Kamil Witek

**Sprawozdanie – scenariusz 5**

Temat ćwiczenia: Budowa i działanie sieci Kohonena dla WTA.

1. Cel ćwiczenia

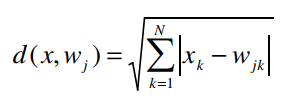
Celem ćwiczenia jest poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły WTA do odwzorowywania istotnych cech kwiatów.

1. Realizacja ćwiczenia

Sieć Kohonena nazywana jest inaczej mapą samoorganizującą. Bazuje na algorytmie uczenia bez nauczyciela, wagi na początku są losowane. Charakterystyczne jest tutaj uczenie konkurencyjne tzn. neurony konkurują ze sobą o prawo do reprezentacji danych wejściowych.

Metoda WTA (Winner Takes All – „Zwycięzca bierze wszystko”) oznacza, że tylko jeden element wyjściowy może znajdować się w stanie aktywnym, którego nazywamy zwycięzcą. Zwycięski neuron osiąga wartość minimalną wektora wag z wartością wektora wejściowego i spełnia relacje:

Gdzie d(x, w) oznacza odległość w sensie wybranej metryki między wektorem ***x*** i wektorem ***w*** a ***n*** to ilość neuronów. Miara według normy L1 (Manhattan):

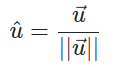


W strategii WTA zmiana wag dotyczy tylko neuronu zwycięzcy wg zależności:

Neurony przegrywające konkurencję nie zmieniają swoich wag.

Dane wejściowe tworzą „bazę wiedzy” na podstawie której sieć uczy się i podejmuje decyzje. Ważną rolę odgrywa nadmiarowość danych uczących, aby minimalizować ryzyko błędów.

Do uczenia sieci użyłem zestawu danych istotnych cech kwiatów zaczerpniętych z Wikipedii. Były to wektory złożone z czterech składowych, które następnie poddałem procesowi normalizacji. Normalizacja polega na podzieleniu każdej ze składowej wektora przez długość tego wektora (jak w poprzednim laboratorium):



Nadmiarowa ilość neuronów wynika z inicjalizacji wag losowo – przez co część neuronów może znaleźć się w strefie, gdzie nie ma danych lub ich liczba jest znikoma. Neurony te automatycznie zostają wykluczone z sieci, ponieważ nie mają szans na wygraną, nazywamy je neuronami martwymi.

1. Wyniki

Każdy kwiat to wektor składający się z czterech składowych. Do nauki wykorzystałem po 15 kwiatów z każdego gatunku w sumie 45 kwiatów. Do testowania wybrałem po 5 kwiatów dla każdego gatunku. Uczenie przeprowadziłem 10-krotnie dla 3 różnych współczynników uczenia: 0.1, 0.01, 0.001. Wykorzystana sieć składała się z 190 neutronów.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LP | Współczynnik uczenia | 0,1 | 0,01 | 0,001 |
| 1 | % Poprawności testowania | 100,00% | 100,00% | 86,66% |
|  | Liczba epok | 11 | 5 | 9 |
| 2 | % Poprawności testowania | 100,00% | 100,00% | 100,00% |
|  | Liczba epok | 5 | 26 | 221 |
| 3 | % Poprawności testowania | 100,00% | 100,00% | 100,00% |
|  | Liczba epok | 1 | 31 | 190 |
| 4 | % Poprawności testowania | 100,00% | 100,00% | 100,00% |
|  | Liczba epok | 5 | 16 | 680 |
| 5 | % Poprawności testowania | 100,00% | 93,33% | 93,33% |
|  | Liczba epok | 7 | 19 | 16 |
| 6 | % Poprawności testowania | 93,33% | 100,00% | 93,33% |
|  | Liczba epok | 1 | 21 | 802 |
| 7 | % Poprawności testowania | 100,00% | 100,00% | 100,00% |
|  | Liczba epok | 4 | 46 | 177 |
| 8 | % Poprawności testowania | 100,00% | 100,00% | 100% |
|  | Liczba epok | 2 | 65 | 29 |
| 9 | % Poprawności testowania | 100,00% | 100,00% | 100,00% |
|  | Liczba epok | 1 | 29 | 51 |
| 10 | % Poprawności testowania | 100,00% | 100,00% | 100,00% |
|  | Liczba epok | 2 | 16 | 496 |

1. Analiza wyników

Jak wynika z powyższych wykresów zdecydowanie widać, że im mniejszy współczynnik nauki tym więcej epok do nauczenia się sieć potrzebowała. Występują dość spore różnice pomiędzy minimalną a maksymalną wartością ilości epok potrzebnych do nauki. Zdecydowanie najlepszym testowanym współczynnikiem uczenia okazał był 0,1, ponieważ uzyskał on procent poprawności danych rzędu 99,33% tak jak współczynnik 0,01 (w obu przypadkach sieć pomyliła się tylko raz), lecz zrobił to w mniejszej ilości epok.

1. Wnioski

Na podstawie powyższych wyników można wnioskować, iż najlepsze wyniki można uzyskać stosując wyższy współczynnik uczenia, ponieważ sieć ze współczynnikiem uczenia równym 0.1 uzyskała najwyższą skuteczność w najmniejszej liczbie epok. Jest to prawdopodobnie spowodowane tym, że większy współczynnik uczenia oznacza szybsze zbliżanie się do siebie wektorów po modyfikacji wag zwycięskiego neuronu. Jednak oczywiście zbyt duży współczynnik uczenia może negatywnie wpłynąć na wyniki.

Bardzo ważnym etapem nauki i testowania jest normalizacja danych wejściowych. Dzięki niej wektory wag neuronów są umieszczane w otoczeniu danych uczących od samego początku. Normalizacja jest niezbędna, ponieważ bez niej wygrywałby zawsze ten sam neuron.

Sieć Kohonena bardzo dobrze poradziła sobie z zadanym problemem, gdzie jak widać poprawność wyników jest na bardzo wysokim poziomie, we wszystkich testach sieć zawsze osiągnęła poprawność powyżej 86,66%.

1. Listing kodu

**public class** Flower {  
  
 *//rozmiar tablicy to: [3][39][4]* **public static double**[][][] *flowerLearn* = {  
 {  
 { 0.809246635, 0.5446852351, 0.217874094, 0.0311248706 },  
 { 0.8281328734, 0.5070201266, 0.2366093924, 0.0338013418 },  
 { 0.8053330754, 0.5483118811, 0.2227517017, 0.0342694926 },  
 { 0.8000302475, 0.5391508189, 0.2608794285, 0.0347839238 },  
 { 0.7904706124, 0.5691388409, 0.2213317715, 0.0474282367 },  
 { 0.7841749863, 0.5663486012, 0.2468699031, 0.058087036 },  
 { 0.7801093557, 0.5766025673, 0.2374245865, 0.0508766971 },  
 { 0.8021849185, 0.5454857446, 0.2406554756, 0.0320873967 },  
 { 0.8064236562, 0.5315065006, 0.2565893451, 0.0366556207 },  
 { 0.81803119, 0.5175299366, 0.2504177112, 0.0166945141 },  
 { 0.8037351881, 0.5507074437, 0.2232597745, 0.0297679699 },  
 { 0.7869910029, 0.5574519604, 0.2623303343, 0.0327912918 },  
 { 0.8230721776, 0.514420111, 0.2400627185, 0.017147337 },  
 { 0.802512599, 0.559892511, 0.2052939207, 0.0186630837 },  
 { 0.8112086464, 0.5594542389, 0.1678362717, 0.0279727119 },  
 { 0.7738111103, 0.5973278746, 0.2036345027, 0.0543025341 },  
 { 0.794289441, 0.5736534852, 0.1912178284, 0.0588362549 },  
 { 0.8032741237, 0.5512665555, 0.2205066222, 0.047251419 },  
 { 0.806828203, 0.5378854687, 0.2406329728, 0.0424646423 },  
 { 0.7796488324, 0.5809148163, 0.2293084801, 0.045861696 },  
 { 0.8173378965, 0.5146201571, 0.2573100785, 0.0302717739 },  
 { 0.7859185787, 0.5701762238, 0.2311525231, 0.0616406728 },  
 { 0.775770746, 0.6071249316, 0.1686458143, 0.0337291629 },  
 { 0.8059779151, 0.5215151215, 0.268659305, 0.0790174427 },  
 { 0.7761140001, 0.5497474167, 0.3072117917, 0.0323380833 },  
 { 0.8264745061, 0.4958847037, 0.264471842, 0.0330589802 },  
 { 0.7977820578, 0.5424917993, 0.2552902585, 0.0638225646 },  
 { 0.806419649, 0.5427824561, 0.2326210526, 0.0310161403 },  
 { 0.8160942667, 0.5336000975, 0.2197176872, 0.031388241 },  
 { 0.7952406381, 0.5414404345, 0.2707202172, 0.0338400272 },  
 { 0.8084658442, 0.5221341911, 0.2694886147, 0.0336860768 },  
 { 0.8222502813, 0.5177131401, 0.2284028559, 0.0609074282 },  
 { 0.7657831085, 0.6037905278, 0.2208989736, 0.0147265982 },  
 { 0.7786744728, 0.5946241429, 0.1982080476, 0.0283154354 },  
 { 0.8176894181, 0.5173137135, 0.2503130872, 0.0333750783 },  
 { 0.8251229525, 0.5280786896, 0.1980295086, 0.0330049181 },  
 { 0.826997544, 0.5262711644, 0.1954721468, 0.030072638 },  
 { 0.7852322109, 0.5769052978, 0.2243520603, 0.0160251472 },  
 { 0.8021241325, 0.5469028176, 0.236991221, 0.0364601878 }  
 }, *//setosa* {  
 { 0.7670110293, 0.3506336134, 0.5149931197, 0.1534022059 },  
 { 0.7454975664, 0.3727487832, 0.5241779763, 0.1747259921 },  
 { 0.7551928518, 0.3392895421, 0.536296373, 0.1641723591 },  
 { 0.753849162, 0.3152460132, 0.548253936, 0.1781825292 },  
 { 0.7581753966, 0.3265986324, 0.536554896, 0.1749635531 },  
 { 0.722329618, 0.3548285843, 0.5702602248, 0.1647418427 },  
 { 0.7263484574, 0.3804682396, 0.5418790079, 0.1844694495 },  
 { 0.7591654715, 0.3718361493, 0.5112747053, 0.1549317289 },  
 { 0.7630185276, 0.3352657167, 0.531800792, 0.1502915282 },  
 { 0.7246023349, 0.3762358277, 0.5434517511, 0.195085244 },  
 { 0.7692307692, 0.3076923077, 0.5384615385, 0.1538461538 },  
 { 0.7392346163, 0.3758820083, 0.5262348116, 0.1879410041 },  
 { 0.7889275246, 0.2892734257, 0.525951683, 0.1314879208 },  
 { 0.73081412, 0.347436221, 0.5630862892, 0.1677278308 },  
 { 0.7591170716, 0.3931141978, 0.4880038317, 0.1762236059 },  
 { 0.7694544447, 0.3560162356, 0.5053133667, 0.1607815258 },  
 { 0.7063189182, 0.3783851348, 0.5675777022, 0.1891925674 },  
 { 0.756764973, 0.3522871426, 0.5349545499, 0.1304767195 },  
 { 0.7644423782, 0.2712537471, 0.55483721, 0.1849457367 },  
 { 0.7618518794, 0.3401124462, 0.530575416, 0.1496494763 },  
 { 0.6985796007, 0.3788906309, 0.5683359464, 0.2131259799 },  
 { 0.7701185383, 0.353497034, 0.5049957628, 0.1641236229 },  
 { 0.7414330662, 0.2942194707, 0.5766701626, 0.1765316824 },  
 { 0.7365989486, 0.3381109928, 0.5675434522, 0.1449047112 },  
 { 0.7674169846, 0.3477358211, 0.5156082865, 0.155881575 },  
 { 0.7678572553, 0.3490260251, 0.5119048369, 0.1628788117 },  
 { 0.7646726946, 0.3148652272, 0.5397689609, 0.1574326136 },  
 { 0.7408857634, 0.331739894, 0.5528998234, 0.18798594 },  
 { 0.7335094873, 0.3545295855, 0.5501321155, 0.1833773718 },  
 { 0.7866747377, 0.3588340909, 0.4830458915, 0.1380131119 },  
 { 0.7652185485, 0.3339135485, 0.5286964517, 0.1530437097 },  
 { 0.7724292478, 0.3370600354, 0.5196342212, 0.1404416814 },  
 { 0.7643498123, 0.3558180161, 0.5139593566, 0.1581413405 },  
 { 0.7077952503, 0.3185078626, 0.6016259627, 0.1887454001 },  
 { 0.6933340942, 0.3851856079, 0.5777784118, 0.1925928039 },  
 { 0.7152493551, 0.4053079679, 0.5364370163, 0.1907331613 },  
 { 0.7545734059, 0.3491309788, 0.5293276131, 0.1689343446 },  
 { 0.7753002086, 0.2830461079, 0.5414795108, 0.1599825827 },  
 { 0.7299244279, 0.3910309435, 0.5344089561, 0.1694467422 }  
 }, *//versicolor* {  
 { 0.653877471, 0.3425072467, 0.6227404486, 0.2594751869 },  
 { 0.690525124, 0.3214513508, 0.6071858849, 0.2262065061 },  
 { 0.7149140499, 0.3020763591, 0.5940835063, 0.2114534514 },  
 { 0.6927679616, 0.3188931887, 0.6157937436, 0.1979337033 },  
 { 0.6861902182, 0.3167031776, 0.61229281, 0.2322489969 },  
 { 0.7095370786, 0.2800804257, 0.6161769366, 0.196056298 },  
 { 0.6705411756, 0.3421128447, 0.6158031204, 0.2326367344 },  
 { 0.7136655737, 0.2835109813, 0.6159031663, 0.1759723332 },  
 { 0.714141252, 0.2664706164, 0.6182118301, 0.1918588438 },  
 { 0.6919878754, 0.3459939377, 0.5862675055, 0.2402735678 },  
 { 0.7156264473, 0.3523084048, 0.5614915202, 0.220192753 },  
 { 0.7157654645, 0.3019635553, 0.5927432753, 0.2124928723 },  
 { 0.7171814812, 0.3164035946, 0.5800732569, 0.2214825163 },  
 { 0.6925517954, 0.3037507875, 0.6075015749, 0.24300063 },  
 { 0.6776792359, 0.3271554932, 0.5958903626, 0.2804189942 },  
 { 0.6958988737, 0.3479494368, 0.5762912548, 0.2500886577 },  
 { 0.7061047399, 0.3258944953, 0.5974732415, 0.1955366972 },  
 { 0.6929909912, 0.3419955541, 0.6029921612, 0.1979974261 },  
 { 0.7060061789, 0.2383916968, 0.6326548876, 0.2108849625 },  
 { 0.727125848, 0.2666128109, 0.6059382067, 0.181781462 },  
 { 0.7055893432, 0.3272298403, 0.5828781531, 0.2351964477 },  
 { 0.6830792286, 0.3415396143, 0.597694325, 0.2439568674 },  
 { 0.7148654283, 0.2599510648, 0.6220257623, 0.185679332 },  
 { 0.7312246431, 0.3133819899, 0.568730278, 0.2089213266 },  
 { 0.6959560109, 0.3427843039, 0.5920819794, 0.2181354661 },  
 { 0.7152945332, 0.3179086814, 0.5960787777, 0.1788236333 },  
 { 0.7278519544, 0.3287073342, 0.5634982873, 0.2113118577 },  
 { 0.7117121386, 0.3500223633, 0.5717031933, 0.210013418 },  
 { 0.6959400158, 0.3044737569, 0.6089475138, 0.2283553177 },  
 { 0.7308985537, 0.304541064, 0.5887793905, 0.1624219008 },  
 { 0.7276615933, 0.2753314137, 0.5998291512, 0.1868320307 },  
 { 0.7157899884, 0.344304045, 0.5798804969, 0.1812126553 },  
 { 0.694177465, 0.3037026409, 0.6074052819, 0.2386235036 },  
 { 0.7236600468, 0.3216266875, 0.5858200379, 0.1723000111 },  
 { 0.6938541359, 0.2957411071, 0.6369808461, 0.1592452115 },  
 { 0.6701748441, 0.3616816619, 0.5957109725, 0.2553047025 },  
 { 0.6980479904, 0.3381169954, 0.5998849918, 0.1963259973 },  
 { 0.7106690545, 0.3553345273, 0.5685352436, 0.2132007164 },  
 { 0.7241525806, 0.325343913, 0.5667281066, 0.2203942637 }  
 } *//virginica* };  
  
 *//rozmiar tablicy to: [3][10][4]* **public static double**[][][] *flowerTest* = {  
 {  
 { 0.8077956849, 0.5385304566, 0.2375869661, 0.0316782622 },  
 { 0.8003330078, 0.5602331055, 0.208086582, 0.0480199805 },  
 { 0.8609385733, 0.4400352708, 0.2487155878, 0.0573959049 },  
 { 0.7860903755, 0.5717020913, 0.2322539746, 0.0357313807 },  
 { 0.788894791, 0.5522263537, 0.2524463331, 0.0946673749 },  
 { 0.766938972, 0.5714447242, 0.2857223621, 0.0601520762 },  
 { 0.8221058465, 0.5138161541, 0.2397808719, 0.0513816154 },  
 { 0.7772909267, 0.5791579454, 0.243855977, 0.0304819971 },  
 { 0.7959478212, 0.5537028322, 0.2422449891, 0.034606427 },  
 { 0.7983702483, 0.5573528148, 0.2259538439, 0.0301271792 }  
 }, *//setosa* {  
 { 0.747141937, 0.3396099714, 0.5433759542, 0.1765971851 },  
 { 0.7326039145, 0.3602970072, 0.552455411, 0.1681386033 },  
 { 0.7626299404, 0.341868594, 0.525951683, 0.1577855049 },  
 { 0.7698687947, 0.3541396456, 0.5081134045, 0.1539737589 },  
 { 0.7354428354, 0.3545885099, 0.5515821266, 0.1707278011 },  
 { 0.7323961773, 0.3854716722, 0.5396603411, 0.1541886689 },  
 { 0.7344604664, 0.3736728689, 0.5411813963, 0.1675085274 },  
 { 0.7572810335, 0.3542120963, 0.5252110393, 0.1587847328 },  
 { 0.7233711848, 0.3419572873, 0.5786969478, 0.1578264403 },  
 { 0.7825805423, 0.3836179129, 0.4603414955, 0.1687918817 }  
 }, *//versicolor* {  
 { 0.6999703739, 0.3238668894, 0.5850498648, 0.2507356563 },  
 { 0.690525124, 0.3214513508, 0.6071858849, 0.2262065061 },  
 { 0.691935021, 0.3256164805, 0.6003553859, 0.2340368453 },  
 { 0.6891487079, 0.3394314531, 0.5862906918, 0.2571450403 },  
 { 0.7215572479, 0.3230853349, 0.5600145805, 0.2476987567 },  
 { 0.7296535933, 0.2895450767, 0.5790901534, 0.2200542583 },  
 { 0.7165389871, 0.3307103017, 0.5732311897, 0.2204735345 },  
 { 0.6746707199, 0.3699807173, 0.5876164334, 0.2502810735 },  
 { 0.7333788618, 0.3294890538, 0.542062637, 0.2444596206 },  
 { 0.6902591586, 0.3509792332, 0.5966646964, 0.2105875399 }  
 } *//virginica* };  
  
}

**import** java.util.Random;  
  
**public class** KohonenWTA {  
  
 **private int noi**; *//ilość wejść* **private double**[] **w**; *//wagi* **public** KohonenWTA ( **int** numbers\_of\_inputs ) {  
 **noi** = numbers\_of\_inputs;  
 **w** = **new double**[**noi**];  
  
 **for** ( **int** i = 0; i < **noi**; i++ )  
 **w**[i] = **new** Random().nextDouble(); *//wagi początkowe sa losowane w zakresie od 0 do 1* }  
  
 *//uczenie poprzez zmniejszenie odległości między wektorem wag a zadanym wektorem* **public void** learn ( **double**[] x, **double** lr ) {  
  
 **for** ( **int** i = 0; i < **noi**; i++ )  
 **w**[i] += lr \* ( x[i] - **w**[i] );  
 }  
  
 *//zwraca wektor wag* **public double**[] getW () {  
 **return w**;  
 }  
}

**public class** Main{  
  
 **private static double** *learningRate* = 0.1; *//współczynnik uczenia się* **private static int** *numberOfInputs* = 4; *//ilość wejść* **private static int** *numberOfNeurons* = 200; *//liczba neuronów* **private static int** *numberOfFlowers* = 3; *//liczba kwiatów* **private static int** *numberOfLearnSamples* = 15; *//liczba danych uczących dla każdego kwiatu* **private static int** *numberOfTestSamples* = 5; *//liczba danych testujacych dla każdego kwiatu* **private static int** *learnLimit* = 10000; *//maksymalny próg epok uczenia* **public static void** main ( String[] args ) {  
  
 **int** successCounter = 0; *//licznik prób uczenia zakończonych powodzeniem* **int** unsuccesCounter = 0; *//licznik prób uczenia zakończonych niepowodzeniem* **while** ( successCounter != 10 && unsuccesCounter != 100 ) {  
  
 KohonenWTA[] kohonens = **new** KohonenWTA[*numberOfNeurons*];  
 **for** ( **int** i = 0; i < *numberOfNeurons*; i++ )  
 kohonens[i] = **new** KohonenWTA( *numberOfInputs* );  
  
 **int** ages = *learn*( kohonens );  
  
 **if** ( ages != *learnLimit* ) {  
 successCounter++;  
  
 **int** winner;  
  
 System.***out***.println( **"PO UCZENIU"** );  
 **for** ( **int** i = 0; i < *numberOfFlowers*; i++ ) {  
 winner = *getWinner*( kohonens, Flower.*flowerLearn*[i][0] );  
 System.***out***.println( **"Flower["** + i + **"] winner = "** + winner );  
 }  
 System.***out***.println();  
  
 System.***out***.println( **"PO TESTOWANIU"** );  
 **for** ( **int** i = 0; i < *numberOfFlowers*; i++ ) {  
 **for** ( **int** j = 0; j < *numberOfTestSamples*; j++ ) {  
 winner = *getWinner*( kohonens, Flower.*flowerTest*[i][j] );  
 System.***out***.println( **"Flower["** + i + **"]["** + j + **"] test winner = "** + winner );  
 }  
 System.***out***.println();  
 }  
 System.***out***.println();  
  
  
 System.***out***.println( **"Ilość epok = "** + ages + **"\n\n\n"** );  
 }  
 **else** unsuccesCounter++;  
 }  
 System.***out***.println( **"\nIlość niepowodzeń = "** + unsuccesCounter );  
 }  
  
  
 *//uczenie sieci* **private static int** learn ( KohonenWTA[] kohonens ) {  
  
 **int** counter = 0;  
 **int** winner;  
  
 **int**[][] winners = **new int**[*numberOfFlowers*][*numberOfLearnSamples*];  
 **for** ( **int** i = 0; i < *numberOfFlowers*; i++ )  
 **for** ( **int** j = 0; j < *numberOfLearnSamples*; j++ )  
 winners[i][j] = - 1;  
  
 **while** ( ! *isUnique*( winners ) ) { *//dopóki sieć się nauczy  
  
 //uczymy sieć po kolei każdy kwiat z każdego gatunku* **for** ( **int** i = 0; i < *numberOfFlowers*; i++ ) {  
 **for** ( **int** j = 0; j < *numberOfLearnSamples*; j++ ) {  
 winner = *getWinner*( kohonens, Flower.*flowerLearn*[i][j] );  
 kohonens[winner].learn( Flower.*flowerLearn*[i][j], *learningRate* );  
 }  
 }  
  
 *//po zakończeniu epoki pobieramy zwycięzców* **for** ( **int** i = 0; i < *numberOfFlowers*; i++ )  
 **for** ( **int** j = 0; j < *numberOfLearnSamples*; j++ )  
 winners[i][j] = *getWinner*( kohonens, Flower.*flowerLearn*[i][j] );  
  
 *//jeśli ilość prób nauczenia osiągnie limit to uczenie uznajemy za nieudane i kończymy* **if** ( ++ counter == *learnLimit* )  
 **break**;  
 }  
  
 **return** counter;  
 }  
  
  
 *//sprawdza czy sieć jest już nauczona* **private static boolean** isUnique ( **int**[][] winners ) {  
  
 *//czy kwiaty danego gatunku mają tylko jednego zwycięzcę* **for** ( **int** i = 0; i < *numberOfFlowers*; i++ )  
 **for** ( **int** j = 1; j < *numberOfLearnSamples*; j++ )  
 **if** ( winners[i][0] != winners[i][j] )  
 **return false**;  
  
 *//czy zwycięzca każdego z gatunków różni się od zwycięzców pozostałych gatunków* **for** ( **int** i = 0; i < *numberOfFlowers*; i++ )  
 **for** ( **int** j = 0; j < *numberOfFlowers*; j++ )  
 **if** ( i != j )  
 **if** ( winners[i][0] == winners[j][0] )  
 **return false**;  
  
 **return true**;  
 }  
  
  
 *//zwraca zwycięzcę dla danego kwiatu* **private static int** getWinner ( KohonenWTA[] kohonens, **double**[] vector ) {  
  
 **int** winner = 0;  
 **double** minDistance = *distanceBetweenVectors*( kohonens[0].getW(), vector );  
  
 *//sprawdza który neuron jest zwycięzcą  
 //miarą zwycięztwa jest odległość między wektorem wag neuronu a wektorem cech kwiatu* **for** ( **int** i = 0; i < *numberOfNeurons*; i++ ) {  
 **if** ( *distanceBetweenVectors*( kohonens[i].getW(), vector ) < minDistance ) {  
 winner = i;  
 minDistance = *distanceBetweenVectors*( kohonens[i].getW(), vector );  
 }  
 }  
  
 **return** winner;  
 }  
  
  
 *//zwraca odległość między zadanymi wektorami* **public static double** distanceBetweenVectors ( **double**[] vector1, **double**[] vector2 ) {  
  
 **double** suma = 0.0;  
  
 **for** ( **int** i = 0; i < vector1.**length**; i++ ) {  
 *//suma += Math.pow( vector1[i] - vector2[i], 2 ); //miara Euklidesowa* suma += Math.*abs*( vector1[i] - vector2[i] ); *//miara Manhattan* }  
  
 **return** Math.*sqrt*( suma );  
 }  
  
}