

**TECHNICKÁ UNIVERZITA V KOŠICIACH**  
**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

## **K-means alghoritm**

**TECHNICKÁ UNIVERZITA V KOŠICIACH**  
**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

**K-means alghoritm**

Študijný program:     Inteligentny systemy

Študijný odbor:       Informatika

Školiace pracovisko:   KKUI

## Obsah

Uvod .....	4
Popis postupu a jednotlivých funkcií.....	5
1. Cluster .....	5
2. Normalization of data .....	6
3. Inicializácia zhlukov .....	7
4. Metóda UpdateMeans .....	8
5. Metóda UpdateClustering.....	9
6. Tu uvediem pomocné funkcie, ktoré odrážajú výsledok: .....	11
Popis dát.....	12
Vyhodnotenie.....	13
Odkazy .....	14

## Uvod

V tejto práci by som chcel opísať, ako funguje algoritmus k-means.

Chcel by som stručne opísať algoritmus:

Metóda k-means je najpopulárnejšou metódou zhlukovania. V päťdesiatych rokoch 20. storočia ju vynášiel matematik Hugo Steingauz a takmer súčasne Stuart Lloyd. Mimoriadnu popularitu získala po práci McQueenena. [1]

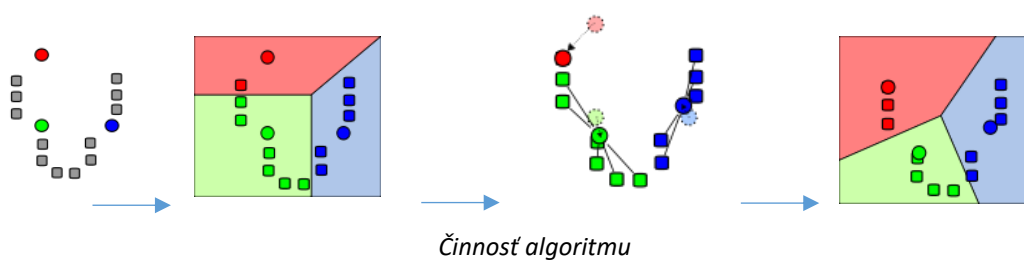
Činnosť algoritmu spočíva v tom, že sa snaží minimalizovať celkovú kvadratickú odchýlku bodov zhlukov od stredov týchto zhlukov:

*Celková kvadratická odchýlka bodov zhlukov od stredov týchto zhlukov*

$$V = \sum_{i=1}^k \sum_{x \in S_i} (x - \mu_i)^2$$

1. k je počet zhlukov
2.  $S_{\{i\}}$  sú získané zhluky
3.  $\mu_i$  stredy hmotností všetkých vektorov X zo zhluku  $S_{\{i\}}$

Ukážka algoritmu:



## Popis postupu a jednotlivých funkcí

### 1. Cluster

```
public static int[] Cluster(double[][] rawData, int numClusters)
{
    double[][] data = Normalized(rawData);

    bool changed = true;
    bool success = true;

    int[] clustering = InitClustering(data.Length, numClusters, 0);
    double[][] means = Allocate(numClusters, data[0].Length);

    int maxCount = data.Length * 10;
    int ct = 0;
    while (changed == true && success == true && ct < maxCount)
    {
        ++ct;
        success = UpdateMeans(data, clustering, means);
        changed = UpdateClustering(data, clustering, means);
    }
    return clustering;
}
```

1. Zhlukovanie sa začína volaním metódy Normalized, ktorá prevedie nespracované údaje, napríklad {65,0, 220,0}, na normalizované údaje, napríklad {-0,05, 0,02}.
2. Logická premenná s názvom "changed" sleduje, či sa zmenili zhluky akýchkoľvek dátových tuplov, alebo ekvivalentne, či sa zmenilo zhlukovanie.
3. Logická premenná "success" (úspech) označuje, či boli priemery zhlukov úspešne vypočítané.
4. Premenné ct a maxCount sa používajú na obmedzenie počtu opakovaní cyklu zhlukovania, čo v podstate funguje ako kontrola správnosti.
5. Cyklus zhlukovania sa ukončí, ak nedošlo k žiadnym zmenám v zhlukovaní alebo ak nie je možné vypočítať jeden alebo viac priemerov.

## 2. Normalization of data

```
private static double[][] Normalized(double[][] rawData)
{
    double[][] result = new double[rawData.Length][];
    for (int i = 0; i < rawData.Length; ++i)
    {
        result[i] = new double[rawData[i].Length];
        Array.Copy(rawData[i], result[i], rawData[i].Length);
    }

    for (int j = 0; j < result[0].Length; ++j)
    {
        double colSum = 0.0;
        for (int i = 0; i < result.Length; ++i)
            colSum += result[i][j];
        double mean = colSum / result.Length;
        double sum = 0.0;
        for (int i = 0; i < result.Length; ++i)
            sum += (result[i][j] - mean) * (result[i][j] - mean);
        double sd = sum / result.Length;
        for (int i = 0; i < result.Length; ++i)
            result[i][j] = (result[i][j] - mean) / sd;
    }
    return result;
}
```

V situáciách, keď majú zložky súboru údajov rôzne stupnice, je zvyčajne vhodné údaje normalizovať. Ide o to, že zložka s väčšími hodnotami (váha v demonštračnom príklade) bude dominovať nad zložkou s menšími hodnotami.

Tento kód používa Gaussovu normalizáciu. Každá nespracovaná hodnota v stĺpci sa prevedie na normalizovanú hodnotu tak, že sa odčíta aritmetický priemer všetkých hodnôt v stĺpci a potom sa vydolí štandardnou odchýlkou hodnôt.

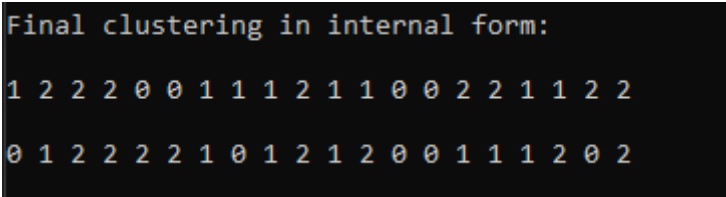
### 3. Inicializácia zhhlukov

```
private static int[] InitClustering(int numTuples, int numClusters,
int randomSeed)
{
    Random random = new Random(randomSeed);
    int[] clustering = new int[numTuples];
    for (int i = 0; i < numClusters; ++i)
        clustering[i] = i;
    for (int i = numClusters; i < clustering.Length; ++i)
        clustering[i] = random.Next(0, numClusters);
    return clustering;
}
```

Táto metóda spočiatku priradzuje každému prvku súboru údajov určitý druh zhľuku, ale v tomto prípade je to najjednoduchšia technika, ktorá v praxi dobre funguje.

A v tomto prípade každý zo zhľukov dostane aspoň jednu hodnotu súboru údajov.

Na tomto obrázku je znázornené konečné zhľukovanie a na druhom riadku počiatočné zhľukovanie.



```
Final clustering in internal form:
1 2 2 2 0 0 1 1 1 2 1 1 0 0 2 2 1 1 2 2
0 1 2 2 2 2 1 0 1 2 1 2 0 0 1 1 1 2 0 2
```

*Počiatočná inicializácia klastrov*

## 4. Metóda UpdateMeans

```
private static bool UpdateMeans(double[][] data, int[] clustering,
double[][] means)
{
    int numClusters = means.Length;
    int[] clusterCounts = new int[numClusters];
    for (int i = 0; i < data.Length; ++i)
    {
        int cluster = clustering[i];
        ++clusterCounts[cluster];
    }

    for (int k = 0; k < numClusters; ++k)
        if (clusterCounts[k] == 0)
            return false;

    for (int k = 0; k < means.Length; ++k)
        for (int j = 0; j < means[k].Length; ++j)
            means[k][j] = 0.0;

    for (int i = 0; i < data.Length; ++i)
    {
        int cluster = clustering[i];
        for (int j = 0; j < data[i].Length; ++j)
            means[cluster][j] += data[i][j];
    }

    for (int k = 0; k < means.Length; ++k)
        for (int j = 0; j < means[k].Length; ++j)
            means[k][j] /= clusterCounts[k];
    return true;
}
```

1. Prvých niekoľko riadkov UpdateMeans prehľadá vstupné zhukovacie pole a spočíta počet tuplov priradených ku každému zhuku.

2. Ak nie sú k žiadnemu zhuku priradené žiadne tuply, metóda sa ukončí a vráti false.

Výpočet priemeru zhukov sa najlepšie vysvetlí na príklade. Predpokladajme, že zhuk pozostáva z troch kôpok:  $d_0 = \{64, 110\}$ ,  $d_1 = \{65, 160\}$ ,  $d_2 = \{72, 180\}$ . Priemer zhuku sa vypočíta ako  $\{(64+65+72)/3, (110+160+180)/3\} = \{67, 0, 150, 0\}$ . Inými slovami, jednoducho vypočítate priemer každej zložky údajov.



## 5. Metóda UpdateClustering

```
private static bool UpdateClustering(double[][] data, int[]
clustering, double[][] means)
{
    int numClusters = means.Length;
    bool changed = false;

    int[] newClustering = new int[clustering.Length];
    Array.Copy(clustering, newClustering, clustering.Length);

    double[] distances = new double[numClusters];

    for (int i = 0; i < data.Length; ++i)
    {
        for (int k = 0; k < numClusters; ++k)
            distances[k] = Distance(data[i], means[k]);

        int newClusterID = MinIndex(distances);
        if (newClusterID != newClustering[i])
        {
            changed = true;
            newClustering[i] = newClusterID;
        }
    }

    if (changed == false)
        return false;

    int[] clusterCounts = new int[numClusters];
    for (int i = 0; i < data.Length; ++i)
    {
        int cluster = newClustering[i];
        ++clusterCounts[cluster];
    }

    for (int k = 0; k < numClusters; ++k)
        if (clusterCounts[k] == 0)
            return false;

    Array.Copy(newClustering, clustering, newClustering.Length);
    return true;
}
```

```

private static double Distance(double[] tuple, double[] mean)
{
    double sumSquaredDiffs = 0.0;
    for (int j = 0; j < tuple.Length; ++j)
        sumSquaredDiffs += Math.Pow((tuple[j] - mean[j]), 2);
    return Math.Sqrt(sumSquaredDiffs);
}

private static int MinIndex(double[] distances)
{
    int indexOfMin = 0;
    double smallDist = distances[0];
    for (int k = 0; k < distances.Length; ++k)
    {
        if (distances[k] < smallDist)
        {
            smallDist = distances[k];
            indexOfMin = k;
        }
    }
    return indexOfMin;
}

```

Metóda UpdateClustering využíva myšlienku vzdialenosti medzi kôpkou údajov a priemerom zhľuku. Euklidovská vzdialenosť medzi dvoma vektormi je odmocnina zo súčtu štvorcov rozdielov medzi hodnotami príslušných zložiek. Predpokladajme napríklad, že nejaký kôpka údajov  $d_0 = \{68, 140\}$  a tri zhľukové priemery:  $c_0 = \{66, 0, 120, 0\}$ ,  $c_1 = \{69, 0, 160, 0\}$  a  $c_2 = \{70, 0, 130, 0\}$ . Vzdialenosť medzi  $d_0$  a  $c_0 = \sqrt{(68 - 66,0)^2 + (140 - 120,0)^2} = 20,10$ . Vzdialenosť medzi  $d_0$  a  $c_1 = \sqrt{(68 - 69,0)^2 + (140 - 160,0)^2} = 20,22$ . A vzdialenosť medzi  $d_0$  a  $c_2 = \sqrt{(68 - 70,0)^2 + (140 - 130,0)^2} = 10,20$ . Tento dátový tuple je najbližšie k strednej hodnote  $c_2$ , a preto bude priradený do zhľuku 2.

Metóda UpdateClustering prehľadá každý tuple údajov, vypočíta vzdialenosti od aktuálneho tuple ku každému z priemerov zhľukov a potom priradí tuple k najbližšiemu priemeru pomocou pomocnej funkcie MinIndex.

Metóda UpdateClustering vypočíta navrhované nové zhľukovanie v lokálnom poli newClustering a potom spočíta počet tuplov priradených ku každému zhľuku v navrhovanom zhľukovaní. Ak nie sú k žiadnemu zhľuku priradené žiadne dátové tuple, UpdateClustering sa ukončí a vráti false. V opačnom prípade sa nové zoskupenie skopíruje do parametra clustering. Počas kopírovania metóda sleduje, či nedošlo k zmene členstva v zhľuku. Ak nedošlo k žiadnej zmene, metóda sa ukončí a vráti false.

6. Tu uvediem pomocné funkcie, ktoré odrážajú výsledok:

```
static void ShowData(double[][] data, int decimals, bool indices, bool
newLine)
{
    for (int i = 0; i < data.Length; ++i)
    {
        if (indices) Console.Write(i.ToString().PadLeft(3) + " ");
        for (int j = 0; j < data[i].Length; ++j)
        {
            if (data[i][j] >= 0.0) Console.Write(" ");
            Console.Write(data[i][j].ToString("F" + decimals) + " ");
        }
        Console.WriteLine("");
    }
    if (newLine) Console.WriteLine("");
}

static void ShowVector(int[] vector, bool newLine)
{
    for (int i = 0; i < vector.Length; ++i)
        Console.Write(vector[i] + " ");
    if (newLine) Console.WriteLine("\n");
}

static void ShowClustered(double[][] data, int[] clustering, int
numClusters, int decimals)
{
    for (int k = 0; k < numClusters; ++k)
    {
        Console.WriteLine("=====");
        for (int i = 0; i < data.Length; ++i)
        {
            int clusterID = clustering[i];
            if (clusterID != k) continue;
            Console.Write(i.ToString().PadLeft(3) + " ");
            for (int j = 0; j < data[i].Length; ++j)
            {
                if (data[i][j] >= 0.0) Console.Write(" ");
                Console.Write(data[i][j].ToString("F" + decimals) + " ");
            }
            Console.WriteLine("");
        }
        Console.WriteLine("=====");
    }
}
```

Prvá funkcia(ShowData) nám zobrazí samotný súbor údajov,  
Druhá funkcia(ShowVector) zobrazí výsledok vo forme reťazca,  
Tretia funkcia(ShowClustered) zobrazí zhľuky jednotlivo

## Popis dát

V tejto časti budem stručne hovoriť o súbore údajov, ktorý som použil.

Použitý súbor údajov

Indian Premier League 2018 Údaje o pálke a bowlingu [2]

-25 parametrov, ponechané len čisté parametre a použitých bolo 21 parametrov

-Úprimne povedané, súbor údajov som vybral náhodne, prvá vec, ktorá ma napadla

-Musel som previesť číselné údaje na typ double a použil som na to tento kód

```
var Data = File.ReadAllLines("output.csv").Select(x =>
x.Split(',').Take(15).Select(s => double.Parse(s, NumberStyles.Any,
CultureInfo.InvariantCulture)).ToArray())
.ToArray();
```

PLAYER	Mat.x	Inns.x	NO	Runs.x	HS	Avg.x	BF	SR.x	X100	X50	X4s	X6s	Mat.y	Inns.y	Ov	Runs.y	Wkts	BBI	Avg.y	Econ	SR.y	X4w	X5w	y
Aaron Finch	10	9	1	134	46	16.75	100	134	0	0	6	8	0	0	0	0	0	0	0	0	0	0	0	0
AB de Villiers	12	11	2	480	90	53.33	275	174.54	0	6	39	30	0	0	0	0	0	0	0	0	0	0	0	0
Abhishek Sharma	3	3	2	63	46	63	33	190.9	0	0	3	5	0	0	0	0	0	0	0	0	0	0	0	0
Ajinkya Rahane	15	14	1	370	65	28.46	313	118.21	0	1	39	5	0	0	0	0	0	0	0	0	0	0	0	0
Alex Hales	6	6	0	148	45	24.66	118	125.42	0	0	13	6	0	0	0	0	0	0	0	0	0	0	0	0
Ambati Rayudu	16	16	2	602	100	43	402	149.75	1	3	53	34	0	0	0	0	0	0	0	0	0	0	0	0
Andre Russell	16	14	3	316	88	28.72	171	184.79	0	1	17	31	16	15	37.5	355	13	0	27.3	9.38	17.46	0	0	0
Andrew Tye	14	8	2	32	14	5.33	38	84.21	0	0	2	1	14	14	56	448	24	0	18.66	8	14	3	0	0
Axar Patel	9	8	2	80	19	13.33	69	115.94	0	0	3	4	9	8	26	218	3	0	72.66	8.38	52	0	0	0
Ben Cutting	9	6	2	96	37	24	58	165.51	0	0	5	8	9	7	17	168	2	0	84	9.88	51	0	0	0
Ben Stokes	13	13	1	196	45	16.33	161	121.73	0	0	13	6	13	12	37	303	8	0	37.87	8.18	27.75	0	0	0
Bhuvneshwar Kumar	12	4	2	13	7	6.5	16	81.25	0	0	1	0	12	12	46.1	354	9	0	39.33	7.66	30.77	0	0	0
Brendon McCullum	6	6	0	127	43	21.16	88	144.31	0	0	16	6	0	0	0	0	0	0	0	0	0	0	0	0
Carlos Brathwaite	4	4	1	75	43	25	48	156.25	0	0	1	8	4	4	10.1	94	5	0	18.8	9.24	12.2	0	0	0
Chris Gayle	11	11	2	368	104	40.88	252	146.03	1	3	30	27	0	0	0	0	0	0	0	0	0	0	0	0
Chris Lynn	16	16	1	491	74	32.73	377	130.23	0	3	56	18	0	0	0	0	0	0	0	0	0	0	0	0
Chris Morris	4	4	3	46	27	46	26	176.92	0	0	3	2	4	4	14	143	3	0	47.66	10.21	28	0	0	0
Chris Woakes	5	4	2	17	11	8.5	19	89.47	0	0	1	1	5	5	18.2	190	8	0	23.75	10.36	13.75	0	0	0
Colin de Grandhomme	9	8	3	131	40	26.2	84	155.95	0	0	4	10	9	7	15	129	2	0	64.5	8.6	45	0	0	0
Colin Munro	5	5	0	63	33	12.6	41	153.65	0	0	7	4	0	0	0	0	0	0	0	0	0	0	0	0
Corey Anderson	3	3	0	17	15	5.66	22	77.27	0	0	0	1	3	3	8.4	115	3	0	38.33	13.26	17.33	0	0	0
D'Arcy Short	7	7	0	115	44	16.42	99	116.16	0	0	11	5	7	2	3	19	1	0	19	6.33	18	0	0	0
Dan Christian	4	3	1	26	13	13	33	78.78	0	0	0	1	4	4	11.5	101	4	0	25.25	8.53	17.75	0	0	0
David Miller	3	3	1	74	26	37	64	115.62	0	0	3	2	0	0	0	0	0	0	0	0	0	0	0	0
Deepak Chahar	12	4	1	50	39	16.66	29	172.41	0	0	1	4	12	12	38.1	278	10	0	27.8	7.28	22.9	0	0	0
Deepak Hooda	9	8	4	87	32	21.75	81	107.4	0	0	2	3	9	2	3	24	0	0	-	8	-	0	0	0
Dinesh Karthik	16	16	6	498	52	49.8	337	147.77	0	2	49	16	0	0	0	0	0	0	0	0	0	0	0	0
Dwayne Bravo	16	10	6	141	68	35.25	91	154.94	0	1	8	10	16	16	53.3	533	14	0	38.07	9.96	22.92	0	0	0
Evin Lewis	13	13	0	382	65	29.38	276	138.4	0	2	32	24	0	0	0	0	0	0	0	0	0	0	0	0
Faf du Plessis	6	6	1	162	67	32.4	129	125.58	0	1	17	6	0	0	0	0	0	0	0	0	0	0	0	0
Gautam Gambhir	6	5	0	85	55	17	88	96.59	0	1	8	1	0	0	0	0	0	0	0	0	0	0	0	0
Glenn Maxwell	12	12	0	169	47	14.08	120	140.83	0	0	14	9	12	10	16	132	5	0	26.4	8.25	19.2	0	0	0
Harbhajan Singh	13	3	0	29	19	9.66	36	80.55	0	0	3	1	13	12	31.5	270	7	0	38.57	8.48	27.28	0	0	0
Hardik Pandya	13	13	4	260	50	28.88	195	133.33	0	1	20	11	13	13	42.4	381	18	0	21.16	8.92	14.22	0	0	0
Harshal Patel	5	2	1	60	36	60	33	181.81	0	0	1	6	5	5	17.3	167	7	0	23.85	9.54	15	0	0	0
Heinrich Klaasen	4	4	1	57	32	19	47	121.27	0	0	5	1	0	0	0	0	0	0	0	0	0	0	0	0
Ishan Kishan	14	12	0	275	62	22.91	184	149.45	0	2	22	17	0	0	0	0	0	0	0	0	0	0	0	0

Ako vyzerá súbor údajov

## Vyhodnotenie

V algoritmoch hlbokého učenia sa metóda k-means niekedy nepoužíva na svoj priamy účel (klasifikácia zhlukovaním), ale na vytvorenie tzv. filtrov (konvolučných jadier, slovníkov). Napríklad na rozpoznávanie obrazu sa v algoritme k-means privádzajú malé náhodné kúsky obrázkov tréningovej vzorky, povedzme o veľkosti 16x16 v podobe lineárneho vektora, ktorého každý prvok kóduje jas svojho bodu. Počet k zhlukov je nastavený ako veľký, napr. 256. Za určitých podmienok natrénovaná metóda k-means vytvára centrá zhlukov (centroidy), ktoré sú vhodnými základmi, na ktorých možno rozložiť akýkoľvek vstupný obraz. Takéto "natrénované" centroidy sa potom používajú ako filtre, napr. pre konvolučné neurónové siete ako konvolučné jadrá alebo iné podobné systémy strojového videnia. Týmto spôsobom sa vykonáva učenie bez učiteľa pomocou metódy k-means. [3]

Keďže tento algoritmus používa neanotované dáta, nemôžeme tu vypočítať F1, presnosť atď.

Algoritmus má tiež niekoľko problémov:

- Nie je zaručené dosiahnutie globálneho minima celkovej kvadratickej odchýlky V, ale len jedného z lokálnych miním.
- Výsledok závisí od voľby počiatočných centier zhlukov, ich optimálna voľba nie je známa.
- Počet zhlukov musí byť známy vopred (ak sa nemýlim, zisťuje sa to ramennou metódou). [4] [5]

```

=====
 0 10,0 9,0 1,0 134,0 46,0 16,8 100,0 134,0 0,0 0,0 6,0 8,0 0,0 0,0 0,0
20 3,0 3,0 0,0 17,0 15,0 5,7 22,0 77,3 0,0 0,0 0,0 1,0 3,0 3,0 8,4
28 6,0 6,0 1,0 162,0 67,0 32,4 129,0 125,6 0,0 1,0 17,0 6,0 0,0 0,0 0,0
34 4,0 4,0 1,0 57,0 32,0 19,0 47,0 121,3 0,0 0,0 5,0 1,0 0,0 0,0 0,0
35 14,0 12,0 0,0 275,0 62,0 22,9 184,0 149,4 0,0 2,0 22,0 17,0 0,0 0,0 0,0
41 13,0 12,0 0,0 301,0 54,0 25,1 221,0 136,2 0,0 2,0 23,0 13,0 0,0 0,0 0,0
47 14,0 13,0 3,0 252,0 47,0 25,2 186,0 135,5 0,0 0,0 16,0 11,0 0,0 0,0 0,0
53 2,0 2,0 0,0 18,0 14,0 9,0 12,0 150,0 0,0 0,0 3,0 0,0 2,0 2,0 5,0
59 9,0 9,0 0,0 245,0 65,0 27,2 160,0 153,1 0,0 2,0 27,0 10,0 0,0 0,0 0,0
69 14,0 14,0 2,0 286,0 94,0 23,8 215,0 133,0 0,0 2,0 25,0 12,0 0,0 0,0 0,0
71 15,0 15,0 1,0 441,0 92,0 31,5 320,0 137,8 0,0 3,0 30,0 19,0 0,0 0,0 0,0
80 13,0 11,0 5,0 203,0 57,0 33,8 139,0 146,0 0,0 1,0 22,0 5,0 0,0 0,0 0,0
83 14,0 14,0 0,0 512,0 72,0 36,6 384,0 133,3 0,0 4,0 61,0 16,0 0,0 0,0 0,0
85 5,0 4,0 1,0 23,0 18,0 7,7 28,0 82,1 0,0 0,0 3,0 0,0 5,0 5,0 10,1
103 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 14,0 14,0 54,0
104 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 4,0 3,0 7,0
111 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 11,0 11,0 40,0
114 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 11,0 11,0 41,2
115 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 2,0 2,0 7,0
116 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 7,0 7,0 27,3
120 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 12,0 12,0 44,0
125 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 2,0 2,0 3,5
=====
 1 12,0 11,0 2,0 480,0 90,0 53,3 275,0 174,5 0,0 6,0 39,0 30,0 0,0 0,0 0,0
 9 9,0 6,0 2,0 96,0 37,0 24,0 58,0 165,5 0,0 0,0 5,0 8,0 9,0 7,0 17,0
14 11,0 11,0 2,0 368,0 104,0 40,9 252,0 146,0 1,0 3,0 30,0 27,0 0,0 0,0 0,0
15 16,0 16,0 1,0 491,0 74,0 32,7 377,0 130,2 0,0 3,0 56,0 18,0 0,0 0,0 0,0
24 12,0 4,0 1,0 50,0 39,0 16,7 29,0 172,4 0,0 0,0 1,0 4,0 12,0 12,0 38,1
37 15,0 7,0 3,0 49,0 26,0 12,2 38,0 128,9 0,0 0,0 6,0 1,0 15,0 15,0 50,2
42 9,0 8,0 1,0 133,0 50,0 19,0 100,0 133,0 0,0 1,0 10,0 7,0 0,0 0,0 0,0
43 15,0 13,0 4,0 126,0 33,0 14,0 64,0 196,9 0,0 0,0 9,0 9,0 15,0 15,0 40,0
48 15,0 13,0 2,0 284,0 62,0 25,8 246,0 115,4 0,0 3,0 22,0 5,0 0,0 0,0 0,0
51 14,0 6,0 4,0 21,0 7,0 10,5 24,0 87,5 0,0 0,0 2,0 0,0 14,0 14,0 44,0
54 11,0 4,0 2,0 25,0 14,0 12,5 22,0 113,6 0,0 0,0 2,0 1,0 11,0 11,0 41,0
66 4,0 4,0 0,0 29,0 16,0 7,2 31,0 93,5 0,0 0,0 4,0 0,0 0,0 0,0 0,0
74 15,0 15,0 1,0 555,0 117,0 39,6 359,0 154,6 2,0 2,0 44,0 35,0 15,0 11,0 28,0
88 7,0 6,0 3,0 65,0 35,0 21,7 38,0 171,1 0,0 0,0 5,0 4,0 7,0 7,0 20,0
96 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 4,0 4,0 10,1
99 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 3,0 3,0 10,0
100 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 8,0 8,0 19,5
102 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 6,0 6,0 23,0
109 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 7,0 7,0 25,0
123 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 14,0 14,0 52,4
126 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 0,0 14,0 14,0 50,0
=====

```

Konečný výsledok zhlukovania

## Odkazy

- [1] M. J., Some methods for classification and analysis of multivariate observations., 1967.
- [2] @aliraza, *Indian Premier League 2018 Batting and Bowling data*, 2018.
- [3] A. C. a. A. Y. Ng., *Learning Feature Representations with K-means*, Stanford University, 2012.
- [4] „wikipedia.org,” [Online]. Available:  
[https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4\\_k-%D1%81%D1%80%D0%B5%D0%B4%D0%BD%D0%B8%D1%85#cite\\_ref-6](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_k-%D1%81%D1%80%D0%B5%D0%B4%D0%BD%D0%B8%D1%85#cite_ref-6).
- [5] P. prof. Ing. Kristína Machová, „Zhukovanie,” Košice.