcal{K}(\boldsymbol{x}_n, \boldsymbol{x}) + \hat{w}_0 \tag{17.93}$$

We also need to kernelize the bias term. This can be done by kernelizing Equation (17.82) as follows:

$$\hat{w}_0 = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \left(\tilde{y}_i - (\sum_{j \in \mathcal{S}} \hat{\alpha}_j \tilde{y}_j \boldsymbol{x}_j)^\mathsf{T} \boldsymbol{x}_i \right) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \left(\tilde{y}_i - \sum_{j \in \mathcal{S}} \hat{\alpha}_j \tilde{y}_j \mathcal{K}(\boldsymbol{x}_j, \boldsymbol{x}_i) \right) \tag{17.94}$$

The kernel trick allows us to avoid having to deal with an explicit feature representation of our data, and allows us to easily apply classifiers to structured objects, such as strings and graphs.

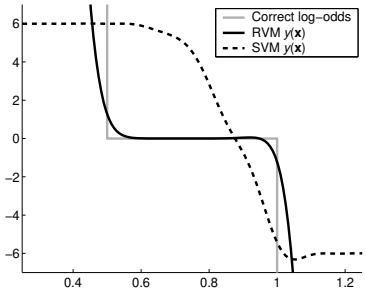


Figure 17.15: Log-odds vs x for 3 different methods. Adapted from Figure 10 of [Tip01]. Used with kind permission of Mike Tipping.

17.3.5 Converting SVM outputs into probabilities

An SVM classifier produces a hard-labeling, $\hat{y}(\boldsymbol{x}) = \text{sign}(f(\boldsymbol{x}))$. However, we often want a measure of confidence in our prediction. One heuristic approach is to interpret $f(\boldsymbol{x})$ as the log-odds ratio, $\log \frac{p(y=1|\boldsymbol{x})}{p(y=0|\boldsymbol{x})}$. We can then convert the output of an SVM to a probability using

$$p(y = 1|\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(af(\boldsymbol{x}) + b) \tag{17.95}$$

where a, b can be estimated by maximum likelihood on a separate validation set. (Using the training set to estimate a and b leads to severe overfitting.) This technique was first proposed in [Pla00], and is known as **Platt scaling**.

However, the resulting probabilities are not particularly well calibrated, since there is nothing in the SVM training procedure that justifies interpreting $f(\boldsymbol{x})$ as a log-odds ratio. To illustrate this, consider an example from [Tip01]. Suppose we have 1d data where $p(x|y = 0) = \text{Unif}(0, 1)$ and $p(x|y = 1) = \text{Unif}(0.5, 1.5)$. Since the class-conditional distributions overlap in the $[0.5, 1]$ range, the log-odds of class 1 over class 0 should be zero in this region, and infinite outside this region. We sampled 1000 points from the model, and then fit a probabilistic kernel classifier (an RVM, described in Section 17.4.1) and an SVM with a Gaussian kernel of width 0.1. Both models can perfectly capture the decision boundary, and achieve a generalization error of 25%, which is Bayes optimal in this problem. The probabilistic output from the RVM is a good approximation to the true log-odds, but this is not the case for the SVM, as shown in Figure 17.15.

17.3.6 Connection with logistic regression

We have seen that data points that are on the correct side of the decision boundary have $\xi_n = 0$; for the others, we have $\xi_n = 1 - \tilde{y}_n f(\boldsymbol{x}_n)$. Therefore we can rewrite the objective in Equation (17.83) as follows:

$$\mathcal{L}(\boldsymbol{w}) = \sum_{n=1}^{N} \ell_{\text{hinge}}(\tilde{y}_n, f(\boldsymbol{x}_n))) + \lambda ||\boldsymbol{w}||^2 \tag{17.96}$$

where $\lambda = (2C)^{-1}$ and $\ell_{\text{hinge}}(y, \eta)$ is the **hinge loss** function defined by

$$\ell_{\text{hinge}}(\tilde{y}, \eta) = \max(0, 1 - \tilde{y}\eta) \tag{17.97}$$

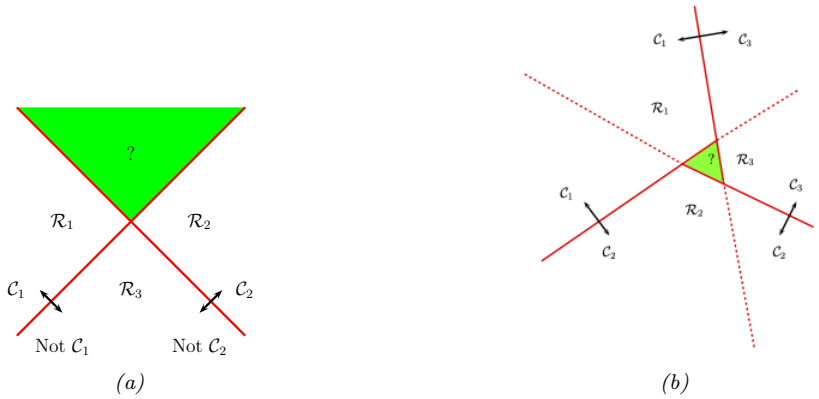


Figure 17.16: (a) The one-versus-rest approach. The green region is predicted to be both class 1 and class 2. (b) The one-versus-one approach. The label of the green region is ambiguous. Adapted from Figure 4.2 of [Bis06].

As we see from Figure 4.2, this is a convex, piecewise differentiable upper bound to the 0-1 loss, that has the shape of a partially open door hinge.

By contrast, (penalized) logistic regression optimizes

$$\mathcal{L}(\boldsymbol{w}) = \sum_{n=1}^{N} \ell_{ll}(\tilde{y}_n, f(\boldsymbol{x}_n))) + \lambda ||\boldsymbol{w}||^2 \tag{17.98}$$

where the **log loss** is given by

$$\ell_{ll}(\tilde{y}, \eta) = -\log p(y|\eta) = \log(1 + e^{-\tilde{y}\eta}) \tag{17.99}$$

This is also plotted in Figure 4.2. We see that it is similar to the hinge loss, but with two important differences. First the hinge loss is piecewise linear, so we cannot use regular gradient methods to optimize it. (We can, however, compute the subgradient at $\tilde{y}\eta = 1$.) Second, the hinge loss has a region where it is strictly 0; this results in sparse estimates.

We see that both functions are convex upper bounds on the 0-1 loss, which is given by

$$\ell_{01}(\tilde{y}, \hat{y}) = \mathbb{I}(\tilde{y} \neq \hat{y}) = \mathbb{I}(\tilde{y}\,\hat{y} < 0) \tag{17.100}$$

These upper bounds are easier to optimize and can be viewed as surrogates for the 0-1 loss. See Section 4.3.2 for details.

17.3.7 Multi-class classification with SVMs

SVMs are inherently a binary classifier. One way to convert them to a multi-class classification model is to train C binary classifiers, where the data from class c is treated as positive, and the data from all the other classes is treated as negative. We then use the rule $\hat{y}(\boldsymbol{x}) = \arg\max_c f_c(\boldsymbol{x})$ to

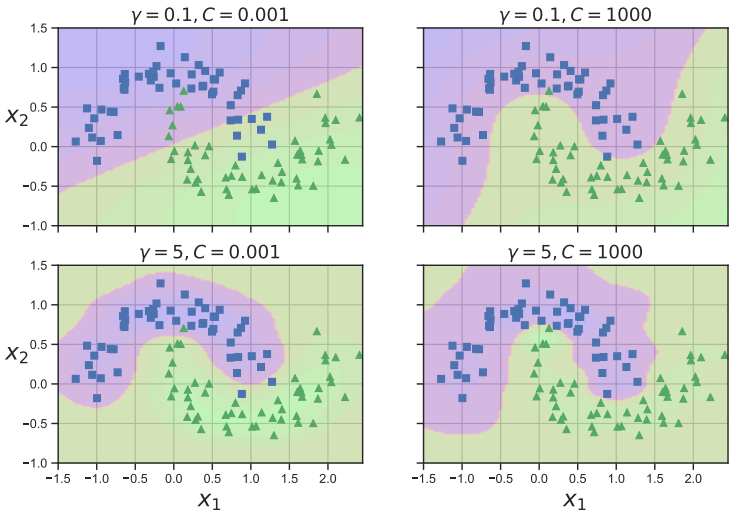


Figure 17.17: SVM classifier with RBF kernel with precision γ and regularizer C applied to two moons data. Adapted from Figure 5.9 of [Gér19]. Generated by svm_classifier_2d.ipynb.

predict the final label, where $f_c(\boldsymbol{x}) = \log \frac{p(c=1|\boldsymbol{x})}{p(c=0|\boldsymbol{x})}$ is the score given by classifier c. This is known as the **one-versus-the-rest** approach (also called **one-vs-all**).

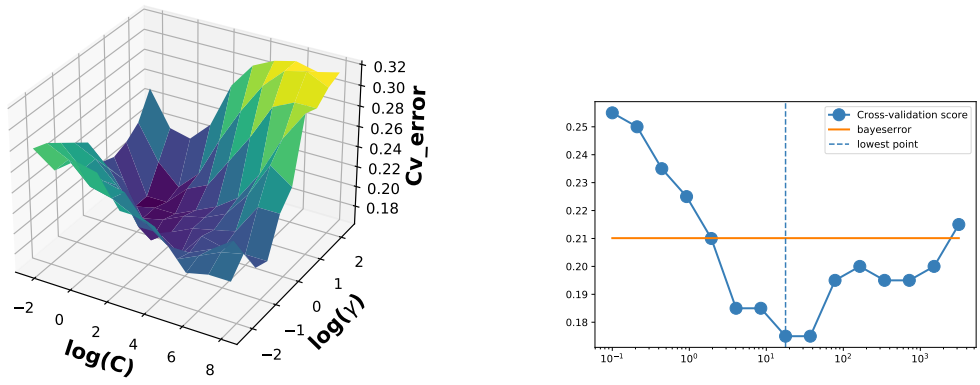
Unfortunately, this approach has several problems. First, it can result in regions of input space which are ambiguously labeled. For example, the green region at the top of Figure 17.16(a) is predicted to be both class 2 and class 1. A second problem is that the magnitude of the f_c's scores are not calibrated with each other, so it is hard to compare them. Finally, each binary subproblem is likely to suffer from the class imbalance problem (Section 10.3.8.2). For example, suppose we have 10 equally represented classes. When training f_1, we will have 10% positive examples and 90% negative examples, which can hurt performance.

Another approach is to use the **one-versus-one** or OVO approach, also called **all pairs**, in which we train $C(C-1)/2$ classifiers to discriminate all pairs $f_{c,c'}$. We then classify a point into the class which has the highest number of votes. However, this can also result in ambiguities, as shown in Figure 17.16(b). Also, this requires fitting $O(C^2)$ models.

17.3.8 How to choose the regularizer C

SVMs require that you specify the kernel function and the parameter C. Typically C is chosen by cross-validation. Note, however, that C interacts quite strongly with the kernel parameters. For example, suppose we are using an RBF kernel with precision $\gamma = \frac{1}{2\sigma^2}$. If γ is large, corresponding to narrow kernels, we may need heavy regularization, and hence small C. If γ is small, a larger value of C should be used. So we see that γ and C are tightly coupled, as illustrated in Figure 17.17.

The authors of libsvm [HCL03] recommend using CV over a 2d grid with values $C \in \{2^{-5}, 2^{-3}, \ldots, 2^{15}\}$



(a) (b)

Figure 17.18: (a) A cross validation estimate of the 0-1 error for an SVM classifier with RBF kernel with different precisions $\gamma = 1/(2\sigma^2)$ and different regularizer $\lambda = 1/C$, applied to a synthetic data set drawn from a mixture of 2 Gaussians. (b) A slice through this surface for $\gamma = 5$ The red dotted line is the Bayes optimal error, computed using Bayes rule applied to the model used to generate the data. Adapted from Figure 12.6 of [HTF09]. Generated by svmCgammaDemo.ipynb.

and $\gamma \in \{2^{-15}, 2^{-13}, \ldots, 2^3\}$. See Figure 17.18 which shows the CV estimate of the 0-1 risk as a function of C and γ.

To choose C efficiently, one can develop a path following algorithm in the spirit of lars (Section 11.4.4). The basic idea is to start with C small, so that the margin is wide, and hence all points are inside of it and have $\alpha_i = 1$. By slowly increasing C, a small set of points will move from inside the margin to outside, and their α_i values will change from 1 to 0, as they cease to be support vectors. When C is maximal, the margin becomes empty, and no support vectors remain. See [Has+04] for the details.

17.3.9 Kernel ridge regression

Recall the equation for ridge regression from Equation (11.55):

$$\hat{\boldsymbol{w}}_{\text{map}} = (\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda \mathbf{I}_D)^{-1}\mathbf{X}^\mathsf{T}\boldsymbol{y} = (\sum_n \boldsymbol{x}_n \boldsymbol{x}_n^\mathsf{T} + \lambda \mathbf{I}_D)^{-1}(\sum_n \tilde{y}_n \boldsymbol{x}_n) \tag{17.101}$$

Using the matrix inversion lemma (Section 7.3.3), we can rewrite the ridge estimate as follows

$$\boldsymbol{w} = \mathbf{X}^\mathsf{T}(\mathbf{X}\mathbf{X}^\mathsf{T} + \lambda \mathbf{I}_N)^{-1}\boldsymbol{y} = \sum_n \boldsymbol{x}_n((\sum_n \boldsymbol{x}_n^\mathsf{T}\boldsymbol{x}_n + \lambda \mathbf{I}_N)^{-1}\boldsymbol{y})_n \tag{17.102}$$

Let us define the following **dual variables**:

$$\boldsymbol{\alpha} \triangleq (\mathbf{X}\mathbf{X}^\mathsf{T} + \lambda \mathbf{I}_N)^{-1}\boldsymbol{y} = (\sum_n \boldsymbol{x}_n^\mathsf{T}\boldsymbol{x}_n + \lambda \mathbf{I}_N)^{-1}\boldsymbol{y} \tag{17.103}$$

Then we can rewrite the **primal variables** as follows

$$\boldsymbol{w} = \mathbf{X}^\mathsf{T}\boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n \tag{17.104}$$

This tells us that the solution vector is just a linear sum of the N training vectors. When we plug this in at test time to compute the predictive mean, we get

$$f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^\mathsf{T}\boldsymbol{x} = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n^\mathsf{T}\boldsymbol{x} \tag{17.105}$$

We can then use the kernel trick to rewrite this as

$$f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{n=1}^{N} \alpha_n \mathcal{K}(\boldsymbol{x}_n, \boldsymbol{x}) \tag{17.106}$$

where

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1}\boldsymbol{y} \tag{17.107}$$

In other words,

$$f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{k}^\mathsf{T}(\mathbf{K} + \lambda \mathbf{I}_N)^{-1}\boldsymbol{y} \tag{17.108}$$

where $\boldsymbol{k} = [\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}_N)]$. This is called **kernel ridge regression**.

The trouble with the above approach is that the solution vector $\boldsymbol{\alpha}$ is not sparse, so predictions at test time will take $O(N)$ time. We discuss a solution to this in Section 17.3.10.

17.3.10 SVMs for regression

Consider the following ℓ_2-regularized ERM problem:

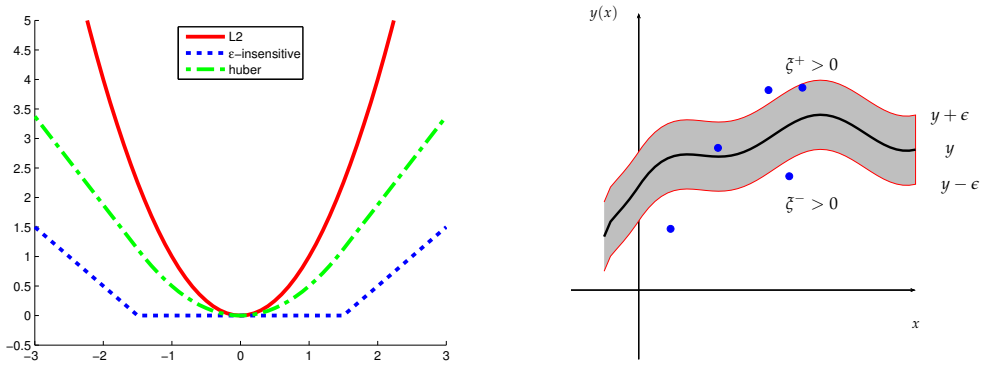
$$J(\boldsymbol{w}, \lambda) = \lambda ||\boldsymbol{w}||^2 + \sum_{n=1}^{N} \ell(\tilde{y}_n, \hat{y}_n) \tag{17.109}$$

where $\hat{y}_n = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n + w_0$. If we use the quadratic loss, $\ell(y, \hat{y}) = (y - \hat{y})^2$, where $y, \hat{y} \in \mathbb{R}$, we recover ridge regression (Section 11.3). If we then apply the kernel trick, we recover kernel ridge regression (Section 17.3.9).

The problem with kernel ridge regression is that the solution depends on all N training points, which makes it computationally intractable. However, by changing the loss function, we can make the optimal set of basis function coefficients, $\boldsymbol{\alpha}^*$, be sparse, as we show below.

In particular, consider the following variant of the Huber loss function (Section 5.1.5.3) called the **epsilon insensitive loss function**:

$$L_\epsilon(y, \hat{y}) \triangleq \begin{cases} 0 & \text{if } |y - \hat{y}| < \epsilon \\ |y - \hat{y}| - \epsilon & \text{otherwise} \end{cases} \tag{17.110}$$



(a) (b)

Figure 17.19: (a) Illustration of ℓ_2, Huber and ϵ-insensitive loss functions, where $\epsilon = 1.5$. Generated by huberLossPlot.ipynb. (b) Illustration of the ϵ-tube used in SVM regression. Points above the tube have $\xi_i^+ > 0$ and $\xi_i^- = 0$. Points below the tube have $\xi_i^+ = 0$ and $\xi_i^- > 0$. Points inside the tube have $\xi_i^+ = \xi_i^- = 0$. Adapted from Figure 7.7 of [Bis06].

This means that any point lying inside an **ϵ-tube** around the prediction is not penalized, as in Figure 17.19.

The corresponding objective function is usually written in the following form

$$J = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{n=1}^{N} L_\epsilon(\tilde{y}_n, \hat{y}_n) \tag{17.111}$$

where $\hat{y}_n = f(\boldsymbol{x}_n) = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n + w_0$ and $C = 1/\lambda$ is a regularization constant. This objective is convex and unconstrained, but not differentiable, because of the absolute value function in the loss term. As in Section 11.4.9, where we discussed the lasso problem, there are several possible algorithms we could use. One popular approach is to formulate the problem as a constrained optimization problem. In particular, we introduce **slack variables** to represent the degree to which each point lies outside the tube:

$$\tilde{y}_n \leq f(\boldsymbol{x}_n) + \epsilon + \xi_n^+ \tag{17.112}$$

$$\tilde{y}_n \geq f(\boldsymbol{x}_n) - \epsilon - \xi_n^- \tag{17.113}$$

Given this, we can rewrite the objective as follows:

$$J = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{n=1}^{N}(\xi_n^+ + \xi_n^-) \tag{17.114}$$

This is a quadratic function of $\boldsymbol{w}$, and must be minimized subject to the linear constraints in Equations 17.112-17.113, as well as the positivity constraints $\xi_n^+ \geq 0$ and $\xi_n^- \geq 0$. This is a standard quadratic program in $2N + D + 1$ variables.