Support vector machines

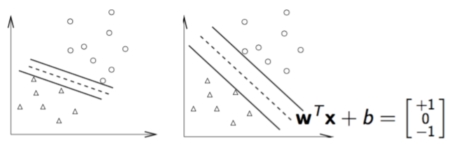
Hung-Hsuan Chen

Many are taken from Prof. C.-J. Lin's and J. Leskovec's slides

(Linear) support vector classification

- Data point *i*: $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})$
- Class label of *i*: y_i
  - Two classes
  - Class 1: $y_i = 1$
  - Class 2: $y_i = -1$
- Find a hyperplane to separate the data points

Assume the dataset is linearly separable



- A hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$

  $\mathbf{w}^T\mathbf{x}_i + b \geq 1$ if $y_i = 1$

  $\mathbf{w}^T\mathbf{x}_i + b \leq -1$ if $y_i = -1$
- Discriminant function $f(\mathbf{x}) = \text{sgn}(\mathbf{w}^T\mathbf{x} + b)$
  - There are **many different choices** of **w** and b

Margin distance

- Given two parallel hyperplanes H_1 and H_2

  $H_1 : \mathbf{w}^T\mathbf{x} = b_1$

  $H_2 : \mathbf{w}^T\mathbf{x} = b_2$
- The distance between H_1 and H_2 is

$$d(H_1, H_2) = \frac{|b_1 - b_2|}{||\mathbf{w}||_2}$$

- Distance between $\mathbf{w}^T\mathbf{x}_i + b = 1$ and $\mathbf{w}^T\mathbf{x}_i + b = -1$:

$$\text{margin} = \frac{2}{||\mathbf{w}||_2}$$

Maximum margin

- $\mathbf{w}, b = \text{argmax}_{\mathbf{w},b} \dfrac{2}{||\mathbf{w}||_2}$
- This is the same as

  $\mathbf{w}, b = \text{argmin}_{\mathbf{w},b} \dfrac{1}{2}\mathbf{w}^T\mathbf{w}$
- This is modeled as a **quadratic programming** problem

  $\min_{\mathbf{w},b} \dfrac{1}{2}\mathbf{w}^T\mathbf{w}$

  Subject to $y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 \ \forall i$

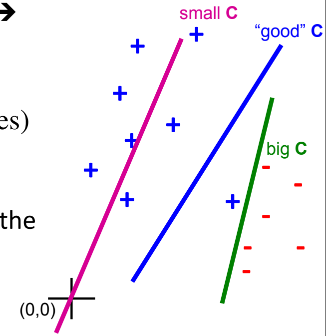
Non-linearly separable dataset

- If non-linearly separable ➔ introduce penalty

  $\min_{\mathbf{w},b} \dfrac{1}{2}\mathbf{w}^T\mathbf{w} + C(\text{\# of mistakes})$

  – If $C \to \infty$: allows no error

  – If $C = 0$: basically ignores the data at all



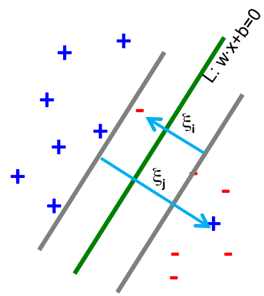
Introduce slack variable

- Not all mistakes are equally bad

  $\min_{\mathbf{w},b} \dfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi_i$

  Subject to

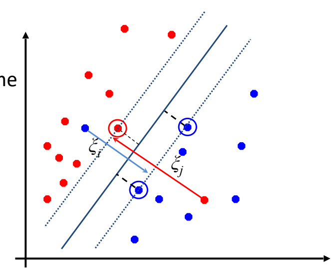
  $y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 - \xi_i \ \forall i$
- If a point is on the wrong side ➔ get penalty ξ_i



For each data point x:
If d(x, L) ≥ 1 and at the right side: don't care
Else: pay linear penalty

Soft margin classification

- Why soft margin
  - The training data may not be linearly separable
  - Even if the training data is linearly separable, allowing some error may increase the margin
- Essentially, there are two objectives (which may against each other)
  - Minimize the training error
    - Prevent error
  - Maximize the margin
    - Prevent overfitting (allow some error)



Soft margin classification formula

- Original (linear) formula

$$\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

Subject to

$$y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 \ \forall i$$

- New formula

$$\min_{\mathbf{w},b} \left(\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i} \xi_i \right)$$

Subject to

$$y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 - \xi_i$$

and $\xi_i > 0 \ \forall i$

- *C*: control overfitting
  - A large *C* makes most ξ_i's to zero
- ξ_i: slack variables

Linear SVM with soft margin

- Linear SVM

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

Subject to

$$y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 - \xi_i \ \forall i$$

- This is the same as

$$\min_{\mathbf{w},b} \left(\underbrace{\frac{1}{2}\mathbf{w}^T\mathbf{w}}_{\text{Margin inverse}} + C\underbrace{\sum_{i=1}^{n} \max\left\{0, \overbrace{1 - y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)}^{\text{If the point is at the wrong side, get loss proportional to } \xi_i}\right\}}_{\text{Empirical loss L (how well we fit training data)}} \right)$$

Regularization parameter

Derivatives

$$f(\mathbf{w}, b) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} \max\left\{0, 1 - y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)\right\}$$

$$\Rightarrow \nabla_{w_j} f = w_j + C\sum_{i=1}^{n} \frac{\partial \max\left\{0, 1 - y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)\right\}}{\partial w_j} = \begin{cases} w_j & \text{if } y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 \\ w_j + C\left(-y_i x_{ij}\right) & \text{else} \end{cases}$$

Solve Linear SVM by GD

Note: *b* is batch size

```
While (true) {
  for (j=1,2, ..., d){
```

$$\nabla_{w_j} f\left(\mathbf{x}_{1:b}\right) = w_j + C\sum_{i=1}^{b} \frac{\partial \max\left\{0, 1 - y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)\right\}}{\partial w_j}$$

$$w_j = w_j - \alpha \nabla_{w_j} f$$

```
  }

  if (w converges) break
}
```

Solve Linear SVM by SGD

```
for (i=1,2, ..., n){
  for (j=1,2, ..., d){
```

$$\nabla_{w_j} f(\mathbf{x}_i) = w_j + C \frac{\partial \max\left\{0, 1 - y_i\left(\mathbf{w}^T \mathbf{x}_i + b\right)\right\}}{\partial w_j}$$

$$w_j = w_j - \alpha \nabla_{w_j} f$$

```
  }

  if (w converges) break
}
```


- Detour
  - Lagrange multiplier
  - KKT condition
- Math caution!
  - If you get lost, I hope you at least understand the linear SVM

Generalized Lagrange multiplier

- Standard form problem

  Minimize $f(\mathbf{x})$ subject to $g_i(\mathbf{x}) \leq 0 \; (i = 1, \dots, p)$

  and $h_j(\mathbf{x}) = 0 \; \left(j = 1, \dots, m\right)$
- Lagrangian

$$\mathscr{L}\left(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}\right) = f(\mathbf{x}) + \sum_{i=1}^{p} \lambda_i g_i(\mathbf{x}) + \sum_{j=1}^{m} \mu_j h_j(\mathbf{x})$$

The characteristic of the solution

SVM (ignore slack variables):

$$\min_{\mathbf{w}, b} \frac{1}{2} \mathbf{w}^T \mathbf{w}$$

Subject to

$$y_i\left(\mathbf{w}^T \mathbf{x}_i + b\right) \geq 1 \;\; \forall i$$

$$\mathscr{L}\left(\mathbf{w}, b, \boldsymbol{\lambda}\right) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum \lambda_i \left[1 - y_i\left(\mathbf{w}^T \mathbf{x}_i + b\right)\right]$$

- No equality constraints (no μ's)

- Based on the KKT condition: if $\mathbf{w}^*$ is the optimal solution to the standard form problem, then there exist KKT multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ such that
  - Lagrangian optimality

    $\nabla \mathscr{L}\left(\mathbf{w}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}\right) = 0 ------ \quad (1)$
  - Primal feasibility

    $g_i\left(\mathbf{w}^*\right) \leq 0 \;\; \forall i ------- \quad (2)$

    $h_j(\mathbf{w}^*) = 0 \;\; \forall j ------- \quad (3)$
  - **Dual feasibility**

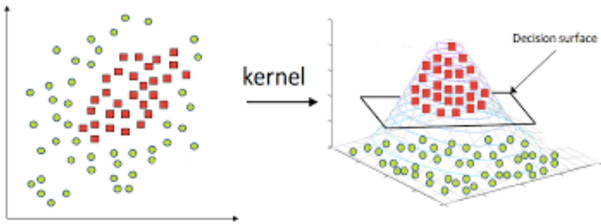
    $\lambda_i \geq 0 \;\; \forall i --------- \quad (4)$
  - **Complementary slackness**

    $\lambda_i g_i\left(\mathbf{w}^*\right) = 0 \;\; \forall i ------ \quad (5)$

$\mathbf{w}^T \mathbf{x}_i + b = 1$ and $\mathbf{w}^T \mathbf{x}_i + b = -1$

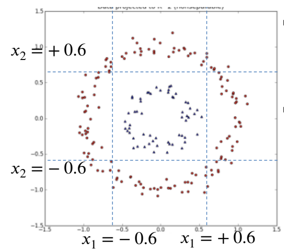
- Assume linearly-separable, by condition (4) and (5):
  - If a training instance $\mathbf{x}_i$ is ***not*** on the two hyperplanes (i.e., $g_i\left(\mathbf{w}^*\right) < 0$), λ_i must be 0
  - If a training instance $\mathbf{x}_i$ is on the two hyperplanes (i.e., $g_i\left(\mathbf{w}^*\right) = 0$), $\lambda_i \geq 0$

Map features to higher dimensional



- Transform the data into a higher dimension feature space so that linear separation is possible
  - Higher dimensional (**could be infinite**) feature space
    - $\phi(\mathbf{x}_i) = [\phi_1(\mathbf{x}_i), \phi_2(\mathbf{x}_i), \dots]^T$

Example



- The positive and negative examples are not linearly- separable by the 2D features $(x_1, \ x_2)$
- If we add one more feature $x_3 = x_1^2 + x_2^2$, the blue points are those with $x_3 \le 0.6^2$, and the red points are those with $x_3 > 0.6^2$
  - $\phi(x_1, x_2) = (x_1, x_2, x_3) = (x_1, x_2, x_1^2 + x$
    - 2D to 3D
  - The points become linearly separable

Kernel SVM

Here we ignore the slack variables for simplicity

$$\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

Subject to $y_i\left(\mathbf{w}^T\phi(\mathbf{x}_i) + b\right) \ge 1$

- Linear SVM: length of **w** is *d* (the same as the size of $\mathbf{x}_i$)
- Kernel SVM: length of **w** is larger than *d* (the same as the size of $\phi(\mathbf{x}_i)$)
  - Kernel SVM can fit a more complex function
    - The size of $\phi(\mathbf{x}_i)$ is large (and could be ***infinity***)
  - How to efficiently compute **w** and $\mathbf{w}^T\phi(\mathbf{x}_i)$?
  - How to store **w**?

Lagrangian of kernel SVM

$$\mathscr{L}(\mathbf{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum \lambda_i\left[1 - y_i\left(\mathbf{w}^T\phi(\mathbf{x}_i) + b\right)\right]$$

$$\begin{cases} \nabla_{\mathbf{w}}\mathscr{L}(\mathbf{w}, b, \boldsymbol{\lambda}) = \mathbf{w} - \sum \lambda_i y_i \phi(\mathbf{x}_i) := 0 \\ \nabla_b \mathscr{L}(\mathbf{w}, b, \boldsymbol{\lambda}) = -\sum \lambda_i y_i := 0 \end{cases}$$

$$\Rightarrow \begin{cases} \mathbf{w} = \sum \lambda_i y_i \phi(\mathbf{x}_i) \\ \sum \lambda_i y_i := 0 \end{cases}$$

- Given a test instance $\mathbf{x}_t$, the discriminant function is

$$f(\mathbf{x}_t) = \mathbf{w}^T\phi(\mathbf{x}_t) + b = \sum \lambda_i y_i \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_t) + b$$

  - Prediction is a **linear combination of training instances** $\mathbf{w}^T\phi(\mathbf{x}_t)$ plus bias

High dimensional mapping example

$$f(\mathbf{x}_t) = \mathbf{w}^T\phi(\mathbf{x}_t) + b = \sum \lambda_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_t) + b$$

- Example:
  - $\mathbf{x}_i = [x_{i1}, x_{i2}]^T \in R^2, \phi(\mathbf{x}_i) \in R^6$
  - If we set
    $\phi(\mathbf{x}_i) = [1, \sqrt{2}x_{i1}, \sqrt{2}x_{i2}, \sqrt{2}x_{i1}x_{i2},\ x_{i1}^2, x_{i2}^2]^T$
  - Then
    $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_t) = 1 + x_{i1}^2x_{t1}^2 + x_{i2}^2x_{t2}^2 + 2x_{i1}x_{t1} + 2x_{i2}x_{t2} + 2x_{i1}x_{t1}x_{i2}x$
  - When the target dimension is large, it is inefficient to generate $\phi(\mathbf{x}_t)$ and $\phi(\mathbf{x}_i)$ $\forall i$ and perform the dot product

Kernel trick example

- If
  $\phi(\mathbf{x}_i) = [1, \sqrt{2}x_{i1}, \sqrt{2}x_{i2}, \sqrt{2}x_{i1}x_{i2},\ x_{i1}^2, x_{i2}^2]^T$,
  then:
  - $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_t) = (1 + \mathbf{x}_i^T\mathbf{x}_T)^2$
  - Computing $(1 + \mathbf{x}_i^T\mathbf{x}_T)^2$ is much more efficient than computing $\phi(\mathbf{x}_i)$, $\phi(\mathbf{x}_t)$, and then $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_t)$

Kernel trick example

$$f(\mathbf{x}_t) = \mathbf{w}^T\phi(\mathbf{x}_t) + b = \sum \lambda_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_t) + b$$

- If $\phi(\mathbf{x}_i)$'s dimension is very high
  - Store $\mathbf{w}$ is costly
  - Compute discriminant function $f(\mathbf{x}_t) = \mathbf{w}^T\phi(\mathbf{x}_t) + b$ is costly
- We may use $(1 + \mathbf{x}_i^T\mathbf{x}_t)^2$ to efficiently map features to higher dimension
- We compute
  $\sum \lambda_i y_i \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_t) + b = \sum \lambda_i y_i (1 + \mathbf{x}_i^T\mathbf{x}_t)^2 + b$ as the discriminant function

Popular kernels

- Linear kernel (i.e., linear SVM)
  $K(\mathbf{x}_i, \mathbf{x}_t) = \mathbf{x}_i^T\mathbf{x}_t = \langle \mathbf{x}_i, \mathbf{x}_t \rangle$
- Polynomial kernel
  $K(\mathbf{x}_i, \mathbf{x}_t) = \left(\langle \mathbf{x}_i, \mathbf{x}_t \rangle + r\right)^d,\ r > 0$
- Gaussian (RBF) kernel
  $K(\mathbf{x}_i, \mathbf{x}_t) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_t\|^2\right)$
- The dimension of $K(\mathbf{x}_i, \mathbf{x}_t)$ could be ***infinity*** (e.g., RBF kernel), but the dimensions of $\mathbf{x}_i$ and $\mathbf{x}_t$ are finite

Mapping to infinite dimensional

- Assume $\mathbf{x}_i \in R^1,\ \gamma > 0$

$$\exp\left(-\gamma\|\mathbf{x}_i - \mathbf{x}_t\|^2\right) = \exp\left(-\gamma(\mathbf{x}_i - \mathbf{x}_t)^2\right) = \exp\left(-\gamma \ldots\right)$$

$$= \exp\left(-\gamma\mathbf{x}_i^2 - \gamma\mathbf{x}_j^2\right)\cdot\exp\left(2\gamma\mathbf{x}_i\mathbf{x}_j\right)$$

$$= \exp\left(-\gamma\mathbf{x}_i^2 - \gamma\mathbf{x}_j^2\right)\left(1 + \frac{2\gamma\mathbf{x}_i\mathbf{x}_j}{1!} + \frac{\left(2\gamma\mathbf{x}_i\mathbf{x}_j\right)^2}{2!} + \frac{\left(2\gamma\mathbf{x}_i\mathbf{x}_j\right)^3}{3!} + \ldots\right)$$

$$= \exp\left(-\gamma\mathbf{x}_i^2 - \gamma\mathbf{x}_j^2\right)\left(1\cdot 1 + \sqrt{\frac{2\gamma}{1!}}\mathbf{x}_i\sqrt{\frac{2\gamma}{1!}}\mathbf{x}_j + \sqrt{\frac{(2\gamma)^2}{2!}}\mathbf{x}_i^2\sqrt{\frac{(2\gamma)^2}{2!}}\mathbf{x}_j^2 + \sqrt{\frac{(2\gamma)^3}{3!}}\mathbf{x}_i^3\sqrt{\frac{(2\gamma)^3}{3!}}\mathbf{x}_j^3 + \ldots\right)$$

$$= \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j),$$

Where $\phi(\mathbf{x}_i) = \exp(-\gamma\mathbf{x}_i^2)\left[1, \sqrt{\frac{2\gamma}{1!}}\mathbf{x}_i,\ \sqrt{\frac{(2\gamma)^2}{2!}}\mathbf{x}_i^2,\ \sqrt{\frac{(2\gamma)^3}{3!}}\mathbf{x}_i^3,\ \ldots\right]^T$

By Taylor expansion:

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

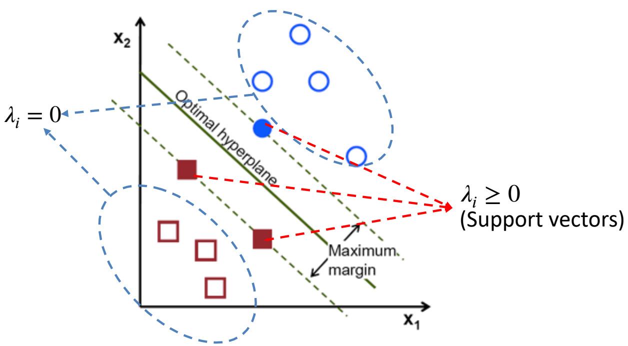
Characteristics of the solution

- Discriminant function

$$f(\mathbf{x}_t) = \mathbf{w}^T\phi(\mathbf{x}_t) + b = \sum \lambda_i y_i \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_t) + b = \sum \lambda_i y_i K(\mathbf{x}_i, \mathbf{x}_t) + b$$

- Many λ_i's are 0
  - Memorizing training instance $(\mathbf{x}_i, y_i)$ only if $\lambda_i > 0$
  - We don't need to form **w** explicitly
- To predict the label of a test instance $\mathbf{x}_t$, we need to compute the ***Kernel*** of the test instance with the training instances **whose λ_i's are larger than zeros**
  - These training instances are called "**support vectors**"

Visualizing "support vectors"



How to obtain the variables in the discriminant function?

$$f(\boldsymbol{x}_t) = \boldsymbol{w}^T\phi(\boldsymbol{x}_t) + b = \sum \lambda_i y_i \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_t) + b = \sum \lambda_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}_t) + b$$

- y_i: training labels (given)
- x_i: training features (given)
- x_t: features of the data point you want to test (given)
- $K(x_i, x_t)$: kernel function, e.g., $(x_i^T x_t + 1)^2$ (can be computed)
- λ_i: unknown (although we know most of them are 0 by the Complementary slackness in KKT condition)
- b: unknown
- How to obtain λ_i and b?

Quadratic programming

- SVM:

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

Subject to $y_i\left(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b\right) \geq 1$

- Transform into quadratic programming form
  - Let $\boldsymbol{u} = \begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix}$, $\boldsymbol{Q} = \begin{bmatrix} 0 & \boldsymbol{0}_{1\times\tilde{d}} \\ \boldsymbol{0}_{\tilde{d}\times 1} & \boldsymbol{I}_{\tilde{d}\times\tilde{d}} \end{bmatrix}$, $\boldsymbol{p} = \begin{bmatrix} 0_{(\tilde{d}+1)\times 1} \end{bmatrix}$
    $\boldsymbol{a}_i^T = y_i\begin{bmatrix} 1 & \phi(\boldsymbol{x}_i) \end{bmatrix}^T$, $\boldsymbol{c}_i = 1$
    - $\tilde{d}$ is the size of $\phi(\boldsymbol{x}_i)$
  - Solve $\boldsymbol{u} = \mathrm{QP}(\boldsymbol{Q}, \boldsymbol{p}, \boldsymbol{A}, \boldsymbol{C})$ by a QP solver
    - Details are ignored
- This involves $\tilde{d} + 1$ unknowns (b and $\boldsymbol{w}$) and n constraints
  - Still challenging when $\tilde{d}$ is large

QP solver format

$$\min_{\boldsymbol{u}} \frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{p}^T\boldsymbol{u}$$

Subject to $\boldsymbol{a}_m^T\boldsymbol{u} \geq c_m$

Primal form (1/2)

- SVM standard form

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

Subject to $y_i\left(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b\right) \geq 1$

- The standard form problem re-formulate as the **Primal problem**

$$\min_{\boldsymbol{w}} \max_{\lambda_i \geq 0,\, \boldsymbol{\mu}} \mathscr{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu})$$

Primal form (2/2)

- Why?

$$\begin{aligned}
&\max_{\lambda_i \geq 0,\, \boldsymbol{\mu}} \mathscr{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \\
&= \max_{\lambda_i \geq 0,\, \boldsymbol{\mu}} \left[f(\boldsymbol{w}) + \sum_{i=1}^{p} \lambda_i g_i(\boldsymbol{w}) + \sum_{j=1}^{m} \mu_j h_j(\boldsymbol{w}) \right] \\
&= \max_{\lambda_i \geq 0,\, \boldsymbol{\mu}} \left[f(\boldsymbol{w}) + \sum_{i=1}^{p} \lambda_i g_i(\boldsymbol{w}) \right] \left(\because h_j(\boldsymbol{w}) = 0 \right) \\
&= f(\boldsymbol{w}) \left(\because g_i(\boldsymbol{w}) \leq 0 \right)
\end{aligned}$$

$$\Rightarrow \min_{\boldsymbol{w}} \max_{\lambda_i \geq 0,\, \boldsymbol{\mu}} \mathscr{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \min_{\boldsymbol{w}} f(\boldsymbol{w})$$

Primal vs dual problem

- Primal problem: $p^* = \min_{\boldsymbol{w}} \max_{\lambda_i \geq 0,\, \boldsymbol{\mu}} \mathscr{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu})$
- Dual problem: $d^* = \max_{\lambda_i \geq 0,\, \boldsymbol{\mu}} \min_{\boldsymbol{w}} \mathscr{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu})$
- $p^* \geq d^*$
  - The min of the max is no less than the max of the min
  - Duality gap: $p^* - d^*$
  - If $p^* = d^*$, we may solve the dual instead of the primal problem

standard kernal SVM

primal form

dual form

Strong duality

- $d^* = p^*$
- If the following conditions are true, then strong duality holds
  1. f and g_i's are convex
  2. h_i's are linear functions (i.e., Exists a_i and b_i such that $h_i(\boldsymbol{x}) = a_i^T \boldsymbol{w} + b_i$)
  3. Exists some $\boldsymbol{w}$ such that $g_i(\boldsymbol{w}) \leq 0$
- In SVM, the above conditions holds
  - We may solve the dual problem instead of the primal problem

Primal and dual problem in SVM

- Primal:

$$p^* = \min_{\boldsymbol{w}} \max_{\lambda_i \geq 0} \mathcal{L}(\boldsymbol{w}, \lambda)$$

We use $\boldsymbol{x}_i$ below for simplicity, but $\boldsymbol{x}_i$ can be replaced by $\phi(\boldsymbol{x}_i)$

$$\begin{cases} \boldsymbol{w} = \sum \lambda_i y_i \boldsymbol{x}_i \\ \sum \lambda_i y_i := 0 \end{cases}$$

- Dual:

$$d^* = \max_{\lambda \geq 0} \min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \lambda) = \max_{\lambda \geq 0} \min_{\boldsymbol{w}} \left(\frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} + \sum \lambda_i \left[1 - y_i (\boldsymbol{w}^T \boldsymbol{x}_i + b) \right] \right) = \max_{\lambda \geq 0} \min_{\boldsymbol{w}} \left(\frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} + \sum \lambda_i - \boldsymbol{w}^T \sum \lambda_i y_i \boldsymbol{x}_i - b \sum \lambda_i y_i \right) = \max_{\lambda \geq 0} \min_{\boldsymbol{w}} \left(\frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} + \sum \lambda_i - \boldsymbol{w}^T \boldsymbol{w} - b0 \right) = \max_{\lambda \geq 0} \min_{\boldsymbol{w}} \left(-\frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} + \sum \lambda_i \right) = \max_{\lambda \geq 0} \min_{\boldsymbol{w}} \left(-\frac{1}{2} \left\| \sum \lambda_i y_i \boldsymbol{x}_i \right\|^2 + \sum \lambda_i \right) = \max_{\lambda \geq 0} \left(-\frac{1}{2} \left\| \sum \lambda_i y_i \boldsymbol{x}_i \right\|^2 + \sum \lambda_i \right) = \min_{\lambda \geq 0} \left(\frac{1}{2} \sum_{p=1}^{n} \sum_{q=1}^{n} \lambda_p \lambda_q y_p y_q \boldsymbol{x}_p^T \boldsymbol{x}_q - \sum \lambda_i \right)$$

The dual problem

QP solver format

$$\min_{\boldsymbol{u}} \frac{1}{2} \boldsymbol{u}^T Q \boldsymbol{u} + \boldsymbol{p}^T \boldsymbol{u}$$

Subject to $\boldsymbol{a}_m^T \boldsymbol{u} \geq c_m$

- Dual form

$$\min_{\lambda_i \geq 0} \left[\frac{1}{2} \sum_{p=1}^{n} \sum_{q=1}^{n} \lambda_p \lambda_q y_p y_q \boldsymbol{x}_p^T \boldsymbol{x}_q - \sum \lambda_i \right]$$

Subject to $\lambda_i \geq 0$ and $\sum \lambda_i y_i = 0$

- This is a quadratic programming (QP) problem
- This involves n unknowns (λ_is) and $n+1$ constraints

Primal vs dual

Primal

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

Subject to $y_i \left(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) + b \right) \geq 1$

- $\tilde{d} + 1$ unknowns
- n constraints

Dual

$$\min_{\lambda_i \geq 0} \left[\frac{1}{2} \sum_{p=1}^{n} \sum_{q=1}^{n} \lambda_p \lambda_q y_p y_q \phi(\boldsymbol{x}_p)^T \phi(\boldsymbol{x}_q) - \sum \lambda_i \right]$$

Subject to $\lambda_i \geq 0$ and $\sum \lambda_i y_i = 0$

- n unknowns
- $n+1$ constraints

- If $\tilde{d} \ll n$, use primal; otherwise use dual
- When using RBF kernel, $\tilde{d} = \infty$, we can only use dual

if use kernel ① ker trick ② convert to dual

A numerical example

- Training data: five 1D data points with labels
  - First class (+1): $x_{1,1} = 1,\ x_{2,1} = 2,\ x_{5,1} = 6$
  - Second class (-1): $x_{3,1} = 4,\ x_{4,1} = 5$
- Non-linearly separable
- Use polynomial kernel with degree 2

$$K(\mathbf{x}_i, \mathbf{x}_t) = \left(\langle \mathbf{x}_i, \mathbf{x}_t \rangle + 1\right)^2$$

- Solve $\lambda_i\ (i = 1, \ldots, 5)$ by a standard QP solver
  - $[\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5] = [0, 2.5,\ 0, 7.333,\ 4.833]$

2

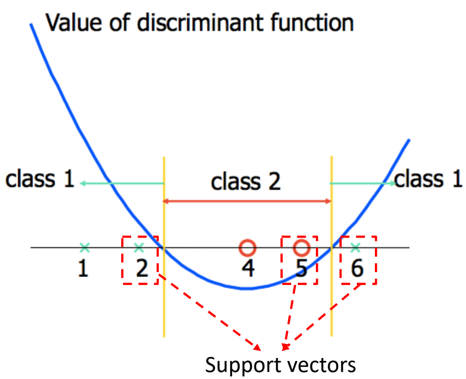
form

The discriminant function of the example

λ_i	x_i	y_i
1	0	1	1
2	2.5	2	1
3	0	4	-1
4	7.5	5	-1
5	4.83	6	1
6		z	□

- $[\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5] = [0, 2.5,\ 0, 7.333,\ 4.833]$
  - Since $\lambda_1 = \lambda_3 = 0$, the support vectors are $(x_2,\ x_4,\ x_5) = (2, 5, 6)$
- The discriminant function
  - $f(z) = \sum \lambda_i y_i \phi(x_i)^T \phi(z) + b$
    $= \left[2.5(1)(2z+1)^2 + 7.333(-1)(5z+1)^2 + 4.833(1)(6z+1)^2\right] + b$
    $= 0.6667z^2 - 5.333z + b$
  - Since $f(x_{2,1}) = f(x_{5,1}) = 1$ and $f(x_{4,1}) = -1$, we can get b=9
  - $f(z) = 0.6667z^2 - 5.333z + 9$

Visualize the discriminant function



Standard SVM

- Standard form

$$\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i$$

Subject to (1) $y_i\left(\mathbf{w}^T\phi(\mathbf{x}_i) + b\right) \geq 1 - \xi$

(2) $\xi_i > 0\ \forall i$

Review of SVM

- Large margin
  - Prevent overfitting
- Soft margin
  - Make the margin become larger
  - Prevent overfitting
- Kernel trick
  - Make the data linearly separable
  - Efficiently compute the inner product of "high-dimensional" features $\phi(\boldsymbol{x}_i)^T\phi\left(\boldsymbol{x}_j\right)$
- Primal vs dual
  - # unknowns goes from $\tilde{d}+1$ to n
  - Use dual if $\tilde{d} \gg n$

Revisiting logistic regression and SVM from another perspective

Deriving SVM and LogReg from regularized linear classification

- We derived SVM from the viewpoint of maximal margin
- We derived logistic regression from maximizing the log-likelihood (or minimizing the cross entropy loss)
- However, both can be considered from the viewpoint of ***regularized linear classification***

Regularized linear classification

- Training data:
  $\left\{\mathbf{x}_i, y_i\right\}_{i=1,\ldots,n}$, $\mathbf{x_i} \in R^d, y_i \in \{\pm 1\}$
- Objective

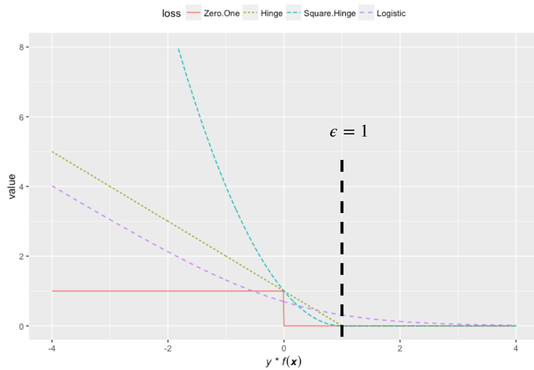
$$\min_{\mathbf{w}} f(\mathbf{w}),\ f(\mathbf{w}) \equiv \frac{\mathbf{w}^T\mathbf{w}}{2} + C\sum_{i=1}^{n}\xi\left(\mathbf{w};\mathbf{x}_i, y_i\right)$$

  - $\xi\left(\mathbf{w};\mathbf{x}_i, y_i\right)$: loss function; we hope $y_i\mathbf{w}^T\mathbf{x} > 0$
    - Trying to fit the training data
  - $\left(\mathbf{w}^T\mathbf{w}\right)/2$: regularization term
    - We skip the L1 regularization term here
    - Prevent over-fit the training data
  - *C*: regularization parameter

Loss functions

- Common loss functions in classification
  - Hinge loss
    - $\xi_{L1}(\mathbf{w}; \mathbf{x}_i, y_i) = \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)$
  - Squared hinge loss
    - $\xi_{L2}(\mathbf{w}; \mathbf{x}_i, y_i) = \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)^2$
  - Logistic loss
    - $\xi_{LR}(\mathbf{w}; \mathbf{x}_i, y_i) = \log\left(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}\right)$
      - This is different from what we derived previously
        - » We used 1/0 to encode two classes before, but here we use +1/-1
- SVM: ξ_{L1}
- Logistic Regression: ξ_{LR}

Visualizing the loss functions



Regularization

L1

- $\|\boldsymbol{w}\|_1$
- Non-differentiable
- Sparse solution; possibly many zeros
  - Feature selection
  - Less storage of $\boldsymbol{w}$

L2

- $\boldsymbol{w}^T \boldsymbol{w}/2$
- Smooth; easier to optimize

Regularized linear classification with kernel

- Training data:

  $\{\mathbf{x}_i, y_i\}_{i=1,\ldots,m}, \ \mathbf{x_i} \in R^n, y_i \in \{\pm 1\}$
- Objective

  $$\min_{\mathbf{w}} f(\mathbf{w}),\ f(\mathbf{w}) \equiv \frac{\mathbf{w}^T\mathbf{w}}{2} + C \sum_{i=1}^{m} \xi\left(\mathbf{w}; \phi(\mathbf{x}_i), y_i\right)$$

  - $\xi\left(\mathbf{w}; \phi(\mathbf{x}_i), y_i\right)$: loss function; we hope $y_i \mathbf{w}^T \phi(\mathbf{x}_i) > 0$
    - Trying to fit the training data
  - $\mathbf{w}^T\mathbf{w}/2$: regularization term
    - We skip the L1 regularization term here
    - Prevent over-fit the training data
  - C: regularization parameter

Logistic regression vs SVM

- Logistic regression and SVM are very related
- Their performance (i.e., test accuracy) is usually similar
- Due to the naming, the typical deriving process, and historical reasons, many believe that SVM and logistic regression are very different
  - This is a misunderstanding

Linear or kernel? SVM or Logistic Regression?

- When people say SVM, they typically mean "**kernel** SVM"
  - But there is linear SVM

$$\min_{\mathbf{w}} f(\mathbf{w}),\ f(\mathbf{w}) \equiv \frac{\mathbf{w}^T\mathbf{w}}{2} + C\sum_{i=1}^{m} \max\left(0,\ 1 - y_i\mathbf{w}^T\mathbf{x}_i\right)$$

- When people say logistic regression, they typically mean "**linear** logistic regression"
  - But there is kernel logistic regression

$$\min_{\mathbf{w}} f(\mathbf{w}),\ f(\mathbf{w}) \equiv \frac{\mathbf{w}^T\mathbf{w}}{2} + C\sum_{i=1}^{m} \log\left(1 + e^{-y_i\mathbf{w}^T\phi(\mathbf{x}_i)}\right)$$

  - However, kernel logistic regression is rarely used in practice
    - Most λ_i are not zero → if we want to apply the kernel trick (instead of storing **w** explicitly), almost all training samples need to be memorized

Regularized linear regression

- Training data:

$$\{\mathbf{x}_i, y_i\}_{i=1,\ldots,m},\ \mathbf{x_i} \in R^n, y_i \in R^1$$

- Objective

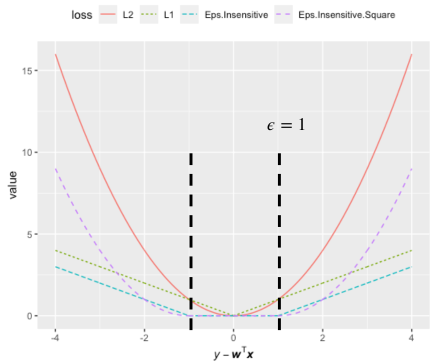
$$\min_{\mathbf{w}}\left(\frac{\mathbf{w}^T\mathbf{w}}{2} + C\sum_{i=1}^{m}\xi\left(\mathbf{w}; \mathbf{x}_i, y_i\right)\right)$$

  - $\xi\left(\mathbf{w}; \mathbf{x}_i, y_i\right)$: loss function
    - Trying to fit the training data
  - $\mathbf{w}^T\mathbf{w}/2$: regularization term
    - We skip the L1 regularization term here
    - Prevent over-fit the training data
  - C: regularization parameter

Loss functions for regression

- Some commonly used loss functions
  - L1 loss
    - $\xi_{L1}\left(\mathbf{w}; \mathbf{x}_i, y_i\right) = \left|y_i - \mathbf{w}^T\mathbf{x}_i\right|$
  - L2 loss
    - $\xi_{L2}\left(\mathbf{w}; \mathbf{x}_i, y_i\right) = \left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$
  - ϵ-insensitive loss
    - $\xi_{\epsilon}\left(\mathbf{w}; \mathbf{x}_i, y_i\right) = \max\left(\left|\mathbf{w}^T\mathbf{x}_i - y_i\right| - \epsilon,\ 0\right)$
  - ϵ-insensitive square loss
    - $\xi_{\epsilon 2}\left(\mathbf{w}; \mathbf{x}_i, y_i\right) = \max\left(\left|\mathbf{w}^T\mathbf{x}_i - y_i\right| - \epsilon,\ 0\right)^2$
- SVM (support vector regression): ξ_{ϵ}, $\xi_{\epsilon 2}$
- Linear Regression: ξ_{L2}

Visualizing the loss functions



Regularized regression with kernel

- Training data:

  $\{\mathbf{x}_i, y_i\}_{i=1,\ldots,m}, \ \mathbf{x_i} \in R^n, y_i \in R^1$

- Objective

$$\min_{\mathbf{w}} f(\mathbf{w}),\ f(\mathbf{w}) \equiv \frac{\mathbf{w}^T\mathbf{w}}{2} + C\sum_{i=1}^{m} \xi\Big(\mathbf{w}; \phi(\mathbf{x}_i), y_i\Big)$$

  – $\xi\Big(\mathbf{w}; \phi(\mathbf{x}_i), y_i\Big)$: loss function
    - Trying to fit the training data
  – $\mathbf{w}^T\mathbf{w}/2$: regularization term
    - We skip the L1 regularization term here
    - Prevent over-fit the training data
  – C: regularization parameter

Summary

- The same classification method can be derived from different ways
  – SVM
    - Maximize margin
    - Minimizing training loss with regularization constraints
  – LR
    - Maximize log-likelihood
    - Minimizing training loss with regularization constraints
- Linear regression and support vector regression are also under the same umbrella
- Understanding the concept of training loss and regularization enables you to self-study many machine learning techniques

Quiz

- What are “support vectors” of SVM?
- When increasing training samples, will the size of “logistic regression model” increase?
- When increasing training samples, will the size of “linear SVM model” increase?
- When increasing training samples, will the size of “kernel SVM model” increase?
- What is “kernel trick”
- Compare the similarity and differences of logistic regression and support vector machines, in terms of loss function and the dimensionality of $\boldsymbol{w}$
- What is “support vector regressor”?