Support Vector Machines 簡介

林宗勳 (daniel@cmlab.csie.ntu.edu.tw)

前言

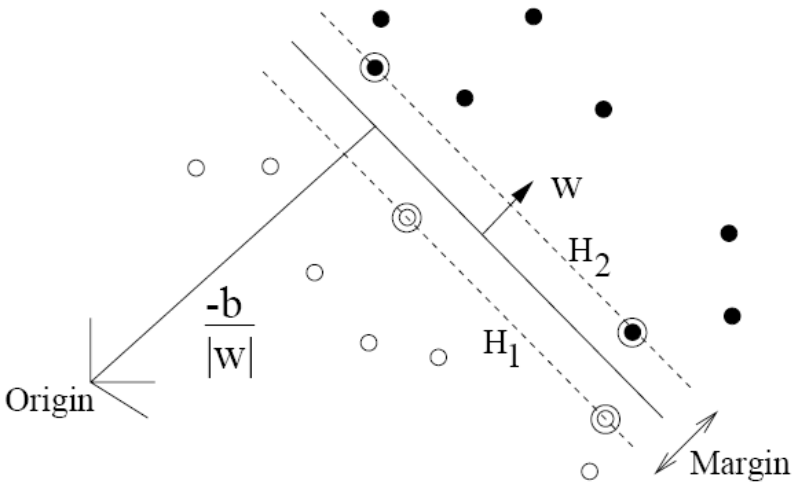
Support Vector Machines(支持向量機，以下都以SVM稱呼)是一種分類(Classification)演算法，由 Vapnik等根據統計學習理論提出的一種新的機器學習方法。

SVM 在解決小樣本、非線性及高維模式識別問題中表現出許多特有的優勢。已經應用於手寫體識別、三維目標識別、人臉識別、文本圖像分類等實際問題中，性能優於已有的學習方法，表現出良好的學習能力。從有限訓練樣本得到的決策規則對獨立的測試集仍能夠得到較小的誤差。

SVM 的概念

簡單來說，SVM 想要解決以下的問題：找出一個超平面(hyperplane)，使之將兩個不同的集合分開。爲什麼使用超平面這個名詞，因爲實際資料可能是屬於高維度的資料，而超平面意指在高維中的平面。

以二維的例子來說，如下圖，我們希望能找出一條線能夠將黑點和白點分開，而且我們還希望這條線距離這兩個集合的邊界(margin)越大越好，這樣我們才能夠很明確的分辨這個點是屬於那個集合，否則在計算上容易因精度的問題而產生誤差。



接下來我們將用更「數學」一點的方式來描述上面的問題

假設我們有一堆點集合 $\{x_i, y_i\}, i = 1,...,n$ and $x_i \in R^d, y_i \in \{+1,-1\}$

我們希望能找到一條直線 $f(x) = w^T x - b$ 使所有 $y_i = -1$ 的點落在 $f(x) < 0$ 的這一邊，且使所有 $y_i = +1$ 的點落在 $f(x) > 0$ 的這一邊，這樣我們就可以根據 $f(x)$ 的正負號來區分這個點是屬於這兩個集合之中的那一個。我們把這樣的超平面稱爲 separating hyperplane，而距離兩邊邊界最大的就稱爲 optimal separating hyperplane (OSH)。接下來的問題就是我們要如何找出這個超平面。

Support Hyperplane

在解決這個問題之前，我先介紹一個新名詞： support hyperplane
support hyperplane 是指與 optimal separating hyperplane 平行，並且最靠近兩邊的超平面。以上面二維的例子來說，那兩條虛線就是 support hyperplane。我們將 support hyperplane 寫成如下的式子：

$$w^T x = b + \delta$$
$$w^T x = b - \delta$$

這是一個有過多待解參數的問題(over-parameterized problem)。假設我們對 x, b, δ 都乘上任意一個常數，等式仍然成立，也就是說有無限多組的 x, b, δ 滿足條件。爲了簡化問題並且消除這個不確定性，我們在等式兩邊乘上一個常數使得 δ=1，這樣就可以去掉一個待解的參數。在接下來的討論，我們所指的 x 及 b 都做過這樣的尺度調整(scale)。

找出 optimal separating hyperplane 的問題就等於找出相距最遠(margin 最大)的 support hyperplane(在兩個 support hyperplane 正中間切一刀就是 OSH)。

我們定義 separating hyperplane 與兩個 support hyperplane 的距離爲 d

$$d = \left(\|b+1| - |b|\right)/\|w\| = 1/\|w\| \quad if\ b \notin (-1,0)$$
$$d = \left(\|b+1| + |b|\right)/\|w\| = 1/\|w\| \quad if\ b \in (-1,0)$$

而兩個 separating hyperplan 之間的距離 margin 自然就是二倍的 d

margin = 2d = 2/||w||　　→ ||w|| 越小則 margin 越大

我們知道 support hyperplan 與 optimal separating hyperplane 的距離在 ±1 以內(我們已經先做過尺度調整)，所以我們將限制條件寫成下面兩個式子：

$$w^T x_i - b \le -1 \quad \forall y_i = -1$$
$$w^T x_i - b \ge +1 \quad \forall y_i = +1$$

上面兩個限制式可以進一步寫成一個限制式，如下：

$$y_i\left(w^T x_i - b\right) - 1 \ge 0$$

總合上面所有的討論，我們有了以下的目標函式：

minimize $\frac{1}{2}\|w\|^2$

subject to $y_i\left(w^T x_i - b\right) - 1 \ge 0 \quad \forall i$

這就是 SVM 所要解決的主要問題(the primal problem of the SVM)

因爲限定條件的關係，上面的最佳化問題有點棘手，還好我們可以利用 Lagrange Multiplier Method將上面的式子轉成一個二次方程式，找出可以使L爲最小值的w, b, α_i。(α_i就是Lagrange Multiplier)

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i \left[y_i\left(w^T x_i - b\right) - 1 \right]$$

符合條件的極值點會出現在：

當 $y_i\left(w^T x_i - b\right) - 1 = 0$ 時，$\alpha_i \ge 0$

當 $y_i\left(w^T x_i - b\right) - 1 > 0$ 時，α_i 必爲 0

Lagrange Multiplier Method 的概念就是把限制條件也變成目標函式的一部分。如果對 Lagrange Multiplier Method 不是很熟悉，可以參考下面的網站：

Lagrange Multiplier Method(中文簡介)
http://episte.math.ntu.edu.tw/entries/en_lagrange_mul/index.html
http://episte.math.ntu.edu.tw/entries/en_lagrange_mul/notes.html

Tutorial on Lagrange Multipliers (英文)
http://www.twocw.net/mit/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-867Machine-LearningFall2002/CFAA1B28-850A-422E-AF84-4A1C1DB51D2C/0/lagrange.pdf

接下來我們要求解 L 的最小值，所以分別對 w 及對 b 偏微

$$w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0 \quad \Rightarrow w^* = \sum_{i=1}^{N} \alpha_i y_i x_i$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

代回去上面的 Lagrangian 目標函式得到

$$\text{maximize } L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

$$\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0 \quad \forall \alpha_i \geq 0$$

我們把上面的條件總合一下

$\partial w L_p = 0 \rightarrow$	$w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0$
$\partial b L_p = 0 \rightarrow$	$\sum_{i=1}^{N} \alpha_i y_i = 0$
	$y_i \left(w^T x_i - b \right) - 1 \geq 0$
Lagrange multiplier condition	$\alpha_i \geq 0$
complementary slackness	$\alpha_i \left[y_i \left(w^T x_i - b \right) - 1 \right] = 0$

上面這些條件又被稱爲KKT條件(Karush-Kuhn-Tucker conditions)

Support Vector

當我們把訓練資料(training data)丟進來時，有些點會滿足上面的KKT條件，而這些點就是剛好落在support hyperplan上面的點，這些點又被稱爲 support vector，並且它們的 α_i 會大於 0

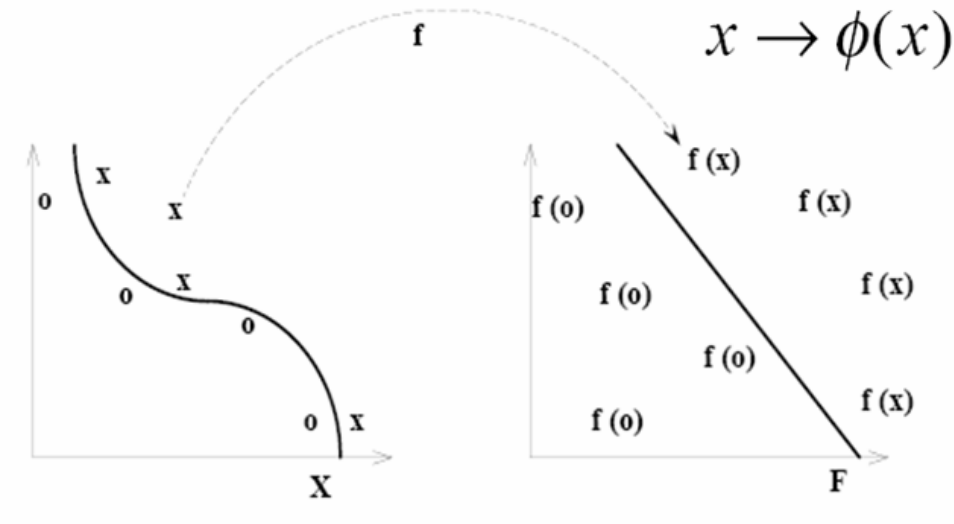
在我們求得 support vector 之後，我們就可以用它們來判斷新加入的點是屬於那一邊的集合。

$$f(x) = w^{*T}x - b^* = \sum_i \alpha_i y_i x_i^T x - b^*$$

$$b^* = \left(\sum_j \alpha_j y_j x_j^T x_i - y_i \right) \quad i \text{ is a support vector}$$

Kernel

上面我們討論的都是資料爲線性的情形，那如果是非線性資料的情形我們又該如何？解決的方法是把資料投射到更高維度的空間或是特徵空間(feature space)去



ϕ 是映射函數，將資料映射到特徵空間，我們透過 ϕ 把 x 映射到特徵空間

$\therefore x_i^T x_j \rightarrow \phi(x_i)^T \phi(x_j)$

在之前的推導中，我們刻意把$x_i^T x_j$ 湊在一起以兩個x項的內積的方式呈現其實有特別的功用。映射函數 ϕ 可能是一個很複雜，不容易求值的函數，但其內積型式卻可能變得很簡單，以 radial based function (RBF) 爲例，RBF是一個頗爲複雜的函數，但將其做內積後所得到的函數卻意外的簡單：

$k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$

$$k(x_i, x_j) = \exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2} \right)$$

像這樣將映射函數做內積所得到的函數在 SVM 中稱之爲 kernel。

資料不可分隔的情形（The Non-Separable case)

真實世界的情形通常比我們的假設更爲複雜，上面的討論都是基於我們能夠找到一個 optimal separating hyperplane 將資料做切分。但在資料本身也可能會有誤差的情形之下，我們很有可能找不到一個 OSH 可以很明確的將兩邊的資料分開。

這樣的資料通當是在邊界的地方有重疊，用上面所討論的方法無法處理這樣的情形，要處理邊界有重疊的情形，我們必須再導入一個誤差項 ξ。

$$w^T x_i - b \le -1 + \xi_i \quad \forall y_i = -1$$
$$w^T x_i - b \ge +1 - \xi_i \quad \forall y_i = +1$$
$$\xi_i \ge 0 \quad \forall i$$

當然我們希望誤差項 ξ的範圍愈小愈好，不然上面的限制條件就沒有作用了。
我們要給有誤差的資料一點懲罰，讓它多一點成本(cost)。
C 是我們決定要給多少懲罰的權重。

Cost = $C\left(\sum_i \xi_i\right)^k$

我們得到一個新的目標函數：

minimize $\frac{1}{2}\|w\|^2 + C\sum_i \xi_i$

subject to $y_i\left(w^T x_i - b\right) - 1 + \xi_i \ge 0 \quad \forall i$

$\xi_i \ge 0 \quad \forall i$

使用 Lagrange Multiplier Method 將限制條件引入

$$L(w,b,\xi,\alpha,\mu) = \frac{1}{2}\|w\|^2 + C\sum_i \xi_i - \sum_{i=1}^{N} \alpha_i \left[y_i\left(w^T x_i - b\right) - 1 + \xi_i\right] - \sum_{i=1}^{N} \mu_i \xi_i$$

新的 KKT 條件

$\partial w L_p = 0 \rightarrow$ $\quad w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0$

$\partial b L_p = 0 \rightarrow$ $\quad \sum_{i=1}^{N} \alpha_i y_i = 0$

$\partial \xi L_p = 0 \rightarrow$ $\quad C - \alpha_i - \mu_i = 0$

$y_i \left(w^T x_i - b \right) - 1 + \xi_i \geq 0$

$\xi_i \geq 0$

Lagrange multiplier condition $\quad \alpha_i \geq 0$

Lagrange multiplier condition $\quad \mu_i \geq 0$

complementary slackness $\quad \alpha_i \left[y_i \left(w^T x_i - b \right) - 1 + \xi_i \right] = 0$

complementary slackness $\quad \mu_i \xi_i = 0$

有關SVM進一步的資料可參考麻省理工學院「開放式課程網頁」機器學習
http://www.twocw.net/mit/Electrical-Engineering-and-Computer-Science/6-867Machine-LearningFall2002/LectureNotes/index.htm

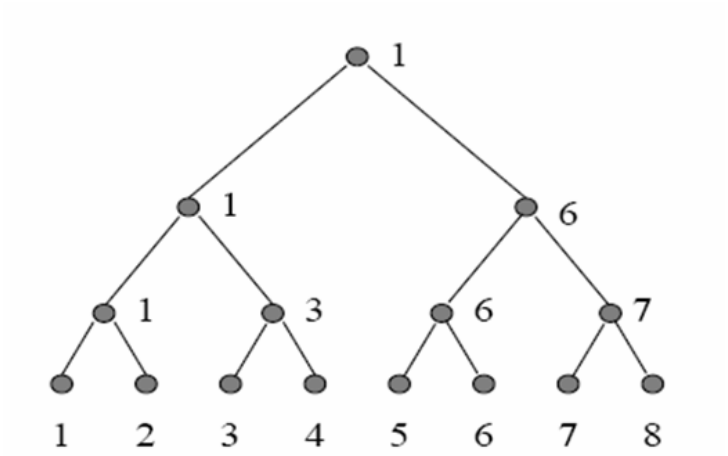
將 SVM 由二元分類擴展到多元分類

基本上 SVM 是一個二元的分類器(binary classifier)，而實際的情形常常是需要多元的分類，我們需要使用一些策略讓我們在多元的分類上也可以使用 SVM。
SVM 在多元分類上的策略主要有兩種：

a. 一對多(one-against-rest)
   於 k 類有 k 個 SVM，第 m 個 SVM 可以將第 m 類和其他類分開，也就是會分辨是不是屬於特定類的 SVM。

b. 一對一(one-against-one)
   對於任兩個類別都造一個 SVM，共需要 k(k-1)/2 個 SVM。
   SVM 只能分辨當初訓練時所使用的兩類。假設當初是用 a 類和 b 類來訓練這個 SVM，那它自然對屬於 a 類或 b 類的資料會分辨的相當好，但如果我們的測試資料還夾雜著其他類的資料，如 c 類，那我們就很難確定這到底是那一

類的資料，我們只能由這個 SVM 知道這是不屬於那類的資料。
我們可藉由一個淘汰賽的方式來判斷這是屬於那一類的資料。比賽是由樹的最底層開始，到最後看是那一類勝出。



以上圖爲例，假設我們的測試資料總共有八類，樹的最底層必須完全包含這八類的 SVM，不然最後的結果可能會有遺漏。在上圖之中，第一次的比賽的結果是 1,3,6,7 這四個類別晉級，接下來我們必須安排下一場比賽，由 1,3 及 6,7 比賽(使用能分辨 1,3 的 SVM 和能分辨 6,7 的 SVM)，這一輪的比賽由 1,6 勝出，我們只要安排 1,6 再比一場就知道這是那一類的資料了。

如何利用 libsvm 來做人臉辨識 (face recognition)

接下來將簡單介紹如何利用 SVM 來進行人臉識別，我們將使用由林智仁老師所開發的 libsvm

libsvm 軟體可在下面的網址取得
http://www.csie.ntu.edu.tw/~cjlin/libsvm/

對於剛接觸 libsvm 的使用者，下面的網址提供非常容易了解的入門指引，這是由其他 libsvm 的使用者提供的，建議先讀一下，了解 libsvm 的用法，才容易了解接下來我們所討論的內容
http://www.csie.ntu.edu.tw/~cjlin/libsvm/otherdocuments/

這次實驗所使用的人臉資料庫
http://cvc.yale.edu/projects/yalefaces/yalefaces.html
這個資料庫包含 15 個人的人臉，每個人有 11 張影像，每張影像拍攝在不同的光照條件(illumination)、有沒有帶眼鏡、各種不同的臉上表情、背景不同…等等。下面是其中一個人的影像資料：

我們取前 8 張做爲訓練資料，後 3 張爲測試資料。

前置處理

在做訓練之前，我們先做些前置處理

1. 將影像縮小到 20 x 20 pixel。根據實驗，20 x 20 的影像已經提供相當高的辨識率。
2. 利用 PCA 做特徵選取(feature extraction)，也就是使用 eigen face 的方法，下面簡述 eigen face 的特徵選取：

a. 先將每個人臉加總求平均，求得的一張 average face

$$\Psi = \frac{1}{M}\sum_{n=1}^{M}\Gamma_n$$ (M 是所有影像的數目)

b. 再對每張臉求與平均臉的差異，將每張臉減 average face，得到一個 vector

$$\Phi_i = \Gamma_i - \Psi$$

c. 把所有的 vector 排成一個矩陣 A

d. 矩陣 A 乘上自已的轉置變成 covariance matrix

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T = \frac{1}{M}\sum_{n=1}^{M}\begin{bmatrix} \mathrm{var}(p_1) & \cdots & \mathrm{cov}(p_1, p_N) \\ \vdots & \ddots & \vdots \\ \mathrm{cov}(p_N, p_1) & \cdots & \mathrm{var}(p_N) \end{bmatrix}_n = AA^T$$

e. 求解這個 covariance matrix 的 eigen vector

f. 我們只取前 14 個 eigne vector

g. 將原來的人臉影像投影到這 14 個 eigen vector 上，所以每張人臉影像會有 14 個維度的資料

用 **libsvm** 做訓練

將所有影像的資料都依 libsvm 規定的輸入格式排好

```
<label> <index1>:<value> <index2>:<value> ...
<label> <index1>:<value> <index2>:<value> ...
...
```

TrainingSet.txt

1	1:599.252014	2:287.783844	3:618.011169	...	14:112.386940
1	1:142.808151	2:540.754883	3:236.278305	...	14:-50.537468
1	1:527.042419	2:183.781281	3:586.409546	...	14:109.996880
1	1:413.529144	2:150.624924	3:604.855286	...	14:62.5474210
1	1:192.096954	2:824.704895	3:-345.25378	...	14:-30.935461
1	1:396.304565	2:291.588989	3:848.980896	...	14:43.3401150
1	1:492.790314	2:294.677399	3:823.742981	...	14:96.3041920
1	1:-1109.4270	2:-385.86004	3:205.677658	...	14:138.645569
2	1:557.744751	2:-123.60966	3:-210.88768	...	14:127.031754
2	1:547.599426	2:-254.80453	3:496.689636	...	14:-159.22918
2	1:404.558716	2:-173.10485	3:-214.58169	...	14:148.403870
2	1:237.526169	2:-147.54145	3:-181.04240	...	14:41.8436470
...					
15	1:-916.79571	2:-569.626953	3:299.984253	...	14:163.308319

爲了數值計算的穩定度，我們需要將資料正規化(normalization) 將每一維度的資料正規化到 [-1,+1] 的範圍。libsvm 提供 svmscale 這個工具來做這件事。

```
svmscale
usage: svmscale [-l lower] [-u upper] [-y y_lower y_upper]
        [-s save_filename] [-r restore_filename] filename
(default: lower = -1, upper = 1, no y scaling)
```

svmscale -l -1 -u 1 -s range TrainingSet.txt > TrainingSet.scale

接下來將剛剛做過尺度調整(scale)的資料丟到 svmtrain 之中進行訓練
svmtrain 會產生一個 TrainingSet.scale.model 的檔案

svmtrain TrainingSet.scale

TrainingSet.scale.model

```
svm_type c_svc
kernel_type rbf
gamma 0.0714286
nr_class 15
total_sv 119
rho 0.0148653 0.0397785 -0.0480648 -0.128948 0.11245 0.239645 0.0185018 …
label 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
nr_sv 8 8 8 8 7 8 8 8 8 8 8 8 8 8 8
SV
…
```

我們已經做完初步的訓練，我們利用 svmpredict 來驗證訓練的結果是否達到我們的要求。

```
svmpredict
Usage: svm-predict [options] test_file model_file output_file
options:
-b probability_estimates: whether to predict probability estimates, 0 or 1 (defa ult 0);
for one-class SVM only 0 is supported

svmpredict    TrainingSet.scale    TrainingSet.scale.model    out

Accuracy = 94.1667% (113/120) (classification)
Mean squared error = 3.95 (regression)
Squared correlation coefficient = 0.804937 (regression)
```

結果有 94.1667% 的正確率，看來這次訓練所得的成果還不錯

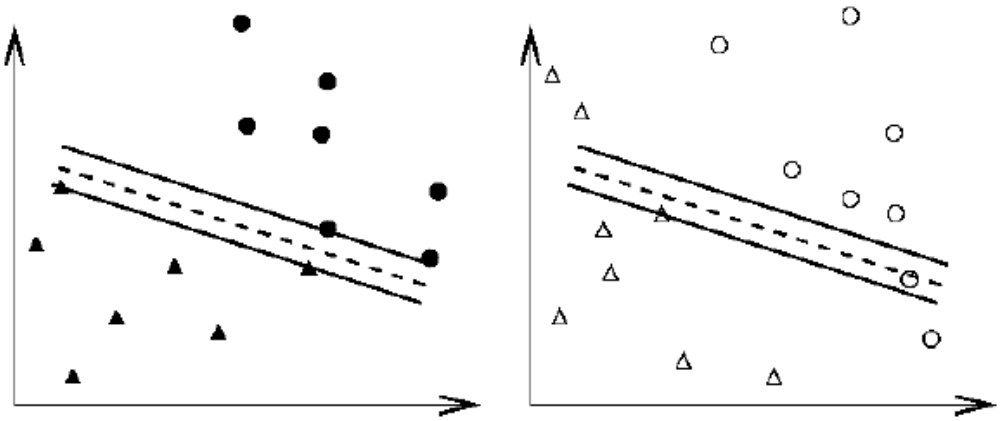
交叉驗證(cross-validation) 及格子點式參數搜尋(grid-search)

在做 SVM 訓練時，有兩個重要的參數要設：cost 及 gamma

1. cost 就是前面討論 SVM in non-separable case 時所提到的 C
2. gamma 是 kernel 內部的一個參數

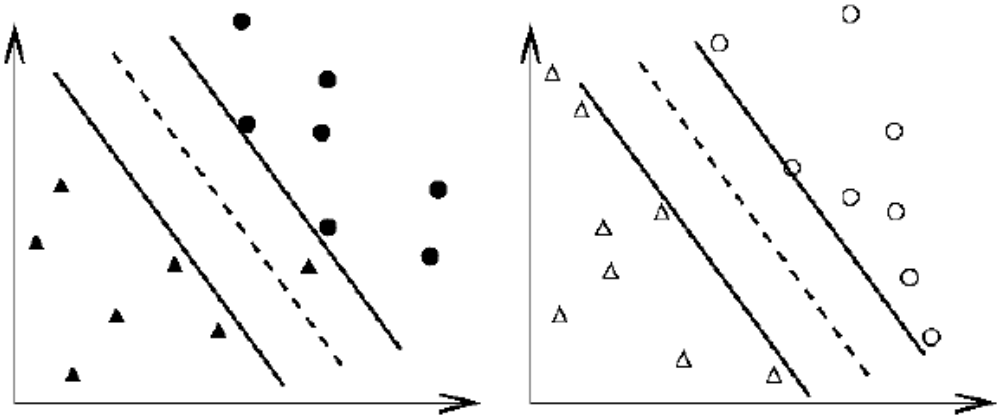
這兩個參數設的好不好對訓練的結果影響重大。既然這樣，那要怎麼決定這兩個參數呢？很可惜，並沒一個公式可以來決定 cost 及 gamma 該設多少，每個不同的案例有不同適用的 cost 及 gamma，我們只能一個個去試。

一般的做法是把訓練資料拆成兩個部分，一個部分用來訓練，另一個部分用來驗證準確度，若準確度不夠的話，換參數再做一次。libsvm 的做法是將資料拆成 n 組相同大小的資料，依序選取一組做爲測試用，剩下其他組的資料則用來做訓練。這個程序稱爲交叉驗證(cross-validation)，交叉驗證可以避免 over-fitting 的發生。下面的圖顯示因參數設定不理想而產生 over-fitting 的情況。



(a) Training data and an overfitting classifier

(b) Applying an overfitting classifier on testing data



(c) Training data and a better classifier

(d) Applying a better classifier on testing data

libsvm 提供一個工具 grid.py 來幫忙做交叉驗證的工作。grid.py 會將參數以指數倍增的方式增加(例如： $C = 2^{-5},2^{-3},...,2^{15}$ $\gamma = 2^{-15},2^{-13},...,2^{3}$)，然後將每一種組合都做測試，這就像將 C 所有要測試的值擺在水平軸上，將 γ 所有要測試的值擺在垂直軸上，兩者相交的格子點(grid)就是 libsvm 測試的目標。

grid.py TrainingSet.scale

…
[local] 13 3 43.3333 (best c=128.0, g=0.0078125, rate=82.5)
[local] 13 -9 82.5 (best c=128.0, g=0.0078125, rate=82.5)

```
[local] 13 -3 80.8333    (best c=128.0, g=0.0078125, rate=82.5)
128.0 0.0078125 82.5
```

grid.py 建議我們使用 cost=128, gamma=0.0078125 來做訓練

訓練完成後，便可將之前準備好的測試資料拿進來測試，這邊要特別注意的是：我們之前的訓練資料是有做過尺度調整(scale)，所以測試資料也要做過尺度調整才行。