A primer on classification with Suport Vector Machines

MIGUEL DE BENITO*

Universität Augsburg

Last update: June 24, 2015

HC SVNT DRACONES

Beware: This document is incomplete and surely contains errors. It is essentially a rip-off of well known, widely available, higher quality, reputable sources which you'd be better off reading for yourself.

TABLE OF CONTENTS

*. This document was entirely created using the open source, awe inspiring, tool from the future of scientific text editing **TEXMACS**, using the STIX fonts and with zero typesetting-related effort. You definitely need to check it out.

1. CLASSIFICATION

[1]Consider the task of assigning labels to images. We have $N = 10000$ of them, each to be assigned to one of $K = 10$ different categories or ***classes***, like "frog", "deer", "airplane" or "cat". As a kickstart, another 50000 images have already been painstakingly labeled by some unfortunate grad student and we want to use this information to automagically classify the remaining 10000. Because the world desperately needs us to.

Images are 32x32 pixels with 24bit RGB data, stored raw for a total of 3072 bytes per image. We may think of an image as a point, or ***sample*** $x \in \mathbb{R}^D$ with $D = 3072$.

1.1. Start easy

We assume first that we only have $K = 2$ classes. Suppose furthermore that there exists some affine hyperplane $\Pi \subset \mathbb{R}^D$ separating the points of one class from the points of the other. We then say that the data are ***linearly separable***. This allows us to assign a number $y_i \in \{-1, +1\}$ to each $x_i \in \mathbb{R}^D$ representing the class to which it belongs, according to whether it lies on one side or the other of Π.

The (trivial) good news: Affine hyperplanes are described by exactly one vector $w \in \mathbb{R}^D$ normal to the plane plus a shift or, in machine learning parlance, ***bias*** $b \in \mathbb{R}$:

$$\Pi = \{x \in \mathbb{R}^D : w^\top x + b = 0\}.$$

We want to find some hyperplane leaving all training samples of class $+1$ at one side and all the others at the other side. So our model is a function y of the data whose value is the class of a sample, based on its position with respect to some hyperplane to be determined:

$$y(x, w, b) = f(w^\top x + b) \in \{-1, +1\},$$

where $f := \chi_{[0,\infty)} - \chi_{(-\infty,0)}$ is our choice of ***activation function***. "All" we need is to compute once a single $(w, b) \in \mathbb{R}^{D+1}$ using all of the training points. Then in order to decide whether a new point x is in class 1 or 2 we simply compute $w^\top x + b$ and check its sign. Of course, the ***training***, i.e. the computation of the optimal w, b may be very costly.

The bad news: There are many possible choices for Π! Which one is best?

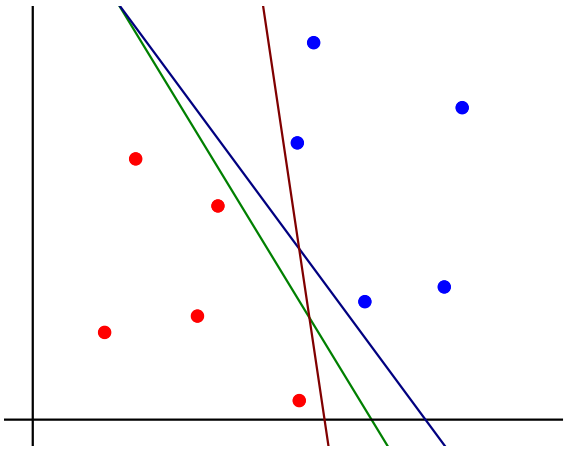


Figure 1. Several possible separating hyperplanes.

1. This is the setup in the CIFAR-10 dataset ([Kri09]).

There are many alternative methods to choose from. Here are three:

- Least squares: turns out to be a poor choice which we will not pursue beyond a quick review.
- Cycle the x_i and adjust w, b according to whether they correctly classify each point in turn. This is the perceptron and turns out to not to be easily generalizable to non-linear data sets.
- Maximize the distance to the separating hyperplane. This is what we will be doing in section 2.

1.2. Least squares

In this approach we can consider an arbitrary number K of classes. Each class has a linear model

$$y_k(\overline{x}) = \overline{w}_k \cdot \overline{x}, \text{ where } \overline{x} = (x, 1), \overline{w} = (w, b) \in \mathbb{R}^{D+1}$$

We can write all models together as

$$y(x) = \overline{W}^\top \overline{x} = \text{“class scores” for sample } \overline{x} \in \mathbb{R}^{D+1},$$

where $\overline{W} \in \mathbb{R}^{(D+1)\times K}$ contains all weight vectors and the winning class for x is the one with maximal y_k. We use a 1-of-K encoding for the training labels: the label for sample x_i is $t_i = (0, \ldots, 1, \ldots, 0) \in \{0, 1\}^K$. We can group all training labels into one matrix: $T \in \mathbb{R}^{N\times K}$. If we write all training samples as $X \in \mathbb{R}^{N\times(D+1)}$, our goal is now to minimise the error

$$E(W) = \frac{1}{2} |\overline{X}\,\overline{W} - T|^2.$$

Being the composition of an affine and a convex function, E is convex so it is enough to set the gradient equal to zero and solve for W to find the unique minimizer

$$\overline{W}^* = (\overline{X}^\top \overline{X})^{-1} \overline{X}^\top T = \overline{X}^\dagger \mathrm{T},$$

where $\overline{X}^\dagger$ is called the ***pseudoinverse***. Note that to write that down, we assumed that $\overline{X}^\top \overline{X}$ is invertible but it need not be.[2] Prediction is done with:

$$y(x) = T^\top \overline{X}^{\dagger\top} \overline{x}.$$

Which is easy but bad! Notice in particular that $y(x)$ is not constrained to be in $\{0, 1\}^K$.

1.3. The Perceptron

Again we must consider only $K = 2$ classes to start with (more on this in section 2.3). Labels are $y \in \{-1, +1\}$ and we assume linearly separable data, that is: there exists $\overline{w}^* = (w^*, b^*) \in \mathbb{R}^{D+1}$ such that for every training sample x_i with label y_i:

$$\overline{w}^* \cdot \overline{x}_i > 0 \Leftrightarrow y_i = +1 \text{ and } \overline{w}^* \cdot \overline{x}_i < 0 \Leftrightarrow y_i = -1.$$

2. A necessary and sufficient condition is that there are no pure correlations among features in the data, i.e. that $\overline{X}$ has full column rank: rank $\overline{X} = d + 1$. Indeed if rank $\overline{X} < d + 1$, then there is $\overline{x} \neq 0$ such that $\overline{X}\,\overline{x} = 0$, but then $\overline{X}^\top \overline{X}\,\overline{x} = 0$ and $\overline{X}^\top \overline{X}$ is not invertible. Conversely, if rank $\overline{X} = d + 1$, then $\ker \overline{X} = \{0\}$ and $\ker \overline{X}^\top \overline{X} = \{0\}$ because $\overline{X}^\top \overline{X}\overline{x} = 0 \Rightarrow 0 = \overline{x}^\top \overline{X}^\top \overline{X}\overline{x} = \|\overline{X}\overline{x}\|^2 \Rightarrow \overline{X}\overline{x} = 0$.

Put another way, we assume that:

$$y_i(\overline{w}^* \cdot \overline{x}_i) > 0 \text{ for every } i = 1, \ldots, N.$$

The model is:

$$y_w(x) = f(\overline{w} \cdot \overline{x}), \text{ with } f = \chi_{[0,\infty)} - \chi_{(-\infty,0)}.$$

How to determine w? Minimisation of $E(\overline{w}) = |\mathscr{M}|$, $\mathscr{M} := \{x_i : y(x_i) \neq y_i\}$ won't be possible with gradient methods because this function is piecewise constant (jumps for w, b where the predicted class for a point x_i flips), hence its gradient vanishes a.e. We have instead the ***Perceptron criterion***: For some $\overline{w} \in \mathbb{R}^{D+1}$ and any sample x:

$$x \text{ is correctly classified by } \overline{w} \Leftrightarrow y_i \overline{w} \cdot \overline{x} > 0.$$

The error is $\sum_{x_i \in \mathscr{M}} y_i \overline{w} \cdot \overline{x} < 0$, so we want to **minimise**

$$E_{\text{per}}(\overline{w}) := -\sum_{x_i \in \mathscr{M}} y_i \overline{w} \cdot \overline{x} \qquad \left(= \sum_{i=1}^{N} \max\{0, -y_i \overline{w} \cdot \overline{x}_i\} \right).$$

Optimization with SGD (more later): Pick $x_i \in \mathscr{M}$, update:

$$\overline{w}_{t+1} = \overline{w}_t - \lambda_t \nabla_w E^i_{\text{per}}(\overline{w}_t) = \overline{w}_t + \lambda_t y_i \overline{x}_i.$$

a) Each step not guaranteed to reduce *overall* error.

b) Convergence guaranteed to *some* solution if data linearly separable.

Interpretation:

1. Pick a sample, if it's correctly classified, pick again.
2. Add or subtract the sample (times the step factor) from the weight vector, shifting it in such a way that the sample is properly classified (i.e. multiplying the vector by y_i).

1.4. What next

- Data almost never, ever, ever is linearly separable. See section 2.2 for how to fix this.
- We typically will have more than two classes. See section 2.3 for several methods to include an arbitrary number of classes.
- We will see that training can be very costly. See section 3.2 for a standard technique solving this.

2. THE OPTIMAL MARGIN CLASSIFIER

As before, assume first that we have linearly separable data belonging to two classes.

Data points live in $\mathbb{R}^D$. There are N of them in the training set. The classes for the training samples are encoded in a vector $y \in \{-1,+1\}^N$. The class for the i-th training sample is $y_i \in \{-1,+1\}$.

2.1. The primal problem

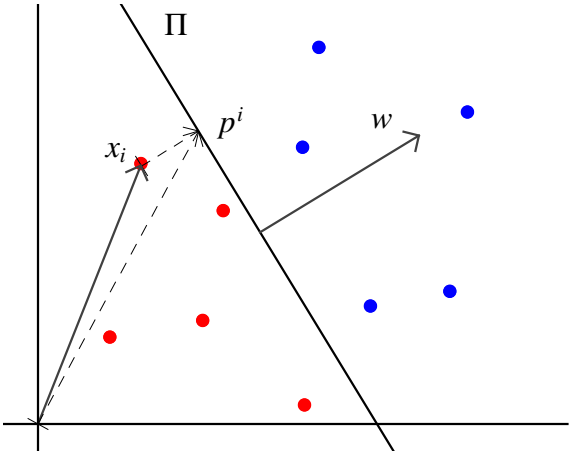


Figure 2. The distance to the separating hyperplane Π. Right $y_i=1$, left $y_i=-1$.

Consider Figure 2. The projection p^i of x_i onto the separating hyperplane

$$\Pi := \{x \in \mathbb{R}^D : w \cdot x + b = 0\}$$

is given by $p^i = x_i \pm \tilde{\gamma}_i \frac{w}{|w|}$, where $\gamma_i \in \mathbb{R}_{\geqslant 0}$ is the distance $|p^i - x_i|$ (in the special situation of the figure $p^i = x_i + \tilde{\gamma}_i \frac{w}{|w|}$). Because $p^i \in \Pi$ we have

$$w \cdot p^i + b = 0 \quad \Rightarrow \quad w \cdot x_i \pm \tilde{\gamma}_i |w| + b = 0 \quad \Rightarrow \quad \tilde{\gamma}_i = \mp\left(\frac{w}{|w|} \cdot x_i + \frac{b}{|w|}\right) \geqslant 0,$$

where the sign is $+$ if x_i is on the side of Π pointed to by w or $-$ otherwise. We can write this using the label $y_i \in \{-1,+1\}$ assigned to sample x_i, and define the ***geometric margin of x_i*** as:

$$\gamma_i := y_i \left(\frac{w}{|w|} \cdot x_i + \frac{b}{|w|}\right). \tag{1}$$

Any hyperplane $\Pi = \{x \in \mathbb{R}^D : w \cdot x + b = 0\}$ splits the space into two half-spaces: a ***positive side***, which we define as the set $\Pi^+ := \{x : w \cdot x > 0\}$, i.e. as those points lying on the half-space in the direction pointed to by w and a ***negative side***, which of course is $\Pi^- := \{x : w \cdot x + b < 0\}$.

Our objective is to maximize the margins (1). Take some training sample $x_i \notin \Pi$ and suppose that $y_i = +1$. Then $\gamma_i > 0$ if $x_i \in \Pi^+$, otherwise it will be negative. In the latter case, maximization of the margin will tend to change w, b so that $x_i \in \Pi^+$. Suppose now that instead $y_i = -1$. Then $\gamma_i < 0$ if $x_i \in \Pi^+$, otherwise it will be positive. In the former case, maximization of the margin will tend to change w, b so that $x_i \in \Pi^-$.

An ***optimal margin classifier*** tries to maximize the minimal geometric margin (see Figure 3) of all points in the training set:

$$\begin{aligned}
(w^*, b^*) &= \operatorname*{argmax}_{w,b} \min_{i=1,\dots,N} \gamma_i(x_i, w, b) \\
&= \operatorname*{argmax}_{w,b} \left\{ \gamma \in \mathbb{R}_+ : \gamma_i = y_i \left(\frac{w}{|w|} \cdot x_i + \frac{b}{|w|} \right) \geqslant \gamma, i = 1, \dots, N \right\} \\
&= \operatorname*{argmax}_{w,b} \left\{ \frac{\hat{\gamma}}{|w|} \in \mathbb{R}_+ : y_i (w \cdot x_i + b) \geqslant \hat{\gamma}, i = 1, \dots, N, \hat{\gamma} = \gamma |w| \right\} \\
&= \operatorname*{argmax}_{w,b} \left\{ \frac{1}{|w|} \in \mathbb{R}_+ : y_i (w \cdot x_i + b) \geqslant 1, i = 1, \dots, N \right\} \\
&= \operatorname*{argmin}_{w,b} \left\{ \frac{1}{2} |w|^2 : y_i (w \cdot x_i + b) \geqslant 1, i = 1, \dots, N \right\}.
\end{aligned}$$

We have obtained the cost function

$$f(w) = \frac{1}{2} |w|^2,$$

which we minimize subject to the conditions

$$y_i (w \cdot x_i + b) \geqslant 1, \forall i = 1, \dots, N.$$

We could now try to solve this optimization problem, but in fact it is not worth it because we were assuming the very uncommon situation of linearly separable data. In section 2.2 we will fix this and in section 2.3 we will see how to manage more than two classes.

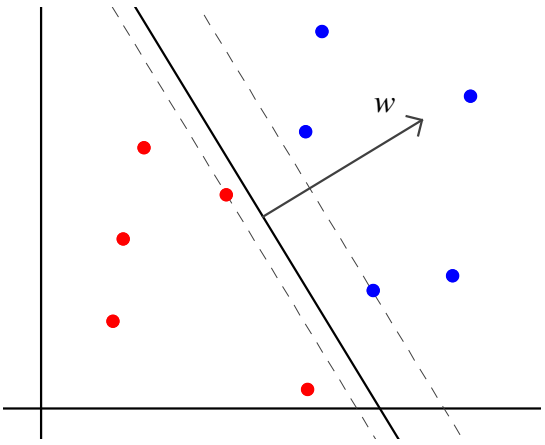


Figure 3. Minimal margins $\gamma = \min_i \gamma_i$ for two classes.

2.2. Non linearly separable classes

To see how simple a data set can be while still not being linearly separable we have the following classical example.

Example. (INFAMOUS-XOR)

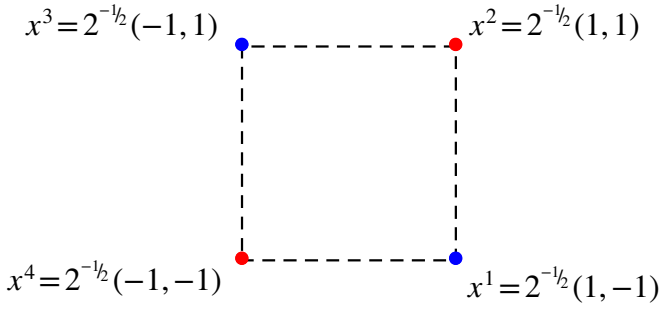


Figure 4. The smallest possible non-linearly separable training set. $y_1 = y_3 = 1$, $y_2 = y_4 = -1$. Someone once said that best way to annoy people working in (classical?) neural networks is to bring up this example. It's still a valid one, though.

Attempting to minimize the primal cost function will either result in ambiguous results or infinite computations, as we show later in section 4.1.

A solution to this problem is to relax the constraints $y_i\,(w \cdot x_i + b) \geqslant 1$ by adding so-called ***slack variables*** ξ_i. Our objective is then to minimize the cost function

$$f(w,b) = \frac{1}{2}|w|^2 + C\sum_{i=1}^{N} \xi_i,$$

for some parameter $C > 0$, while being subject to the constraints

$$y_i(w \cdot x_i + b) \geqslant 1 - \xi_i, \quad \text{and} \quad \xi_i \geqslant 0.$$

A greater value of ξ_i allows for greater violation of the strict margin constraint but is penalized in the cost. Lower values of C allow for "more slack".

Because of the constraints $\xi_i \geqslant 0$, the highest lower bounds to the ξ_i are actually given by $\max\{0,\, 1 - y_i\,(w \cdot x_i + b)\}$ and this means that minimizing $f(w, b)$ is equivalent to minimizing the cost function:

$$f(w,b) := \frac{1}{2}|w|^2 + C\sum_{i=1}^{N} \max\{0, 1 - y_i(w \cdot x_i + b)\}. \tag{2}$$

In this manner we obtain an unconstrained minimization problem

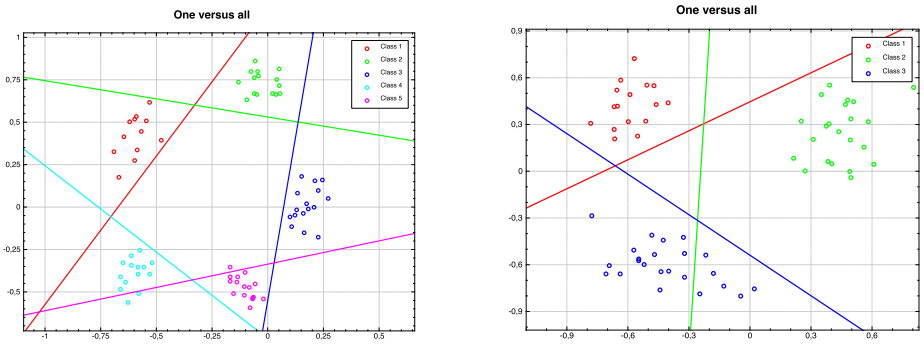
$$\operatorname*{argmin}_{w,b} f(w,b).$$

2.3. Multiclass training

Here are three techniques:

2.3.1. One versus the rest

[3]Fix some class C_k. Pool all the remaining data points of the remaining $K-1$ classes into a fake class C_k^c and find the w_k, b_k separating both. Repeat for all k for a total of K different SVMs trained. This has the problem that the space is not partitioned and later classification of test points may be ambiguous, with several SVMs (or none) claiming a point as theirs (see Figure 5(a)). Therefore a trick is often used, which simply sets $y(x_i) = \text{argmax}\,\{k\colon w_k^\top x_i + b_k\}$, but this has the problem that each SVM was trained on different data and the numbers $w_k^\top x_i + b_k$ may have completely different scales (normalization might help though).



(a) There is ambiguity in the classification of some of these points.

(b) For linearly separable data, *one versus all* works just fine.

Figure 5. In *one versus all*, K separating hyperplanes are computed for K classes.

2.3.2. One versus one

Fix some class C_k. For each of the other $C_j, j \neq k$ find the w_j, b_j which optimally separate C_k from C_j. Repeat for all k for a total of $\binom{k}{2}$ different SVMs trained. To classify a test point [we choose the class having the highest number of votes]. This technique suffers from issues similar to those that afflict the *one versus the rest* technique.

2.3.3. Simultaneous multiclass training

[4]We can change our aproach and simultaneously find the vectors $w_1, \ldots, w_K$ and biases $b_1, \ldots, b_K$. The constrained problem is to minimise [add the derivation]

$$f(W, b, \xi) := \frac{1}{2}|W|^2 + C\sum_{i=1}^{N}\sum_{k\neq y_i}\xi_{ik},$$

subject to

$$(w_{y_i}\cdot x_i + b_{y_i}) - (w_k \cdot x_i + b_k) \geqslant 2 - \xi_{ik}, \text{ and } \xi_{ik} \geqslant 0,$$

where $W = (w_1, \ldots, w_K) \in \mathbb{R}^{K\times D}$, $b \in \mathbb{R}^K$, $\xi \in \mathbb{R}^{N\times K}$, $y \in \{1, \ldots, K\}^N$, $C > 0$, and y_i is the correct class for sample x_i. As in the case of two classes, we may transform this into an unconstrained problem:

$$\underset{W,b}{\text{argmin}}\,\frac{1}{2}|W|^2 + C\sum_{i=1}^{N}\sum_{k\neq y_i}\max\{0, (w_k\cdot x_i + b_k) - (w_{y_i}\cdot x_i + b_{y_i}) + \delta\},$$

3. This is also called ***one versus all***.

4. This idea was put forth in [WW98], but we won't be using their dual formulation. Instead we will optimize the primal problem using stochastic gradient descent (section 3.2).

for some parameter $\delta>0$ which we substituted for the 2 above. [why?]
[Some intuition as to what is going on...]

3. SOLVING THE OPTIMIZATION PROBLEM

Once we have the cost function we may choose to optimize it in several ways. We explain two of them.

3.1. Gradient descent

We can optimize (2) using gradient descent. Its gradient is

$$\nabla_{w,b} f(w,b) = (w,0) + C\sum_{i=1}^{N} \nabla_{w,b} \max\{0, 1-y_i(w\cdot x_i+b)\}$$

where because of the non-differentiability of the max function, we must use the subgradient

$$\nabla_{w,b} \max\{\dots\} = \begin{cases} 0 \text{if } 1-y_i(w\cdot x_i)<0, \\ [0,1] \text{if } 1-y_i(w\cdot x_i)=0, \\ -y_i(x_i,1) \text{otherwise.} \end{cases}$$

[Prove that we may use $\partial_w \max\{\dots\} := 0$] in the case $1-y_i(w\cdot x_i+b)=0$.[5] Choosing a step size λ_t which we may dynamically adapt, the update rule is:

$$\begin{aligned} w_{t+1} &= w_t - \lambda_t \nabla_w f(w_t, b_t) \\ &= (1-\lambda_t) w_t + \lambda_t C \sum_{i=1}^{N} \chi_{(0,\infty)}(1-y_i(w_t\cdot x_i)) y_i x_i. \end{aligned}$$

and

$$\begin{aligned} b_{t+1} &= b_t - \lambda_t \partial_b f(w_t, b_t) \\ &= b_t + \lambda_t C \sum_{i=1}^{N} \chi_{(0,\infty)}(1-y_i(w_t\cdot x_i)) y_i. \end{aligned}$$

[We still have to answer] the following:

- What is the stopping criterion?
- How do we choose λ_t?

Furthermore each step is very costly since we must evaluate the gradient on all N samples, a potentially huge number. A handy alternative is stochastic gradient descent.

3.2. Stochastic gradient descent

Warning. This is in pre-pre-alpha-minus state.

5. An alternative would be to use a smooth approximation of the cost function, e.g. the "Huber / hinge" loss of [Cha07, §2.4.2].

We begin by very briefly placing the concepts seen above within the framework of empirical risk minimization. Define the ***loss*** of (mis)classification for a sample $x \sim X$, given the parameters w to be some scalar function $l(x, w)$. Learning w consists of minimizing the ***expected risk function*** $\mathbb{E}_X[l(x,w)]$, which measures the expected performance on future samples (that is, the *generalization performance*). However, this is something which we cannot do because the distribution over X is unknown. Instead we resort to minimising the ***empirical risk*** as an approximation:

$$f(x,w,b) := \frac{1}{N}\sum_{i=1}^{N} l(x_i,w,b).$$

Recall from section 3.1 that the gradient descent update is a costly evaluation of the gradient of the loss on all N samples. In stochastic gradient descent we compute instead the gradient over one random sample x_i resulting in the update rule

$$w_{t+1} = w_t - \lambda_t \nabla_w l(x_i, w_t, b_t) \quad \text{and} \quad b_{t+1} = b_t - \lambda_t \partial_b l(x_i, w_t, b_t).$$

Notice that the expected value $\mathbb{E}[w_{t+1}] = w_t - \lambda_t \frac{1}{N}\sum_{i=1}^{N} \nabla_w l(x_i, w_t)$ wrt. the empirical distribution is exactly the update rule for gradient descent applied to the empirical risk and the same applies for $\mathbb{E}[b_{t+1}]$. See [Bot04] for some details on the convergence properties and a justification of the choices made in the following:

Algorithm SGD

1. Pick x_i at random from the training set.
2. Update the parameters according to

   $$w_{t+1} = w_t - \lambda_t \nabla_w l(x_i, w_t, b_t) \quad \text{and} \quad b_{t+1} = b_t - \lambda_t \partial_b l(x_i, w_t, b_t).$$

   where $\lambda_t \to 0$ for $t \to \infty$ and $\sum \lambda_t^2 < \infty$, $\sum \lambda_t = \infty$.
3. Repeat from 1. until convergence (to be defined).

Alternatively:

2'. ***Mini-batch update:*** for some $r \in \mathbb{N}$, pick $x_{i_1}, \ldots, x_{i_r}$ at random and do

$$w_{t+1} = w_t - \lambda_t \sum_{j=1}^{r} \nabla_w l(x_{i_j}, w_t)$$

Why? Being a more accurate approximation of the true gradient, this provides better convergence rates while incurring in little computational overhead (for small values of r) if parallelized in-process.

Advantages and disadvantages of SGD:

- Faster convergence for redundant data sets (reported without proof in [Bot91], maybe more in [Bot04]).
- Moderate ability to escape local optima (link with simmulated annealing). See [Bot91, §4.2.]

- Generalizable: take any random function g of the weights w such that $\mathbb{E}_X[g(w)] = \nabla_{(w,b)} f(w)$, then the following update also converges:

$$(w_{t+1}, b_{t+1}) = (w_t, b_t) - \lambda_t g(w_t).$$

- Non obvious / inconvenient stopping criteria. See [KB09, Chapter 2].

3.2.1. SGD for 2 class SVM

We define the loss of (mis)classification for a sample $x \in X$, given the parameters w as

$$l(x, w, b) = \frac{1}{2}|w|^2 + C \max\{0, 1 - y_x (w \cdot x + b)\},$$

where $C > 0$ and y_x is the class of sample x. As explained above, we want to minimise the empirical risk (as an approximation of the expected loss):

$$f(w, b) := \frac{1}{N} \sum_{i=1}^{N} l(x_i, w, b) = \frac{1}{2}|w|^2 + \frac{C}{N} \sum_{i=1}^{N} \max\{0, 1 - y_i (w \cdot x_i + b)\}.$$

Using the subgradient of the loss

$$\begin{aligned} \nabla_{(w,b)} l(x_i, w_t, b_t) &= (w_t, 0)^\top + C \partial_w \max\{0, 1 - y_i (w_t \cdot x_i + b)\} \\ &= (w_t, 0)^\top - C \chi_{(0,\infty)}(1 - y_i (w_t \cdot x_i + b)) y_i (x_i, 1)^\top, \end{aligned}$$

our update rule at a sample x_i becomes

$$w_{t+1} = (1 - \lambda_t) w_t + \lambda_t C \chi_{(0,\infty)}(1 - y_i (w_t \cdot x_i + b)) y_i x_i$$

and

$$b_{t+1} = b_t + \lambda_t C \chi_{(0,\infty)}(1 - y_i (w_t \cdot x_i + b)) y_i.$$

3.2.2. SGD for multiclass SVM

We now develop the approach for multiple classes described in section 2.3.3. In order to use SGD for the optimization step we must consider the loss for one sample:

$$l(x, W, b) = \frac{1}{2}|W|^2 + C \sum_{k \neq y(x)} \max\{0, m_k(x)\},$$

where

$$m_k(x) := (w_k^\top x + b_k) - (w_{y(x)}^\top x + b_{y(x)}) + 2$$

and $y(x)$ is of course the class of training sample x. The gradient along one of the rows w_r of W is

$$\nabla_{w_r} l(x, W, b) = w_r + C \sum_{k \neq y(x)} \chi_{(0,\infty)}(m_k(x)) (\delta_{kr} - \delta_{y(x)r}) x_i$$

and

$$\partial_{b_r} l(x, W, b) = C \sum_{k \neq y(x)} \chi_{(0,\infty)}(m_k(x)) (\delta_{kr} - \delta_{y(x)r}).$$

In the case where we differentiate wrt. the set of weights (w_{y_i}, b_{y_i}) for the class y_i corresponding to sample x_i this gradient becomes

$$\begin{aligned} \nabla_{(w_{y_i}, b_{y_i})} l(x_i, W, b) &= (w_r, 0)^\top - C \sum_{k \neq y_i} \chi_{(0,\infty)}(m_k(x_i)) (x_i, 1)^\top. \\ &= (w_r, 0)^\top - C M (x_i, 1)^\top, \end{aligned}$$

where $M \in \{0, \dots, K-1\}$ denotes the number of classes k where the condition $m_k(x_i) > 0$ is met. For all the other rows we readily see that

$$\nabla_{(w_r,b_r)} l(x_i, W, b) = (w_r, 0)^\top + C\, \chi_{(0,\infty)}(m_r(x_i))\, (x_i, 1)^\top.$$

Consequently, our update rule at a sample x_i is

$$(W_{t+1}, b_{t+1}) = (W_t, b_t) - \lambda_t (W_t, 0) - \lambda_t C \begin{pmatrix} \chi_{(0,\infty)}(m_1(x_i))\, (x_i, 1) \\ \vdots \\ -\sum_{k \neq y_i} \chi_{(0,\infty)}(m_k(x_i))\, (x_i, 1) \\ \vdots \\ \chi_{(0,\infty)}(m_K(x_i))\, (x_i, 1) \end{pmatrix}.$$

When we implement this, we can speed things up by doing the following [notation is too complicated]:

1. Compute the whole vector $m = (m_1(x_i), \dots, m_K(x_i))$.
2. Apply $\chi_{(0,\infty)}$ component-wise to m: $\tilde{m}(x_i) = \chi_{(0,\infty)}(m(x_i))$.
3. Set $M = \#\{k : m_k(x_i) > 0\}$.
4. Set the y_i-th entry of $\tilde{m}$ to its correct value: $\tilde{m}_{y_i} = -M$.
5. Update the weights

$$(W_{t+1}, b_{t+1}) = ((1 - \lambda_t)\, W_t, b_t) - \lambda_t C \tilde{m} \otimes (x_i, 1).$$

4. SUPPORT VECTOR MACHINES

In this section we rewrite the optimization problem ⟨eqref | ⟩ in its dual form using Lagrange multipliers, then generalize it to use so-called kernels and finally discuss two algorithms for SVMs.

4.1. The dual problem

Write the problem

$$\min_x f(x) \text{ subject to } h_i \leqslant 0, g_j = 0$$

as

$$\min_x \max_{\alpha,\beta} \mathscr{L}(x, \alpha, \beta) \text{ subject to } \alpha_i \geqslant 0, \beta_j \geqslant 0$$

where

$$\mathscr{L}(x, \alpha, \beta) := f(x) + \sum \alpha_i h_i + \sum \beta_j g_j.$$

Notice that

$$\max_{\alpha,\beta} \mathscr{L}(x, \alpha, \beta) = \begin{cases} f(x) & \text{if the constraints are satisfied,} \\ \infty & \text{otherwise.,} \end{cases}$$

so the problems are indeed equivalent. Now flip the minimum and the maximum:

$$\max_{\alpha,\beta} \min_x \mathscr{L}(x, \alpha, \beta) \text{ subject to } \alpha_i \geqslant 0, \beta_j \geqslant 0.$$

We have in general $\max_y \min_x F(x,y) \leqslant \min_x \max_y F(x,y)$.[6] So the first problem is an upper bound for the second. The difference

$$p^\star - d^\star = \min_x \max_{\alpha,\beta} \mathscr{L}(x,\alpha,\beta) - \max_{\alpha,\beta} \min_x \mathscr{L}(x,\alpha,\beta) \geqslant 0$$

is called the ***duality gap***. For convex f, h_i and affine g_j, if points x^*, α^*, β^* satisfy the ***Karush-Kuhn-Tucker conditions*** [explain/link], then these points are already the primal and dual optimal $x^\star = x^*, \alpha^\star = \alpha^*, \beta^\star = \beta^*$ and the duality gap is zero.

4.2. Dual formulation of classifiers

4.2.1. The Perceptron

Example. (INFAMOUS-XOR REVISITED)
Assume two classes and linearly separable data. We have the dual problem:

$$\min_{\alpha \in \mathbb{R}_+^N} \frac{1}{2} \sum_{i,j=1}^N y_i y_j \langle x_i, x_j \rangle \alpha_i \alpha_j - \sum_{i=1}^N \alpha_i,$$

subject to:

$$0 \leqslant \alpha_i \text{ and } \sum_{i=1}^N y_i \alpha_i = 0, \quad \text{for all } i = 1, \ldots, N.$$

Attempting to minimize the dual objective will catastrophically fail in the situation of figure 4:

$$\begin{aligned}
&\frac{1}{2} \sum_{i=1}^4 \sum_{j=1}^4 y_i y_j \langle x_i, x_j \rangle \alpha_i \alpha_j - \sum_{i=1}^4 \alpha_i \\
&\quad = \frac{1}{2} (|x^1|^2 \alpha_1^2 - \langle x^1, x^2 \rangle \alpha_1 \alpha_2 + \langle x^1, x^3 \rangle \alpha_1 \alpha_3 - \langle x^1, x^4 \rangle \alpha_1 \alpha_4 \\
&\qquad - \langle x^2, x^1 \rangle \alpha_2 \alpha_1 + |x^2|^2 \alpha_2^2 - \langle x^2, x^3 \rangle \alpha_2 \alpha_3 + \langle x^2, x^4 \rangle \alpha_2 \alpha_4 \\
&\qquad + \langle x^3, x^1 \rangle \alpha_3 \alpha_1 - \langle x^3, x^2 \rangle \alpha_3 \alpha_2 + |x^3|^2 \alpha_3^2 - \langle x^3, x^4 \rangle \alpha_3 \alpha_4 \\
&\qquad - \langle x^4, x^1 \rangle \alpha_4 \alpha_1 + \langle x^4, x^2 \rangle \alpha_4 \alpha_2 - \langle x^4, x^3 \rangle \alpha_4 \alpha_3 + |x^4|^2 \alpha_4^2) - \sum_{i=1}^4 \alpha_i \\
&\quad = \frac{1}{2} \sum_{i=1}^4 \alpha_i^2 - \alpha_1 \alpha_3 - \alpha_2 \alpha_4 - \sum_{i=1}^4 \alpha_i \\
&\quad = \frac{1}{2} [(\alpha_1 - \alpha_3)^2 + (\alpha_2 - \alpha_4)^2] - \sum_{i=1}^4 \alpha_i.
\end{aligned}$$

Hence the minimum over $\alpha_i \in \mathbb{R}_+$ is $-\infty$.

4.2.2. Optimal Margin Classifier with slack variables

As above, we assume $K = 2$. We introduce Lagrange multipliers $\alpha_i \geqslant 0$ for the constraints

$$y_i (w \cdot x_i + b) \geqslant 1 - \xi_i,$$

6. Indeed, letting $m := \max_y \min_x F(x,y)$ this is equivalent to: $\forall x, \exists y$: $F(x,y) \geqslant m$ ($\Leftrightarrow \max_y F(x,y) \geqslant m$ for all $x \Rightarrow \min_x \max_y F(x,y) \geqslant m$). Now suppose this statement doesn't hold: then there exists some x_0 such that $F(x_0,y) < m$ for all y. Consequently $\min_x F(x,y) \leqslant F(x_0,y) < m$ for all y and $m = \max_y \min_x F(x,y) < m$ a contradiction.

and $\beta_i \geqslant 0$ for the constraints $\xi_i \geqslant 0$. We [compute, tweak, tweak, compute... and] obtain the dual problem:

$$\max_{\alpha \in \mathbb{R}_+^N} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N y_i y_j \langle x_i, x_j \rangle \alpha_i \alpha_j,$$

which we can write as a minimisation problem in the form

$$\min_{\alpha \in \mathbb{R}_+^N} g(\alpha), \quad g(\alpha) := \frac{1}{2} \sum_{i,j=1}^N y_i y_j \langle x_i, x_j \rangle \alpha_i \alpha_j - \sum_{i=1}^N \alpha_i \tag{3}$$

subject to:

$$0 \leqslant \alpha_i \leqslant C \quad \text{for all } i = 1, \ldots, N \text{ and } \sum_{i=1}^N y_i \alpha_i = 0. \tag{4}$$

Reading the Karush-Kuhn-Tucker conditions we have for optimal values $\alpha_i^\star, \overline{w}^\star$:

$$\begin{cases} \alpha_i^\star = 0 & \Leftrightarrow \; y_i \overline{w}^\star \cdot \overline{x}_i > 1 \; \Leftrightarrow \; x_i \text{ is “away”}, \\ 0 < \alpha_i^\star < C & \Leftrightarrow \; y_i \overline{w}^\star \cdot \overline{x}_i = 1 \; \Leftrightarrow \; x_i \text{ is on the margin}, \\ \alpha_i^\star = C & \Leftrightarrow \; y_i \overline{w}^\star \cdot \overline{x}_i < 1 \; \Leftrightarrow \; x_i \text{ is inside the margin}. \end{cases}$$

Most samples will be away. The others (by design few) are the **support vectors**.

4.3. Kernels

Notice that the data appear only in the scalar products $\langle x_i, x_j \rangle$. One way to generalize this is the “kernel trick”. One applies an arbitrary transformation $x \mapsto \varphi(x)$ at the beginning to obtain constraints $y_i (w \cdot \varphi(x_i) + b) \geqslant 1$, then [...]. Define $K(x, z) = \varphi(x)\, \varphi(z)$. [Then...]

Now go the other way around: define $K(x, z)$ in some “easy” way and compute the matrix $K_{ij} = K(x_i, x_j)$ [Under suitable conditions, Mercer's theorem] K will be a kernel and K_{ij} its ***kernel matrix***. [Then...]

4.4. Solving the optimization problem

4.4.1. Sequential Minimal Optimization

We now detail an algorithm introduced in [Pla98a, Pla98b].

At each step, optimize (3) in the minimal number of Lagrange multipliers possible such that the constraints are satisfied. Because of the linear constraint $\sum_{i=1}^N y_i \alpha_i = 0$, it is not possible to adjust just one parameter α_i while keeping the rest fixed. Therefore we need to modifiy at least two at each step.

W.l.o.g. let α_1, α_2 be the coefficients which we optimize. The constraint $\sum_{i=1}^N y_i \alpha_i = 0$ implies that

$$y_1 \alpha_1 + y_2 \alpha_2 = -\sum_{i=3}^N y_i \alpha_i.$$

Multiplying the equation by y_2 and writing $s := y_1 y_2$ we obtain

$$\alpha_2 = -y_2 \sum_{i=3}^N y_i \alpha_i - s \alpha_1 = a - s \alpha_1.$$

We have two cases (see Figure 6)

a) $y_1 = y_2 \Rightarrow s = 1$ and then $\alpha_2 = a - \alpha_1$. The range for α_2 is

$$\alpha_2 \in [L, H], \text{ where } L = \max\{0, a - C\}, H = \min\{C, a\}.$$

b) $y_1 \neq y_2 \Rightarrow s = -1$ and then $\alpha_2 = a + \alpha_1$. The range for α_2 is

$$\alpha_2 \in [L, H], \text{ where } L = \max\{0, -a\}, H = \min\{C, C - a\}.$$

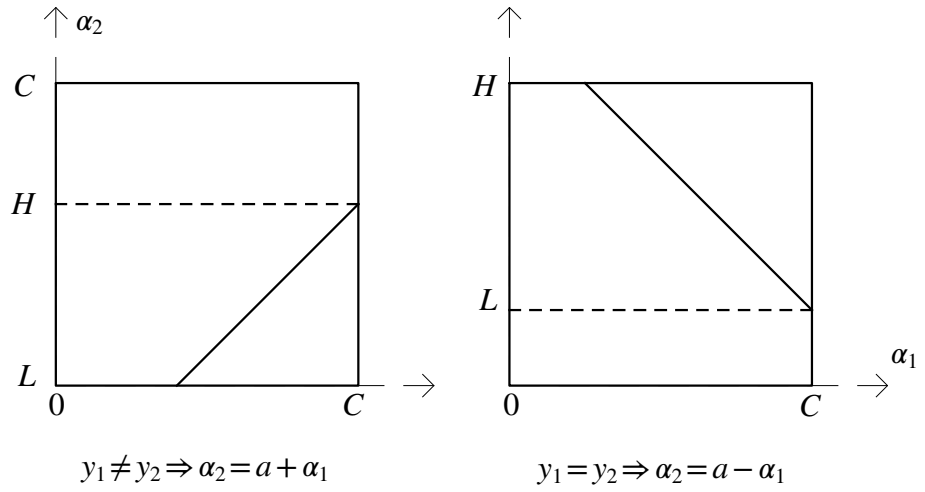


Figure 6. The box constraints for α_1, α_2 and their ranges.

Now write the objective function $g(\alpha)$ in (3) as a function of α_1, α_2:

$$g(\alpha_1, \alpha_2, \ldots) = \frac{1}{2}(\alpha_1^2 k_{11} + \alpha_2^2 k_{22} + 2 s \alpha_1 \alpha_2 k_{12} + 2 y_1 \alpha_1 v_1 + 2 y_2 \alpha_2 v_2) - (\alpha_1 + \alpha_2) + C,$$

where

$$v_i = \sum_{j>2} y_j \alpha_j k_{ij}, \quad i = 1, 2$$

and C represents terms independent of α_1, α_2. Using the relation $\alpha_2 = a - s\alpha_1$ we obtain a function of α_1 only:

$$\begin{aligned} \overline{g}(\alpha_1) \;=\; & \frac{1}{2}\alpha_1^2 k_{11} + \frac{1}{2}(a - s\alpha_1)^2 k_{22} + s\alpha_1(a - s\alpha_1) k_{12} + y_1 \alpha_1 v_1 + y_2 (a - s\alpha_1) v_2 \\ & - (\alpha_1 + (a - s\alpha_1)) + C. \end{aligned}$$

And differentiating we have:

$$\begin{aligned} \overline{g}'(\alpha_1) \;&=\; \alpha_1 k_{11} - s(a - s\alpha_1) k_{22} + s a k_{12} - 2\alpha_1 k_{12} + y_1 v_1 - y_2 s v_2 - (1 - s) \\ &=\; \alpha_1 \underbrace{(k_{11} + k_{22} - 2k_{12})}_{=:\eta} + s a (k_{12} - k_{22}) + y_1 (v_1 - v_2) - (1 - s), \\ \overline{g}''(\alpha_1) \;&=\; \eta. \end{aligned}$$

Let now $\tilde{\alpha}_1$, $\tilde{\alpha}_2$ be values of these parameters from the prior optimization step. Because they also fulfilled the linear constraint on the α_i we have:

$$-a + (\alpha_2 + s\alpha_1) = 0 = -a + (\tilde{\alpha}_2 + s\tilde{\alpha}_1) \Rightarrow a = \tilde{\alpha}_2 + s\tilde{\alpha}_1.$$

Notice too that we may rewrite v_i as

$$v_i = \sum_{j=1}^{N} y_j \tilde{\alpha}_j k_{ij} - y_1 \tilde{\alpha}_1 k_{i1} - y_2 \tilde{\alpha}_2 k_{i2} = \tilde{w} \cdot x - y_1 \tilde{\alpha}_1 k_{i1} - y_2 \tilde{\alpha}_2 k_{i2}$$

and substituting into the previous equation yields

$$\begin{aligned} \overline{g}'(\alpha_1) &= \alpha_1 \eta + (s\tilde{\alpha}_2 + \tilde{\alpha}_1)(k_{12} - k_{22}) + y_1(v_1 - v_2) - (1-s) \\ &= \alpha_1 \eta + s\tilde{\alpha}_2 k_{12} - s\tilde{\alpha}_2 k_{22} + \tilde{\alpha}_1 k_{12} - \tilde{\alpha}_1 k_{22} + y_1 \tilde{w}(x_1 - x_2) \\ &\quad + y_1(y_1 \alpha_1 k_{11} - y_2 \alpha_2 k_{12}) - y_1(y_1 \alpha_1 k_{21} - y_2 \alpha_2 k_{22}) - (1-s) \\ &= \alpha_1 \eta - \tilde{\alpha}_1 \eta + y_1[(\tilde{w} \cdot x_1 - y_1) - (\tilde{w} \cdot x_2 - y_2)]. \end{aligned}$$

Therefore, for $\eta > 0$ (which will be the case if k is positive definite, and this iff the data matrix has full column rank), the minimiser is

$$\alpha_1^\star = \tilde{\alpha}_1 - \frac{y_1[(\tilde{w} \cdot x_1 - y_1) - (\tilde{w} \cdot x_2 - y_2)]}{\eta}.$$

We must now clip it to the bounding box $[0, C]^2$ so that the actual minimiser is

$$\alpha_1^* = \max\{0, \min\{C, \alpha_1^\star\}\}.$$

We can compute α_2^* from this value using that $\alpha_2^* = a - s\alpha_1^* = \tilde{\alpha}_2 + s\tilde{\alpha}_1 - s\alpha_1^*$:

$$\alpha_2^* = \tilde{\alpha}_2 + s(\tilde{\alpha}_1 - \alpha_1^*).$$

In order to choose the variables to optimise, SMO uses two heuristics: [Finish!]

APPENDIX A. IMPLEMENTATION

Source code in C++ is ⟨hlink|available at BitBucket|⟩. You will need:

- A C++11 compiler. Any recent version of GCC or CLANG should do.
- CMake version ⩾ 3.0.2.
- The Qt4 libraries if you want to try the examples with a graphical interface.
- The Armadillo linear algebra library, version ⩾5.200. Its developers recommend using OpenBLAS with it for its parallelization.
- Optionally some datasets: I've used CIFAR-10 ([Kri09]) and MNIST.

BIBLIOGRAPHY

[Bot91] Léon Bottou. Stochastic Gradient learning in Neural Networks. In *Proceedings of Neuro-Nîmes 91*. Nimes, France, 1991. EC2.

[Bot04] Léon Bottou. Stochastic learning. In Olivier Bousquet and Ulrike von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, Berlin, 2004.

[Cha07] Olivier Chapelle. Training a Support Vector Machine in the primal. In *Large-scale Kernel Machines*, Neural Information Processing. MIT Press, 2007.

[KB09] Joseph Keshet and Samy Bengio, editors. *Automatic Speech and Speaker Recognition: Large Margin and Kernel Methods*. Jan 2009.

[Kri09] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, Department of Computer Science, University of Toronto, Toronto, apr 2009.

[Pla98a] John Platt. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Technical report MSR-TR-98-14, Microsoft Research, apr 1998.

[Pla98b] John C. Platt. Fast Training of Support Vector Machines Using Sequential Minimal Optimization. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, jan 1998.

[WW98] Jason Weston and Chris Watkins. Multi-class Support Vector Machines. Technical report, Royal Holloway University of London, 1998.