{plik w7som.tex 27 listopada 2007, popr.: 20.11.09, 14.12.2010 }

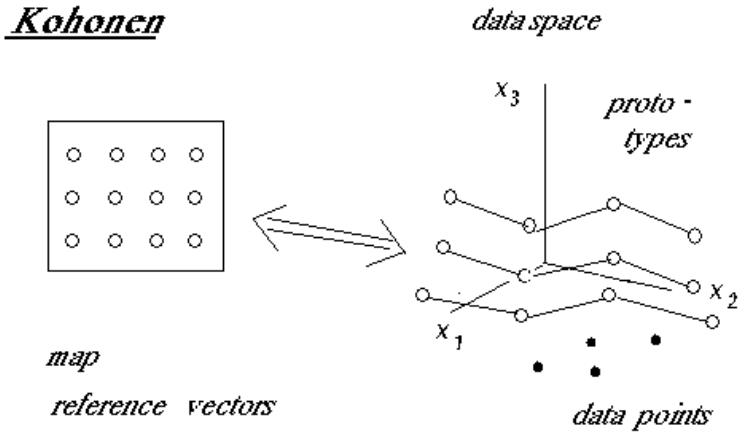
7 Mapy Kohonena

7.1 Zasady konstrukcji mapy SOM

SOM jest skrótem od 'Self Organizing Maps'. Typowy przedstawiciel takich sieci: Mapa Kohonena na płaszczyźnie[1]. SOMy realizują generalnie dwa zadania:

1. Wektorowej kwantyzacji (kompresji danych).
2. Wizualizacja przestrzennej organizacji danych wejściowych.

Istotną nowością jest pojawienie się tutaj **mapy** pokazującej topologiczne sąsiedztwo odwzorowywanych punktów danych z R^d. Ponadto, neurony mapy uczą się w warunkach konkurencji.



Rysunek 7.1: *Koncepcja mapy Kohonena. W trakcie 'uczenia' mapy punkty–prototypy są przyciągane do obszarów njwiększej gęstości danych; jednak przy ograniczeniu, że musi być zachowane sąsiedztwo punktów referencyjnych na mapie {kohtif1.eps}*

Mapa może być różnego kształtu i wymiaru. Mapa może być jednowymiarowa, dwuwymiarowa na płaszczyźnie lub torusie, trójwymiarowa itp. Najbardziej typową i najczęściej używaną jest mapa dwuwymiarowa o $M = m1 \times m2$ neuronach. Analizowane dane mają być odwzorowane na mapie (rozmiary mapy są zadawane przez użytkownika). Samoorganizacji podlega cały zbiór danych: nie rozróżnia się próbki uczącej i testowej, chociaż istnieje możliwość skonstruowania mapy dla części danych (stanowiących próbkę uczącą), a następnie odwzorowanie innej części danych na tej samej mapie.

7.2 Podstawowe pojęcia i oznaczenia

7.2.1 Wektory referencyjne i wektory wagowe

Niech $\mathbf{x} = (x_1, \ldots, x_d)^T$ oznacza d-wymiarowy wektor danych, tzw. próbkę lub wzorzec. Wektor $\mathbf{x}$ może być interpretowany jako punkt d-wymiarowej przestrzeni: $\mathbf{x} \in R^d$.

Zakładamy, że na mapie znajduje się M neuronów. Liczba M jest deklarowana, lub też przyjmuje się domyślnie: $M = 5 \times \sqrt{N}$, gdzie N oznacza liczbę wektorów danych.

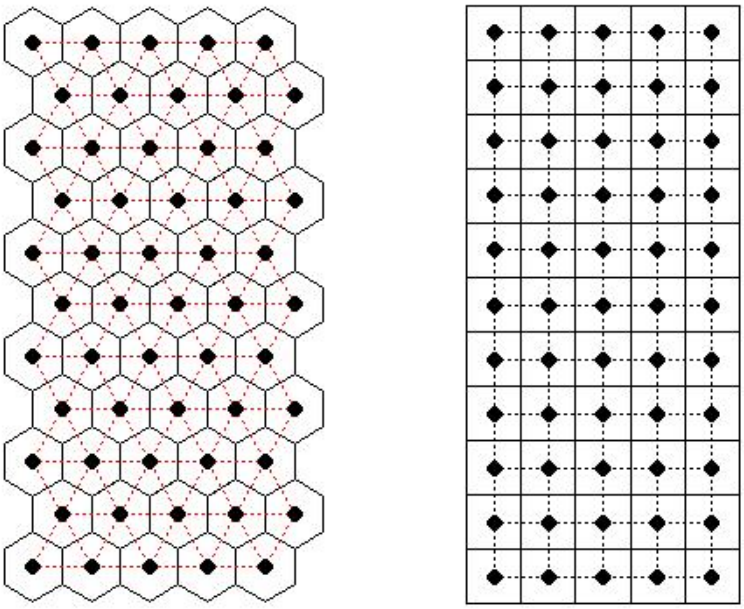
[1] opisana np. w książce Osowskiego, str 249–275

Każdy neuron (o numerze $m = 1, \ldots, M$) jest scharakteryzowany swoim **wektorem referencyjnym** $\mathbf{r}_m$ pokazującym pozycję tego neuronu na mapie. Generalnie, punkty referencyjne $\mathbf{r}_i$ połączone ze sobą określają węzły siatki pokrywającej mapę. Ponadto, Każdy neuron (m) posiada również swój **wektor wag** $\mathbf{w}_m$, nazywany czasem **wektorem kodowym**, **wektorem Voronoia**, lub po prostu **prototypem**). Wektory $\{\mathbf{w}_m\}$ leżą w przestrzeni danych: $\mathbf{w}_m \in R^d$. Posumowując: Każdy wektor referencyjny wskazuje na odpowiedni wektor wag w przestrzeni R^d – i na odwrót. Tak więc istnieje jednoznaczna odpowiedność:

$$\mathbf{r}_i \rightleftharpoons \mathbf{w}_m, \quad m = 1, \ldots, M\,.$$

7.2.2 Przykładowe mapy dwuwymiarowe

Na rysunku 7.2 podajemy dwa przykłady mapy dwuwymiarowej utworzonej na płaszczyźnie. Neurony są tam ułożone w siatkę heksagonalną i prostokątną.



Rysunek 7.2: *Sąsiedztwo na mapach Kohonena. Strona lewa: Neurony ułożone w siatkę hexagonalną. Strona prawa: neurony w siatce prostokątnej. {gridhr.JPG}*

Mapa składa się z takich samych jednostek (kwadratów lub sześcioboków foremnych) parkietujących mapę. W środku każdej jednostki znajduje się punkt nazywany wektorem referencyjnym; każdy taki punkt reprezentuje jeden neuron. Wektory–punkty referencyjne na mapie są połączone w sztywną siatke prostokątną lub hexagonalną, która nie ulega zmianie podczas uczenia. Siatki takie są pokazane na rys. 7.2. Widać tam, że na siatce heksagonalnej każdy neuron ma 6, a na prostokątnej 8 sąsiadów pierwszego rzędu. Można liczyć również sąsiedztwo 2-go rzędu, i dalsze.

7.2.3 Niektóre zasady określania sąsiedztwa

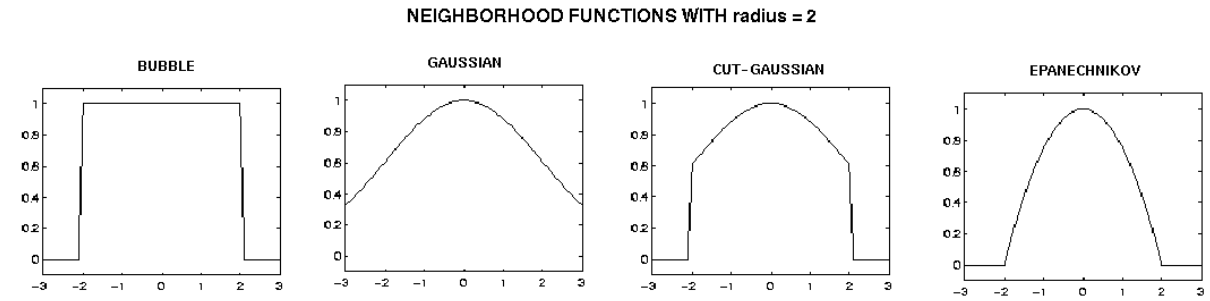
Przy uczeniu się mapy bardzo ważną rolę odgrywa pojęcie sąsiedztwa neuronów. Sąsiedztwo to wyznacza się według położenia wektorów referencyjnych na mapie.

Sąsiedztwo neuronu oznaczonego numerem c będziemy oznaczać $\mathcal{N}_c$ (od neighbourhood). Jeżeli chcemy wyraźnie napisać, że jest to sąsiedztwo neuronu c który zwyciężyl w k-tej iteracji, to zapiszemy

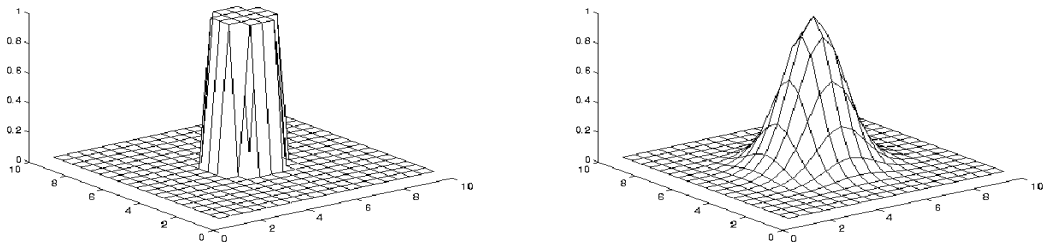
$$\mathcal{N}_c = \mathcal{N}_c(k).$$

Zmienna c oznacza na ogół numer zwycięskiego neuronu (ang. *conqueror*) w czasie uczenia ze współzawodnictwem. Na ogół sąsiedztwo $\mathcal{N}_c(k)$ charakteryzuje się pewnym promieniem, który maleje wraz z upływem czasu uczenia t, czyli w miarę zwiększania się wskaźnika k.

Sąsiedztwo może być sztywne (0 - nie należy, 1 - należy) lub też wyznaczane funkcją przynależności (ang. membership function) przyjmującej wartości rzeczywiste z przedziału [0,1]. Funkcja intensywności przynależności do $\mathcal{N}_c$ jest oznaczana symbolem $h_{cm}(k)$ lub $G(m, c, k, \boldsymbol{\theta})$, gdzie m, c są dwoma numerami ze zbioru $1, \ldots, M$, a symbol θ oznacza dodatkowe parametry characteryzujące daną funkcję. Funkcje te opierają się na odległości neuronu m od zwycięskiego neuronu c. Funkcje te mają bardzo często postać funkcji radialnych scentrowanych w punkcie $\mathbf{r}_c$.



Rysunek 7.3: *Jednowymiarowe funkcje sąsiedztwa dla promienia R=2: bubble, gaussian, cut-gaussian, Epanechnikov {figs7/neigh4.ps}.*



Rysunek 7.4: *Funkcje sąsiedztwa: bubble i gaussian określone na płaszczyźnie. Funkcja bubble wyznacza sąsiedztwo w sposób ostry: 1 - tak, 0 - nie; funkcja gaussian w sposób łagodny jako liczbę z przedziału (0, 1]. {bubble2.ps, gauss2.ps}*

Najbardziej popularnymi funkcjami sąsiedztwa są *bubble* i *gaussian*. Są one pokazane na rysunku 7.4. Przykładowo funkcja radialna *gaussian* przyjmuje postać ([2], str. 257):

$$h_{c,m}^{gauss} = exp\left(-\frac{||\mathbf{r}_m - \mathbf{r}_c||^2}{2\sigma^2(k)}\right),$$

gdzie wektory $\mathbf{r}_m$, $\mathbf{r}_c$ są odpowiednimi wektorami referencyjnymi, symbol k oznacza numer iteracji, $2\sigma^2(k)$ oznacza promień sąsiedztwa. Jak widać z tej definicji, wielkość promienia sąsiedztwa $2\sigma(k)^2$ maleje ze wzrostem k^2.

Jeszcze inne funkcje sąsiedztwa (*cut-gaussian* i *Epanechnikov*) są pokazane na rysunku 7.3.

Podsumowując: Funkcje sąsiedztwa określamy najczęściej w ten sposób, aby przyjmowały one wartości z przedziału (0,1] lub nawet przyjmowały tylko dwie wartości: 1, gdy dany element należy, i 0, gdy nie należy do sąsiedztwa.

W szczególności, funkcja $h_{cm}(k)$ może być określona w następujący sposób:

$$h_{cm}(k) = \begin{cases} 1 & \text{gdy } \; m \in N_c(k)\,, \\ 0 & \text{gdy } \; m \notin N_c(k)\,. \end{cases}$$

Funkcja $h_{cm}(k)$ może też zależeć bezpośrednio od odległości $D = D(\mathbf{r}_m, \mathbf{r}_c)$ między odpowiednimi wektorami referencyjnymi, np.

$$h_{cm}(k) = \begin{cases} \mathrm{g}(D(c,m)) & \text{gdy } \; m \in N_c(k)\,, \\ 0 & \text{gdy } \; m \notin N_c(k)\,, \end{cases}$$

gdzie $g(\cdot)$ jest funkcją malejącą (dokładniej: nierosnącą) swojego argumentu. W szczególnym przypadku mogą to być np. funkcje *bubble* lub *gaussian* pokazane na rysunku 7.4.

7.3 Uczenie się neuronów mapy Kohonena

7.3.1 Dwie fazy uczenia

Na ogół uczenie przebiega w dwóch fazach. Najpierw przyjmuje się duże wartości współczynnika uczenia η (patrz następna podsekcja) i duży promień sąsiedztwa. W drugiej fazie (*fine tuning*) obydwie te wielkości ulegają istotnemu zmniejszeniu; w szczególności promień sąsiedztwa maleje do zera.

Pierwsza faza – przebiega według zasady *Winner Takes Most* (*WTM*) – promień sąsiedztwa jest duży, co powoduje, że oprócz neuronu-zwycięzcy również jego sąsiedzi (z mapy) zmieniają swoje wektory kodowe. Współczynnik uczenia η jest w tej fazie stosunkowo duży. Zmiany wag następują według wzoru (7.1):

$$\mathbf{w}_m(k+1) = \mathbf{w}_m(k) + \eta(k)\; h_{cm}(k)\; [\mathbf{x}(k) - \mathbf{w}_m(k)], \quad m = 1, \ldots, M. \tag{7.1}$$

We wzorze powyższym:
⋆ $\eta(k)$ oznacza współczynnik uczenia - piszemy o nim w następnej podsekcji,
⋆ c oznacza numer wektora–zwycięzcy, tj. numer wektora $\mathbf{w}$ znajdującego się najbliżej prezentowanego w k-tym kroku wektora $\mathbf{x}(k)$,
⋆ wartość funkcji $h_{ci}(k)$ określa, w jakim stopniu należy uwzględnić przynależność neuronu i do sąsiedztwa zwycięskiego neuronu o numerze $c = c(k)$.

Tak więc, przy każdej prezentacji kolejnego wektora $\mathbf{x}$ zostanie do niego przyciągnięty odpowiadający mu wektor-zwycięzca $\mathbf{w}_c$ który pociąga za sobą wagi tych neuronów które znalazły się w jego sąsiedztwie na mapie.

[2] W niektórych źródłach jako funkcję gaussowską przyjmuje się po prostu

$$h_{c,m}^{gauss} = exp\left(-\frac{||\mathbf{r}_m - \mathbf{r}_c||^2}{\sigma(k)}\right),$$

a promieniem sąsiedztwa nazywa się wielkość $\sigma(k)$

Druga faza uczenia. Obowiązuje tu zasada *Winner Takes All* (*WTA*). Adaptacji podlega tylko neuron–zwycięzca c, ponieważ promień sąsiedztwa zmalał do zera. Zmienia się tylko wektor wagowy $\mathbf{w}_c$ według wzoru:

$$\mathbf{w}_c(k+1) = \mathbf{w}_c(k) + \eta(k)\,[\mathbf{x}(k) - \mathbf{w}_c(k)].$$

7.3.2 Współczynnik uczenia

Współczynnik uczenia $\eta(k)$ maleje zazwyczaj wraz z upływem czasu uczenia wyznaczanego numerem iteracji k.

Niech T oznacza maksymalną liczbę iteracji. Liczbę tę ustala się z góry[3]. Dość często stosuje się następujące wzory na zmniejszanie współczynnika uczenia:

1. Liniowe zmniejszanie
   $\eta(t) = \eta_0\,(T-t)/T, \quad t = 1, 2, \ldots, T$.
2. Wykładnicze zmniejszanie
   $\eta(t) = \eta_0\, exp(-Ct), \quad t = 1, 2, \ldots, T, \quad C > 0$ jest pewną stałą.
3. Hiperboliczne zmniejszanie
   $\eta(t) = C_1/(C_2 + t), \quad t = 1, 2, \ldots, T, \quad C_1, C_2 > 0$ pewne stałe.
4. Indywidualny współczynnik uczenia, np.
   $\eta_i(t) = 1/n_i(t)$, gdzie $n_i(t)$ oznacza liczbę zwyciestw i-tego neuronu.

7.3.3 Uczenie wsadowe

Dotychczas omawiane uczenie *sekwencyjne*, inaczej *na bieżąco*, lub *on–line* polegało na tym, że dla $t = 1, 2, \ldots$ prezentowaliśmy sieci kolejno pojedyncze wektory danych $\mathbf{x}(t)$, po czym następowało uaktualnienie wag zwycięskiego neuronu (i ewentualnie jego sąsiadów) wedlug zasad opisanych zasadami WTA lub WTM.

Wsadowe uczenie podobno (przynajmniej w przypadku SOM-ów) jest znacznie szybsze i bardziej stabilne; polega na wykonywaniu aktualizacji wag tylko na zakończenie każdej epoki (tj. gdy zostały zaprezentowane wszystkie próbki danych wynikające z ich randomizacji). Wariant uczenia wsadowego jest wariantem domyślnym w pakiecie somtoolbox2 przy trenowaniu sieci Kohonena. Algorytm uczenia wsadowego jest następujący (por. Skubalska [6], str. 187, za Kohonenem [1], również Vesanto [3], str. 9):

1. Ustal M początkowych wektorów kodowych. Początkowymi wektorami kodowymi mogą być wektory wygenerowane losowo, lub też M dowolnych wektorów danych z próbki uczącej.
2. Rozpocznij nową epokę i przedstawiaj sieci według w porządku zrandomizowanym (np. spermutowana tablica danych) elementy próbki uczącej $\mathbf{x}_k = \mathbf{x}(k)$. Zaprezentuj sieci w ten sposób N wektorów danych.

   Zapamiętuj w czasie prezentacji dla każdego wektora $\mathbf{w}_m$ zbiór wektorów uczących $\{\mathbf{x}_k\}$ które oddziaływałyby na $\mathbf{w}_i$ w zwykłym algorytmie uczenia, oraz intensywność sąsiedztwa $h_{c(\mathbf{x}_k)m}$.

[3] należy zwrócić uwagę, co oznacza faktycznie maksymalna liczba iteracji; często jest to liczebność próbki uczącej przemnożona przez liczbę epok

3. Na koniec epoki wyznacz nowe wartości wag (symbol $c(\mathbf{x}_k)$ oznacza neuron wygrywający przy prezentacji wektora danych $\mathbf{x}_k$, natomiast N jest ogólną liczebnością próbek uczących)
$$\mathbf{w}_m = \sum_{k=1}^{N} \mathbf{x}_k h_{c(\mathbf{x}_k)m} / \sum_{k=1}^{N} h_{c(\mathbf{x}_k)m}$$
4. Jeśli nie jest spełnione kryterium STOP-u (nie podaliśmy go), wróć do kroku 2.

7.3.4 Algorytm organizowania się mapy

Uczenie się sieci Kohonena przebiega warunkach konkurencji z dodatkowo nałożonym warukiem sąsiedztwa wektorów referencyjnych. Uczenie dokonuje się w następujący sposób: Po przedstawieniu kolejnego wektora $\mathbf{x}(k)$ zostaje znaleziony punkt–zwycięzca w przestrzeni R^d. Na mocy ogólnej zasady zwycięzca ten uzyskuje przywilej aktualizacji swoich wag, tzn. przysunięcia się w kierunku punktu $\mathbf{x}(k)$. Zwycięzca ma prawo pociągnąć za sobą swoich sąsiadów z mapy, którzy dzięki bliskości ze zwycięzcą mogą dzielić z nim po części przywilej adaptacji swoich wag i zbliżyć się również w kierunku przedstawionego wektora $\mathbf{x}(k)$. W pierwszej fazie uczenia przesunięcia wektorów kodowych moga być duże, w drugiej fazie uczenia znacznie mniejsze, gdyż promień sąsiedztwa zawęża się do zera. Promień zerowy oznacza, że zmienia się tylko lokalizacja wektora-zwycięzcy.

Wskutek takiego uczenia wektory kodowe w przestrzeni R^d *(a)* przemieszczają się w kierunku największych skupień danych i jednocześnie *(b)* zaczynają się grupować wokół siebie zgodnie z sąsiedztwem na mapie.

Ostatecznie cała przestrzeń R^d w której znajdują się dane, zostanie podzielona na strefy wpływów poszczególnych neuronów (obszary Voronoi'a). Sąsiedztwo wektory referencyjnych na mapie powinno odzwierciedlać (topologicznie) sąsiedztwo wektorów kodowych w przestrzeni danych.

Kohonen stwierdził, że można znacznie przyspieszyć proces uczenia, jeśli zamiast czysto losowego inicjowania wektorów losowych rozstawi się je na płaszczyźnie pierwszych dwóch składowych głównych analizowanych danych. To spostrzeżenia przyczynia się do wprowadzenia częściowego porządku, i zaoszczędza (być może) tysiace iteracji, które byłyby potrzebne aby takie częściowe uporządkowanie otrzymać[4].

Sytuacja taka jest pokazana w module demonstracyjnym som_demo1 pakietu somtb2. Rysunek 7.5 (otrzymany tym modułem) przedstawia proces formowania się mapy dla danych wygenerowanych w trójwymiarowej kostce.

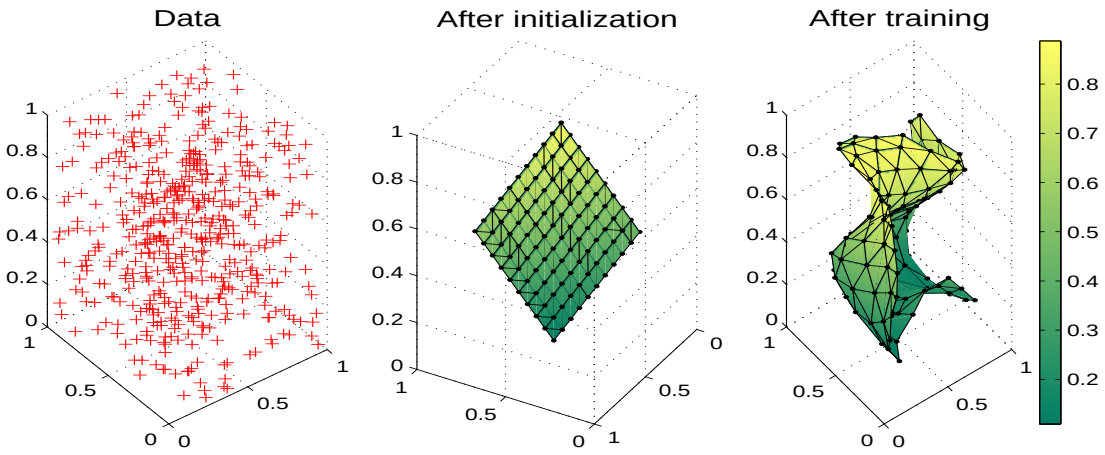
Kohonen nazwał proces tworzenia reprezentantów danych kwantowaniem wektorowym (*Vector Quantization*), lub dokładniej: adaptacyjnym kwantowaniem wektorowym (*LVQ, Learning Vector Quantization*). Wektory wagowe neuronów zostały przez Kohonena nazwane *słowami kodowymi* (*codebook vectors*), a ich kolekcja – *książką kodową* (*codebook*).

7.3.5 Jakość reprezentacji

Rozważa się tu najczęściej dwa wskaźniki (por. Vesanto i in., [4]):

Błąd kwantyzacji – liczony jako średnia odległość punktów–danych od ich reprezentantów, czyli prototypów (data representation accuracy, average quantization error between data vectors and their BMUs - best maching units). Zamiast średniej odległości można obliczać inny wskaźnik rozproszenia, np. medianę.

[4]Mapy tworzone w pakiecie somtb2 korzystają domyślnie z tej zasady – chyba że wyrażono inne życzenie



Rysunek 7.5: *Organizowanie się mapy Kohonena na płaszczyźnie dwóch pierwszych składowych głównych. Lewa: punkty–dane zaznaczone krzyżykami. Środek: Startowa płaszczyzna rozpięta na PC1 i PC2 wraz z siatką punktów referencyjnych. Prawa: Punkty kodowe w przestrzeni R^3 odpowiadające punktom referencyjnym w R^2. {demo1ss.eps}*

Błąd topologicznej reprezentacji - określany jako procent punktów-danych, dla których pierwsi dwaj najbliżsi reprezentanci nie są sąsiadami na mapie (data set representation accuracy, the topographic error; percentage of data vectors for which the first- and second-BMU are not represented in adjacent units of the map).

7.4 Dostępne dla nas oprogramowanie

SOM_PAK – zestaw programów w języku C autorstwa Kohonena i jego zespołu. Jest dostępny w internecie, tworzy grafikę w postaci .ps lub .eps; wymaga kompilacji (make). Pracuje zarówno na Unix-ie jak i na komputerach PC. Adres internetowy: http://www.cis.hut.fi/research/som_lvq_pak

somtoolbox vs. 2 ([4]) – zestaw skryptów w postaci M-files autorstwa J. Vesanto i współautorów. Dostępny w internecie pod adresem (wersja z 17.03.2005r) : http://www.cis.hut.fi/projects/somtoolbox

Netlab Pakiet ten zawiera funkcje som, somtrain, somfwd umożliwiające trenowanie mapy prostokątnej, oraz plik demom1 pokazujący możliwości korzystania z tych funkcji. Nie wykreśla map.

7.5 Przykłady zastosowań

Przykład 1. Mapa Kohonena dla 49 województw polskich

Pokażemy tu wyniki wizualizacji typu *umat* otrzymane za pomocą pakietu som_pak Kohonena. Rozpatrujemy tablicę danych o wymiarze 49×9, której wiersze odpowiadają 49 województwom polskim (dane pochodzą z r. 1990), z których każde zostało scharakteryzowane przez 9 cech socjo-ekonomicznych (kolumny tablicy). Tym samym każde województwo może być interpretowane jako punkt w R^9.

Na rysunku 7.6 pokazujemy mapy otrzymaną w wyniku obliczeń oryginalnym programem SOM_PAK Kohonena. Mapy te zostały wyznaczone dla tych samych, ale mają różne rozmiary: Pierwsza mapa jest rozmiaru 6×6, o liczbie neuronów $M = 54$, druga 10×10 o liczbie neuronów $M = 100$.

To co widzimy na mapie jest obrazem wielowymiarowej przestrzeni. Sporządzona mapa składa się z obszarów heksagonalnych, w których środkach znajdują się wektory referencyjne odpowiadające wektorom kodowym umiejscowionym w R^9. Oznacza to, że każdemu neuronowi o wspÓłrzędnych $\mathbf{r}_m$ $(m = 1, \ldots, M)$ na mapie odpowiada prototyp (codebook vector) $\mathbf{w}_m$ w przestrzeni wejściowej R^9.

Faktyczne odległości między wektorami $\mathbf{w}_i$ są obrazowane odcieniami szarości na mapie (według techniki 'umat'): obszary bliskie są jasne, ciemny kolor oznacza duże odległości, a więc może oznaczać granice klasterów z R^9 (inne pakiety, np. somtb2, operują w tym celu kolorami z odpowiednim kluczem na oznaczenie bliskich i dalekich wektorów wagowych). Na utworzonej mapie właściwe heksagony (j) zawierające węzły mapy są otoczone dodatkowymi heksagonami pokazującymi kolorystycznie, jaka jest średnia odległość wektora kodowego (j) od sąsiadujących z nim wektorów kodowych, jeśli patrzymy w kierunku wschodnio-południowym mapy (odległości te wygładzone numerycznie).

Taki sposób wizualizacji nosi nazwę 'umat' – od informatyka o nazwisku Ultsch, który zaproponował taki sposób wizualizacji map. Technikę tę omówimy trochę bardziej dokładnie w sekcji 7.7.

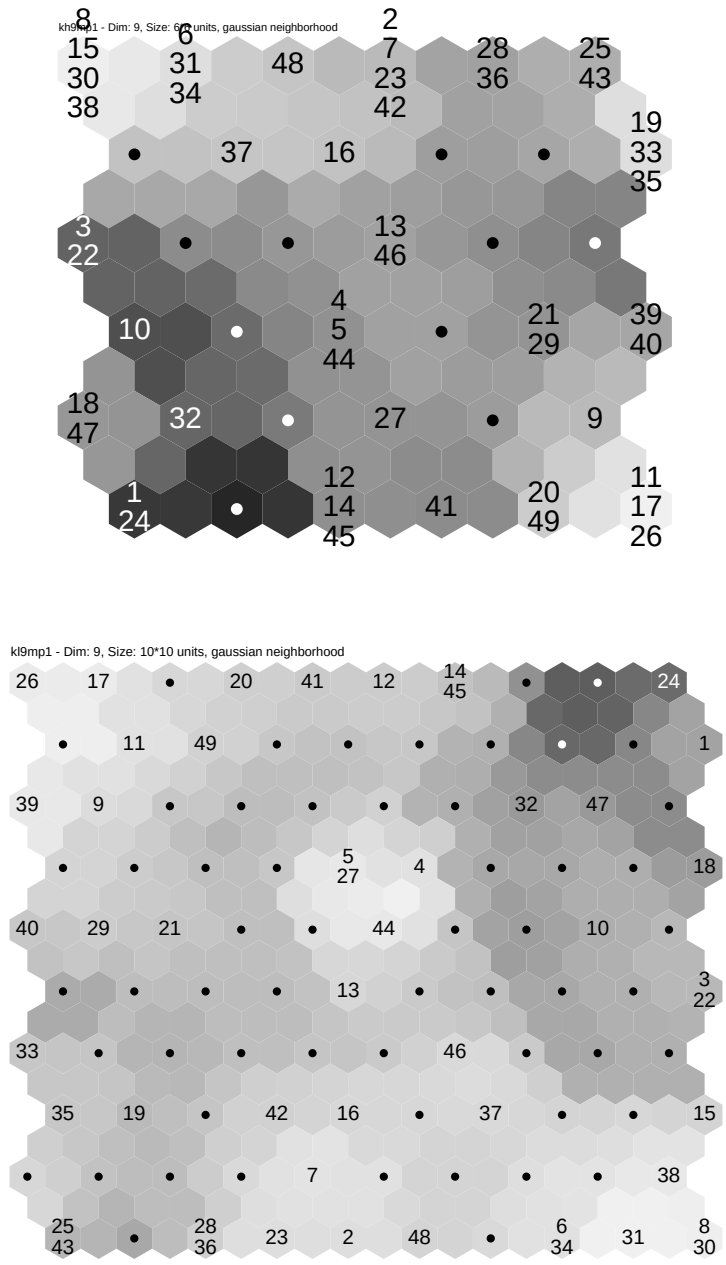
Mając klucz do województw możemy próbować interpretować powstałe zgrupowania. Punkty 1 i 24 to województwa Warszawskie i Łódzkie. Punkty 18 i 47 to Kraków i Wrocław; punkt 32: Poznań; punkt 3 i 22 to Białystok i Lublin; punkt 10: Gdynia-Gdańsk. Wszystkie te punkty to miasta uniwersyteckie z pewną tradycją. Tworzą one wyraźny klaster, oddzielony od pozostałych punktów rozleglą doliną zawierającą puste węzły.

Innym widocznym na mapie klasterom można przypisać również ciekawą interpretację.

Mapy zobrazowane na rysunkach 7.6 otrzymano z tych samych danych, jednak przy założeniu różnych rozmiarów mapy. W obu mapach początkowe wagi (czyli prototypy danych) były inicjowane losowo. W rezultacie powstałe mapy są do pewnego stopnia 'podobne', – mówimy, że jest to topologiczne podobieństwo. Przypatrując się położeniu 'geograficznemu' punktów na mapie, stwierdzamy dużą odmienność rozłożenia punktów–województw na obu mapach.

Punkty 43 i 25 są położone w przeciwległych narożnikach. Punkty 8, 15, 30, 38 z północno-zachodniego narożnika mapy 6×6 pojawily się w przeciwległym narożniku drugiej mapy ulegając rozbiciu: tylko punkty $< 8, 30 >$ znalazły się w narożniku południowozachodnim tej mapy, natomiast punkt 38 stał się pierwszym sąsiadem, a punkt 15 drugim sąsiadem wymienionych punktów $< 8, 30 >$.

Start z płaszczyzny pierwszych 2 składowych głownych daje bardziej podobne reprezentacje na mapie.



Rysunek 7.6: *Mapy Kohonena o wymiarach* 6×6 *(góra) i* 10×10 *(dół) obrazująca zgrupowania 49 województw polskich ze względu na 9 cech socjo-ekonometrycznych. Niektóre węzły mapy pozostały puste, inne wektory kodowe zdołały przyciągnąć po kilka punktów-województw. Interesujący jest klucz do wojwwództw: np. 1 - Warszawa, 24 - Łódź, 18 - Kraków, 47 - Wrocław, 32 - Poznań, 3 - Białystok, 22 - Lublin, 10 - Gdańsk, 3 - Białystok, 22 - Lublin, 10 - Gdańsk {map1.eps, map2.eps}*

Dalsze przykłady zastosowań sieci Kohonena

Osowski (str 268–275) podaje następujące przykłady zastosowań sieci samoorganizujących się. Są to przykłady opisujące rzeczywiste dane i wynikające z rzeczywistych zapotrzebowań, takie jak: • Kompresja obrazów, • Wykrywanie typu uszkodzeń, • Prognozowanie obciążeń systemu elektroenergetycznego.

7.6 Pakiet Somtoolbox2 – ogólne wprowadzenie

Ogólne informacje

Pakiet Somtoolbox2 nazywany dalej w skrócie somtb2 został opracowany przez Vesanto i wsp. z Helsinek[5]. Pakiet ten, zaprogramowany pierwotnie w języku MATLAB 5 ulega ewolucji wraz z kolejnymi rozszerzeniami Matlab-a, które wprowadzają znaczne zmiany w możliwościach programowania. Dzisiaj (14.12.2005) mamy do dyspozycji Matlab 7.0 (R14) i wersję Somtoolbox version 0Beta 2.0 zawierająca około 150 funkcji służących głównie konstrukcji map Kohonena i eksploratywnej wizualizacji danych oraz map Kohonena.

Podstawą obliczeń są dwie struktury: data-struct (sD) i som-strukt (sM). Są to obiekty, które zawierają nie tylko dane (w przypadku sD) i wektory kodowe (w przypadku sM), ale również różne pomocnicze informacje w rodzaju etykiet cech i osobników, informacji o normalizacji danych, typie i strukturze mapy, oraz historii uczenia (się) mapy.

Moduły demonstracyjne

O możliwościach pakietu somtb2 możemy się przekonać uruchamiając moduły demonstracyjne. Jest ich cztery. I tak:

som_demo1 – Pokazuje na prostych przykładach generowanych danych dwu- i trzy-wymiarowych, jak się tworzy mapę, jak wektory kodowe podczas procesu trenowania adaptują się do danych. Wyjaśnia, co to jest BMU (Best Matching Unit) i som_quality (quantization error i topology representation).

som_demo2 – Podstawowe funkcje (basic usage), co można pokazać na mapie.

som_demo3 – Moduł jest poświęcony różnym możliwościom wizualizacji mapy Kohonena za pomocą funkcji som_show i som_grid.

som_demo4 – Pokazuje różne możliwości eksploratywnej analizy danych na podstawie wizualizacji sporządzanej za pomocą procedur graficznych znajdujących się w pakiecie.

Cztery grupy procedur

W dalszym ciągu omówimy cztery grupy procedur służących konstrukcji i wizualizacji map Kohonena. Są to:

- Tworzenie struktury danych (oznaczanej umownie `sD`) – Struktura ta pełni funkcję pomocniczą, nie jest niezbędna do utworzenia mapy, ale znacznie to ułatwia.
  Mamy tu procedury som_read_data i som_data_struct.

[5] Vesanto J., Himberg J., Alhoniemi E., Parhankangas J., *SOM Toolbox for Matlab 5*. Som Toolbox team, Helsinki University of Technology, Finland, Libella Oy, Espoo 2000, 1–54. Strona domowa: http://www.cis.hut.fi/projects/somtoolbox/ Version 0beta 2.0, Ostatnia aktualizacja: 17 marzec 2005

- Tworzenie struktury mapowej (oznaczanej umownie sM) – to jest jądro pakietu, któremu jest podporządkowanych wiele innych funkcji tego pakietu. Mamy tu przede wszystkim procedurę som_make.
- Wizualizacja mapy i różnych jej treści – procedura som_show.
- Połączona wizualizacja danych typu plot3 pokazująca również sieć neuronów na których jest oparta mapa Kohonena – procedura som_grid.

Z wymienionymi typami zagadnień i realizującymi je głównymi procedurami związanych jest wiele innych procedur, które służą uszczegółowieniu niektórych tematów pojawiających się w aspekcie głównych zagadnień. Użytkownik ma możliwość dostępu do całego kodu źródłowego i zmieniać go według swojego uznania.

Zanim przejdziemy do omówienia czterech wymienionych grup procedur, powiemy coś o najprostszym wykonaniu obliczeń według wariantów domyślnych.

Szybka konstrukcja mapy i jej wizualizacja

Najprostszy sposób sporządzenia mapy Kohonena i jej wizualizacji polega na wykonaniu następującego skryptu wykorzystującego trzy (z normalizacją: cztery) podstawowe funkcje somtoolbox-u:

```
sD = som_read_data('iris.data');
% alternatywa gdy D jest tablicą n × d znajsdującą się w workspace:
   % sD=som_data_struct(D,'name','sD');
% som_normalize(sD,'var');
sM = som_make(sD);
% som_denormalize(sM);
som_show(sM);
```

Skrypt ten wczytuje dane z pliku tekstowego iris.data. Należy obejrzeć ten plik tekstowy i zobaczyć, jak należy przygotować dane. Próbka tych danych zostanie pokazana niżej. Zaleca się, aby dane te zostały znormalizowane - ale nie jest to niezbędne.

Zostaje utworzona struktura mapowo–sieciowa sM o domyślnej liczbie neuronów i domyślnych rozmiarach mapy. Neurony są reprezentowane dualnie: jako punkty referencyjne na mapie i jako prototypy w przestrzeni danych R^d. Następuje trenowanie mapy, tzn. punkty–prototypy w R^d dostosowują się do gęstości punktów–danych. Następuje swoisty proces uczenia w warunkach konkurencji i wykorzystywaniu sąsiedztwa punktów referencyjnych na mapie.

Utworzona w ten sposób mapa zostanie pokazana na płaszczyźnie za pomocą procedury som_show. Jeżeli przed utworzeniem mapy (som_make) dane zostały znormalizowane, to wizualicja mapy będzie w jednostkach znormalizowanych. Możemy otrzymać wizualizację w jednostkach oryginalnych danych przez wykonanie tzw. denormalizacji (som_denormalize).

7.7 Somtoolbox: Tworzenie struktury danych sD i normalizacja

A. Opis struktury sD, przygotowanie danych do wczytania

Struktura danych (oznaczana w dalszym ciągu umownie symbolem sD, chociaż może to być dowolnie przyjęta nazwa) może być utworzona na dwa sposoby:

- Korzystając z tablicy matlabowskiej postaci D o wymiarach $N \times d$, gdzie N oznacza liczbę 'osobników', a d liczbę cech , podając ewentualnie nazwy zmiennych lub

osobników w parametrach procedury, tablica D powinna się znajdować w przestrzeni roboczej (workspace) Matlaba; korzystamy tu z procedury som_data_struct;

- czytając dane ze specjalnie przygotowanego pliku, w którym umieszczono informacje opisujące tworzoną strukturę danych; korzystamy wtedy z procedury som_read_data.

W dalszym ciągu omówimy szczegółowo obydwa sposoby.

B. Pierwszy sposób – korzystanie z tablicy matlabowskiej D

Sposób ten wykorzystuje tablicę D która została już wczytana do przestrzeni roboczej (workspace) Matlaba. Wtedy możemy utworzyć strukturę sD za pomocą rozkazu:
sD = som_data_struct(D,'name','iris-sD', 'comp_names', {'SepalL', 'SepalW', 'PetalL','PetalW'});

Utworzona w ten sposób struktura danych zawiera nazwy zmiennych ('SepalL, 'SepalW', 'PetalL', 'PetalW'), ale nie zawiera nazw (etykiet) osobników, czyli nazw wektorów wierszy. Nazwy te można dodać za pomocą rozkazu som_label. Przykładowo irysom ze zbioru *iris* można nadać nazwy za pomocą rozkazów:
sD = som_label(sD,'add',[1:50]','Setosa');
sD = som_label(sD,'add',[51:100]','Versicolor');
sD = som_label(sD,'add',[101:150]','Virginica');

C. Drugi sposób – czytanie danych tekstowych

Dane w pliku (np. o nazwie 'iris.data') powinny mieć następującą postać (kropki oznaczają, że opuściliśmy tu pewne fragmenty tych danych):

```
4
#n SepalL SepalW PetalL PetalW
5.1 3.5 1.4 0.2 Setosa
4.9 3.0 1.4 0.2 Setosa
...
5.0 3.3 1.4 0.2 Setosa
7.0 3.2 4.7 1.4 Versicolor
...
...
5.9 3.0 5.1 1.8 Virginica
```

Strukturę zawierającą te dane można utworzyć za pomocą rozkazu:
sD=som_read_data('iris.data'); Oczywiście, nazwa sD jest przykładowa.

Gdybyśmy nie umieścili w czytanym pliku tekstowym nazw-etykiet osobników (nazw przypisanych poszczególnym wektorom-wierszom tablicy danych), to możemy te nazwy dodać później do utworzonej struktury sD za pomocą rozkazu som_label – por. 7.8. punkt IV.

Procedura som_read_data może mieć dodatkowy argument (,'x') określający braki w danych czyli *missing values*,

D. Pola struktury danych sD

Struktura sD ma następujące pola:

```
sD =
           type: 'som_data'
           data: [150x4 double]
         labels: {150x1 cell}
           name: 'iris.data'
     comp_names: {4x1 cell}
      comp_norm: {4x1 cell}
    label_names: []
```

type – typ struktury. Może być: 'som_data' i 'som_map' ,

data – dane liczbowe i ich wymiar; jest to po prostu wymiar zapamiętanej tablicy danych,

labels – są to etykiety (nazwy) osobników, czyli wierszy tablicy danych, pole to może być (częściowo) niewypełnione lub też zawierać powtarzające się nazwy,

name – nazwa struktury, jeżeli nie zadeklarowano specjalnie, zostaje podstawiona nazwa danych,

comp_names – w pakiecie somtb2 *components* $\equiv$ *variables*, tutaj znajdują się nazwy zmiennych; jeśli użytkownik nazw tych nie podał, to zostają podsatwione wartości domyślne $Var + numer\_zmiennej$,

comp_norm – informacje o normalizacji zmiennych; omawiamy to dalej,

label_names – tego pola tutaj nie omawiamy

E. Normalizacja i denormalizacja zmiennych

Dopuszcza się następujące **możliwości normalizacji**:

'var' – (na $\mu = 0,\ \sigma = 1$), tj. odjąć od każdej obserwacji jej średnią μ i podzielić przez odchylenie standardowe σ,

'range' – tj. na min–max,

'log' – ($x^{new} = ln(x - min(x) + 1)$),

'logistic' – (softmax; $\hat{x} = (x - \bar{x})/\sigma_x$; $x^{tr} = 1/(1 + exp\{-\hat{x}\})$,

'histD' – również 'histC' – histogram equalization.

Normalizacja danych wykonuje się za pomocą rozkazu:
sD = som_normalize(sD,'var'); % zamiast 'var' może być inny sposób, np. 'range', 'log', ...

Funkcja som_normalize może zawierać jeszcze trzeci argument określający które zmienne mają być normalizowane. Np.

sD = som_normalize(sD,'log',[1 3]); – zostaną zlogarytmowane tylko pierwsza i trzecia zmienna.

Przykładowa informacja zawarta w pierwszej komórce pola sD.comp_norm i zawierająca informacje dotyczące normalizacji pierwszej zmiennej:

```
sM.comp_norm{1}
ans =
      type: 'som_norm'
    method: 'var'
    params: [5.8433 0.8281]
    status: 'done'
```

Normalizacja wykonana na strukturze sD może zostać wykonana w ten sam sposób na innej (nowej) macierzy o nazwie Dn. Należy wydać w tym celu rozkaz:
Dn = som_normalize(Dn, sD);

De-normalizacja danych jest wykonywana za pomocą dualnego rozkazu:
sD = som_denormalize(sD);
Zostaną unieważnione wszystkie normalizacje wykazane w polu .comp_norm i przywrócone wartości danych sprzed normalizacji.

Zwróćmy uwagę, że dane zostają 'zdenormalizowane', ale opis normalizacji pozostaje. Aby usunąć opis normalizacji należy użyć przy denormalizacji kwalifikatora 'remove':
sD = som_denormalize(sD,'remove');

7.8 Somtoolbox: Tworzenie mapy – funkcja som_make

I. Postępowanie standardowe – som_make z wartościami domyślnymi

Struktura–mapa sM może być utworzona za pomocą rozkazu (zaleca się, żeby dane zostały najpierw znormalizowane):

sM = som_make(sD);

Procedura som_make wywołana bez dalszych parametrów inicjalizuje i trenuje mapę według wartości domyślnych wbudowanych w procedurę. Zostają automatycznie określone: M - liczba neuronów ($M \approx 5 \times \sqrt{N}$) i $m_1 \times m_2$ - rozmiary mapy (na podstawie stosunku wartości własnych m. kowariancji obliczanych danych).
Potem następuje uczenie wsadowe (training using batch algorithm) w dwóch fazach (Rough training phase... Fine tuning phase...).
Na koniec zostaje obliczony błąd kwantyzacji i błąd reprezentacji topologicznej (opisane wcześniej w sekcji 7.3.5). Obliczenia błędów są wykonywane za pomocą procedury
[q,t] = som_quality(sM,D) D oznacza tu Tablicę danych lub strukturę danych sD

Dla danych *iris* otrzymujemy:
Final quantization error: 0.393 Final topographic error: 0.013

II. Pola struktury map-struct sM

Utworzona struktura sM zawiera następujące pola:

```
sM =
          type: 'som_map'
      codebook: [66x4 double]
         topol: [1x1 struct]
        labels: {66x1 cell}
         neigh: 'gaussian'
          mask: [4x1 double]
     trainhist: [1x3 struct]
          name: 'SOM 29-Dec-2000'
    comp_names: {4x1 cell}
     comp_norm: {4x1 cell}
```

Omówimy teraz krótko te pola.

type – typem struktury jest tu mapa, dokładniej 'som_map',

codebook – zawiera współrzędne wektorów wagowych, nazywanych wektorami kodowymi; jest to tablica wymiaru $M \times d$, przyporządkowanie wektorów kodowych węzłom siatki następuje kolumnami (jak w Fortranie); Przykładowo dla danych *iris* otrzymano:

```
sM.codebook
ans =
   -1.4152    0.0229   -1.3285   -1.3285
   -1.3005    0.0068   -1.2852   -1.2746
      ...       ...       ...       ...
    1.2796    0.4892    1.1476    1.3781
    1.6463    0.5612    1.2880    1.3040
```

topol – zawiera informacje o topologii utworzonej mapy; umieszcza się tu istotne informacje o wymiarach mapy (size), rodzaju siatki (u nas było lattice 'hexa', czyli siatka hexagonalna, mogłoby być lattice 'rect', czyli siatka prostokątna), oraz kształcie mapy (shape 'sheet') oznacza arkusz, mogłoby być: shape 'cyl' czyli cylinder, lub shape 'toroid' czyli mapa na torusie.

Przykładowo dla danych *iris* otrzymano:

```
sM.topol ans =
       type: 'som_topol'
      msize: [11 6]
    lattice: 'hexa'
      shape: 'sheet'
```

labels – może zawierać nazwy wektorów wagowych. Na początku, po utworzeniu mapy, pole to jest puste. Odpowiednie nazwy można nadać za pomocą procedury som_autolabel – patrz niżej.

mask – pole to zawiera tzw. maskę na zmienne: jeśli elementem maski jest wartość zero, to odpowiednia zmienna nie jest uwzględniana w obliczeniach. Dla danych *iris* były brane do obliczeń wszystkie 4 zmienne , wobec czego pole 'mask' zawiera cztery jedynki:

```
sM.mask'
ans =     1     1     1     1
```

component_names – to nazwy zmiennych; przeszły one ze struktury sD. Dla danych *iris* są to:

```
sM.comp_names' ans =     'SepalL'    'SepalW'    'PetalL'    'PetalW'
```

comp_norm – pole to zawiera informacje o normalizacji, takie same jak struktura sD omawiana wcześniej.

III. Postępowanie niestandardowe – funkcja som_make z deklarowanymi wartościami argumentów

Nagłówek funkcji:
function sMap = som_make(D, varargin);
Ogólna postać wywołania funkcji som_make:
sMap = som_make(D, [[argID,] value, ...]);

Dla możliwych argumentów najpierw podaje się identyfikator argumentu, a potem jego wartość. Przykłady:

sMap = som_make(D); % tylko obowiązkowy argument
sMap = som_make(D, 'munits', 20);% mapa o 20 węzłach
sMap = som_make(D, 'munits', 20, 'hexa', 'sheet');
sMap = som_make(D, 'msize', [4 6 7], 'lattice', 'rect');

Jako D może wystąpić (matrix) training data – o wymiarze dlen x dim, lub (struct) data struct.

Dalszymi argumentami mogą być:

```
'init'       *(string) initialization: 'randinit' or 'lininit' (default)
'algorithm'  *(string) training: 'seq' or 'batch' (default) or 'sompak'
'munits'      (scalar) the preferred number of map units
'msize'       (vector) map grid size
'mapsize'    *(string) do you want a 'small', 'normal' or 'big' map
                       Any explicit settings of munits or msize override this.
'lattice'    *(string) map lattice, 'hexa' or 'rect'
'shape'      *(string) map shape, 'sheet', 'cyl' or 'toroid'
'neigh'      *(string) neighborhood function, 'gaussian', 'cutgauss',
                       'ep' or 'bubble'
'topol'      *(struct) topology struct
'som_topol','sTopol' = 'topol'
'mask'        (vector) BMU search mask, size dim x 1
'name'        (string) map name
'comp_names'  (string array | cellstr) component names, size dim x 1
'tracking'    (scalar) how much to report, default = 1
'training'    (string) 'short', 'default', 'long'
              (vector) size 1 x 2, first length of rough training in epochs,
                       and then length of finetuning in epochs
```

IV. Funkcje som_label i som_autolabel

Funkcje som_label

Formalny nagłówek: som_label(sTo, mode, inds, [labels])

Przykłady zastosowń:
sM = som_label(sM, 'add', [1; 10], 'x'); % Dodanie próbkom nr. 1 i 10 nazwy 'x'
sD = som_label(sD, 'clear', 'all'); % usuwa wszystkie nazwy z danych
sD = som_label(sD, 'replace', [1:10]', 'topten');
% zamienia nazwy próbek nr 1–10 na nazwę 'topten'

Ostatnia instrukcja mogłaby być zastąpiona następującymi dwoma:
sD = som_label(sD, 'clear', [1:10]'); sD = som_label(sD, 'add', [1:10]', 'topten');

Jeszcze inne zastosowanie: usunięcie pustych etykiet z wszystkich jednostek mapy:
sD = som_label(sM, 'prune', 'all');

Funkcja som_autolabel

Ogólna postać procedury: som_autolabel(sTo, sFrom, [mode], [inds]);
znaczenie: dokąd wstawić, skąd pobrać, sposób, wskaźniki

```
mode: 'add'   -- po prostu dodać labele, mogą się powtarzać
      'add1'  -- zostaje zapamiętana tylko jedna etykieta
      'freq'  -- dla powtarzających nazw zostaje zapamiętana tylko jedna
                 i jej frekwencja
```

```
'vote'  -- zostaje dodana nazwa o największej frekwencji
           w przyp. losowania: pierwsza wylosowana
```

Wymienione operacje nie zmieniają starych nazw znajdujących się w sTo.
Przyklady stosowania:
sM = som_autolabel(sM,sD) % oznakowanie wg danych
sD = som_autolabel(sD,sM) % oznakowanie wg mapy (codebook vectors)
sM = som_autolabel(sM,sD,'vote',[5]) % etykietuje jednostki mapy
%według nazw występujących w 5-tej kolumnie danych, na zasadzie 'vote'.

som_show_clear % moze się przydać, usuwa z pokazanej mapy wszystkie markery typu
% hits, labels, trajectories, naniesione przez som_show_add
% i mozna zacząć pokazywanie od nowa

7.9 Somtoolbox: Wizualizacja som_show, umat, som_show_add

Standardowe wywołanie: som_show (sM); Wykreśla plansze typu umat i mapy o siatce zapamiętanej w strukturze sM. Przykładowe wykresy są przedstawione na rysunku 7.8 – jednak należy pokreślić, że tutaj dokonano wyboru treści prezentacji przez deklarowanie dodatkowych parametrów funkcji som_show – patrz odpowiednie makro.

Można używać różnych palet (colormap) kolorów: paletą domyślną jest jet. Jeżeli sporządzamy mapy z zamiarem drukowania na drukarce monochromatycznej (czarno-białej), to rozróżnialne odcienie wychodzą przy paletach hot, summer, pink, copper, autumn, gray.

Metoda wizualizacji umat

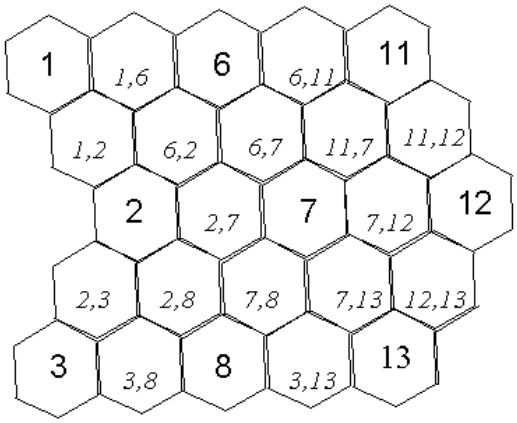
Zilustrujemy tę metodę na przykładzie mapy o siatce 'hexa'. Sama metoda pochodzi od informatyka o nazwisku *Ultsch*. Przypuśćmy, że rozważamy mapę o rozmiarze $M = m_1 \times m_2$. Mapę tę poszerzamy w kierunku wschodnim o $m_2 - 1$ pustych jednostek ulokowanych między oryginalnymi jednostkami mapy (zawierającymi wektory referencyjne). Otrzymaną w ten sposób strukturę poszerzamy jeszcze raz w kierunku południowym, lokując między każdą parą wierszy dodatkowy wiersz zawierający $2m_2 - 1$ pustych jednostek. Ostatecznie otrzymujemy strukturę zawierającą $(2m_1 - 1) \times (2m_2 - 1)$ jednostek. Fragment takiej struktury jest pokazany na rys. 7.7

Tak więc co drugi element macierzy U (U(1:2:size(U(,1), 1:2:size(U,2))) opisuje jednostki mapy zawierające prototypy – por. macro job_grid.m w podsekcji 9.7, fragment konstruujący subplot(2,2,2).

Teraz obliczamy tablicę odległości U$[1 : (2m_1 - 1), 1 : (2m_2 - 1)]$ według zasady:

- Jeśli jednostka zawiera wektor referencyjny $\mathbf{r}_i$, to dla odpowiadającego prototypu $\mathbf{w}_i$ obliczamy medianę odległości tego prototypu od wszystkich jego sąsiadów (sąsiedztwo jest liczone na mapie, tj. dla wektorów referencyjnych, por. np. rys. 7.7). Przypominamy, że na mapie wektory referencyjne są numerowane kolumnami. Obliczona mediana zostaje umieszczona w odpowiednim elemencie tablicy U.
- Jeśli jednostka jest pusta, to odpowiadający jej element tablicy U otrzymuje wartość mediany odległości między parą sąsiednich jednostek zawierających wektory referencyjne (patrz rysunek 7.7).

Zamiast mediany odległości można posługiwać się średnią, min lub max, (parametr mode funkcji umat o wartości mean, min, max odpowiednio). Na wykresie umat wysokie



Rysunek 7.7: Zasada wizualizacji 'umat' na przykładzie fragmentu mapy $m_1 \times m_2 = 5 \times 3$. Oryginalna mapa została poszerzona o dwie dodatkowe jednostki w każdym wierszu mapy; ponadto wstawiono w kierunku południowym – między każde dwa poszerzone wiersze – dodatkowy wiersz zawierający tyle samo (tj. $2m_2 - 1 = 5$) pustych jednostek. {plik hexdist1.ps}

wartości oznaczają granice klasterów; niskie wartości rozłożone obok siebie wskazują na same klastery.

Wykresy typowej mapy o siatce *hexa*

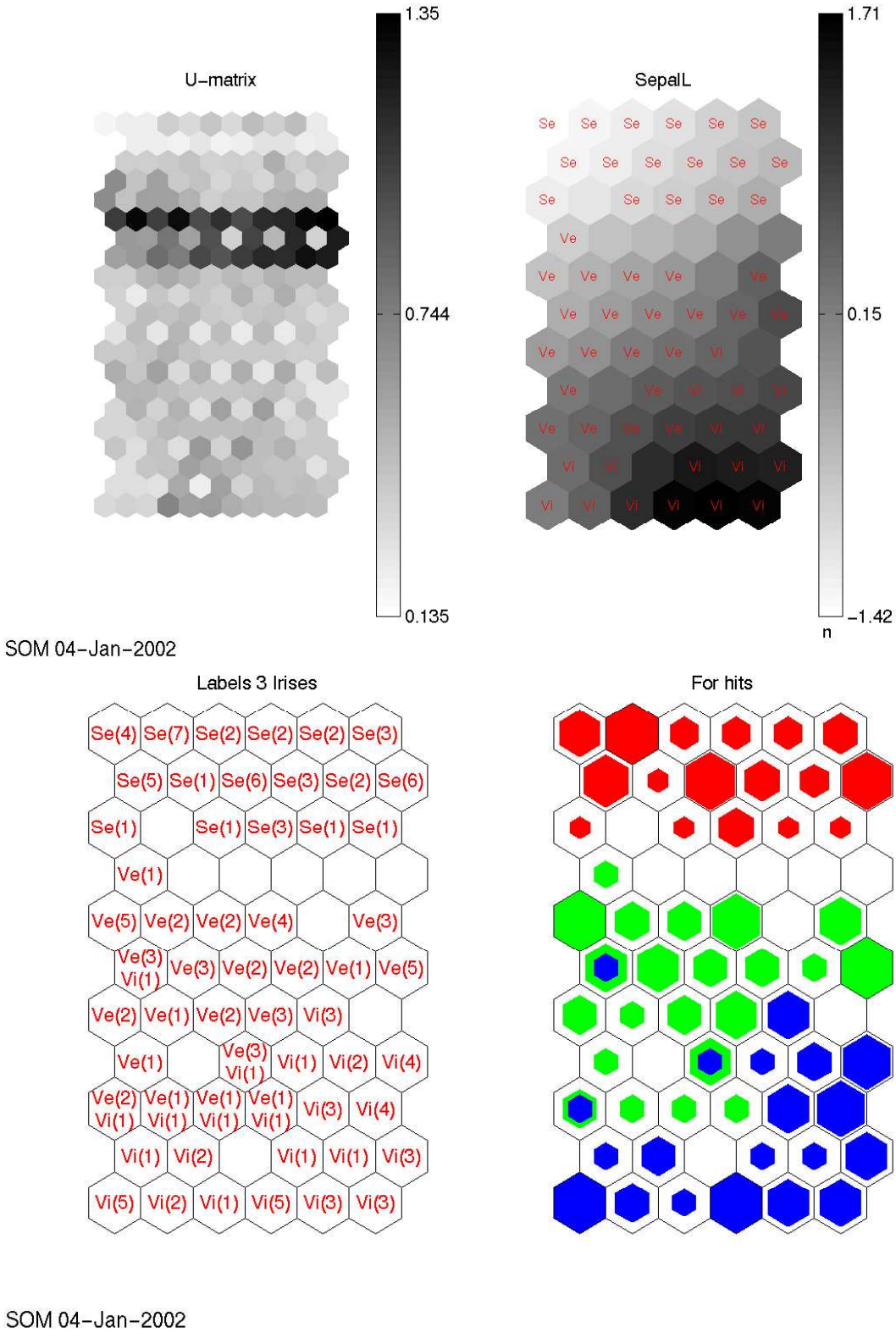
Ten typ wykresu jest pokazany na wykresie 7.8 w subplotach (2,2,2), (2,2,3), (2,2,4).

Na wykresach typu 'mapa' można nanosić różnego rodzaju informacje o wektorach danych które znalazły się w strefie wpływów wektora kodowego stojącego w odpowiedniości z danym heksagonem.

Przykładowo subplot (2,2,3) na wykresie 7.8 pokazuje gatunki i frekwencje irysów należących do danego heksagonu, a subplot (2,2,4) pokazuje kolorem i wielkością gatunek oraz liczbę odpowiednich irysów.

Subplot (2,2,4) na wykresie 7.8 pokazuje tzw. component-plane dla cechy irysa nazywającej się *SepalL*. Tutaj dla każdego heksagonu jest zaznaczona wartość cechy 'SepalL' wektora kodowego z R^d – odpowiadającemu wektorowi referencyjnemu znajdującemu się w środku danego hexagonu.

Wykresy typu component-plane można otrzymać dla każdej lub dla wybranych zmiennych. Pokazują one, jak zmienia się wartość danej cechy, gdy przechodzimy od jednego wektora kodowego do drugiego.



Rysunek 7.8: *Dwa przykłady map Kohonena otrzymane za pomocą pakietu* somtoolbox2. Góra: *Mapa 'umat' po lewej, mapa cechy nr 1 (SepalL) po prawej.* Dół: *Nazwy wektorów kodowych po lewej (opcja 'freq'), liczby trafień (hits) znakowane wielkością heksagonu – po prawej,*

7.10 Somtoolbox: Użyteczne pomocnicze procedury i przykłady

som_show_add – nanosi na wykres – wyprodukowany przez som_show – dodatkowe informacje dotyczące etykiet (labels), obsadzenia jednostek (hits) i trajektorii (trajectories), może dodawać labels, hits, trajectories.

som_autolabel – nadaje etykiety jednostkom, is used to categorize the units (or some units) by giving them names,

som_hits – Hit histograms are actually markers that show the distribution of the best matching units for a given data set.

som_trajectory – show the best matching units for a given data set that is time series (or any ordered series).

Przykłady użycia

```
som_show(sM, 'umat', 'all', 'comp', 1:4, 'empty', 'Labels', 'norm','d');
som_show_add('label',sM, 'subplot',6);
som_show(sM, 'comp',[1 3 2 4], 'umat', {[1 2],'1,2 only'}, 'empty', 'Empty plane');
```

7.11 Somtoolbox: Przykładowe skrypty dla danych *iris*

Podajemy przykładowe skrypty które rysują mapy Kohonena i pokazują na nich różnego rodzaje informacje. Przykładowo posługujemy się danymi *iris*.

Pierwszy skrypt, o nazwie `job_som.m` daje dwie mapy pokazane na rysunku 7.8 w górnym rzędzie. *Pierwsza* z nich, typu 'umat', pokazuje kolorem odległości między wektorami kodowymi w przestrzeni R^d. *Druga* mapa, typu *component plane*, pokazuje intensywnością szarości (paleta '1-gray') wartość pierwszej zmiennej (tj. SepalL) przyjmowanej przez wektory kodowe reprezentujące poszczególne hexagony. Na każdy hexagon naniesiono nazwę tego gatunku irysów, który stanowi większość w danym hexagonie. Nazwy zostały naniesione jako napisy o kolorze czerwonym.

Drugi skrypt konstruuje kolejne dwie mapy – nazwijmy je trzecią i czwartą. Są one pokazane w dolnej części rysunku 7.8.

Trzecia mapa pokazuje detalicznie, ile irysów spośród 3 gatunków oznaczonych umownie etykietami *Se*, *Ve*, *Vi* jest reprezentowanych przez kolejne wektory kodowe.

Czwarta mapa pokazuje to samo, co trzecia, ale na inny sposób. Każdy gatunek irysa jest rysowany odrębnym kolorem (czerwonym, zielonym, niebieskim). Liczba irysów danego gatunku reprezentowanych przez kolejne wektory kodowe jest zaznaczona odpowiedniej wielkości hexagonem. Widać wyraźnie, jak poszczególne gatunki irysa rozlokowały się na mapie.

Ponieważ każdy hexagon na mapie odpowiada pewnemu regionowi Voronoia w przestrzeni cech, łatwo wnioskujemy, że punkty indywidualne irysów należących do różnych gatunków są ulokowane w różnych obszarach przestrzeni R^d.

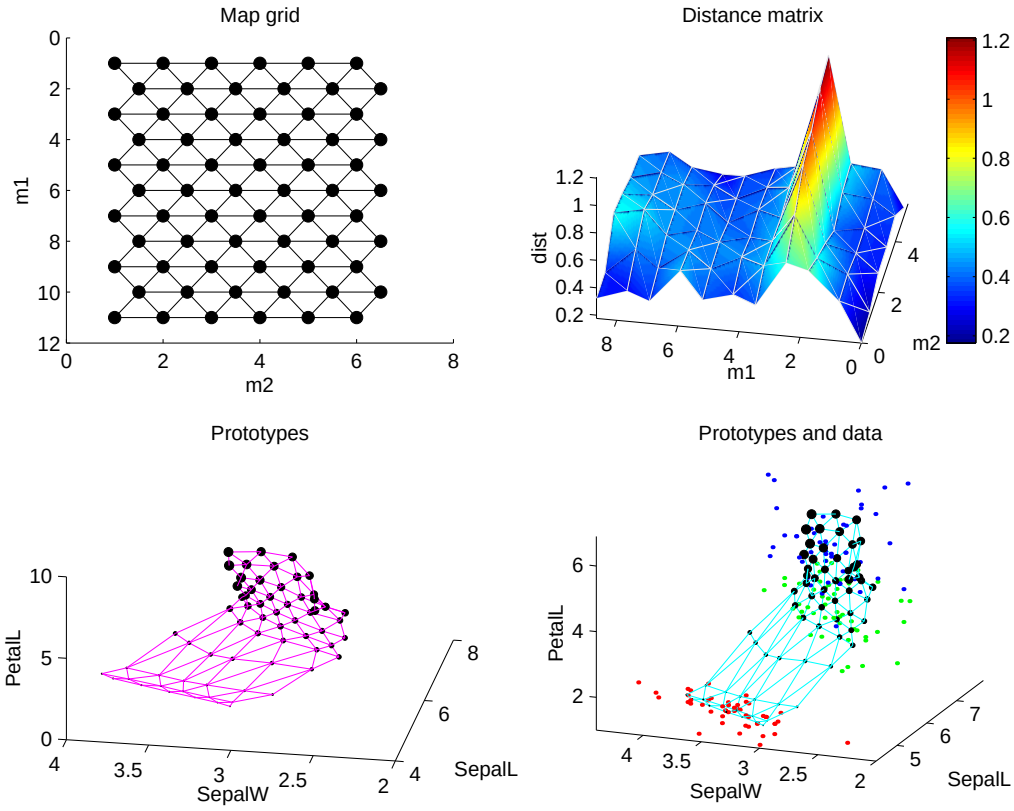
```
 %% job_som1.m
 sD=som_read_data('iris.data'); sD=som_normalize(sD,'var')
     sD=som_label(sD,'replace',[1:50],'Se');
     sD=som_label(sD,'replace',[51:100],'Ve');
     sD=som_label(sD,'replace',[101:150],'Vi');
 sM=som_make(sD)
 sM=som_autolabel(sM,sD,'vote'); % 'freq', 'add', 'add1'
 colormap(1-gray); % bedzie w odcieniach szarosci
 som_show(sM,'umat','all','comp',1);
 som_show_add('label',sM,'TextSize',8,'TextColor', 'r',...
                     'subplot',2);
%% -------------------------------------
 %% job_som2.m
 % Przeczytanie danych, ich normalizacja
 sD=som_read_data('iris.data'); sD=som_normalize(sD,'var')
 % Utworzenie struktury--mapy, nadanie etykiet przez 'freq'
 %    i pokazanie dwoch map na razie bez etykiet
 sD = som_label(sD, 'replace', [1:50]', 'Se');
 sD = som_label(sD, 'replace', [51:100]', 'Ve');
 sD = som_label(sD, 'replace', [101:150]', 'Vi');
 sM=som_make(sD)
 sM=som_autolabel(sM,sD,'freq');    % 'add', 'add1', 'freq'
 som_show(sM,'empty','Labels 3 Irises', 'empty', 'For hits');
 %    Pokazanie etykiet na mapie
 som_show_add('label',sM,'TextSize',10,'TextColor', 'r','subplot',1);
 %    Obliczenie tzw. trafien ("hits"):
 %      ile probek reprezentuja poszczegolne neurony
 %      liczba trafien zostanie pokazana na mapie przez
 %      wielkosc heksagonow, a gatunek irisa odmiennym kolorem
 h1=som_hits(sM,sD.data(1:50,:));  h2=som_hits(sM,sD.data(51:100,:));
 h3=som_hits(sM,sD.data(101:150,:));
 som_show_add('hit',[h1, h2, h3],'MarkerColor',...
   [1 0 0; 0 1 0; 0 0 1], 'subplot', 2);
 % Otrzymujemy plansze z dwoma mapami:  z etykietami i trafieniami
%% -------------------------------------
```

7.12 Somtoolbox: Wizualizacja mapy – procedura som_grid

Procedura pełni funkcję podobną do mesh w MatLabie. Przedstawia graficznie dane dwu- i trójwymiarowe zawarte w strukturach sD i sM, jednocześnie pozwala na swobodne operowanie markerami, kolorami i grubościami linii dostępnymi w Matlabie.

Dalej pokazujemy przykładowy skrypt o nazwie som_grid.m i rysunki otrzymane po wykonaniu tego skryptu. Instrukcje w skrypcie zostały wypisane z modułu demonstracyjnego som_demo2.m

```
%% -----------------------------------
%% job_grid.m    -- som_demo2.m
sD=som_read_data('iris.data');
       % terazskrocenie etykiet probek
 sD = som_label(sD,'replace',[1:50]','se');
 sD = som_label(sD,'replace',[51:100]','ve');
 sD = som_label(sD,'replace',[101:150]','vi');

     %%  Normalizacja danych i utworzenie mapy
     %%
sD = som_normalize(sD,'var');
sM = som_make(sD);
%     - The map grid in the output space.

fh2=figure; subplot(2,2,1)
som_grid(sM,'Linecolor','k');   view(0,-90), title('Map grid')
xlabel('m2'); ylabel('m1');
%     - A surface plot of distance matrix: both color and
%       z-coordinate indicate average distance to neighboring
%       map units. This is closely related to the U-matrix.

subplot(2,2,2)
Co=som_unit_coords(sM); U=som_umat(sM); U=U(1:2:size(U,1),1:2:size(U,2));
som_grid(sM,'Coord',[Co, U(:)],'Surf',U(:),'Marker','none');
view(-80,45), axis tight, title('Distance matrix')
colorbar; xlabel('m2'); zlabel('dist'); ylabel('m1');
%     - The map grid in the output space. Three first components
%       determine the 3D-coordinates of the map unit, and the size
%       of the marker is determined by the fourth component.
%       Note that the values have been denormalized.

subplot(2,2,3)
M = som_denormalize(sM.codebook,sM);
som_grid(sM,'Coord',M(:,1:3),'MarkerSize',M(:,4)*2)
view(-80,45),  title('Prototypes')
xlabel('SepalL'); ylabel('SepalW'); zlabel('PetalL');
%     - Map grid as above, but the original data has been plotted
%       also: coordinates show the values of three first components
%       and color indicates the species of each sample.  Fourth
%       component is not shown.

subplot(2,2,4)
som_grid(sM,'Coord',M(:,1:3),'MarkerSize',M(:,4)*2), hold on
D = som_denormalize(sD.data,sD);
plot3(D(1:50,1),D(1:50,2),D(1:50,3),'r.',...
      D(51:100,1),D(51:100,2),D(51:100,3),'g.',...
      D(101:150,1),D(101:150,2),D(101:150,3),'b.')
view(-72,64), axis tight, title('Prototypes and data')
xlabel('SepalL'); ylabel('SepalW'); zlabel('PetalL');
%%
%%-------------------------------------------------
```



Rysunek 7.9: *Fragmenty rysunków wykonywanych przez moduł* som-demo2 *pakietu* somtoolbox. ***Góra:*** *Ułożenie wektorów referencyjnych na mapie, odległości między prototypami w* R^d*, ułożonymi wg. odpowiadających im punktom referencyjnym na mapie, oraz* ***Dół:*** *Wykresy trzech pierwszych cech wektorów kodowych oraz danych iris. Czwarta cecha wektorów kodowych jest pokazana na subplocie (2,2,3) wielkością markera.*

7.13 Grupowanie prototypów na zasadzie k-means

Mamy tu trzy funkcje:

`[codes,clusters,err] = som_kmeans(method, D, k, [epochs], [verbose]);` gdzie
`codes` – wyznaczone centra, `method`: 'batch' lub 'seq'

`[c, p, err, ind] = kmeans_clusters(sD, [n_max], [c_max], [verbose])`, lub

`[c, p, err, ind] = kmeans_clusters(sD);`, sama wybiera odpowiednią liczbę clustrów na podstawie wskaźnika Daviesa-Bouldina opisanego np. w [7, 8].

`color = som_kmeanscolor(sM, C, [initRGB], [contrast])`
C: maximum nb of clusters

Literatura

[1] T. Kohonen, *Self-organizing Maps*. Springer, Berlin - Heidelberg, 1995. Third extended Edition: Springer, London 2001.

[2] S. Osowski, *Sieci neuronowe w ujęciu algorytmicznym*. WNT W-wa 1996.

[3] J. Vesanto, SOM-based data visualization methods. *Intelligent Data Analysis* 1999, 111–126.

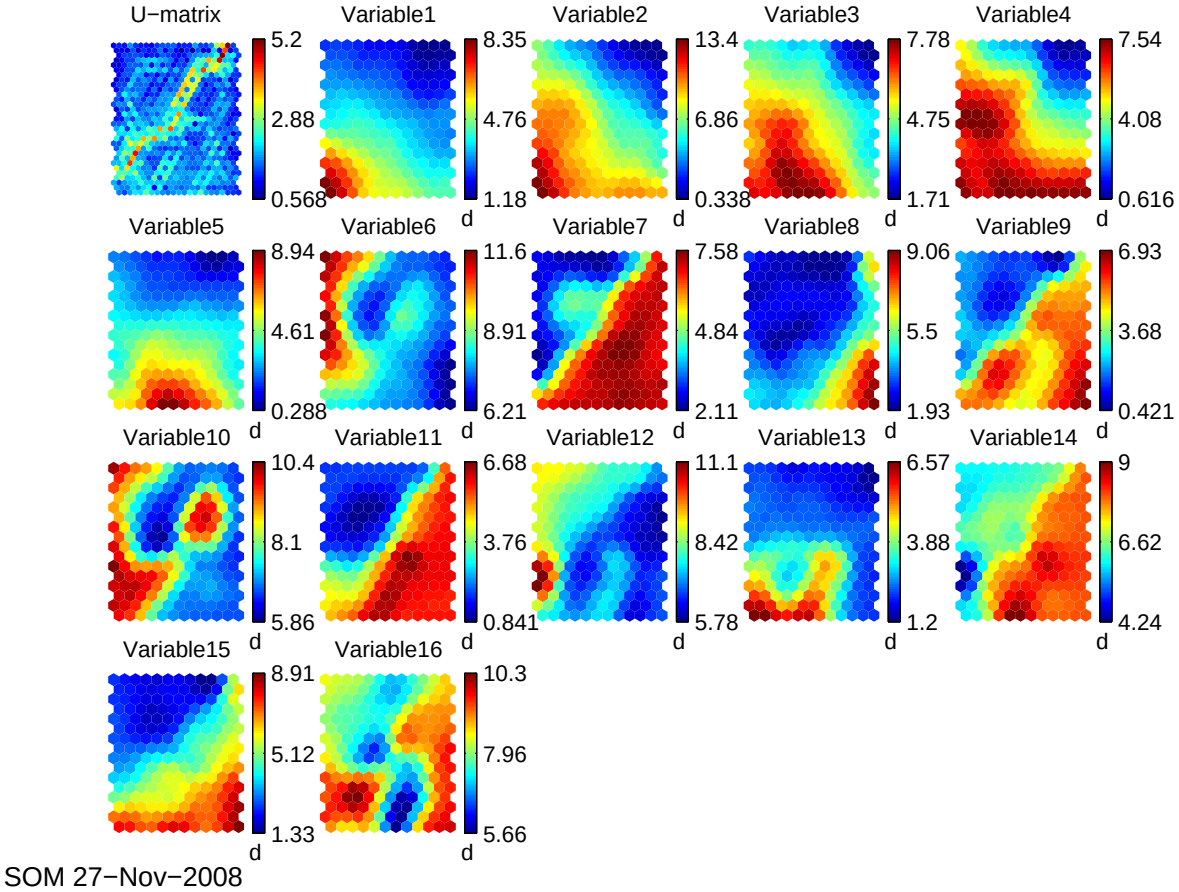
[4] J. Vesanto, J. Himberg, E. Alhoniemi, J. Parhankangas, *SOM Toolbox for Matlab 5*. Som Toolbox team, Helsinki University of Technology, Finland, Libella Oy, Espoo 2000, 1–54. http://www.cis.hut.fi/projects/somtoolbox/ somtoolbox2_Mar17_2005.zip.

[5] Raul Rojas, *Neural Networks – A systematic Introduction*. Springer 1996. Chapter 15. Kohonen networks, 391–412.

[6] Ewa Skubalska-Rafajłowicz, Samoorganizujące sieci neuronowe. W: M. Nałęcz (red), *Biocybernetyka i Inżynieria Biomedyczna 2000. Tom 6: Sieci neuronowe,* str. 187–188.

[7] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data*. Prentice Hall, 1988.

[8] R.C. Dubes and A.K. Jain, Validity studies in clustering methods. *Pattern Recognition*, **11** (1971), 235–254.



7.14 Dodatek: Konstrukcja mapy Kohonena dla liter A i B

```
%%.....................................................
% makroM, Mapa Kohonena dla dwoch literek A, B, dane z UCI

lettersUci % wczytanie danych 26 literkowych

% przygotowanie danych
keep2 A B letters
nrAB=find(or(letters(:,1)==1, letters(:,1)==2));
len=length(nrAB)                      %1555
dane=letters(nrAB,2:end); size(dane) %1555  16

%Mapa Kohonena
sD=som_data_struct(dane)
sM=som_make(sD)
  \% map size [16, 12]   batch
  \% Final quantization error: 2.786, Final topographic error:  0.055
%    sM =
%            type: 'som_map'
%       codebook: [192x16 double]
%          topol: [1x1 struct]
%         labels: {192x1 cell}
%          neigh: 'gaussian'
%           mask: [16x1 double]
%      trainhist: [1x3 struct]
%           name: 'SOM 27-Nov-2008'
%     comp_names: {16x1 cell}
%      comp_norm: {16x1 cell}
som_show(sM)
print -depsc mapaAB

% porownaj wektory kodowe z 4 naroznikow mapy i ze srodka
%.....................................................
```

Spis treści