Classifying by Maximizing the Margin

Ryan P. Adams
COS 324 – Elements of Machine Learning
Princeton University

The support vector machine (SVM) is one of the most elegant and successful ideas in machine learning. It takes a very different approach to binary classification than does logistic regression. Rather than directly specifying a simple loss function or constructing a likelihood function, the SVM starts with the premise that the data *are* linearly separable and then identifies an intuitive and appealing way to choose among possible separating hyperplanes: maximizing the distance between the "hardest" examples and the decision boundary. This is a way to handle undesirable behavior we have seen in other approaches. Specifically, the perceptron learning algorithm stops at the first separating hyperplane it finds, while the weights of unregularized logistic regression can explode.[1]

Reasoning About the Margin

We're going to tackle the binary classification problem where we have N training examples as usual. This time around, we're going to take the y_n to be in $\{-1, +1\}$. We're also going to imagine we're dealing with some set of basis functions right from the beginning. I'll denote the vector function that constructs the basis as $\Phi : \mathcal{X} \to \mathbb{R}^J$. One slight twist here is that for the SVM story we're going to keep the bias (intercept) term separate from the basis, so our classification function is going to be:

$$f(\boldsymbol{x}) = \boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}) + b, \tag{1}$$

where $b \in \mathbb{R}$ is the bias, and $\boldsymbol{w} \in \mathbb{R}^J$ is the vector of weights. As in the perceptron, we will classify using the sign function, i.e.,

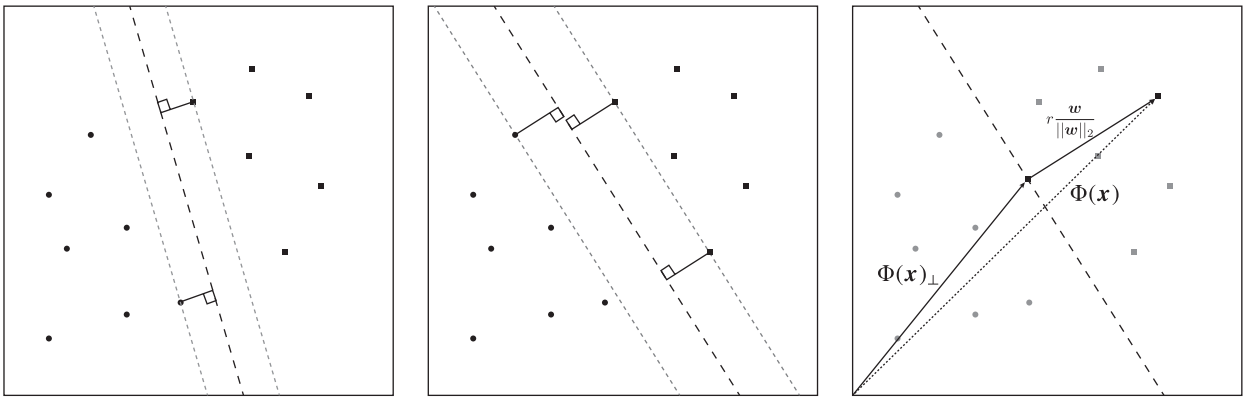
$$y = \mathrm{sgn}(f(\boldsymbol{x})) = \mathrm{sgn}(\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}) + b)\,. \tag{2}$$

Let's assume that the data are linearly separable. That means there exist some weights $\boldsymbol{w}$ such that

$$\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b > 0 \qquad \text{if } y_n = +1 \tag{3}$$

$$\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b \leq 0 \qquad \text{if } y_n = -1\,. \tag{4}$$

[1] Notes heavily influenced by Harvard CS181 course notes by Avi Pfeffer and David Parkes, which were in turn based on Bishop (1998).



(a) Small margin (b) Larger margin (c) Decomposing a point into projection and signed margin

Figure 1: Illustration of margin concept and projecting onto decision boundary.

The decision boundary, as in previous discussions, is the place where $\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}) + b = 0$. The *margin* is the distance from the closest example $\boldsymbol{x}_n$ to the decision boundary, *as computed in the basis function space* $\mathbb{R}^J$. That is, the margin measures the distance from the decision boundary to the example that is closest to being misclassified; we might think of this as the "most difficult" example to classify.

Figures 1a and 1b show the margin on the closest examples in $\mathbb{R}^2$ for a pair of classifiers (as indicated by its linear decision boundary.) One thing to note is that there are several closest examples to the hyperplane. This often happens in practice with SVMs. SVMs find a linear separator in possibly high dimensional feature space $\mathbb{R}^J$. The goal of an SVM is to find a linear separator that maximizes the margin given a particular input representation $\boldsymbol{x} \in \mathcal{X}$ and a feature mapping $\Phi(\cdot)$. Figure 1a shows a hypothesis with a smaller margin than Figure 1b. Intuitively, the first hypothesis is less ideal, because it has less robustness in being able to classify unseen examples that are distributed around the current examples. We might therefore expect it to generalize less well to unseen examples.

We can reason about the geometry of this J-dimensional space to figure out an expression for the margin. Any input $\boldsymbol{x}$ results in a point $\Phi(\boldsymbol{x})$ that can be decomposed into the sum of two vectors: a vector that is orthogonal to the hyperplane and a vector that is the projection of $\Phi(\boldsymbol{x})$ onto the hyperplane. See Figure 1c for an illustration. I'll denote this latter quantity as $\Phi(\boldsymbol{x})_\perp$. The former quantity—the vector going from the orthogonal projection $\Phi(\boldsymbol{x})_\perp$ to $\Phi(\boldsymbol{x})$, must be a scaled version of $\boldsymbol{w}$. You can convince yourself of this by considering any two points $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ that are right on the decision boundary. The vector $\Phi(\boldsymbol{x}_a) - \Phi(\boldsymbol{x}_b)$ must be parallel to the plane, but

$$\boldsymbol{w}^\mathsf{T}(\Phi(\boldsymbol{x}_a) - \Phi(\boldsymbol{x}_b)) = \boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_a) - \boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_b) = (-b) - (-b) = 0\,. \tag{5}$$

Because of this orthogonality, we can write the unit vector pointing from the hyperplane to any point $\Phi(\boldsymbol{x})$ as $\frac{\boldsymbol{w}}{||\boldsymbol{w}||_2}$. This allows us to write the point $\Phi(\boldsymbol{x})$ as an explicit sum:

$$\Phi(\boldsymbol{x}) = \Phi(\boldsymbol{x})_\perp + r\frac{\boldsymbol{w}}{||\boldsymbol{w}||_2}\,. \tag{6}$$

We just don't know what r should be because we don't know how far $\Phi(\boldsymbol{x})$ is from $\Phi(\boldsymbol{x})_\perp$. Let's figure this out by left-multiplying this decomposition by $\boldsymbol{w}^\mathsf{T}$:

$$\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}) = \boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x})_\perp + r\frac{\boldsymbol{w}^\mathsf{T}\boldsymbol{w}}{||\boldsymbol{w}||_2} \tag{7}$$

$$= -b + r||\boldsymbol{w}||_2\,. \tag{8}$$

This second line takes advantage of the fact that $\Phi(\boldsymbol{x})_\perp$ is in the hyperplane, so $\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x})_\perp + b = 0$, and that the squared L^2 norm is just the inner product of a vector with itself, i.e., $||\boldsymbol{w}||_2 = \sqrt{\boldsymbol{w}^\mathsf{T}\boldsymbol{w}}$. So now, for given parameters $\boldsymbol{w}, b$ and a fixed point in the input space $\boldsymbol{x}$, we can solve for r, which is the signed distance to the hyperplane, as

$$r = \frac{\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}) + b}{||\boldsymbol{w}||_2}\,. \tag{9}$$

Note that the numerator is just the classification function $f(\boldsymbol{x})$. So the signed distance to the margin is the classification function divided by the norm of the weight vector. For an actual example $\boldsymbol{x}_n$ where we know y_n, if we assume $\boldsymbol{w}$ is a linear separator, then the margin for the nth example is:

$$\frac{y_n(\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b)}{||\boldsymbol{w}||_2}\,. \tag{10}$$

Now we actually have a mathematical expression for the margin associated with one of our examples. We can talk about the overall margin of the classifier—as a function of parameters $\boldsymbol{w}$ and b—as the worst case margin over all the data in our training set:

$$\text{margin}(\boldsymbol{w}, b) = \min_{n\in 1,\ldots,N}\left\{\frac{y_n(\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b)}{||\boldsymbol{w}||_2}\right\} = \frac{1}{||\boldsymbol{w}||_2}\min_{n\in 1,\ldots,N}\left\{y_n(\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b)\right\}\,. \tag{11}$$

So this is saying that the quality of a given set of linearly separating parameters is determined by how much margin there is between the resulting hyperplane and the hardest training example. We could try to turn this into an objective function directly:

$$\boldsymbol{w}^\star, b^\star = \arg\max_{\boldsymbol{w},b}\left\{\frac{1}{||\boldsymbol{w}||_2}\min_{n\in 1,\ldots,N}\left\{y_n(\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b)\right\}\right\}, \tag{12}$$

but this would be difficult due to the nonlinearity arising from the norm. However, notice that we can multiply the weights and the bias by any constant $c > 0$ and it does not change the margin of any of the examples. That is, define $\boldsymbol{w}' = c\boldsymbol{w}$ and $b' = cb$, and observe:

$$\frac{y_n((\boldsymbol{w}')^\mathsf{T}\Phi(\boldsymbol{x}_n) + b')}{||\boldsymbol{w}'||_2} = \frac{c\,y_n(\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b)}{c||\boldsymbol{w}||_2} = \frac{y_n\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}) + b)}{||\boldsymbol{w}||_2}\,. \tag{13}$$

This means we can modify the scale of the problem arbitrarily. Recall that we are further assuming the data are linearly separable. Together, these properties mean we can simply fix the worst case to be one:

$$\min_{n\in 1,\ldots,N}\left\{y_n(\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b)\right\} = 1 \tag{14}$$

leading to the constraint

$$y_n(\boldsymbol{w}^\top \Phi(\boldsymbol{x}_n) + b) \geq 1 \qquad \text{for all } n \in 1, \ldots, N\,. \tag{15}$$

These constraints do not cause us to lose any generality in the solution space for the overall optimization problem. Now, looking again at Eqn 12, we can see there are two competing things: trying to maximize the minimum over the data, while also trying to minimize the norm of $\boldsymbol{w}$ (smaller $||\boldsymbol{w}||_2$ means its inverse is bigger). We've essentially declared that the former term will be one due to our freedom with the scale of $\boldsymbol{w}$, leaving just the objective of minimizing $||\boldsymbol{w}||_2$ (or equivalently minimizing $||\boldsymbol{w}||_2^2$). This gives us a new way to frame the optimization problem:

$$\boldsymbol{w}^\star, b^\star = \arg\min_{\boldsymbol{w},b} \left\{ \frac{1}{2}||\boldsymbol{w}||_2^2 \right\} \qquad \text{s.t.} \quad y_n(\boldsymbol{w}^\top \Phi(\boldsymbol{x}_n) + b) \geq 1 \quad \forall n \in 1, \ldots, N\,. \tag{16}$$

This problem looks very different, but it's actually quite nice: it is a quadratic program. That is, we are trying to minimize a quadratic subject to linear constraints. This is convex and a lot of smart people have thought hard about how to solve this kind of problem efficiently and at scale. In particular, there are methods that provably solve it in polynomial time.

Duality

Having now turned our margin maximization problem into a quadratic program, we're in good shape. When the data are linearly separable, we'll find a solution that makes sense in polynomial time using standard software. However, let's take a deeper dive and get a better understanding of why this approach to classification would be called a "support vector" machine.

The first thing to discuss is the notion of *duality*. Duality is an optimization principle that allows us to transform one problem into another equivalent problem. Ideally, this transformation can take something difficult and turn it into something easier. In a constrained optimization problem, the basic concept is to introduce the constraints into the objective function, adding new variables that govern the importance of each constraint to the solution.

Thinking about duality will require you to pull the concept of Lagrange multipliers out of neuronal cold storage where they have been since introductory calculus. We're going to introduce a Lagrange multiplier for every one of our constraints from Eqn. 16, giving us a set of variables $\alpha_1, \ldots, \alpha_N \geq 0$. There is one constraint for each of our data, so there is one Lagrange multiplier per datum. Now we can turn this into a Lagrangian, with variables $\boldsymbol{w}$, b, and $\boldsymbol{\alpha}$, and a new term for each constraint:

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\boldsymbol{w}||_2^2 - \sum_{n=1}^{N} \alpha_n (y_n(\boldsymbol{w}^\top \Phi(\boldsymbol{x}_n) + b) - 1)\,. \tag{17}$$

This produces an equivalent problem to Eqn 16 via

$$\boldsymbol{w}^\star, b^\star = \arg\min_{\boldsymbol{w},b} \max_{\boldsymbol{\alpha} \geq 0} L(\boldsymbol{w}, b, \boldsymbol{\alpha})\,. \tag{18}$$

The inner maximization encodes the constraints because a constraint violation means $y_n(\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n)+b) < 1$ for some n, and so then subtracting 1 makes the quantity weighted by α_n less than zero. Since this quantity is being *subtracted* in the Lagrangian, the inner maximization over $\boldsymbol{\alpha}$ can make it an arbitrarily large penalty. In the case where $\boldsymbol{w}, b$ are such that

$$y_n(\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b) \geq 1 \qquad \text{for all } n, \tag{19}$$

then

$$\max_{\boldsymbol{\alpha}\geq 0} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\boldsymbol{w}||_2^2 \tag{20}$$

since $\alpha_n = 0$ unless $y_n(\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b) = 1$, and $\alpha_n(y_n(\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b) - 1) = 0$ for all n. Moreover, we also have weak duality

$$\min_{\boldsymbol{w},b} \max_{\boldsymbol{\alpha}\geq 0} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) \geq \max_{\boldsymbol{\alpha}\geq 0} \min_{\boldsymbol{w},b} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) \tag{21}$$

and in fact, because the problem in Equation 16 has a quadratic objective and linear constraints we have strong duality

$$\min_{\boldsymbol{w},b} \max_{\boldsymbol{\alpha}\geq 0} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha}\geq 0} \min_{\boldsymbol{w},b} L(\boldsymbol{w}, b, \boldsymbol{\alpha})\,. \tag{22}$$

Thus, the order of min and max can be interchanged in Eqn. 16. This is very helpful, because it now means that for a given $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N)$, we can solve the first-order conditions for $\boldsymbol{w}$ and b, and achieve a further simplification. Specifically, we can require

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0, \qquad \frac{\partial L}{\partial b} = 0. \tag{23}$$

So for optimality we require

$$\frac{\partial L}{\partial w_j} = w_j - \sum_{n=1}^{N} \alpha_n\, y_n\, \phi_j(\boldsymbol{x}_n) = 0, \quad \forall j \in \{1, \ldots, J\} \tag{24}$$

and so

$$\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n\, y_n\, \Phi(\boldsymbol{x}_n)\,. \tag{25}$$

We also solve for b with the partial derivative of the Lagrangian set to zero and get

$$\frac{\partial L}{\partial b} = -\sum_{n=1}^{N} \alpha_n y_n = 0 \tag{26}$$

$$\sum_{n=1}^{N} \alpha_n y_n = 0\,. \tag{27}$$

We can now go back in and substitute Eqns 25 and 27 into the Lagrangian from Eqn 17:

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\boldsymbol{w}||_2^2 - \sum_{n=1}^{N} \alpha_n (y_n(\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b) - 1) \tag{28}$$

$$= \frac{1}{2}\boldsymbol{w}^\mathsf{T}\boldsymbol{w} - \left(\sum_{n=1}^{N} \alpha_n y_n \boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n)\right) - b\left(\sum_{n=1}^{N} \alpha_n y_n\right) + \sum_{n=1}^{N} \alpha_n \tag{29}$$

$$= \frac{1}{2}\boldsymbol{w}^\mathsf{T}\boldsymbol{w} - \boldsymbol{w}^\mathsf{T}\left(\sum_{n=1}^{N} \alpha_n y_n \Phi(\boldsymbol{x}_n)\right) + \sum_{n=1}^{N} \alpha_n \tag{30}$$

$$= \frac{1}{2}\boldsymbol{w}^\mathsf{T}\boldsymbol{w} - \boldsymbol{w}^\mathsf{T}\boldsymbol{w} + \sum_{n=1}^{N} \alpha_n \tag{31}$$

$$= \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\boldsymbol{w}^\mathsf{T}\boldsymbol{w} \tag{32}$$

$$= \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{n'=1}^{N} \alpha_n \alpha_{n'} y_n y_{n'} \Phi(\boldsymbol{x}_n)^\mathsf{T}\Phi(\boldsymbol{x}_{n'})\,. \tag{33}$$

Now we have another quadratic program, but with a different structure:

$$\boldsymbol{\alpha}^\star = \arg\max_{\boldsymbol{\alpha}} \left\{ \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{n'=1}^{N} \alpha_n \alpha_{n'} y_n y_{n'} \Phi(\boldsymbol{x}_n)^\mathsf{T}\Phi(\boldsymbol{x}_{n'}) \right\} \tag{34}$$

$$\text{where} \quad \alpha_n \geq 0 \quad \forall n \in 1, \ldots, N \qquad \text{and} \quad \sum_{n=1}^{N} \alpha_n y_n = 0\,. \tag{35}$$

This problem is defined only in terms of the Lagrange multipliers $\boldsymbol{\alpha}$, and there are as many of them as there are data. If you have a very large basis, this may be easier than working with $J+1$ variables. In the next lecture, we'll see how we can use a kernel function to make this representation even easier.

We solve this quadratic program to get an optimal set of variables $\boldsymbol{\alpha}^\star \in \mathbb{R}_+^N$, and then we can go back into Eq 25 to compute $\boldsymbol{w}$:

$$\boldsymbol{w}^\star = \sum_{n=1}^{N} \alpha_n^\star\, y_n\, \Phi(\boldsymbol{x}_n)\,. \tag{36}$$

That, along with $b^\star$ (the computation of which we'll examine in a moment), allows us to construct the classification function $f(\boldsymbol{x}) = (\boldsymbol{w}^\star)^\mathsf{T}\Phi(\boldsymbol{x}) + b^\star$. We can substitute Eqn. 36 back into this to get

$$f(\boldsymbol{x}) = \sum_{n=1}^{N} \alpha_n^\star y_n \Phi(\boldsymbol{x}_n)^\mathsf{T}\Phi(\boldsymbol{x}) + b^\star\,. \tag{37}$$

The inner product $\Phi(\boldsymbol{x}_n)^\mathsf{T}\Phi(\boldsymbol{x})$ will turn out to have a lot of significance when we discuss kernels next lecture.

Finally, now we can examine why this construction is called a *support vector machine*. The *support vectors* are the examples whose margin determines the location of the decision boundary, i.e., the ones closest to the hyperplane. These are the examples for which the corresponding $\alpha_n^\star$ is greater than zero, as their constraints were tight. We can see this by considering that the Karush-Kuhn-Tucker (KKT) conditions on the solution require

$$\alpha_n \left(y_n \left((\boldsymbol{w}^\star)^\mathsf{T}\Phi(\boldsymbol{x}_n) + b^\star\right) - 1\right) = 0, \quad \forall n \in \{1, \ldots, N\} \tag{38}$$

at the optimum. There are only two ways this can be achieved: by $\alpha_n^\star = 0$ or by $y_n((\boldsymbol{w}^\star)^\mathsf{T}\Phi(\boldsymbol{x}_n) + b^\star) = 1$. Any example that satisfies the latter case has $\alpha_n^\star > 0$ and is a support vector. Note that the prediction function in Eqn 37 only depends on these examples. That is, if we denote the set of indices of the support vectors as $\mathcal{S} = \{n : \alpha_n^\star, n \in 1, \ldots, N\}$, then we make predictions for a new point $\boldsymbol{x}$ via

$$\text{sgn}\left\{\sum_{n\in\mathcal{S}} \alpha_n^\star \, y_n \, \Phi(\boldsymbol{x}_n)^\mathsf{T}\Phi(\boldsymbol{x}) + b^\star\right\} . \tag{39}$$

We can view the SVM classification process as comparing the new instance $\boldsymbol{x}$ to each of the support vectors. The inner product $\Phi(\boldsymbol{x}_n)^\mathsf{T}\Phi(\boldsymbol{x})$ measures how similar the new point is to the support vectors. The support vector weights $\alpha_n^\star$ specify how important that support vector is to the decision boundary.

Coming back to $b^\star$, we can find this by recalling that for any support vector, $y_n((\boldsymbol{w}^\star)^\mathsf{T}\Phi(\boldsymbol{x}_n) + b^\star) = 1$, so if we take one of the support vectors we can get $b^\star$ via

$$b^\star = y_n - (\boldsymbol{w}^\star)^\mathsf{T}\Phi(\boldsymbol{x}_n) . \tag{40}$$

In practice, it is standard to average over the support vectors:

$$b^\star = \frac{1}{|\mathcal{S}|}\sum_{n\in\mathcal{S}} \left(y_n - (\boldsymbol{w}^\star)^\mathsf{T}\Phi(\boldsymbol{x}_n)\right) . \tag{41}$$

Non-separable Data

Everything we have discussed so far about SVMs has assumed that the data are linearly separable. What if they aren't? What if the data are noisy and there exists no decision boundary that is perfect on test data? It turns out that one can generalize support vector machines to allow for a "soft" margin where some examples are misclassified. In this setting, it is possible to construct a new primal:

$$\begin{aligned} \boldsymbol{w}^\star, b^\star = \arg\min_{\boldsymbol{w},b} &\left\{\frac{1}{2}||\boldsymbol{w}||_2^2 + C\sum_{n=1}^{N}\xi_n\right\} \\ \text{s.t.} \quad & y_n \cdot (\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b) \geq 1 - \xi_n, \quad \forall n \in \{1, \ldots, N\} \\ & \xi_n \geq 0, \quad \forall n \in \{1, \ldots, N\} \end{aligned} \tag{42}$$

for some $C > 0$. In this view, the norm on $\boldsymbol{w}$ can be viewed as a regularizer, trying to shrink the weights subject to improvement in the classification. The second term (with the C) is essentially an upper bound on the empirical zero-one training loss. To see this, consider a positive example with $y_n = 1$. Then, if $\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b = 0$, and the point is nearly misclassified, we have $\xi_n = 1$. For $\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b < 0$ then ξ_n increases. For $\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b > 0$ then ξ_n decreases and is zero when $\boldsymbol{w}^\mathsf{T}\Phi(\boldsymbol{x}_n) + b \geq 1$. This is a convex upper bound on the error. We think of C now as being a regularization parameter that must be set, with larger values of C emphasizing training error and lower values emphasizing the norm of $\boldsymbol{w}$. Typically, C would be set via cross validation.

Conveniently, it turns out that the dual of this "softened" primal optimization problem is just a small change to the "hard" problem. All that happens is we have an upper bound now on the α_n:

$$0 \leq \alpha_n \leq C, \quad \forall n \in \{1, \ldots, N\}\,. \tag{43}$$

Everything else is essentially the same, except for the way we compute $b^\star$.

Changelog

- 9 October 2018 – Initial version.
- 11 October 2018 – Added soft margin section.
- 15 October 2018 – Added complete dual problem statement for clarity.