國立政治大學應用數學系

碩士學位論文

大資料多元尺度在網路使用者偏好分析之應用

The application of large data multidimensional scaling method in network user preference

碩士班學生：潘靜儒　撰

指導教授：　曾正男　博士

中華民國　101 年 7 月 2 日

大資料多元尺度在網路使用者偏好分析之應用

學生： 潘靜儒　　指導教授： 曾正男博士

國立政治大學應用數學研究所

摘　　要

由於現在網路發展的非常快速，資料的產生速度以及使用方式已經超過人們的分析能力與解讀能力，因此近來大資料計算是一個很重要的研究課題，許多科學家與研究單位都積極地投入大量的研究資源，目的就是要研究這些龐大的資料要如何去分析，或是解讀。特別是大型網站上的資料，使用者的數量和可點選的項目是隨著時間不斷增加，如何分析這類的資料是一個重要的課題。

我們將介紹如何利用Python程式的特性對大型網站進行使用者偏好分析，透過亂數投影和分解-合成多元尺度法的合作，做到使用者偏好網路的建制，協助大型網站對使用者進行即時性的閱讀項目推薦。我們提出Data is cache for dimension reduction 的概念，說明大資料計算必須配合資料庫才能達到真正的快速計算結果。

ABSTRACT

The development of the Internet is very fast in recent year. Because the speed of data increases very fast and the type of data usage is over the analysis of recent technique, big data becomes a big deal in this century. There are many scientists involve the research of this field. They want to know how to analysis in time and make the analysis efficiently, especially when the data is growing.

We will introduce how to use the properties of Python language to analysis the user preference in web service. By using the hybrid of random projection and the split-and-combine multidimensional scaling method, we can project user browse history to a low dimensional feature space. In this feature space users are located nearly have similar preference, such that we can recommend the further reading to the similar users. We propose the concept "Database is cache" such that the database should share some simple calculation to improve the computational speed.

誌　　謝

首先我要感謝我的指導老師曾正男老師，讓我可以順利完成這篇論文。在碩一的時候，修了老師的課，也接觸到Python這個程式語言，並且能重拾寫程式的信心。另外，在我面對考試的挫敗時用聖經的話鼓勵我，讓我可以繼續完成學業。在研究上，也從老師身上學到很多知識，並且老師在研究上的態度也是我的榜樣，總是很專注並且很投入，不過最讓我佩服的還是老師的idea總是多到不行，對於欠缺創意的我，幫助很大。我還要感謝教過我課的老師們，讓我在各領域的知識上都有些長進，特別是實變函數論的授課老師陳天進老師，讓我知道我在許多觀念上不是那樣得嚴謹，並且有許多的不足。

再來我要感謝神，一路帶領我，帶我進入這所學校，遇到很多不錯的老師，還有許多厲害的學長姐和同學，以及學弟妹。在我面臨許多挫折時，讓我不斷地轉向祂，使我不信靠自己，能夠將這些都交在祂手中，最後這半年，更是經歷祂信心的帶領。我要感謝我的家人，一路上的支持與陪伴，特別是在決定要邁入第三年的時候，對我的支持與鼓勵。也常常在我需要得時候，陪同我禱告。

我要感謝研究室的大家們，無論是教我許多寫程式技巧的葉長青學長，或是陪我一起讀實變的學長姐同學學弟妹們，還是一起念資格考的同學學弟們，以及常常陪我嘻嘻哈哈的大家。謝謝大家總是包容我，特別是最後壓力很大的一年，總是用大吼來發洩自己的情緒，我很抱歉，並且謝謝大家的耳朵傾聽我的聲音!!

目　　錄

圖　目　錄

第 一 章　論文簡介

由於網路發達之迅速，資料產生的速度以及使用方式已經超越人們的分析能力與解讀能力。因此近年來大資料計算成為最熱門的話題，許多科學家與研究單位紛紛投入研究資源在如何分析、解讀並利用這些持續成長，並且不容易掌握趨勢方向的資料。美國總統歐巴馬也在2012 年宣佈"Big data is a big deal"，美國將積極投入兩億美元在大資料的計算研究，讓美國能取得大資料計算與應用的領先地位。[1]

現今最容易被人聯想到的大資料計算領域有網路服務中的雲端服務，即時推薦系統，客戶偏好分析，客戶未來行為預測等；在生物資訊方面有基因資料的處理，特點基因(oncogene)的選取，以及基因調控研究；在財務金融方面有客戶關係處理，客戶下單訊息推薦，自營商選股組合等。這些都是熱門的大資料處理服務，資料量不只龐大，並且資料在變數上經常浮動，並且持續成長，因此需要的數學與分析方法與以往有限的資料有很大的不同。

大資料計算大致可以分成幾個重要的部份，第一是資料處理，第二是重要訊息處理，第三是建模，最後是預測。每一部份在大資料計算的領域都可以獨立再單獨出來成為一個討論的議題。由於資料的來源與應用方式廣泛，上述的分類可能會因著重的應用領域不同而有不同的分類方式。

我們把資料處理也歸類於大資料計算的領域有兩個原因，第一是資料處理關乎效率問題，在讀取、搜尋與整理資料上演算法會直接影響獲取資料的速度，在即時服務的應用上，獲取資料的速度影響效能甚巨。第二是我們延伸John Gage宣示"網路即電腦"的概念[2]，資料庫也能抽象化成計算單元(CPU)的一部分，這部份是本論文的一個重點，旨在說明如何利用資料庫搜尋與暫存的特性來加速大型矩陣的運算。我們將模擬入口網站的瀏覽資料，透過計算網路使用者瀏覽資訊的紀錄，建構使用者之間的相關性，並且用快速MDS 方法將使用者投影到二維的平面上，之後我們可以利用二維平面上的距離來定義使用者喜好的相似度，這對產品和廣告的推薦有很大的幫助。

我們將完整地介紹如何利用Python 程式處理資料，建制sparse 大型矩陣的資料型別，再利用資料庫的特性來計算大型矩陣的過程，展示大資料計算會面臨的

問題，以及如何利用亂數投影(random projection)方法幫助SCMDS 計算並推演其計算量。希望透過這樣完整地介紹，讓有數學背景的讀者能分享到實作的經驗，並且對於網路服務應用有初步的了解。

1.1 為何使用Python 程式

在處理字串型的資料上Perl 一直是被推薦的一套語言，它具有搜尋快速以及跨平台的特性，對於資料整理可以得到很好的效果。然而Python 程式在處理字串型式的資料上與Perl 語言有相近的表現，除了跨平台與快速之外它有很好的社群在背後支持這套語言的發展與套件開發，使得Python 很迅速地在計算科學、生物資訊、電腦3D圖像、資料庫處理與維護以及遊戲領域上被廣泛接受。

對於數學學門的學生與研究者來說，由於Python 語言的發明人Guido van Rossum 也是數學背景的緣故，Python 語法的邏輯很容易被數學背景的人所習慣。另外Python 在資料型別上有很大的彈性，因此對於大型網站資料的處理與分析，新的資料型態不斷地產生並成長，Python 程式的彈性成為這套語言能被應用非常重要的關鍵。

一般的程式語言常用的型別有整數，浮點數，布林值，字元，字串，矩陣（或是陣列）。Python 中多了一些特別的型別使得在程式撰寫上，又多了一些彈性。首先，Python 中的串列（List）等同於Matlab 中的異質陣列（Cell），串列裡的元素型態不需拘泥於同一種格式，串列又可視為陣列的一種擴充，只是在型態上更具彈性。彈性有時是好事，但過度的彈性又會讓程式撰寫人員一不小心就製造一個難以糾舉的臭蟲(bug)，因此在Python 中特別出現一種無法更改內容的型別稱為元組（tuple）。在Python中串列是用中括號[...] 表示，而元組是用小括號(...) 表示。無論是串列或是元組，都是一種有序的型別，意思是可以用指標(index) 去讀取特定位置所儲存的資料。Python 的另一個特別的型別叫做辭典集，辭典集是一個無序的資料型別，用大括號{... }來表示，並且其中所儲存的資料都必須依附在一個金鑰底下，要讀取辭典內的資料必須用金鑰來開啓。例如A = {'name': 'John', 'age': 18, 'num': '98751065' }。要讀取A 辭典內的資料，不能用A{1}, A{1:2} 這種方式，因為辭典是一個無序的資料型別。A 中的name, age以及num 我們稱為金鑰，金鑰後面加一個冒號(:)，冒號後面所接的，就是存放在

這金鑰所保管的箱子裡的資料。

我們從串列以及辭典型別可以看到其變數的內容與資料長度彈性很大，正因為這樣的彈性可以適應於大型的網站資料分析。我們會在下一節介紹一些Python在大型靜態資料整理上的技巧，讓讀者看到Python 方便之處，並且介紹Python在資料庫處理上的應用，讓讀者看見如何利用Python 程式與網站的資料庫溝通訊息。

1.2 Python 在大型靜態資料讀取技巧

在處理大型靜態資料時，通常這一類的資料都會用某種特定的格式儲存，避免在檔案中出現過多為了分隔資料的分隔符號，使得原本龐大的資料量變得更無法處理。為了讓程式設計師可以知道哪一個位元所存放的資料是以何種型別儲存並知道它佔據多少位元組，定義這種格式的檔案我們稱為Profile。我們可以依照Profile 對格式的描述，理解每一筆資料所在的位置，並利用程式從龐大的檔案中讀出我們需要的資料。

健保資料庫[3]是全球僅有的全民醫療記錄並且可供台灣學者學術研究所用的資料庫，它累積了台灣兩千多萬人近十二年的病歷資料。 我們以健保資料庫93 年以後的門診處方及治療明細檔（CD檔）為例， 依照Profile[4]的說明，資料起始位置1到位置6 是記錄發生費用的年月，格式為YYYYMM，資料形態為C（character 亦即字元）。所以我們可以用char 的資料型別，將位元1 到6 的資料分別存放到一個長度為6 的字串中。例如：201108，我們就知道發生費用的時間是2011年8 月。起始位置第七個位元是存放申報類別，資料形態也是C，內容為字元1（代表送核）或是字元2（代表補報）。因此我們只要從檔案的第7 個位元讀出資料，內容就應該是1 或是2 。再來的第三個資料格式是醫事機構代號，佔有34 個位元，接下來是申報日期，佔有8 個位元。一筆CD 資料共有37 個變數，共佔據300個位元，因此第301個位元開始，就是另外一筆CD 資料。這樣在每一筆資料中的變數與變數之間就無須再用分隔符號來區隔變數，可以達到最精簡空間又容易讀取與儲存目的。

當我們使用Python 來讀取大型靜態資料時，有一些很值得分享的技巧值得分

享，我們放在附錄A供讀者參考。而藉由Python 操作Mysql 資料庫，在附錄B也有簡單的操作供讀者參考。

1.3 論文佈局

第一章主要是引起讀者對大資料計算的興趣，第一節介紹最適合處理大型資料的Python 程式語言；第二節介紹如何利用Python 程式的特性，使用最少的資源與最高的效能達到靜態大型資料整理的目的；第三節為本節。

第二章我們回顧一些在大資料計算上重要的數學定理與觀念，並在之後的小節介紹網路推薦系統主要會使用到的快速算法，第一節我們介紹亂數投影，這是一個當矩陣的大小過大時經常用來降低維度的方法；第二節我們介紹SCMDS 方法，這是傳統MDS 方法的加速版，對於我們作網路推薦系統中如何計算特徵空間有很大的幫助。

第三章我們要介紹當資料過大以至於無法用矩陣的方式來儲存時，我們要如何計算其特徵值或是計算MDS。我們以sparse 矩陣儲存方式的特性，撰寫sparse 矩陣的右乘法與左乘法，並計算其計算量。當乘法可行時，介紹如何利用修正亂數投影方法，透過正交亂數投影法將資料投影在估計的space 上，使得超大型矩陣的SCMDS 變成可計算。

第四章我們會以模擬資料來展示我們的方法是否能正確的將相似行為的使用者投影到靠近的位置，而將行為差異很大的使用者投影到距離較遠的位置，來說明我們的方法適合應用在大型網站上的推薦分析。

第五章根據第四章的實驗結果，下結論Python是適合作網路資料分析的程式語言並且利用Python語言可以在資料庫作降為的運算，因此資料庫是一個資料預先降維的快取空間。 而正交亂數投影可以改善亂數投影的誤差，配合這樣的結果，所以正交亂數投影與SCMDS可以協助大型網路資料分析。

而在附錄A以及附錄B中介紹一些Python的使用技巧，以及如何使用Python直接讀取與寫入Mysql 資料庫。

第 二 章　大資料計算相關數學回顧

在大資料計算中計算快速與精準是被同時要求的兩個要件，然而這兩個要件基本上是互相違背的。為了要計算的精準，所以會花費更多的時間。為了減少計算的速度，可能又會傷害了計算的精準度。因此如何拿捏尺度，便是一項藝術。然而在不同的應用領域上對要求的計算時間與精準度也會有所不同。例如生物資訊方面，準確度的要求會勝於速度的要求。因為生物實驗資料的取得，經常是以天或周甚至是月為單位的。所以，花了三個月取得的生物資料，實驗室並不在乎分析人員是花費一個星期還是兩個星期才把數值結果計算出來。這時計算的精準度便遠比速度來的重要。然而在網路應用服務上，由於訊息氾濫的緣故，訊息的些微差異通常不會嚴重影響到使用者的觀感，使用者更在乎的是使用媒體的流暢度。因此，這類應用服務要求的速度表現會遠比精準度來的重要。

大資料計算當然會希望在速度與準確度上都能同時提昇，因此我們會對於一些可透過調整計算時間來達到不同準確度的方法特別感興趣。以下我們要介紹兩個很有機會在大資料計算上用到的數值方法。第一個是random projection 演算法，這是計算大型矩陣需要降低維度時經常用到的方法，第二是拆解-合成多元尺度法(SCMDS)，這是一個可以快速作到傳統MDS 的方法。

2.1　亂數投影

給定一個矩陣$A \in M_{m,n}(R)$ 如果我們可以對A 矩陣實施奇異值分解（SVD: Singular value decomposition），

$$A = U\Sigma V^T,$$

其中U，V 是正交矩陣（orthonormal matrix），Σ 是一個對角矩陣，並且對角線的值都大於等於零$\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$。 我們令

$$U = \begin{pmatrix} | & & | \\ u_1 & \cdots & u_m \\ | & & | \end{pmatrix},$$

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & \\ 0 & \sigma_2 & \\ & & \ddots \end{pmatrix}$$

以及

$$V = \begin{pmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{pmatrix}$$

當我們希望用更精簡的矩陣計算來取代直接對A 矩陣作運算時，我們可以選取一個一個r 使得 $r < m, r < n$。並且定義

$$U_r = \begin{pmatrix} | & & | \\ u_1 & \cdots & u_r \\ | & & | \end{pmatrix} \in M_{m,r}(R)，$$

$$\Sigma_r = \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{pmatrix} \in M_r(R)$$

以及

$$V_r = \begin{pmatrix} | & & | \\ v_1 & \cdots & v_r \\ | & & | \end{pmatrix} \in M_{n,r}(R)$$

並且定義

$$A_r = U_r \Sigma_r V_r^T$$

作為A 矩陣的逼近矩陣。 這樣當我們要計算矩陣A 和其他向量x 的乘積時，我們計算$U_r * \Sigma_r * V_r^T * x$ 來取代Ax。

當矩陣很大時，SVD 分解本身就是一個計算量很大的問題，我們希望能降低矩陣的維度，又希望能保留矩陣的主項。這時候如何對矩陣A 的主項作估計就非常重要。 亂數投影（random projection）就是一個常用的方法，若$A \in M_{m,n}$ 代表A 是一個n 維資料，有m 個樣本。我們在A 矩陣的右邊乘上一個$R \in M_{n,r}$， 其中R 的每一個元素是亂數產生的，$A_R = A * R$ 就是一個從維度n 降到維度r 的資料。關於random projection 降維時對資料的影響，我們可以參考Fradkin and Madigan 在2003 年的作品[5]。

一般來說，我們作降維度時是希望資料透過一個線性轉換後能盡量保值其結構，例如先前我們用A_r 取代A 矩陣，因為SVD 的特性，前面的向量對應到的會是奇異值較大的部份，因此它是主項。雖然我們把$r+1$ 項之後的向量給省略了，但因為省略的部份對應的奇異值較小，對整體結構影響的程度也較小。

然而亂數投影卻沒有這樣的好處，嚴格說來亂數投影也不算是投影，因為我們並不能保證R 的每一列是互相正交的。但當n 夠大時，兩個亂數產生的向量會自然具有正交性，所以這部份我們可以姑且接受A 乘上R 之後像是個投影。另外一個問題是我們並不能控制R 的作用會分佈在A 的 row space 的主項上。除非我們特別挑選過，不然R 還是有很高的機率把資料投影到不是主項展開的子空間。下一章我們會對此部份做出修正，讓亂數投影多投影一些分量在我們希望的空間上。

2.2 拆解-合成多元尺度法

多元尺度法（MDS: Multidimensional Scaling）是一個把相對距離關係的資料轉換成低維度座標表示的方法，這方法最早是用在如何把三維的地球上的城市與城市之間的距離轉換成二維的地圖。多元尺度法希望高維度資料彼此間的距離，轉換成低維度座標表示時能繼續保持資料的結構。這個方法經常被使用在電腦圖學上，為了讓人們可以用視覺理解的方式來解讀資料。[6]

傳統的多元尺度法主要是使用差異矩陣（矩陣(i,j) 位置的值代表第i 個物件和第j 個物件間的距離）或是相似矩陣（矩陣(i,j) 位置的值代表第i 個物件和第j 個物件間的相似度）用Double centering 的方法轉換成另外一個矩陣，然後再對該矩陣作SVD 方解，因為該矩陣是對稱矩陣，所以我們可以用SVD 的結果對Double centering 之後的矩陣開方根，開方根後得到的矩陣取前面的r 行，就是資料在r 維空間上的座標表示。以下是詳細的傳統多元尺度的數學推導過程。

假設 X 是一個 p 乘N 的矩陣，其中N 是樣本數的個數，p 是資料的維度。$X - \frac{1}{N}X\mathbf{i}\mathbf{i}^T$ 相當於把X 的資料平移到質心為0。 令 $D = X^TX$ 是X 自己的直積，我們定義$\mathbf{i}$ 是一個N 乘1 的向量，它的每一個元素都是1。 我們定義一個B 矩陣為

$$\begin{aligned} B &= \left(X - \frac{1}{N}X\mathrm{i}\mathrm{i}^T\right)^T \left(X - \frac{1}{N}X\mathrm{i}\mathrm{i}^T\right) \\ &= D - \frac{1}{N}D\mathrm{i}\mathrm{i}^T - \frac{1}{N}\mathrm{i}\mathrm{i}^T D + \frac{1}{N^2}\mathrm{i}\mathrm{i}^T D\mathrm{i}\mathrm{i}^T \\ &= D - \bar{D}_r - \bar{D}_c + \bar{D}_g \end{aligned}$$

，其中 $\bar{D}_r = \frac{1}{N}D\mathrm{i}\mathrm{i}^T$ 是 D 矩陣沿著列方向作平均所成的矩陣（亦即給定$\bar{D}_r$ 的某一列，其每一元素都相等，都是D 在該列上的平均值）， $\bar{D}_c = \frac{1}{N}\mathrm{i}\mathrm{i}^T D$ 是 D 矩陣沿著行方向作平均所成的矩陣，以及 $\bar{D}_g = \frac{1}{N^2}\mathrm{i}\mathrm{i}^T D\mathrm{i}\mathrm{i}^T$ 是 D 矩陣作總平均所成的矩陣（$\bar{D}_g$ 是一個常數矩陣，其值為D 矩陣的總平均）。 從D 矩陣轉變乘 $D - \bar{D}_r - \bar{D}_c + \bar{D}_g$ 的過程我們稱為 double centering。 若我們定義一個新矩陣 H 為

$$H = I - \frac{1}{N}\mathrm{i}\mathrm{i}^T,$$

B 矩陣可以簡單寫成$B = HDH$，由於D 矩陣是對稱矩陣，B 矩陣也會是對稱。因為B 矩陣為對稱矩陣，所以B 的SVD 分解可以寫成

$$B = Z\Sigma Z^T。$$

因此，我們有

$$\sqrt{B} = Z\Sigma^{\frac{1}{2}}P^T = (X - \frac{1}{N}X\mathrm{i}\mathrm{i}^T)^T，$$

這裡的P 矩陣是某一個unitary matrix，在實作上我們經常選取$P = I$ 來取得MDS的結果，不同的P 只是把座標作一個旋轉而已，不會影響物件和物件的相對位置。 這個分析一開始使用的D矩陣還不是差異矩陣，真正的差異矩陣是定義成$d_{i,j} = \sqrt{(x_i - x_j)^T(x_i - x_j)}$，這時$D^{\cdot 2} = -2B$，其中$D^{\cdot 2}$ 代表D 的每一個元素自己平方。所以，當D 是名符其實的差異矩陣時，我們先把D 矩陣的每個元素自己平方後作Double centering ，然後在乘上$-\frac{1}{2}$，然後再作SVD取出B 的開方，就是MDS 的結果。 因為SVD 的結果是依照奇異值的大小排序的，所以當我們要把資料用二維的形式呈現時，我們只取Z 的前兩行與Σ 的左上角2 乘2 的子矩陣來計算二維的座標表示。若需要更高維度的表示就依此類推增加非零的奇異值與行數。

由於傳統的MDS 方法最主要的過程是SVD，這在矩陣的大小很大時會變得不可施行，因為SVD 的計算量是$O(N^3)$ ，所以當N 很大時，例如6 萬，現今的機器

大概都無法處理3 次方的計算量，因此我們需要可以因應大資料計算的多元尺度法。拆解-合成多元尺度法（SCMDS: Split-and-combine MDS）[7]就是傳統多元尺度法的一個改良，它的核心精神是把大資料拆解成好幾個重疊的子集合，先針對每一個子集合的資料作傳統的多元尺度法，然後再利用重疊的部份去計算如何把兩個不同的座標表示，合成為相同的座標表示。拆解資料，然後對個別的子集合作傳統多元尺度法是容易的，需要介紹的是如何把不同的座標表示，合成回一個座標表示。

若假設X_1 和 X_2 是兩個子集合重疊的部份分別用不同子集合群體計算傳統多元尺度法得到的座標表示。因此存在一個仿射映射(affine mapping)，可以將X_1 座標表示映到X_2 的座標表示。 令 $\bar{X}_1$ 和 $\bar{X}_2$ 分別是 X_1 和 X_2 行向量的平均向量。為了得到affine mapping 的內容我們應用QR 分解在 $X_1 - \bar{X}_1\dot{1}^T$ 和 $X_2 - \bar{X}_2\dot{1}^T$上。做平移是為了讓兩個重疊的資料質心都是零，這樣我們只要專注在旋轉的部份即可。

QR 分解後我們得到 $X_1 - \bar{X}_1\dot{1}^T = Q_1R_1$ 以及 $X_2 - \bar{X}_2\dot{1}^T = Q_2R_2$。 因為 X_1 和 X_2 本質上是相同的資料，因此$X_1 - \bar{X}_1\dot{1}^T$ 和 $X_2 - \bar{X}_2\dot{1}^T$ 這兩個座標表示的差別是旋轉。 因此，當沒有誤差干擾的情況下 R_1 和 R_2 應該是一樣的矩陣（最多在對應的行上是異號的）。 假設我們調整了R_1 和 R_2 的對角線的符號，對應Q_2的該行向量也要作同步的調整。

若 Q_2 經過調整後，我們有

$$Q_1^T(X_1 - \bar{X}_1\dot{1}^T) = Q_2^T(X_2 - \bar{X}_2\dot{1}^T)。$$

進一步我們可以將上式寫成

$$X_1 = Q_1Q_2^TX_2 - Q_1Q_2^T(\bar{X}_2\dot{1}^T) + \bar{X}_1\dot{1}^T。$$

因此，旋轉矩陣的部份就是 $U = Q_1Q_2^T$ 並且平移向量為 $b = -Q_1Q_2^T\bar{X}_2 + \bar{X}_1$。 有了旋轉矩陣和平移向量，我們就有把$X_2$ 轉換成X_1 的affine mapping。

雖然為了計算 affine mapping 使用到 QR 分解，而QR 分解的計算量是 $O(k^3)$， 這裡的k 是 X_1 和 X_2 的行數，亦即兩個重疊的集合重疊部份的元素個數。 因此，這 $O(k^3)$ 的計算量是受限於兩個重疊的集合重疊部份的元素個數，通常這數字是遠比原來的資料大小要小得多，雖然是三次方的計算量，但對整體計算而言卻是小數。以下是SCMDS 的計算量分析。

假設我們有 N 個樣本在資料裡，我們將這N 個樣本拆解成 K 群有重疊的子集合，其中 N_G 代表每一個子集合內所含元素的個數， N_I 代表重疊的區域的元素個數， 因此，我們有下列關係。

$$KN_G - (K-1)N_I = N$$

或寫成

$$K = \frac{(N - N_I)}{(N_G - N_I)}$$

。

對於每一個子集合來說，計算傳統MDS 需要 $O(N_G^3)$ 的計算量，為了合併每一個子集合計算出來 的座標表示，合併時在兩個子集合間需要 $O(N_I^3)$ 的計算量。 假設每一筆資料的維度是 p，並且每一個 N_I 的最小值是 $p+1$，為了方便起見，我們令 $N_G = \alpha p$ 其中 $\alpha > 2$。 因此整套的SCMDS 的計算量為

$$\frac{N-p}{(\alpha-1)p}O(\alpha^3 p^3) + \frac{N-\alpha p}{(\alpha-1)p}O(p^3) \approx O(p^2 N).$$

加法前面是計算K 群MDS 需要的計算量，後面那一項是計算連結 $K-1$ 個座標時需要的計算量。

當 $p << N$ 時，計算量 $O(p^2N)$ 會遠小於 $O(\sqrt{N}N)$，這樣的計算量會比當今最快速的MDS 方法 由Morrison et al, 於2003 年提出的最快速的MDS 方法[8]都還要快速 。 因此，我們會推薦SCMDS 方法在大資料計算上。

第 三 章　巨大資料的SCMDS方法

本論文所談論的巨大資料，是指資料大到無法用矩陣表示的方式儲存，並且變數與樣本數的數目是持續增加的，特別在網路應用服務時，這類的資料是以稀疏矩陣（Sparse matrix）的方式來表示可以達到最節省記憶體的方式。 這一章的第一節我們要介紹如何利用Python 程式以稀疏矩陣的方式來儲存資訊，並且做到和正常矩陣相同的矩陣與向量運算；第二節我們介紹在Sparse 矩陣格式下如何做到降維的動作；第三節我們介紹如何在Sparse 格式的資料上做SCMDS。

3.1　稀疏矩陣之儲存與運算

假設一個大型網站上有N 位使用者，網站內有M 個可供讀者點擊的內容（content）。N 與M 都隨著時間在增加。我們令$\mathcal{A}$ 是記錄此網站使用者在某一個時間區間的瀏覽歷史資料表，$\mathcal{A}$[i] 代表第i 位使用者的紀錄，$\mathcal{A}$[i] 的內容是一個辭典集，亦即

$$\begin{aligned}\mathcal{A}[\mathrm{i}] = \{\mathrm{j}{:}t_j | &\text{第}i\text{ 位使用者曾經點擊過第}j\text{ 筆內容，}\infty \leq | \leq \mathcal{M},\\ &t_j\text{ 是使用者}i\text{ 曾經點擊過內容}j\text{ 的次數，}t_j \in \mathcal{N}\}.\end{aligned} \tag{3.1}$$

辭典集裡面的每一個元素是一個由":" 分開的數對，前者記錄曾經點擊過的項目的指標，後者記錄曾經點擊過該項目的次數。若我們在Python 程式中用串列型變數A 記錄每一個辭典集，呼叫出第i 位使用者的點擊記錄正好就是$A[i]$，要呼叫出$\mathcal{A}$[i] 的第j 個項目的點擊數字，正好就是$A[i][j]$，亦即程式裡的$A[i][j]$ 正好就是稀疏矩陣$\mathcal{A}_{i,j}$ 的元素。並且若我們要修改$\mathcal{A}$[i] 的第j 筆資料內容，我們直接用$A[i][j] = x$ 來修改。這樣使用串列型別和辭典集型別來記錄稀疏矩陣的方式，可以讓讀取和修改稀疏矩陣的方式和一般矩陣的操作方法一樣。[9]我們特別用花體的英文字母代表資料存在資料庫中，正體的英文字母代表資料存在RAM裡。

1. **稀疏矩陣的加法**

   我們令n_i 為辭典集$\mathcal{A}$[i] 的元素個數，在Python 裡 $n_i = \mathrm{len}(A[i])$。 首先我們定義兩個資料表的加法。在使用意義上，若$\mathcal{A}$ 是第一天的點擊記錄， $\mathcal{B}$ 是

第二天的點擊記錄，我們想要整併這兩天的點擊記錄時，我們便希望能計算$\mathcal{A}+\mathcal{B}$，對於每一個Uid i 來說我們定義兩個辭典集的加法為

$$\begin{aligned}\mathcal{A}[\mathrm{i}]+\mathcal{B}[\mathrm{i}]=\{\mathrm{j}{:}t_u| \quad & t_u=\mathrm{A[i][j]+B[i][j]} \text{ if } j\in(\mathrm{A[i].keys()}\cap\mathrm{B[i].keys()});\\ & t_u=\mathrm{A[i][j]} \text{ if } j\in(\mathrm{A[i].keys()}-\mathrm{B[i].keys()});\\ & \text{and } t_u=\mathrm{B[i][j]} \text{ if } j\in(\mathrm{B[i].keys()}-\mathrm{A[i].keys()})\}\text{，}(3.2)\end{aligned}$$

若i 小於等於 $\mathcal{A}$ 或 $\mathcal{B}$ 的長度；

$$\mathcal{A}[\mathrm{i}]+\mathcal{B}[\mathrm{i}]=\mathcal{A}[\mathrm{i}]\text{，}$$

若$\mathcal{B}[\mathrm{i}]=\emptyset$；

$$\mathcal{A}[\mathrm{i}]+\mathcal{B}[\mathrm{i}]=\mathcal{B}[\mathrm{i}]\text{，}$$

若$\mathcal{A}[\mathrm{i}]=\emptyset$。$\mathcal{A}+\mathcal{B}$ 即對每一個i 都做一次上述加法，這樣我們就可以將兩段不同時間發生的資料表整併成一個資料表。計算兩個稀疏矩陣加法的Python 程式如下：

```
def dict_plus(A,B):
    #type(A) is list,type(A[i]) is dict
    #type(B) is list,type(B[i]) is dict
    from copy import copy
    C=copy(A)
    for i in range(len(A)):
        Ak=A[i].keys()
        #Aks=set(Ak)
        for Bk in B[i].keys():
            if Bk in Ak:
                C[i][Bk]+=B[i][Bk]
            else:
                C[i][Bk]=B[i][Bk]
    if len(A)<len(B):
        for i in range(len(B)-len(A)):
            C.append(B[len(A)+i])

    return C
```

2. **稀疏矩陣右乘向量**

要能夠進行稀疏矩陣的運算，在不把矩陣全部展開的情況下，我們一定要有矩陣右乘向量的功能[10]。要在電腦裡開一個很長的向量並不困難，所以我們直接用正常的陣列方式來儲存。令$x[i]$ 為 x 的第i 個元素我們定義$y = \mathcal{A}$x為

$$y[j] = \sum_{i \in \text{A[j].keys()}} A[j][i] * x[i]。$$

稀疏矩陣的右乘非常精簡，只要把金鑰（keys）抓出來計算即可。對應的Python 程式如下：

```
def Ax(A,x):
    #A:1xN  type(A) is list, type(A[i]) is dict
    #Al:1xN type(Al) is list w.r.t A
    #x: Nx1 type(x) is list
    #Let x be 1xN
    Al=len(A)
    t=[0 for j in range(Al)]
    for i in range(Al):
        tmpA=A.pop()
        j=len(A)-1
        for k in tmpA.keys():
            t[j]=t[j]+(x[k])*(tmpA[k])
    return t
```

在這裡用到附錄A的後端拋值(.pop())，為了減少記憶體的使用。

定理 若$\mathcal{A}$ 是一個高度為N 的稀疏矩陣且$\mathcal{A}[i]$ 的平均長度為n，$n << N$，則$\mathcal{A}x$ 乘法的計算量為$O(N)$。

證明： 令 n_i 等於$\mathcal{A}[i]$ 的元素個數，因為

$$y[j] = \sum_{i \in \text{A[j].keys()}} A[j][i] * x[i]。$$

所以得到$y[i]$ 需要的計算量為n_i。因此計算$\mathcal{A}x$ 的總計算量為

$$\sum_{i=1}^{N} n_i = N(\frac{1}{N}\sum_{i=1}^{N} n_i) = Nn = O(N)。$$

上述定理說明，若$\mathcal{A}$ 每一列的長度遠小於$\mathcal{A}$ 的高度時，計算稀疏矩陣的計算量是線性的。為了爭取計算速度，若N 還是過大使得線性的計算量仍舊無法滿足即時服務時，因為計算每一個$y[i]$ 都是獨立的，因此我們可以用多核心計算的方式來加速這一個乘法運算。

3. **稀疏矩陣左乘向量**

接下來我們定義稀疏矩陣左乘向量的運算，因為資料結構的緣故，左乘的程式沒有右乘直覺。若$y = x^T\mathcal{A}$

$$y[i] = \sum_{j \in \mathcal{I}_i} x[j]A[j][i]，$$

其中

$$\mathcal{I}_i = \{j | i \in A[j].keys()\}。$$

對應的Python 程式如下：

```
def xA(A,x):
    #A:1xN  type(A) is list, type(A[i]) is dict
    #y : 1xN type(x) is list
    #x:1xN
    d=dict()
    Al=len(A)
    for i in range(len(x)):
        tmpd=[0 for j in range(len(x))]          #set initial tmpd
                                                 #A[i]*x[i]
        tmpA=A.pop()
        j=len(A)-1
        for k in tmpA.keys():
            tmpd[k]=tmpd[k]+(x[j])*(tmpA[k])
        d[i]=tmpd                                #d={i:tmpd_i}
    t=[0 for i in range(len(x))]                 #set initial t
    for k in d.values():
for i in range(len(k)):
t[i]=t[i]+k[i]
```

```
    return t
```

定理 若$\mathcal{A}$ 是一個高度為N 的稀疏矩陣且$\mathcal{A}[i]$ 的平均長度為n，$n << N$，則$x^T\mathcal{A}$ 乘法的計算量亦為$O(N)$。

4. **稀疏矩陣的成對距離矩陣**

稀疏矩陣若要計算成對距離矩陣（pair-wise distance matrix），由於每一行的資料多半是無需記錄的零資料，因此在計算成對距離矩陣時的算法也略有不同。對於正常的矩陣$A \in M_{m,n}(R)$ 來說，其成對距離矩陣D 的定義為

$$D_{i,j} = \sqrt{\sum_{k=1}^{n}(A_{i,k} - A_{j,k})^2}。$$

計算稀疏矩陣的成對距離矩陣$\mathcal{D}$ 時，$\mathcal{D}_{i,j}$ 定義如下

$$\mathcal{D}_{i,j} = \sqrt{\sum_{k\in\mathcal{I}_{i\sim j}} A[i][k]^2 + \sum_{k\in\mathcal{I}_{i\wedge j}}(A[i][k] - A[j][k])^2 + \sum_{k\in\mathcal{I}_{j\sim i}} A[j][k]^2}，$$

其中

$$\mathcal{I}_{i\sim j} = \{k|k \in \text{A[i].keys() and } k \notin \text{A[j].keys()}\}，$$

$$\mathcal{I}_{j\sim i} = \{k|k \in \text{A[j].keys() and } k \notin \text{A[i].keys()}\}，$$

以及

$$\mathcal{I}_{i\wedge j} = \{k|k \in \text{A[i].keys() and } k \in \text{A[j].keys()}\}。$$

對應的Python程式如下:

```
def distance(A):
    Al=len(A)
    D=[[0 for x in range(Al)] for y in range(Al)]
    for i in range(Al):
        tmpA=A.pop()
        tmpj=len(A)-1
        for j in range(len(A)):
            InterA=set(A[j].keys()).intersection(set(tmpA.keys()))
```

```
            disjA1=set(A[j].keys())-InterA  # j's key not in tmpA.keys()
            disjA2=set(tmpA.keys())-InterA  # tmpA's key not in j.keys()
            tmpDij=0
            for k in disjA1:
                tmpDij+=(A[j][k])**2
            for k in disjA2:
                tmpDij+=(tmpA[k])**2
            for k in InterA:
                tmpDij+=(A[j][k]-tmpA[k])**2
            D[j][tmpj+1]=tmpDij
            D[tmpj+1][j]=tmpDij

    return D
```

上述的運算和正常的矩陣運算非常類似，特別把有值的指標寫成集合的方式，是為了强調在即時運算時，我們不能浪費時間在沒有意義的搜尋上面。這樣才能真正做到線性的計算複雜度。

3.2 巨大資料的降維方法

假設一個矩陣型的資料$A \in M_{m,n}(R)$，其中m 是樣本數，n 是樣本資料的維度。若維度過大時，我們要對樣本做分類或分析時會遇上維度咒詛的限制，降低分析的精準度，因此有經驗的分析者都會在分析高維度資料前做降維的動作。先前我們介紹了亂數投影法如何應用在降低資料維度。這一節我們說明如何在巨大的稀疏矩陣上套用亂數投影法。

由於亂數投影可能會因為亂數選取基底的緣故，造成資料的結構扭曲（我們將在下一個章節顯示這個性質），我們會修正亂數投影的過程，讓使用亂數投影降維的結果，與原來的資料結構更加近似。

我們給一個亂數矩陣$R \in M_{r,N}$，其中 r 是我們想要達到降維的維度，每一個$R_{i,j}$ 都取值於$[0,1]$區間的均勻分佈。我們令$P = R\mathcal{A}$，再將P^T 對行向量做正交

化得到Q ，這裡的Q 是orthonormal。接著我們令$A_r = \mathcal{A}Q$，則A_r即為降維之後的矩陣。

在這裡$R \in M_{r,N}$ 使用到稀疏矩陣的左乘向量，將P^T 正交化這裡會用到$\frac{1}{2}r^2N$的計算量，這也是$O(N)$ 的水準。再計算$\mathcal{A}Q$ 也是rN 的計算量，因此，稀疏矩陣的降維都是$O(N)$的計算量。

這裡的降維方法和亂數投影的方法差別在於亂數投影時，矩陣Q 是未經正交化的亂數矩陣，而我們的Q 矩陣是透過亂數左乘矩陣$\mathcal{A}$ 之後，得到矩陣$\mathcal{A}$ 的row space 主項後，再正交化得來。

一開始我們會先選擇在資料庫上作亂數正交投影，之後載下資料再進行SCMDS計算。這是因為要藉由亂數正交投影解決有未確定值的因素，在第四章實驗會再詳述。

3.3 巨大資料的SCMDS技巧

若一個大矩陣$A \in M_{m,n}(R)$ 利用SCMDS 降維，我們會先亂數將A 的列向量重排，在假設A 的秩（rank）小於r 的情況下，給定$N_I > r$ 以及 $N_G = 3N_I$作為SCMDS 的參數，其中N_G 是每一個分群的個數，N_I 是群與群交集的元素個數。$N_I > r$ 的條件，可以保證經過SCMDS 降維後的資料結構與降維前的結構相似。

細部的做法是，將分群後的矩陣個別計算其成對距離矩陣，然後對此成對距離矩陣計算傳統MDS 方法，然後用交集的元素的MDS 坐標，利用QR 分解取得Affine mapping 的旋轉矩陣和平移向量，然後將兩個坐標合併成一個坐標系。

同樣的過程在巨大資料時，我們可以用稀疏矩陣計算成對距離矩陣的方式，來取得每一個分群的成對距離矩陣。其餘SCMDS 步驟皆與大矩陣的SCMDS 步驟相同。然而這裡有一個難題，就是如何估計巨大資料所對應的稀疏矩陣的秩（rank）。即使每一列的稀疏矩陣紀錄的長度是很小的，我們也無法貿然相信，它的秩是小的。例如，identity matrix，用稀疏矩陣表示每一列只有一個元素非零，然而它卻是滿秩(full rank)。

我們也知道，若N_I 的設定小於矩陣的秩，則SCMDS 會荒腔走板。因此，如何能穩定地在巨大資料上使用SCMDS 降維，製作出物件彼此的關係圖，是我們介紹亂數投影的原因。透過預先使用亂數投影讓矩陣由$M \times N$ 降至 $M \times r$，這部分的結構會稍微損失一些，但大部分的結構仍舊被保持，然後再利用SCMDS 選取$N_I > r$ 來進行不會損失結構的降維，來達到巨大矩陣降維的目的。我們將在下一章比較不使用亂數投影與使用亂數投影的SCMDS 的差別，讓讀者知道對無法估計正確秩的稀疏矩陣，亂數投影有其必要性。

第 四 章　實驗結果

本章將介紹電腦模擬實驗的結果，第一節我們在小矩陣上比較亂數投影與正交亂數投影的差別，說明正交亂數投影的必要性；第二節我們在稀疏矩陣上比較SCMDS 降維與混合正交亂數投影之SCMDS 降維，說明混合正交亂數投影之SCMDS 降維可以得到較好的穩定性；第三節我們介紹如何應用混合正交亂數投影的SCMDS 降維法在大型網站建立使用者偏好關係圖的過程。

4.1　小矩陣亂數投影與亂數正交投影之比較

首先我們先製作一個小樣本的模擬資料，共計400個樣本，每一個樣本都是維度200的資料。首先定出前兩個維度，有50筆資料是以(0,0)為圓心並且半徑為10的資料，有50筆資料是以(0,4)為圓心並且半徑為1.5的資料，有50筆資料是以(3,0)為圓心並且半徑為1.5的資料，有50筆資料是以(-3,0)為圓心並且半徑為1.5的資料，剩下200筆用程式取亂數取點。之後將後面198個維度都用程式取亂數乘上0.05補上。

這400個樣本的前兩個維度是主要結構，其中50 個點圍成一個大圈，另外三組50 個點各以三角形的三個頂點圍成三個小圈，另外200個點是亂數產生，這400個點的主結構如圖 4.1(a)其餘維度的資料，我們用不影響結構的亂數產生。

因此，資料為400 乘200 的矩陣，這樣的矩陣大小，可以用傳統的MDS 方法，來觀察資料在沒有計算損失的情況下，其結構為何，方便我們比較其他MDS 方法。利用傳統MDS 方法得到的2 維結構投影如圖 4.1(b)，這結果和只畫出資料前兩維的結果相似。

接著我們使用亂數投影，將資料分別從200 維降到50、40、30、20和10維，之後將降維後的資料進行傳統的MDS計算，得到的2維較穩定的結構投影圖圖 4.2(e)、 4.2(c)、 4.2(a)、 4.1(e)、 4.1(c)。因為亂數投影有時會造成結構的破壞，因此我們把不穩定的結構投影圖列出供讀者參考。對應維度為50、40、30、20和10維的結構投影圖如 4.2(f)、 4.2(d)、 4.2(b)、 4.1(f)、 4.1(d)。

再來我們使用亂數正交投影，亦即我們先用亂數投影的方式取得列矩陣，然後我們對列矩陣做正交化，在利用正交化後得到的orthonormal matrix 做降維的投影。因為這方法得到的結果都很穩定，所以，我們只展示一批結構圖，對應降維50、40、30、20和10維的2維結構圖，其結果如圖 4.3(e)、 4.3(d)、 4.3(c)、 4.3(b)和 4.3(a)。由圖二到圖 4.3(a)我們得知，正交亂數投影的確可以保持降維後的結構。

為了更精確地度量不同的方法對2維結構的影響，我們利用比較2 維投影資料的成對距離矩陣的差異，來度量不同方法之間的穩定性。圖 4.4(a)是直接使用SCMDS降維與SCMDS 以及混合亂數正交投影之SCMDS 降維的比較，方法是先用MDS 方法計算其2 維投影結構，然後比較SCMDS 以及混合亂數正交投影之SCMDS 降維至50、40、30、20和10維的2 維投影結構的相對誤差，橫軸表示維度，縱軸表示相對誤差值。在上方的線段是SCMDS作降維與MDS比較的相對誤差，下方的線段是混合亂數正交投影之SCMDS 降維與MDS比較的相對誤差，明顯看到SCMDS的降維相對誤差較混合亂數正交投影之SCMDS 降維大，所以更確定用亂數正交投影之SCMDS這方法是較佳的。

4.2 SCMDS與亂數正交投影之SCMDS 於稀疏矩陣之比較

接著我們比較稀疏矩陣在SCMDS 與混合亂數正交投影之SCMDS 的差異。因為SCMDS 要能正確顯示資料結構的前提是拆解分群時，群與群交疊的元素個數要大於資料結構的主要維度，若低估了資料結構的主要維度使得交疊的元素個素過低，則SCMDS 計算後的降維結構，經常是荒腔走板的。

稀疏矩陣很容易出現列資料很稀疏，但是矩陣的秩仍舊很高的情況，因此我們用稀疏矩陣來比較標準SCMDS 降維與混合亂數正交投影之後的SCMDS 降維。我們建立一個大小為2000乘1000 的稀疏矩陣矩陣。先用MDS 方法計算其2 維投影結構，然後比較SCMDS 以及混合亂數正交投影之SCMDS 降維至100、90、80、70、60、50、40、30、20和10維的2 維投影結構，而圖 4.4(b)就是比較相對誤差的結果，橫軸表示維度，縱軸表示誤差值。在上方的線段

是SCMDS作降維與MDS比較的相對誤差，下方的線段是混合亂數正交投影之SCMDS 降維與MDS比較的相對誤差，明顯看到SCMDS的降維相對誤差較混合亂數正交投影之SCMDS 降維大，所以確定即使在稀疏矩陣上，用亂數正交投影之SCMDS這方法也是較佳的。

4.3 大型網站資料建立使用者偏好關係圖實作

現今的網站服務已經不像早期的靜態網站只提供單項的網頁內容提供網友閱讀，在大型入口網站的建制上，通常也不是只有一臺伺服器來面對廣大的使用者，一般的情況下，會是許多伺服器同時在線上服務，因此我們會將使用者進行切割，讓不同偏好的使用者由不同的伺服器來服務。進而，對於不同偏好的使用者，網站會進行對應的廣告服務與內容推薦。因此，如何建制即時的使用者偏好關係圖，讓伺服器有更清楚的原則對廣告做精準地推播，是現在很熱門的課題。

要對使用者的偏好做分析與預測，我們對每一個使用者會要求一個獨一無二的代號，稱為Uid（User index），對於每一個網站上的資料（項目）我們會給一個Tid（Term index），將使用者在一段期間點擊過某項目的次數記錄下來的矩陣，我們稱項目與使用者矩陣（Term-User matrix），我們會因為使用者點擊項目的記錄，來區分使用者與使用者彼此間的相似程度。透過相似程度的描述，將使用者投影在較低維度的空間，然後對使用者進行分類，我們就能知道哪些使用者之間有類似的偏好，進而對該類使用者進行更精確地分析與廣告推薦。

首先我們先模擬一個大型網站上的使用者點擊記錄，利用稀疏矩陣的方式來儲存這些記錄。由於資料龐大的特性，若將資料全部讀入記憶體再計算，讀取資料的時間往往都大過網路服務可以忍受的時間。因此，我們在上一節才會特別强調稀疏矩陣的運算，這些看似多餘的運算寫法，對應的就是直接利用資料庫的搜尋和寫入功能，把資料庫當做CPU 旁的cache，讓資料庫的處理成為計算單元的一部份。藉由資料庫先進行初步的降維，使用的方法是亂數正交投影，接著再使用SCMDS來定出使用者的座標位置，之後觀察這些座標位置，來分析使用者的偏好關係。而先使用亂數正交投影的方式將資料降維而不直接使用SCMDS 將資料降維，主要是因為資料有未確定值。使用者沒有點擊的項目，不見得是因為他沒有興趣，而模擬的資料就是有這樣的未確定值存在，有未確定值的時候就

無法直接使用SCMDS。加上在上一節可以看到單純作SCMDS與亂數正交投影混合SCMDS相比，亂數正交投影混合SCMDS是相對較佳的。因此實驗流程是利用在資料庫上先藉由亂數投影的方式來降維，之後載下資料再進行SCMDS計算，藉此找出使用者的偏好關係。

我們模擬10400個使用者的大型網站記錄，每一個使用者點擊單項項目的次數不超過5，並且每一個使用者點擊的項目總數不超過2000。對應的超大稀疏矩陣我們計算其亂數正交投影的SCMDS方法，先用亂數正交投影的方式將資料降維至200，然後再用SCMDS 將資料降維至30，最後再繪出2 維的使用者偏好關係圖。

在模擬資料時，先將資料分成三種

1. 任意:任意是指利用程式亂數選取少於1000個指標當作使用者點擊的項目，再利用程式亂數選取1到5的整數當作項目的點擊次數，以此方法模擬10000筆資料。

2. 相似:一開始先亂數選取一個大於等於80小於1000的整數u來決定點擊項目的個數，再任意選取u個項目的指標$j = (j_1, j_2, ..., j_u)$，再利用程式亂數選取1到5的整數u次，使得次數$v = (v_1, v_2, ..., v_u)$)為對應項目的點擊次數，藉此當作建構相似群的標準$D = \{j_1 : v_1, j_2 : v_2, ..., j_u : v_u\}$。之後再細分成三種

   (a) 次數加一:任意從$j = (j_1, j_2, ..., j_u)$選取一個指標，將選取到的指標k，次數加一，也就是$\{j_1 : v_1, j_2 : v_2, ..., j_k : v_k + 1, ..., j_u : v_u\}$。這樣的使用者共100個。

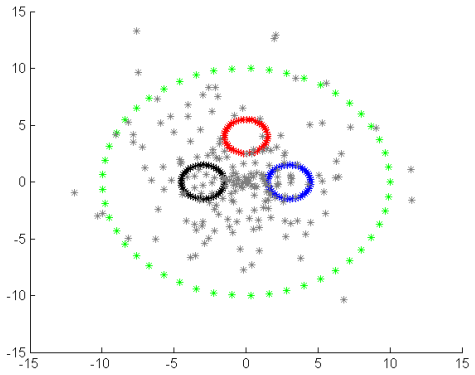
   (b) 次數少一:任意從$j = (j_1, j_2, ..., j_u)$選取一個指標，將選取到的指標k，次數少一，也就是$\{j_1 : v_1, j_2 : v_2, ..., j_k : v_k - 1, ..., j_u : v_u\}$。這樣的使用者共100個。

   (c) 多一項目:任意在$j = (j_1, j_2, ..., j_u)$之外選取一個指標$s, s \notin \{j_1, j_2, ..., j_u\}$，並且令其次數為1，也就是$\{j_1 : v_1, j_2 : v_2, ..., j_k : v_k - 1, ..., j_u : v_u, s : 1\}$。這樣的使用者共100個。

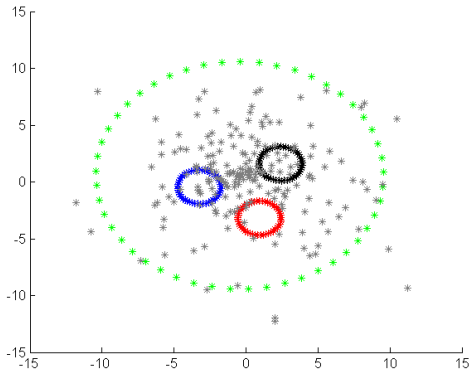
3. 不相關:利用程式亂數選取j以外的指標，並且每一使用者點擊的項目並不重複，使他們點擊的內容毫無交集，共100筆。

這樣我們就可以透過觀察相似的是否座落在同一個區域，以及不相關的是否在關係圖上是分散的，來驗證我們的方法是否能成功應用在巨大資料上。其結果如圖 4.3，由圖形可以看到黃色與藍色都集中在縱軸上面，並且藍色代表的是毫無交集的那一群，他們是靠近縱軸的下方，是比較無意義的部分，而在右方縱軸較遠的紅色那一群是交集很大，也就是相似的那一群(偏好極為相同)，他們是較具意義的部分，可藉由他們較為相似，推斷他們可能具有相同的偏好或行為。之後將相似的群另外作MDS，單看相似群，如圖 4.6(a)所示，看到大部分都緊緊的靠在一起。在圖 4.6(b)中，則是將不同群，用不同的圖型標記，分別使用菱形和原圈O，以及加號+，而利用顏色代表他們些許的不同，紅色與藍色是代表這兩個使用者他們都點擊相同的項目，只是點及其中一個項目次數多1；而藍色與黑色的差異在於，黑色座標的使用者比藍色的使用者多點擊一個項目，並且點及次數為1。

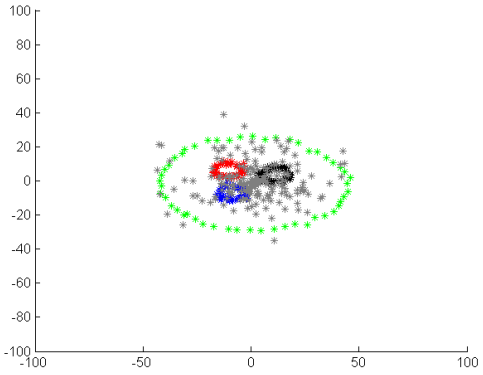
以上詳細的code請參考`http://dl.dropbox.com/u/16496288/material.rar`。若有疑問可以寫mail到look76god@gmail.com給作者。



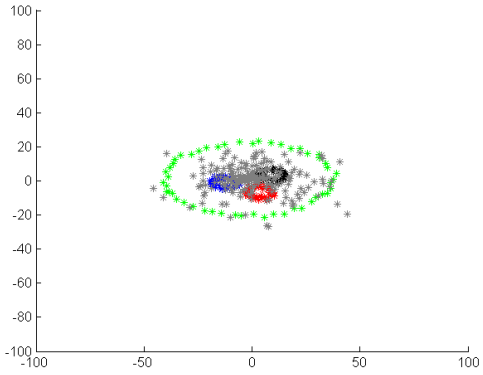
(a) 原資料的圖



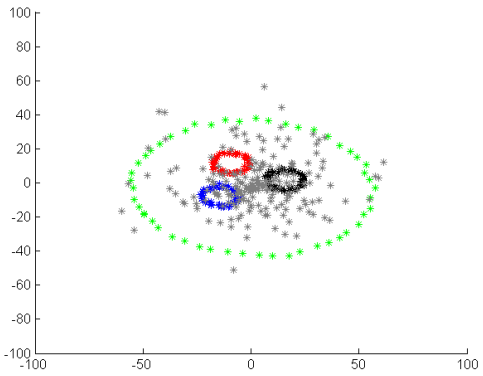
(b) 傳統MDS的圖



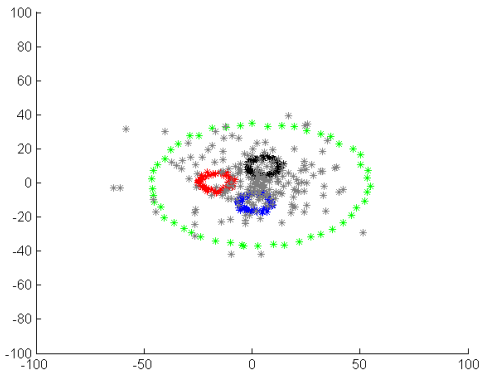
(c) 維度降到10的random projection 穩定



(d) 維度降到10的random projection 不穩定

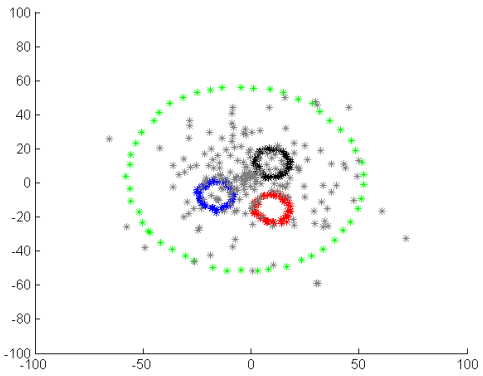


(e) 維度降到20的random projection 穩定

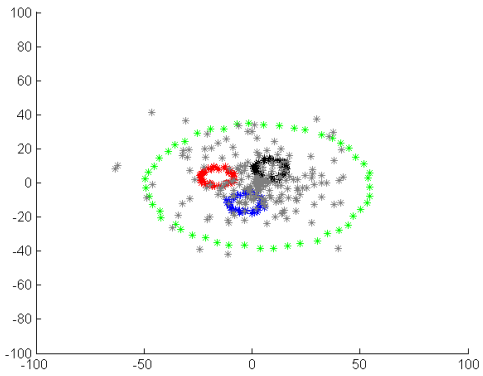


(f) 維度降到20的random projection 不穩定

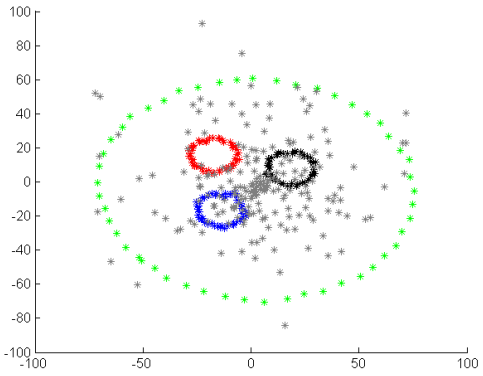
圖 4.1: 原資料的圖與各種方法的比較



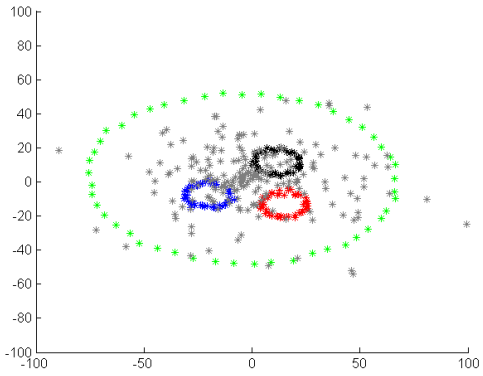
(a) 維度降到30的random projection 穩定



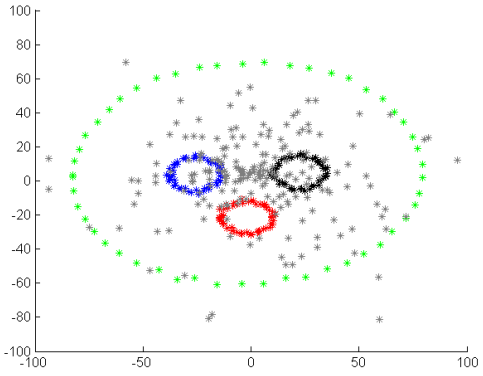
(b) 維度降到30的random projection 不穩定



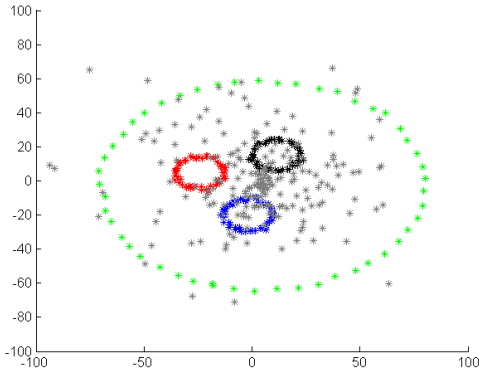
(c) 維度降到40的random projection 穩定



(d) 維度降到40的random projection 不穩定

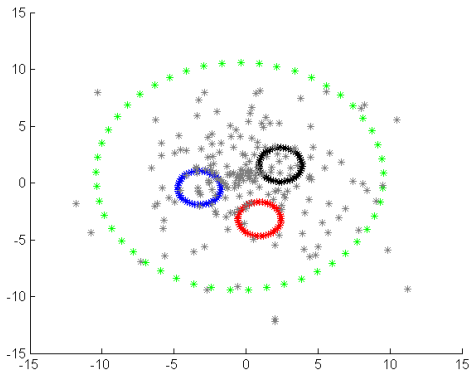


(e) 維度降到50的random projection 穩定

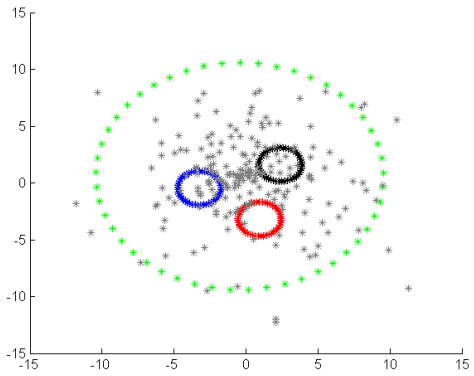


(f) 維度降到50的random projection 不穩定

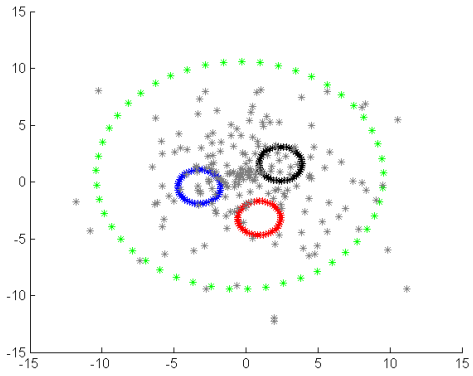
圖 4.2: 降維度random projection的結果比較



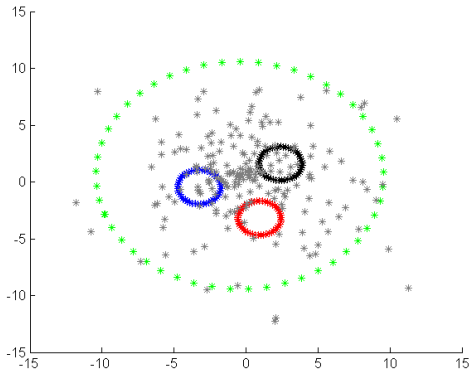
(a) 維度降到10的random projection並且作正交化



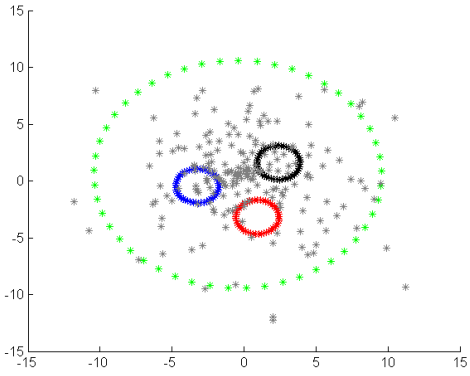
(b) 維度降到20的random projection並且作正交化



(c) 維度降到30的random projection並且作正交化

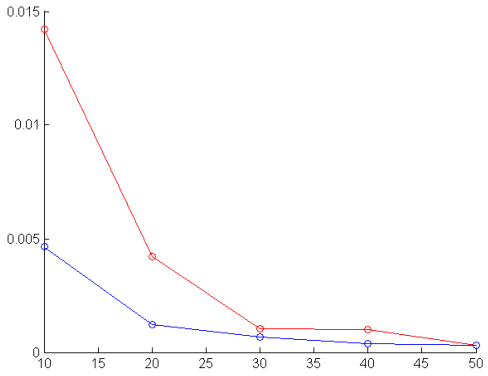


(d) 維度降到40的random projection並且作正交化

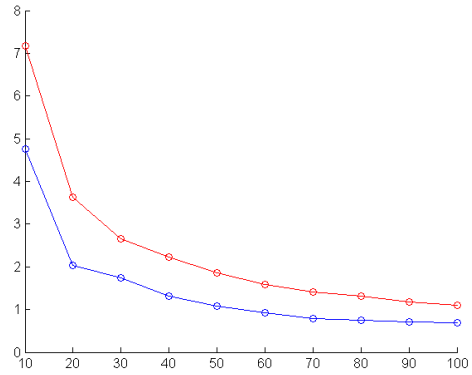


(e) 維度降到10的random projection並且作正交化

圖 4.3: 降維度random projection並且作正交化的結果比較



(a) 小矩陣亂數直接作SCMDS降維與正交化作SCMDS降維5個維度的相對誤差比較



(b) 稀疏矩陣直接作SCMDS降維與正交化作SCMDS降維各維度的相對誤差比較

圖 4.4: 直接作SCMDS降維與正交化作SCMDS降維的相對誤差比較

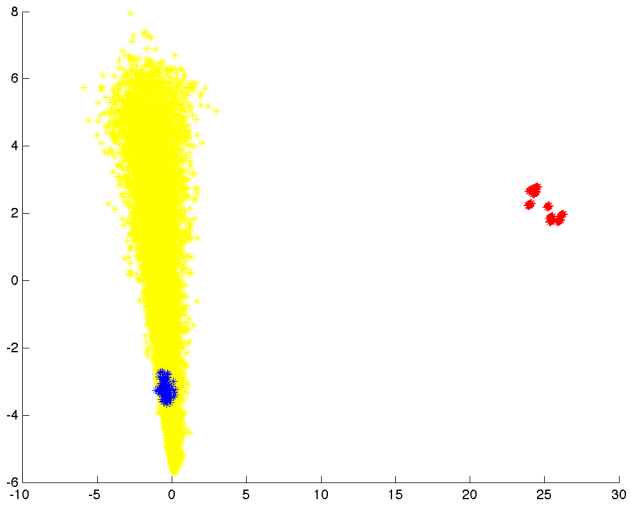
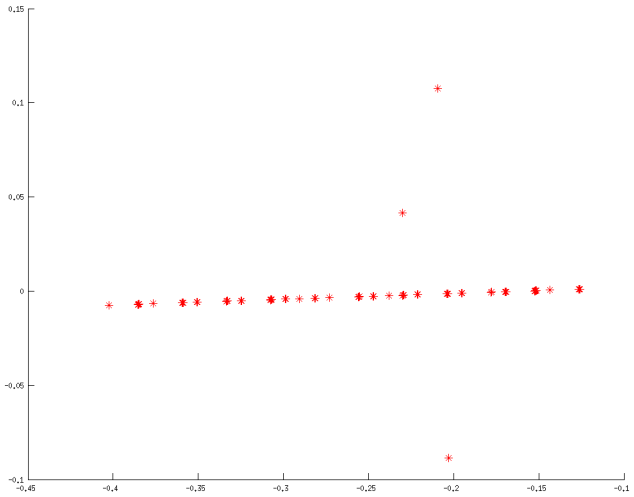
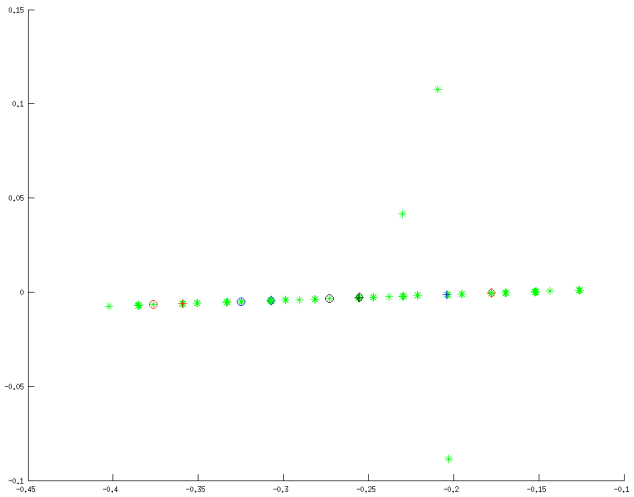


圖 4.5: 模擬10400筆資料作亂數投影正交化與SCMDS的結果



(a) 10400筆資料相似的群



(b) 10400筆資料相似的群之間的比較

圖 4.6: 10400筆資料相似的群

第 五 章　結論

藉由本論文實驗結果，我們可以得到以下結論:

- Python是適合作網路資料分析的程式語言。
- 資料庫是一個資料預先降維的快取空間。
- 正交亂數投影可以改善亂數投影的誤差。
- 正交亂數投影與SCMDS可以協助大型網路資料分析。

由於Python此程式語言可以任意混用型別，所以在儲存資料上是非常方便的，並且它的運算速度也非常的快速，加上他有許多社群支持擁有許多完整的套件，所以我們實驗方法用到的亂數投影以及SCMDS都是可以用它來完成的，所以它真的是一個適合作網路資料分析的程式語言。

Data is cache for dimension reduction.資料庫是一個資料預先降維的快取空間。我們利用在資料庫上先藉由亂數投影的方式來降維，那何不直接使用SCMDS進行降維，而要使用亂數投影? 這是因為我們的資料有未確定值，在這樣的情況下是無法直接作SCMDS，所以必須先藉由亂數投影降維，把有未確定值這因素消除才可進行SCMDS。那在資料庫用亂數投影降維不僅可以使資料庫分擔一些計算，並且還可以達到降維的作用，之後載下資料再進行SCMDS計算，藉此找出使用者的偏好關係，能藉此分析進行下一步的推薦服務，讓此資料成為我們推薦的一個基礎，能更有效率且精確的給予建議。

藉由上一章的第一節可以確認正交亂數投影可以改善亂數投影的誤差，經由正交的修正，可以確保亂數投影的穩定性，並且維度就算降到很低對於資料的結構也不會影響很大。所以在作亂數投影時，若要降的維度多，選擇正交亂數投影準沒錯。

我們利用在資料庫進行正交亂數投影降低維度，再藉由SCMDS定出使用者間的座標，利用這些座標可以對相似的使用者進行合適的推薦，所以正交亂數投影與SCMDS可以協助大型網路資料分析。

參考文獻

[1]http://www.whitehouse.gov/blog/2012/03/29/big-data-big-deal

[2]http://en.wikipedia.org/wiki/John_Gage

[3]http://nhird.nhri.org.tw/

[4]http://nhird.nhri.org.tw/date_01.htm

[5]Fradkin, D. and Madigan, D: Experiments with random projections for machine learning, *In Proceedings of KDD-03, The Ninth International Conference on Knowledge Discovery and Data Mining* , 517-522,2003

[6]Borg, I., Groenen, P.: Modern Multidimensional Scaling: theory and applications (2nd ed.),*New York: Springer-Verlag*,pp. 207–212. ISBN 0-387-94845-7,2005.

[7]Jengnan Tzeng1, Henry HS Lu2 and Wen-Hsiung Li : Multidimensional scaling for large genomic data sets ,*BMC Bioinformatics* , Vol.9, No.179.(SCI),2008.04.

[8]Morrison A, Ross G, Chalmers M: Fast multidimensional scaling through sampling, springs and interpolation,*Information Visualization* 2:68-77,2003.

[9]F.G. Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition.*ACM Transactions on Mathematical Software* , 4(3):250–269, 1978.

[10]X. Huang and V.Y. Pan. Fast rectangular matrix multiplications and applications ,*Journal of Complexity* , 14:257–299, 1998.

附錄A：Python技巧分享

1. **pop() 指令由後端拋值**:

   pop() 指令在Python 的串列、集合以及字典型別都有pop() 這個函數可以使用，特別在串列型別時pop() 可以有序的將資料丟出，並且讓丟出的資料自動移除。舉例來說，若a = [1,2,3,4] 是一個長度是4 的串列。a.pop() 會自動把a 的最後一個元素丟出，亦即4。之後a 串列就成為 [1,2,3] 長度為3 的串列。我們也可以指定要丟出的資料位置，例如，a.pop(1)，會把a 的index 等於1 的位置的該元素丟出，並且讓a 的長度變短。

   舉例來說，當我們要計算一個樣本數很大的資料彼此間的距離（pair-wise distance）時，就很適合用到pop() 指令。假設a 是一個高度是3000 寬度是2 的numpy.ndarray，a.shape = (3000,2)，其中3000 代表樣本的數目，2 代表樣本的維度。若我們要計算這三千個樣本彼此間的距離。我們用如下的程式來計算：

```
import numpy as np
D = np.zeros([3000,3000])
while len(a)>0:
    b = a.pop()
    for i in range(len(a)):
        D[len(a),i] = np.linalg.norm(b-a[i,:])
D = D+D.T
```

   當資料很大時，隨著pop() 指令從最後一個index 所指到的元素開始往外丟，一來使得資料長度越來越短，二來程式不需要管index 需要重排的問題。倘若我們只使用for 迴圈來撰寫，我們就需要去計算當我們拿出第i 個index 的項目時，下一個for 迴圈要在哪一個index 停止才不會多算或少算。

2. **Python 的型別轉換非常迅速**：

   Python 程式有一個很特別的特性，就是不管變數內含有的資料有多大，在做型別轉換時都非常的迅速。若一個變數a 是串列型別，我們想把a 變成集

合型別，我們只要用a = set(a) 就可以輕鬆的把a 的型別從串列變成集合。例如我們有兩個很大的串列a 和 b，我們想知道這兩個串列交集的部份為何，若是使用for 迴圈來撰寫，我們需要兩個for 迴圈來達成。但Python 具有快速變換型別的特性，因此我們可以使用下列程式來達成。

```
a = set(a)
b = set(b)
c = a.intersection(b)
```

短短三行的程式就可以快速達成目的，c 變數就是a 串列和b 串列的交集元素所成的集合。

3. **辭典集型別適合稀疏矩陣的資料紀錄**

Python 的辭典集型別是以"金鑰:值"的方式紀錄每一個金鑰所對應的值，這種資料型別正好適合紀錄大型稀疏矩陣非零值的內容，例如我們可以定義一個辭典集為

```
A = {(1,3):5, (2,7):-1, (15,80):3, ...}
```

":" 前面的元組是紀錄矩陣元素的位置，後面的值代表該元素的值，這樣就可以用最少的記憶體，儲存大型稀疏矩陣。在網路服務應用上，若我們想紀錄使用者在該網站的所有瀏覽紀錄，使用者的index 為Uid(User index)，網站內容的index 為Tid(Term index)，點擊過的次數為n ，則我們可以用{Uid:(Tid,n)} 的形式來紀錄，沒有點擊過的資訊都不會佔用server 的紀錄空間。

4. **串列內的內容型別不受限制**

在大型網站的推薦系統應用上，如何快速地對某一位使用者進行相似性偏好的即時推薦，此服務最大的瓶頸在於如何快速地搜尋到與該使用者偏好相近的其他使用者，這樣的服務不可能在每一次推薦時都進行一次搜尋的動作，而是先把搜尋好的資料建成表格的形式，即時服務時時只對該表格的內容進行搜尋，在流量低的時候進行網站再對表格進行校正更新的動作。

Python 的串列型別非常適合應用在推薦系統表格的建制，例如我們想紀錄每一個使用者有哪些與他/她偏好相近的使用者，我們使用串列型別來放置

集合變數，第i 個index 所對應的集合內存放與第i 位使用者偏好相近的使用者Uid。例如

```
S = [set([5,8,7]),set([9,3]),set([100,25,49,91]),...]
```

這代表Uid=0 的使用者與他/她偏好相近的使用者Uid 為$5, 8, 7$。若我們除了要紀錄偏好相近的使用者是誰，並且還要紀錄偏好相近的程度，我們也可以在每一個串列中放置辭典集的變數，例如

```
S = [{5:0.8,8:0.6,7:0.9},{9:0.7,3:0.65},{100:0.78,25:0.88,49:0.63,91:0.67},...]
```

這樣就表示Uid=0 的使用者與Uid=5 的使用者偏好相近程度為0.8，與Uid=8的使用者偏好相近程度為0.6。Python 的資料內容不受限制的特性，讓Python 程式在網站分析應用上非常方便。

5. **for 迴圈的操作不侷限於數字**

   先前我們提到我們可以利用辭典集型別來紀錄大型稀疏矩陣或是網站使用者點擊的紀錄，在一個資料長度一直更新的情況，傳統的程式寫法我們必須要先確認該資料的長度，才能用for 迴圈確保每一個元素都被讀取到，這樣計算資料長度都會成為一種負擔。Python 的for 迴圈可以直接呼叫其內容作為有序或無序的取值，例如我們有一個辭典集為

   ```
   d={'a':10,'b':12}
   ```

   我們用以下的方式來列印出辭典集內的資料

   ```
   for x in d.keys():
       print d[x]
   ```

6. **從Python 到Matlab** 因為在Python裡面可以利用串列開出很像Matlab的矩陣，而這樣的串列也可以轉成Matlab的矩陣，好處是可以利用Python來整理資料，整理完以後，可以將整理過的資料交由Matlab來作運算。例如現在有一個串列為

   ```
   L=[[1,2,3] ,[4,5,6]]
   ```

然後，我們引用套件 scipy.io

```
from scipy.io import *
m_dict=dict()
m_dict['L']=L                # 'L' 是要給matlab 看的變數名稱，L 是資料
savemat('L.mat',m_dict)
```

之後在Matlab中顯示的結果就是一個2×3的矩陣

```
L=
     1     2     3
     4     5     6
```

7. **串列與陣列的轉換** 在Python裡面可以使用套件Scipy，讓串列可以轉成像Matlab中的矩陣(陣列)，例如:現在有一個串列

```
A=[[1,2],[3,4],[4,5]]
```

，然後引入套件Scipy中的陣列(array)，令變成陣列的為B

```
from scipy import array
B=array(A)
B=
      array([[1, 2],
       [3, 4],
       [4, 5]])
```

。那要如過要將陣列B變回串列呢?我們會使用以下指令

```
B=[list(y) for y in B]
B=
     [[1, 2], [3, 4], [4, 5]]
```

B就變回串列了。

附錄B：Python 操作Mysql 資料庫

假設我們有一個Mysql database，我們建立一個Mysql 資料庫名為Yahoo，並且在Yahoo 資料庫底下建立一個資料表名稱為Userhist，其內容為三列，第一列記錄使用者Uid（User index），第二列記錄使用者點擊項目的Tid（Term index），第三列為使用者點擊該項目發生的時間（Time）。 Python 程式要能成功讀取Mysql 資料庫，必須先安裝一個MySQLdb 模組，安裝完成之後便可以用下列程式來讀取Mysql 資料表。

```
#引入MySQLdb 模組
import MySQLdb

#連接到MySQL
db = MySQLdb.connect(host="localhost",user="db_user",
                                    passwd="db_pass",db="Yahoo")
cursor = db.cursor()

#執行MySQL 語句
cursor.execute("SELECT * FROM Userhist")
result = cursor.fetchall()

#篩選使用者A 的點擊資料
cursor.execute("SELECT * FROM Userhist where user="A")
result_A = cursor.fetchall()
```

我們用MySQLdb.connect() 指令來建立與Mysql 資料表的連結，其中的參數需要配合實際的使用者名稱和密碼做修改。 這時Python 會建立一個db 物件，我們用db.cursor() 來進行對此物件的操作。 接著我們可以用cursor.execute() 來執行MySQL語法，之後的操作就與直接在MySQL 環境下一模一樣。 當我們用fetchall()指令時，我們把cursor.execute() 篩選後的資料存成Python 的tuple 型別，這樣我們就可以讓Python 接手，進行我們需要的運算。 舉例來說，我們在MySQL 環境下的資料內容為

```
mysql> select * from Userhist;
+------+------+------+
| user | tid  | time |
+------+------+------+
| A    |    1 |  345 |
| B    |    1 |  399 |
| A    |    2 | 1099 |
| B    |    3 |  145 |
| C    |    3 |  169 |
| D    |    3 |  125 |
| D    |    4 | 1995 |
+------+------+------+
```

在Python 環境中我們列出result 變數的結果為

```
(('A', 1L, 345L), ('B', 1L, 399L), ('A', 2L, 1099L), ('B', 3L, 145L),
('C', 3L, 169L), ('D', 3L, 125L), ('D', 4L, 1995L))
```

若我們指對某位使用者進行資料篩選，在MySQL 環境下為

```
mysql> select * from Userhist where user='A';
+------+------+------+
| user | tid  | time |
+------+------+------+
| A    |    1 |  345 |
| A    |    2 | 1099 |
+------+------+------+
```

在Python 環境下我們列出result_A 的結果如下：

```
(('A', 1L, 345L), ('A', 2L, 1099L))
```