# 第三章

# 畫風探勘

畫風探勘的過程中，我們先萃取出畫作的低階影像特徵，再利用 data mining 的方法，找出每個畫家其所有畫作裡共同的特徵。再利用 data mining 中的 classification 演算法來進行畫家間特徵的比較，並把比較的結果，加以定量的描述，就是每個畫家獨特的風格。

本論文中，我們提出兩個 classification 的演算法來建立畫風規則。在這一章中，3.1 節描述進行畫風探勘時所需的影像特徵，3.2 節則會介紹如何探勘影像特徵和畫家類別之間的關聯，3.3 節則提出兩個修改自 associative classification 的演算法，以提高畫風探勘的效果。

## 3.1 影像特徵擷取(Image Feature Extraction)

在本論文中，我們萃取出顏色（color）、相鄰顏色（color relationship）與 MPEG-7 color descriptor 等特徵。然而在萃取顏色與相鄰顏色之前，我們要先進行顏色空間（color space）轉換與降色（quantization），減少使用顏色的數目。

顏色空間轉換：為了程式處理方便，我們將繪畫影像的影像格式均轉換成 BMP 型式，其顏色空間是 RGB。然而 RGB color space 在降色的過程中，在降色區間內的所有顏色與所產生的區間代表色間，以人類的視覺感受上不是那麼相似。所以我們選用 HSV color space 來解決這個問題，並將繪畫影像的 color space 從 RGB 轉換成 HSV。表示 HSV 的顏色方面，我們以(h, s, v)的方式來表示。其中 h 代表色度，$0 \leqq h < 360$。而 s 代表彩度，v 代表亮度，$0 \leqq s,v \leqq 100$。

降色：我們對 H, S, V 分別進行 uniform quantization，並取其區間的第一個值當代表。因為人類對色度較為敏感，因此在進行降色時，可以針對色度選擇不同的降色程度。以降色成 256 色為例，H 取 16 種，S 與 V 取 4 種。如此 S 就有[0, 24]，[25, 49]，[50, 74]，

[75, 100]四個區間，代表值就是 0, 25, 50, 75。

經由以上的步驟後，我們可以將全彩影像降色，再進行影像特徵萃取。

顏色：我們萃取主要顏色（dominant color）做為 color feature。一般的做法是計算繪畫影像降色過的各代表色的面積，即 color histogram，再以面積最大的數個顏色做為主要顏色特徵。但是有些畫家的畫風是用色的喜好集中，而另外有些畫家用色分散，這時固定選用前數名面積最大的顏色當主要顏色特徵則不能表現畫家用色的習慣。所以，我們計算繪畫影像降色過的各代表色的面積後，去掉出現面積太少的顏色，做為主要顏色特徵，目前門檻值為畫面總面積的 1%。

相鄰顏色（nearest color relationship）：根據色彩調和的理論，顏色和顏色的搭配會營造出各式不同的感覺[43, 44, 45]。例如，淡黃色與紫色的組合會帶給人類不安焦慮的感覺。為了找出顏色相鄰的關係，我們利用以顏色、材質為基準的 image segmentation 技術 JSEG[12]。JSEG 是一個 unsupervised segmentation，整個 segmentation 的過程可以分成兩大主要步驟：color quantization 與 spatial segmentation。

color quantization 是為了找出幾個足以代表、區分相鄰區域的顏色，對於自然影像而言，大概只需要 10 到 20 個顏色。找出代表顏色後，給予分類標誌，並且將影像中所有的顏色換成相對應的分類標誌，形成所謂的 class-map image。而這個 class-map image 可以視為一種特殊的材質佈局。

第二步驟的 spatial segmentation 是採取 region growing 的方式，而在開始之前，要先找出 region seed。這些 region seed 是 class-map image 中，材質最一致的地方。在找出 region seed 之後，再慢慢合併周遭四連通且材質一致的區域。最後再合併相連且顏色相近的區域。

圖 3.1 是根據 JSEG 找出繪畫影像中所有顏色相近的區域的示意圖。之後再以各個 region 的面積最大的顏色當代表色，找出各個 region 間代表色的相鄰狀況。現在我們只考量兩兩相接的情況，並以＜$(h_1, s_1, v_1)$・$(h_2, s_2, v_2)$＞的方式表示相鄰顏色。而此表示的方式跟顏色順序無關，例如＜黃・紅＞與＜紅・黃＞代表同樣的相鄰顏色。

例如以圖 3.1 的例子，image segmentation 後有四個 region，找出來的相鄰顏色有五個，分別是＜黃・紅＞、＜黃・藍＞、＜紅・藍＞、＜紅・綠＞、及＜藍・綠＞。

以楊三郎於 1988 年的作品，「澎湖鯨魚洞」為例，在圖 3.2 中，(a)圖是從網路上下載的原始畫作，而如 (b)圖所呈現的是降色成 256 色的畫面（H:16, S:4, V:4），在(c)圖中出現的，就是面積超過畫面 1%的顏色，最後(d)圖則是以做完 image segmentation 後，所

有顏色相近的 region。

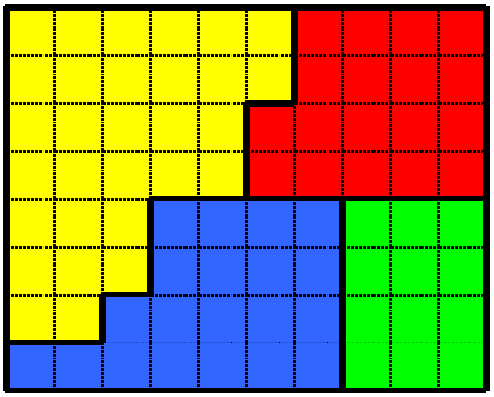


圖 3.1 Image segmentation 後，所有顏色相近的區域

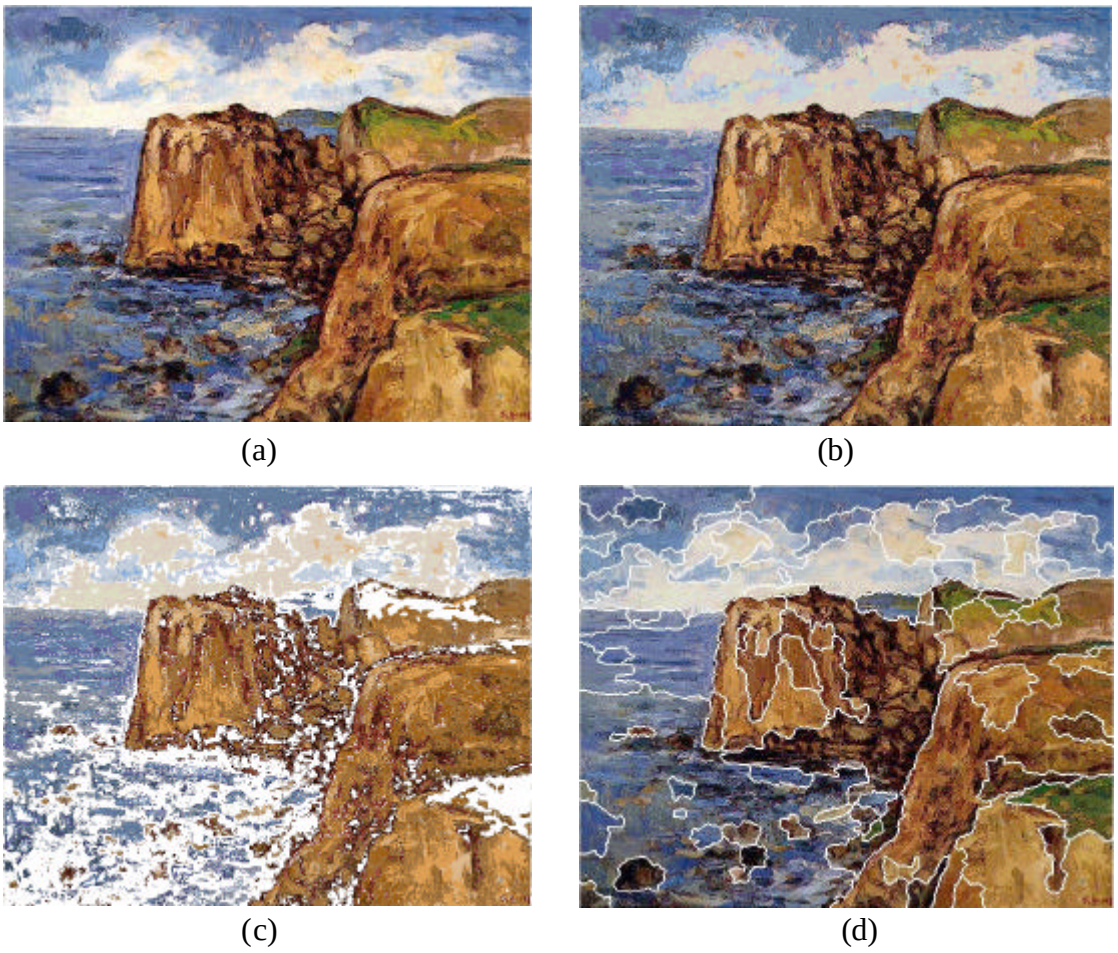
Dominant color descriptor：MPEG-7 XM[28]中萃取 dominant color descriptor 的方式是利用 Generalized Lloyd Algorithm[17]，對影像的所有 pixel 上的顏色進行叢集分析。當開始萃取 dominant color descriptor 的時候，全部的顏色屬於一個叢集。而對於叢集 $C_i$，存在一個 distortion $D_i$：

$$D_i = \sum_n h(n)\left\| x(n) - c_i \right\|^2, \qquad x(n) \in C_i$$

其中，$x(n)$是叢集 $C_i$ 中第 $n$ 個顏色，$c_i$ 則是叢集 $C_i$ 的代表顏色（重心的顏色），而 $h(n)$則是依據 local pixel statistics 後，對叢集 $C_i$ 中第 $n$ 個顏色的加權，加權的目的在於人類對於一塊顏色平滑的區域中突出、不規則的顏色比起相同材質、規律變化的顏色較為敏感[22]。演算法會找出 distortion 值最高的叢集並加以分割，直到最高的 distortion 值小於 threshold 或是叢集數達到最高值才停止。

當叢集演算法停止後，再計算每個叢集所佔畫面面積、四連通（four connectivity）個數，以及各個顏色區間的變異數，再根據使用者指定的方式轉換 color space model 及降色，最後 output 使用者指定的前 n 個 dominant colors。在論文中，我們使用 8 個 dominant color。

Scalable color descriptor：如第二章所述，scalable color descriptor 就是記錄 color histogram。在 MPEG-7 XM 中萃取 scalable color 的方式是先將 image 的 color space 轉換成 HSV，並且依 H:16，S:4，V:4 的方式降成 256 色，並計算其 color histogram，再經過 Haar transform 後，output 使用者指定的前 n 個 harr 係數。在論文中，我們使用 128 色，未經 Haar transform 過的 color histogram。



(a) (b)

(c) (d)

**圖** 3.2 (a)**楊三郎原畫，澎湖鯨魚洞** (b)**降色後的影像**
(c)**去掉出現面積少於畫面面積** 1%**的顏色** (d)**所有顏色相近的區域**

Color structure descriptor：Color structure descriptor **以統計的方式，提供顏色在空間上的資訊。在** MPEG-7 XM **中萃取** color structure **的方式是先將** image **的** color space **轉換成** HMMD**，並且降色成** 256 **色。**Color structure descriptor **利用** 8x8 **的** structure element **來統計。而在** structure element **中只記錄顏色有無出現。等到最後** structure element **統計過全部的畫面後，再統計所有** 256 **個顏色在幾個** structure element **中出現過，這就是** color structure histogram**。最後再依使用者要求，**output 256 **色** color structure histogram **或** uniform quantization **成** 32、64、128 **色** color structure histogram **再** output**。在論文中，我們使用** 128 **色的** color structure histogram**。**

Color Layout descriptor：Color Layout descriptor **記錄了空間佈局的資訊。在** MPEG-7 XM **中萃取** color structure **的方式可以分成四個步驟。首先將** image **的** color space **轉換成** YCbCr**，然後為了** color layout descriptor **能應用在不同的** resolution **與** scale **上，於是把**

image 切成 8x8 共 64 個 block。第二個步驟針對每一個 block 找出一個代表色，雖然用任何方式找代表色都可以，不過 XM 中建議使用平均的方式，得到 8x8 個 YCbCr 代表色。接著將 8x8 個 YCbCr 代表色分成 Y、Cb、Cr 三個 plane。對每個 plane 再分別進行 2D DCT 轉換，得到三組 DCT 係數，每組都包括 8x8 個 DCT 係數。最後針對每組 DCT 係數，利用 zigzag scan 將二維的 DCT 係數轉成一維。最後再輸出使用者要求的前 n 個 DCT 係數。在本論文中，我們使用全部 64 個 DCT 係數。

## 3.2 頻繁樣式探勘(Frequent Pattern Mining)

在接下來的二小節，我們將分別介紹我們所萃取出的主要顏色，相鄰顏色以及 MPEG-7 中的 dominant color descriptor，scalable color descriptor，color structure descriptor，color layout descriptor 的資料探勘方式。

### 3.2.1 主要顏色與相鄰顏色特徵之探勘

顏色：由於畫家的畫風可能來自於顏色的搭配，如白色與藍色一起用。因此在探勘繪畫影像風格的過程中，我們應用 association rule mining 中找 frequent itemset 的技術。應用在本論文上時，我們將一幅繪畫影像視為一筆 transaction，而這幅繪畫影像所萃取出來的低階影像特徵，以 dominant color 為例，就是這個 transaction 的 itemset，而一個顏色，就是一個 item。

例如，假設有一畫家的繪畫影像四幅，每一幅萃取主要顏色為低階影像特徵，如下表 3.1 所示。當 minimum support(min_sup)設為 50%時， mining 出來的 frequent itemset 為{白，紅}與{紫，藍}，其 support 皆為 2。這表示在這位畫家 50%的畫中同時出現白與紅兩種顏色，此外紫與藍兩種顏色也出現在 50%的畫中。

表 3.1 Database D 主要顏色 1

| 畫作 ID | 主要顏色 |
|---|---|
| 100 | 紅 白 綠 橙 |
| 200 | 藍 白 紫 |
| 300 | 紅 黃 白 |
| 400 | 紅 藍 紫 靛 |

雖然 association rule 能找出畫家常搭配使用的顏色，然而畫家的風格可能不只有顏色的關係，例如畫家喜歡以高亮度的顏色做畫。此時就要個別考慮色度或彩度或亮度間

的關係。例如有一畫家的四幅畫及其 2 個主要顏色，如下表 3.2 所示：

表 3.2 Database D 主要顏色 2

| 畫作 ID | 主要顏色 |
|---|---|
| 100 | (225, 50, 25), (135, 75, 25) |
| 200 | (90, 50, 25), (315, 75, 25) |
| 300 | (225, 50, 25), (135, 75, 25) |
| 400 | (0, 25, 25), (270, 75, 50) |

設 min_sup 為 50%，我們可以發現 frequent itemset 有{(225, 50, 25), (135, 75, 25)}。可是當 min_sup 提高到 75%時，association rule 便找不出 frequent itemset。然而我們發現，如果不考量色度的狀況下，畫家常將彩度 50 且亮度 25 與彩度 75 且亮度 25 的顏色一起搭配使用，且其 support 值有 75%；而在只考慮亮度，不考慮色度與彩度時，畫家常使用兩個亮度 25 的顏色，其 support 值有 100%。

由於傳統的 itemset 不能直接產生這種 pattern，於是 mining 時要利用 multi-level association rule 的概念表示顏色間的類別關係，如圖 3.3 所示。我們的作法是將每個顏色轉化成一個對應集合。針對每一個 color，產生所有的組合，包括 H，S，V 以及 HSV 兩兩間的組合。以第一幅畫為例，(225, 50, 25)這顏色其對應的集合包括下列元素：(225, *, *)，(*,50, *)，(*, *, 25)，(225, 50, *)，(225, *, 25)，(*, 50, 25)。轉化之後，設 min_sup 為 75%時所找出的 maximum frequent pattern如下：(*, *, 25), support 為 4，以及(*, 25, 25) (*, 60, 25)，support 為 3。

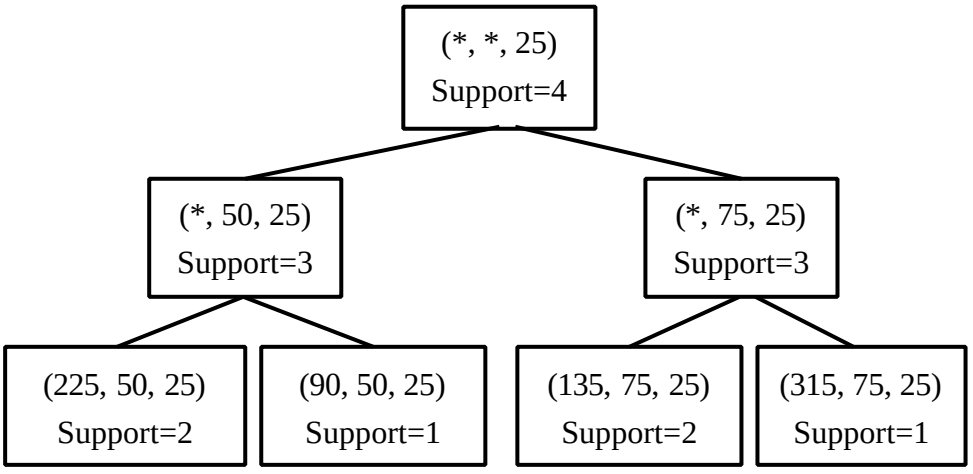



圖 3.3 Multi-level association rule 示意圖

相鄰顏色：相鄰顏色的 mining 過程也和主要顏色相似，此時每一個 item 就是一組相鄰顏色，例如＜紅・藍＞。而找出來的 frequent itemset 代表畫家常使用的相鄰顏色組

合，例如{＜紅・藍＞, ＜靛・黃＞}。

相鄰顏色的 mining 過程也可以套用 multi-level association rule。在轉化的過程中，每一組相鄰顏色也會對 H，S，V 以及 HSV 兩兩間的產生所有的組合，而且一組相鄰顏色中的兩個顏色，只產生同樣維度的組合。例如以＜(0, 25, 75)・(135, 50, 25)＞這組相鄰顏色為例，其對應的集合有：＜(0, *, *)・(135, *, )＞、＜(*, 25, *)・(*, 50, *)＞、＜(*, *, 75)・(*, *, 25)＞、＜(0, 25, *)・(135, 50, *)＞、＜(0, *, 75)・(135, *, 25)＞、＜(*, 25, 75)・(*, 50, 25)＞。

### 3.2.2 MPEG-7 Descriptors 之探勘

在本小節，我們將敘述 MPEG-7 descriptors 特徵值的資料探勘方式。

Dominant color descriptor：在這個 descriptor 中描述的是有關前 8 個主要顏色的統計資料，例如各個主要顏色佔的面積、顏色區間的變異數、所有主要顏色的平均空間分佈。因此我們利用 C4.5 進行分類，並把 C4.5 所產生的 decision tree 的 rules 整合進我們的分類器。例如：主要顏色的 spatial coherency 值≦10 是 A 畫家的畫；主要顏色的 spatial coherency 值>10 且第一主要顏色所佔的面積 < 50% 是 B 畫家的畫。

由於 scalable color descriptor 與 color structure descriptor 同樣是屬於 color histogram 的一種，屬於統計資料。所以我們也利用 C4.5 分別建出 decision tree。例如：第 10 個顏色的 scalable 係數在 100 到 200 之間的是 C 畫家的畫。

MPEG-7 的 color layout descriptor 描述的是一張影像的畫面佈局，因此藉由 color layout descriptor mining，我們可以找出畫家常用的畫面佈局。然而 color layout descriptor 中記錄的是 zigzag scan 過的 DCT 參數，不能直接應用現有的 data mining 技術。因此我們把 color layout descriptor 還原成原來的 8x8 個顏色，並利用 2D String 表示，將 color layout descriptor mining 的問題轉化為從 2D String database 中找出頻繁的 2D subsequence 的問題。2D String 的詳細定義如下：

【定義 3.1】令 S 是物件代號（object symbol）的集合。R 是三個符號{'=', '<', ':'}的集合，表示物件與物件的空間關係。符號'<'代表物件間是從左到右或是從下到上的空間關係；符號'='代表二個物件位於同一個位置的空間關係；符號':'代表是屬於同一集合的關係。

1D String over S 是指任意字串 s 具有下列型式： $u_1u_2u_3\Lambda\ u_n$ , $n \geq 0$，其中 $u_i$ 屬於 S，

而 1D String 的長度，記為 length(s)，其值為 n。

2D String over S∪R，寫成（U，V）定義成：$(u_1 r_1^x u_2 \Lambda\ r_{n-1}^x u_n, u_{p(1)} r_1^y u_{p(2)} \Lambda\ r_{n-1}^y u_{p(n)})$，

其中 $u_1 u_2 \Lambda\ u_n$ 和 $u_{p(1)} u_{p(2)} \Lambda\ u_{p(n)}$ 是 1D String over S

p: {1, …, n} → {1, …, n} 是 permutation function over {1, …, n}

$r_1^x r_2^x \Lambda\ r_{n-1}^x$ 和 $r_1^y r_2^y \Lambda\ r_{n-1}^y$ 是 1D String over R

例如圖 3.4 中物件的空間關係可以表示成 2D String $(u_1 r_1^x u_2 r_2^x u_3 r_3^x u_4, u_{p(1)} r_1^y u_{p(2)} r_2^y u_{p(3)} r_3^y u_{p(4)}) = (A < B = D < C, B < A < D < C)$，其中 $u_1 u_2 u_3 u_4$ 是 ABDC，$u_2 u_1 u_3 u_4$ 是 BADC，p(1)=2，p(2)=1，p(3)=3，p(4)=4，$r_1^x r_2^x r_3^x$ 是<=<，$r_1^y r_2^y r_3^y$ 是 <<<。

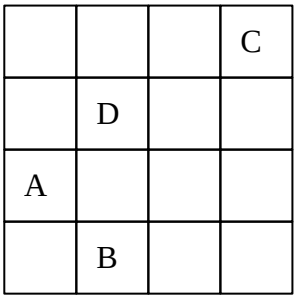



圖 3.4 物件空間關係圖

例如假設 database 中有四個 2D String 如表 3.3 所示，其畫面如圖 3.5 所示。當 minimum support(min_sup)設為 75%時， mining 出來的 frequent 2D subsequence 為{紅<紫<橙,紫<橙<紅}，其 support 為 3。這表示四個 2D String 所代表的畫面中，有 75%的畫面具有圖 3.6 的畫面佈局。

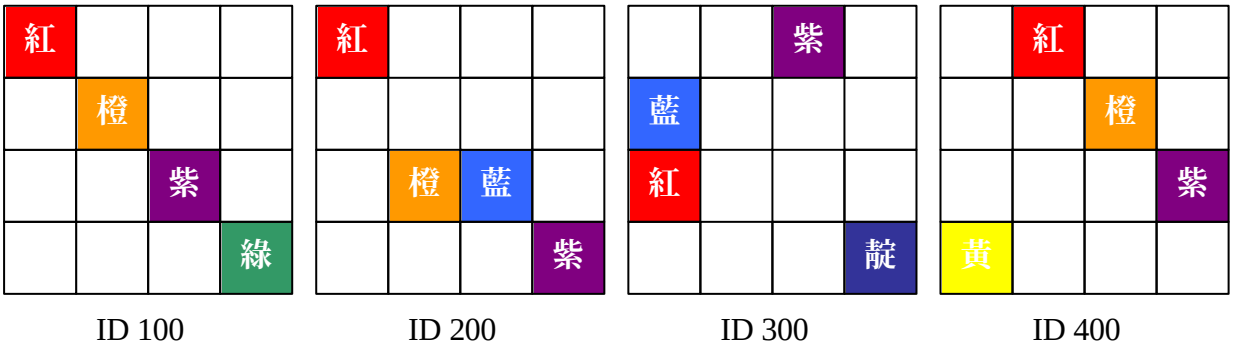



圖 3.5 畫面佈局示意圖

表 3.3 Database $DB_{string}$

| 畫作 ID | 2D Strings |
|---|---|
| 100 | (紅<橙<紫<綠，綠<紫<橙<紅) |
| 200 | (紅<藍<橙<紫，紫<藍=橙<紅) |
| 300 | (紅=藍<紫<靛，靛<紅<藍<紫) |
| 400 | (黃<紅<橙<紫，黃<紫<橙<紅) |

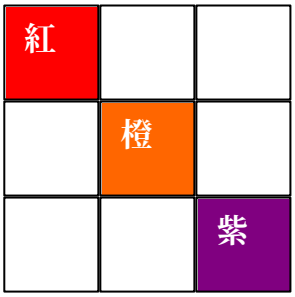



圖 3.6 2D String {紅<橙<紫, 紫<橙<紅} 示意圖

然而 2-D string 是二維的 pattern，現有的 sequential pattern mining 都是針一維的 pattern，因此，我們針對 2-D string 提出 2-D subsequence pattern mining。

【定義 3.2】字串 U'是字串 U 的 1D subsequence，如果

(1) U'包含在 U 之中，

且 (2)令 $a_1w_1b_1$ 是 U'的 substring，$a_2b_2$ 是 U 中的符號，且 $a_1=a_2$，$b_1=b_2$，則 $r(b_2)-r(a_2) \geqq r(b_1)-r(a_1)>0$ 或 $r(b_2)-r(a_2)=r(b_1)-r(a_1)=0$。

其中 r(x)代表著有多少次的'<'出現在 x 之前再加 1。

【定義 3.3】令(U, V)與(U', V')分別是代表圖 I 與 I'的 2D String。(U', V')是(U, V)的 2D subsequence，如果 U'是 U 的 1D subsequence 且 V'是 V 的 1D subsequence。我們以 $(U', V') \subset (U, V)$ 代表(U', V')是(U, V)的 2D subsequence。反之，則稱(U, V)是(U', V')的 2D supersequence。

【定義 3.3】在一個包括多個 2D String database $DB_{2D\ String}$ 中，2D String (U', V')的 support 定義為 $DB_{2D\ String}$ 中，屬於(U', V')的 2D supersequence 的個數。

以表 3.3 為例，2D String {紅<橙<紫, 紫<橙<紅} 的 support 值為 3。

【定義 3.4】給定一個 2D String (U, V)，若(U, V)的 support 大於給定的 minimum support，且 U 的長度 length(U) = $k$，則我們稱 2D String (U, V)為 frequent 2D sequence，亦稱為 frequent 2D $k$-sequence。

**欲從** 2D String database **中探勘出** frequent 2D sequence**，我們修改** Apriori-based sequential pattern mining **的演算法，如下圖** 3.7 **所示。**

**Algorithm *Mining Frequent 2D Sequence***
Input: 2D string database *DB*, min. support *min_sup*
Output: *Frequent 2D sequential Pattern*
1. All *1*-string are candidate *1*-sequences
2. Frequent *1*-sequences = candidate *1*-sequences whose support ≥ *min_sup*
3. Generate candidate 2D *2*-sequences from frequent *1*-sequences
4. k = *2*
5. While the set of candidate 2D *k*-sequences is not empty do
6. For each 2D sequence *ds* in *DB* do
7. For each candidate *c* do
8. Increase the support of *c* if *c* is contained in *ds*
9. Endfor
10. Endfor
11. Frequent 2D *k*-sequences = candidate 2D *k*-sequences whose support ≥ *min_sup*
12. Generate candidate 2D (*k+1*)-sequences from frequent 2D *k*-sequences
13. k = k+1
14. Endwhile

**圖** 3.7 Mining Frequent 2D Sequence **演算法**

**Algorithm *Candidate Generation of 2D n-Sequence***
Input: large 2D *n*-sequence $(U_x, U_y)$, $(V_x, V_y)$
Output: candidate set of 2D *n+1*-sequence
1. for *i*=1 to n-1 do
2. if $U_{xi} \neq V_{xi}$ OR $r(U_{yi})–r(U_{yi-1}) \neq r(V_{yi})–r(V_{yi-1})$
3. return $\phi$
4. endfor
5. let $candidate\_set_x$ = candidate_generate_1D_sequence($U_x$, $V_x$)
6. let $candidate\_set_y$ = candidate_generate_1D_sequence($U_y$, $V_y$)
7. for each $candidate_x$ in $candidate\_set_x$ do
8. for each $candidate_y$ in $candidate\_set_y$ do
9. put ($candidate_x$, $candidate_y$) into *candidate_set*
10. endfor
11. endfor
12. return *candidate_set*

**圖** 3.8 candidate generation of 2D sequence **演算法**

**Algorithm** ***Candidate Generation of 1D n-Sequence***

Input: large 1D *n*-sequence U, V

Output: candidate set of 1D *n+1*-sequence

1. let $U=(u_1r_{u1}u_2\ldots u_{n-1}r_{un-1}u_n)$, $V=(v_1r_{v1}v_2\ldots v_{n-1}r_{vn-1}v_n)$
2. Exists *r, s* let $U'=(u_1r_{u1}u_2\ldots u_{r-1}r_{ur-1}u_{r+1}r_{ur+1}\ldots u_{n-1}r_{un-1}u_n)$ and $V'=(v_1r_{v1}v_2\ldots v_{r-1}r_{vn-1}v_{r+1}r_{vr+1}\ldots v_{n-1}r_{vn-1}v_n)$
3. if U'=V' then
4. if *r*=*s* then
5. put $(u_1r_{u1}u_2\ldots u_{r-1}r_{ur-1}\mathbf{u_r}<\mathbf{v_r}r_{ur}u_{r+1}r_{ur+1}\ldots u_{n-1}r_{un-1}u_n)$ into *candidate_set*
6. put $(u_1r_{u1}u_2\ldots u_{r-1}r_{ur-1}\mathbf{v_r}<\mathbf{u_r}r_{ur}u_{r+1}r_{ur+1}\ldots u_{n-1}r_{un-1}u_n)$ into *candidate_set*
7. put $(u_1r_{u1}u_2\ldots u_{r-1}r_{ur-1}\mathbf{u_r}=\mathbf{v_r}r_{ur}u_{r+1}r_{ur+1}\ldots u_{n-1}r_{un-1}u_n)$ into *candidate_set*
8. else
9. put $(u_1r_{u1}u_2\ldots u_{r-1}r_{ur-1}\mathbf{u_r}r_{ur}u_{r+1}r_{ur+1}\ldots u_s r_{us}\mathbf{v_s}r_{vs}u_{s+1}r_{us+1}\ldots u_{n-1}r_{un-1}u_n)$ into *candidate_set*
10. endif
11. endif
12. return *candidate_set*

**圖** 3.9 candidate generation of 1D sequence **演算法**

**產生** 2D n-squence candidate **的演算法，如圖** 3.8 **所示，而其中所使用的** candidate_generate_1D_sequence **演算法，則列在圖** 3.9 **中。**

**例如現在我們有** 2D String **的** database $DB_{string}$**，訂** minimum support **為** 3**，要找出** frequent 2D sequences **的步驟如下：**

1. **先** scan **整個** database $DB_{string}$**，找出所有** 1D 1-sequences **及其** support**。如圖** 3.10 **中的** $C_1$**。去掉** support **低於** minimum support **的** sequences **後，得到所有的** frequent 1-sequences **的集合** $L_1$**。**

2. **把** $L_1$ **中的** frequent sequences **組合成** candidate 2D 2-sequences $C_2$**。由於** $L_1$ **中並未包含順序關係，因此** item **間的順序關係必須在產生** candidate sequences $C_2$ **時加以考慮。以** $L_1$ **中的**{**紅**}、{**橙**}**為例，會產生下列九個** candidate sequences：{**紅**<**橙**, **紅**<**橙**}、{**紅**<**橙**, **橙**<**紅**}、{**紅**<**橙**, **紅**=**橙**}、{**橙**<**紅**, **紅**<**橙**}、{**橙**<**紅**, **橙**<**紅**}、{**橙**<**紅**, **紅**=**橙**}、{**紅**=**橙**, **紅**<**橙**}、{**橙**=**紅**, **橙**<**紅**}、{**紅**=**橙**, **紅**=**橙**}。
3. Scan **整個** database D**，找出** $C_2$ **所有的** support**。去掉** support **低於** minimum support **的** sequences **後，得到所有** frequent 2D 2-sequences **的集合** $L_2$**。**
4. **同樣的，把** $L_2$ **中的** sequences **組合成** candidate 2D 3-sequences $C_3$**。再** scan **整個** database D**，找出** $C_3$ **所有的** support**。去掉** support **低於** minimum support **的** sequences **後，得到所有** frequent 2D 3- sequences **的集合** $L_3$**。**

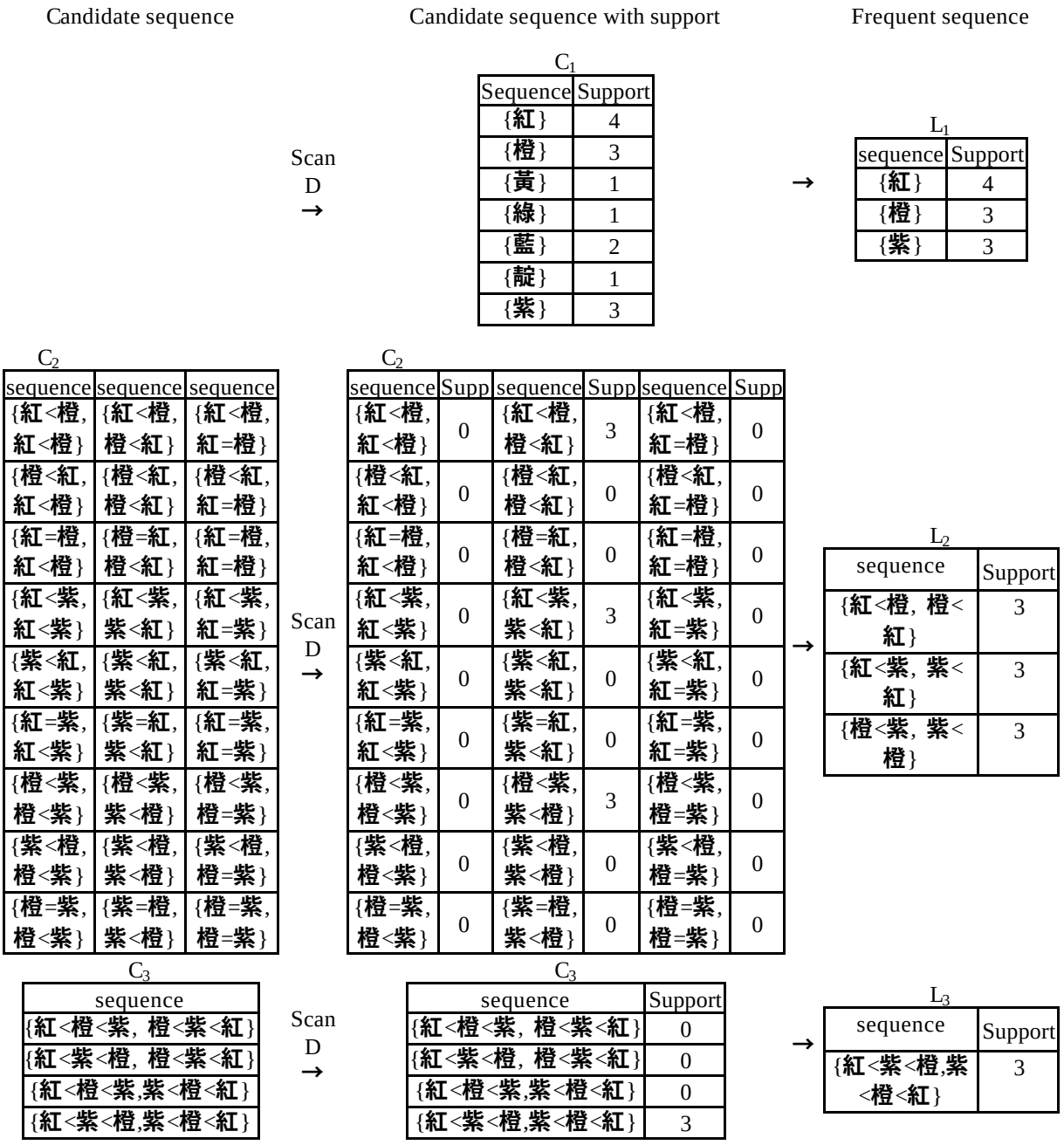



**圖** 3.10 Generation of candidate sequence and frequent sequence

5. **由於** $L_3$ **只有一組** frequent sequence**，無法組出** $C_4$**，因此找** frequent sequence **的步驟就結束了。**

## 3.3 建構分類器(Classification)

**在本節，我們修改** associative classification **並提出兩個改進** associative classification **的演**

算法，最後利用 bagging classification 提昇我們 classifier 的準確度。

### 3.3.1 Associative Classification

我們修改 associative classification 以結合 association rule 與 classification 的分類方法，分類的依據是利用 training set 中各類別間所探勘出來的 frequent pattern。由於利用 data mining 技術所探勘出來的 frequent pattern，可以代表該分類資料的共同特性。例如，影像資料庫中有 X，Y 兩類畫，如果 X 類畫中白與紅兩種顏色常常一起出現，而 Y 類畫中藍與黃兩種顏色常常一起出現，則可以表示成下列兩條分類規則：{白，紅}→X，{藍，黃}→Y。若不知類別的畫作中是白與紅兩種顏色同時出現，那就會被分到 X 類。

Associative classifier 的組成是由分類規則（classifier rule）與 default class 所組成。要建出 classifier 時，分為兩個步驟，第一個步驟是利用 association rule mining 的方法找出 training set 中各類別的 frequent itemsets，並分別記錄該 itemset 在其類別的 support。例如，現在影像資料庫中有 X，Y 兩類畫，每一類畫各 5 幅，其萃取出來的主要顏色如表 3.4。在指定 min_sup 為 50%的情況下，針對 X，Y 兩類各進行 mining association rule 後，得到的 frequent itemsets 如表 3.5。

接著我們用下列的方式，從兩類別的 frequent itemsets 中得到相對應的分類規則（classifier rule）。令 itemset L 是 X 類的 frequent itemset，那麼分類規則的 candidate rule 的型式以{L}→X 表示，代表只要有 L 子集的 transaction 就會分類成 X 類。如果 frequent itemset L 在 X，Y 兩類別中都出現，而且 $support_X < support_Y$，那麼 L 會代表 Y 類，其 rule 則為{L}→Y。如果兩邊 support 相同，則任選一個類別。

令{L}→X 為分類規則的 candidate rule。其 support 值則是 itemset L 在 database X 中的 support 值，而 confidence 則依下列公式計算：

$$confidence = \frac{\mathrm{MAX}\{support_X, support_Y\}}{support_X + support_Y}$$

表 3.4 X 與 Y 的畫作 database

| Class X | | Class Y | |
|---|---|---|---|
| 畫作 ID | 主要顏色 | 畫作 ID | 主要顏色 |
| 101 | 紅 綠 紫 | 201 | 紅 綠 黃 |
| 102 | 紅 綠 黃 | 202 | 紫 藍 白 |
| 103 | 綠 藍 紫 | 203 | 綠 黃 白 |
| 104 | 紅 綠 白 | 204 | 綠 紫 黃 |
| 105 | 藍 紫 白 | 205 | 紅 紫 白 |

表 3.5 X 與 Y 在 min_sup 為 50%的 frequent itemsets

| Class X | | Class Y | |
|---|---|---|---|
| Itemset | $support_X$ | Itemset | $support_Y$ |
| 紅 | 3 | 綠 | 3 |
| 綠 | 4 | 紫 | 3 |
| 紫 | 3 | 黃 | 3 |
| 紅綠 | 3 | 白 | 3 |
| | | 綠黃 | 3 |

最後，將 candidate 分類規則的 rule 依 confidence 及 support 的順序排序，結果如表 3.6。

然後進行第二個步驟，Training-Classifier，以建出 classifier，Training-Classifier 的演算法如圖 3.11 所示。首先產生一個空的 classifier，針對 training set 進行分類。接著依排序完的 candidate 分類規則順序進行測試。這裡的測試是指這條 candidate 分類規則能不能對目前剩下的 training set 進行正確的分類。如果正確的分類，才將預選 rule 依次加進 classifier，詳細過程如圖 3.12。

表 3.6 排序完的 candidate 分類規則

| Candidate rules | Confidence | Support |
|---|---|---|
| {綠，黃}→Y | 100% | 3 |
| {黃}→Y | 100% | 3 |
| {紅，綠}→X | 100% | 3 |
| {白}→Y | 100% | 3 |
| {紅}→X | 100% | 3 |
| {綠}→X | 50% | 3 |
| {紫}→X | 50% | 3 |

除了記錄 classifier rule 外，classifier 隨時記錄一個 default class，代表如果不符合 classifier 中的所有 rule 時，要分類成那一個類別。Default class 的取得是 training set 中剩下未被分類的畫中數量最多幅的類別。例如第一條 candidate 分類規則：{綠，黃}→Y 被加進 classifier 後，分類了畫作 ID 為 102、201、203、204 的四幅畫作，剩下的 training set 中，有四幅 X 類，二幅 Y 類，因此這時 default class 為 X。

每加進一條 rule 便對原先的 training set 進行分類並記錄準確率。所謂的準確率是 training set 中畫家的畫被分到正確的那一類的比例。例如{綠，黃}→Y 這條 rule 被加進 classifier 時，將畫作 ID 為 201、203、204 的三幅畫作正確的分類，而 default class 又把畫作 ID 為 101、103、104、105 的四幅畫作正確的分類，總共正確的分類佔整個 training set 的 70%，因此{綠，黃}→Y 這條 rule 的正確率為 70%。

而已被分類的 training set 會被刪除，再測試下一條 candidate 分類規則。例如第二條 rule，{黃}→Y 被測試之前，畫作 ID 為 102、201、203、204 就從 training set 中刪除。最後直到 training set 為空時，才停止建立 classifier。

建立 classifier 之後，再進行刪減 classify rule 的動作，只留下從第一條 rule 到最先出現正確率最大值的地方，以圖 3.12 為例，最先出現正確率最大值的是 rule：{白}→Y，之後的 rule 都被刪除。最後 associative classifier 為：

- {綠，黃}→Y
- {紅，綠}→X
- {白}→Y
- default class: X

**Algorithm *Training-Classifier***

Input: painting database *DB, frequent rules of all classes*

Output: *Classifier*

1. sort rules by confidence and support
2. for each rule *r* do
3. for each painting object *p* in *DB* do
4. if *r* satisfies *p* then mark *p* to be classified
5. if ∃*p* be classified correctly by *r* then
6. remove all marked *p* from *DB*
7. insert *r* into the end of *Classifier*
8. choose majority category of painting in *DB* as *default_class(r)*
9. count total error of *Classifier*
10. remove rules after the first rule *r'* with the lowest total error in *Classifier*
11. insert the *default_class(r')* to the end of *Classifier*
12. return *Classifier*

圖 3.11 Training-Classifier 演算法

### 3.3.2 Variant Support 分類器(SFVS)

由於每個畫家作畫習慣不同，因此 min_sup 會影響低階影像特徵的探勘結果。例如，一位畫家用色風格較為集中，那在 min_sup 的值較高時，仍然可以找出 frequent pattern，

可是如果有一位畫家用色風格較為多樣，那麼找出 frequent pattern 就會較少。這時候建出來的 classification 就會偏向用色較為集中的畫家，造成用色多樣的畫家準確率降低。然而傳統的 associative classification 中所有類別的 min_sup 是一樣的。

為此，本論文提出第一個改進的演算法，single-feature variant support (SFVS) classification，容許各個 class 進行不同 minimum support 的 mining。

在 training 的過程中，SFVS 會自動嘗試所有可能的 min_sup 組合，並選擇分類效果最佳的 min_sup 配對。而為了避免分類器的效果偏向特定一群 data，造成不公平，我們採取 five fold cross validation，計算分類效果。分類前將資料分成五群。每一次分類取四群當 training set，另一群當 validation set，一共進行五次建立 classifier 的動作，再將分類準確度加以平均。因此每試一組 min_sup 的配對，就會進行五次分類，並得到一組平均過的準確度。最後在試過所有可能的 min_sup 組合後，再依準確度選擇分類效果最佳的 min_sup 配對，及對應的分類器。SFVS classification 的演算法如圖 3.13。

例如，現在影像資料庫中有 X，Y 兩類畫，每一類畫各 10 幅，與其萃取出來的主要顏色，再分成 5 群，如表 3.7。在指定 min_sup 為 X 類 30%，Y 類 60%的 min_sup 配對情況下，使用 five fold cross validation。於是，在 min_sup 配對為 X 類 30%，Y 類 50%的情形下，SFVS classifier 的平均分類準確率為 65%（$\frac{75\%+50\%+100\%+75\%+25\%}{5}=65\%$）。由於第 3 fold 所建出來的 classifier 準確率為 100%，在 5 個 fold 裡準確率最高，因此在 min_sup 配對為 X 類 30%，Y 類 50%的情形下，我們選由第 3 fold 的 classifier。

表 3.7 X 與 Y 的畫作 database

| Fold No. | Class X | | Class Y | |
|---|---|---|---|---|
| | 畫作 ID | 主要顏色 | 畫作 ID | 主要顏色 |
| 1 | 101 | 紅 綠 紫 | 201 | 紅 綠 黃 |
| | 106 | 紅 綠 白 | 206 | 綠 紫 黃 |
| 2 | 102 | 紅 綠 黃 | 202 | 綠 藍 白 |
| | 107 | 紅 藍 白 | 207 | 紅 綠 黃 |
| 3 | 103 | 綠 藍 紫 | 203 | 綠 黃 白 |
| | 108 | 紅 綠 藍 | 208 | 綠 藍 黃 |
| 4 | 104 | 紅 綠 白 | 204 | 綠 紫 黃 |
| | 109 | 綠 藍 橙 | 209 | 紅 紫 黃 |
| 5 | 105 | 紅 藍 紫 | 205 | 紅 紫 白 |
| | 110 | 紅 綠 黃 | 210 | 綠 橙 黃 |

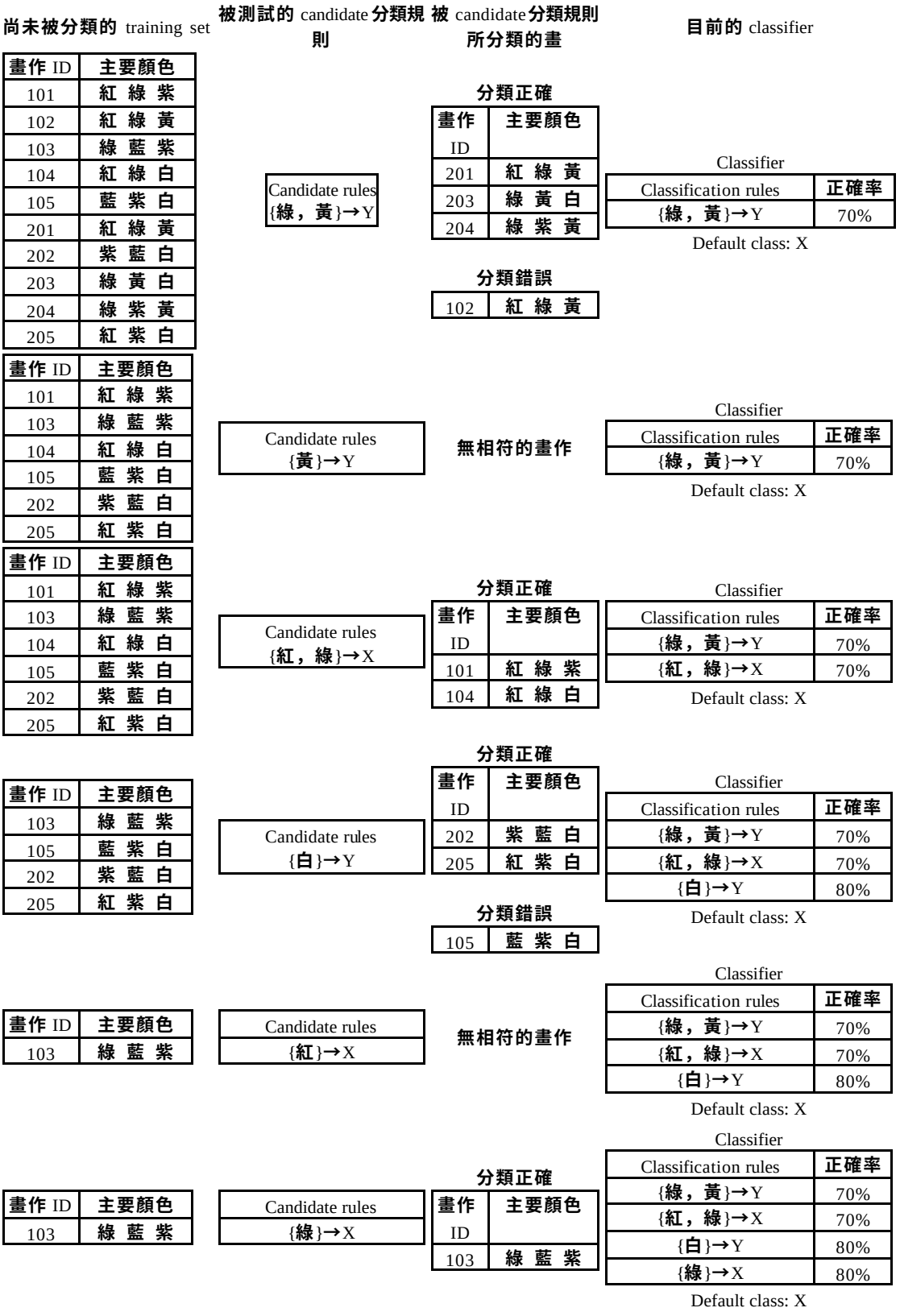



**圖 3.12 建立 classifier 的過程**

### 3.3.3 Multiple-Feature Variant Support 分類器(MFVS)

SFVS classification 考慮的是每一分類內的資料特性不同。然而，畫家的風格可能是由不同的低階繪畫特徵所組合而成，例如，一個畫家的風格是紅色系顏色與低彩度且低亮度的顏色(*, 25, 25)常一起用，另外他在相鄰顏色上使用藍色與白色搭配。

雖然經由 multi-level association rule 可以知道畫家常用的顏色組合，或彩度、色度、明亮度的喜好。但是基本的 associative classification 只能找出在同一種 feature 間的相關性，卻不知道除了常用的顏色組合外，畫家是否一起使用了相鄰顏色的搭配組合在作畫。

**Algorithm *Single-Feature-Variant-Supports-Classification***
Input: painting database *DB*, candidates of min_sup *MS*, *Pattern (Color histogram, Color Bins, or Color Relationships)*
Output: *Classifier*
1.divide training data of each class *y* into 5 subsets $T_{y,k}$ of approximately equal size
2.for each combination of *min_sups* of all categories do
3. for $k$ = 1 to 5 do
4. for each class *y* do
5. $training\_set_y = \bigcup_{i \neq k} T_{y,i}$
6. $validation\_set_y = T_{y,k}$
7. mine mixed frequent patterns from $training\_set_y$
8. for each frequent patterns do
9. calculate confidence of frequent patterns
10. *Classifier* = ***Training-Classifier***
11. classify each $validation\_set_y$ by *Classifier* and store the accuracy $a_k$
12. accuracy of the *min_sups* combination $= \sum a_k / 5$
13. return the *Classifier* with the highest accuracy

圖 3.13 SFVS classification 演算法

因此，我們基於 SFVS classification 提出了另一個改進的演算法 multiple-feature variant support (MFVS) classification，允許同時使用不同低階影像特徵進行分類。如圖 3.14 所示即是不同特徵組合而成的 classifier 的例子。

因此 MFVS classification 會結合由 multi-level association rule mining、2D sequential pattern mining, C4.5 等產生的顏色、相鄰顏色與 MPEG-7 descriptor 等多種特徵的 frequent pattern，依其 confidence 排列，建立起 classifier。同 SFVS classification 的作法，MFVS

classification 利用 five-fold cross validation 進行 min_sup 配對的微調，並傳出分類效果最佳的 min_sup 配對，及對應的分類器。MFVS classification 的演算法如圖 3.15。

Dominant color:＜紅 (*,25,25)＞→class: X
Relationship: ＜藍・白＞→ class: X
Dominant color: ＜黃 綠＞→ class: Y
Dominant color descriptor: spatial coherency > 15 → class: X
Color layout descriptor: (橙<紫<(*,25,*)，紫<橙< (*,25,*)) → class: Y
default class: class X

圖 3.14 MFVS classifier 的例子

### 3.3.4 Bagging Predictors

接下來將介紹如何應用 bagging predictors 到我們的 MFVS 分類器。

Bagging predictors 是一個 framework，其內容是結合數個不同的分類器，以形成一個分類效果較好的 bagging classifier [4]。這些分類器最主要的差別是其 training set 間有極大的不同，每個分類器的 training set 都是原本的 training set 的子集，使得這些分類器可以辨識原本 training set 中較細微的部分。而 bagging classifier 在分類新進的資料方法是分別把這新進的資料給不同版本的分類器進行分類後，最後再進行投票。bagging classifier 以多數決的方式決定類別。我們把 five-fold cross validation 過程中所產生的五個 MFVS classifier 利用 bagging predictors 的方法結合起來，結果如圖 3.16 所示。

**Algorithm *Multiple-Feature-Variant-Supports-Classification***

Input: painting database *DB*, candidates of *min_sup MS*

Output: *Classifier*

1. divide training data of each category *y* into 5 subsets $T_{y,k}$ of approximately equal size
2. for each combination of *min_sups* and multiple features of all classes do
3. for $k = 1$ to 5 do
4. for each category *y* do
5. $training\_set_y = \bigcup_{i \neq k} T_{y,i}$
6. $validation\_set_y = T_{y,k}$
7. mine frequent Color histogram, Dominant Color and Color relationships from $training\_set_y$
8. for each frequent Color histogram, Dominant Color and Color relationships do
9. calculate confidence of pattern
10. *Classifier* = ***Training-Classifier***
11. classify each $validation\_set_y$ by *Classifier* and store the accuracy $a_k$
12. accuracy of the *min_sups* and multiple patterns combination $= \sum a_k / 5$

**圖** 3.15 MFVS classification **演算法**

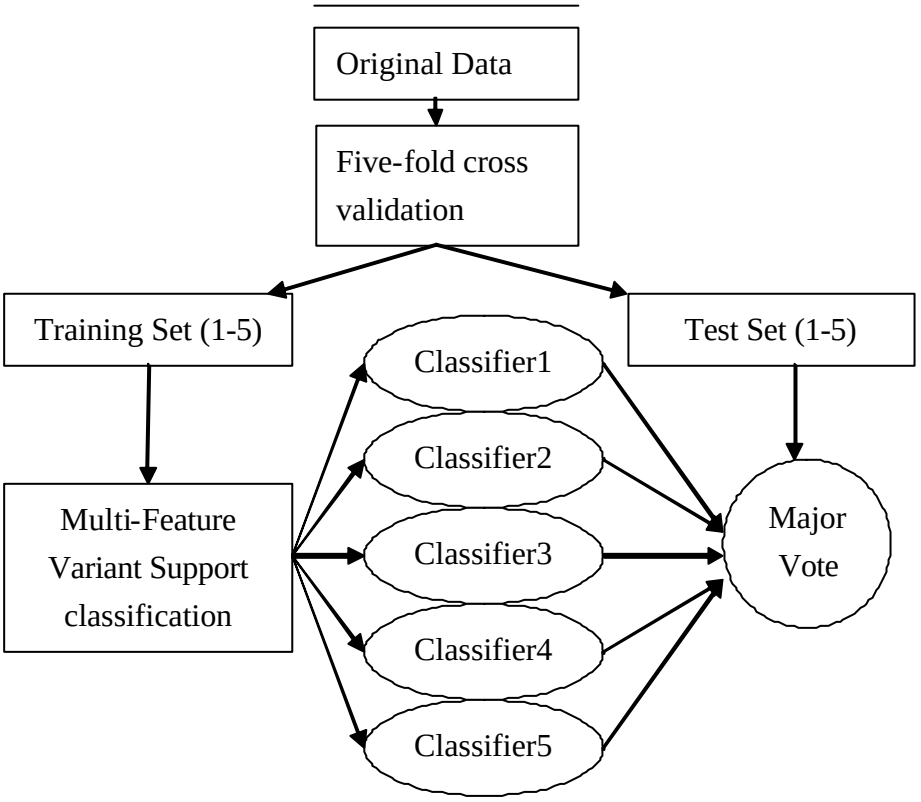



**圖** 3.16 Bagging predictors