View this tutorial in: **English Only** **TraditionalChinese Only** **Both (Default)** (req. JavaScript if you want to switch languages)

Core StyleSheets: **Chocolate** **Midnight** **Modernist** **Oldstyle** **Steely** **Swiss** **Traditional** **Ultramarine**

Core StyleSheets: **Chocolate** **Midnight** **Modernist** **Oldstyle** **Steely** **Swiss** **Traditional** **Ultramarine**

* This document is written in multilingual format. We strongly suggest that you choose your language first to get a better display.

piaip's Using (lib)SVM Tutorial
piaip 的 (lib)SVM 簡易入門

piaip at csie dot ntu dot edu dot tw,
Hung-Te Lin
Fri Apr 18 15:04:53 CST 2003
$Id: svm_tutorial.html,v 1.13 2007/10/02 05:51:55 piaip Exp piaip $

Why this tutorial is here

我一直覺得 SVM 是個很有趣的東西，不過也一直沒辦法 (mostly 衝堂) 去聽林智仁老師 的 Data mining 跟 SVM 的課； 後來看了一些網路上的文件跟聽 kcwu 講了一下 **libsvm** 的用法後，就想整理一下，算是對於並不需要知道完整 SVM 理論的人提供使用 **libsvm** 的入門。 原始 libsvm 的 README 跟 FAQ 也是很好的文件， 不過你可能要先對 svm 跟流程有點了解才看得懂 (我在看時有這樣的感覺)； 這篇入門就是爲了從零開始的人而寫的。

I've been considering SVM as an interesting and useful tool but couldn't attend the "Data mining and SVM" course by prof. cjline about it (mostly due to scheduling conflicts). After reading some materials on the internet and discussing **libsvm** with some of my classmates and friends , I wanted to provide some notes here as a tutorial for those who do not need to know the complete theory behind SVM theory to use **libsvm** . The original README and FAQ files that comes with libsvm are good documents too. But you may need to have some basic knowledge of SVM and its workflow (that's how I felt when I was reading them).

This tutorial is specificly for those starting from zero.

後來還有一些人提供意見，所以在此要感謝：
I must thank these guys who provided feedback and helped me make this tutorial:

kcwu, biboshen, puffer, somi

不過請記得底下可能有些說法不一定對，但是對於只是想用 SVM 的人來說我覺得這樣說明會比較易懂。
Remember that some aspect below may not be correct. But for those who just wish to "USE" SVM, I think the explanation below is easier to understand.

這篇入門原則上是給會寫基本程式的人看的，也是給我自己一個備忘, 不用太多數學底子，也不用對 SVM 有任何先備知識。
This tutorial is basically for people who already know how to program. It's also a memo to myself. Neither too much mathmatics nor prior SVM knowledge is required.

還看不懂的話有三個情形, 一是我講的不夠清楚, 二是你的常識不足, 三是你是小白 ^^;
If you still can't understand this tutorial, there are three possibilities: 1. I didn't explain clearly enough, 2. You lack sufficient common knowledge, 3. You don't use your brain properly ^^;

我自己是以完全不懂的角度開始的，這篇入門也有不少一樣不懂 SVM 的人 看過、而且看完多半都有一定程度的理解，所以假設情況一不會發生， 那如果不懂一定是後兩個情況 :P 也所以, 有問題別問我。
Since I begin writing this myself with no understanding of the subject, ans this document has been read by many people who also didn't understand SVM but gained a certain level of understanding after reading it, possibility 1 can be ruled out. Thus if you can't understand it you must belong to the latter two categories, :P thus even if you have any questions after reading this, don't ask me.

SVM: What is it and what can it do for me?

SVM, Support Vector Machine , 簡而言之它是個起源跟類神經網路有點像的東西， 不過現今最常拿來就是做分類 (classification)。 也就是說，如果我有一堆已經分好類的東西 （ 可是分類的依據是未知的！） ，那當收到新的東西時，SVM 可以預測 (predict) 新的資料要分到哪一堆去。
SVM, Support Vector Machine , is something that has similar roots with neural networks. But recently it has been widely used in **Classification**. That means, if I

have some sets of things classified **(But you know nothing about HOW I CLASSIFIED THEM, or say you don't know the rules used for classification)**, when a new data comes, SVM can **PREDICT** which set it should belong to.

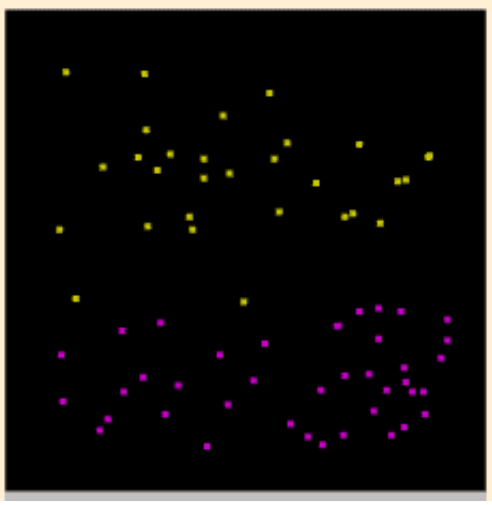
聽起來是很神奇的事（如果你覺得不神奇，請重想一想這句話代表什麼： 分類的依據是未知的！，還是不神奇的話就請你寫個程式 解解看這個問題），也很像要 AI 之類的高等技巧... 不過 SVM 基於 統計學習理論 可以在合理的時間內漂亮的解決這個問題。

It sounds marvelous and would seem to require advanced techniques like AI searching or some time-consuming complex computation. But SVM used some **Statistical Learning Theory** to solve this problem in reasonable time.

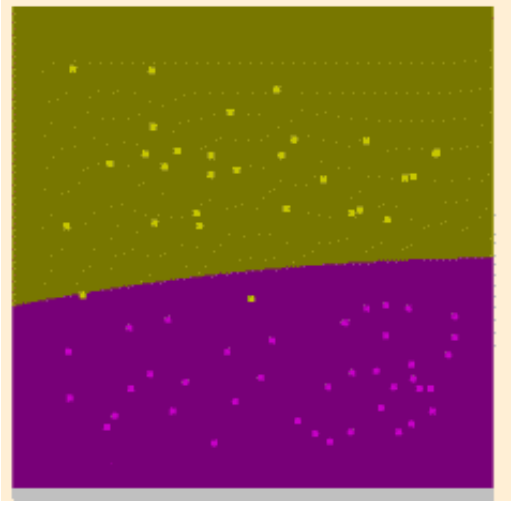
以圖形化的例子來說明(by SVMToy), 像假定我在空間中標了一堆用顏色分類的點, 點的顏色就是他的類別, 位置就是他的資料, 那 SVM 就可以找出區隔這些點的方程式, 依此就可以分出一區區的區域; 拿到新的點(資料) 時, 只要對照該位置在哪一區就可以(predict) 找出他應該是哪一顏色(類別)了:

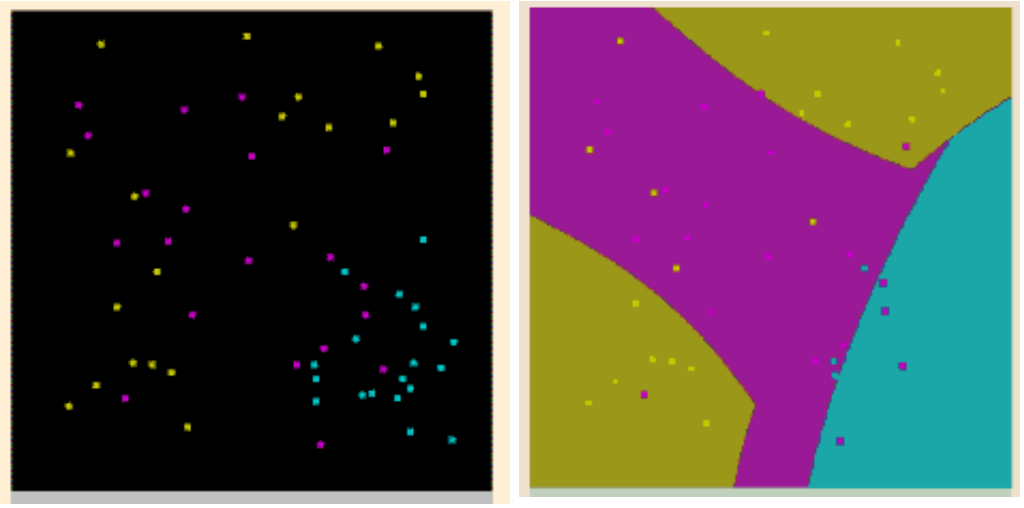
Now we explain with a graphical example(by SVMToy), I marked lots of points with different colors on a plane, the color of each point is its "class" and the location is its data. SVM can then find equations to split these points and with these equations we can get colored regions. When a new point(data) comes, we can find (predict) what color (class) a point should be just by using the point's location (data)

原始資料分佈
Original Data



SVM找出來的區域
SVM Regions





當然 SVM 不是真的只有畫圖分區那麼簡單, 不過看上面的例子應該可以了解 SVM 大概在作什麼.
Of course SVM is not really just about painting and marking regions, but with the example above you should should be able to get some idea about what SVM is doing.

要對 SVM 再多懂一點點，可以參考 cjlin 在 data mining 課的 slides: **pdf** or **ps** 。底下我試著在不用看那個 slide 的情況 解釋及使用 libsvm。
To get yourself more familiar with SVM, you may refer to the slides cjlin used in his Data Mining course : **pdf** or **ps** .
I'm going to try to explain and use libSVM without those slides.

所以, 我們可以把 SVM 當個黑盒子, 資料丟進去讓他處理然後我們再來用就好了.
Thus we can consider SVM as a black box. Just push data into SVM and use the output.

How do I get SVM?

林智仁**(cjlin)**老師的 **libsvm** 當然是最完美的工具.
Chih-Jen Lin's **libsvm** is of course the best tool you can ever find.

Download libsvm

下載處:
Download Location:

libsvm.zip or **libsvm.tar.gz**

.zip 跟 .tar.gz 基本上是一樣的, 只是看你的 OS; 習慣上 Windows 用 .zip 比較方便 (因爲有 WinZIP, 不過我都用 WinRAR), UNIX 則是用 .tar.gz

Contents in the .zip and .tar.gz are the same. People using Windows usually like to use .zip files because they have WinZIP, which I always replace with WinRAR. UNIX users mostly prefer .tar.gz

Build libsvm

解開來後, 假定是 UNIX 系統, 直接打 make 就可以了; 編不出來的話請 詳讀說明和運用常識. 因爲這是 tutorial, 所以我不花時間細談, 而且 會編不出來的情形真是少之又少, 通常一定是你的系統有問題或你太笨了. 其他的子目錄可以不管, 只要 *svm-train, svm-scale, svm-predict* 三個執行檔有編出來就可以了.
After you extracted the archives, just type **make** if you are using UNIX. You may ignore some of the subdirectories. We only need these executable files: *svm-train, svm-scale, and svm-predict*

Windows 的用戶要自己重編當然也是可以, 不過已經有編好的 binary 在裡面了: 請檢查 windows 子目錄, 應該會有 *svmtrain.exe, svmscale.exe, svmpredict.exe, svmtoy.exe* .
Windows users may rebuild from source if you want, but there're already some prebuilt binaries in the archive: just check your "windows" subdirectory and you should find *svmtrain.exe, svmscale.exe, svmpredict.exe, and svmtoy.exe* .

Using SVM

libsvm 有很多種用法, 這篇 tutorial 只打算講簡單的部分.
libsvm has lots of functions. This tutorial will only explain the easier parts (mostly classification with default model).

The programs

解釋一下幾個主要執行檔的作用: (UNIX/Windows 下檔名稍有不同, 請用常識理解我在講哪個)
I'm going to describe how to use the most important executables here. The filenames are a little bit different under Unix and Windows, apply common sense to see which I'm referring to.

svmtrain

Train (訓練) data. 跑 SVM 被戲稱爲 "開火車" 也是由於這個程式名而來. train 會接受特定格式的輸入, 產生一個 "Model" 檔. 這個 model 你可以想像成 SVM

的內部資料, 因爲 predict 要 model 才能 predict, 不能直接吃原始資料. 想想也很合理, 假定 train 本身是很耗時的動作, 而 train 好可以以某種形式存起內部資料, 那下次要 predict 時直接把那些內部資料 load 進來就快多了.
Use your data for training. Running SVM is often referred to as 'driving trains' by its non-native English speaking authors because of this program. svmtrain accepts some specifically format which will be explained below and then generate a 'Model' file. You may think of a 'Model' as a storage format for the internal data of SVM. This should appear very reasonable after some thought, since training with data is a time-consuming process, so we 'train' first and store the result enabling the 'predict' operation to go much faster.

svmpredict

依照已經 train 好的 model, 再加上給定的輸入 (新值), 輸出 predict (預測) 新值所對應的類別 (class).
Output the *predicted* class of the new input data according to a pre-trained model.

svmscale

Rescale data. 因爲原始資料可能範圍過大或過小, svmscale 可以先將資料重新 scale (縮放) 到適當範圍.
Rescale data. The original data maybe too huge or small in range, thus we can rescale them to the proper range so that training and predicting will be faster.

File Format

檔案格式要先交代一下. 你可以參考 libsvm 裡面附的 "heart_scale":
This is the input file format of SVM. You may also refer to the file "heart_scale" which is bundled in official libsvm source archive.

```
[label] [index1]:[value1] [index2]:[value2] ...
[label] [index1]:[value1] [index2]:[value2] ...
.
.
```

一行一筆資料，如
One record per line, as:

+1 1:0.708 2:1 3:1 4:-0.320 5:-0.105 6:-1

label

或說是 class, 就是你要分類的種類，通常是一些整數。

Sometimes referred to as 'class', the class (or set) of your classification. Usually we put integers here.

index

是有順序的索引，通常是放連續的整數。
Ordered indexes. usually continuous integers.

value

就是用來 train 的資料，通常是一堆實數。
The data for training. Usually lots of real (floating point) numbers.

每一行都是如上的結構, 意思就是: 我有一排資料, 分別是 value1, value2, ... valueN, (而且它們的順序已由 indexN 分別指定)，這排資料的分類結果就是 label。
Each line has the structure described above. It means, I have an array(vector) of data(numbers): value1, value2, ... valueN (and the order of the values are specified by the respective index), and the class (or the result) of this array is label.

或許你會不太懂，爲什麼會是 value1,value2,.... 這樣一排呢？ 這牽涉到 SVM 的原理。 你可以這樣想（我沒說這是正確的）， 它的名字就叫 Support "Vector" Machine， 所以輸入的 training data 是 "Vector"(向量), 也就是一排的 x1, x2, x3, ... 這些值就是 valueN，而 x[n] 的 n 就是 由 indexN 指定。 這些東西又稱爲 "attribute"。

真實的情況是， 大部份時候我們給定的資料可能有很多 "特徵(feature)" 或說 "屬性(attribute)"，所以輸入會是 一組的。 舉例來說，以前面畫點分區的例子 來說，我們不是每個點都有 X 跟 Y 的座標嗎？ 所以它就有 兩種 attribute。 假定我有兩個點： (0,3) 跟 (5,8) 分別在 label(class) 1 跟 2 ，那就會寫成

```
1 1:0 2:3
2 1:5 2:8
```

同理，空間中的三維座標就等於有三組 attribute。
Maybe it's confusing to you: why value, value2, ...? The reason is usually the input data to the problem you were trying to solve involves lots of 'features', or say 'attributes', so the input will be a set (or say vector/array). Take the **Marking points and find region** example described above, we assumed each point has coordinates X and Y so it has two attributes (X and Y). To describe two points (0,3) and (5,8) as having labels(classes) 1 and 2, we will write them as:

```
1 1:0 2:3
2 1:5 2:8
```

And 3-dimensional points will have 3 attributes.

這種檔案格式最大的好處就是可以使用 sparse matrix， 或說有些 data 的 attribute 可以不存在。
This kind of fileformat has the advantage that we can specify a sparse matrix, ie. some attribute of a record can be omitted.

To Run libsvm

來解釋一下 libsvm 的程式怎麼用。 你可以先拿 libsvm 附的 heart_scale 來做輸入，底下也以它爲例：
Now I'll show you how to use libsvm. You may use the heart_scale file in the libsvm source archive as input, as I'll do in this example:

看到這裡你應該也了解，使用 SVM 的流程大概就是：
You should have a sense that using libsvm is basically:

1. 準備資料並做成指定格式 (有必要時需 svmscale)
   Prepare data in specified **format** and svmscale it if necessary.
2. 用 svmtrain 來 train 成 model
   Train the data to create a model with svmtrain.
3. 對新的輸入，使用 svmpredict 來 predict 新資料的 class
   Predict new input data with svmpredict and get the result.

svmtrain

svmtrain 的語法大致就是:

The syntax of svmtrain is basically:

svmtrain [options] training_set_file [model_file]

training_set_file 就是之前的格式，而 model_file 如果不給就會 叫 [training_set_file].model。 options 可以先不要給。

The format of training_set_files is described above. If the model_file is not specified, it'll be [training_set_file].model by default. Options can be ignored at first.

下列程式執行結果會產生 heart_scale.model 檔：(螢幕輸出不是很重要，沒有錯誤就好了)
The following command will generate the heart_scale.model file. The screen output may be ignored if there were no errors.

```
./svm-train heart_scale
optimization finished, #iter = 219
nu = 0.431030
obj = -100.877286, rho = 0.424632
nSV = 132, nBSV = 107
Total nSV = 132
```

svmpredict

svmpredict 的語法是：
The syntax to svm-predict is:

svmpredict test_file model_file output_file

test_file 就是我們要 predict 的資料。它的格式跟 svmtrain 的輸入，也就是 training_set_file 是一樣的！ predict 完會順便拿 predict 出來的值跟 test_file 裡面寫的值去做比對，這代表： test_file 寫的 label 是真正的分類結果，拿來跟我們 predict 的結果比對就可以 知道 predict 有沒有猜對了。
test_file is the data the we are going to 'predict'. Its format is almost exactly the same as the training_set_file, which we fed as input to svmtrain. After predicting svm-predict will compare the predicted label with the label written in test_file. That means, test_file has the real (or correct) result of classification, and after comparing with our predicted result we can know whether the prediction is correct or not.

也所以，我們可以拿原 training set 當做 test_file再丟給 svmpredict 去 predict (因爲格式一樣)，看看正確率有多高， 方便後面調參數。
So we can use the original training_set_file as test_file and feed it to svmpredict for prediction (nothing different in file format) and see how high the accuracy is so we can optimize the arguments.

其它參數就很好理解了： **model_file** 就是 svmtrain 出來 的檔案， **output_file** 是存輸出結果的檔案。
Other arguments should be easy to figure out now: **model_file** is the model trained by svmtrain, and **output_file** is where we store the output result.

輸出的格式很簡單，每行一個 label，對應到你的 test_file 裡面的各行。
Format of output is simple. Each line contains a label corresponding to your test_file.

下列程式執行結果會產生 heart_scale.out：

The following commands will generate heart_scale.out:

```
./svm-predict heart_scale heart_scale.model heart_scale.out
Accuracy = 86.6667% (234/270) (classification)
Mean squared error = 0.533333 (regression)
Squared correlation coefficient = 0.532639(regression)
```

As you can see，我們把原輸入丟回去 predict， 第一行的 Accuracy 就是預測的正確率了。 如果輸入沒有 label 的話，那就是真的 predict 了。
As you can see, after we 'predict'ed the original input, we got 'Accuracy=86.6667%" on first line as accuracy of prediction. If we don't put labels in input, the result is real prediction.

看到這裡，基本上你應該已經可以利用 svm 來作事了： 你只要寫程式輸出正確格式的資料，交給 svm 去 train， 後來再 predict 並讀入結果即可。
Now you can use SVM to do whatever you want! Just write a program to output its data in the correct format, feed the data to SVM for training, then predct and read the output.

Advanced Topics

後面可以說是一些稍微進階的部份，我可能不會講的很清楚， 因爲我的重點是想表達一些觀念和解釋一些你看相關文件時 很容易碰到的名詞。
These are a little advanced and I may not explain very clearly. Because I just want to help you get familiar with some of the terminology and ideas that you'll encounter when you read other (lib)SVM documents.

Scaling

svm-scale 目前不太好用，不過它有其必要性。因爲 適當的scale有助於參數的選擇(後述)還有解svm的速度。
svmscale 會對每個 attribute 做scale。 範圍用 -l, -u 指定， 通常是[0,1]或是[-1,1]。 輸出在 stdout。
另外要注意的(常常會忘記)是 testing data 和 training data要一起scale。
而 svm-scale 最難用的地方就是沒辦法指定 testing data/training data(不同檔案)然後一起scale。

svm-scale is not easy to use right now, but it is important. Scaling aids the choosing of arguments (described below) and the speed of solving SVM.

svmscale rescales all atrributes with the specified (by *-l, -u*) range, usually [0,1] or [-1,1].
Please keep in mind that testing data and training data MUST BE SCALED WITH THE SAME RANGE. Don't forget to scale your testing data before you predict.
We can't specify the testing and training data file together and scale them in one command, that's why svm-scale is not so easy to use right now.

Arguments

前面提到，在 train 的時候可以下一些參數。(直接執行 svm-train 不指定輸入檔與參數會列出所有參數及語法說明) 這些參數對應到原始 SVM 公式的一些參數，所以會影響 predict 的正確與否。
舉例來說，改個 c=10:

```
./svm-train -c 10 heart_scale
```

再來 predict ，正確率馬上變成 92.2% (249/270)。

We know that we can use some arguments when we were training data (Running svm-train without any input file or arguments will cause it to print its list syntax help and complete arguments). These arguments corresponds to some arguments in original SVM equations so they will affect the accuracy of prediction.
Let's use c=10 as an example:

```
./svm-train -c 10 heart_scale
```

If you predict again now, the accuracy will be 92.2% (249/270).

Cross Validation

一般而言， SVM 使用的方式(在決定參數時)常是這樣：

1. 先有已分好類的一堆資料
2. 亂數拆成好幾組 training set
3. 用某組參數去 train 並 predict 別組看正確率
4. 正確率不夠的話，換參數再重複 train/predict

Mostly people use SVM while following this workflow:

1. Prepare lots of pre-classified (correct) data
2. Split them into several training sets randomly.
3. Train with some arguments and predict other sets of data to calculate the accuracy.

4. Change the arguments and repeat until we get good accuracy.

等找到一組不錯的參數後，就拿這組參數來建 model 並用來做最後對未知資料的 predict。 這整個過程叫 **cross validation** ， 也就是交叉比對。
When we got some nice arguments, we will then use them to train the model and use the model for final prediction (on unknown test data). This whole process is called **cross validation** .

在我們找參數的過程中，可以利用 svmtrain 的內建 cross validation 功能來幫忙：
-v n: n-fold cross validation
n 就是要拆成幾組，像 n=3 就會拆成三組，然後先拿 1跟2來 train model 並 predict 3 以得到正確率； 再來拿 2跟 3 train 並 predict 1，最後 1,3 train 並 predict 2。其它以此類推。
In the process of experimenting with the arguments, we can use the built-in support for validation of svmtrain:
-v n: n-fold cross validation
n is how many sets to split your input data. Specifing n=3 will split data into 3 sets; train the model with data set 1 and 2 first then predict data set 3 to get the accuracy, then train with data set 2 and 3 and predict data set 1, finally train 1,3 and predict 2, ... ad infinitum.

如果沒有交叉比對的話，很容易找到只在特定輸入時好的 參數。像前面我們 c=10 得到 92.2%，不過拿 -v 5 來看看：

```
./svm-train -v 5 -c 10 heart_scale
...
Cross Validation Accuracy = 80.3704%
```

平均之後才只有 80.37%，比一開始的 86 還差。
If we don't use cross validation, sometimes we may be fooled by some arguments only good for some special input. Like the example we used above, c=10 has 92.2%. If we do so with -v 5:

```
./svm-train -v 5 -c 10 heart_scale
...
Cross Validation Accuracy = 80.3704%
```

After the prediction results is averaged with cross validation we have only 80.37% accuracy, even worse than with the original argument (86%).

What arguments rules?

通常而言，比較重要的參數是 *gamma (-g)* 跟 *cost (-c)* 。而 cross validation (-v) 的參數常用 5。

Generally speaking, you will only modify two important arguments when you are using training with data: *gamma (-g)* and *cost (-c)* . And cross validation (-v) is usually set to 5.

cost 預設值是 1, gamma 預設值是 1/k ，k 等於輸入 資料筆數。 那我們怎麼知道要用多少來當參數呢？

用 試 的

是的，別懷疑，就是 Try 參數找比較好的值。

cost is 1 by default, and gamma has default value = 1/k , k = number of input records. Then how do we know what value to choose as arguments?

T R Y

Yes. Just by trial and error.

Try 參數的過程常用 exponential 指數成長的方式來增加與減少參數的數值， 也就是 2^n (2 的 n 次方)。
When experimenting with arguments, the value usually increases and decreases in exponential order. i.e., 2^n.

因爲有兩組參數，所以等於要 try n*n=n^2 次。 這個過程是不連續的成長，所以可以想成我們在一個 X-Y 平面上指定的範圍內找一群格子點 (**grid**， 如果你不太明白，想成方格紙或我們把平面上所有 整數交點都打個點，就是那樣)，每個格子點的 X 跟 Y 經過換算 (如 2^x, 2^y) 就拿去當 cost 跟 gamma 的值來 cross validation。
Because we have two important arguments, we have to try n*n=n^2 times. The whole process is discontinous and can be thought of as finding the **grid** points on a specified region (range) of the X-Y plane (Think of marking all integer interception points on a paper). Convert each grid point's X and Y coordinate to exponential values (like 2^x, 2^y) then we can use them as value of cost and gamme for cross validation.

所以現在你應該懂得 libsvm 的 python 子目錄下面 有個 grid.py 是做啥的了： 它把上面的過程自動化， 在你給定的範圍內呼叫 svm-train 去 try 所有的參數值。 python 是一種語言，在這裡我不做介紹，因爲我會了 :P (just a joke，真正原因是 -- 這是 libsvm 的 tutorial)。 grid.py 還會把結果 plot 出來，方便你尋找參數。 libsvm 有很多跟 python 結合的部份，由此可見 python 是強大方便的工具。很多神奇的功能，像自動登入多台 機器去平行跑 grid等等都是 python 幫忙的。不過 SVM 本身可以完全不需要 python，只是會比較方便。
So look for 'grid.py' in the 'python' subdirectory inside the libsvm archive. You should know what it does now: automatically execute the procedure above, try all

argument values by calling svm-train within the region specified by you. Python is a programming language which I'm not going to explain here. grid.py will also plot the result graphically to help you look for good arguments. There're also many parts of libsvm powered by python, like logging into several hosts and running grids at the same time parallel. Keep in mind that libsvm can be used without python entirely. Python just only helped us to do thinks quickly.

跑 grid (基本上用 grid.py 跑當然是最方便，不過 如果你不懂 python 而且覺得很難搞，那你要自己產生 參數來跑也是可以的) 通常好的範圍是
[c,g]=[2^-10,2^10][2^-10,2^10]*
另外其實 **grid** 用 **[-8,8]** 也很夠了。
Running for grids (it's more convenient to just use grid.py but it's also ok if you don't) you may choose the range as
[c,g]=[2^-10,2^10][2^-10,2^10]*
Usually [-8,8] is enough for grids.

Regression

另一個值得一提的是 regression。

簡單來說，前面都是拿 SVM 來做分類 (classification), 所以 label 的值都是 discrete data、或說已知的固定值。 而 regression 則是求 continuous 的值、或說未知的值。 你也可以說，一般是 binary classification, 而 regression是可以預測一個實數。

比如說我知道股市指數受到某些因素影響, 然後我想預測股市.. 股市的指數就是我們的 label, 那些因素量化以後變成 attributes。 以後蒐集那些 attributes 給 SVM 它就會 預測出指數(可能是沒出現過的數字)，這就要用 regression。 那樂透開獎的號碼呢？ 因爲都是固定已知的數字， 很明顯我們應該用一般 SVM 的 classification 來 predict。 (註：這是真實的例子 -- llwang 就寫過這樣的東西)

所以說 label 也要 scale, 用
```
svm-scale -y lower upper
```

但是比較糟糕的情況是 grid.py 不支援 regression ， 而且cross validation 對 regression 也常常不是很有效。

總而言之，regression 是非常有趣的東西，不過也是比較 進階的用法。 在這裡我們不細談了，有興趣的人請再 參考 SVM 與 libsvm 的其它文件。

The other important issue is "Regression".

To explain briefly, we only used SVM to do classification in this tutorial. The type of label we used are always discrete data (ie. a known fixed value). "Regression" in this context means to predict labels with continuous values (or unknown values). You can think of classification as predictions with only binary outcomes, and regression as predictions that output real (floating point) numbers.

Thus to predict lottery numbers (since they are always fixed numbers) you should use classification, and to predict the stock market you need regression.

The labels must also be scaled when you use regression, by

```
svm-scale -y lower upper
```

However grid.py does not support regression, and cross validation sometimes does not work well with regression.

Regression is interesting but also advanced. Please refer to other documents for details.

Epilogue

到此我已經簡單的說明了 libsvm 的使用方式， 更完整的用法請參考 libsvm 的說明跟 **cjlin** 的網站、 SVM 的相關文件，或是去上 cjlin 的課。
Here we have already briefly explained the libsvm software. For complete usage guides please refer to documents inside the libsvm archive, **cjlin's website**, SVM-related documents, or go take cjlin's course if you are a student at National Taiwan University :)

對於 SVM 的新手來說， **libsvmtools** 有很多好東西。像 SVM for dummies 就是很方便觀察 libsvm 流程的東西。
Take a glance at **libsvmtools** especially "SVM for dummies" there. Those are good tools for SVM newbies that helps in observing libsvm workflow.

Copyright

All HTML/text typed within VIM on Solaris.
Style sheet from W3C Core StyleSheets.

Original URL: **http://www.csie.ntu.edu.tw/~r91034/svm/svm_tutorial.html**