A Bag-of-Features Algorithm for Applications Using a NoSQL Database

Marcin Gabryel(✉)

Institute of Computational Intelligence, Częstochowa University of Technology,
Al. Armii Krajowej 36, 42-200 Częstochowa, Poland
marcin.gabryel@iisi.pcz.pl

Abstract. In this paper we present a Bag-of-Words (also known as a Bag-of-Features) method developed for the use of its implementation in NoSQL databases. When working with this algorithm special attention was brought to facilitating its implementation and reducing the number of computations to a minimum so as to use what the database engine has to offer to its maximum. The algorithm is presented using an example of image storing and retrieving. In this case it proves necessary to use an additional step of preprocessing, during which image characteristic features are retrieved and to use a clustering algorithm in order to create a dictionary. We present our own k-means algorithm which automatically selects the number of clusters. This algorithm does not comprise any computationally complicated classification algorithms, but it uses the majority vote method. This makes it possible to significantly simplify computations and use the Javascript language used in a common NoSQL database.

Keywords: NoSQL database · Image classification · Bag-of-Features · Modified *k*-means algorithm

1 Introduction

Relational databases enjoy a considerable popularity and have been used for a number of decades now. Their main advantages include durability of data storage, transaction processing, relational model, error handling and the SQL language. However, in distributed systems and in the case of data with different structure so-called NoSQL databases are used much more frequently. For Big Data processing special databases are developed, which work on special file systems supported for instance in Hadoop and dedicated frameworks for fast and parallel processing (e.g. MapReduce). NoSQL databases are different from commonly-used relational databases (RDBMS) in terms of the following aspects: not using the SQL language, not having to follow the ACID model (Atomicity, Consistency, Isolation, Durability), and also having no relationships and tables of a defined structure.

One of the tasks in which a NoSQL database can be used is effective browsing and searching a large number of images. NoSQL databases can successfully store enormous amounts of data including image data. In solving image processing and retrieval problems algorithms from different fields of computational intelligence are used [18, 22, 23, 25], in particular fuzzy systems [14, 15], rough neuro-fuzzy systems [16, 17, 26],

G. Dregvaite and R. Damasevicius (Eds.): ICIST 2016, CCIS 639, pp. 332–343, 2016.
DOI: 10.1007/978-3-319-46254-7_26

evolutionary algorithms [20, 24], swarm intelligence [27–29], mathematics [21, 30, 31], decision tree [19] and data mining [32, 33]. One of the most popular and widely spread algorithms used for indexation and image retrieval is the bag-of-words model (BoW) [4, 5], known also as a Bag-of-Features (BoF) or Bag-of-Visual-Words. This algorithm is based on a concept of text search methods within collections of documents. Single words are stored in dictionaries with emphasis on appearing in various documents. The BoW in a similar way creates dictionaries of characteristic features appearing in images. Additionally, the classification process enables during the search to determine what type of image class we are dealing with.

Practical aspects of the BoF algorithm implementation in image classification are rather rare in the literature. There are many modifications of this algorithm which, for example, use various image features [6–8] or various clustering and classification algorithms [9, 10], but there are no examples of practical applications of this algorithm. Most simulations and experiments are carried out with the use of OpenCV library, Matlab environment or multi-core processors. In practice direct usage of this particular kind of algorithms is connected with considerable computing capacities required when using classification algorithms and having no information concerning using data bases. The possibility of using a NoSQL database allows us to make a quite simple use of a number of computers to store large amount of data and do parallel computing. In most cases parallelism can be successfully carried out on a database which has been properly managed.

The article is divided into a few sections. Section 2 outlines the algorithms of which the whole image storing system. Section 3 presents the results of the experimental research testing the efficiency of the presented algorithms as well as the details connected with the NoSQL database being used for the implementation of this method. The conclusions in the last section present ideas concerning further improvement of the system efficiency.

2 Description of Algorithms

We are considering herein a set of given images $\mathbf{I}_i$, where $i = 1, \ldots, \mathbf{I}_M$ and M is the number of all images. Each image $\mathbf{I}_i$ has a class $c(\mathbf{I}_i)$ assigned to it, where $c(\mathbf{I}_i) \in \Omega$, $\Omega = \{\omega_1, \ldots, \omega_C\}$ is a set of all classes and C is the number of all classes. The images $\mathbf{I}_i$ make the initial data which will be stored in the NoSQL database and will be used to create a dictionary for the BoF method (see Sect. 2.3). K-means algorithm (see Sect. 2.2) groups characteristic features retrieved from an image concurrently reducing their number and creates words included in the dictionary. The modification of the k-means algorithm which we introduce allows for an automatic selection of the number of groups. Image characteristic features are obtained as a result of the operation of the SURF algorithm. (see Sect. 2.1).

2.1 SURF

SURF (Speeded Up Robust Features) is a robust local feature detector, first presented in [1]. It is partly inspired by the SIFT descriptor [2]. SURF gives description of an image by selecting its characteristic features. First, an integral image and filter approximation of block Hessian determinant is applied. Next, to detect interesting points, a special Hessian-matrix approximation is used. For features, orientation is based on information from circular region around the pixel. Then, a square region aligned to selected orientation is constructed and the SURF descriptor is extracted from it. It uses the sum of the Haar wavelet responses around an interest point. The local feature around the point is described by a 64-number vector $\mathbf{x} = [x_1, \ldots, x_{64}]$.

2.2 Modified k-Means Algorithm

The *k*-means algorithm is the most frequently used clustering algorithm used in the BoF. Its only drawback involves having to define the initial number of classes c. In this section we present an automatic selection mechanism of the number of classes during the operation of this algorithm. We have used the growing method used in the Growing Self-Organizing Map (GSOM) algorithm [3]. In that method a cluster is divided when the number of its data exceeds a certain threshold value Θ. Operation of the said algorithm starts with setting the threshold value Θ and defining two clusters ($c = 2$). In the subsequent steps the algorithm works as a classic *k*-means with the only difference being that at the end of each iteration the number of points belonging to each cluster $\tau_j, j = 1, \ldots, c$ is checked. If the number τ_j exceeds the threshold already set at Θ, then another cluster $c + 1$ is created. The algorithm is presented below in detail.

Let $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_n$ be a set of points in d-dimensional space, and $\mathbf{V} = \mathbf{v}_1, \ldots, \mathbf{v}_c$ be cluster centers, where n is the number of samples, $\mathbf{x}_i = [x_{i1}, \ldots, x_{id}]$, c is the number of clusters, and $\mathbf{v}_j = [v_{j1}, \ldots, v_{jd}]$.

1. Let the number of cluster $c = 2$. Determine Θ.
2. Randomly select c cluster centers $\mathbf{v}_j$, $j = 1, \ldots, c$, for example:

$$v_{ji} = rand\left(\min\left(x_{ij}\right), \max\left(x_{ij}\right)\right), \tag{1}$$

   where $rand(a, b)$ is a random number generated from the interval $[a; b]$.

3. Calculate the distance d_{ij} between each data point $\mathbf{x}_i$ and cluster centers $\mathbf{v}_j$:

$$d_{ij} = \| \mathbf{x}_i - \mathbf{v}_j \|, \tag{2}$$

   where $\| \cdot \|$ is a distance measure between two vectors (e.g. Euclidian or Manhattan distance).

4. Assign the data point $\mathbf{x}_i$ to the cluster center $\mathbf{v}_s$ whose distance from the cluster center is a minimum of all the cluster centers

$$\mathbf{x}_i \in \mathbf{v}_s \rightarrow d_{is} \leq d_{im},\ m = 1, \ldots, c \tag{3}$$

and increase counter of winnings $\tau_s = \tau_s + 1$.

5. Recalculate the new cluster center using:

$$\mathbf{v}_i = \frac{1}{c_i} \Sigma_{j=1}^{c_i} \mathbf{x}_j, \tag{4}$$

where c_i represents the number of data points in i-th cluster.

6. If in the center s the number τ_s is greater than the threshold value Θ, create a new cluster, $c := c + 1$ and

$$\mathbf{v}_c = \mathbf{x}_{rand(j)}, \tag{5}$$

where $rand(j)$ generates a random index of point $\mathbf{x}$ belonging to center $\mathbf{v}_s$.

7. Remove clusters for which $\tau_s = 0$. Refresh the number of clusters c.
8. If no data point was reassigned, then stop; otherwise, repeat starting from step 3.

As a result of the algorithm operation we obtain c clusters with the centers in points $\mathbf{v}_j$, $j = 1, \ldots, c$.

2.3 The Bag-of-Features Algorithm

The classic Bag-of-Features algorithm used in image classification most frequently uses classifiers (e.g. Support Vector Machine – SVM is used) during the stage when the decision is being made on the image class. The BoF comprises several stages:

1. Generate of characteristic features from images, which are most frequently saved in the form of number vectors.
2. Characteristic features are clustered and obtained clusters are treated as words, which create a dictionary.
3. Words (cluster centers) to which characteristic features of a given image belong make a histogram. Each element in the histogram specifies how many times a given word is present in the histogram.
4. The classifier is learnt to recognise histograms and to assign particular classes to them.

The points listed above show that it is possible to store three groups of data in a database, i.e. image characteristic features, centers of the clusters found, and histograms presenting membership of features in clusters. After a classifier has been learnt, it no longer needs to have access to the database.

A problem occurs when a query is made. The query image needs to have its features assigned to specific clusters in such way so as to create a histogram and to have it classified. This process requires a large number of computations given in formula (2). Thus the data-storing cluster centers need to be taken from the database. We can use this situation to our advantage and instead of using an ordinary query taking required data we can use the database engine in order to do computations and classification concurrently. A complex classifier (e.g. the abovementioned SVM) can be successfully replaced by the majority vote. We present below a BoF algorithm version which consists of two modules (one module – preparing data and the other – the classification process), which has been created in order to be able to use a NoSQL database.

The first of the BoF algorithm modules is supposed to prepare the database by creating a dictionary of characteristic features of sample images (to be used in the system learning process). This is carried out in a few steps presented below.

1. Starting operation of the algorithm generating image characteristic features. In our case we have used a well-known and fast SURF algorithm [1] which provides 64-number vectors $\mathbf{x}_i = [x_{i1}, \ldots, x_{id}]$ describing the surrounding of a characteristic point, where $i = 1, \ldots, L$, L – the total number of all characteristic points, d – the dimension of the vector describing a characteristic point ($d = 64$).
2. Starting operation of the k-means clustering algorithm. We have used the algorithm version presented in Sect. 2.2. As a result we obtain c clusters with the centers in points $\mathbf{v}_j$, $j = 1, \ldots, c$, which are treated as words in the BoF dictionary.
3. The value of the number of classes i of cluster j is calculated and defined as k_{ji}. This value is computed by counting the points $\mathbf{x}_n$ which belong to the center j provided that $\mathbf{x}_n \in \mathbf{I}$ and $c(\mathbf{I}) = \omega_i$:

$$k_{ji} = \Sigma_{n=1}^{L} \delta_{nj}(i),\ j = 1, \ldots, c,\quad i = 1, \ldots, C, \tag{6}$$

Where

$$\delta_{nj}(i) = \begin{cases} 1 & \text{if } d_{nj} < d_{nm} \text{ for } \mathbf{x}_n \in \mathbf{I} \text{ and } c(\mathbf{I}) = \omega_i,\ m = 1, \ldots, c,\ j \neq m \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The variable $\delta_{nj}(i)$ is an indicator if a cluster $\mathbf{v}_j$ is the closest vector (a winner) for any sample $\mathbf{x}_n$ from an image $\mathbf{I}$ and $c(\mathbf{I}) = \omega_i$. Next, the values k_{ji} are normalised:

$$k_{ji} = \frac{k_{ji}}{\sum_{j=1,\ldots,c} k_{ji}} \tag{8}$$

4. Saving the values of centers $\mathbf{v}_j$ together with the information about the number of classes k_{ji} in the database.

The classification process, i.e. the process testing whether a given query image belongs to a particular class, requires that an additional pre-processing module be applied. This module is supposed to:

1. Use a feature extraction algorithm (the SURF algorithm – see Sect. 2.1) on query image $\mathbf{I}_q$ in order to obtain values of characteristic features $\mathbf{x}_i^q$, $i = 1, \ldots, L_q$, L_q – the number of obtained features.
2. Save points $\mathbf{x}^q$ in the database.
3. Assign points $\mathbf{x}^q$ to clusters in such way so as to compute values k_i^q. Computations are carried out as follows:

$$k_i^q = \Sigma_{n=1}^{L_q} \alpha_n(i),\ i = 1, \ldots, C \tag{9}$$

$$\alpha_n(i) = \begin{cases} k_{ji} & \text{if } d_{jn}^q \leq d_{mn}^q,\ j \neq m \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Where

$$d_{jn}^q = \parallel \mathbf{v}_j - \mathbf{x}_n^q \parallel,\ j = 1, \ldots, c \tag{11}$$

and

$$d_{mn}^q = \parallel \mathbf{v}_m - \mathbf{x}_n^q \parallel,\ m = 1, \ldots, c. \tag{12}$$

4. Assigning to class $c(\mathbf{I}_q)$ is done by the majority vote checking the maximum value k_i^q:

$$c(\mathbf{I}_q) = \text{argmax}_{i=1,\ldots,c}\, k_i^q \tag{13}$$

The algorithm facilitates an easy implementation by using only one SQL query. The algorithm's details and experimental research are presented in Sect. 3.

3 Experimental Research

In this section we present the results of the BoF algorithm discussed in Sect. 2.3 together with the modified k-means algorithm (outlined in Sect. 2.2). Practical application of the presented method in image classification with the use of the database needs two modules to be used:

- the module which is supposed to fill with the use of the Bag-of-Features algorithm the database with the cluster center values and also with the information on which class they belong to, i.e. creating the dictionary. The module creating the dictionary for the database should comprise the following steps:
  - preprocesing, i.e. extracting characteristic features from training images,
  - starting the operation of the *k*-means algorithm in order to perform clustering of the data and obtaining clusters with image characteristic features,
  - save the above data in the database;
- the module preparing for query image classification and saving its features in the database. It also comprises a few steps:
  - preprocesing, i.e. extracting characteristic features from a query image,
  - saving data in the database,
  - performing a query which will produce information on the class to which a particular image belongs.

Preprocessing algorithms were implemented in the Java language with the use JavaCV [11] library function. JavaCV is a library which adopts functions available in OpenCV [12] for the Java language needs. The research was performed on the Caltech 101 image database [13]. Six sample categories comprising motorbikes, car sides, revolvers, airplanes, leopards and wrenches were selected. Out of the remaining group of images, 20 % are randomly selected and marked as a set of testing images. During the operation of the SURF algorithm over 100,000 characteristic points are identified in the database for 180 images. The main value which is used to compute classification efficiency is a percentage of correctly classified images.

Three structures are used in data storing: learning image features, testing image features, and for values of the cluster centres. Structures are stored in collections:

- `train_images` and `eval_images` – collections storing the features values of learning and testing images have the following values:
  `imageId` – image identifier, file name,
  `classId` – image class,
  `pk1, pk2, ..., pk64` – values of the feature vector obtained as a result of the SURF algorithm operation.
- `centers_500` – the collection stores cluster centers and k_{ji} values (see formula (8)):
  `center_id` – cluster identification,
  `c1, c2, ..., c64` – cluster center values,
  `k1,...,k6` – k_{ji} values for each of the class.

The script performing the query image classification comprises two stages. The `centers_temporary` collection is created during the first stage and it stores temporary results of computations checking membership of query image features in

particular clusters. Particularly, values $\alpha_n(i)$ are computed here (according to formula (10)) and values k_{ji} from cluster j that are closest to the sample n are returned for all classes i. The script made for a commonly-used MongoDB database presents as follows:

```
db.centers_temporary.drop();
db.eval_images.find( { imageId : IMAGE_ID }, {_id:0}
).forEach(
  function( feature ) {
    var minDistance = { mm : -1 };
    db.centers_500.find().forEach(
      function( center ) {
        var oneDistance = { mm : 0,
          imageId : feature.imageId,
          class: [ center.k0, center.k1,
                   center.k2, center.k3,
                   center.k4, center.k5],
          cluster_id : center.cluster_id };
        oneDistance.mm =
          Math.abs(feature.pk0 - center.c0) +
          Math.abs(feature.pk1 - center.c1) +
          Math.abs(feature.pk2 - center.c2) +
          ...
          Math.abs(feature.pk63 - center.c63);
        if ( minDistance.mm > oneDistance.mm ||
             minDistance.mm == -1  ) {
          minDistance = oneDistance;
        }
      }
    );
    db.centers_temporary.insert( minDistance );
  }
);
```

The next step is to call the `mapReduce` function, which is supposed to compute for each class the k_i^q value (according to formula (9)) and to classify an image by applying the majority vote method according to (13):

```
db.centers_temporary.mapReduce(
  function() {
    classId = 0;
    for (var i=1; i<this.class.length; i++) {
      if (this.class[classId] < this.class[i]) {
        classId = i;
      }
    }
    emit( classId, this.class[classId] );
  },

  function (key, values) {
    return Array.sum( values );
  },
  {
    query: { imageId: IMAGE_ID },
    out: "outputResults"
  }
);
```

Table 1 presents the results of the operation of the algorithm for different Θ values (the parameter for the *k*-means method). The other columns present: Θ threshold value, the number of obtained clusters as a result of the operation of the *k*-means algorithm, and also the efficiency (given in percentages) of image recognition accuracy for the training and testing groups. As it can be noticed, the results proved best for $\theta = 50$ and $\theta = 100$. However, as far as the number of obtained clusters is concerned, value $\theta = 100$ proves to be the best choice under this research.

Table 1. Percentage efficiency of image classification for the presented algorithm in relation to θ threshold value.

Threshold Θ	Number of clusters	Train [%]	Test [%]
10000	38	45.50	46.85
5000	106	49.38	48.95
2000	852	60.49	53.84
1000	1958	68.61	63.63
500	3120	76.89	74.12
250	4077	80.77	73.42
100	7532	90.47	74.82
50	14591	94.70	74.82
25	28752	97.88	72.02
10	51915	99.83	71.32

4 Conclusions

In this paper we present a Bag-of-Features algorithm which we have developed. This algorithm works on the basis of a NoSQL database. We have presented the algorithm implementation on the example of image storage and classification. We have modified the classic Bag-of-Words algorithm so as to facilitate its implementation in a NoSQL database without compromising its efficiency. The tests which we have carried out show that the algorithm operates correctly, thus proving its efficiency. Although operation of the BoF algorithm cannot be compared to other image-recognition algorithms (e.g. a deep-learning network), it still proves efficient enough and simple in its implementation so that it can be successfully used in applications working on less sophisticated hardware. Better query efficiency can be achieved by means of reducing lengths of the vectors describing particular image features. The methods which reduce the number of dimensions can be used for this purpose and these methods, for instance, include the PCA algorithm, or some other completely different feature generating algorithms. Another possible procedure reducing the number of calculations is the exclusions of those clusters which are the least likely to affect unequivocally image class determination.

References

1. Bay, H., Tuytelaars, T., Van Gool, L.: SURF: speeded up robust features. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006, Part I. LNCS, vol. 3951, pp. 404–417. Springer, Heidelberg (2006)
2. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision **60**(2), 91–110 (2004)
3. Fritzke, B.: Growing grid a self-organizing network with constant neighbourhood range and adaptation strength. Neural Process. Lett. **2**(5), 9–13 (1995)
4. Csurka, G., Dance, C.R., Fan, L., Willamowski, J., Bray, C.: Visual categorization with bags of keypoints. In: Workshop on Statistical Learning in Computer Vision, ECCV, pp. 1−22 (2004)
5. Liu, J.: Image retrieval based on bag-of-words model. CoRR abs/1304.5168 (2013). http://arxiv.org/abs/1304.5168
6. Lazebnik, S., Schmid, C., Ponce, J.: Beyond bags of features: spatial pyramid matching for recognizing natural scene categories. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2, pp. 2169−2178 (2006)
7. Li, W., Dong, P., Xiao, B., Zhou, L.: Object recognition based on the region of interest and optimal bag of words model. Neurocomputing **172**, 271–280 (2016)
8. Nanni, L., Melucci M.: Combination of projectors, standard texture descriptors and bag of features for classifying images. Neurocomputing **173**(P3), 1602–1614 (2016)
9. Gao, H., Dou, L., Chen, W., Sun, J.: Image classification with bag-of-words model based on improved sift algorithm. In: 2013 9th Asian Control Conference (ASCC), pp. 1−6 (2013)
10. Zhao, C., Li, X., Cang, Y.: Bisecting k-means clustering based face recognition using block-based bag of words model. Optik – Int. J. Light Electron Opt. **126**(19), 1761–1766 (2015)
11. Audet, S.: JavaCV (2016). http://bytedeco.org/. Accessed 1 Apr 2016

12. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
13. Fei-Fei, L., Fergus, R., Perona, P.: Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In: 2004 Conference on Computer Vision and Pattern Recognition Workshop, CVPRW 2004, p. 178, June 2004
14. Cpalka, K.: A new method for design and reduction of neuro-fuzzy classification systems. IEEE Trans. Neural Netw. **20**(4), 701–714 (2009)
15. Starczewski, J.T.: Centroid of triangular and gaussian type-2 fuzzy sets. Inf. Sci. **280**, 289–306 (2014)
16. Nowak, B.A., Nowicki, R.K., Starczewski, J.T., Marvuglia, A.: The learning of neuro-fuzzy classifier with fuzzy rough sets for imprecise datasets. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2014, Part I. LNCS, vol. 8467, pp. 256–266. Springer, Heidelberg (2014)
17. Nowicki, R.: Rough sets in the neuro-fuzzy architectures based on monotonic fuzzy implications. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 510–517. Springer, Heidelberg (2004)
18. Sakurai, S., Nishizawa, M.: A new approach for discovering top-k sequential patterns based on the variety of items. J. Artif. Intell. Soft Comput. Res. **5**(2), 141–153 (2015)
19. Tambouratzis, T., Souliou, D., Chalikias, M., Gregoriades, A.: Maximising accuracy and efficiency of traffic accident prediction combining information mining with computational intelligence approaches and decision trees. J. Artif. Intell. Soft Comput. Res. **4**(1), 31–42 (2014)
20. El-Samak, A.F., Ashour, W.: Optimization of traveling salesman problem using affinity propagation clustering and genetic algorithm. J. Artif. Intell. Soft Comput. Res. **5**(4), 239–245 (2015)
21. Woźniak, M., Kempa, W.M., Gabryel, M., Nowicki, R.K.: A finite-buffer queue with single vacation policy - analytical study with evolutionary positioning. Int. J. Appl. Math. Comput. Sci. **24**(4), 887–900 (2014)
22. Gabryel, M., Grycuk, R., Korytkowski, M., Holotyak, T.: Image indexing and retrieval using GSOM algorithm. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2015. LNCS, vol. 9119, pp. 706–714. Springer, Heidelberg (2015)
23. Grycuk, R., Gabryel, M., Korytkowski, M., Scherer, R., Voloshynovskiy, S.: From single image to list of objects based on edge and blob detection. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2014, Part II. LNCS, vol. 8468, pp. 605–615. Springer, Heidelberg (2014)
24. Gabryel, M., Woźniak, M., Damaševičius, R.: An application of differential evolution to positioning queueing systems. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2015. LNCS, vol. 9120, pp. 379–390. Springer, Heidelberg (2015)
25. Nowak, B.A., Nowicki, R.K., Woźniak, M., Napoli, C.: Multi-class nearest neighbour classifier for incomplete data handling. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2015. LNCS, vol. 9119, pp. 469–480. Springer, Heidelberg (2015)
26. Nowicki, R.K., Nowak, B.A., Woźniak, M.: Application of rough sets in k nearest neighbours algorithm for classification of incomplete samples. In: Kunifuji, S., Papadopoulos, G.A., Skulimowski, A.M.J., Kacprzyk, J. (eds.) KICSS 2014. AISC, vol. 416, pp. 243–257. Springer, Heidelberg (2016)

27. Połap, D., Woźniak, M., Napoli, C., Tramontana, E.: Real-time cloud-based game management system via cuckoo search algorithm. Int. J. Electron. Telecommun. **61**(4), 333–338 (2015)
28. Połap, D., Woźniak, M., Napoli, C., Tramontana, E.: Is swarm intelligence able to create mazes? Int. J. Electron. Telecommun. **61**(4), 305–310 (2015)
29. Woźniak, M., Gabryel, M., Nowicki, R.K., Nowak, B.A.: An application of firefly algorithm to position traffic in NoSQL database systems. In: Kunifuji, S., Papadopoulos, G.A., Skulimowski, A.M.J., Kacprzyk, J. (eds.) KICSS 2014. AISC, vol. 416, pp. 259–272. Springer, Heidelberg (2016)
30. Woźniak, M., Marszałek, Z., Gabryel, M., Nowicki, R.K.: Preprocessing large data sets by the use of quick sort algorithm. In: Skulimowski, A.M.J., Kacprzyk, J. (eds.) KICSS 2013. AISC, vol. 364, pp. 111−121. Springer, Heidelberg (2016)
31. Woźniak, M., Marszałek, Z., Gabryel, M., Nowicki, R.K.: Modified merge sort algorithm for large scale data sets. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2013, Part II. LNCS, vol. 7895, pp. 612–622. Springer, Heidelberg (2013)
32. Rutkowski, L., Jaworski, M., Pietruczuk, L., Duda, P.: Decision trees for mining data streams based on the Gaussian approximation. IEEE Trans. Knowl. Data Eng. **26**(1), 108–119 (2014)
33. Rutkowski, L., Jaworski, M., Pietruczuk, L., Duda, P.: A new method for data stream mining based on the misclassification error. IEEE Trans. Neural Networks Learn. Syst. **26**(5), 1048–1059 (2015)