Support vector machines

Jianxin Wu

LAMDA Group
National Key Lab for Novel Software Technology
Nanjing University, China
wujx2001@gmail.com

February 11, 2020

Contents

Support Vector Machines (SVMs) are a family of widely applied classification methods that have exhibited excellent accuracy in various classification problems. Complex mathematics are involved in various aspects of SVMs—e.g., proof of their generalization bounds, their optimization, designing and proving the validity of various non-linear kernels, etc. We will, however, not focus on the mathematics. The main purpose of this chapter is to introduce how the ideas in SVMs are formulated, why are they reasonable, and how various simplifications are useful in shaping the SVM primal form.

We will not touch on any generalization bounds of SVMs, although this is an area that has attracted intensive research efforts. We will not talk about how the optimization problem in SVMs can be solved or approximately solved (for efficiency and scalability). These choices enable us to focus on the key ideas that lead to SVMs and the strategies in SVMs that may be helpful in other domains. We encourage readers to also pay attention to these aspects of SVMs while reading this chapter.

1 The key SVM idea

How shall we accomplish a classification task? In Chapter 4, we discussed the Bayes decision theory, which suggests that we can estimate the probability distribution $\Pr(\boldsymbol{x}, y)$ or the density function $p(\boldsymbol{x}, y)$ using the training data. With these distributions, the Bayes decision theory will guide us on how to classify a new example. These types of probabilistic methods are called generative methods. An alternative type of method is the discriminative method, which directly estimates the probability $\Pr(y|\boldsymbol{x})$ or $p(y|\boldsymbol{x})$.

In SVMs, we do not model either the generative or the discriminative distributions based on the training data. Instead, the SVM seeks to directly find the best classification boundary that divides the domain of $\boldsymbol{x}$ into different regions. Examples fall in one region all belong to one category, and different regions corresponding to different categories or classes.

The rationale behind this choice is: estimation of probability distributions or densities is a difficult task, which might be even more difficult than the classification task itself, especially when there are only a few training examples.[1] In other words, classification based on density estimation might be taking a detour. Then, why not directly estimate the classification boundaries? As illustrated by Figure 2, it might be a much easier task to estimate the boundary in some scenarios.

Then, the natural question is: which classification boundary is considered good or even the best?

[1] We will discuss density estimation in the next chapter.

1.1 Simplify it, simplify it, simplify it!

One key to answering this question is actually to simplify the problem, in order to find out in which scenarios it will be easy to determine a good classification boundary. The heading says "simplify it" three times—this is not only because it is important (so must be repeated three times), but also we are making three simplifying assumptions.

Given datasets (or classification problems) such as those appearing in Figure 1, it is difficult to figure out which classifier is good. In Figure 1a, the two classes (whose examples are denoted by black squares and red circles, respectively) cannot be classified by any linear boundary. Using a complex curve (i.e., a non-linear boundary), we can separate these two classes with zero training error. However, it is not obvious either how to compare two complex non-linear classification boundaries. Hence, we make the first two assumptions:

- *linear boundary* (or, linear classifier); and,
- *separable* (i.e., linear classifiers can achieve 100% accuracy on the training set).

These two are usually combined to require a *linearly separable* problem.

The problem in Figure 1b is in fact linearly separable: the four classes can be divided into non-overlapping regions by linear boundaries—e.g., the three lines that enclose the red class examples. However, it is not easy to judge whether one set of triple-lines is better than another triplet either. And, there are many other possible linear boundaries beyond triplets enclosing red examples. Hence, we make the third assumption:

- The problem is *binary*.

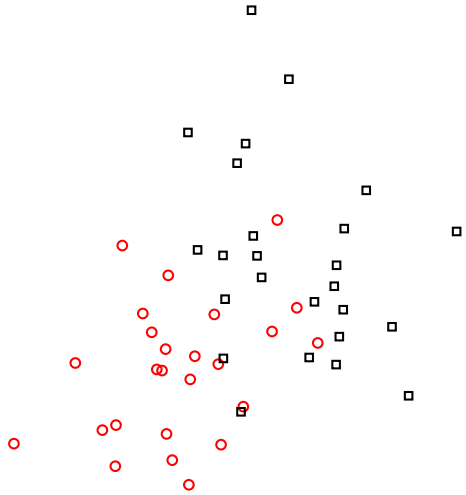
Putting all assumptions together, we start by only considering *linearly separable binary* classification problems.

We want to emphasize that *these simplifications are reasonable*—i.e., they will not change the essence of the classification problem. As we will introduce in this chapter, all these assumptions will be relaxed and taken care of in SVM. Hence, reasonable simplifying assumptions help us find ideas and good solutions to a complex problem, and we have the option to re-consider these assumptions later such that our method has the potential to solve complex tasks, too. However, if an assumption changes some basic characteristics of our problem, it is better avoided.

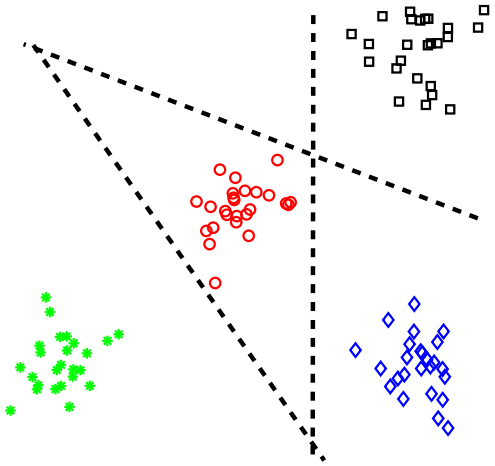
1.2 Finding a max (or large) margin classifier

Figure 2a shows a linearly separable binary problem. It is clear that any line between the cluster of red circles and the cluster of black squares can separate them perfectly. In Figure 2a, three example boundaries are illustrated: the blue solid line, the red dashed line, and the black dotted line.

Furthermore, it becomes easier to determine which classifier is better (or the best) in this linearly separable binary example. Most people will agree that



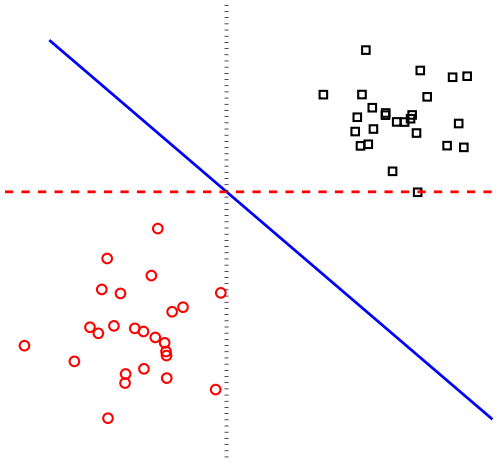
(a) Inseparable data are messy



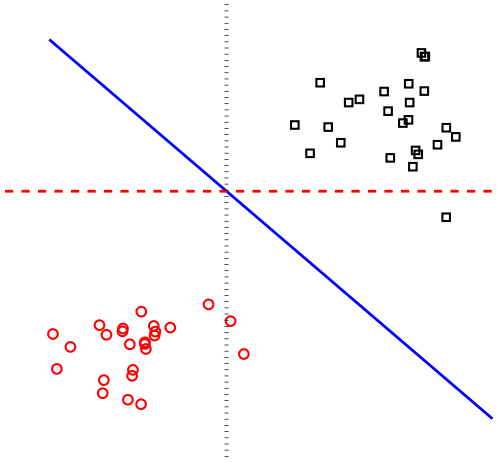
(b) Multiclass data are complex

Figure 1: Illustration of complex classification problems ().

the classification boundary determined by the blue solid line is better than the other two. The other two boundaries are too close to the examples, leaving very small (almost no) room for variations or noise. The examples in Figure 2a and 2b are generated using the same underlying distribution and contain the same number of training examples. However, because of the randomness, two red circles appear on the right side of the black dotted line, and one black square



(a) Large vs. small margin boundaries



(b) A small margin can cause problems

Figure 2: Illustration of the large margin idea. ()

appears below the red dashed line in Figure 2b. These examples are errors for the two boundaries, respectively.

The blue solid line, however, correctly classifies all examples in Figure 2b. The reason for this robustness is: this boundary is far away from all training

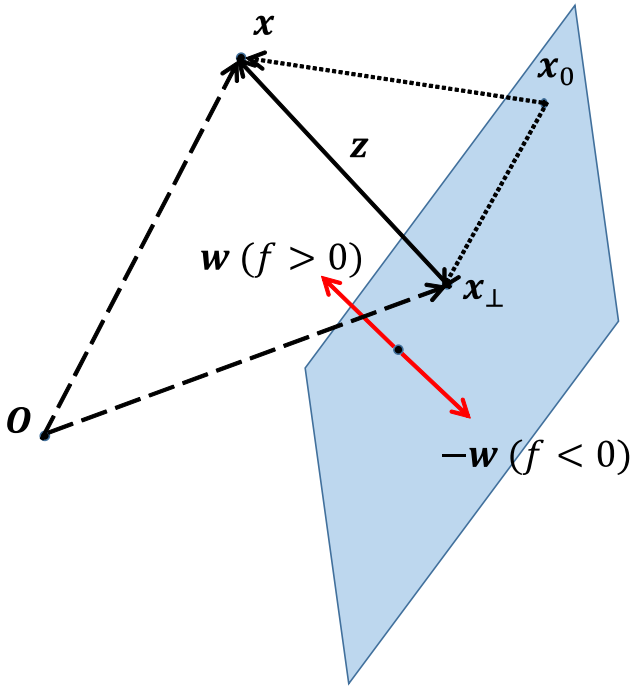


Figure 3: Illustration of projection, margin, and normal vector. ()

examples; hence, when variations or noise are in effect, this classifier has the *margin* to accommodate these changes since the scale of such changes is usually small. As shown in Figure 2, the margin of a classifier is the distance from it to the training examples that are the closest to it.

Hence, the margin of one example (with respect to a classification boundary) is the distance from that point to the boundary. Naturally, the margin of a dataset is the minimum of the margins of all its samples.

Because a large margin is beneficial for classification, some classifiers directly maximize the margin; these are called *max-margin classifiers*. Some classifiers seek a compromise between a large margin and other properties, and are called *large margin classifiers*. The SVM is a max-margin classifier.

And, of course, the next task after having had the max-margin idea is: how shall we translate the max-margin idea into operational procedures? We need to first formalize the max-margin idea.

2 Visualizing and calculating the margin

Given a linear classification boundary and a point $\boldsymbol{x}$, how do we calculate the margin of $\boldsymbol{x}$? We know in a d-dimensional space, a linear classification boundary is a hyperplane. An illustration such as Figure 3 is helpful.

2.1 Visualizing the geometry

We have two ways to specify a vector $\boldsymbol{x}$ in an illustration. One way is to draw an arrow from the origin (O) to the coordinates specified by the elements of $\boldsymbol{x}$. We use this way to draw $\boldsymbol{x}$, $\boldsymbol{x}_\perp$ and $\boldsymbol{x}_0$ in Figure 3 (denoted by dashed arrows, but the arrow for $\boldsymbol{x}_0$ is omitted to preserve the illustration's clarity). Another way is to use two end points to specify a vector, such that the vector is the difference between the two points. For example, $\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{x}_\perp$ (denoted by the black solid arrow) and two vectors involving $\boldsymbol{x}_0$ (denoted by dotted arrows). $\boldsymbol{w}$ and $-\boldsymbol{w}$ are also specified in this way.

All points in a hyperplane in the d-dimensional space are specified by an equation

$$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b = 0\,,$$

where $\boldsymbol{w}$ determines the direction of the hyperplane. The direction of $\boldsymbol{w}$ (i.e., $\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$) is called the normal vector of the hyperplane. A hyperplane is perpendicular to its normal vector. In other words, if $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are two different points on the hyperplane (i.e., satisfying $f(\boldsymbol{x}_1) = f(\boldsymbol{x}_2) = 0$ and $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$), we always have

$$\boldsymbol{w}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0\,.$$

If $\boldsymbol{x}$ is on the hyperplane, then $f(\boldsymbol{x}) = 0$; if $\boldsymbol{x}$ is not on the hyperplane, but in the same direction of $\boldsymbol{w}$ with respect to the hyperplane, then $f(\boldsymbol{x}) > 0$; and, $f(\boldsymbol{x}) < 0$ holds for points in the opposite direction (cf. the red solid arrows).

In Figure 3, $\boldsymbol{x}_\perp$ is the projection of $\boldsymbol{x}$ onto the hyperplane. That is, $\boldsymbol{x}$ is decomposed as

$$\boldsymbol{x} = \boldsymbol{x}_\perp + \boldsymbol{z}\,.$$

Given any point $\boldsymbol{x}_0 (\neq \boldsymbol{x}_\perp)$ on the hyperplane, we always have

$$\boldsymbol{z} \perp (\boldsymbol{x}_\perp - \boldsymbol{x}_0)\,.$$

However, $\boldsymbol{x}_\perp$ is not perpendicular to $\boldsymbol{z}$ if $b \neq 0$. This geometry tells us that $\|\boldsymbol{z}\|$ is the distance we are looking for. Hence, a visualization with proper notations helps us translate the description "distance" into precise mathematics.

2.2 Calculating the margin as an optimization

What is even better, the visualization also hints at how to calculate the distance. Because $\boldsymbol{z} \perp (\boldsymbol{x}_\perp - \boldsymbol{x}_0)$, we know $\|\boldsymbol{z}\| \leq \|\boldsymbol{x} - \boldsymbol{x}_0\|$ for any $\boldsymbol{x}_0$ on the hyperplane, because of the Pythagorean theorem, which also holds in a space $\mathbb{R}^d$ whose dimensionality is higher than 2 ($d > 2$). That is, $\boldsymbol{x}_\perp$ is the solution of the following optimization problem:

$$\arg\min_{\boldsymbol{y}} \quad \|\boldsymbol{x} - \boldsymbol{y}\|^2 \tag{1}$$

$$\text{s.t.} \quad f(\boldsymbol{y}) = 0\,. \tag{2}$$

This is an optimization problem with one equality constraint. The Lagrangian is

$$L(\boldsymbol{y}, \lambda) = \|\boldsymbol{x} - \boldsymbol{y}\|^2 - \lambda(\boldsymbol{w}^T\boldsymbol{y} + b)\,.$$

Setting $\frac{\partial L}{\partial \boldsymbol{y}} = 0$ gives $2(\boldsymbol{y} - \boldsymbol{x}) = \lambda\boldsymbol{w}$. That is,

$$\boldsymbol{y} = \boldsymbol{x} + \frac{\lambda}{2}\boldsymbol{w}\,.$$

Plugging it into $f(\boldsymbol{y}) = 0$, we get $\frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w} + f(\boldsymbol{x}) = 0$, hence

$$\lambda = -\frac{2f(\boldsymbol{x})}{\boldsymbol{w}^T\boldsymbol{w}}\,.$$

Then, the projection is

$$\boldsymbol{x}_\perp = \boldsymbol{x} + \frac{\lambda}{2}\boldsymbol{w} = \boldsymbol{x} - \frac{f(\boldsymbol{x})}{\boldsymbol{w}^T\boldsymbol{w}}\boldsymbol{w}\,.$$

Now, it is easy to obtain

$$\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{x}_\perp = \frac{f(\boldsymbol{x})}{\boldsymbol{w}^T\boldsymbol{w}}\boldsymbol{w}\,,$$

hence, the distance (and the margin of $\boldsymbol{x}$) is

$$\left\|\frac{f(\boldsymbol{x})}{\boldsymbol{w}^T\boldsymbol{w}}\boldsymbol{w}\right\| = \left|\frac{f(\boldsymbol{x})}{\|\boldsymbol{w}\|^2}\right|\|\boldsymbol{w}\| = \frac{|f(\boldsymbol{x})|}{\|\boldsymbol{w}\|}\,. \tag{3}$$

There are simpler ways to find the margin than what is presented here. However, in this section we familiarized ourselves with the geometry useful for SVM, found the actual value of $\boldsymbol{x}_\perp$, exercised the process of formalization and optimization, and applied the method of Lagrange multipliers. These are all fruitful outcomes.

3 Maximizing the margin

The SVM tries to maximize the margin of the dataset—i.e., the smallest margin of all training points.

3.1 The formalization

Let us denote the training set as $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is an example, and $y \in \mathcal{Y}$. For a binary problem, $\mathcal{Y} = \{1, 2\}$. To maximize the margin of this dataset with respect to a linear boundary $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$ means to solve the following optimization problem:

$$\max_{\boldsymbol{w}, b} \min_{1 \le i \le n} \frac{|f(\boldsymbol{x}_i)|}{\|\boldsymbol{w}\|}\,, \tag{4}$$

with the additional constraint that $f(\boldsymbol{x})$ can classify all training examples correctly. Hence, we also need to mathematically describe "correctly classified."

This is quite simple if we change the definition of $\mathcal{Y}$ slightly. If we set $\mathcal{Y} = \{+1, -1\}$, the meaning of it will not change—the two numbers refer to the labels of the two classes, and the specific values of them are not relevant (e.g., 2 vs. -1), so long as the two values are different. However, with this slight change, the following statement holds:

$$\begin{aligned} &f(\boldsymbol{x}) \text{ correctly classifies all training examples} \\ &\text{iff } y_i f(\boldsymbol{x}_i) > 0 \text{ for all } 1 \le i \le n, \end{aligned}$$

in which "iff" means if and only if. When $y_i = 1$ (i.e., a positive example), $y_i f(\boldsymbol{x}_i) > 0$ means $f(\boldsymbol{x}_i) > 0$, thus the prediction is also positive; When $y_i = -1$, $f(\boldsymbol{x}_i) < 0$ and it predicts $\boldsymbol{x}_i$ as negative.

The SVM then optimizes the following problem for a binary linearly separable dataset:

$$\max_{\boldsymbol{w},b} \quad \min_{1\le i\le n} \frac{|f(\boldsymbol{x}_i)|}{\|\boldsymbol{w}\|} \tag{5}$$

$$\text{s.t.} \quad y_i f(\boldsymbol{x}_i) > 0, \qquad 1 \le i \le n\,. \tag{6}$$

3.2 The simplifications

The constraints can be handled by a Lagrangian. However, the objective contains a fraction, an absolute value, a vector norm and the maximization of a minimum, all of which are hostile to optimization. Fortunately, by looking at this objective more carefully, there exist ways to avoid all these difficulties.

Our assumption ensures that $y_i f(\boldsymbol{x}_i) > 0$ for all $1 \le i \le n$—that is, we always have

$$y_i f(\boldsymbol{x}_i) = |f(\boldsymbol{x}_i)|$$

because $|y_i| = 1$. Hence, the objective can be rewritten as

$$\min_{1\le i\le n} \frac{|f(\boldsymbol{x}_i)|}{\|\boldsymbol{w}\|} = \min_{1\le i\le n} \frac{y_i f(\boldsymbol{x}_i)}{\|\boldsymbol{w}\|} \tag{7}$$

$$= \frac{1}{\|\boldsymbol{w}\|} \min_{1\le i\le n} \left(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)\right)\,. \tag{8}$$

This type of objective (as a ratio) has appeared many times (e.g., in PCA). If $(\boldsymbol{w}^\star, b^\star)$ is a maximizer of the above objective, so will $(c\boldsymbol{w}^\star, cb^\star)$ be for any nonzero constant $c \in \mathbb{R}$. However, since this is a constrained optimization problem, $c \le 0$ will make all the constraints invalid. When $c > 0$, $(c\boldsymbol{w}^\star, cb^\star)$ will not change the objective, and all constraints remain satisfied. That is, we have the freedom to choose any $c > 0$. In the past, we have chosen a c such that $\|\boldsymbol{w}\| = 1$. If we use this same assumption, the optimization becomes

$$\max_{\boldsymbol{w},b} \quad \min_{1\le i\le n} y_i f(\boldsymbol{x}_i) \tag{9}$$

$$\text{s.t.} \quad y_i f(\boldsymbol{x}_i) > 0, \qquad 1 \le i \le n \tag{10}$$

$$\boldsymbol{w}^T \boldsymbol{w} = 1\,. \tag{11}$$

The objective is still a maximization of a minimum. And, this optimization problem is still difficult to solve.

However, note that $y_i f(\boldsymbol{x}_i)$ appears in both the objective and the constraints; there is a clever trick to further simplify it. If $(\boldsymbol{w}^\star, b^\star)$ is a maximizer of the original objective $\min_{1\le i\le n} \frac{|f(\boldsymbol{x}_i)|}{\|\boldsymbol{w}\|}$, we choose

$$c = \min_{1\le i\le n} y_i((\boldsymbol{w}^\star)^T \boldsymbol{x} + b^\star)\,.$$

Obviously $c > 0$, hence $\frac{1}{c}(\boldsymbol{w}^\star, b^\star)$ is also a maximizer of the original problem.

Let us consider this particular choice of c, which leads to

$$\min_{1\le i\le n} y_i \left(\left(\frac{1}{c}\boldsymbol{w}^\star \right)^T \boldsymbol{x} + \frac{1}{c} b^\star \right) = 1 > 0\,.$$

Hence, we can add the following constraint to our optimization problem without changing the optimal objective value:

$$\min_{1\le i\le n} y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) = 1\,, \tag{12}$$

or, equivalently,

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \ge 1 (> 0)\,, \tag{13}$$

for all $1 \le i \le n$—which means all constraints in the original problem are automatically satisfied!

To be precise, $\min_{1\le i\le n} y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) = 1$ is not equivalent to $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \ge 1$ for all i. When $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \ge 1$, it is possible that $\min_{1\le i\le n} y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) > 1$, for example, it is possible that $\min_{1\le i\le n} y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) = 2$. However, since in an SVM the objective is to minimize $\frac{1}{2}\boldsymbol{w}^T \boldsymbol{w}$, the constraints $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \ge 1$ altogether imply $\min_{1\le i\le n} y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) = 1$ in the optimal solution. The proof of this fact is pretty simple, and is left to the reader.

In other words, we can convert the original problem into the following equivalent one:

$$\max_{\boldsymbol{w}, b} \quad \frac{\min_{1\le i\le n} y_i(\boldsymbol{w}^T \boldsymbol{x} + b)}{\|\boldsymbol{w}\|} = \frac{1}{\|\boldsymbol{w}\|} \tag{14}$$

$$\text{s.t.} \quad y_i f(\boldsymbol{x}_i) \ge 1, \qquad 1 \le i \le n\,. \tag{15}$$

A final step is to change the maximization of $\frac{1}{\|\boldsymbol{w}\|}$ to the minimization of $\|\boldsymbol{w}\|$ and furthermore (equivalently) to the minimization of $\boldsymbol{w}^T \boldsymbol{w}$ and $\frac{1}{2}\boldsymbol{w}^T \boldsymbol{w}$, which leads to

$$\min_{\boldsymbol{w}, b} \quad \frac{1}{2}\boldsymbol{w}^T \boldsymbol{w} \tag{16}$$

$$\text{s.t.} \quad y_i f(\boldsymbol{x}_i) \geq 1, \qquad 1 \leq i \leq n\,. \tag{17}$$

The additional term $\frac{1}{2}$ will make later derivations easier.

In this series of transformations and simplifications, we get a series of *equivalent* problems and hence will get the same optimal objective value as that of the original problem. The new constraints are very similar to the original ones (and are handled by the method of Lagrange multipliers in the same manner). However, the objective is now a quadratic form ($\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$), and very amenable to optimization.

We will work (a little bit) on the optimization of Equations 16–17, and they are called the primal form of the SVM formulation, or simply a primal SVM.

4 The optimization and the solution

We use the method of Lagrange multipliers to handle the n inequality constraints and state the necessary and sufficient conditions for the minimizers of the primal SVM. The proof is beyond the scope of this book and is thus omitted. With these conditions, the primal form turns into an equivalent dual form.

4.1 The Lagrangian and the KKT conditions

We still define one Lagrange multiplier α_i for each constraint, and the constraints $y_i f(\boldsymbol{x}_i) \geq 1$ are rewritten as $y_i f(\boldsymbol{x}_i) - 1 \geq 0$. The Lagrangian is

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - \sum_{i=1}^{n} \alpha_i \left(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1\right) \tag{18}$$

$$\text{s.t.} \quad \alpha_i \geq 0, \quad 1 \leq i \leq n\,, \tag{19}$$

in which

$$\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)^T$$

is the vector of Lagrange multipliers.

For inequality constraints, however, the multipliers are not free any more—they are required to be non-negative. If one constraint is violated—i.e., if $y_i f(\boldsymbol{x}_i) - 1 < 0$ for some i—we have $-\alpha_i \left(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1\right) > 0$, meaning a punishment term is added to $L(\boldsymbol{w}, b, \boldsymbol{\alpha})$. Hence, setting $\alpha_i \geq 0$ for all i is somehow enforcing the i-th constraint. Also, if

$$\alpha_i \left(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1\right) = 0 \tag{20}$$

for all $1 \leq i \leq n$, then $L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$ matches the primal objective. Later, we will show the optimality conditions indeed specifies $\alpha_i \left(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1\right) = 0$ for all $1 \leq i \leq n$.

As usual, we compute the gradients and set them to $\mathbf{0}$ or 0, respectively:

$$\frac{\partial L}{\partial w} = \mathbf{0} \quad \Longrightarrow \quad \boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i\,, \tag{21}$$

$$\frac{\partial L}{\partial b} = 0 \quad \Longrightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0\,. \tag{22}$$

The above three equality conditions, the original inequality constraints, and the constraints on the Lagrange multipliers form the Karush–Kuhn–Tucker (KKT) conditions for the primal SVM optimization problem:

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i \tag{23}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{24}$$

$$\alpha_i \left(y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1 \right) = 0 \qquad i = 1, 2, \ldots, n \tag{25}$$

$$\alpha_i \geq 0 \qquad i = 1, 2, \ldots, n \tag{26}$$

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 \qquad i = 1, 2, \ldots, n\,. \tag{27}$$

These KKT conditions are not necessary and sufficient in general. However, for the primal SVM problem, these conditions are both necessary and sufficient to specify optimal solutions.

Equation 23 states that the optimal solution $\boldsymbol{w}$ can be represented by a weighted combination of the training examples; the signs of the weights are determined by the labels of the examples, and the magnitudes of these weights are the Lagrange multipliers. In more general large margin learning, the *representer theorem* ensures this weighted average representation is still valid in many other situations–e.g., in kernel SVMs.

Equation 24 states that these weights are *balanced* in the two classes. If we sum the Lagrange multipliers for all positive training examples, this must equal the sum of weights for all negative training examples. Because these Lagrange multipliers can be considered as weights for positive and negative examples, one way to interpret Equation 24 is: an SVM has an internal mechanism to balance weights between the positive and negative examples, which might be useful in handling imbalanced datasets so long as the level of imbalance is not large.

Equation 25 is called the *complementary slackness* property, which is important in SVMs, and we will soon discuss this property in more depth. The other two conditions are the non-negative constraints for the Lagrange multipliers and the original constraints.

4.2 The dual SVM formulation

Equations 23 and 24 allow us to remove the original parameters $(\boldsymbol{w}, b)$ from the Lagrangian. Note that

$$\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j\,, \tag{28}$$

$$\sum_{i=1}^{n} \alpha_i y_i \boldsymbol{w}^T \boldsymbol{x}_i = \boldsymbol{w}^T \boldsymbol{w} = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \,, \tag{29}$$

$$\sum_{i=1}^{n} \alpha_i y_i b = \left(\sum_{i=1}^{n} \alpha_i y_i \right) b = 0 \,, \tag{30}$$

when the KKT conditions hold. Hence, the Lagrangian becomes

$$-\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j + \sum_{i=1}^{n} \alpha_i \,, \tag{31}$$

which does not involve $\boldsymbol{w}$ or b anymore. Putting the constraints on α_i in, we get

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \tag{32}$$

$$\text{s.t.} \quad \alpha_i \geq 0, \quad i = 1, 2, \ldots, n \,, \tag{33}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \,. \tag{34}$$

which is called the *dual SVM formulation*, or simply *the dual*.

In more detail, the above choices for $\boldsymbol{w}$ and b lead to

$$g(\boldsymbol{\alpha}) = \inf_{(\boldsymbol{w}, b)} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) \,,$$

in which inf means the infimum (i.e., greatest lower bound). g is called the Lagrange dual function, and it is always concave. The *maximization* of g with respect to $\boldsymbol{\alpha}$ is always smaller than or equal to the minimization of the original (primal) problem, and the difference between them is called the duality gap. When the duality gap is 0, we can maximize the dual instead of solving the original minimization problem.

One notable difference between the dual and the primal SVM is: in the dual, the training data never appear alone (as in the primal), they *always appear in a pair as a dot-product* $\boldsymbol{x}_i^T \boldsymbol{x}_j$.

In general, the optimal objective value of the primal and dual form of an optimization problem are not equal (i.e., there exists a duality gap). In SVMs, the duality gap is 0—the dual and the primal forms will lead to the same optimal value. We will not talk about the optimization techniques in this chapter, which are beyond the scope of this book. However, many high quality optimization toolboxes (including those specifically designed for SVMs) are available to solve either the primal or the dual SVM optimization.

One can solve the SVM optimization using the primal form and obtain $(\boldsymbol{w}^\star, b^\star)$ directly; one can also solve the dual form and obtain the optimal Lagrange multipliers $\boldsymbol{\alpha}^\star$, and Equation 23 will give us the optimal $\boldsymbol{w}^\star$.

4.3 The optimal b value and support vectors

To obtain the optimal value for b is a little bit more tricky in the dual form, which hinges on the complementary slackness property:

$$\alpha_i \left(y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1 \right) = 0, \qquad i = 1, 2, \ldots, n\,. \tag{35}$$

Based on complementary slackness, if we find one constraint whose corresponding Lagrange multiplier is positive (i.e., $\alpha_i > 0$), then we must have $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1 = 0$. Hence,

$$b^\star = y_i - (\boldsymbol{w}^\star)^T \boldsymbol{x}_i\,. \tag{36}$$

for this particular i. This solution is easy to derive. Because

$$y_i b = 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i \tag{37}$$

when $\alpha_i > 0$, and note that $y_i^2 = 1$, we can multiply both sides of Equation 37 by y_i and obtain Equation 36.

The optimal b value obtained in this manner may not be very reliable because some numerical errors can reduce its accuracy. Hence, we can also find all those examples whose corresponding α_i is non-zero, compute b from them individually, and then use their average as $b^\star$.

Those examples that are useful for computing $b^\star$ are special in the training set, whose corresponding Lagrange multipliers are positive. They are called the *support vectors*, and consequently the classification method is called the support vector machine.

When $\alpha_i > 0$, we know $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1 = 0$, which means the i-th constraint has to be active—i.e., the margin of the training example $\boldsymbol{x}_i$ must be 1, as shown in Figure 4. In Figure 4, examples in two classes are denoted by blue circles and orange squares, respectively; the black line is the classification boundary $f(\boldsymbol{x}) = 0$, while the red lines correspond to $f(\boldsymbol{x}) = \pm 1$—i.e., all examples whose margin $y_i f(\boldsymbol{x}_i)$ is 1 (remember in the SVM formulation we assume $y_i = \pm 1$ and the smallest margin of all training examples is 1). The three examples (two filled circles and one filled square) are the support vectors, whose corresponding $\alpha_i > 0$ and $y_i f(\boldsymbol{x}_i) = 1$.

The classification boundary $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$ is a line in linear SVMs. Hence, we can explicitly compute $\boldsymbol{w} = \sum_{i=1}^n \alpha_i y_i \boldsymbol{x}_i \in \mathbb{R}^d$, and consequently it is very efficient to compute the decision value $f(\boldsymbol{x})$, which only requires one dot-product computation. The linear SVM is hence both accurate and efficient. However, the derivation of decision values for more complex SVM classifiers—e.g., non-linear SVMs—requires a lot of computations.

The prediction (after the decision value $f(\boldsymbol{x})$ is computed) is a simple task. As illustrated in Figure 4, the prediction is simply $\text{sign}(f(\boldsymbol{x}))$, in which sign is the sign function:

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \end{cases}. \tag{38}$$

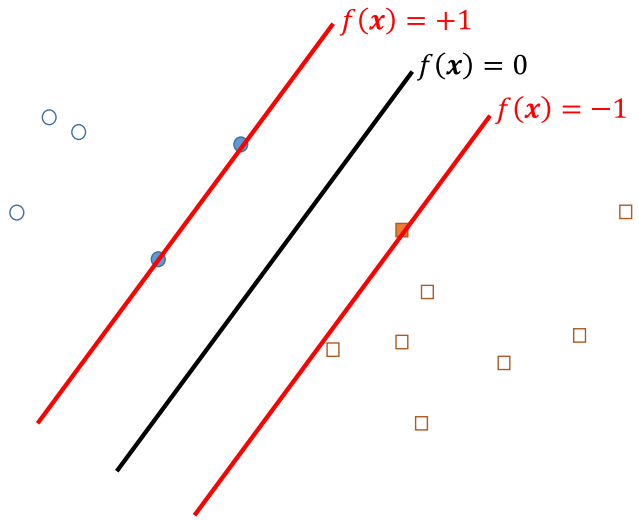


Figure 4: Illustration of support vectors. ()

When a test example $\boldsymbol{x}$ happens to have $f(\boldsymbol{x}) = 0$, the above sign function returns 0, which is not suitable for our classification problem. We can output a prediction of either $+1$ or -1 in an ad-hoc manner; we can also randomly assign a label for these special (and rare) testing examples.

4.4 Looking at the primal and dual simultaneously

Since the primal and dual give the same answer in SVMs, we can look at them simultaneously. After the optimal primal and dual variables are obtained, they guarantee the margin of all training examples ≥ 1, that is, the distance between any positive and negative training example is ≥ 2. This observation provides a lower bound of the distance between examples from different classes.

The objective $\boldsymbol{w}^T\boldsymbol{w}$ is equal to $\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j$. The primal and dual parameters in this section refer to the optimal ones. However, we omit the * superscript to make the notations simpler. Let us define some more notations:

$$\alpha_+ = \sum_{y_i=+1} \alpha_i \,, \tag{39}$$

$$\alpha_- = \sum_{y_i=-1} \alpha_i \,, \tag{40}$$

$$\boldsymbol{s}_+ = \sum_{y_i=+1} \alpha_i \boldsymbol{x}_i \,, \tag{41}$$

$$\boldsymbol{s}_- = \sum_{y_i=-1} \alpha_i \boldsymbol{x}_i \,, \tag{42}$$

are the sums of Lagrange multipliers (weights) for all positive examples and neg-

ative examples and the weighted sums of all positive examples and all negative examples, respectively.

Equation 24 leads to

$$\alpha_+ = \alpha_- \,,$$

and Equation 23 shows

$$\boldsymbol{w} = \boldsymbol{s}_+ - \boldsymbol{s}_- \,.$$

Hence, the primal objective is to minimize $\|\boldsymbol{s}_+ - \boldsymbol{s}_-\|^2$, which is also equivalent to

$$\left\| \frac{\boldsymbol{s}_+}{\alpha_+} - \frac{\boldsymbol{s}_-}{\alpha_-} \right\| \,. \tag{43}$$

$\frac{\boldsymbol{s}_+}{\alpha_+}$ ($\frac{\boldsymbol{s}_-}{\alpha_-}$) is the weighted average of all positive (negative) examples. Hence, the distance between them must be ≥ 2, and the equality can only happen if $\alpha_i = 0$ for all training examples whose margin is > 1.

If any training example $\boldsymbol{x}_i$ has a margin $y_i f(\boldsymbol{x}_i) > 1$ and $\alpha_i > 0$, then $\frac{\boldsymbol{s}_+}{\alpha_+}$ or $\frac{\boldsymbol{s}_-}{\alpha_-}$ will not reside on the two lines whose margin is 1 (cf. the red lines in Figure 4) and the distance between them will be > 2.

Hence, in the linearly separable case, all support vectors reside on these two lines and are sparse (i.e., the number of support vectors is only a small fraction of the number of all training examples, as shown in Figure 4).

5 Extensions for linearly inseparable and multi-class problems

We have made three assumptions in the above derivation: linear classifier, separable, and binary problems. These assumptions (and restrictions) have to be relaxed now that we already have a solution for the simplified problem. In this section, we deal with the non-separable case (still with linear classifiers), and the extension to multiclass problems. Nonlinear classifiers will be discussed in the next section.

5.1 Linear classifiers for non-separable problems

In some binary problems, the training set is not linearly separable—i.e., there is no single line that can perfectly separate examples in the two classes. However, as in the example of Figure 1a, a linear classifier still seems the best model we can find using our eyes and brains. In other words, the positive and negative examples are roughly linearly separable.

In fact, the two classes in Figure 1a are generated by two Gaussians with the same covariance matrix and equal prior, and the optimal classification boundary (in the minimum cost sense) is indeed a linear one. This statement will become obvious after we discuss probabilistic methods in the next chapter.

The linear SVM classifier can be extended to handle such roughly linearly separable problems for real-world problems, using a trick called slack variables.

The current constraint, $y_i f(\boldsymbol{x}_i) \geq 1$, means that not only the i-th example is correctly classified; its distance to the linear boundary is also relatively far (> 1), leaving room to accommodate possible variations in test examples.

However, must we strictly maintain this constraint? Maybe we can achieve a much larger margin of the dataset[2] at the cost of only slightly violating this single constraint (e.g., $y_i f(\boldsymbol{x}_i) = 0.9 < 1$.) So long as $y_i f(\boldsymbol{x}_i) > 0$ (e.g., 0.9), this particular training example $\boldsymbol{x}_i$ is still correctly classified, but larger margins are made possible for other examples.

At another extreme, there may be one outlier $\boldsymbol{x}_i$ in the training set, e.g., $y_i = +1$, but it is in a cluster of negative examples. If we want to strictly maintain the constraint associated with $\boldsymbol{x}_i$, we must pay the price that the problem is not solvable or the margin of other examples are greatly reduced. We need a mechanism to allow this outlier to be wrongly classified—i.e., to allow $y_i f(\boldsymbol{x}_i) \leq 0$ in a few cases.

These exceptions (either $0 < y_i f(\boldsymbol{x}_i) < 1$ or $y_i f(\boldsymbol{x}_i) \leq 0$), however, cannot happen too frequently. We also need a mechanism to specify that the total price (cost) we pay for such relatively rare or extreme cases is small. In SVMs, a slack variable ξ_i is introduced as the price we pay for $\boldsymbol{x}_i$, and its associated constraint is changed to two constraints:

$$y_i f(\boldsymbol{x}_i) \geq 1 - \xi_i \,, \tag{44}$$

$$\xi_i \geq 0 \,. \tag{45}$$

There can be three possible cases for ξ_i:

- When $\xi_i = 0$ for an $\boldsymbol{x}_i$, the original constraint is maintained, and no price is paid;
- When $0 < \xi_i \leq 1$, this example is still correctly classified, but the margin is $1 - \xi_i < 1$, hence the price is ξ_i.
- When $\xi_i > 1$, this example is wrongly classified, and the price is still ξ_i.

Hence, the total price is

$$\sum_{i=1}^{n} \xi_i \,.$$

To require the total price to be small, we just add this term to the (minimization) objective as a *regularizer*, and the primal linear SVM formulation becomes:

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{n} \xi_i \tag{46}$$

$$\text{s.t.} \quad y_i f(\boldsymbol{x}_i) \geq 1 - \xi_i, \qquad 1 \leq i \leq n \,, \tag{47}$$

$$\xi_i \geq 0, \qquad 1 \leq i \leq n \,. \tag{48}$$

[2] In the SVM formulation, this is equivalent to reducing $\boldsymbol{w}^T\boldsymbol{w}$ by a large amount while fixing the margin to 1.

In the above SVM formulation, a new symbol C is introduced. $C > 0$ is a scalar that determines the relative importance between large margin (minimizing $\boldsymbol{w}^T\boldsymbol{w}$) and paying small total price ($\sum_{i=1}^n \xi_i$). When C is large, the small total price part is more important; and when $C \to \infty$, the margin requirement is completely ignored. When $C < 1$, the large margin requirement is emphasized; and when $C \to 0$, no price is paid at all. Hence, in problems showing different properties, C values have to be tuned to seek the best accuracy.

Using almost the same procedure as in the linearly separable case, we obtain the new dual form

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j \tag{49}$$

$$\text{s.t.} \quad C \geq \alpha_i \geq 0, \quad i = 1, 2, \ldots, n\,, \tag{50}$$

$$\sum_{i=1}^n \alpha_i y_i = 0\,. \tag{51}$$

Note that the only difference between this formulation and the one for linearly separable SVMs is in the constraints on dual variables: $\alpha_i \geq 0$ is changed to $0 \leq \alpha_i \leq C$, which hardly increases the difficulty of solving the dual problem, but the capacity of our SVM formulation has been greatly extended.

We can also view the new objective from a different perspective. The total cost $\sum_{i=1}^n \xi_i$ can be seen as minimizing the cost over the training set—i.e., the empirical cost (or empirical loss). The term $\boldsymbol{w}^T\boldsymbol{w}$ can alternatively been viewed as a regularization term, which encourages linear classification boundaries that are less complex because the large components in $\boldsymbol{w}$ allow larger variations in it and are hence more complex.

Since $\xi_i \geq 0$ and $\xi_i \geq 1 - y_i f(\boldsymbol{x}_i)$ (based on Equations 44 and 45), we can combine them as

$$\xi_i = (1 - y_i f(\boldsymbol{x}_i))_+\,, \tag{52}$$

in which x_+ is called the *hinge loss* for x; $x_+ = x$ if $x \geq 0$, and $x_+ = 0$ if $x < 0$. It is obvious that

$$x_+ = \max(0, x)\,.$$

The hinge loss incurs no penalty if $y_i f(\boldsymbol{x})_i \geq 1$—i.e., when the margin for $\boldsymbol{x}_i$ is larger than 1. Only when the margin is small ($0 < y_i f(\boldsymbol{x}_i) < 1$) or $\boldsymbol{x}_i$ is wrongly classified ($y_i f(\boldsymbol{x}_i) < 0$) is the loss $1 - y_i f(\boldsymbol{x}_i)$ incurred. Using the hinge loss, we can rewrite the primal SVM as

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^n \xi_i\,, \quad \text{or} \tag{53}$$

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^n (1 - y_i f(\boldsymbol{x}_i))_+\,. \tag{54}$$

In this formulation, the constraints are implicitly encoded in the hinge loss and do not appear as constraints any more.

One practical question is: how to find the optimal C value for this tradeoff parameter? We will leave this to later sections.

5.2 Multiclass SVMs

There are two main strategies to handle multiclass problems in the SVM literature: one is to combine multiple binary SVMs to solve the multiclass one; the other is to extend the max-margin idea and its mathematics into multiclass using similar techniques, then solve the multiclass problem using a single optimization. We will introduce the first strategy. In fact, this strategy is not only valid for SVM classifiers, but also effective for extending many binary classification methods to the multiclass case.

The One-vs.-Rest (or one-versus-all, or OvA) method trains m binary SVM classifiers for an m-class problem. In training the i-th binary SVM, examples belonging to the i-th class are used as the positive class examples in the binary problem, and all other training examples are used as the negative class examples. Let us denote this binary SVM classifier as $f_i(\boldsymbol{x})$. In the binary case, when an example is far away from the classification boundary (i.e., with a large margin), there is good reason to believe it will be more tolerant of variations in the distribution of examples. In other words, we are more *confident* about our prediction if it has a large margin.

In the multiclass case, if $f_i(\boldsymbol{x})$ is large (and positive), we can also say with high confidence that $\boldsymbol{x}$ belongs to the i-th class. Hence, the m binary classifiers $f_i(\boldsymbol{x})$ $(1 \leq i \leq m)$ give us m confidence scores, and our prediction for $\boldsymbol{x}$ is simply the class having the highest confidence. That is, the predicted class for a test example $\boldsymbol{x}$ is

$$\arg\max_{1 \leq i \leq m} f_i(\boldsymbol{x}) \tag{55}$$

in the OvA method.

The One-vs.-One (or one-versus-one, or OvO) method trains $\binom{m}{2} = \frac{m(m-1)}{2}$ binary SVM classifiers. For any pair (i, j) satisfying $1 \leq i < j \leq m$, we can use the examples in the i-th class as positive ones, and those in the j-th class as negative ones; all other training examples are ignored for this pair. The trained classifier $f_{i,j}(\boldsymbol{x})$ (after the sign function) determines whether $\boldsymbol{x}$ should be assigned to the i- or j-th class, although it is possible that $\boldsymbol{x}$ is neither in the i- nor the j-th class; and we say the i- or j-th class receives one vote.

For any test example $\boldsymbol{x}$, it will receive $\binom{m}{2}$ votes, distributed among the m classes. If the groundtruth class for $\boldsymbol{x}$ is k $(1 \leq k \leq m)$, we expect the k-th class will receive the largest amount of votes. Hence, in the OvO method, the prediction for any example $\boldsymbol{x}$ is the class that receives the largest number of votes.

There have been empirical comparisons between the OvA and OvO methods. In terms of classification accuracy, there is not an obvious winner. Hence, both methods are applicable when m is not large.

However, for problems with a large number of classes—e.g., when $m = 1000$—the number of binary SVM classifiers to be trained in OvO is $\binom{m}{2}$ (499,500

if $m = 1000$). To train a large number of classifiers requires an excessively long time, and to store and apply all these trained models is also prohibitively expensive. Hence, the OvA method is more popular when m is large.

6 Kernel SVMs

The final major missing block is a nonlinear SVM in which the classification boundary is made up not of lines but of complex curves such as those in Figure 5. Figure 5a shows a problem with 200 training examples while Figure 5b has 2000. The blue hyperbolic curves are the groundtruth decision boundary that separates the two classes, and the green curves are decision boundaries constructed by an SVM using the nonlinear RBF kernel, which we will explain in this section. As shown in these figures, the kernel (or nonlinear) SVM can approximate the groundtruth decision boundary well, especially in regions where many training examples are available.

Kernel methods are a class of methods that are popular in nonlinear learning, and SVMs are a typical example of kernel methods.

6.1 The kernel trick

First, let us consider and reexamine the second term in the SVM dual objective,

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \,.$$

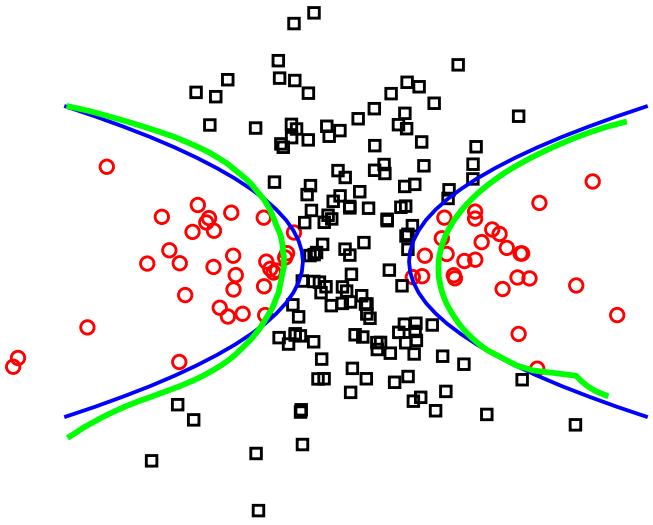
Inside the summations is an inner product between $\alpha_i y_i \boldsymbol{x}_i$ and $\alpha_j y_j \boldsymbol{x}_j$—i.e., inner product between two training examples whose sign is determined by the labels and the weights by the dual variables.

It is well known that the dot-product between two vectors can be treated as a measure of their similarity.[3] If both vectors are unit ones, the dot-product equals the cosine of the angle between these two vectors. A small dot-product value means the angle is close to 90° and the two vectors are far away from each other; a large positive dot-product result means the angle is close to 0 and the two vectors are close to each other; and, a large negative dot-product means the angle is close to 180° and the two vectors are almost the farthest apart possible on a unit hypersphere.

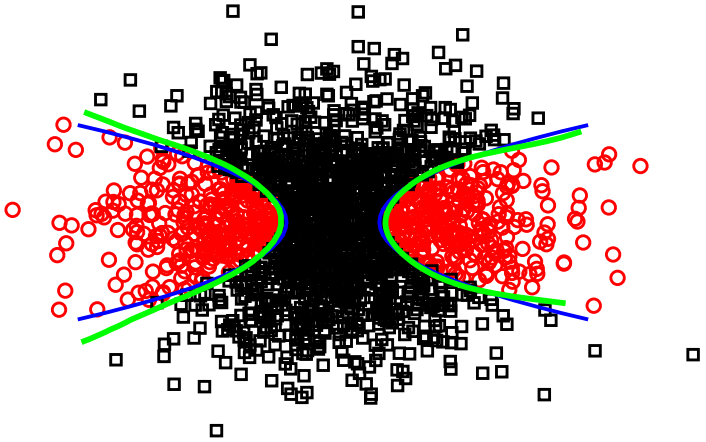
Hence, it is natural to guess: *if we replace the dot-product with some other nonlinear similarity metrics, can we get a nonlinear boundary better than a line?* This guess seems viable because in the dual SVM formulation, as we have observed, the training examples always appear as dot-products (or loosely speaking, always as similarity comparisons!)

Some theories (unfortunately beyond the scope of this book) endorse the above guess, but the nonlinear similarity measure must satisfy a certain condition (called Mercer's condition). Let κ be a nonlinear function satisfying

[3] We will discuss similarity and dissimilarity metrics in more detail in Chapter 9.



(a) 200 training examples



(b) 2000 training examples

Figure 5: Illustration of non-linear classifiers. ()

Mercer's condition; then the nonlinear dual SVM formulation is

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{56}$$

$$\text{s.t.} \quad C \geq \alpha_i \geq 0, \quad i = 1, 2, \ldots, n\,, \tag{57}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0\,. \tag{58}$$

The optimization is the same as in the linear case, just replacing the (constant) values $\boldsymbol{x}_i^T \boldsymbol{x}_j$ by $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

6.2 Mercer's condition and feature mapping

To introduce Mercer's condition, we need to define the quadratically integrable (or square integrable) function concept. A function $g : \mathbb{R}^d \mapsto \mathbb{R}$ is square integrable if

$$\int_{-\infty}^{\infty} g^2(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} < \infty\,. \tag{59}$$

A function $\kappa(\cdot,\cdot) : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ satisfies Mercer's condition if for *any* square integrable function $g(\boldsymbol{x})$, the following inequality is always true:

$$\iint \kappa(\boldsymbol{x},\boldsymbol{y}) g(\boldsymbol{x}) g(\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y} \geq 0\,, \tag{60}$$

and it is symmetric, i.e.,

$$\kappa(\boldsymbol{x},\boldsymbol{y}) = \kappa(\boldsymbol{y},\boldsymbol{x})\,.$$

In the context of SVMs, Mercer's condition translates to another way to check whether κ is a valid kernel (i.e., meets Mercer's condition or not). For a symmetric function $\kappa(\cdot,\cdot)$ and a set of examples $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$, one can define a matrix $K \in \mathbb{R}^n \times \mathbb{R}^n$, with

$$[K]_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

for $1 \leq i, j \leq n$. If the matrix K is positive semi-definite for an *arbitrary* integer $n > 0$ and an *arbitrary* set of examples $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, then κ is a valid kernel function. When κ is a valid kernel, K is called the kernel matrix (induced by κ).

When κ is a valid kernel, then there exists a mapping $\phi : \mathbb{R}^d \mapsto \mathcal{X}$, which maps the input $\boldsymbol{x}$ from the input space to a *feature space* $\mathcal{X}$ and satisfies

$$\kappa(\boldsymbol{x},\boldsymbol{y}) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{y}) \tag{61}$$

for *any* $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$. Although in many cases we do not know how to compute this mapping associated with κ, its existence has been rigorously proved. When we cannot explicitly spell out ϕ, we call it an implicit mapping.

The feature space $\mathcal{X}$ is usually high dimensional. Its dimensionality is often much larger than d, and for many kernels is infinite! Although we skip the advanced concepts (e.g., reproducing kernel Hilbert space), we want to assure the reader that so long as an infinite dimensional space satisfies certain conditions (such as the mappings in kernel SVMs), the inner product between any two vectors in it is well-defined.

Conceptually, the kernel method has *turned a nonlinear classification problem (in the input space) into an equivalent linear classification problem in the (usually much higher dimensional) feature space*, as shown by the dual formulation for kernel SVMs. In the feature space, the linear boundary enjoys the same max-margin benefit. Fortunately, because of the kernel trick, we do not need to compute the dot-product explicitly: $\kappa(\boldsymbol{x}, \boldsymbol{y})$ replaces $\phi(\boldsymbol{x})^T \phi(\boldsymbol{y})$.

The representer theorem once again gives us

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \phi(\boldsymbol{x}_i) \,.$$

However, $\boldsymbol{w}$ can be infinite dimensional, too. Hence, the prediction is performed through the kernel trick, as

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x}) + b \tag{62}$$

$$= \sum_{i=1}^{n} \alpha_i y_i \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}) + b \tag{63}$$

$$= \sum_{i=1}^{n} \alpha_i y_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}) + b \,. \tag{64}$$

This computation, however, is much more expensive than that in linear SVMs. Assuming the complexity of $\kappa(\boldsymbol{x}_i, \boldsymbol{x})$ is $\mathcal{O}(d)$, predicting one example may take $\mathcal{O}(nd)$ steps. When the number of training examples is large (e.g., $n = 1,000,000$), kernel SVM prediction is very slow, and we need to store all training examples in the SVM model, which incurs very high storage costs.

The actual SVM prediction cost, however, is lower than $\mathcal{O}(nd)$. If a training example $\boldsymbol{x}_i$ is not a support vector, then its Lagrange multiplier α_i is 0, and is useless in the above summation. Hence, *only support vectors are used in prediction and stored in the SVM model.*

The Lagrange multipliers are sparse—i.e., many α_i are zero. Let n' denote the number of support vectors, usually $n' \ll n$, and the kernel SVM prediction complexity $\mathcal{O}(n'd)$ is much lower than nd.

6.3 Popular kernels and the hyper-parameters

Commonly used kernel functions include

$$\text{Linear:} \quad \kappa(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{y} \,, \tag{65}$$

$$\text{RBF:} \quad \kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{y}\|^2) \,, \tag{66}$$

$$\text{Polynomial:} \quad \kappa(\boldsymbol{x}, \boldsymbol{y}) = (\gamma \boldsymbol{x}^T \boldsymbol{y} + c)^D \,. \tag{67}$$

The RBF (radial basic function) kernel is sometimes called the Gaussian kernel. Some of these kernels are parameterized—e.g., the RBF kernel has a parameter $\gamma > 0$, and the polynomial kernel has three parameters γ, c, and D (D is a positive integer, called the degree of the polynomial).

We can examine a special case of the polynomial kernel when $\gamma = 1$, $c = 1$, and $D = 2$. With these parameters, the polynomial kernel is

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = (1 + \boldsymbol{x}^T \boldsymbol{y})^2 .$$

If we assume the examples are in the two dimensional space—i.e., $\boldsymbol{x} = (x_1, x_2)^T$, $\boldsymbol{y} = (y_1, y_2)^T$—then

$$\begin{aligned}
\kappa(\boldsymbol{x}, \boldsymbol{y}) &= (1 + x_1 y_1 + x_2 y_2)^2 && (68)\\
&= 1 + 2x_1 y_1 + 2x_2 y_2 + x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 x_2 y_1 y_2 && (69)\\
&= \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ x_1^2 \\ x_2^2 \\ \sqrt{2}x_1 x_2 \end{bmatrix}^T \begin{bmatrix} 1 \\ \sqrt{2}y_1 \\ \sqrt{2}y_2 \\ y_1^2 \\ y_2^2 \\ \sqrt{2}y_1 y_2 \end{bmatrix} && (70)\\
&= \phi(\boldsymbol{x})^T \phi(\boldsymbol{y}) . && (71)
\end{aligned}$$

Hence, for the degree 2 polynomial kernel, we can explicitly write down its mapping from the input space to the feature space, as

$$\phi(\boldsymbol{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2)^T .$$

The feature space is six-dimensional in this particular example, which is three times that of the input space.

The tradeoff parameter C in the SVM formulation and the kernel parameters γ, c, and D are hyperparameters. Note that different hyperparameters lead to different kernels and different SVM solutions, even if the same training set is used!

However, there is very little theoretical work to guide us on how to choose hyperparameters. In practice, the most commonly used method is to use cross-validation to choose them. For example, we can try a few candidate values for C in the set

$$\{2^{-10}, 2^{-8}, 2^{-6}, 2^{-4}, 2^{-2}, 2^0, 2^2, 2^4, 2^6, 2^8, 2^{10}\} .$$

Using the training set, we can obtain the cross-validation error rate *on the training set* using each of these values. The C value that achieves the smallest cross-validation error rate is chosen.

This strategy, however, is very time consuming. Assuming an m class problem and a K-fold cross validation; $K \times \binom{m}{2}$ binary SVM classifiers are required to obtain a single cross-validation error rate if the OvO method is used. If there are two hyperparameters (e.g., C and γ if the RBF kernel is used), and K_1 and K_2 candidate values exist for them, respectively, then the total number of binary SVMs to train is $KK_1K_2\binom{m}{2}$, which is very large.

6.4 SVM complexity, tradeoff, and more

In the early days, the popular SVM solver was the SMO (sequential minimal optimization) method, whose complexity is rather difficult to precisely estimate. Empirical evaluations usually show complexity higher than $\mathcal{O}(n^{2.5}d)$, where n and d are the number and dimensionality of training examples, respectively. This complexity is too high for large scale problems.

For linear SVMs, however, many fast methods have been proposed, which can complete the learning in $\mathcal{O}(nd)$ steps, using techniques such as column generation, stochastic gradient descent, and coordinate descent. These new methods enable the training of linear SVMs for large problems—e.g., with millions of examples and/or millions of dimensions.

Although there are also acceleration techniques for nonlinear SVM learning, its training complexity is still much higher than that of linear SVMs. The same also applies to the testing complexity. Linear SVM testing only requires a single dot-product in a binary problem, while nonlinear SVM testing is much slower.

Nonlinear SVMs, if they can be trained in viable time for a problem, can usually achieve higher (and significantly higher in some problems) accuracy than linear ones. Hence, there is a tradeoff between the accuracy and complexity of SVM classifiers. Nonlinear SVMs are a good choice when the dataset is small or medium sized, and linear SVMs are a good option for large problems, particularly when there are many feature dimensions.

For some special kernels, such as the power mean kernels, algorithms specifically designed for them can obtain both fast speed and highly accurate models.

High quality SVM solvers for both general nonlinear and fast linear kernels are available. Examples include the LIBSVM package (for general nonlinear SVMs) and LIBLINEAR (for fast linear SVMs). We will discuss more details about these topics in the exercise problems.

SVMs can also be extended to handle regression problems, called support vector regression (SVR), which is also implemented in many software packages, including both LIBSVM and LIBLINEAR.

Exercises

1. LIBSVM is a widely used software package for learning kernel SVM classifiers and regressors. In this problem, we will experiment with this software. LIBSVM also has an accompanying page that collects datasets.

   (a) Download the LIBSVM software package from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`. Read the instructions and compile the software from the source code.

   (b) Download the `svmguide1` dataset from the LIBSVM datasets page (`https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`).

   For each of the following setups, train SVM model(s) using the training set, and compare the accuracy on the test set under different setups.

   i. Use the default parameters (i.e., $C = 1$ and using the RBF kernel);

   ii. Use the `svm-scale` utility to normalize the features. Make sure your scaling operation is *proper*: use the parameters from the training set to normalize the test set data. Read the instructions for this utility program carefully;

   iii. Use the linear kernel instead of the RBF kernel;

   iv. Use $C = 1000$ and the RBF kernel;

   v. Use the `easy.py` utility to determine hyperparameters C and γ in the RBF kernel.

   What have you learned from these experiments?

   (c) Play with the datasets in the LIBSVM datasets page. Can you find an imbalanced dataset? The LIBSVM parameter `-wi` is useful in handling imbalanced data. Try this parameter for the imbalanced dataset you have found. Is it helpful?

2. (Additive kernels) A kernel is called additive if it has the following form:

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{d} \kappa(x_i, y_i)$$

   for two d-dimensional vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. Note that we used the same symbol κ to denote the kernel for comparing two vectors and two scalars.

   (a) The *histogram intersection kernel* is a widely used additive kernel, which is defined *for vectors with only non-negative values*, as

$$\kappa_{\mathrm{HI}}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{d} \min(x_i, y_i)\,.$$

Show that if $\kappa_{\text{HI}}(x, y) = \min(x, y)$ for non-negative scalars x and y is a valid kernel, then κ_{HI} is a valid kernel for non-negative vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

(b) Now we consider κ_{HI} for scalars. For *any* set of non-negative scalar values $x_1, x_2, \ldots, x_n$, the kernel matrix formed by these values is

$$[X]_{ij} = \min(x_i, x_j)$$

for $1 \leq i, j \leq n$, in which n is an arbitrary positive integer. Let $y_1, y_2, \ldots, y_n$ be a *permutation* of $x_1, x_2, \ldots, x_n$ such that $y_1 \leq y_2 \leq \cdots \leq y_n$, and similarly define an $n \times n$ matrix Y with

$$y_{ij} = \min(y_i, y_j) = y_{\min(i,j)} \,.$$

Prove that X is positive (semi-)definite if and only if Y is positive (semi-)definite.

(c) For any set of values $0 \leq x_1 \leq x_2 \leq \cdots \leq x_n$, prove that the kernel matrix X with

$$x_{ij} = \min(x_i, x_j) = x_{\min(i,j)}$$

is positive semi-definite. (Hint: what is the LDL factorization of X?)

Note that combining the above results have proved that κ_{HI} is a valid kernel for non-negative vectors.

(d) The χ^2 *kernel* (Chi-square kernel) is another widely used additive kernel defined for positive data, which is defined as

$$\kappa_{\chi^2}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{d} \frac{2x_i y_i}{x_i + y_i} \,.$$

For positive data, this kernel is positive definite (see the next problem for a proof). Prove that

$$\kappa_{\text{HI}}(\boldsymbol{x}, \boldsymbol{y}) \leq \kappa_{\chi^2}(\boldsymbol{x}, \boldsymbol{y}) \,.$$

(e) *Hellinger's kernel* is defined for non-negative data, as

$$\kappa_{\text{HE}}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{d} \sqrt{x_i y_i}.$$

Prove that this is a valid kernel and that

$$\kappa_{\text{HE}}(\boldsymbol{x}, \boldsymbol{y}) \geq \kappa_{\chi^2}(\boldsymbol{x}, \boldsymbol{y})$$

for positive data.

(f) Additive kernels are particularly effective when the feature vector is a *histogram*. A histogram (which is not normalized) contains natural numbers (i.e., zero or positive integers). Write out an explicit mapping for the histogram intersection kernel when the features are all natural numbers.

3. (Power mean kernels) In this problem, we introduce the *power mean kernels*, which are a family of additive kernels and are closely related to the *generalized mean* in mathematics.

   (a) Read the information in the page https://en.wikipedia.org/wiki/Generalized_mean. When we only consider the power mean of two *positive* numbers x and y, what are $M_0(x, y)$, $M_{-1}(x, y)$ and $M_{-\infty}(x, y)$?

   (b) The power mean kernels are a family of additive kernels, each indexed by a *negative or zero* real number p. For two positive vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ with $x_i, y_i > 0$ $(1 \le i \le d)$, the power mean kernel M_p is defined as

$$M_p(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{d} M_p(x_i, y_i) = \sum_{i=1}^{d} \left(\frac{x^p + y^p}{2} \right)^{1/p} .$$

   Power mean kernels were proposed by J. Wu in a paper titled *Power Mean SVMs for Large Scale Visual Classification*, CVPR 2012. Show that the three additive kernels discussed in the previous problem (histogram intersection, χ^2 and Hellinger's) are all special cases of the power mean kernels.

   (c) (Conditionally positive definite kernel) A function κ is called a *conditionally positive definite kernel* if for an arbitrary positive integer n and any $c_1, c_2, \ldots, c_n \in \mathbb{R}$ that satisfies

$$\sum_{i=1}^{n} c_i = 0 \,,$$

   the inequality

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \kappa(x_i, x_j) \ge 0$$

   holds for *arbitrary* $x_1, x_2, \ldots, x_n \in \mathbb{R}$. Prove that

$$-\frac{x^p + y^p}{2}$$

   is conditionally positive definite for positive data.

   (d) Use the following theorem to prove that power mean kernels are positive definite kernels when $-\infty \le p \le 0$—i.e., they satisfy Mercer's condition.

   If a kernel κ is conditionally positive definite and negative valued, then $\frac{1}{(-\kappa)^\delta}$ is positive definite for all $\delta \ge 0$.

   (e) When we use the power mean kernel $M_{-\infty}$ to replace the histogram intersection kernel in a programming environment, we have to use a p value that is far away from the zero value to replace $-\infty$. For example, we can use $p = -32$—i.e., using M_{-32} to approximate $M_{-\infty}$ / κ_{HI}.

Write a simple Matlab or Octave program to evaluate the largest absolute error and relative error between $M_{-32}(x, y)$ and $M_{-\infty}(x, y)$, where x and y are generated by the command `x=0.01:0.01:1` and `y=0.01:0.01:1`, respectively. Is M_{-32} a good approximation for κ_{HI}? That is, compare $\kappa_{\text{HI}}(x, y)$ and $M_{-32}(x, y)$ for the 10,000 pairs of x and y values.

(f) To illustrate the effectiveness of power mean kernels, visit https://sites.google.com/site/wujx2001/home/power-mean-svm and read the instructions there for installation and usage instructions. Use the resources on this page to generate training and testing data for the Caltech 101 dataset. When you generate the data, use `K = 100` rather than `K = 2000`. Try different p values, and apply the PmSVM software to this dataset. Use the LIBSVM software and the RBF kernel on the same dataset. Which one has higher accuracy on this dataset (whose feature vectors are histograms)? Note that if you use the tool provided in LIBSVM to choose optimal C and gamma hyper-parameter values, it will take a very long time to finish. Which software has higher training and testing speed?

4. (SVMs without bias) In the linear SVM formulation, we assume a classification boundary of the form $\boldsymbol{w}^T\boldsymbol{x} + b$, which includes a bias term $b \in \mathbb{R}$. However, it is also possible to learn a linear SVM classifier *without* the bias term—i.e., using $\boldsymbol{w}^T\boldsymbol{x}$ as the classification boundary.

   (a) Without the bias term, what is the optimization problem in the primal space? Use the notations in this chapter.

   (b) Without the bias term, show that the dual form is as follows.

$$\min_{\boldsymbol{\alpha}} \quad f(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j - \sum_{i=1}^{n}\alpha_i \tag{72}$$

$$\text{s.t.} \quad 0 \le \alpha_i \le C, \quad i = 1, 2, \ldots, n\,. \tag{73}$$

   If the solution for $\boldsymbol{\alpha}$ is $\boldsymbol{\alpha}^\star$, how do you find the optimal decision boundary $\boldsymbol{w}^\star$?

   (c) When a bias is preferred, this formulation (without bias) can still be useful. Given a training dataset $(\boldsymbol{x}_i, y_i)$ $(1 \le i \le n)$, one can convert any $\boldsymbol{x} \in \mathbb{R}^d$ into $\hat{\boldsymbol{x}}$ in $\mathbb{R}^{d+1}$ by adding an extra dimension to $\boldsymbol{x}$. The added dimension always has constant value 1. Suppose $\boldsymbol{\alpha}^\star$ is the optimal solution for the dual formulation, and suppose the classification boundary is $\boldsymbol{w}^T\boldsymbol{x} + b$; what is the optimal b value?

5. (Dual coordinate descent) In this problem, we introduce the *dual coordinate descent* algorithm for solving a linear SVM without the bias term in the dual space.

   (a) Using the notations in the previous problem, find $\frac{\partial f}{\partial \alpha_i}$. Find a way to calculate $\frac{\partial f}{\partial \alpha_i}$ in $\mathcal{O}(d)$ steps. We use $f'(\boldsymbol{\alpha})$ to denote the partial derivative

of f with respect to $\boldsymbol{\alpha}$—i.e., $f'(\boldsymbol{\alpha})_i = \frac{\partial f}{\partial \alpha_i}$.

(b) The dual coordinate descent (DCD) algorithm was proposed by Hsieh et al. in a paper titled *A Dual Coordinate Descent Method for Large-scale Linear SVMs*, which was published in ICML 2008. Among the n Lagrange multipliers α_i ($1 \le i \le n$), DCD updates one multiplier at a time. One training epoch updates $\alpha_1, \alpha_2, \ldots, \alpha_n$ sequentially one by one. The DCD algorithm was proved to converge, and in practice it often converges in a few epochs.

Now, suppose we fix $\alpha_1, \ldots, \alpha_{i-1}$ and $\alpha_{i+1}, \ldots, \alpha_n$, and want to find a better value for α_i. Denote the set of Lagrange multipliers after this update as $\boldsymbol{\alpha}'$, in which $\alpha'_i = \alpha_i + d$ and $\alpha'_j = \alpha_j$ for $j \neq i$. Show that

$$f(\boldsymbol{\alpha}') - f(\boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{x}_i\|^2 d^2 + f'(\boldsymbol{\alpha}) d\,.$$

(c) Remember that α_i and α'_i should both be between 0 and C. Then, what is the optimal value for α'_i?

(d) A practical implementation of the DCD algorithm needs to pay attention to a lot of implementation details—e.g., randomize the order of updating for different α_i values in different epochs. Read the DCD paper and understand which details are important. This paper also provides proof for DCD's convergence.

A C++ implementation is provided at https://www.csie.ntu.edu.tw/~cjlin/liblinear/. Download this software and learn how to use it. This software has different solvers for linear classification and regression problems and several parameters. How do you specify parameters to perform classification in equations 72–73?

(e) Download the `rcv1.binary` training set from the LIBSVM data page (https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/).

Use 5-fold cross validation and obtain the training time and the cross-validation accuracy. Compare LIBSVM and LIBLINEAR on this task: use the linear kernel in LIBSVM and use the problem specified by equations 72–73 in LIBLINEAR. Set $C = 4$ in these experiments.