Linear Algebra and the Primal Support Vector Machine

Matthew A. Nazari

December 2020

Contents

1 Cover Letter

To this paper's editors,

The original intent of this paper was to provide a comprehensive introduction to a intermediate computer scientist like myself into the field of machine learning and explore the Support Vector Machine as an example. With the advice of Tarun and Dusty I realized this would entail an egregious amount of pages of mostly non-mathematical discussion on the principles behind general machine learning practices and concepts. As evident in my paper's length, I still prioritized offering a thorough and slow introduction to machine learning over brevity. However, the derivation of my paper ends with the primal form of the SVM which is an ideal balance between what this project expected of me and my original intentions. Almost the entirely of my original paper was scraped after the rough draft and by doing so I have achieved a paper which I am proud of.

The paper begins with an introduction for the motivation of an algorithm like the SVM in the machine learning setting. With concepts of linear algebra introduced between the lectures and the problem sets I develop various forms of the primal SVM. By viewing the primal SVM from multiple perspectives (hard margin, soft margin, regularization, etc.), I build intuition for not just the SVM as an algorithm but for the rationale behind modelling decisions and derivations in machine learning I wish I had at the beginning of this project. In this way, I am very proud to have preserved my original intention while fulfilling the expectations of the project.

Sincerely,
Matthew A. Nazari

Name	Age	Weight	Height
Juan	18	162lb	6ft 1in
Joe	34	146lb	5ft 10in
Mei	39	193lb	5ft 8in
Bella	21	124lb	5ft 3in
Abdul	57	141lb	5ft 7in
Hanna	40	134lb	5ft 4in

(a) *Data in raw representation.*

Age	Weight (kg)	Height (cm)
18	73.48	185.42
34	66.22	177.80
39	87.54	172.72
21	56.25	160.02
57	63.96	170.18
40	60.78	162.56

(b) *Data in numerical representation.*

Figure 1: *Data representations; raw data must be given a chosen numerical representations in order to be operated on by an algorithm.*

2 The Primal Support Vector Machine

2.1 An Introduction

It is common for us to employ a machine learning algorithm to predict one of many discrete outcomes. A video streaming platform, for example, in order to abide by FTC guidelines might age-restrict videos they deem inappropriate for younger or more sensitive users. Consequently, this company might want an algorithm to predict whether a video is either "mature" or "family friendly". When there are only two discrete outcomes ("classes"), as in this example, the machine learning task is called binary classification. In this paper we will derive a fantastically competent algorithm called the support vector machine (SVM) designed to perform this exceedingly common machine learning task.

2.1.1 Data Representation in Machine Learning

The idea behind any machine learning algorithm is to reveal patterns and extrapolate information from data[1]. Consider the example data set in Figure 1a. like much of real world data sets each data point or "example" (represented by rows) have non-numerical properties or "features" (represented by columns). Hence, a programmer must chose a numerical representation for the data. With the aid of a domain expert, a chosen numerical representation might look like Figure 1b. Some features like the individual's name may be left out not only for privacy but because that information probably does not lend insight on the task at hand. In this paper, and in machine learning convention, we denote the number of examples in a data set with N and the number of features in each example with D. Therefore we can represent any example in a given data set as the vector $\boldsymbol{x}_n \in \mathbb{R}^D$:

[1]For the scope of this paper we have omitted discussion of basic machine learning vocabulary; all concepts and terminology can be explored further in the *Mathematics for Machine Learning* [1].

Definition 2.1 (Example vector)**.** Given a data set of N data points with D features, the example vector $\boldsymbol{x}_n \in \mathbb{R}^D$ is the vector representation of the n-th data point in the data set, i.e., the d-th entry $\boldsymbol{x}_n^{(d)}$ of $\boldsymbol{x}_n$ is the d-th feature.

With data representation understood we can now use concepts in linear algebra to extract information from the data. For example, measuring the distance between two example vectors is an empirical measurement of the similarity between those individuals.

2.1.2 The SVM as an Algorithm

Recall the task performed by the SVM is binary classification: therefore when presented an example the SVM must predict a class. The two binary classes are referred to as the "positive class" and "negative class" and expressed with the labels $+1$ and -1 respectively. A SVM that, for example, predicts diabetes in patients will output the label $+1$ for a positive diagnosis (positive class) and the label -1 for a negative diagnosis (negative class). This is done with the a predictor function:

Definition 2.2 (SVM predictor)**.** An SVM predictor takes the form of a function which classifies an example $\boldsymbol{x} \in \mathbb{R}^D$ with the label $+1$ or negative -1 corresponding to either the positive or negative binary classes respectively:

$$f : \mathbb{R}^D \to \{+1, -1\}. \tag{1}$$

The predictor function in any machine learning algorithm is a function employed by the algorithm's "model" to perform the task at hand. The model is a mathematical construction that often only differs subtly from the predictor function. In a linear regression [4] setting, the model of the algorithm is a continuous function calculated by fitting certain "model parameters" to a training data set with linear regression. By taking the machine learning approach we know our SVM algorithm will also have model parameters we will need to optimize on a training data set. The process of optimizing our model with training data is referred to as training or "learning" a model.

As in any machine learning task, we would like to estimate parameters of our model given a training data set that minimizes classification error: recalling our previous example, mistakenly classifying a violent or pornographic video as family friendly is an error in the streaming platform's SVM algorithm the company would ideally minimize. The amount of error an algorithm incurs on unseen data like this is called "variance" which is minimized by finding optimal parameters to our model that will fit unseen data accurately. Recalling our familiarity with linear regression, we are also minimizing variance by finding a line of best fit by deriving that line from a given training data set. In a linear regression situation, the model of the algorithm is a continuous function and we train the model by performing linear regression to find the best fit. In a similar way we must train our SVM model which we will see takes the form of solving an optimization problem on our training data. Before we derive what this optimization problem looks like, we must design our model.

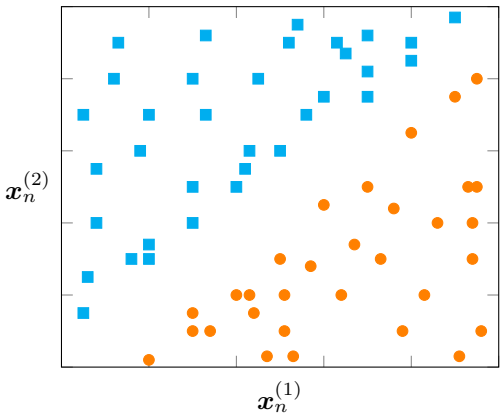


Figure 2: *An example* 2*-dimensional data set. The data binary classes can observably be separated by drawing a line.*

2.2 The SVM Model

2.2.1 The Hyperplane

A training data set for binary classification consists of example-label pairs $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$ where $\boldsymbol{x}_n \in \mathbb{R}^D$ and $y_n \in \{+1, -1\}$. An example of this type of data set is illustrated in Figure 2. Each example $\boldsymbol{x}_n \in \mathbb{R}^2$ is a 2-dimensional location $(\boldsymbol{x}_n^{(1)}, \boldsymbol{x}_n^{(2)})$ and its corresponding binary label y_n is represented as either a cyan square or orange circle. The two classes have examples with features arranged in a way that we can seemingly separate or classify the data by simply drawing a line between the classes. From this observation we choose the model of the SVM (pertaining to the example in Figure 2) to be a line. We describe this line mathematically as an affine subspace of $\mathbb{R}^D$:

Definition 2.3 (Affine subspace)**.** Let V be a vectorspace, U be a subspace of V, and $w \in V$. Then the subset

$$L = \boldsymbol{w} + H = \{\boldsymbol{w} + \boldsymbol{x} : \boldsymbol{x} \in H\} \subseteq V. \tag{2}$$

is an affine subspace of V.

A rudimentary geometric interpretation of the affine subspace $\boldsymbol{w} + H$ is the parallel subspace H pushed away or translated by the geometric vector $\boldsymbol{w}$. Lines and planes in $\mathbb{R}^3$ are examples of affine subspaces: they themselves do not go through the origin but parallel to subspaces that do. But how do we choose the model of a SVM pertaining to data in higher dimensions?

The idea of any classification task is to represent data in $\mathbb{R}^D$ then partition that space, ideally in such a way examples in the same class fall in the same partition. In binary classification, that space is split into two partitions corresponding to the positive and negative classes. Recall that we divide $\mathbb{R}^2$ into

two partitions with a line which is a 1-dimensional affine subspace. We divide $\mathbb{R}^3$ into two partitions with a plane which is a 2-dimensional affine subspace. It turns out to divide any data space $\mathbb{R}^D$ into two partitions, we use a $(D-1)$-dimensional affine subspace. These affine subspaces are called hyperplanes.

Definition 2.4 (Hyperplane)**.** A hyperplane in $\mathbb{R}^D$ is a $(D-1)$-dimensional affine subspace of $\mathbb{R}^D$. Consider a function

$$f : \mathbb{R}^D \to \mathbb{R} \tag{3a}$$

$$\boldsymbol{x} \mapsto f(\boldsymbol{x}) := \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \tag{3b}$$

parametrized by $\boldsymbol{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$. Then $f(x) = c$ is the hyperplane

$$\{\boldsymbol{x} \in \mathbb{R}^D : f(x) = c\}. \tag{4}$$

Recall that lines are hyperplanes in $\mathbb{R}^2$ and planes are hyperplanes in $\mathbb{R}^3$. The abstract nature of how we defined the hyperplane in (4) may obscure how this definition represents both of these objects. Consider the equation for a line $y = mx + b$ in $\mathbb{R}^2$. We can reparametrize this as $0 = w_1 x_1 + w_2 x_2 + b$. Using the Euclidean dot product this simplifies again to $0 = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ where $\boldsymbol{w}, \boldsymbol{x} \in \mathbb{R}^2$. We can do this again with the equation for a plane $z = mx + ny + b$ in $\mathbb{R}^3$. We can reparametrize this as $0 = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$ which is again equivalent to $0 = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ where $\boldsymbol{w}, \boldsymbol{x} \in \mathbb{R}^3$. So although (4) uses an inner product it is the familiar general equation for a hyperplane in $\mathbb{R}^D$.

Finally, we can chose a model for our SVM which divides the data space in two partitions even in dimensions higher than that in Figure 2:

$$\{\boldsymbol{x} \in \mathbb{R}^D : f(x) = 0\}. \tag{5}$$

The model parameters are $\boldsymbol{w} \in \mathbb{R}^D$ (the parameter vector or "weight" vector) and $b \in \mathbb{R}$. The parameter b intuitively affects the distance of the hyperplane to the origin. However, the parameter vector $\boldsymbol{w}$ dictating the direction of the hyperplane is not intuitive:

Proposition 2.1. *The parameter vector $\boldsymbol{w}$ is normal to the hyperplane in* (5)*.*

Proof. Let $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathbb{R}^D$ be arbitrary. Suppose $\boldsymbol{x}_i, \boldsymbol{x}_j$ are on the hyperplane in (5). Consider $\langle \boldsymbol{w}, \boldsymbol{x}_i - \boldsymbol{x}_j \rangle$. By a property of inner products we obtain

$$\begin{aligned} & \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - \langle \boldsymbol{w}, \boldsymbol{x}_j \rangle && \text{(6a)} \\ = \; & \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b - (\langle \boldsymbol{w}, \boldsymbol{x}_j \rangle + b) && \text{(6b)} \\ = \; & f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j). && \text{(6c)} \end{aligned}$$

Since $\boldsymbol{x}_i, \boldsymbol{x}_j$ are on the hyperplane, by (5) we know $f(\boldsymbol{x}_i) = 0$ and $f(\boldsymbol{x}_j) = 0$. Therefore we obtain that $\boldsymbol{w}$ is perpendicular to the vector between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$:

$$\langle \boldsymbol{w}, \boldsymbol{x}_i - \boldsymbol{x}_j \rangle = f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j) = 0. \tag{7}$$

Therefore $\boldsymbol{w}$ is perpendicular to the hyperplane in (5). □

Remark. While $\boldsymbol{x}$ in (5) represents an data point, the parameter vector $\boldsymbol{w}$ is a geometric vector even through $\boldsymbol{w}$ is also a vector in the data space $\mathbb{R}^D$. Imagine $\boldsymbol{w}$ as an arrow pointing in the direction of the hyperplane (perpendicular to the surface).

2.2.2 The Hyperplane as a Linear Classifier

By defining a hyperplane with 4 we effectively defined a direction. The positive direction indicates the positive side of the hyperplane which corresponds to the positive class. Examples that fall on the positive side of the hyperplane are given the label $+1$ corresponding to the positive class. This reasoning applies likewise with the negative case with the label -1. Numerically speaking, given a test example $\boldsymbol{x}$ we classify it as $+1$ if $f(\boldsymbol{x}) \geq 0$ or -1 if otherwise. Consequently, the hyperplane is often referred to as a "linear classifier" separating the positive and negative classes. We train the classifier, i.e., train the model, ensuring the training examples remain on the correct side of the hyperplane:

$$f(\boldsymbol{x}_n) \geq 0 \text{ when } y_n = +1 \tag{8a}$$

$$f(\boldsymbol{x}_n) < 0 \text{ when } y_n = -1. \tag{8b}$$

Conventionally, derivations express these conditions as a single inequality

$$y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geq 0. \tag{9}$$

At this point we pause to consider where we have come. Given linearly separable training data and the task of binary classification we have derived a model (a separating hyperplane) which classifies examples corresponding to the side of the hyperplane it falls in.

2.3 The Hard Margin SVM

Our derivation of the SVM algorithm is yet to be complete: consider, for instance, a linearly separable training data set and the hyperplane that solves the classification problem. As illustrated in Figure 3, there are often infinite many candidate linear classifiers which will classify the training data without incurring any training error or "bias"; hence, we need to further develop our algorithm to systematically find a unique linear classifier. We elaborate on how we choose this unique ideal separator in Section 2.3.1.

Even more concerning model's inability to train on a non-linearly separable data set. Since linearly separable training data is rarely available in the real world, we need to tweak our model by allowing examples to fall on the wrong side of the hyperplane thereby allowing the model to incur training error as explored in Section 2.4.

2.3.1 Maximum Margin Derivation

To find the ideal linear classifier we look for a hyperplane with the largest buffer between it and the nearest example. This buffer is called the margin and

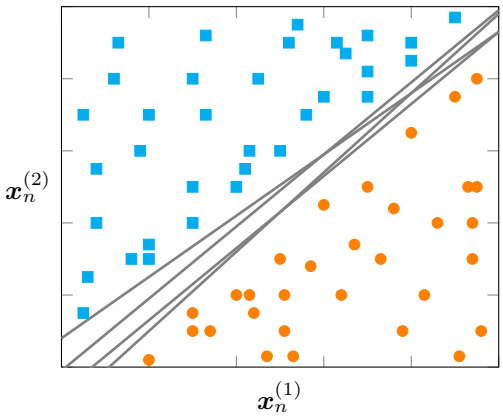


Figure 3: *Candidate linear classifiers. There are more multiple hyperplanes that classify the training data without error.*

is a prevailing concept in not just this paper but in the entire field of machine learning[2]. Imagine a two-way street running between two blocks of houses. The center line which separates the two lanes represents the hyperplane. This center line is painted where the distance between it and the houses on either side is maximized and equivalent. This is so the lanes on either side are at their widest which is ideal for traffic. Similarly, we seek to find the candidate hyperplane with the largest margin in between it and nearest training data. Before rigorously solving for this hyperplane, there is a technical point we need to first consider: the scale to measure distance from the hyperplane. We cannot use the scale of the data because we can change the units of measurement of any feature in $\boldsymbol{x}_n$ and the scale would change thereby changing its distance to the hyperplane. The solution is to instead base the scale on the equation for the hyperplane itself.

Consider an example $\boldsymbol{x}_n \in \mathbb{R}^D$ and the hyperplane $\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b = 0$. Suppose $\boldsymbol{x}_n$ lies on the positive side of the hyperplane and let $r > 0$ be the distance between $\boldsymbol{x}_n$ and the hyperplane. Recall the Orthogonal Decomposition Theorem [4] states the distance between $\boldsymbol{y} \in \mathbb{R}^D$ and a subspace U of $\mathbb{R}^D$ is the distance from $\boldsymbol{y}$ to the orthogonal projection $\pi_U(\boldsymbol{y}_i)$ of $\boldsymbol{y}$ onto U. With this intuition we can find the distance r with the orthogonal projection of $\boldsymbol{x}_n$ onto the hyperplane:

Definition 2.5 (Orthogonal decomposition for affine subspaces)**.** Let $L = \boldsymbol{w} + H$ be an affine subspace of $\mathbb{R}^D$. The vector $\boldsymbol{x} \in \mathbb{R}^D$ can be written uniquely as

$$\boldsymbol{x} = \pi_L(\boldsymbol{x}) + \boldsymbol{z} \tag{10a}$$

$$= \pi_H(\boldsymbol{x} - \boldsymbol{w}) + \boldsymbol{w} + \boldsymbol{z} \tag{10b}$$

[2]Refer to *Learning with Kernels* [6] to explore how concepts like the margin and empirical risk minimization extend into other machine learning algorithms

where $\pi_L(\boldsymbol{x})$ is the orthogonal projection of $\boldsymbol{x}$ onto the affine subspace L.

The projection $\pi_L(\boldsymbol{x})$ is identical to the orthogonal projection of $\boldsymbol{x} - \boldsymbol{w}$ onto H translated back onto L by adding $\boldsymbol{w}$. More importantly, the closest vector to $\boldsymbol{x}$ on L is still the orthogonal projection $\pi_L(\boldsymbol{x})$:

$$\begin{aligned} \operatorname{dist}(\boldsymbol{x}, L) &= \big\|\boldsymbol{x} - \pi_L(\boldsymbol{x})\big\| = \big\|\boldsymbol{x} - (\pi_H(\boldsymbol{x} - \boldsymbol{w}) + \boldsymbol{w})\big\| && \text{(11a)} \\ &= \operatorname{dist}(\boldsymbol{x} - \boldsymbol{w}, \pi_H(\boldsymbol{x} - \boldsymbol{w})) && \text{(11b)} \\ &= \operatorname{dist}(\boldsymbol{x} - \boldsymbol{w}, H) && \text{(11c)} \end{aligned}$$

Recall we are trying to find the distance $r > 0$ from an example $\boldsymbol{x}_n$ to the hyperplane. We do this with the orthogonal projection $\hat{\boldsymbol{x}}_n$ of $\boldsymbol{x}_n$ onto the hyperplane. We know $\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n$ is orthogonal to the hyperplane and hence from Proposition 2.1 a scaling of $\boldsymbol{w}$. By normalizing $\boldsymbol{w}$ then scaling it by the distance r we obtain

$$\boldsymbol{x}_n = \hat{\boldsymbol{x}}_n + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}. \tag{12}$$

We suppose r is our margin, i.e., $\boldsymbol{x}_n$ is the nearest example to the hyperplane; therefore, other examples must be at least a distance r from the hyperplane in the positive or negative direction. This constraint combined with (9) gives the inequality

$$y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geq r. \tag{13}$$

We also suppose $\boldsymbol{w}$ is of unit length using the Euclidean norm, i.e., $\|\boldsymbol{w}\| = \sqrt{\boldsymbol{w}^\intercal \boldsymbol{w}} = 1$. Recall the goal is to maximize the margin r under this new constraint such that each example is distanced r or greater from the hyperplane given $r > 0$ and $\|\boldsymbol{w}\| = 1$. We translate this objective and constraints into the constrained optimization problem

$$\begin{aligned} \max_{\boldsymbol{w}, b, r} \quad & r && \text{(14a)} \\ \text{subject to} \quad & y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geq r, && \text{(14b)} \\ & \|\boldsymbol{w}\| = 1, && \text{(14c)} \\ & r > 0 && \text{(14d)} \\ & \text{for all} \quad n = 1, \ldots, N. && \text{(14e)} \end{aligned}$$

Fortunately, this turns out to be a convex optimization problem for which there exists many competent programming libraries. When handed off to a package, a program will solve for the values of $\boldsymbol{w}, b, r$ such that margin is maximized and the data is lying on the correct side of the hyperplane.

At last we have formalized a rudimentary SVM algorithm able to be performed by a computer. In order to develop this algorithm further, we pause to consider the traditional approach at deriving the margin.

2.3.2 Tradition Margin Derivation

We obtained a constrained optimization problem (14) which solves for the ideal hyperplane that classifies linearly separable data. Our derivation in Section

2.3.1 is not the conventional and often more confusing derivation. In our first derivation we assumed the parameter vector $\boldsymbol{w}$ is of unit length, i.e., $\|\boldsymbol{w}\| = 1$. Traditionally, however, we choose a scale for our data rather than choosing the length of our parameter vector. We choose a scale such that the predictor $y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b)$ has the value 1 at the closest example to the hyperplane $\boldsymbol{x}_a$. Geometrically, we can understand this by imagining two parallel hyperplanes $\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b = -1$ and $\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b = +1$ acting as boundaries which the data must lie outside of. Note that this implies there should be no data in the buffer between boundaries and the hyperplane. By definition since the orthogonal projection $\hat{\boldsymbol{x}}_a$ is on the hyperplane, $\langle \boldsymbol{w}, \hat{\boldsymbol{x}}_a \rangle + b = 0$. This with (12) and the bilinearity of the inner product gives

$$\langle \boldsymbol{w}, \boldsymbol{x}_a - r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \rangle + b = 0 \tag{15a}$$

$$\langle \boldsymbol{w}, \boldsymbol{x}_a \rangle + b - r\frac{\langle \boldsymbol{w}, \boldsymbol{w} \rangle}{\|\boldsymbol{w}\|} = 0 \tag{15b}$$

$$\langle \boldsymbol{w}, \boldsymbol{x}_a \rangle + b = r\|\boldsymbol{w}\| \,. \tag{15c}$$

By our assumption $\langle \boldsymbol{w}, \boldsymbol{x}_a \rangle + b = 1$, we observe the first term in the last line is 1. Therefore we obtain the crucial observation

$$r = \frac{1}{\|\boldsymbol{w}\|}. \tag{16}$$

We combine this with Equation 8 and get the condition

$$y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geq 1. \tag{17}$$

This brings us to a constrained optimization problem similar to (9),

$$\begin{aligned} \max_{\boldsymbol{w},b} \quad & \frac{1}{\|\boldsymbol{w}\|} \\ \text{subject to} \quad & y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geq r \\ & \text{for all} \quad n = 1, \ldots, N. \end{aligned} \tag{18}$$

Maximizing $\frac{1}{\|\boldsymbol{w}\|}$ is observably the same objective as minimizing $\|\boldsymbol{w}\|$. Conventionally $\|\boldsymbol{w}\|$ is squared and multiplied by the constant $\frac{1}{2}$ as to allow for tidier calculations when computing the gradient and using Lagrange multipliers. Note that our objective does not change. With these alterations we arrive at the traditional algorithm for the *hard margin SVM*.

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{subject to} \quad & y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geq r \\ & \text{for all} \quad n = 1, \ldots, N. \end{aligned} \tag{19}$$

Theorem 2.1. *Maximizing the margin r under constraints as in* (14) *is equivalent to scaling the data such that the margin is 1:*

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 && \text{(20a)} \\ \text{subject to} \quad & y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geq r && \text{(20b)} \\ & \text{for all} \quad n = 1, \ldots, N. && \text{(20c)} \end{aligned}$$

Proof. Consider (14). Let $\boldsymbol{w'} = \lambda \boldsymbol{w}$ for some scalar $\lambda \in \mathbb{R}$. Since $\|\boldsymbol{w}\| = 1$ from (14c),

$$\boldsymbol{w} = \frac{\boldsymbol{w'}}{\lambda} = \frac{\boldsymbol{w'}}{\|\boldsymbol{w'}\|}. \tag{21}$$

We reparametrize to obtain

$$\begin{aligned} \max_{\boldsymbol{w'},b,r} \quad & r && \text{(22a)} \\ \text{subject to} \quad & y_n(\langle \frac{\boldsymbol{w'}}{\|\boldsymbol{w'}\|}, \boldsymbol{x}_n \rangle + b) \geq r, && \text{(22b)} \\ & r > 0 && \text{(22c)} \\ & \text{for all} \quad n = 1, \ldots, N. && \text{(22d)} \end{aligned}$$

Let $\boldsymbol{w''} = \frac{\boldsymbol{w'}}{\|\boldsymbol{w'}\| r}$ and $b'' = \frac{b}{r}$. The constraint (22c) states the distance r is positive. Therefore, we divide both sides of (22a) by r and reparametrize to obtain

$$\begin{aligned} \max_{\boldsymbol{w''},b,r} \quad & r && \text{(23a)} \\ \text{subject to} \quad & y_n(\langle \boldsymbol{w''}, \boldsymbol{x}_n \rangle + b'') \geq 1, && \text{(23b)} \\ & r > 0 && \text{(23c)} \\ & \text{for all} \quad n = 1, \ldots, N. && \text{(23d)} \end{aligned}$$

Rearranging $\boldsymbol{w''} = \frac{\boldsymbol{w'}}{\|\boldsymbol{w'}\| r}$ for r gives

$$\left\|\boldsymbol{w''}\right\| = \left\| \frac{\boldsymbol{w'}}{\|\boldsymbol{w'}\| r} \right\| = \frac{1}{r} \left\| \frac{\boldsymbol{w'}}{\|\boldsymbol{w'}\|} \right\| = \frac{1}{r}. \tag{24}$$

The objective does not change if we chose to maximize r^2 instead of r since the square is strictly monotonic for $r > 0$. By substituting we obtain

$$\begin{aligned} \max_{\boldsymbol{w''},b} \quad & \frac{1}{\|\boldsymbol{w''}\|} && \text{(25a)} \\ \text{subject to} \quad & y_n(\langle \boldsymbol{w''}, \boldsymbol{x}_n \rangle + b'') \geq 1, && \text{(25b)} \\ & \text{for all} \quad n = 1, \ldots, N. && \text{(25c)} \end{aligned}$$

Maximizing $\frac{1}{\|\boldsymbol{w''}\|}$ is observably equivalent to minimizing $\frac{1}{2}\left\|\boldsymbol{w''}\right\|$ so we choose this to be the objective, thereby giving (20). □

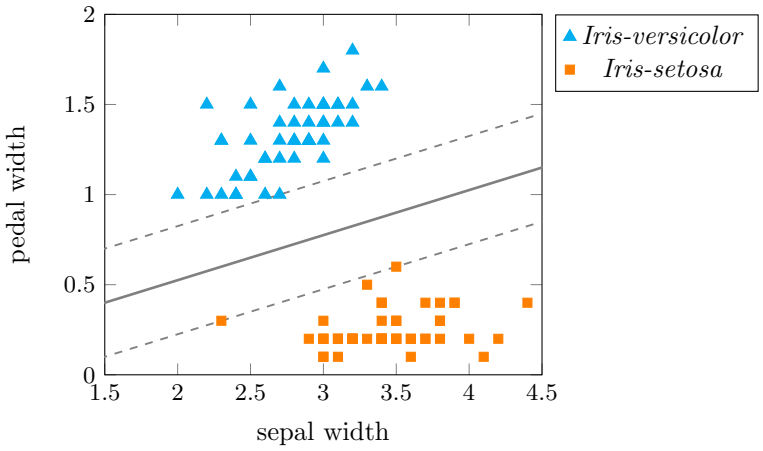


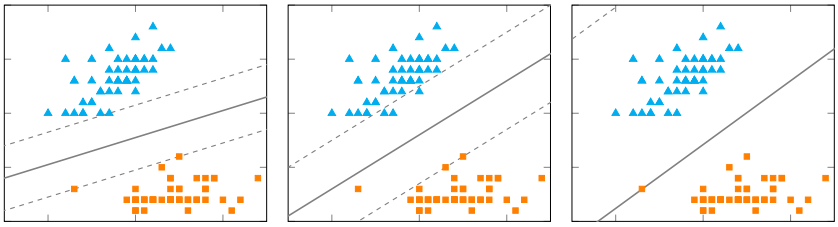
Figure 4: *Fisher's Iris Data Set classified with a hard margin SVM. Two classes of iris flowers and two features.*

Theorem 2.1 gives us the optimization problem of the hard margin SVM (20). The "hard margin" comes from how examples are not allowed to be within the margin or on the wrong side of the hyperplane. With our new algorithm let's perform a test on real world data.

2.3.3 Hard Margin SVM and the Iris Data Set

Take the example of the *Fisher's Iris data set* [2], an exceptionally popular data set in the data science community. It consists of measurements of 150 iris flowers across three species: *Iris-setosa*, *Iris-versicolor*, and *Iris-virginica*. For each individual flower 4 measurements were made: pedal width, pedal length, sepal width, sepal length. Since the hard margin SVM algorithm we have designed only performs binary classification on linearly separable data, we chose two of the three classes (*Iris-setosa*, *Iris-versicolor*) and two of the four features (sepal width, pedal width).

If one were to draw a linear classifier by hand, there would be many to choose from. However, the "best" linear classifier (as we defined in Section 2.3.1) coincides with the linear classifier returned by our SVM. As evident in Figure 4, the SVM returned a linear separator with the largest possible margin. Flowers of the same species fall on the same side of the hyperplane, thus incurring no training error and therefore no bias. We already have an extremely competent algorithm able to optimally linearly separable binary classification data. Unfortunately, until now we have only considered linearly separable data. In most cases the data will not be linearly separable as in Figure4. To adapt our algorithm, we reexamine the margin.



(a) $C = 10^2$; *slack minimization prioritized significantly over margin maximization.*

(b) $C = 10^0$; *slack minimization and margin maximization prioritized evenly.*

(c) $C = 10^{-2}$; *margin maximization prioritized significantly over slack minimization.*

Figure 5: *Fisher's Iris Data Set classified using a soft margin SVM at varying values for the regularization parameter C.*

2.4 The Soft Margin SVM

2.4.1 Slack and Regularization

To adapt the algorithm to be able to train on non-linearly separable data sets, we need to allow some examples to fall within the margin region or even on the wrong side of the hyperplane. We do this by giving each example-label pair $(\boldsymbol{x}_n, y_n)$ a "slack variable" $\xi_n > 0$. When training the model, this slack allows examples to be correctly classified while falling on the wrong side of the margin. We still want to minimize the number of examples which fall on the wrong side of the margin, therefore we aim to minimize the total slack across all examples-label pairs. Combining this additional objective and constraints we obtain the optimization problem

$$\min_{\boldsymbol{w},b,\xi} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{n=1}^{N}\xi_n \tag{26a}$$

$$\text{subject to} \quad y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n\rangle + b) \geq r - \xi_n \tag{26b}$$

$$\xi_n \geq 0 \tag{26c}$$

$$\text{for all} \quad n = 1, \ldots, N. \tag{26d}$$

The SVM with this optimization problem is called the soft margin SVM since examples can cross the margin. The constant $C > 0$ is referred to as the "regularization parameter". The regularization parameter trades off total amount of slack for maximizing the margin. A larger value of C gives a higher priority to minimize slack in the objective: the SVM will emphasize minimizing how many examples fall on the incorrect side of the margin over maximizing the margin. In the real world, the value of C is optimized with cross-validation, a process which estimates the model parameters that produce the lowest variance.

C	bias (%)	variance (%)
10^{-2}	23.72	28.4
10^{-1}	7.88	18.4
10^{0}	1.12	16.8
10^{1}	0.16	19.4
10^{2}	0.08	19.4
10^{3}	0.08	19.4

Figure 6: *Training and test error on different values for the regularization parameter C.*

2.4.2 Soft Margin SVM and the Iris Data Set

Again consider the same *Fisher's Iris data set* as before[3]. We used a hard margin SVM to find the linear classifier in Figure 4. The hard margin SVM is equivalent to the soft margin SVM with infinitely expensive slack, i.e., a soft margin SVM where the regularization parameter is infinitely large. The table in Figure 6 demonstrates how the training error approaches 0% as the regularization parameter grows larger. Different values of the regularization parameter is illustrated in Figure 5. As we can see, with a lower value of the regularization parameter as in Figure 5c means maximizing the margin even if training error rises. The table reveals that the model is optimal, i.e., has the least variance, when $C = 10^0$ (see Figure 5b). By simply introducing the ability for examples to cross the margin, we have derived an algorithm which empirically reduces testing error. This allows us to more confidently employ this algorithm in real world situations.

[3]The axis and legend of Figure 5 are consistent with Figure 4.

3 Conclusion

3.1 Furthur Reading

The soft margin SVM derived in this paper is widely employed in all fields of science including biology (seen with the *Fisher's Iris data set* examples), medicine (in patient diagnoses), and in aerospace engineering for object detection. To read more on the applications of the concepts talk about in the scope of this paper, refer to *Support Vector Machine Applications* [5] which takes a particular emphasis on the SVM in medicine. A natural continuation of this paper is to learn about the Dual SVM; *An Introduction to Support Vector Machines: And Other Kernel-Based Learning Methods* [3] explores the dual form of our SVM algorithm which optimizes performance on more complex data sets sometimes upwards of a million dimensions. A slower and more thorough iteration of this paper's derivations can be found in *Support Vector Machines* [7].

3.2 Bibliography

References

[1] M.P. Deisenroth, A.A. Faisal, and C.S. Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[2] Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017.

[3] Cristianini Nello Shawe-Taylor John. *An Introduction to Support Vector Machines: And Other Kernel-Based Learning Methods*. ITPro collection. Cambridge University Press, 2014.

[4] D.C. Lay, S.R. Lay, and J. McDonald. *Linear Algebra and Its Applications*. Pearson, 2016.

[5] Y. Ma and G. Guo. *Support Vector Machines Applications*. SpringerLink : Bücher. Springer International Publishing, 2014.

[6] B. Schölkopf, B.S.A.J. Smola, A.J. Smola, F. Bach, MIT Press, and M.D.M.P.I.B.C.T.G.P.B. Scholkopf. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive computation and machine learning. MIT Press, 2002.

[7] I. Steinwart and A. Christmann. *Support Vector Machines*. Information Science and Statistics. Springer New York, 2008.