Support vector machines unit:

- Linear programming (last week Wednesday)
- SVM concepts (last week Friday)
- Lab: SVM applications (Monday)
- The math of SVMs (Wednesday)
- SVM algorithms (**today**)

Today:

- Summary of hard-margin version, with algorithm
- Quadratic programming and QP solver library
- Soft-margin version
- Kernelized version
- Put it all together in an algorithm

The most important source for all of this was Stephen Marsland, *Machine Learning: An Algorithmic Perspective*, 2015, pg 179–183.

Given training data $\mathbf{X}, \boldsymbol{y}$, where $y_n \in \{-1, +1\}$, find $\boldsymbol{w}$, b, and r, specifically

$$\begin{aligned} \text{maximize} \quad & r \\ \text{subject to the constraints} \quad & \forall\ \boldsymbol{x_n}, y_n, \quad y_n(\boldsymbol{w}^T \boldsymbol{x_n} + b) \geq r \\ & ||\boldsymbol{w}|| = 1 \\ & r > 0 \end{aligned}$$

Or, equivalently

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}||\boldsymbol{w}||^2 \\ \text{subject to the constraints} \quad & \forall\ \boldsymbol{x_n}, y_n, \quad y_n(\boldsymbol{w}^T \boldsymbol{x_n} + b) \geq 1 \\ \text{alternately written as} \quad & \forall\ \boldsymbol{x_n}, y_n, \quad 1 - y_n(\boldsymbol{w}^T \boldsymbol{x_n} + b) \geq 0 \end{aligned}$$

Combining the margin maximization with the fact that examples need to be on the correct side of the hyperplane (based on their labels) gives us

$$\max_{\boldsymbol{w},b} \quad \frac{1}{\|\boldsymbol{w}\|} \tag{12.16}$$

$$\text{subject to } y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geqslant 1 \quad \text{for all} \quad n = 1, \ldots, N. \tag{12.17}$$

The squared norm results in a convex quadratic programming problem for the SVM (Section 12.5).

Instead of maximizing the reciprocal of the norm as in (12.16), we often minimize the squared norm. We also often include a constant $\frac{1}{2}$ that does not affect the optimal $\boldsymbol{w}, b$ but yields a tidier form when we compute the gradient. Then our objective becomes

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 \tag{12.18}$$

$$\text{subject to } y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geqslant 1 \quad \text{for all} \quad n = 1, \ldots, N. \tag{12.19}$$

hard margin SVM

Equation (12.18) is known as the *hard margin SVM*. The reason for the expression "hard" is because the preceding formulation does not allow for any violations of the margin condition. We will see in Section 12.2.4 that this "hard" condition can be relaxed to accommodate violations if the data is not linearly separable.

12.2.3 Why We Can Set the Margin to 1

"scaling the data is misleading since if $\vec{w}$ & b not we are changing not X

In Section 12.2.1, we argued that we would like to maximize some value r, which represents the distance of the closest example to the hyperplane. In Section 12.2.2, we scaled the data such that the closest example is of distance 1 to the hyperplane. In this section, we relate the two derivations and show that they are equivalent.

Theorem 12.1. *Maximizing the margin r, where we consider normalized weights as in* (12.10),

$$\max_{\boldsymbol{w},b,r} \quad \underbrace{r}_{\text{margin}}$$

$$\text{subject to} \quad \underbrace{y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b)}_{\text{data fitting}} \geqslant r, \quad \underbrace{\|\boldsymbol{w}\| = 1}_{\text{normalization}}, \quad r > 0, \tag{12.20}$$

is equivalent to scaling the data, such that the margin is unity:

$$\min_{\boldsymbol{w},b} \quad \underbrace{\frac{1}{2}\|\boldsymbol{w}\|^2}_{\text{margin}}$$

$$\text{subject to} \quad \underbrace{y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b)}_{\text{data fitting}} \geqslant 1. \tag{12.21}$$

Proof Consider (12.20). Since the square is a strictly monotonic transformation for nonnegative arguments, the maximum stays the same if we consider r^2 in the objective. Since $\|\boldsymbol{w}\| = 1$, we can reparametrize the equation with a new weight vector $\boldsymbol{w}'$ that is not normalized by explicitly using $\frac{\boldsymbol{w}'}{\|\boldsymbol{w}'\|}$. We obtain

$$\max_{\boldsymbol{w}',b,r} \quad r^2$$

$$\text{subject to} \quad y_n\left(\left\langle \frac{\boldsymbol{w}'}{\|\boldsymbol{w}'\|}, \boldsymbol{x}_n \right\rangle + b\right) \geqslant r, \quad r > 0. \tag{12.22}$$

Equation (12.22) explicitly states that the distance r is positive. Therefore, we can divide the first constraint by r, which yields

Note that $r > 0$ because we assumed linear separability, and hence there is no issue to divide by r.

$$\max_{\boldsymbol{w}',b,r} \quad r^2$$

$$\text{subject to} \quad y_n\left(\left\langle \underbrace{\frac{\boldsymbol{w}'}{\|\boldsymbol{w}'\|r}}_{\boldsymbol{w}''}, \boldsymbol{x}_n \right\rangle + \underbrace{\frac{b}{r}}_{b''}\right) \geqslant 1, \quad r > 0 \tag{12.23}$$

renaming the parameters to $\boldsymbol{w}''$ and b''. Since $\boldsymbol{w}'' = \frac{\boldsymbol{w}'}{\|\boldsymbol{w}'\|r}$, rearranging for r gives

$$\|\boldsymbol{w}''\| = \left\| \frac{\boldsymbol{w}'}{\|\boldsymbol{w}'\|r} \right\| = \frac{1}{r} \cdot \left\| \frac{\boldsymbol{w}'}{\|\boldsymbol{w}'\|} \right\| = \frac{1}{r}. \tag{12.24}$$

By substituting this result into (12.23), we obtain

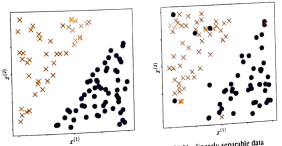
$$\max_{\boldsymbol{w}'',b''} \quad \frac{1}{\|\boldsymbol{w}''\|^2}$$

$$\text{subject to} \quad y_n(\langle \boldsymbol{w}'', \boldsymbol{x}_n \rangle + b'') \geqslant 1. \tag{12.25}$$

The final step is to observe that maximizing $\frac{1}{\|\boldsymbol{w}''\|^2}$ yields the same solution as minimizing $\frac{1}{2}\|\boldsymbol{w}''\|^2$, which concludes the proof of Theorem 12.1. □

12.2.4 Soft Margin SVM: Geometric View

In the case where data is not linearly separable, we may wish to allow some examples to fall within the margin region, or even to be on the wrong side of the hyperplane as illustrated in Figure 12.6.



(a) Linearly separable data, with a large margin

(b) Nonlinearly separable data

Figure 12.6 (a) Linearly separable and (b) nonlinearly separable data.

The Lagrangian of the hard-margin version is

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}||w||^2 + \sum_{n=0}^{N-1} \lambda_n (1 - y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b))$$

Take the gradient with respect to $\boldsymbol{w}$ and b, set to $\mathbf{0}$ or 0

$$\begin{aligned} \nabla_{\boldsymbol{w}} \mathcal{L} &= \boldsymbol{w} - \textstyle\sum_{n=0}^{N-1} \lambda_n y_n \boldsymbol{x}_n \\ \mathbf{0} &= \boldsymbol{w} - \textstyle\sum_{n=0}^{N-1} \lambda_n y_n \boldsymbol{x}_n \\ \boldsymbol{w}* &= \textstyle\sum_{n=0}^{N-1} \lambda_n y_n \boldsymbol{x}_n \end{aligned} \qquad \begin{aligned} \nabla_b \mathcal{L} &= -\textstyle\sum_{n=0}^{N-1} \lambda_n y_n \\ 0 &= -\textstyle\sum_{n=0}^{N-1} \lambda_n y_n \end{aligned}$$

Wait, how does this help us find $b*$?

. . . where $\boldsymbol{w}*$ and $b*$ are the optimal weights and bias.

$\boldsymbol{w}* = \sum_{n=0}^{N-1} \lambda_n y_n \boldsymbol{x}_n = \left[\ldots \sum_{n=0}^{N-1} \lambda_n y_n \boldsymbol{x}_{n,i} \ldots \right]$ and $\sum_{n=0}^{N-1} \lambda_n y_n = 0.$

Theorem 3.4.3 in Han Veiga and Ged tells us that with $\boldsymbol{w}*$, $b*$, and $\boldsymbol{\lambda}*$,

$$\forall\ \boldsymbol{x}_n, y_n, \quad \lambda_n(1 - y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)) = 0$$

. . . which implies $\lambda_n = 0$ for all *non-support vectors* $\boldsymbol{x}_n$

Substitute $\boldsymbol{w}*$ and $b*$ into the Lagrangian to make it a function just of $\boldsymbol{\lambda}$.

$$\underbrace{\mathcal{D}(\boldsymbol{\lambda})}_{\text{dual}} = \mathcal{L}(\underbrace{\boldsymbol{w}*, b*}_{\text{filled-in}}, \boldsymbol{\lambda}) = \tfrac{1}{2}||\boldsymbol{w}*||^2 + \sum_{n=0}^{N-1} \lambda_n(1 - y_n(\boldsymbol{w}*^T \boldsymbol{x}_n + b*))$$

Simplify this based on results above into a quadratic program involving only $\mathbf{X}$, $\boldsymbol{y}$, and $\boldsymbol{\lambda}$ with constraints $\sum_{n=0}^{N-1} \lambda_n y_n = 0$.

A *quadratic programming* problem (or a *quadratic program*) can be stated as, minimize

$$\frac{1}{2}\boldsymbol{x}^T\mathbf{P}\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x}$$

subject to

$$\begin{aligned} \mathbf{G}\boldsymbol{x} &\leq \boldsymbol{h} \\ \mathbf{A}\boldsymbol{x} &= \boldsymbol{b} \end{aligned}$$

In this formula, let n be the number of variables, m be the number of inequality constraints, and ℓ be the number of equality constraints.

Soft-margin form:

$$\text{minimize} \quad \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{n=0}^{N-1}\xi_n$$

$$\text{subject to the constraints} \quad \forall\, \boldsymbol{x_n}, y_n, \quad y_n(\boldsymbol{w}^T\boldsymbol{x_n} + b) \geq 1 - \xi_n$$

Define a hyperplane

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

such that

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

classifies $\boldsymbol{x}$.

ϕ is a feature map that projects the vectors $\boldsymbol{x}$ into higher dimensions. Assume k is a kernel function corresponding to ϕ for efficiently computing dot products in these higher dimensions.

$$k(\boldsymbol{x}_a, \boldsymbol{x}_b) = \phi(\boldsymbol{x}_a)^T \phi(\boldsymbol{x}_b)$$

Kernelized form (hard- or soft-margin):

$$\text{minimize} \quad \frac{1}{2}||\boldsymbol{w}||^2 \quad \left[+C\sum_{n=0}^{N-1}\xi_n\right]$$

$$\text{subject to the constraints} \quad \forall\, \boldsymbol{x_n}, y_n, \quad y_n(\boldsymbol{w}^T\phi(\boldsymbol{x_n}+b)) \geq 1 \; [-\xi_n]$$

(This doesn't actually use the kernel function... but when we put the problem in this form, we anticipate using the kernel.)

This *quadratic programming problem* has an equivalent *Lagrangian* function

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{n=0}^{N-1} \lambda_n y_n \left(\boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b \right)$$

This function has the *dual representation*

$$\mathcal{D}(\boldsymbol{\lambda}) = \sum_{n=0}^{N-1} \lambda_n - \frac{1}{2} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} \lambda_n \lambda_m y_n y_m \cdot k(\boldsymbol{x_n}, \boldsymbol{x_m})$$

which we want to maximize subject to constraints

$$0 \leq \lambda_n \; [\leq C],$$

$$\textstyle\sum_{n=0}^{N-1} \lambda_n y_n = 0$$

where $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^T \phi(\boldsymbol{x}_m)$

Let $\mathbf{K}$ be the *kernel matrix* for data set $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots \boldsymbol{x}_{N-1}$:

$$\mathbf{K} = \begin{pmatrix} k(\boldsymbol{x_0}, \boldsymbol{x_0}) & k(\boldsymbol{x_1}, \boldsymbol{x_0}) & \cdots & k(\boldsymbol{x_{N-1}}, \boldsymbol{x_0}) \\ k(\boldsymbol{x_0}, \boldsymbol{x_1}) & k(\boldsymbol{x_1}, \boldsymbol{x_1}) & \cdots & k(\boldsymbol{x_{N-1}}, \boldsymbol{x_1}) \\ \vdots & & & \vdots \\ k(\boldsymbol{x_0}, \boldsymbol{x_{N-1}}) & k(\boldsymbol{x_1}, \boldsymbol{x_{N-1}}) & \cdots & k(\boldsymbol{x_{N-1}}, \boldsymbol{x_{N-1}}) \end{pmatrix}$$

Then

$$\mathcal{D}(\boldsymbol{\lambda}) = \sum_{n=1}^{N-1} \lambda_n - \frac{1}{2} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} \lambda_n \lambda_m y_n y_m \cdot k(\boldsymbol{x_n}, \boldsymbol{x_m})$$

becomes

$$\mathcal{D}(\boldsymbol{\lambda}) = \mathbf{1}^T \boldsymbol{\lambda} - \frac{1}{2} \boldsymbol{\lambda}^T (\boldsymbol{y}\boldsymbol{y}^T \circ \mathbf{K}) \boldsymbol{\lambda}$$

where $\mathbf{1} = [1\,1\,1 \cdots 1]^T$ and $\circ$ indicates the Hadamard product.

In the formula

$$\mathcal{D}(\boldsymbol{\lambda}) = \mathbf{1}^T\boldsymbol{\lambda} - \frac{1}{2}\boldsymbol{\lambda}^T(\boldsymbol{y}\boldsymbol{y}^T \circ \mathbf{K})\boldsymbol{\lambda}$$

the Hadamard product $\circ$ gives us

$$\boldsymbol{y}\boldsymbol{y}^T \circ \mathbf{K} = \begin{pmatrix} y_0 \cdot y_0 \cdot k(\boldsymbol{x_0}, \boldsymbol{x_0}) & y_1 \cdot y_0 \cdot k(\boldsymbol{x_1}, \boldsymbol{x_0}) & \cdots & y_{N-1} \cdot y_0 \cdot k(\boldsymbol{x_{N-1}}, \boldsymbol{x_0}) \\ y_0 \cdot y_1 \cdot k(\boldsymbol{x_0}, \boldsymbol{x_1}) & y_1 \cdot y_1 \cdot k(\boldsymbol{x_1}, \boldsymbol{x_1}) & \cdots & y_{N-1} \cdot y_1 \cdot k(\boldsymbol{x_{N-1}}, \boldsymbol{x_1}) \\ \vdots & & & \vdots \\ y_0 \cdot y_{N-1} \cdot k(\boldsymbol{x_0}, \boldsymbol{x_{N-1}}) & y_1 \cdot y_{N-1} \cdot k(\boldsymbol{x_1}, \boldsymbol{x_{N-1}}) & \cdots & y_{N-1} \cdot y_{N-1} \cdot k(\boldsymbol{x_{N-1}}, \boldsymbol{x_{N-1}}) \end{pmatrix}$$

Quadratic programming problem:

$$\min \quad \tfrac{1}{2}\boldsymbol{x}^T\mathbf{P}\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x} \qquad \begin{aligned} \mathbf{G}\boldsymbol{x} &\leq \boldsymbol{h} \\ \mathbf{A}\boldsymbol{x} &= \boldsymbol{b} \end{aligned}$$

Our problem:

$$\max \quad \mathbf{1}^T\boldsymbol{\lambda} - \tfrac{1}{2}\boldsymbol{\lambda}^T(\boldsymbol{y}\boldsymbol{y}^T \circ \mathbf{K})\boldsymbol{\lambda} \qquad \begin{aligned} 0 \leq \lambda_i \quad [\leq \quad C] \\ \textstyle\sum_{i=0}^{N-1} \lambda_i y_i \quad = \quad 0 \end{aligned}$$

We want to find $\boldsymbol{\lambda}$. Let $\mathbf{P} = \boldsymbol{y}\boldsymbol{y}^T \circ \mathbf{K}$, $\boldsymbol{q} = \begin{pmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{pmatrix}$, $\mathbf{A} = \begin{pmatrix} y_0 & y_1 & \cdots & y_{N-1} \end{pmatrix}$, and $\boldsymbol{b} = \begin{pmatrix} 0 \end{pmatrix}$.

Quadratic programming problem:

$$\min \quad \tfrac{1}{2}\boldsymbol{x}^T\mathbf{P}\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x} \qquad \begin{aligned} \mathbf{G}\boldsymbol{x} &\leq \boldsymbol{h} \\ \mathbf{A}\boldsymbol{x} &= \boldsymbol{b} \end{aligned}$$

For *hard* margin classification ($0 \leq \lambda_i$),

$$\mathbf{G} = -\mathcal{I} = \begin{pmatrix} -1 & 0 & \dots & 0 \\ 0 & -1 & \dots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \dots & -1 \end{pmatrix}$$

$$\boldsymbol{h} = \mathbf{0} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Quadratic programming problem:

$$\min \quad \tfrac{1}{2}\boldsymbol{x}^T\mathbf{P}\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x}$$

$$\begin{aligned} \mathbf{G}\boldsymbol{x} &\leq \boldsymbol{h} \\ \mathbf{A}\boldsymbol{x} &= \boldsymbol{b} \end{aligned}$$

For *soft* margin classification ($0 \leq \lambda_i \leq C$),

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \vdots & 1 \\ -1 & 0 & \dots & 0 \\ 0 & -1 & \dots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \dots & -1 \end{pmatrix} \qquad \boldsymbol{h} = \begin{pmatrix} C \\ C \\ \vdots \\ C \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Support vectors are $\{\boldsymbol{x}_i \mid \lambda_i \neq 0\}$.

Weights are

$$\boldsymbol{w} = \sum_{n=0}^{N-1} \lambda_n y_n \boldsymbol{x}_n = \sum_{n \mid \lambda_n \neq 0} \lambda_n y_n \boldsymbol{x}_n$$

Intercept/bias is

$$b = \frac{1}{|\{\lambda_n \mid \lambda_n \neq 0\}|} \sum_{m \mid \lambda_j \neq 0} \left(y_j - \sum_{n \mid \lambda_n \neq 0} \lambda_n y_n \cdot k(\boldsymbol{x}_n, \boldsymbol{x}_m) \right)$$

To train a classifier for hard margin classification:

Given **data X**, **targets** $\boldsymbol{y}$, and **kernel function** k,

- Compute kernel matrix $\mathbf{K}$
- Compute $\mathbf{P} = \boldsymbol{y}\boldsymbol{y}^T \circ \mathbf{K}$
- Assemble $\boldsymbol{q}$ vector of -1s
- Assemble $\mathbf{A}$ matrix of y_i along the diagonal
- Assemble $\mathbf{G}$ matrix of -1s along the diagonal
- Assemble $\boldsymbol{h}$ vector of 0s
- Compute $\boldsymbol{\lambda}$ vector by feeding $\mathbf{P}$, $\boldsymbol{q}$, $\boldsymbol{G}$, $\boldsymbol{h}$, $\mathbf{A}$, and $\boldsymbol{b} = [0]$ into QP solver
- Select support vectors from $\boldsymbol{\lambda}$ that are not zero
- Compute b

(For soft margin, modify $\mathbf{G}$ and $\boldsymbol{h}$.)

To classify new data point $\boldsymbol{x}$, compute $\text{sign}(\sum_{n \mid \lambda_n \neq 0} \lambda_n y_n k(\boldsymbol{x}_n, \boldsymbol{x}) + b)$

Coming up:

Due Fri, Mar 7:
Take SVM quiz

Due Wed, Mar 19:
Implement SVM classification

(Midterm on Fri, Mar 21)