Machine Learning
the Fundamentals

William J. Deuschle

Harvard College
Cambridge, Massachusetts
April, 2019

Contents

Chapter 5

Support Vector Machines

In this chapter, we will explore what is known as a support vector machine, or SVM for short. SVMs are broadly useful for problems in classification and regression, and they draw on techniques from what are known as *kernel algorithms*. This means they make use of an arbitrary kernel function $k(\cdot)$ to compare data points in our training set to new data points about which we wish to make a prediction. One of the most important aspects of SVMs is that they decompose into convex optimization problems, for which we can find a global optimum with relative ease. We will explore the mathematical underpinnings of SVMs, which can be slightly more challenging than our previous topics, as well as their typical use cases.

5.1 Motivation

While SVMs can be used for classification or regression, we will reason about them in the classification case as it is more straightforward.

The grand idea behind SVMs is that we should construct a linear hyperplane in our feature space that maximally separates our classes, which means that the different classes should be as far from that hyperplane as possible. The distance of our data from the hyperplane is known as *margin*.

Definition 5.1.1 (Margin): Margin is the distance of data points from the separating hyperplane of an SVM model. Larger margins lead to more generalizable models.

A larger margin tends to mean that our model will generalize better, since it provides more wiggle room to correctly classify unseen data.

This idea of margin is quite intuitive for humans. If you were presented with Figure 5.1 and were asked to separate the two classes, you would likely draw the line that keeps data points as far from it as possible. SVMs and other margin-based methods will attempt to algorithmically recreate this intuition.

5.1.1 Max Margin Methods

SVMs are a specific instance of a broader class of model known as *max margin methods*. Their name describes them well: they deal with creating a maximum margin between training data and a model, with the idea that this leads to model generalizability.

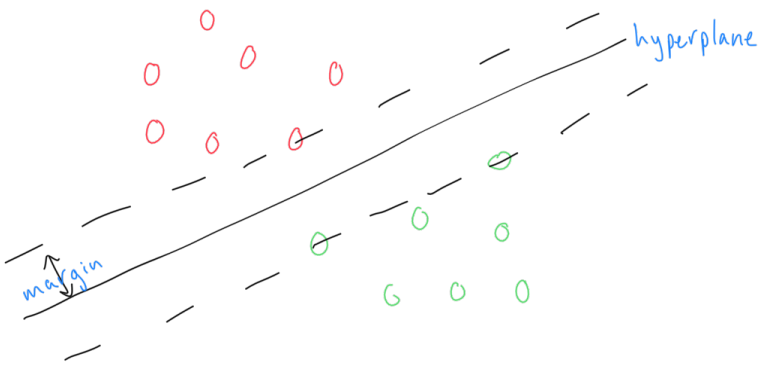

Figure 5.1: Hyperplane with margin between different classes.

Other max margin methods are outside the scope of this textbook. Note that these alternative methods typically differ from SVMs in some non-trivial manner. For example, SVMs do not produce posterior probability distributions but rather decision rules for handling new data points. If you needed posterior probabilities for your model, there are other max margin methods better suited to that task.

ML Framework Cube: Support Vector Machines

SVMs typically operate over inputs that exist in a continuous feature space. We need labeled training data to identify the relevant hyperplane in an SVM model. Finally, SVMs operate in a non-probabilistic setting.

Domain	***Training***	***Probabilistic***
Continuous	Supervised	No

5.1.2 Applications

The theory behind SVMs has been around for quite some time (since 1963), so prior to the rise of neural networks and other more computationally intensive techniques, SVMs were used quite extensively for image recognition, object categorization, and other typical machine learning tasks.

In particular, SVMs were and still are widely used for a subset of classification problems known as anomaly detection.

⋆ The purpose of anomaly detection is to identify unusual data points. For example, if we are manufacturing widgets, we may wish to inspect and flag any widget that seems atypical with respect to the rest of the widgets we produce.

Anomaly detection can be as simple as a binary classification problem where the data set is comprised of anomalous and non-anomalous data points. As we will see, an SVM can be constructed from this data set to identify future anomalous points very efficiently.

5.2 Hard Margin Classifier

We will learn the theory behind SVMs by starting with a simple two-class classification problem, as we've seen several times in previous chapters. We will constrain the problem even further

by declaring that the two classes are linearly separable, which is the basis of the *hard margin* formulation for SVMs.

⋆ The expression 'hard margin' simply means that the data are linearly separable, so we can find a hyperplane that perfectly separates the data based on class.

5.2.1 Why the Hard Margin

The hard margin constraint, which enforces that our data is linearly separable, is not actually a requirement for constructing an SVM, but it simplifies the problem initially and makes our derivations significantly easier. After we've established the hard margin formulation, we will extend the technique to work in situations where our data is not linearly separable.

5.2.2 Deriving our Optimization Problem

Recall that our goal is to define a hyperplane that both separates our data points and keeps maximum distance from them. To uncover this hyperplane, we start with a simple linear model for a two-class classification problem:

$$f(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) + w_0 \tag{5.1}$$

where we have N multidimensional data points $\mathbf{x}_1, ..., \mathbf{x}_N$, $\phi(\cdot)$ is a standard basis transformation function, and there is a bias term w_0. Each of our data point has a class $y_1, ..., y_N$ which is either 1 or -1, and we assign new data points to class 1 or -1 according to the sign produced by our trained model $f(\mathbf{x})$.

By specifying our model this way, we have implicity defined a hyperplane separating our two classes given by:

$$\mathbf{w}^T\phi(\mathbf{x}) + w_0 = 0 \tag{5.2}$$

This is not an intuitive result, but we can increase clarity through a simple derivation.

Derivation 5.2.1 (Hyperplane Derivation): Imagine two data points x_1 and x_2 on the hyperplane defined by $\mathbf{w}^T\phi(\mathbf{x}) + w_0 = 0$. When we project their difference onto our model $\mathbf{w}$, we find:

$$\mathbf{w}^T(\mathbf{x}_1 - \mathbf{x}_2) = \mathbf{w}^T\mathbf{x}_1 - \mathbf{w}^T\mathbf{x}_2 = -w_0 - (-w_0) = 0 \tag{5.3}$$

which means that $\mathbf{w}$ is orthogonal to our hyperplane. We can visualize this in Figure 5.2.

Remember that we're trying to maximize the margin between our training data and the hyperplane. The fact that $\mathbf{w}$ is orthogonal to our hyperplane will help with this.

To determine the distance between a data point $\mathbf{x}$ and the hyperplane, which we denote d, we need the distance in the direction of $\mathbf{w}$ between the point and the hyperplane. We denote $\mathbf{x}_\perp$ to be the projection of $\mathbf{x}$ onto the hyperplane, which allows us to decompose $\mathbf{x}$ as the following:

$$\mathbf{x} = \mathbf{x}_\perp + d\frac{\mathbf{w}}{||\mathbf{w}||} \tag{5.4}$$

which is simply the sum of the portion of $\mathbf{x}$ perpendicular to $\mathbf{w}$ and the portion of $\mathbf{x}$ that is parallel to $\mathbf{w}$. From here we can solve for d:

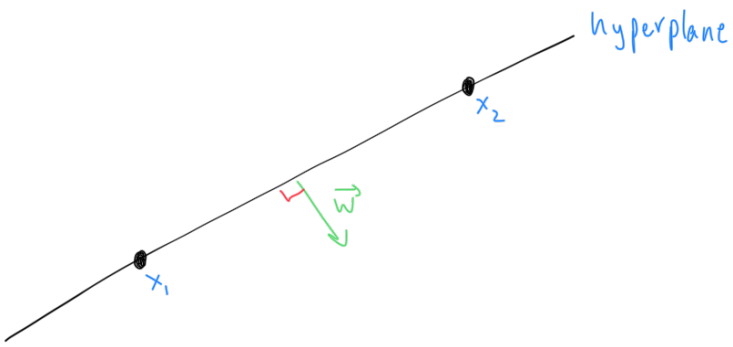

Figure 5.2: Our weight vector $\mathbf{w}$ is orthogonal to the separating hyperplane.

Derivation 5.2.2 (Distance from Hyperplane Derivation): We start by left multiplying Equation 5.4 with $\mathbf{w}^T$.

$$\mathbf{w}^T\mathbf{x} = \mathbf{w}^T\mathbf{x}_\perp + d\frac{\mathbf{w}^T\mathbf{w}}{||\mathbf{w}||}$$

Simplifying (note that $\mathbf{w}^T\mathbf{x}_\perp = -w_0$ from Equation 5.3):

$$\mathbf{w}^T\mathbf{x} = -w_0 + d||\mathbf{w}||$$

Rearranging:

$$d = \frac{\mathbf{w}^T\mathbf{x} + w_0}{||\mathbf{w}||}$$

which means that for each data point $\mathbf{x}$, we now have the signed distance of that data point from the hyperplane.

To classify a data point correctly, the distance d should be positive for class $y = 1$, and d should be negative for class $y = -1$. For training purposes, we can make the margin positive whenever we correctly classify data by multiplying y and d. Then, the margin for an individual data point $\mathbf{x}_n$ is given by:

$$\frac{y_n(\mathbf{w}^T\mathbf{x}_n + w_0)}{||\mathbf{w}||} \tag{5.5}$$

The margin for an entire data set is given by the margin to the closest point in the data set, given by:

$$\min_n \frac{y_n(\mathbf{w}^T\mathbf{x}_n + w_0)}{||\mathbf{w}||} \tag{5.6}$$

Then, it is our goal to maximize this margin with respect to our model parameters $\mathbf{w}$ and w_0. This is given by:

$$\arg\max_{\mathbf{w},w_0}\Big\{\frac{1}{||\mathbf{w}||}\min_n y_n(\mathbf{w}^T\mathbf{x}_n + w_0)\Big\} \tag{5.7}$$

This is a hard problem to optimize, but we can make it more tractable by recognizing some important features of Equation 5.7. First, rescaling $\mathbf{w} \to \alpha\mathbf{w}$ and $w_0 \to \beta w_0$ has no impact on the

relative distance of any data point $\mathbf{x}_n$ from the hyperplane. We can use this rescaling liberty to enforce

$$y_n(\mathbf{w}^T\mathbf{x}_n + w_0) = 1 \tag{5.8}$$

for the data point closest to the hyperplane. Thus, all of our data points have a margin that is greater than or equal to 1:

$$\forall n \; y_n(\mathbf{w}^T\mathbf{x}_n + w_0) \geq 1 \tag{5.9}$$

which is used as a constraint in the optimization problem of Equation 5.7. Thus our optimization problem now looks like:

$$\arg\max_{\mathbf{w},w_0} \frac{1}{||\mathbf{w}||} \quad \text{s.t.} \quad \forall n \; y_n(\mathbf{w}^T\mathbf{x}_n + w_0) \geq 1 \tag{5.10}$$

Notice that maximizing $\frac{1}{||\mathbf{w}||}$ is equivalent to minimizing $||\mathbf{w}||^2$. We will also add a constant term $\frac{1}{2}$ for convenience, leaving our final optimization problem:

$$\arg\min_{\mathbf{w},w_0} \frac{1}{2}||\mathbf{w}||^2 \quad \text{s.t.} \quad \forall n \; y_n(\mathbf{w}^T\mathbf{x}_n + w_0) \geq 1 \tag{5.11}$$

Note that Equation 5.11 is now a quadratic programming problem, which means we wish to optimize a quadratic function subject to a set of linear constraints on our parameters. Arriving at this form was the motivation for the preceding mathematic manipulations. We will discuss shortly how we actually optimize this function.

5.2.3 What is a Support Vector

Up until now, we have discussed Support Vector Machines without identifying what a support vector is. We now have enough information from the previous section to define them.

Definition 5.2.1 (Support Vector): The support vectors in an SVM are the data points closest to the hyperplane.

In the hard margin case we have constrained the closest data points to being a distance of 1 from the hyperplane, so the support vectors are all $d = 1$ from the hyperplane. Figure 5.3 shows a hard margin SVM solution with the corresponding support vectors.

⋆ After we have optimized an SVM in the hard margin case, we must have at least two support vectors with a distance of 1 from the hyperplane.

5.3 Soft Margin Classifier

Thus far, we've been operating under the assumption that our data is linearly separable in feature space, which afforded us several convenient guarantees in the derivations of the previous section. For example, given that our data was linearly separable, we could guarantee that every data point would be on the correct side of the hyperplane, which was part of what allowed us to enforce the constraint that $d = 1$ for the points closest to the hyperplane. We now seek to generalize the work of the previous section to situations where our data is not so nice.

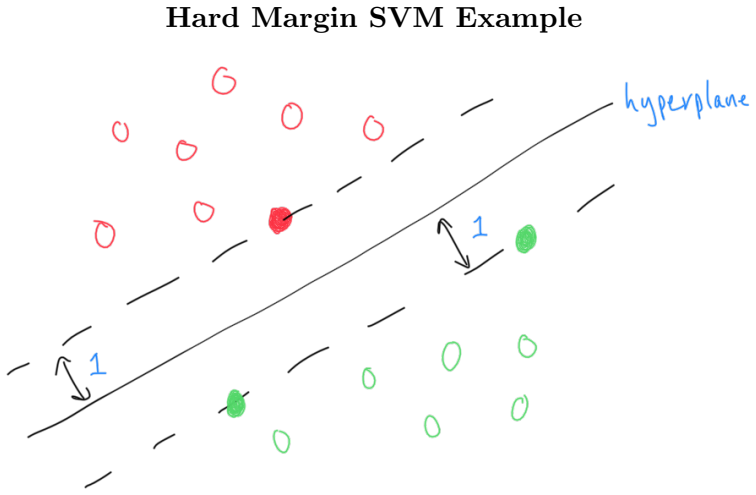

Figure 5.3: Example of the resulting hyperplane for a hard margin SVM. The filled in data points are support vectors.

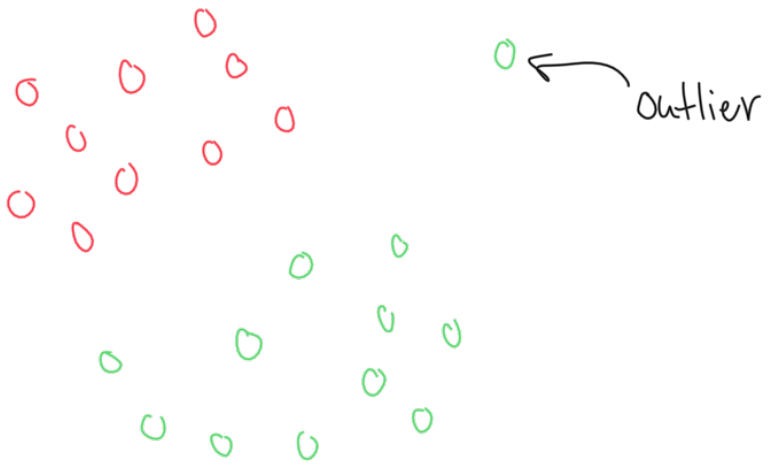

Figure 5.4: An outlier can make the hard margin formulation impractical.

5.3.1 Why the Soft Margin?

What if our data is not linearly separable in transformed feature space? For a real, complex problem domain, it's highly unlikely that it would be. Unfortunately, our current hard margin SVM formulation would be useless with non-linearly separable data. That is why we need the soft margin SVM.

At a high level, the soft margin SVM allows for some of our data points to be closer to or even on the incorrect side of the hyperplane. This is necessary if our data set is not linearly separable, and it is also quite intuitive. Examining Figure 5.4, we see that we have a single outlier data point. We can still create a good model by just allowing this single data point to be close to the hyperplane. That is what the soft margin formulation will allow for.

5.3.2 Updated Optimization Problem for Soft Margins

To enable the soft margin formulation, we introduce what are known as *slack variables* denoted $\xi_n \geq 0$, which simply relax the constraint from Equation 5.9 that we imposed in the hard margin formulation. There is a slack variable $\xi_n \geq 0$ for every data point $\mathbf{x}_n$, and they take the following values according to how we classify $\mathbf{x}_n$:

$$\xi_n = \begin{cases} = 0 & \text{if correctly classified} \\ \in (0, 1] & \text{if correctly classified but inside margin} \\ > 1 & \text{if incorrectly classified} \end{cases} \tag{5.12}$$

These slack variable penalize data points on the wrong side of the margin defined by the hyperplane, but they don't forbid us from allowing data points to be on the wrong side if it produces the best model. We now reformulate our optimization problem as:

$$\arg\min_{\mathbf{w}, w_0} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{n=1}^{N} \xi_n \quad \text{s.t.} \quad \forall n \, y_n(\mathbf{w}^T\mathbf{x}_n + w_0) \geq 1 - \xi_n, \quad \xi_n \geq 0 \tag{5.13}$$

where C is a regularization parameter that determines how heavily we penalize violations of the hard margin constraints. A large C penalizes violation of the hard margin constraints more heavily, which means our model will follow the data closely and have little regularization. A small C won't heavily penalize having data points inside the margin region, relaxing the constraint and allowing our model to somewhat disregard more of the data. This means more regularization.

⋆ Unlike most regularization parameters we've seen thus far, C increases regularization as it gets smaller.

5.3.3 Soft Margin Support Vectors

Under the the hard margin formulation, the support vectors were those data points exactly $d = 1$ from the hyperplane, and they were also guaranteed to be the points closest to the hyperplane. Under the soft margin formulation, we no longer have this guarantee since we explicity relaxed it in the name of creating better, more generalizable models.

The support vectors for the soft margin case are determined by the value of the slack variable ξ_n defined in Equation 5.12: data points for which $\xi_n > 0$ are the support vectors. Note that these support vectors are either correctly classified but within/on the margin ($\xi_n \in [0, 1]$) or incorrectly classified ($\xi_n > 1$). We can visualize this in Figure 5.5.

⋆ Data points for which $\xi_n = 0$ are on the margin and are still considered support vectors.

5.4 Conversion to Dual Form

Now that we have the formulation of the optimization problem for SVMs, we need to discuss how we actually go about optimizing to produce a model solution. This will involve converting to a *dual form* of our problem. We will do this in the hard margin case for notational simplicity, but our solution will apply to the soft margin formulation as well.

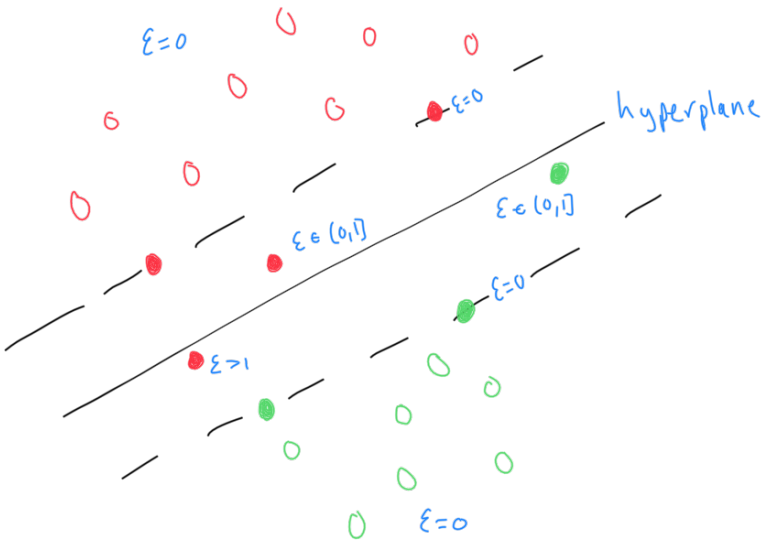

Figure 5.5: Example of the resulting hyperplane for a soft margin SVM. The filled in data points are support vectors.

⋆ A dual form is simply an equivalent manner of representing some expression, in this case the quadratic programming problem we need to optimize.

5.4.1 Lagrange Multipliers

Before we get into deriving the dual form, we need to be aware of a critical piece of math that will enable us to solve our optimization problem: *Langrange multipliers*.

A Lagrange multiplier is used to find optima of a function subject to certain constraints. This is exactly what we need to solve the optimization problem described by Equation 5.11.

The underlying theory behind Lagrange multipliers is not overly difficult to understand, but it is beyond the scope of this textbook. We will simply offer the method by which you can use them to solve optimization problems.

If you have a function $f(\mathbf{x})$ which you need to optimize (let's say maximization here to be concrete, but minimization applies just the same) subject to the constraint that some function $g(\mathbf{x}) = 0$, you can take the following steps. First, construct the Lagrangian function $L(\mathbf{x}, \lambda)$:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x}) \tag{5.14}$$

Then, set the derivative of L with respect to both $\mathbf{x}$ and λ equal to 0:

$$\nabla L_{\mathbf{x}} = 0, \qquad \frac{\partial L}{\partial \lambda} = g(\mathbf{x}) = 0$$

If $\mathbf{x}$ is D-dimensional, this will give you a system of $D+1$ equations. You can solve these equations for $\mathbf{x}$ to find the optimal value of $f(\mathbf{x})$ subject to the constraint $g(\mathbf{x})$.

You should also be aware of the case where your constraint $g(\mathbf{x})$ is an inequality. If we have $g(\mathbf{x}) \geq 0$, we will still have the Lagrangian function given by Equation 5.14. On the other hand, if the inequality constraint is $g(\mathbf{x}) \leq 0$, construct your Lagrangian function as follows:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x})$$

where only the sign has changed between the $f(\mathbf{x})$ and $g(\mathbf{x})$ terms. We will work in the $g(\mathbf{x}) \geq 0$ case, which like before we optimize with respect to the parameters $\mathbf{x}$ and λ:

$$\nabla L_{\mathbf{x}} = 0, \qquad \frac{\partial L}{\partial \lambda} = g(\mathbf{x}) \geq 0$$

Regardless of the direction of the inequality, we also now optimize subject to the constraints:

$$\lambda \geq 0, \qquad \lambda g(\mathbf{x}) = 0$$

⋆ The exact reason for these new constraints when working with inequalities is beyond the scope of this textbook, but it's important to remember that they be met.

Finally, note that when you are optimizing a function under many constraints, you simply introduce a Lagrange multiplier for each constraint.

Example 5.1 (Langrange Multiplier Example): Let's say we had a function $f(\mathbf{x})$ we wish to maximize:

$$f(\mathbf{x}) = x_1^2 + 3x_2^2 + 3$$

subject to the constraint $g(\mathbf{x})$:

$$g(\mathbf{x}) = 4x_1 + 4x_2 - 6 = 0$$

We begin by construction our Lagrangian function $L(\mathbf{x}, \lambda)$:

$$L(\mathbf{x}, \lambda) = x_1^2 + 3x_2^2 + 3 + \lambda(4x_1 + 4x_2 - 6)$$

We then compute $\nabla L_{\mathbf{x}}$ and $\frac{\partial L}{\partial \lambda}$ to get a system of equations:

$$\begin{aligned} 2x_1 + 4\lambda &= 0 \\ 6x_2 + 4\lambda &= 0 \\ 4x_1 + 4x_2 - 6 &= 0 \end{aligned}$$

Solving these equations for x_1, x_2, and λ leaves us with the following mazimized solution:

$$(x_1^*, x_2^*) = \left(\frac{9}{8}, \frac{3}{8}\right); \qquad \lambda^* = -\frac{9}{16}$$

5.4.2 Deriving the Dual Formulation

Now that we understand Lagrange multipliers and how we can use them for constrained optimization, we can get back to the hard margin SVM optimization problem:

$$\arg\min_{\mathbf{w}, w_0} \frac{1}{2}||\mathbf{w}||^2 \quad \text{s.t.} \quad \forall n \; y_n(\mathbf{w}^T\mathbf{x}_n + w_0) \geq 1 \tag{5.15}$$

To satisfy these N constraints, we introduce N Lagrange multipliers $\lambda_0, ..., \lambda_{N-1} \geq 0$. We then have our Lagrangian function:

$$L(\mathbf{w}, w_0, \boldsymbol{\lambda}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{n=1}^{N} \lambda_n (y_n(\mathbf{w}^T \mathbf{x}_n + w_0) - 1) \tag{5.16}$$

where $\boldsymbol{\lambda} = \lambda_0, ..., \lambda_{N-1}$. Using this Lagrangian function, we allow ourselves to switch from solving Equation 5.15 to instead solving:

$$\arg\min_{\mathbf{w}, w_0} \max_{\boldsymbol{\lambda} \geq 0} L(\mathbf{w}, w_0, \boldsymbol{\lambda}) \tag{5.17}$$

⋆ The '$\arg\min_{\mathbf{w}, w_0} \max_{\boldsymbol{\lambda} \geq 0}$' in Equation 5.17 may be initially confusing. We introduce the maximum because the values of $\boldsymbol{\lambda}$ are free and should be set such that $L(\mathbf{w}, w_0, \boldsymbol{\lambda})$ is as large as possible prior to minimizing with respect to $\mathbf{w}, w_0$.

We now wish to convert the objective in Equation 5.17 to a dual objective. Under the sufficient conditions of strong duality which hold for this problem but whose explanation is beyond the scope of this textbook, we can reformulate the objective as:

$$\max_{\boldsymbol{\lambda} \geq 0} \arg\min_{\mathbf{w}, w_0} L(\mathbf{w}, w_0, \boldsymbol{\lambda}) \tag{5.18}$$

At this point, we can solve for $\mathbf{w}, w_0$. Taking the gradient, setting equal to 0, and solving for $\mathbf{w}$:

$$\nabla L_{\mathbf{w}} = \mathbf{w} - \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n = 0$$

$$\mathbf{w}^* = \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n \tag{5.19}$$

And now for w_0:

$$\frac{\partial L}{\partial w_0} = -\sum_{n=1}^{N} \lambda_n y_n = 0$$

$$\sum_{n=1}^{N} \lambda_n y_n = 0 \tag{5.20}$$

Now that we have these equations defining the optimal values for $\mathbf{w}$ and w_0 we plug them back into our our Lagrangian function:

$$\begin{aligned} L(\mathbf{w}, w_0, \boldsymbol{\lambda}) &= \frac{1}{2}||\sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n||^2 - \sum_{n=1}^{N} \lambda_n y_n (\sum_{m=1}^{N} \lambda_m y_m \mathbf{x}_m)^T \mathbf{x}_n - \sum_{n=1}^{N} \lambda_n y_n w_0 + \sum_{n=1}^{N} \lambda_n \\ &= \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \mathbf{x}_n^T \mathbf{x}_m - \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \mathbf{x}_n^T \mathbf{x}_m - w_0 \sum_{n=1}^{N} \lambda_n y_n + \sum_{n=1}^{N} \lambda_n \\ &= \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \mathbf{x}_n^T \mathbf{x}_m \end{aligned} \tag{5.21}$$

We're left with the result of Equation 5.21, which is another quadratic function that we wish to maximize subject to the constraint $\boldsymbol{\lambda} \geq 0$ and $\sum_{n=1}^{N} \lambda_n y_n = 0$. We can solve for $\boldsymbol{\lambda}$ using the same Langrangian techniques described above.

Why do we wish to solve for the values of $\boldsymbol{\lambda}$ at all? To see this, we need to understand how to make predictions on new data points.

5.4.3 Making Predictions

Recall the classification function described by Equation 5.1 at the beginning of this chapter:

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + w_0$$

We classifying new data points by looking at the sign produced by this function. Now that we've solved for $\mathbf{w}$, we can rewrite this function as:

$$f(\mathbf{x}) = \sum_{n=1}^{N} \lambda_n y_n \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_n) + w_0 \tag{5.22}$$

Data points for which $f(\mathbf{x}) > 0$ are classified as 1, while all other data points are classified as -1. Remember also that since $\boldsymbol{\lambda}$ are Lagrange multipliers, we are bound by the constraints:

$$\lambda_n \geq 0, \quad \lambda_n \{y_n f(\mathbf{x}_n) - 1\} = 0$$

which means that for every data point summed in the Equation 5.22, we either have that $\lambda_n = 0$ or $y_n f(\mathbf{x}_n) = 1$. Any data points where $\lambda_n = 0$ don't appear in the summation and thus don't have any impact on the prediction.

The rest of the data points, for which $\lambda_n > 0$, are known as **support vectors**, and because we have that $y_n f(\mathbf{x}_n) = 1$, these are the data points that lie on the margin of the hyperplane in Figure 5.5.

This is a major takeaway for the usefulness of SVMs: once we've trained our model, we can discard most of our data. We only need to keep the support vectors to make predictions. This also illustrates why we need to solve for the values of $\boldsymbol{\lambda}$: those values dictate which data points are the support vectors for our model.

Finally, note that the extension of this solution from the hard margin to the soft margin case is straightforward. In fact, the Lagrangian function ends up being identical and we only have a few extra constraints.

5.4.4 Why is the Dual Formulation Necessary?

At this point, you might be wondering what exactly we gained by moving to the dual formulation of this problem.

First, the complexity of the optimization problem we're solving changed from one that is dependent on the number of features M in our particular data domain to one that is linearly dependent on the number of data points N. Thus, our model complexity is now independent of the dimensionality of the feature space.

Second, in the dual formulation, we have the opportunity to take advantage of what's known as the *kernel trick* to map our data $\mathbf{x}_n$ into higher dimensions without incurring performance costs. This works as follows: notice that during our training process and at prediction time through Equation 5.22, we have the term $\phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m)$. We rewrite this as follows:

$$k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m) \tag{5.23}$$

where $k(\mathbf{x}_n, \mathbf{x}_m)$ is known as the kernel function. Under certain conditions, we can skip the step of projecting data points $\mathbf{x}_n$ into the larger basis defined by the transformation ϕ, instead directly computing the value of the kernel function. We can use this trick as long as $k()$ is a valid kernel, which we turn to next.

⋆ This trick can even be used to work in an infinite basis. If that is of interest, you should look into Taylor series basis expansion.

5.4.5 Kernel Composition

Now that we've seen the usefulness of the kernel trick for working in higher dimensional spaces without incurring performance costs or memory overhead, it's reasonable to wonder what sort of valid kernels we can construct.

To be explicit, by 'kernel' we mean a function that produces a scalar product from two vectors in your desired feature space. In other words, when applied to two data points, your kernel should produce a number. Although it is beyond the scope of this textbook, the condition for a kernel $k(\cdot)$ to be valid is that the matrix of elements given by $k(\mathbf{x}_n, \mathbf{x}_m)$ should be positive semidefinite for all choices of the data set $\{\mathbf{x}_n\}$.

⋆ The matrix of elements $k(\mathbf{x}_n, \mathbf{x}_m)$ is known as the *Gram matrix*, and is denoted by $\mathbf{K}$.

In practice we care that we can compose valid kernels from smaller valid kernels. To this end, there exists a set of rules that will preserve the validity of kernels through transformations. These include such things as scalar multiplication $\big(c \cdot k(\mathbf{x}_n, \mathbf{x}_m)\big)$, exponentiation $\big(\exp\{k(\mathbf{x}_n, \mathbf{x}_m)\}\big)$, and addition $\big(k_1(\mathbf{x}_n, \mathbf{x}_m) + k_2(\mathbf{x}_n, \mathbf{x}_m)\big)$ and multiplication $\big(k_1(\mathbf{x}_n, \mathbf{x}_m) \cdot k_2(\mathbf{x}_n, \mathbf{x}_m)\big)$ of different kernels. It is always possible to test the validity of a given kernel by demonstrating that its Gram matrix $\mathbf{K}$ is positive semidefinite.