Introduction to Statistical Learning and Machine Learning

Chap 5 & Chap6 – SVM and Kernel Methods

Yanwei Fu

School of Data Science, Fudan University

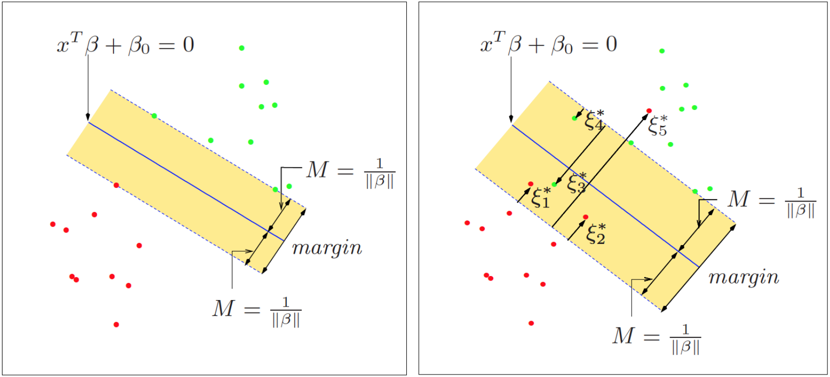


FIGURE 12.1. *Support vector classifiers. The left panel shows the separable case. The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width* $2M = 2/\|\beta\|$. *The right panel shows the nonseparable (overlap) case. The points labeled* ξ_j^* *are on the wrong side of their margin by an amount* $\xi_j^* = M\xi_j$; *points on the correct side have* $\xi_j^* = 0$. *The margin is maximized subject to a total budget* $\sum \xi_i \leq$ *constant. Hence* $\sum \xi_j^*$ *is the total distance of points on the wrong side of their margin.*

Different Forms of SVM (seperated cases)

$$\max_{\beta,\beta_0,\|\beta\|_2=1} M$$
$$s.t. y_i \left(x_i^T \beta + \beta_0\right) \geqslant M, \quad i = 1, \cdots, n \qquad (1)$$

which is equivalent to

$$\min \parallel \beta \parallel_2$$
$$s.t. y_i \left(x_i^T \beta + \beta_0\right) \geqq 1, \quad i = 1, \cdots, n$$

A natural way to modify the constraint in Eq(1) is by introducing the slack variable $\xi = (\xi_1, \cdots, \xi_n)$:

$$y_i \left(x_i^T \beta + \beta_0\right) \geqslant M \left(1 - \xi_i\right)$$

$\forall i, \xi_i \geqslant 0, \sum_i \xi_i \leqslant \mathit{constant}$

Remark: $M \sum_i \xi_i$ measures the total amount distance of points on the wrong side of their margin.

Different Forms of SVM (non-seperatable cases)

$$\min \| \beta \|_2^2$$
$$s.t. y_i \left(x_i^T \beta + \beta_0\right) \geqslant 1 - \xi_i, \quad i = 1, \cdots, n \tag{2}$$
$$\xi_i \geqslant 0, \sum_i \xi_i \leqslant constant$$

$$\min \frac{1}{2} \| \beta \|_2^2 + C \sum_i \xi_i$$
$$s.t. y_i \left(x_i^T \beta + \beta_0\right) \geqslant 1 - \xi_i, \quad i = 1, \cdots, n, \xi_i \geqslant 0 \tag{3}$$

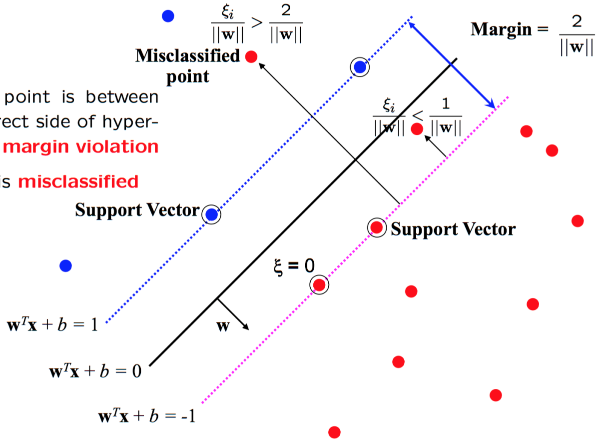
$$\min \sum_{i=1}^{n} \left[1 - y_i \left(x_i^T \beta + \beta_0\right)\right]_+ + \frac{\lambda}{2} \| \beta \|_2^2 \tag{4}$$

where x_+ indicates the positive part of x. If $\lambda = C/2$, then Eq(3) and Eq(4) are equivalent.

Introduce "slack" variables

$\xi_i \geq 0$

- for $0 < \xi \leq 1$ point is between margin and correct side of hyperplane. This is a **margin violation**
- for $\xi > 1$ point is **misclassified**



Learning an SVM has been formulated as a constrained optimization problem over $\mathbf{w}$ and ξ

$$\min_{\mathbf{w}\in\mathbb{R}^d, \xi_i \in \mathbb{R}^+} ||\mathbf{w}||^2 + C\sum_i^N \xi_i \text{ subject to } y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 - \xi_i \text{ for } i = 1 \dots N$$

The constraint $y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 - \xi_i$, can be written more concisely as

$$y_i f(\mathbf{x}_i) \geq 1 - \xi_i$$

which, together with $\xi_i \geq 0$, is equivalent to

$$\xi_i = \max\left(0, 1 - y_i f(\mathbf{x}_i)\right)$$

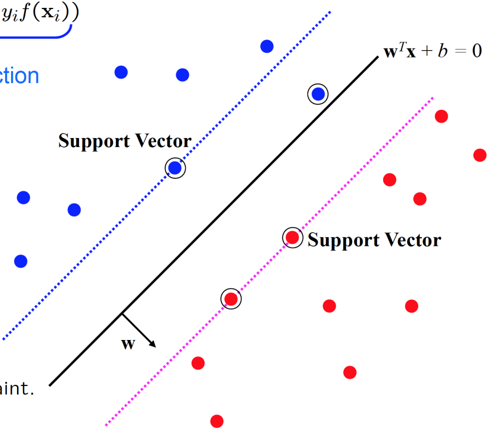
Hence the learning problem is equivalent to the unconstrained optimization problem over $\mathbf{w}$

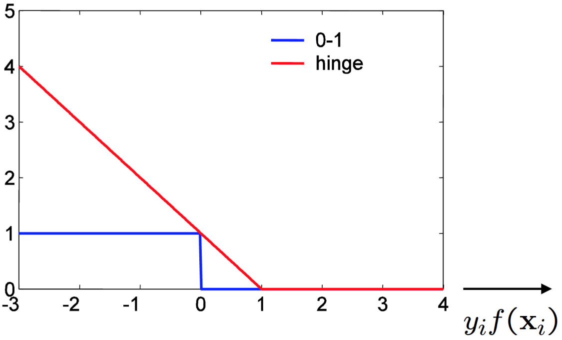
$$\min_{\mathbf{w}\in\mathbb{R}^d} \underbrace{||\mathbf{w}||^2}_{\text{regularization}} + C\sum_i^N \underbrace{\max\left(0, 1 - y_i f(\mathbf{x}_i)\right)}_{\text{loss function}}$$

$$\min_{\mathbf{w}\in\mathbb{R}^d} ||\mathbf{w}||^2 + C\sum_{i}^{N} \underbrace{\max(0, 1 - y_i f(\mathbf{x}_i))}_{\text{loss function}}$$

Points are in three categories:

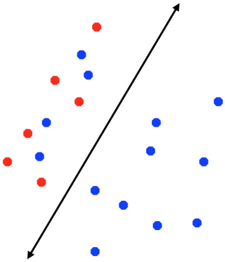
1. $y_i f(x_i) > 1$
   Point is outside margin.
   No contribution to loss
2. $y_i f(x_i) = 1$
   Point is on margin.
   No contribution to loss.
   As in hard margin case.
3. $y_i f(x_i) < 1$
   Point violates margin constraint.
   Contributes to loss





- SVM uses “hinge” loss $\max(0, 1 - y_i f(x_i))$
- an approximation to the 0-1 loss

How to deal with imbalanced data?



- In many practical applications we may have **imbalanced** data sets
- We may want errors to be equally distributed between the positive and negative classes
- A slight modification to the SVM objective does the trick!

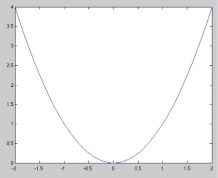
$$\min_{w,b} \frac{1}{2}||w||_2^2 + \frac{C}{N_+} \sum_{j\,:\,y_j=+1} \xi_j + \frac{C}{N_-} \sum_{j\,:\,y_j=-1} \xi_j$$

Class-specific weighting of the slack variables

Constrained optimization

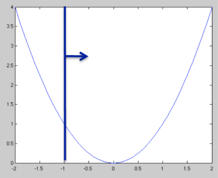
$$\min_x \; x^2 \quad \text{s.t.} \quad x \geq b$$

No Constraint



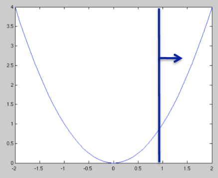
x*=0

x ≥ -1



x*=0

x ≥ 1

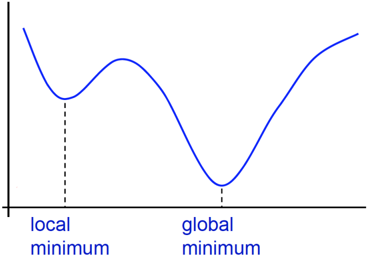


x*=1

How do we solve with constraints?

→ Lagrange Multipliers!!!

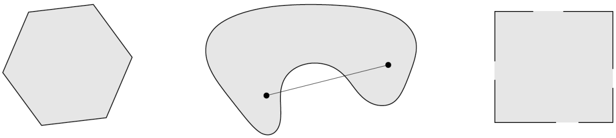
$$\min_{\mathbf{w}\in\mathbb{R}^d} C\sum_i^N \max\left(0, 1 - y_i f(\mathbf{x}_i)\right) + ||\mathbf{w}||^2$$



- Does this cost function have a unique solution?
- Does the solution depend on the starting point of an iterative optimization algorithm (such as gradient descent)?

If the cost function is convex, then a locally optimal point is globally optimal (provided the optimization is over a convex set, which it is in our case) contains the line segment between any two points in the set

$$x_1, x_2 \in C, \quad 0 \leq \theta \leq 1 \quad \Longrightarrow \quad \theta x_1 + (1-\theta)x_2 \in C$$



convex   not convex   not convex

Affine set: solution set of linear equations $Ax = b$

Halfspace: solution of one linear inequality $a^T x \leq b$ $(a \neq 0)$

Polyhedron: solution of finitely many linear inequalities $Ax \leq b$

Ellipsoid: solution of positive definite quadratic inequality

$$(x - x_c)^T A(x - x_c) \leq 1 \qquad (A \text{ positive definite})$$

Norm ball: solution of $\|x\| \leq R$ (for any norm)

Positive semidefinite cone: $\mathbf{S}^n_+ = \{X \in \mathbf{S}^n \mid X \succeq 0\}$

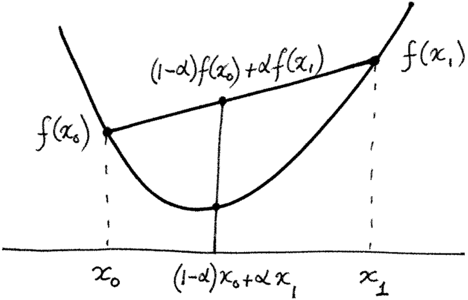
the **intersection** of any number of convex sets is convex

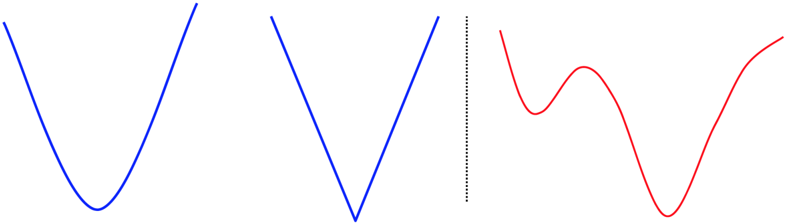
D – a domain in $\mathbb{R}^n$.

A convex function $f : D \to \mathbb{R}$ is one that satisfies, for any $\mathbf{x}_0$ and $\mathbf{x}_1$ in D:

$$f((1-\alpha)\mathbf{x}_0 + \alpha\mathbf{x}_1) \leq (1-\alpha)f(\mathbf{x}_0) + \alpha f(\mathbf{x}_1) \quad .$$

Line joining $(\mathbf{x}_0, f(\mathbf{x}_0))$ and $(\mathbf{x}_1, f(\mathbf{x}_1))$ lies above the function graph.





convex

Not convex

A non-negative sum of convex functions is convex

- linear and affine functions are convex and concave
- $\exp x$, $-\log x$, $x \log x$ are convex
- x^α is convex for $x > 0$ and $\alpha \geq 1$ or $\alpha \leq 0$; $|x|^\alpha$ is convex for $\alpha \geq 1$
- norms are convex
- quadratic-over-linear function $x^T x/t$ is convex in x, t for $t > 0$
- geometric mean $(x_1 x_2 \cdots x_n)^{1/n}$ is concave for $x \geq 0$
- $\log \det X$ is concave on set of positive definite matrices
- $\log(e^{x_1} + \cdots e^{x_n})$ is convex

SVM

$$\min_{\mathbf{w}\in\mathbb{R}^d} C\sum_{i}^{N} \max\left(0, 1 - y_i f(\mathbf{x}_i)\right) + ||\mathbf{w}||^2$$

convex

Gradient (or steepest) descent algorithm for SVM

To minimize a cost function $\mathcal{C}(\mathbf{w})$ use the iterative update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \mathcal{C}(\mathbf{w}_t)$$

where η is the learning rate.

First, rewrite the optimization problem as an average

$$\begin{aligned}\min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) &= \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{N}\sum_i^N \max\left(0, 1 - y_i f(\mathbf{x}_i)\right) \\ &= \frac{1}{N}\sum_i^N \left(\frac{\lambda}{2}||\mathbf{w}||^2 + \max\left(0, 1 - y_i f(\mathbf{x}_i)\right)\right)\end{aligned}$$

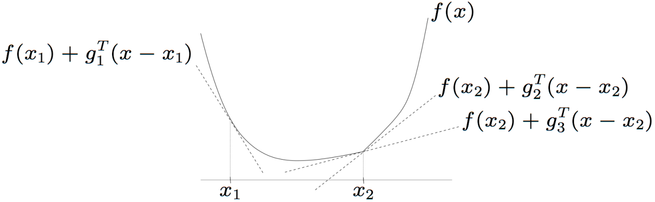
(with $\lambda = 2/(NC)$ up to an overall scale of the problem) and $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$

Because the hinge loss is not differentiable, a sub-gradient is computed

g is a **subgradient** of f (not necessarily convex) at x if

$$f(y) \geq f(x) + g^T(y - x) \quad \text{for all } y$$

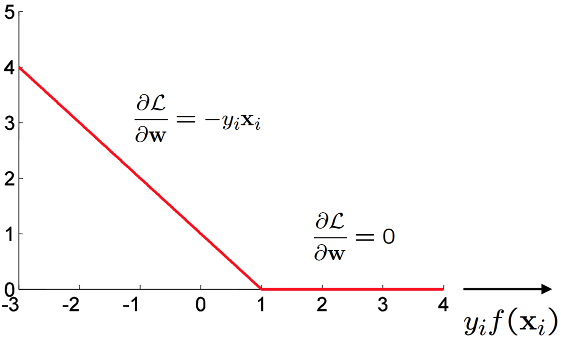
($\iff (g, -1)$ supports **epi** f at $(x, f(x))$)



g_2, g_3 are subgradients at x_2; g_1 is a subgradient at x_1

Prof. S. Boyd, EE392o, Stanford University

$$\mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}) = \max(0, 1 - y_i f(\mathbf{x}_i)) \qquad f(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i + b$$



$$\mathcal{C}(\mathbf{w}) = \frac{1}{N}\sum_{i}^{N}\left(\frac{\lambda}{2}||\mathbf{w}||^2 + \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w})\right)$$

The iterative update is

$$\begin{aligned}\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta \nabla_{\mathbf{w}_t}\mathcal{C}(\mathbf{w}_t) \\ &\leftarrow \mathbf{w}_t - \eta\frac{1}{N}\sum_{i}^{N}(\lambda\mathbf{w}_t + \nabla_{\mathbf{w}}\mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}_t))\end{aligned}$$

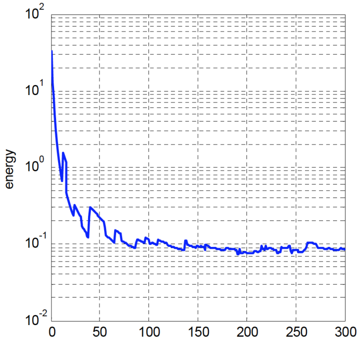
where η is the learning rate.

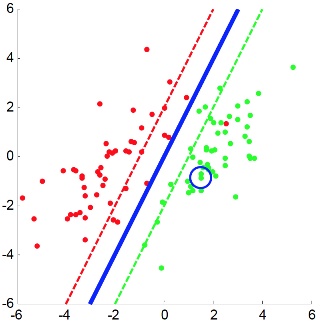
Then each iteration t involves cycling through the training data with the updates:

$$\begin{aligned}\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta(\lambda\mathbf{w}_t - y_i\mathbf{x}_i) && \text{if } y_i f(\mathbf{x}_i) < 1 \\ &\leftarrow \mathbf{w}_t - \eta\lambda\mathbf{w}_t && \text{otherwise}\end{aligned}$$

In the Pegasos algorithm the learning rate is set at $\eta_t = \frac{1}{\lambda t}$

Randomly sample from the training data





Pegasos: Primal Estimated sub-GrAdient SOlver for SVM (ICML 2007)

Advanced issues of Dual form and Kernels of SVM
Detailed duality, please refer to Page 215 – 229, (Chap 5), Stephen Byod et al. “Convex Optimization” 2004, Cambridge University Press

- We have seen that for an SVM learning a linear classifier

$$f(x) = \mathbf{w}^\top \mathbf{x} + b$$

is formulated as solving an optimization problem over $\mathbf{w}$:

$$\min_{\mathbf{w} \in \mathbb{R}^d} ||\mathbf{w}||^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

- This quadratic optimization problem is known as the primal problem.

- Instead, the SVM can be formulated to learn a linear classifier

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i (\mathbf{x}_i{}^\top \mathbf{x}) + b$$

by solving an optimization problem over α_i.

- This is know as the dual problem, and we will look at the advantages of this formulation.

The Representer Theorem states that the solution $\mathbf{w}$ can always be written as a linear combination of the training data:

$$\mathbf{w} = \sum_{j=1}^{N} \alpha_j y_j \mathbf{x}_j$$

Now, substitute for $\mathbf{w}$ in $f(x) = \mathbf{w}^\top \mathbf{x} + b$

$$f(x) = \left(\sum_{j=1}^{N} \alpha_j y_j \mathbf{x}_j \right)^\top \mathbf{x} \underset{\text{Text}}{+} b = \sum_{j=1}^{N} \alpha_j y_j \left(\mathbf{x}_j{}^\top \mathbf{x} \right) + b$$

and for $\mathbf{w}$ in the cost function $\min_{\mathbf{w}} ||\mathbf{w}||^2$ subject to $y_i \left(\mathbf{w}^\top \mathbf{x}_i + b \right) \geq 1, \forall i$

$$||\mathbf{w}||^2 = \left\{ \sum_j \alpha_j y_j \mathbf{x}_j \right\}^\top \left\{ \sum_k \alpha_k y_k \mathbf{x}_k \right\} = \sum_{jk} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j{}^\top \mathbf{x}_k)$$

Hence, an equivalent optimization problem is over α_j

$$\min_{\alpha_j} \sum_{jk} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j{}^\top \mathbf{x}_k) \quad \text{subject to } y_i \left(\sum_{j=1}^{N} \alpha_j y_j (\mathbf{x}_j{}^\top \mathbf{x}_i) + b \right) \geq 1, \forall i$$

and a few more steps are required to complete the derivation.

N is number of training points, and d is dimension of feature vector $\mathbf{x}$.

Primal problem: for $\mathbf{w} \in \mathbb{R}^d$

$$\min_{\mathbf{w}\in\mathbb{R}^d} ||\mathbf{w}||^2 + C\sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

Dual problem: for $\boldsymbol{\alpha} \in \mathbb{R}^N$ (stated without proof):

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2}\sum_{jk} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j{}^\top \mathbf{x}_k) \text{ subject to } 0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

- Need to learn d parameters for primal, and N for dual
- If $N << d$ then more efficient to solve for α than $\mathbf{w}$
- Dual form only involves $(\mathbf{x}_j{}^\top \mathbf{x}_k)$. We will return to why this is an advantage when we look at kernels.

Primal version of classifier:

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

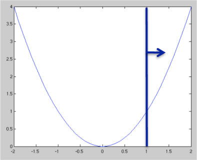
Dual version of classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i (\mathbf{x}_i{}^\top \mathbf{x}) + b$$

At first sight the dual form appears to have the disadvantage of a K-NN classifier – it requires the training data points $\mathbf{x}_i$. However, many of the α_i's are zero. The ones that are non-zero define the support vectors $\mathbf{x}_i$.

OK! Let's prove it by Lagrange multipliers.

Lagrange multipliers – Dual variables



$$\min_x \; x^2$$
$$\text{s.t.} \quad x \geq b$$

Add Lagrange multiplier

Rewrite Constraint

Introduce Lagrangian (objective):

$$L(x, \alpha) = x^2 - \alpha(x - b)$$

We will solve:

$$\min_x \max_\alpha \; L(x, \alpha)$$
$$\text{s.t.} \quad \alpha \geq 0$$

Add new constraint

Why is this equivalent?

- min is fighting max!

$x<b$ → $(x-b)<0$ → $\max_\alpha -\alpha(x-b) = \infty$

- min won't let this happen!

$x>b$, $\alpha \geq 0$ → $(x-b)>0$ → $\max_\alpha -\alpha(x-b) = 0$, $\alpha^*=0$

- min is cool with 0, and $L(x, \alpha)=x^2$ (original objective)

$x=b$ → α can be anything, and $L(x, \alpha)=x^2$ (original objective)

The *min* on the outside forces *max* to behave, so constraints will be satisfied.

Original optimization problem:

$$\text{minimize}_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1, \ \ \forall j$$

Rewrite constraints

One Lagrange multiplier per example

Lagrangian:

$$L(\mathbf{w}, \alpha) = \frac{1}{2}\mathbf{w}.\mathbf{w} - \sum_j \alpha_j \left[\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j - 1\right]$$
$$\alpha_j \geq 0, \ \ \forall j$$

Our goal now is to solve: $\min_{\vec{w},b} \ \max_{\vec{\alpha}\geq 0} \ L(\vec{w}, \vec{\alpha})$

Dual SVM derivation (2) – the linearly separable case

(Primal)
$$\min_{\vec{w},b} \max_{\vec{\alpha}\geq 0} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1 \right]$$

Swap min and max

(Dual)
$$\max_{\vec{\alpha}\geq 0} \min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1 \right]$$

Slater's condition from convex optimization guarantees that these two optimization problems are equivalent!

Dual SVM derivation (3) – the linearly separable case

(Dual) $$\max_{\vec{\alpha}\geq 0} \min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1 \right]$$

Can solve for optimal **w**, b as function of α:

$$\frac{\partial L}{\partial w} = w - \sum_j \alpha_j y_j x_j \quad \rightarrow \quad \mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

$$\frac{\partial L}{\partial b} = -\sum_j \alpha_j y_j \quad \rightarrow \quad \sum_j \alpha_j y_j = 0$$

Substituting these values back in (and simplifying), we obtain:

(Dual) $$\max_{\vec{\alpha}\geq 0,\ \sum_j \alpha_j y_j = 0} \sum_j \alpha_j - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j \left(\vec{x}_i \cdot \vec{x}_j\right)$$

Sums over all training examples — scalars — dot product

Dual SVM derivation (3) – the linearly separable case

(Dual) $$\max_{\vec{\alpha}\geq 0} \min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1\right]$$

Can solve for optimal **w**, b as function of α:

$$\frac{\partial L}{\partial w} = w - \sum_j \alpha_j y_j x_j \quad \rightarrow \quad \mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

$$\frac{\partial L}{\partial b} = -\sum_j \alpha_j y_j \quad \rightarrow \quad \sum_j \alpha_j y_j = 0$$

Substituting these values back in (and simplifying), we obtain:

(Dual) $$\max_{\vec{\alpha}\geq 0,\ \sum_j \alpha_j y_j = 0} \sum_j \alpha_j - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j \left(\vec{x}_i \cdot \vec{x}_j\right)$$

So, in dual formulation we will solve for α directly!

- **w** and b are computed from α (if needed)

Lagrangian:

$$L(\mathbf{w},\alpha) = \frac{1}{2}\mathbf{w}.\mathbf{w} - \sum_j \alpha_j \left[\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j - 1\right]$$

$$\alpha_j \geq 0,\ \ \forall j$$

$\alpha_j > 0$ for some j implies constraint is tight. We use this to obtain b:

$$y_j\left(\vec{w}\cdot\vec{x}_j + b\right) = 1 \quad (1)$$

$$y_j y_j\left(\vec{w}\cdot\vec{x}_j + b\right) = y_j \quad (2)$$

$$\left(\vec{w}\cdot\vec{x}_j + b\right) = y_j \quad (3)$$

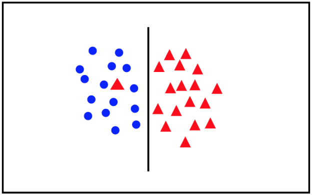
$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$b = y_k - \mathbf{w}.\mathbf{x}_k$$

for any k where $\alpha_k > 0$

Handling data that is not linearly separable

motivation for introducing the dual form of SVM

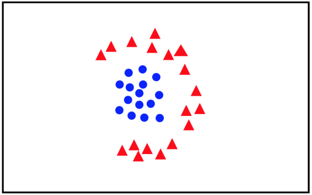


- introduce slack variables

$$\min_{\mathbf{w}\in\mathbb{R}^d, \xi_i\in\mathbb{R}^+} ||\mathbf{w}||^2 + C\sum_i^N \xi_i$$

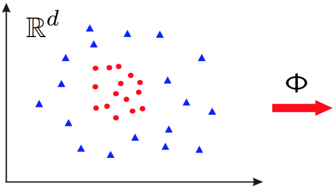
subject to

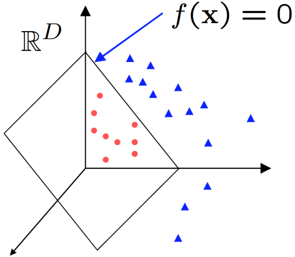
$$y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 - \xi_i \text{ for } i = 1\ldots N$$



- linear classifier not appropriate

??





$\Phi : \mathbf{x} \to \Phi(\mathbf{x}) \quad \mathbb{R}^d \to \mathbb{R}^D$

Learn classifier linear in $\mathbf{w}$ for $\mathbb{R}^D$:

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$$

$\Phi(\mathbf{x})$ is a feature map

Primal Classifier in transformed feature space

Classifier, with $\mathbf{w} \in \mathbb{R}^D$:

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$$

Learning, for $\mathbf{w} \in \mathbb{R}^D$

$$\min_{\mathbf{w} \in \mathbb{R}^D} ||\mathbf{w}||^2 + C \sum_{i}^{N} \max(0, 1 - y_i f(\mathbf{x}_i))$$

- Simply map $\mathbf{x}$ to $\Phi(\mathbf{x})$ where data is separable
- Solve for $\mathbf{w}$ in high dimensional space $\mathbb{R}^D$
- If $D >> d$ then there are many more parameters to learn for $\mathbf{w}$. Can this be avoided?

Classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \mathbf{x}_i^\top \mathbf{x} + b$$

$$\rightarrow f(\mathbf{x}) = \sum_i^N \alpha_i y_i \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}) + b$$

Learning:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \mathbf{x}_j^\top \mathbf{x}_k$$

$$\rightarrow \max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_k)$$

subject to

$$0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

Dual Classifier in transformed feature space

- Note, that $\Phi(\mathbf{x})$ only occurs in pairs $\Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$
- Once the scalar products are computed, only the N dimensional vector $\boldsymbol{\alpha}$ needs to be learnt; it is not necessary to learn in the D dimensional space, as it is for the primal
- Write $k(\mathbf{x}_j, \mathbf{x}_i) = \Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$. This is known as a Kernel

Classifier:

$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i \, k(\mathbf{x}_i, \mathbf{x}) + b$$

Learning:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \, k(\mathbf{x}_j, \mathbf{x}_k)$$

subject to

$$0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and} \sum_i \alpha_i y_i = 0$$

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \to \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \qquad \mathbb{R}^2 \to \mathbb{R}^3$$

$$\begin{aligned} \Phi(\mathbf{x})^\top\Phi(\mathbf{z}) &= (x_1^2, x_2^2, \sqrt{2}x_1x_2) \begin{pmatrix} z_1^2 \\ z_2^2 \\ \sqrt{2}z_1z_2 \end{pmatrix} \\ &= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\ &= (x_1z_1 + x_2z_2)^2 \\ &= (\mathbf{x}^\top\mathbf{z})^2 \end{aligned}$$

Kernel Trick

- Classifier can be learnt and applied without explicitly computing $\Phi(\mathbf{x})$
- All that is required is the kernel $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top\mathbf{z})^2$
- Complexity of learning depends on N

- Linear kernels $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$
- Polynomial kernels $k(\mathbf{x}, \mathbf{x}') = \left(1 + \mathbf{x}^\top \mathbf{x}'\right)^d$ for any $d > 0$
  - Contains all polynomials terms up to degree d
- Gaussian kernels $k(\mathbf{x}, \mathbf{x}') = \exp\left(-||\mathbf{x} - \mathbf{x}'||^2 / 2\sigma^2\right)$ for $\sigma > 0$
  - Infinite dimensional feature space

SVM classifier with Gaussian kernel

N = size of training data

$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b$$

weight (may be zero)

support vector

Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp\left(-||\mathbf{x} - \mathbf{x}'||^2 / 2\sigma^2\right)$

Radial Basis Function (RBF) SVM

$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i \exp\left(-||\mathbf{x} - \mathbf{x}_i||^2 / 2\sigma^2\right) + b$$

Checking if a given function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel can be hard.

- $k(x, \bar{x}) = \tanh(1 + \langle x, \bar{x} \rangle)$?
- $k(x, \bar{x}) = \exp(-$ edit distance between two strings x and $\bar{x}$) ?
- $k(x, \bar{x}) = 1 - \|x - \bar{x}\|^2$?

Easier: construct functions that are garanteed to be kernels:

Construct explicitly:

- any $\phi : \mathcal{X} \to \mathbb{R}^m$ induces a kernel $k(x, \bar{x}) = \langle \phi(x), \phi(\bar{x}) \rangle$.
  in particular any $f : \mathcal{X} \to \mathbb{R}$, $\quad k(x, \bar{x}) = f(x)f(\bar{x})$

Construction from other kernels:

- If k is a kernel and $\alpha \in \mathbb{R}^+$, then $k + \alpha$ and αk are kernels.
- if k_1, k_2 are kernels, then $k_1 + k_2$ and $k_1 \cdot k_2$ are kernels.
- if k is a kernel, then $\exp(k)$ is a kernel.

Kernel algebra

kernel composition	feature composition
a) $k(\mathbf{x},\mathbf{v}) = k_a(\mathbf{x},\mathbf{v}) + k_b(\mathbf{x},\mathbf{v})$	$\boldsymbol{\phi}(\mathbf{x}) = (\boldsymbol{\phi}_a(\mathbf{x}), \boldsymbol{\phi}_b(\mathbf{x})),$
b) $k(\mathbf{x},\mathbf{v}) = f k_a(\mathbf{x},\mathbf{v}),\ f > 0$	$\boldsymbol{\phi}(\mathbf{x}) = \sqrt{f}\boldsymbol{\phi}_a(\mathbf{x})$
c) $k(\mathbf{x},\mathbf{v}) = k_a(\mathbf{x},\mathbf{v})k_b(\mathbf{x},\mathbf{v})$	$\phi_m(\mathbf{x}) = \phi_{ai}(\mathbf{x})\phi_{bj}(\mathbf{x})$
d) $k(\mathbf{x},\mathbf{v}) = \mathbf{x}^T A\mathbf{v}$, A positive semi-definite	$\boldsymbol{\phi}(\mathbf{x}) = L^T\mathbf{x}$, where $A = LL^T$.
e) $k(\mathbf{x},\mathbf{v}) = f(\mathbf{x})f(\mathbf{v})k_a(\mathbf{x},\mathbf{v})$	$\phi(\mathbf{x}) = f(\mathbf{x})\phi_a(\mathbf{x})$

Q: How would you prove that the "Gaussian kernel" is a valid kernel?
A: Expand the Euclidean norm as follows:

$$\exp\left(-\frac{||\vec{u}-\vec{v}||_2^2}{2\sigma^2}\right) = \exp\left(-\frac{||\vec{u}||_2^2}{2\sigma^2}\right)\exp\left(-\frac{||\vec{v}||_2^2}{2\sigma^2}\right)\exp\left(\frac{\vec{u}\cdot\vec{v}}{\sigma^2}\right)$$

Then, apply (e) from above

To see that this is a kernel, use the Taylor series expansion of the exponential, together with repeated application of (a), (b), and (c):

$$e^x = \sum_{n=0}^{\infty}\frac{x^n}{n!}$$

The feature mapping is infinite dimensional!

- Huge feature space with kernels: should we worry about overfitting?
  - SVM objective seeks a solution with large margin
    - Theory says that large margin leads to good generalization.
  - But everything overfits sometimes!!!
  - Can control by:
    - Setting C
    - Choosing a better Kernel
    - Varying parameters of the Kernel (width of Gaussian, etc.)

Appendix–Practical Issues in Machine Learning Experiments

Optimizing the SVM Dual (kernelized)

How to solve the QP

$$\max_{\alpha^1,\dots,\alpha^n \in \mathbb{R}} \quad -\frac{1}{2}\sum_{i,j=1}^{n} \alpha^i \alpha^j y^i y^j k(x^i, x^j) + \sum_{i=1}^{n} \alpha^i$$

subject to $\sum_i \alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$, for $i = 1, \dots, n$.

Observations:

- Kernel matrix K (with entries $k_{ij} = k(x^i, x^j)$) might be too big to fit into memory.
- In the optimum, many of the α_i are 0 and do not contribute. If we knew which ones, we would save a lot of work

Optimizing the SVM Dual (kernelized)

Working set training [Osuna 1997]

1: $S = \emptyset$
2: **repeat**
3: $\alpha \leftarrow$ solve QP with variables α_i for $i \in S$ and $\alpha_i = 0$ for $i \notin S$
4: **for** $i = 1 \ldots, n$ **do**
5: **if** if $i \in S$ and $\alpha_i = 0$ **then** remove i from S
6: **if** if $i \notin S$ and α_i not optimal **then** add i to S
7: **end for**
8: **until** convergence

Advantages:

- objective value increases monotonously
- converges to global optimum

Disadvantages:

- each step is computationally costly, since S can become large

Sequential Minimal Optimization (SMO) [Platt 1998]

```
α ← 0
repeat
    pick index i such that α_i is not optimal
    pick index j ≠ i arbitrarily (usually based on some heuristic)
    α_i, α_j ← solve QP for α_i, α_j and all other α_k fixed
until convergence
```

Advantages:

- convergences monotonously to global optimum
- each step optimizes a subproblem of smallest possible size: 2 unknowns (1 doesn't work because of constraint $\sum_i \alpha_i y_i = 0$)
- subproblems have a closed-form solution

Disadvantages:

- many iterations are required
- many kernel values $k(x^i, x^j)$ are computed more than once (unless K is stored as matrix)

For optimization, the *bias term* is an annoyance

- In primal optimization, it often requires a different stepsize.
- In dual optimization, it is not straight-forward to recover.
- It couples the dual variables by an equality constraint: $\sum_i \alpha_i y_i = 0$.

We can get rid of the bias by the **augmentation trick**.

Original:

- $f(x) = \langle w, x \rangle_{\mathbb{R}^d} + b$, with $w \in \mathbb{R}^d, b \in \mathbb{R}$.

New augmented:

- linear: $f(x) = \langle \tilde{w}, \tilde{x} \rangle_{\mathbb{R}^{d+1}}$, with $\tilde{w} = (w, b)$, $\tilde{x} = (x, 1)$.
- generalized: $f(x) = \langle \tilde{w}, \tilde{\phi}(x) \rangle_{\tilde{\mathcal{H}}}$ with $\tilde{w} = (w, b)$, $\tilde{\phi}(x) = (\phi(x), 1)$.
- kernelize: $\tilde{k}(x, \bar{x}) = \langle \tilde{\phi}(x), \tilde{\phi}(\bar{x}) \rangle_{\tilde{\mathcal{H}}} = k(x, \bar{x}) + 1$.

SVMs Without Bias Term– Optimization

SVM without bias term – primal optimization problem

$$\min_{w \in \mathbb{R}^d, \xi \in \mathbb{R}^n} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi^i$$

subject to, for $i = 1, \ldots, n$,

$$y^i \langle w, x^i \rangle \geq 1 - \xi^i, \qquad \text{and} \qquad \xi^i \geq 0.$$

Difference: no b variable to optimize over

SVM without bias term – primal optimization problem

$$\min_{w\in\mathbb{R}^d,\xi\in\mathbb{R}^n} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi^i$$

subject to, for $i = 1,\dots,n$,

$$y^i\langle w, x^i\rangle \geq 1 - \xi^i, \qquad \text{and} \qquad \xi^i \geq 0.$$

Difference: no b variable to optimize over

SVM without bias term – dual optimization problem

$$\max_{\alpha} \quad -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y^i y^j\, k(x^i, x^j) + \sum_i \alpha_i$$

subject to, $\quad 0 \leq \alpha_i \leq C, \quad$ for $i = 1,\dots,n$.

Difference: no constraint $\sum_i y_i\alpha_i = 0$.

Linear SVM Optimization in the Dual

Stochastic Coordinate Dual Ascent

$\alpha \leftarrow \mathbf{0}$.
for $t = 1, \ldots, T$ **do**
  $i \leftarrow$ random index (uniformly random or in epochs)
  solve QP w.r.t. α_i with all α_j for $j \neq i$ fixed.
end for
return α

Properties:

- converges monotonically to global optimum
- each subproblem has smallest possible size

Open Problem:

- how to make each step efficient?

What's the complexity of the update step? Derive an explicit expression:

Original problem: $\mathbf{max}_{\alpha\in[0,C]^n} \quad -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y^i y^j\, k(x^i, x^j) + \sum_i \alpha_i$

SVM Optimization in the Dual

What's the complexity of the update step? Derive an explicit expression:

Original problem: $\mathbf{max}_{\alpha\in[0,C]^n} \quad -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y^i y^j\, k(x^i,x^j) + \sum_i \alpha_i$

When all α_j except α_i are fixed: $\quad \mathbf{max}_{\alpha_i\in[0,C]} F(\alpha_i)$, with

$$F(\alpha_i) = -\frac{1}{2}\alpha_i^2 k(x^i,x^i) + \alpha_i\Big(1 - y^i\sum_{j\neq i}\alpha_j y^j\, k(x^i,x^j)\Big) + \text{const.}$$

$$\frac{\partial}{\partial\alpha_i}F(\alpha_i) = -\alpha_i k(x^i,x^i) + \Big(1 - y^i\sum_{j\neq i}\alpha_j y^j\, k(x^i,x^j)\Big) + \text{const.}$$

$$\alpha_i^{\text{opt}} = \alpha_i + \frac{1 - y^i\sum_{j=1}^n \alpha_j y^j\, k(x^i,x^j)}{k(x^i,x^i)}, \qquad \alpha_i = \begin{cases} 0 & \text{if } \alpha_i^{\text{opt}} < 0, \\ C & \text{if } \alpha_i^{\text{opt}} > C, \\ \alpha_i^{\text{opt}} & \text{otherwise.} \end{cases}$$

(except if $k(x^i,x^i)=0$, but then $k(x^i,x^j)=0$, so α_i has no influence)

Observation: each update has complexity $O(n)$.

(Generalized) Linear SVM Optimization in the Dual

Let $k(x, \bar{x}) = \langle \phi(x), \phi(\bar{x}) \rangle_{\mathbb{R}^d}$ for explicitly known $\phi : \mathcal{X} \to \mathbb{R}^d$.

$$\alpha_i^{\text{opt}} = \alpha_i + \frac{1 - y^i \sum_j \alpha_j y^j \, k(x^i, x^j)}{k(x^i, x^i)},$$

remember $w = \sum_j \alpha_j y_j \phi(x^j)$

$$= \alpha_i + \frac{1 - y^i \langle w, \phi(x^i) \rangle}{\|\phi(x^i)\|^2},$$

- each update takes $O(d)$, independent of n
  - $\langle w, \phi(x^i) \rangle$ takes at most $O(d)$ for explicit $w \in \mathbb{R}^d, \phi(x^i) \in \mathbb{R}^d$
  - we must also take care that w remains up to date (also at most $O(d)$)

(Generalized) Linear SVM Optimization in the Dual

SCDA for (Generalized) Linear SVMs [Hsieh, 2008]

initialize $\alpha \leftarrow \mathbf{0}$, $w \leftarrow \mathbf{0}$

for $t = 1, \dots, T$ **do**

$\quad i \leftarrow$ random index (uniformly random or in epochs)

$\quad \delta \leftarrow \frac{1 - y^i \langle w, \phi(x^i) \rangle}{\|\phi(x^i)\|^2}$

$$\alpha_i \leftarrow \begin{cases} 0, & \text{if } \alpha_i + \delta < 0, \\ C, & \text{if } \alpha_i + \delta > C, \\ \alpha_i + \delta, & \text{otherwise.} \end{cases}$$

$\quad w \leftarrow w + \delta y^i \phi(x^i)$

end for

return α, w

Properties:

- converges monotonically to global optimum
- complexity of each step is independent of n
- resembles stochastic gradient method, but **automatic step size**

You've trained a new predictor, $g : \mathcal{X} \to \mathcal{Y}$, and you want to tell the world how good it is. How to measure this?

Reminder:

- The average loss on the training set, $\frac{1}{|\mathcal{D}_{trn}|}\sum_{(x,y)\in\mathcal{D}_{trn}} \ell(y, g(x))$ tells us (almost) nothing about the future loss.
  Reporting it would be misleading as best.
- The relevant quantity is the expected risk,

$$\mathcal{R}(g) = \mathbb{E}_{(x,y)\sim p(x,y)}\ \ell(y, g(x))$$

  which unfornately we cannot compute, since $p(x, y)$ is unknown.
- If we have data $\mathcal{D}_{tst} \overset{i.i.d.}{\sim} p(x, y)$, we have,

$$\frac{1}{|\mathcal{D}_{tst}|}\sum_{(x,y)\in\mathcal{D}_{tst}} \ell(y, g(x)) \quad \overset{|\mathcal{D}_{tst}|\to\infty}{\longrightarrow} \quad \mathbb{E}_{(x,y)\sim p(x,y)}\ \ell(y, g(x))$$

- Problem: samples $\ell(y, g(x))$ must me independent, otherwise law of large numbers doesn't hold.
- Make sure that g is independent of $\mathcal{D}_{tst}$.

Classifier Training (idealized)

input training data $\mathcal{D}_{trn}$
input learning procedure A
$g \leftarrow A[\mathcal{D}]$ (apply A with $\mathcal{D}$ as training set)
output resulting classifier $g : \mathcal{X} \to \mathcal{Y}$

Classifier Evaluation

input trained classifier $g : \mathcal{X} \to \mathcal{Y}$
input test data $\mathcal{D}_{tst}$
apply g to $\mathcal{D}_{tst}$ and measure performance R_{tst}
output performance estimate R_{tst}

Classifier Training (idealized)

input training data $\mathcal{D}_{trn}$
input learning procedure A
$g \leftarrow A[\mathcal{D}]$ (apply A with $\mathcal{D}$ as training set)
output resulting classifier $g : \mathcal{X} \to \mathcal{Y}$

Classifier Evaluation

input trained classifier $g : \mathcal{X} \to \mathcal{Y}$
input test data $\mathcal{D}_{tst}$
apply g to $\mathcal{D}_{tst}$ and measure performance R_{tst}
output performance estimate R_{tst}

Remark: In commercial applications, this is realistic:

- given some training set one builds a single system,
- one deploys it to the customers,
- the customers use it on their own data, and complain if disappointed

In research, one typically has no customer, but only a fixed amount of data to work with, so one *simulates* the above protocol.

Classifier Training and Evaluation

input data $\mathcal{D}$
input learning method A
  split $\mathcal{D} = \mathcal{D}_{trn} \dot{\cup} \mathcal{D}_{tst}$ disjointly
  set aside $\mathcal{D}_{tst}$ to a safe place // do not look at it
  $g \leftarrow A[\mathcal{D}_{trn}]$ // learn a predictor from $\mathcal{D}_{trn}$
  apply g to $\mathcal{D}_{tst}$ and measure performance R_{tst}
output performance estimate R_{tst}

Classifier Training and Evaluation

input data $\mathcal{D}$
input learning method A
split $\mathcal{D} = \mathcal{D}_{trn} \dot\cup \mathcal{D}_{tst}$ disjointly
set aside $\mathcal{D}_{tst}$ to a safe place // do not look at it
$g \leftarrow A[\mathcal{D}_{trn}]$ // learn a predictor from $\mathcal{D}_{trn}$
apply g to $\mathcal{D}_{tst}$ and measure performance R_{tst}
output performance estimate R_{tst}

Remark. $\mathcal{D}_{tst}$ should be as small as possible, to keep $\mathcal{D}_{trn}$ as big as possible, but large enough to be convincing.

- sometimes: 50%/50% for small datasets
- more often: 80% training data, 20% test data
- for large datasets: 90% training, 10% test data.

Remark: The split because $\mathcal{D}_{trn}$ and $\mathcal{D}_{tst}$ must be absolute.

- Do not use $\mathcal{D}_{tst}$ for anything except the very last step.
- Do not look at $\mathcal{D}_{tst}$! Even if the learning algorithm doesn't see it, you looking at it can and will influence your model design or parameter selection (human overfitting).
- In particular, this applies to datasets that come with predefined set of test data, such as MNIST, PASCAL VOC, ImageNet, etc.

Remark: The split because $\mathcal{D}_{trn}$ and $\mathcal{D}_{tst}$ must be absolute.

- Do not use $\mathcal{D}_{tst}$ for anything except the very last step.
- Do not look at $\mathcal{D}_{tst}$! Even if the learning algorithm doesn't see it, you looking at it can and will influence your model design or parameter selection (human overfitting).
- In particular, this applies to datasets that come with predefined set of test data, such as MNIST, PASCAL VOC, ImageNet, etc.

In practice we often want more: not just evaluate one classifier, but

- select the best algorithm or parameters amongst multiple ones

We simulate the classifier evaluation step during the training procedure. This needs (at least) one additional data split:

Training and Selecting between Multiple Models

input data $\mathcal{D}$
input set of method $\mathcal{A} = \{A_1, \ldots, A_K\}$
split $\mathcal{D} = \mathcal{D}_{trnval} \dot{\cup} \mathcal{D}_{tst}$ disjointly
set aside $\mathcal{D}_{tst}$ to a safe place (do not look at it)

split $\mathcal{D}_{trnval} = \mathcal{D}_{trn} \dot{\cup} \mathcal{D}_{val}$ disjointly
for all models $A_i \in \mathcal{A}$ **do**
$g_i \leftarrow A_i[\mathcal{D}_{trn}]$
apply g_i to $\mathcal{D}_{val}$ and measure performance $E_{val}(A_i)$
end for
pick best performing A_i

(optional) $g_i \leftarrow A_i[\mathcal{D}_{trnval}]$ // retrain on larger dataset
apply g_i to $\mathcal{D}_{tst}$ and measure performance R_{tst}
output performance estimate R_{tst}

How to split? For example $1/3$–$1/3$–$1/3$ or 70%–10%–20%.

Discussion.

- Each algorithm is trained on $\mathcal{D}_{trn}$ and evaluated on disjoint $\mathcal{D}_{val}$ ✓
- You select a predictor based on E_{val} (its performance on $\mathcal{D}_{val}$), only afterwards $\mathcal{D}_{tst}$ is used. ✓
- $\mathcal{D}_{tst}$ is used to evaluate the final predictor and nothing else. ✓

Discussion.

- Each algorithm is trained on $\mathcal{D}_{trn}$ and evaluated on disjoint $\mathcal{D}_{val}$ ✓
- You select a predictor based on E_{val} (its performance on $\mathcal{D}_{val}$), only afterwards $\mathcal{D}_{tst}$ is used. ✓
- $\mathcal{D}_{tst}$ is used to evaluate the final predictor and nothing else. ✓

Problems.

- small $\mathcal{D}_{val}$ is bad: E_{val} could be bad estimate of g_A's true performance, and we might pick a suboptimal method.
- large $\mathcal{D}_{val}$ is bad: $\mathcal{D}_{trn}$ is much smaller than $\mathcal{D}_{trnval}$, so the classifier learned on $\mathcal{D}_{trn}$ might be much worse than necessary.
- retraining the best model on $\mathcal{D}_{trnval}$ might overcome that, but it comes at a risk: just because a model worked well when trained on $\mathcal{D}_{trn}$, this does not mean it'll also work well when trained on $\mathcal{D}_{trnval}$.

Leave-one-out Evaluation (for a single model/algorithm)

input algorithm A
input loss function ℓ
input data $\mathcal{D}$ (trnval part only: test part set aside earlier)
  for all $(x^i, y^i) \in \mathcal{D}$ **do**
    $g^{\neg i} \leftarrow A[\,\mathcal{D} \setminus \{(x^i, y^i)\}\,]$ // $\mathcal{D}_{trn}$ is $\mathcal{D}$ with i-th example removed
    $r^i \leftarrow \ell(y^i, g^{\neg i}(x^i))$ // $\mathcal{D}_{val} = \{(x^i, y^i)\}$, disjoint to $\mathcal{D}_{trn}$
  end for
output $R_{loo} = \frac{1}{n}\sum_{i=1}^{n} r^i$ (average leave-one-out risk)

Properties.

- Each r^i is a unbiased (but noisy) estimate of the risk $\mathcal{R}(g^{\neg i})$
- $\mathcal{D} \setminus \{(x^i, y^i)\}$ is almost the same as $\mathcal{D}$, so we can hope that each $g^{\neg i}$ is almost the same as $g = A[\mathcal{D}]$.
- Therefore, R_{loo} can be expected a good estimate of $\mathcal{R}(g)$

Problem: slow, trains n times on $n-1$ examples instead of once on n

Compromise: use fixed number of small $\mathcal{D}_{val}$

K-fold Cross Validation (CV)

input algorithm A, loss function ℓ, data $\mathcal{D}$ (trnval part)

split $\mathcal{D} = \dot{\bigcup}_{k=1}^{K} \mathcal{D}_k$ into K equal sized disjoint parts

for $k = 1, \dots, K$ **do**

$g^{\neg k} \leftarrow A[\,\mathcal{D} \setminus \mathcal{D}_k\,]$

$r^k \leftarrow \frac{1}{|\mathcal{D}_k|} \sum_{(x,y)\in\mathcal{D}_k} \ell(y^i, g^{\neg k}(x))$

end for

output $R_{K\text{-CV}} = \frac{1}{K} \sum_{k=1}^{n} r^k$ (K-fold cross-validation risk)

Observation.

- for $K = |\mathcal{D}|$ same as leave-one-out error.
- approximately k times increase in runtime.
- most common: $k = 10$ or $k = 5$.

Problem: training sets overlap, so the error estimates are correlated.
Exception: $K = 2$

5×2 Cross Validation (5×2-CV)

input algorithm A, loss function ℓ, data $\mathcal{D}$ (trnval part)
  for $k = 1, \ldots, 5$ **do**
    Split $\mathcal{D} = \mathcal{D}_1 \dot{\cup} \mathcal{D}_2$
    $g_1 \leftarrow A[\mathcal{D}_1]$,
    $r_1^k \leftarrow$ evaluate g_1 on $\mathcal{D}_2$
    $g_2 \leftarrow A[\mathcal{D}_2]$,
    $r_2^k \leftarrow$ evaluate g_2 on $\mathcal{D}_1$
    $r^k \leftarrow \frac{1}{2}(r_k^1 + r_k^2)$
  end for
output $E_{5\times 2} = \frac{1}{5}\sum_{k=1}^{5} r^k$

Observation.

- 5×2-CV is really the average of 5 runs of 2-fold CV
- very easy to implement: shuffle the data and split into halfs
- within each run the training sets are disjoint and the classifiers g_1 and g_2 are independent

Problem: training sets are smaller than in 5- or 10-fold CV.

Acknowledgement

Some slides are in courtesy of
the slides of C19 Machine Learning lectures by Prof. Andrew Zisserman (Oxford University)