AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Metody Inżynierii Wiedzy

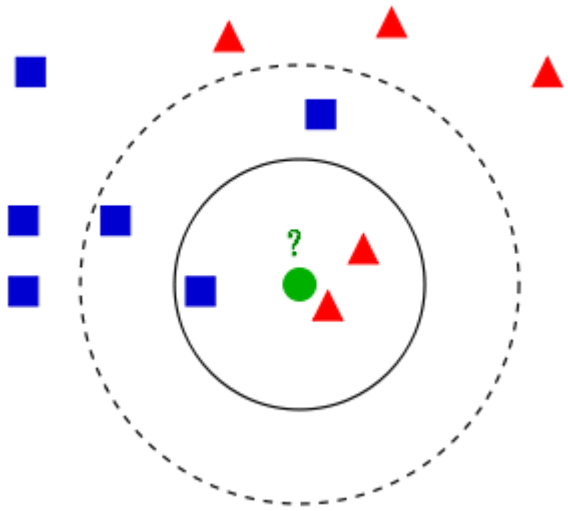
Maciej Radziszowski

KNN IRIS

Metoda KNN z użyciem wag

Przykładowe wyniki:

77	Iris-versicolor	Iris-setosa : 0.00 Iris-versicolor : 0.17 Iris-virginica : 0.28	Iris-virginica
78	Iris-versicolor	Iris-setosa : 0.00 Iris-versicolor : 0.40 Iris-virginica : 0.07	Iris-versicolor
79	Iris-versicolor	Iris-setosa : 0.00 Iris-versicolor : 0.43 Iris-virginica : 0.00	Iris-versicolor
80	Iris-versicolor	Iris-setosa : 0.00 Iris-versicolor : 0.46 Iris-virginica : 0.00	Iris-versicolor
81	Iris-versicolor	Iris-setosa : 0.00 Iris-versicolor : 0.45 Iris-virginica : 0.00	Iris-versicolor
82	Iris-versicolor	Iris-setosa : 0.00 Iris-versicolor : 0.48 Iris-virginica : 0.00	Iris-versicolor
83	Iris-versicolor	Iris-setosa : 0.00 Iris-versicolor : 0.12 Iris-virginica : 0.34	Iris-virginica



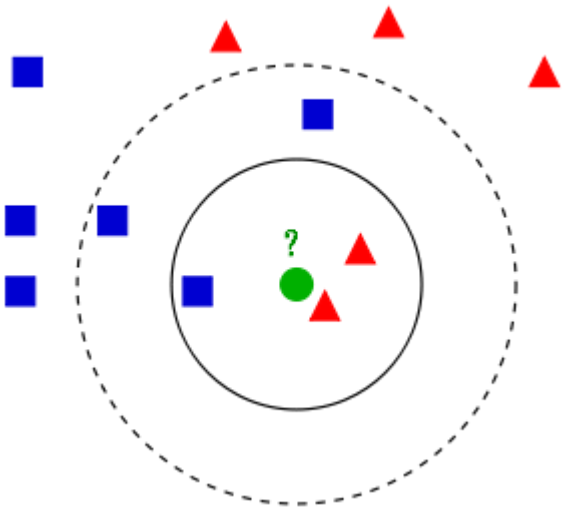
Pod uwagę bierzemy nie tylko ilość najbliższych elementów ale także ich odległość (wagę) od elementu klasyfikowanego

Skuteczność 98%
przy K = 19 lub 21

KNN IRIS

Metoda KNN z użyciem wag

W tabeli można zaobserwować wyniki dla różnych wartości parametru K. Jak widać skuteczność cały czas utrzymuje się na wysokim poziomie jednak najwyższą wartość (98%) osiąga dla K = 19 lub K = 21



IRIS	
K	%
10	96
11	97.33
12	96.66
13	96.66
14	97.33
15	97.33
16	97.33
17	97.33
18	97.33
19	98
20	97.33
21	98
22	97.33
23	96.66
24	96.66
25	96.66
26	96.66
27	96.66
28	96.66
29	95.3
30	96

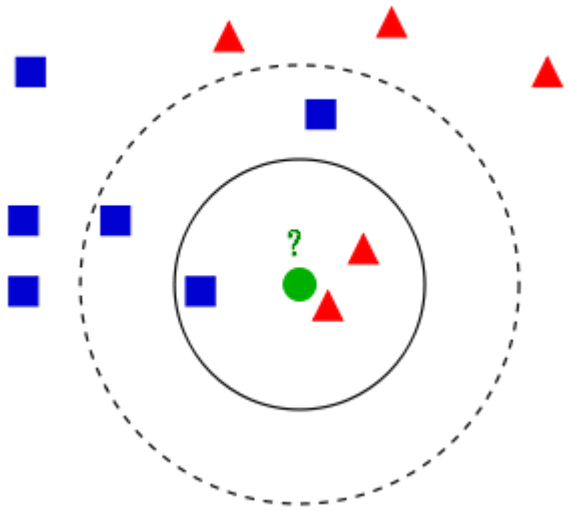
KNN WINE

Metoda KNN z użyciem wag

Przykładowe wyniki:

Skuteczność 75.2%
przy K = 6

Id	Class	Results (weights sum)	KNN Class
0	1	1 : 0.12 2 : 0.00 3 : 0.00	1
1	1	1 : 0.14 2 : 0.00 3 : 0.00	1
2	1	1 : 0.12 2 : 0.00 3 : 0.00	1
3	1	1 : 0.10 2 : 0.00 3 : 0.00	1
4	1	1 : 0.03 2 : 0.04 3 : 0.08	3



Dla porównania zostały przeprowadzone badania na zbiorze „wine” z machine learning repository.

KNN WINE

Skuteczność 75.2%
przy K = 6

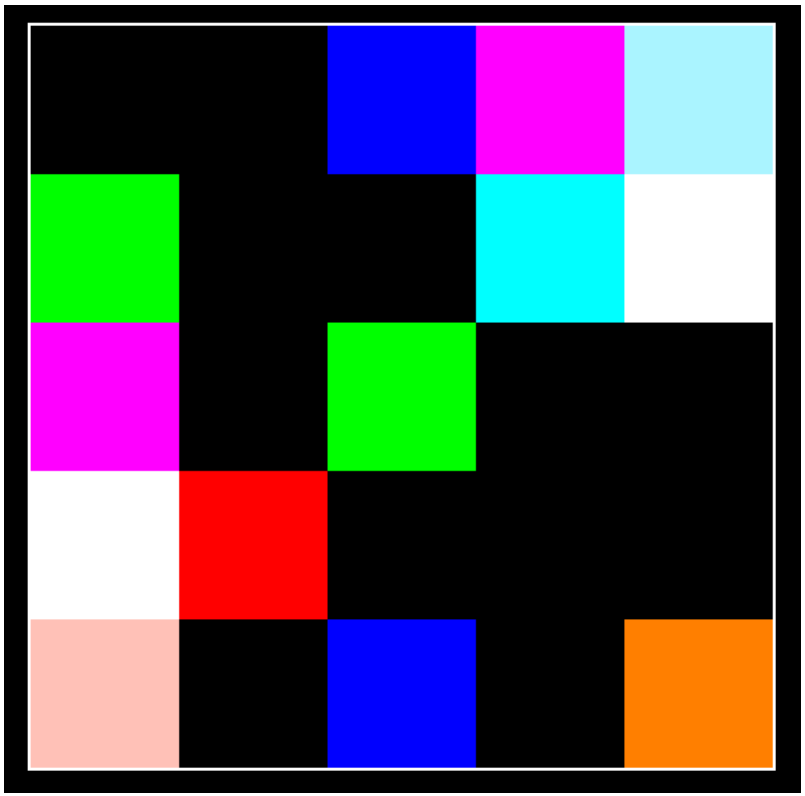
WINE	
K	%
1	76.96
2	77.52
3	74.71
4	71.34
5	69.66
6	75.2
7	70.78
8	72.47
9	73.03
10	73.59
11	72.47
12	73.34
13	71.98
14	69.66
15	70.78
16	70.78
17	71.91
18	73.03
19	73.03
20	71.91
21	71.34
22	70.78
23	70.78
24	70.78
25	70.22
26	71.34
27	72.47
28	71.91
29	71.34
30	71.34

Jak widać teoretycznie największą skuteczność osiągamy dla K = 1 natomiast taka mała wartość parametru powoduje duże niebezpieczeństwo błędnej klasyfikacji (np.. Jeżeli niektóre wzorce są błędne lub odstające) w związku z tym zostało przyjęte iż najlepsze wyniki gwarantuje K = 6

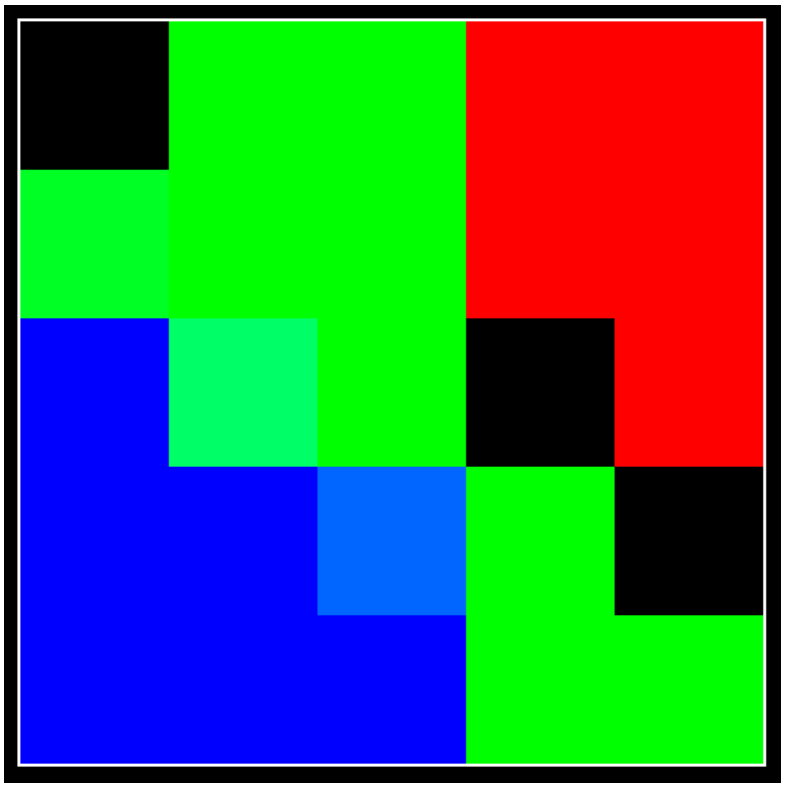
Self-Organizing Map (SOM)

Wizualizacja działania metody SOM

Początek



Koniec



Self-Organizing Map (SOM)

Opis parametrów

width: szerokość siatki
height: wysokość siatki
r0 – promień początkowy
y0 – współczynnik adaptacji wag
alfa – stała zawężania
maxIter – ilość iteracji wykonywanych przez program

Struktura wizualizacji

```
<div class="container">
    <h2 id="iter">Iter: </h2>
    <div id="neurons-table">


    </div>
</div>
```

Do wizualizacji zostały użyte technologie internetowe HTML5/CSS3

Self-Organizing Map (SOM)

Algorytm SOM

1. Stworzenie sieci neuronów
2. Inicjalizacja wag neuronów niewielkimi liczbami losowymi (różnymi od zera)
3. Obliczanie wartości wyjściowej każdego neuronu na podstawie odległości wzorca wejściowego od wektora wag
4. Wyznaczenie najmocniej pobudzonego neurony dla każdego wzorca wejściowego
5. Aktualizacja wag sąsiednich neuronów. Im dalej znajduje się sąsiad od neuronu zwycięskiego tym słabiej jest modyfikowany.
6. Powrót do kroku nr. 3 dopóki wszystkie wzorce nie są reprezentowane z wystarczającą dokładnością

Odległość o której mowa w punkcie nr. 3 jest liczona według następującego wzoru:

$$d(A,B) = \sqrt{(x_{1A} - x_{1B})^2 + (x_{2A} - x_{2B})^2 + \ldots + (x_{nA} - x_{nB})^2} = \sqrt{\sum_{i=1}^{n}((x_{iA} - x_{iB})^2)}$$

Self-Organizing Map (SOM)

Aktualizacja wag

```
for (var l = 0; l < nodes.length; l++) {
    for (var m = 0; m < nodes[l].length; m++) {
        var bestNdist = distNeurons(l, m, curr_iris[i].node.i, curr_iris[i].node.j),
            r_squared = r0 * r0,
            m_influence = Math.pow(Math.exp(1), -(bestNdist / (2 * r_squared))),
            irisArr = $.map(curr_iris[i], function (value, key) {
                if (key != "class" && key != "node")
                    return [value];
            });
        for (var n = 0; n < curr_nodes[l][m].length; n++) {
            curr_nodes[l][m][n] += y0 * m_influence * (irisArr[n] - curr_nodes[l][m][n]);
        }
    }
}
```

Parametry

```
var width = 5,
    height = 5,
    r0 = 5,
    alfa = 1000,
    y0 = 1,
    maxIter = 150,
    iter = 0;
```

Powyższy kod przedstawia proces aktualizacji wag dla każdego neuronu. Dla neuronu reprezentującego dany wzorzec liczymy dystans do sąsiednich neuronów i na podstawie tego dystansu odpowiednio modyfikujemy wagi.

Self-Organizing Map (SOM)

Tworzenie wizualizacji neuronów

```
function drawNodes(size, iris) {

    var countMatches = [],
        stage = $("#neurons-table"),
        clas = 0;

    for (var i = 0; i < size; i++) {
        countMatches[i] = [];
        for (var j = 0; j < size; j++) {
            countMatches[i][j] = [];
            stage.append('<div class="node"></div>');
        }
    }

    $('.node').css({'width': 'calc(100%/' + size + ')', 'height': 'calc(100%/' + size + ')'});

    countMatches.map(function (arr) {
        arr.map(function (elem) {
            elem[0] = 0;
            elem[1] = 0;
            elem[2] = 0;
        })
    });
```

Dzięki zastosowaniu odpowiednich styli (CSS3) niezależnie od rozmiaru siatki wszystkie elementy zachowują stosowne proporcje

Każdy neuron jest zdefiniowany jako element HTML. Zależnie od wybranego rozmiaru siatki odpowiednia ilość takich elementów jest dodawana do kodu HTML

Self-Organizing Map (SOM)

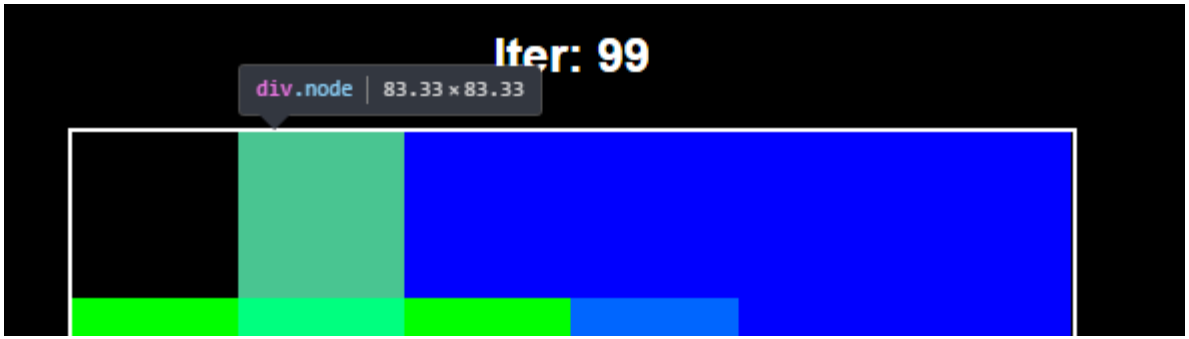
Tworzenie wizualizacji neuronów

```
▼<div id="neurons-table">
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 0, 0);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 255, 0);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 0, 255);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 0, 255);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 0, 255);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 0, 255);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 255, 0);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 255, 127);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 255, 0);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 102, 255);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 0, 255);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 0, 255);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 255, 0);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 255, 0);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 255, 0);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 85, 255);"></div>
  <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color: rgb(0, 0, 255);"></div>
```

Używając przeglądarki internetowej (w tym przypadku chrome) można zobaczyć elementy składowe danej strony przy użyciu „developer tools” (aby włączyć należy otworzyć przeglądarkę i nacisnąć f12). Na powyższym zdjęciu widać elementy HTML wygenerowane przez wcześniej opisaną funkcję.

Self-Organizing Map (SOM)

Tworzenie wizualizacji neuronów



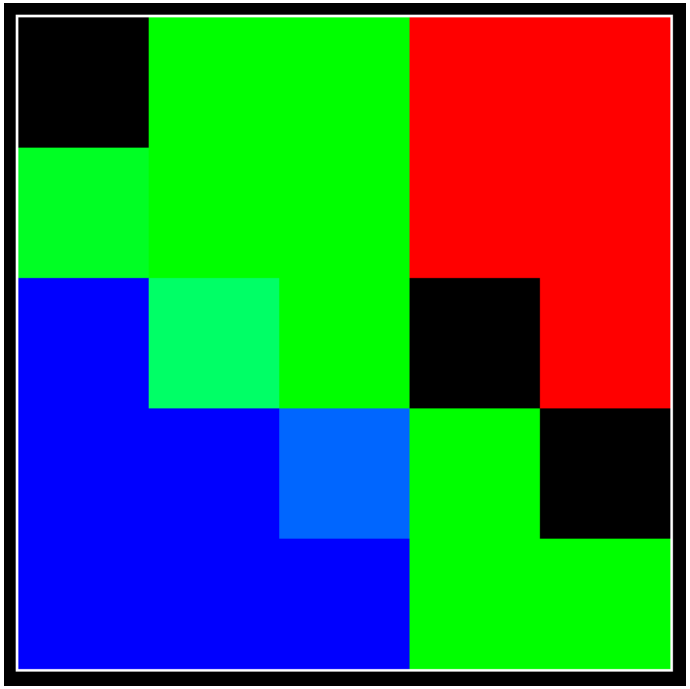

```
element.style {
    width: calc(16.6667%);
    height: calc(16.6667%);
    background-color: rgb(0, 255, 0);
}
```

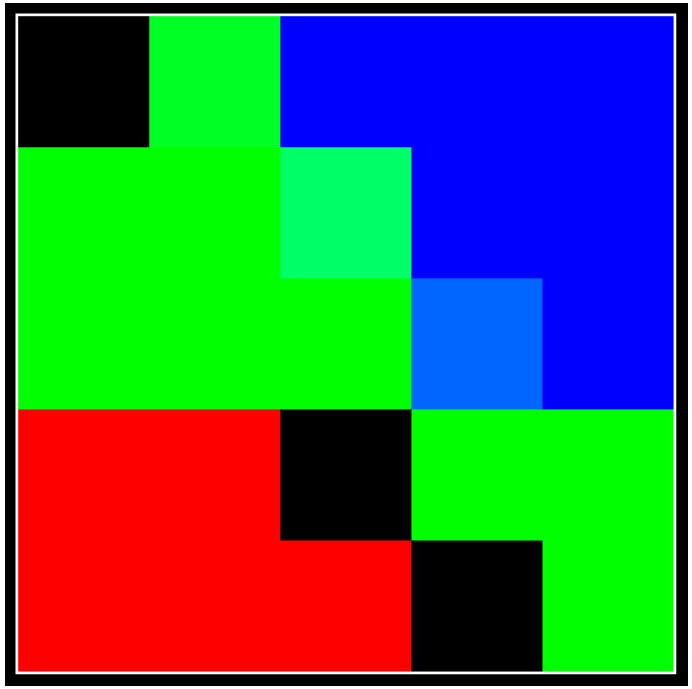
```
▼<div id="neurons-table">
    <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color:
    rgb(0, 0, 0);"></div>
    <div class="node" style="width: calc(16.6667%); height: calc(16.6667%); background-color:
    rgb(0, 255, 0);"></div> == $0
```

Po zaznaczeniu myszką któregoś z elementów zostaje od podświetlony oraz podane są informację odnośnie jego atrybutów (np. kolor, wysokość, szerokość)

Self-Organizing Map (SOM)

Poniżej widzimy dwa końcowe efekty działania algorytmu SOM dla zestawu IRIS.
Należy mieć na uwadze, iż rozmieszczenie poszczególnych wyznaczonych grup może być różne, ponieważ niektóre relacje nie są zachowane. Dobrym przykładem jest klasyfikowanie kolorów. Jeżeli mamy zestaw danych zawierający trzy kolory (czerwony, żółty, zielony) to SOM wyodrębni nam trzy obszary natomiast nie ma gwarancji, że obszar żółty (wynik zmieszania zielonego i czerwonego) będzie w środku.





Self-Organizing Map (SOM)

Własny zestaw danych (mockaroo)

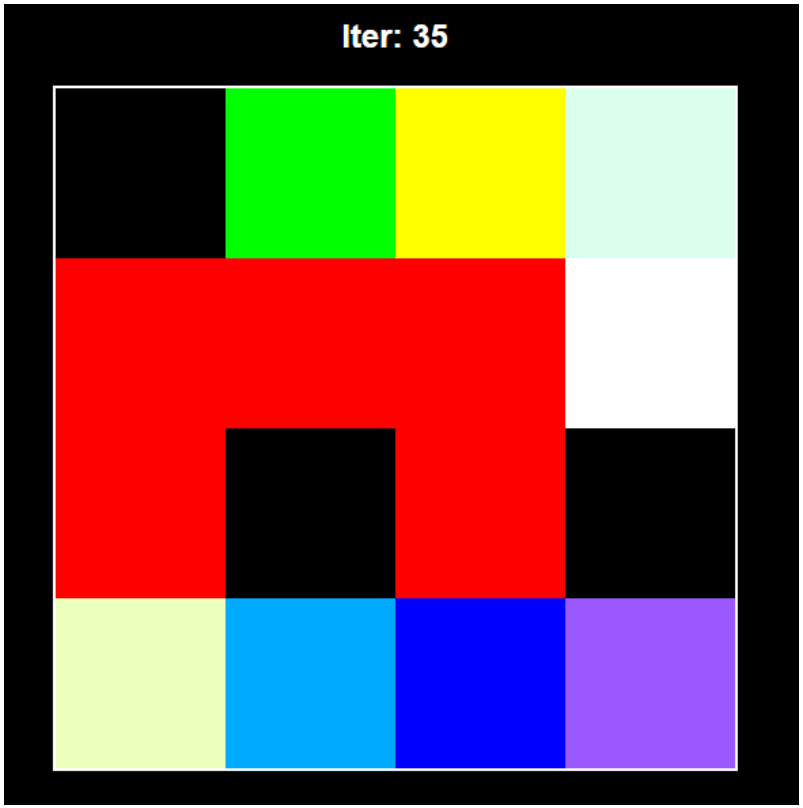
Field Name	Type	Options
prop1	Number	min: 2 max: 3 decimals: 0 blank: 0 %
prop2	Number	min: 3 max: 4 decimals: 0 blank: 0 %
prop3	Number	min: 5 max: 6 decimals: 0 blank: 0 %

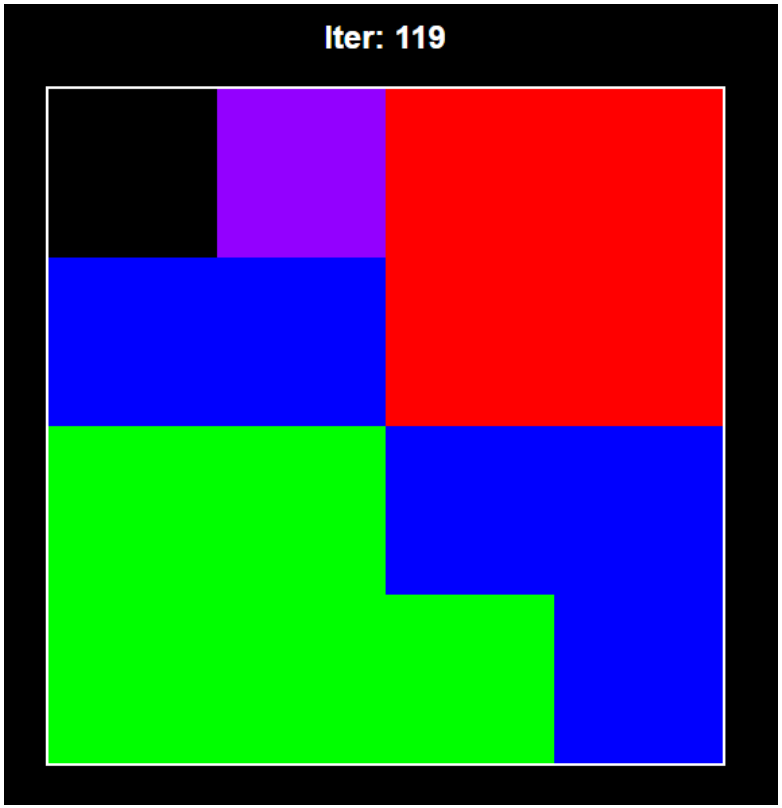
Add another field

Aby lepiej pokazać działanie SOM zostały stworzone dodatkowe zestawy danych przy użyciu narzędzia „mockaroo”. Narzędzie to zapewnia możliwość definicji wielu własności oraz duży wybór typów danych.

Self-Organizing Map (SOM)

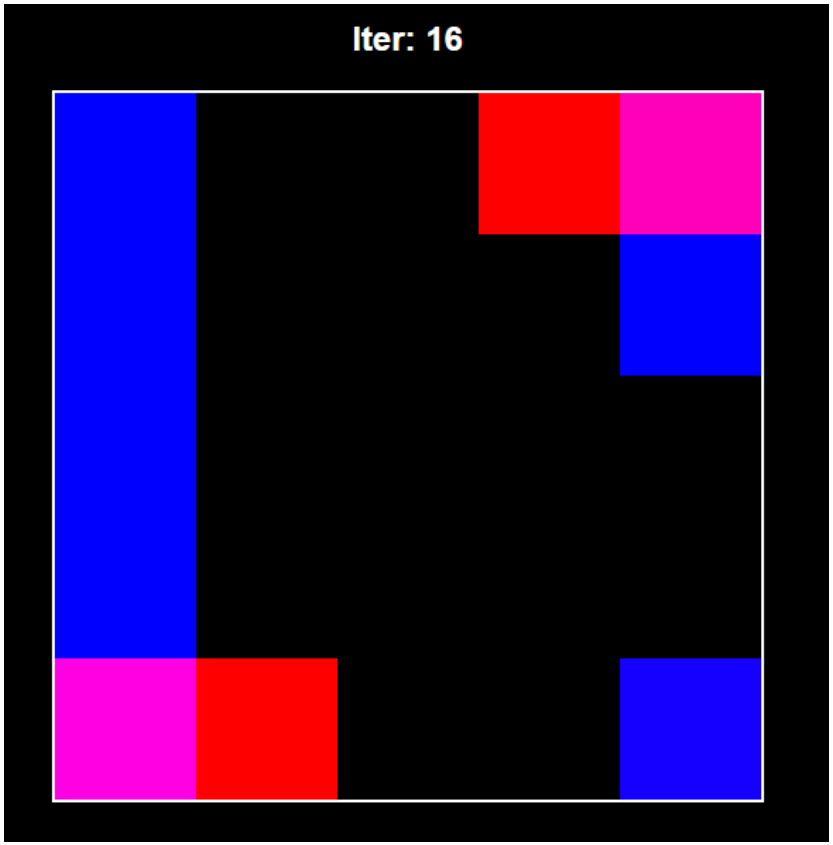
Zestaw składający się z trzech różnych typów obiektów został prawidłowo rozpoznany czego efektem końcowym są trzy wyznaczone obszary

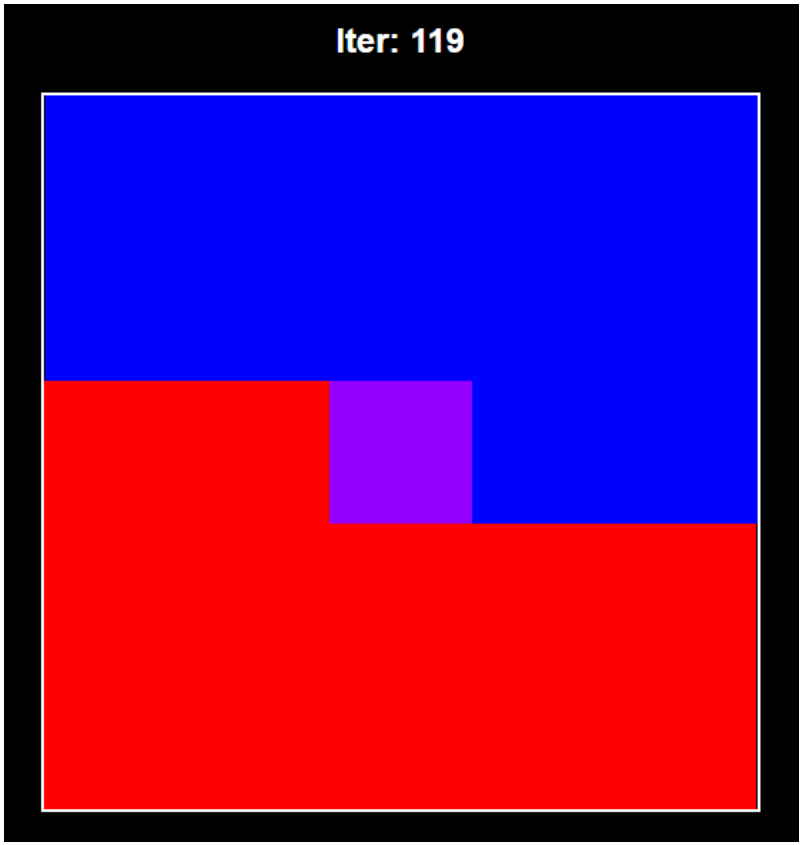




Self-Organizing Map (SOM)

Test został powtórzony kolejny raz dla zestawu danych zawierającego dwa typy obiektów. Również tym razem osiągnięto zamierzony efekt (dwa obszary)





APRIORI (IRIS, GRIMM)

Algorytm APIORI wykorzystujmy w celu wyznaczenia / wyszukania sekwencji jedno bądź wieloelementowych których częstotliwość występowania jest powyżej pewnego zdefiniowanego progu. Algorytm ten składa się z następujących kroków.

- Faza oczyszczania

1. Przeszukanie wszystkich wzorców w celu ustalenia ilości występowania każdego kandydata w całym zbiorze.
2. Ilość wystąpień każdego z kandydata jest porównywana z wymaganym wsparciem. Na podstawie porównania kandydat jest umieszczany w zbiorze wzorców częstych albo odrzucany.

- Faza łączenia

1. Wzorce częste są ze sobą łączone w celu wygenerowania k+1 elementowych wzorców (kandydatów). Po łączeniu po raz kolejny sprawdzamy wsparcie nowo powstałych wzorców.

APRIORI (IRIS, GRIMM)

Przed rozpoczęciem zliczania wystąpień zależy wygenerować tablice kandydatów. Pomocna w tym celu jest poniższa funkcja

```
function getUniqueValues(data) {
    let uniqArr = [];
    data.map(function (arr) {
        arr.map(function (elem) {
            if (uniqArr.indexOf(elem) == -1) {
                uniqArr.push(elem);
            }
        })
    });

    return uniqArr;

}
```

Funkcja ta sprawdza po kolei wartości każdego obiektu. Jeżeli dana wartość nie jest jeszcze obecna w tablicy zostaje do niej wówczas dodana

APRIORI (IRIS, GRIMM)

```
function countOccurances(data, uniqArr) {
    let occ = {}; // occurances table for example [{'A': 2(times)}]
    data.map(function (obj) {
        uniqArr.map(function (elem) {
            if (typeof(elem) != "object") {
                elem = [elem];
            }
            let key = String(elem);
            if (!occ[key] && subArray(obj, elem)) {
                occ[key] = 1;
            }
            else if (subArray(obj, elem)) {
                occ[key]++;
            }
        })
    });

    return occ;

}
```

Przykład wyniku działania funkcji widocznego z poziomu „developer tools – google chrome” (baśnie grimm)

```
▼Object {and,the: 63, and,to: 47, the,to: 51}
   and,the: 63
   and,to: 47
   the,to: 51
```

Kolejnym krokiem jest zliczanie wystąpień każdego z kandydatów

Funkcja zwraca nam informacje na temat częstości wystąpień w formie obiektu. Każdy klucz obiektu jest naszym kandydatem a wartość pod danym kluczem oznacza częstość wystąpień danego kandydata.

{
A: 2,
B: 3
}

APRIORI (IRIS, GRIMM)

```
function checkSupport(occ, data, s) {

    let supported = [];
    let dataLength = data.length;
    for (let key in occ) {
        if (occ[key] / dataLength >= s) {
            supported.push(key);
        }
    }
    return supported;

}
```

Mając policzone częstości danych wzorców należy sprawdzić czy spełniają one docelowe wsparcie. Jeżeli częstość kandydata jest wystarczająca zostaje on dodany do listy wzorców częstych.

Na zdjęciu poniżej widzimy przykład przedstawiający zestaw jedno elementowych kandydatów spełniających wsparcie na poziomie 30% w zbiorze „baśnie grimm”. Po lewej stronie jest kandydat a po prawej liczba jego wystąpień.

```
kandydat: and | ilość wystąpień: 64(68.82%)
kandydat: the | ilość wystąpień: 71(76.34%)
kandydat: to | ilość wystąpień: 52(55.91%)
kandydat: was | ilość wystąpień: 29(31.18%)
kandydat: his | ilość wystąpień: 28(30.11%)
kandydat: he | ilość wystąpień: 42(45.16%)
```

APRIORI (IRIS, GRIMM, INNE)

Algorytm został przetestowany dla różnych kombinacji. Obok można zaobserwować otrzymane rezultaty. Poniżej znajduje się kod odpowiedzialny ze pętlę która filtruje i łączy zbiór wzorców częstych

IRIS support 5% wyniki dla kombinacji 2 elementowych:

```
1.40,0.20 support: 5.3 %
```

GRIMM support 40% wyniki dla kombinacji 3 elementowych:

```
and,the,to support: 49.5 %
and,the,he support: 40.9 %
```

GRIMM support 20% wyniki dla kombinacji 4 elementowych:

```
and,the,to,was support: 23.7 %
and,the,to,of support: 20.4 %
and,the,to,his support: 22.6 %
and,the,to,he support: 34.4 %
and,the,was,he support: 23.7 %
and,the,his,he support: 21.5 %
```

```
while (supported.length > 0 && size <= maxSize) {
    Util.getAllPossibleCombinations(supported, size, combinations);
    lastOcc = occ;
    occ = countOccurances(data, combinations);
    lastSupportedComb = supportedComb;
    supportedComb = checkSupport(occ, data, supp);
    combinations = [];
    size++;
}
```

AGDS

Konwersja zbioru IRIS do postaci AGDS

Przykładowy obiekt

```
{
    "leaf-length": "5,00",
    "leaf-width": "3,00",
    "petal-length": "1,60",
    "petal-width": "0,20",
    "class": "Iris-setosa"
},
```

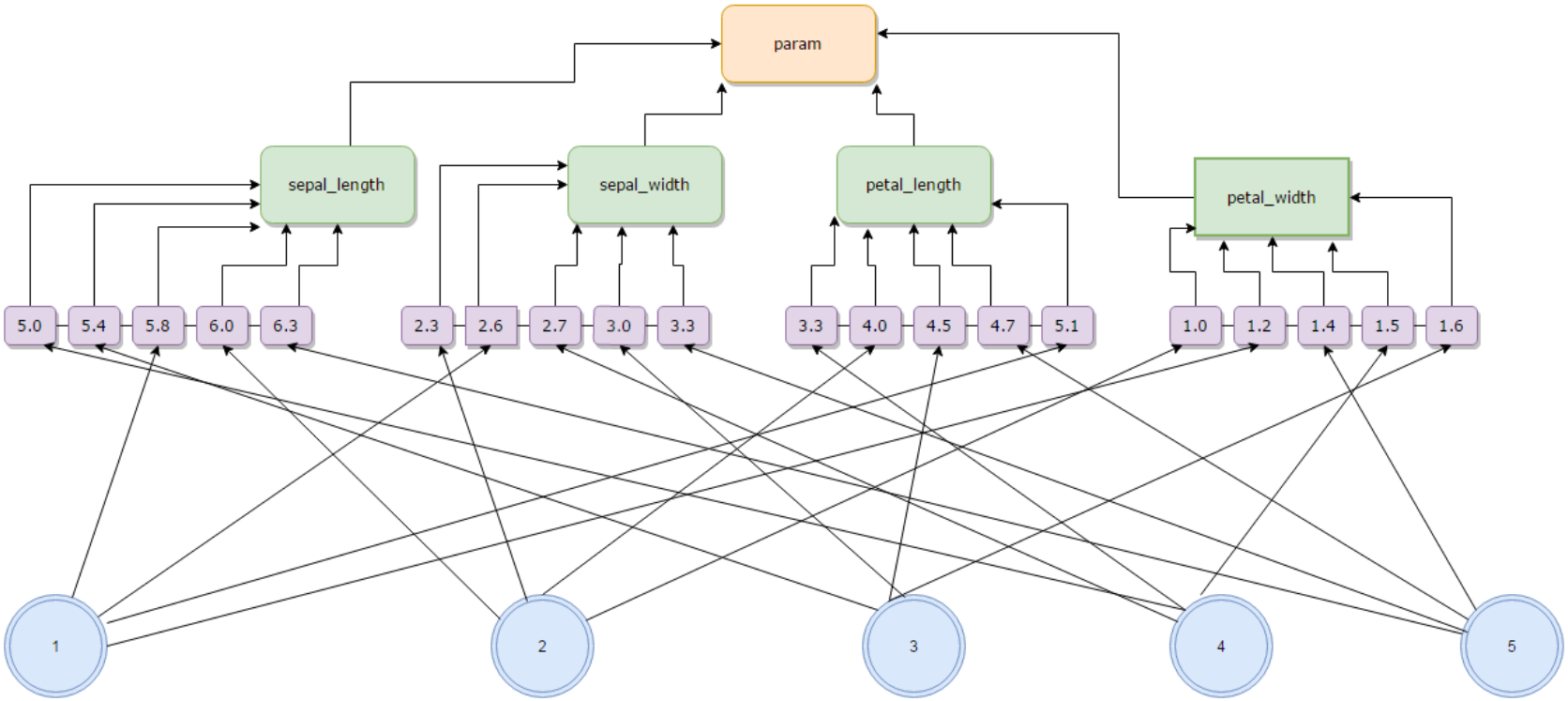
Obiekty podobne

```
▶ (4) ["0.100", "0.100", "0.100", "0.100"] "96.8%"
▶ Object {leaf-length: "4.90", leaf-width: "3.10", petal-length: "1.50", petal-width: "0.10", class: "Iris-setosa"}
▶ (4) ["0.100", "0.100", "0.100", "0.100"] "96.8%"
▶ Object {leaf-length: "4.90", leaf-width: "3.10", petal-length: "1.50", petal-width: "0.10", class: "Iris-setosa"}
▶ (4) ["0.100", "0.100", "0.100", "0.100"] "96.8%"
▶ Object {leaf-length: "4.90", leaf-width: "3.10", petal-length: "1.50", petal-width: "0.10", class: "Iris-setosa"}
```


```
{
   "param": {
      "leaf-length": {
         "max": 7.9,
         "min": 4.3,
         "values": [
            {
               "4.30": [
                  13
               ],
               "weight_next": 0.9722222222222221,
               "weight": 0.8055555555555556
            },
            {
               "4.40": [
                  8,
                  38,
                  42
               ],
               "weight_prev": 0.9722222222222221,
               "weight_next": 0.9722222222222223,
               "weight": 0.8333333333333335
            },
            {
               "4.50": [
                  41
               ],
               "weight_prev": 0.9722222222222223,
               "weight_next": 0.9722222222222223,
               "weight": 0.8611111111111112
            },
            {
               "4.60": [
                  3,
                  6,
                  22,
                  47
               ],
               "weight_prev": 0.9722222222222223,
               "weight_next": 0.9722222222222221,
               "weight": 0.8888888888888888
            },
```

AGDS - WIZUALIZACJA

Poniższa wizualizacja przedstawia architekturę grafu AGDS na podstawie wycinka zbioru IRIS



AGDS

Wyszukiwanie obiektów z danym parametrem w podanym zakresie

```
w.agds.findPropertyInRange = (property, min, max, objects, graph) => {

    let results = [];
    graph[property].values.filter((x) => {
        let firstKey = Object.keys(x)[0];

        if (firstKey <= max && firstKey >= min) {
            x[firstKey].forEach((id) => {
                results.push(objects[id]);
            })
        }
    });

    return results;

};
```

AGDS dostarcza wiele przydatnych funkcjonalności. Jedną z nich jest możliwość szybkiego wyszukania obiektów o danym parametrze mieszącym się w podanym zakresie

Przy wywołaniu funkcji należy podać nazwę własności, minimalna wartość, maksymalna wartość, zestaw danych oraz obiekt grafu AGDS.

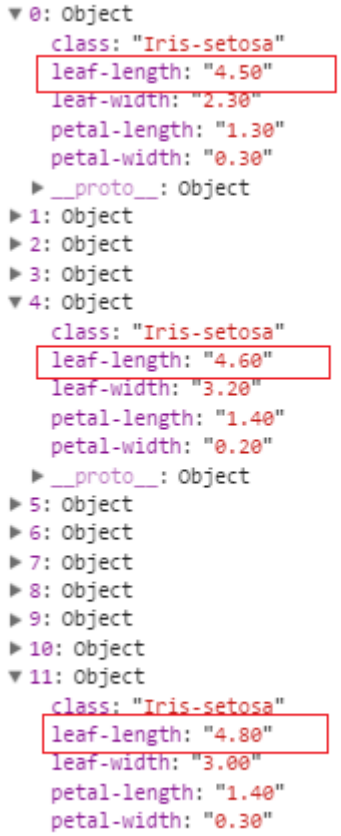
```
let propertyInRange = agds.findPropertyInRange("petal-width", 0.4, 0.5, iris, graph.param);
```

Wyszukiwanie obiektów z danym parametrem w podanym zakresie

Przykład:

Parametr: „leaf-length”
Min: 4.5
Max: 4.8

Wywołanie omówionej funkcji dla podanych wyżej parametrów skutkuje otrzymaniem widocznych po prawej stronie listy obiektów. Jak widać wartości wybranego parametru mieszczą się w zadanym zakresie.



AGDS - Wyszukiwanie obiektów z danym parametrem w podanym zakresie

Wyszukiwanie obiektów z danym parametrem w podanym zakresie

Przykład:

Parametr: „petal-width”
Min: 0.4
Max: 0.5

Wywołanie tej samej funkcji dla innych parametrów w celach demonstracyjnych

AGDS - Wyszukiwanie obiektów z danymi parametrami w podanym zakresie

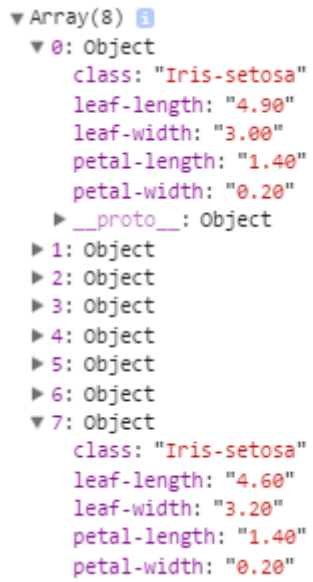
```
w.agds.findPropertiesInRange = (propertiesObj, objects, graph) => {
    let results,
        iterations_results = [];

    propertiesObj.forEach((property, index) => {


        iterations_results[index] = [];

        graph[property.name].values.filter((x, i) => {
            let firstKey = Object.keys(x)[0];
            if (firstKey <= property.max && firstKey >= property.min) {
                x[firstKey].forEach((id) => {
                    iterations_results[index].push(objects[id]);
                })
            }
        });

    });

    results = iterations_results.shift().filter(function (v) {
        return iterations_results.every(function (a) {
            return a.indexOf(v) !== -1;
        });
    });

    return results;


};
```

Kolejna przydatną funkcjonalnością jest możliwość szukania obiektów których pewien zestaw atrybutów znajduje się w podanym zakresie

AGDS - Wyszukiwanie obiektów z danymi parametrami w podanym zakresie

Przykład użycia powyższej funkcji:

```
let propertiesInRange = agds.findPropertiesInRange([
    {
        name: "leaf-width",
        min: 3.0,
        max: 3.2
    },
    {
        name: "petal-length",
        min: 1.4,
        max: 1.5
    }
], iris, graph.param);
console.log(propertiesInRange);
```



Jak widać tym razem jako argument podjemy tablice obiektów. Każdy obiekt zawiera nazwę atrybuty oraz zakres wartości jakie ten atrybut powinien przyjmować. Po prawej stronie widać listę obiektów spełniających zadane wymagania.

AGDS – WYSZUKIWANIE DANEJ KLASY

Wyszukiwanie obiektów danej klasy

```
w.agds.findClass = (cls, objects, graph) => {
    let results = [],
        classObjsIds = graph["class"].values.filter(x => {
            let firstKey = Object.keys(x)[0];
            return firstKey == cls;
        });

    let firstKey = Object.keys(classObjsIds[0])[0];
    classObjsIds[0][firstKey].forEach((x) => {
        results.push(objects[x]);
    });

    return results;

};
```

Następna istotną funkcjonalnością jest wyszukiwanie obiektów danej klasy.

Jako argument podajemy nazwę szukanej klasy, zestaw danych oraz obiekt grafu AGDS. Rezultatem wywołania funkcji w sposób podany poniżej jest otrzymanie wszystkich obiektów o klasie „Iris-setosa”

```
agds.findClass("Iris-setosa", iris, graph.param);
```

AGDS - WYDAJNOŚĆ

Porównanie prędkości działania AGDS i MySQL na zbiorze wygenerowanym przy pomocy narzędzia „mockaroo” zawierającym 1000 obiektów. Każdy obiekt posiada 3 atrybuty.

Wydajność kodu JavaScript została zmierzona przy pomocy wbudowanej funkcji ‚performance.now()”. Natomiast wydajność zapytań SQL została zmierzona przy użyciu narzędzi dostępnych w „phpMyAdmin”

Konfiguracja:

PC: ASUS S46CM
CPU: I5-3317U
RAM: 12GB DDR3
Server: localhost
Baza: MySQL

Function	Time [ms]	Type
findPropertyInRange	0.11	AGDS
findPropertiesInRange	0.38	AGDS
findClass	0.02	AGDS

Function	Time [ms]	Type
findPropertyInRange	0.33	MySQL
findPropertiesInRange	0.51	MySQL
findClass	0.08	MySQL