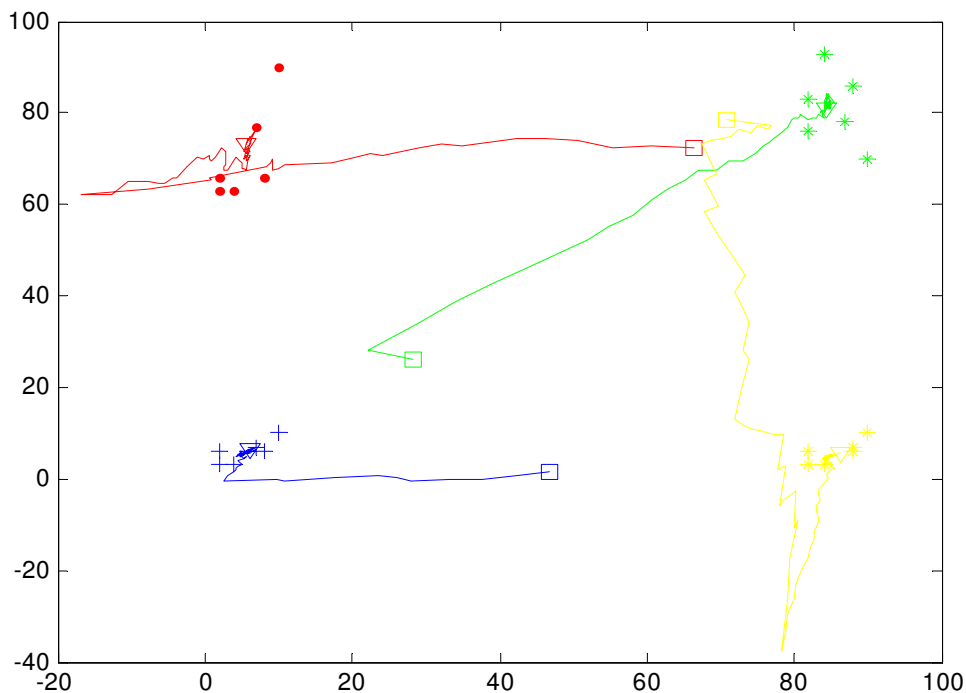


| | | | | |
|---|-----------|-------------|---------------------|-----------|
| Uniwersytet Zielonogórski | Wykonali: | Grupa: | Nr ćwiczenia: 10 | Ocena: |
| Rozpoznawanie obrazów - Laboratorium | | | | |
| Temat ćwiczenia: Algorytm LVQ ver1, ver2.1, ver3 | | Prowadzący: | Data wyk. | Data odd. |

Algorytm LVQ (Learned Vector Quantization - Adaptacyjne kwantowanie wektorowe) opracowany został przez Tuevo Kohonena (Fausett, 1994; Kohonen, 1990), który stworzył też samoorganizującą się mapę cech (nazywaną tu siecią Kohonena). LVQ jest nadzorowaną wersją algorytmu uczącego Kohonena. Standardowy algorytm Kohonena iteracyjnie dopasowuje położenia wektorów wzorcowych, przechowywanych w warstwie radialnej sieci Kohonena, rozpatrując jedynie pozycje istniejących wektorów i dane uczące. W istocie, algorytm próbuje przemieścić wektory wzorcowe na pozycje odpowiadające centrom skupień występujących w danych. Nie są przy tym brane pod uwagę etykiety klas przypadków uczących. Dla osiągnięcia najlepszej jakości klasyfikacji pożądane jest, aby wektory wzorcowe rozmieszczone były, w pewnym stopniu w zakresie klas. Tak, by reprezentowały naturalne skupienia wewnątrz każdej klasy. Wektor zlokalizowany na granicy klas, w jednakowej odległości od przypadków z jednej i drugiej klasy, jest nieprzydatny do klasyfikowania. Natomiast wektory znajdujące się dokładnie wewnątrz granic klas działają bardzo dobrze.

Istnieje kilka wariantów algorytmu LVQ. Podstawowa wersja, LVQ1, jest bardzo podobna do algorytmu uczącego Kohonena, gdzie wyszukiwany jest wektor najbliższy danego przypadku uczącego i odpowiednio modyfikowane jest jego położenie. O ile jednak algorytm Kohonena przesuwa wektor w stronę przypadku uczącego, to LVQ1 sprawdza zgodność klas wektora i przypadku. Jeśli są zgodne, wektor wzorcowy przesuwany jest w stronę przypadku uczącego a w przeciwnym przypadku odsuwany jest od niego. Działanie algorytmu przedstawia rys 1.



Rys. 1. Wynik działania algorytmu LVQ ver.1

Skrypt, wykorzystany do wygenerowania rysunku pierwszego zamieszczamy poniżej:

```
m=[2 3;4 3;2 6; 8 6; 7 7; 10 10;2 63;4 63;2 66;8 66; 7 77; 10 90;82 83;84 93;82 76;88 86;87 78;90 70;82
3;84 3;82 6;88 6;88 7;90 10];
c=[1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4];
p=[];

plot(m(1:6,1)',m(1:6,2)', '+b');
hold on
plot(m(7:12,1)',m(7:12,2)', '.r');
plot(m(13:18,1)',m(13:18,2)', '*g');
plot(m(19:24,1)',m(19:24,2)', '*y');

c1=rand(1,2)*100;
c2=rand(1,2)*100;
c3=rand(1,2)*100;
c4=rand(1,2)*100;
c1_h=c1;
c2_h=c2;
c3_h=c3;
c4_h=c4;
zc=[1 2 3 4];
iteracje=1;

    plot(c1(1),c1(2), 'vb');
    plot(c2(1),c2(2), 'vr');
    plot(c3(1),c3(2), 'vg');
    plot(c4(1),c4(2), 'vy');

while(iteracje<400)

    iteracje=iteracje+1;

    d=[0 0 0 0 ];

    %for i=1:length(c),
    i=round(rand(1)*23+1);
    d(1)=sum(abs(m(i,:)-c1));
    d(2)=sum(abs(m(i,:)-c2));
    d(3)=sum(abs(m(i,:)-c3));
    d(4)=sum(abs(m(i,:)-c4));

    [mw,index]=min(d);

    if index==c(i),
        if index==1,
            c1=c1+0.1*(m(i,:)-c1);
            c1_h=[c1_h;c1];
        elseif index==2,
            c2=c2+0.1*(m(i,:)-c2);
            c2_h=[c2_h;c2];
        elseif index==3,
            c3=c3+0.1*(m(i,:)-c3);
            c3_h=[c3_h;c3];
        elseif index==4,
            c4=c4+0.1*(m(i,:)-c4);
            c4_h=[c4_h;c4];
        end
    else
        if index==1,
            c1=c1-0.1*(m(i,:)-c1);
            c1_h=[c1_h;c1];
        elseif index==2,
            c2=c2-0.1*(m(i,:)-c2);
            c2_h=[c2_h;c2];

        elseif index==3,
            c3=c3-0.1*(m(i,:)-c3);
            c3_h=[c3_h;c3];
        elseif index==4,
            c4=c4-0.1*(m(i,:)-c4);
            c4_h=[c4_h;c4];
        end
    end
end
end
%end
```

```
end
figure,
size(m)
plot(m(1:6,1),m(1:6,2),'+b');
hold on
plot(m(7:12,1),m(7:12,2),' .r');
plot(m(13:18,1),m(13:18,2),'*g');
plot(m(19:24,1),m(19:24,2),'*y');

plot(c1_h(1,1),c1_h(1,2),'sb');
plot(c2_h(1,1),c2_h(1,2),'sr');
plot(c3_h(1,1),c3_h(1,2),'sg');
plot(c4_h(1,1),c4_h(1,2),'sy');

plot(c1_h(:,1),c1_h(:,2),'b');
plot(c2_h(:,1),c2_h(:,2),'r');5
plot(c3_h(:,1),c3_h(:,2),'g');
plot(c4_h(:,1),c4_h(:,2),'y');

plot(c1(1),c1(2),'vb');
plot(c2(1),c2(2),'vr');
plot(c3(1),c3(2),'vg');
plot(c4(1),c4(2),'vy');
```

Bardziej wyszukane są algorytmy LVQ2.1 i LVQ3, uwzględniają one więcej informacji. Wyszukują dwa wektory najbliższe danemu przypadkowi uczącemu. Jeżeli jeden z wektorów ma klasę zgodną z przypadkiem, a drugi niezgodną, to zgodny wektor przesuwany jest do przypadku a niezgodny odsuwany jest od przypadku uczącego. LVQ3 przybliża oba wektory do przypadku jeżeli oba mają zgodne z przypadkiem klasy. W obydwu algorytmach, LVQ2.1 i LVQ3, zasadą jest odsuwanie wektorów wzorcowych od miejsc zagrożonych błędnymi klasyfikacjami. Szczegóły techniczne. Podstawową regułą zmiany wag jest:

$$\omega_t = \omega_{t-1} - \eta_t (x - \omega_{t-1})$$

przy zgodności klasy wektora wzorcowego i przypadku uczącego, a

$$\omega_t = \omega_{t-1} - \eta_t (x + \omega_{t-1})$$

przy niezgodności.

x jest przypadkiem uczącym, η jest współczynnikiem uczenia.

W algorytmie LVQ2.1, korygowane są dwa najbliższe wektory wzorcowe, ale tylko wtedy, gdy jeden jest zgodnej klasy, drugi niezgodnej, a oba są w "podobnej" odległości od przypadku uczącego. Jakie odległości są "podobne" określa parametr ϵ , i poniższe wzory:

$$\min \left(\frac{d_1}{d}, \frac{d_2}{d_1} \right) > 1 - \epsilon$$

$$\max \left(\frac{d_1}{d}, \frac{d_2}{d_1} \right) < 1 + \epsilon$$

W algorytmie LVQ3 nieco inny jest wzór określający "podobne" odległości:

$$\min \left(\frac{d_1}{d}, \frac{d_2}{d_1} \right) > (1 - \epsilon)(1 + \epsilon)$$

Ponadto, w LVQ3, w przypadku gdy oba najbliższe wektory wzorcowe są tej samej klasy co przypadek uczący, to oba są przysuwane do przypadku, z aktualnym, dla danej epoki współczynnikiem uczenia pomnożonym przez współczynnik beta, którego wartość określana jest przez użytkownika.