sciendo

DOI:10.2478/ijasitels-2019-0007

DBSCAN ALGORITHM FOR DOCUMENT CLUSTERING

Radu G. CREŢULESCU[1], Daniel I. MORARIU[1], Macarie BREAZU[1], Daniel VOLOVICI[1],

[1] „Lucian Blaga" University of Sibiu, Engineering Faculty, Computer Science and Electrical and Electronics Engineering Department

Abstract

Document clustering is a problem of automatically grouping similar document into categories based on some similarity metrics. Almost all available data, usually on the web, are unclassified so we need powerful clustering algorithms that work with these types of data. All common search engines return a list of pages relevant to the user query. This list needs to be generated fast and as correct as possible. For this type of problems, because the web pages are unclassified, we need powerful clustering algorithms. In this paper we present a clustering algorithm called DBSCAN – Density-Based Spatial Clustering of Applications with Noise – and its limitations on documents (or web pages) clustering. Documents are represented using the "bag-of-words" representation (word occurrence frequency). For this type o representation usually a lot of algorithms fail. In this paper we use Information Gain as feature selection method and evaluate the DBSCAN algorithm by its capacity to integrate in the clusters all the samples from the dataset.

Keywords: Document Classification, Information Gain, Naive Bayes, Weka framework

1 Introduction

As storage capacity increases, the amount of information saved increases too and become more and more difficult to retrieve and use the saved information. We need more powerful methods that become capable to process this huge quantity of information and offer us an easy and fast access to this information. The text document clustering problem is a special case of an unsupervised learning process in the data mining problem. In order to solve a text document clustering problem some steps are required. The common steps are [6]: feature extraction, feature selection, grouping, evaluation and visualization. The WEKA [9] is a framework that helps us with all these steps. WEKA was initially developed as a library of java classes that help us to implement data mining applications. In the last years, in order to avoid java programming skills, the components from WEKA are also available into a visual form inside the "WEKA Knowledge Flow Environment"..

2 Experimental framework

2.1 Dataset

In order to make some experiments to validate the algorithm functionality we use a Reuters dataset [8], that is a collection of news published by Reuters agency into an XML format, that is close to a text file format. For analyzing and evaluating the learning algorithm we use the Weka learning application that has a lot of learning algorithms already implemented and prepared to be used. In the first part we need to convert the Reuters XML files into a format accepted by Weka, and we need to make lot of steps (classical text mining preprocessing steps [4],[5]) as word extraction, eliminating the common words, keep only the stem of the word and feature selection. We have represented the documents in a vector space model as frequency of word occurrences in document. All the preprocessing steps for transforming the dataset from XML into a Weka format (called arff format) were done into a proper implemented java application. In the resulting file each document is represented on a line as a vector of 1000 different attributes (words). Because Weka has some problems using large numbers of vectors and vectors having a great number of elements, we decide to use only 542 different samples (documents) represented by most relevant 1000 features. The arff format that must be applied to WEKA contains a list with all attributes used into the dataset (defined with name and type as in the next example) and after the "@data" directive it contains a list of each sample (one on a line) with values for each attribute. We prefer the last attribute to be the class (yes/no - if the document is assigned into a class or not). Each sample is classified into a single class (most relevant from the Reuters proposed topic) and we decide to learn only one class. So, for all samples from the dataset that are in that specific class we write 'yes' and for the rest of the samples we write 'no' for the class attribute.

The format for the arff file is as follows:

```
@relation Reuters
@attribute 'A0' numeric
@attribute 'A1' numeric
….
@attribute 'A998' numeric
@attribute 'A999' numeric
@attribute 'class' {'yes', 'no'}
@data
2,2,1,1,1,…,0,0,0,0,0,no
0,0,1,0,1,…,0,2,0,1,0,yes
…
```

In the clustering algorithm we don't need to use the class value that are present in the file in the learning step. We use those values in the feature selection step in order to apply evaluation method for obtain most relevant attributes.

2.2 Weka

For training and evaluating the presented dataset we use the Weka KnowledgeFlow Environment [9],[3] and the experiment uses 6 Weka modules as shown in Figure 1.

The 'ArffLoader' module permits us to load the prepared arff file with the Reuters represented documents. Even if we select the most relevant 1000 features in the document representation step, in this experiment we use a different number of features between 100 to 1000 for evaluating the accuracy of the DBScan algorithm. For modifying the number of features we use a 'Attribute Selection' module that has implemented 'Information Gain' and a method for selecting relevant features.

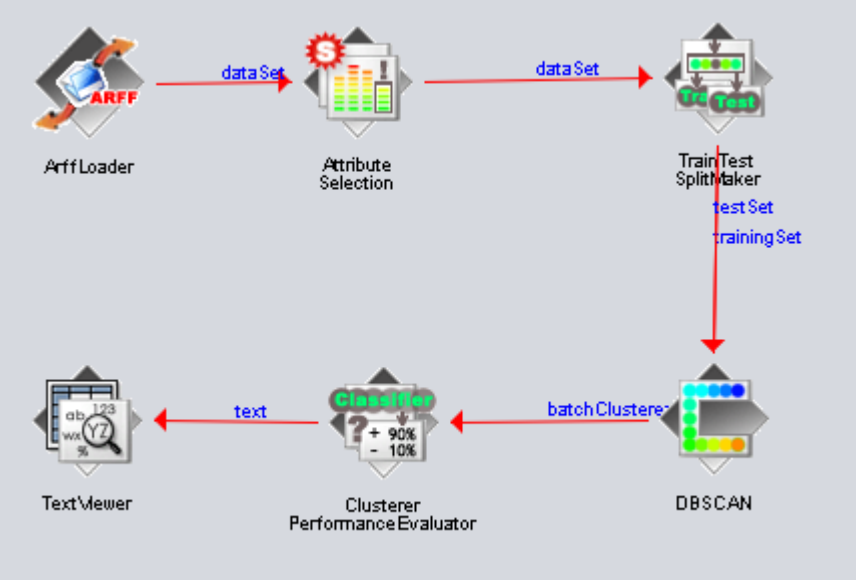

Figure 1. DBScan Experiment in KnowledgeFlow Weka environment

After selection we use a 'TrainTestSplitMarker' that splits the dataset in two parts, one part for training that has 66% of the dataset and the rest for the test.

The 'DBScan' module contains the algorithm that we want to evaluate in this article for the document clustering context. We have some parameters to be specified before running the algorithm: the minimum number of points, the method for distance computing and the value for epsilon (as in figure 2).

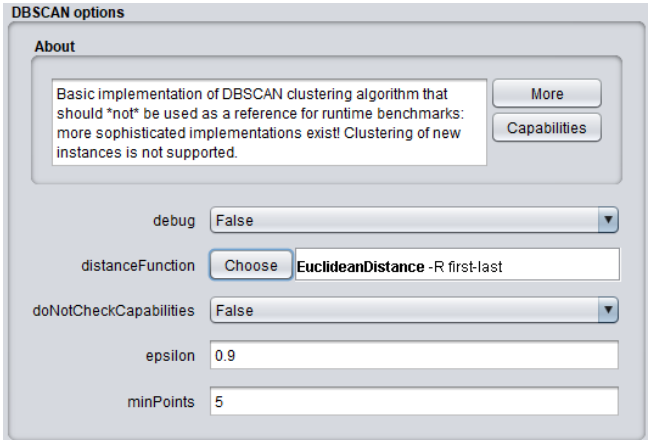

Figure 2. The DBScan configuration

The two last modules are for the cluster evaluation step. We evaluate the algorithm regarding the number of elements present in each class and the needed training time.

3 The clustering algorithm

3.1 DBSCAN (Density-Based Spatial Clustering) Algorithm

DBSCAN [4],[5] is a clustering algorithm based on finding *core objects* and creates a number of groups that are equal with the number of core objects found. A core object is an object that has dense neighborhoods. If the algorithm finds some objects with no dense neighborhoods these are considered noise. It connects core objects and their neighborhoods in order to form dense regions as clusters.

For the configuration of the application we need to be specify the parameter **ε** > 0 for the radius of a neighborhood that is considered for every object. The ε-neighborhood of an object *o* is the space within a radius centered in *o*. The density of a neighborhood can be measured by the number of objects in the neighborhood. Another user-specified parameter is *MinPts*. This parameter specifies the minimum density threshold for a dense region to be considered as core object. An object is considered to be a **core object** if the ε-neighborhood of the object contains at least *MinPts* objects.

For a core object *q* and an object *p*, we say that ***p is directly density-reachable from q*** if *p* is within the ε -neighborhood of *q*. In figure 3 are represented al symbols.

Density reachability respects following rules:

- *p* is density-reachable from *q* (with respect to *ε* and *MinPts* in dataset) if there is a chain of objects $p_1, \ldots, p_n$, such that $p_1 = q$, $p_n = p$, and p_{i+1} is directly density-reachable from p_i with respect to ε and *MinPts*;

- if o_1 and o_2 are core objects and o_1 is density-reachable from o_2, then o_2 is density-reachable from o_1;

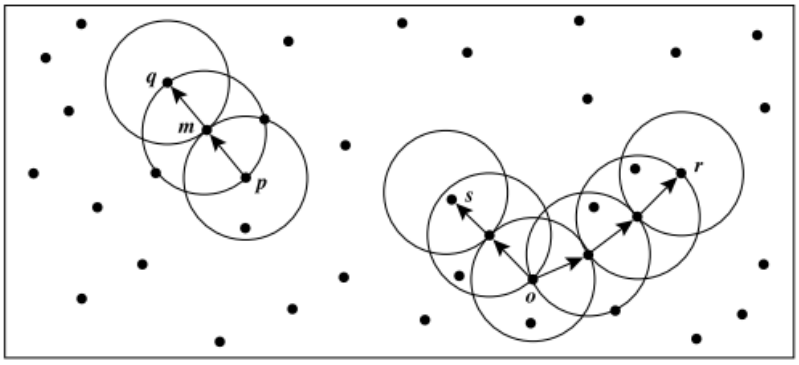

Figure 3 Density-reachability and density-connectivity (3)

- if o_2 is a core object but o_1 is not, then o_1 may be density-reachable from o_2, but only in this direction.

Density connectivity respects following rules:

- p_1, p_2 are density-connected with respect to ε and *MinPts* if there is an object q such that both p_1 and p_2 are density-reachable from q with respect to ε and *MinPts*;
- Unlike density-reachability, density-connectedness is an equivalence relation. If o_1 and o_2 are density-connected, and o_2 and o_3 are density-connected, then also o_1 and o_3 are density-connected.

The density-connectedness is used to find connected dense regions as clusters. Each closed set is considered to be a density-based cluster. A subset of elements from dataset is considered a cluster if it satisfies:

- for any two objects o_1, o_2 are considered to be clusters if o_1 and o_2 are density-connected;
- there does not exist an object o that is a cluster and another object o' that is not a cluster but o and o' are density-connected.

The advantage of the algorithm is that it achieves all the time same results but has a huge disadvantage because it makes nothing to represent the training dataset into a simplified manner. It uses all the time the entire dataset, that means the testing part is time consuming.

3.2 Evaluating clustering performance

The evaluation of the clustering algorithm is a problem because the groups are created automatically by the algorithm, from algorithm point of view, and don't corresponds with the class saved in the dataset. For cluster evaluation Weka [9] offers three different methods depending on the selected clustering method:

- *Use training set* (this is the default mode), when after generating the cluster Weka will classify the training instances into clusters according to the cluster representation and computes the percentage of instances assigned to each cluster;
- *In supplied test* set or *Percentage split:* it is used a separate test dataset for evaluating the clustering. This works only if the cluster representation is probabilistic;
- *Classes to cluster* where Weka first ignores the class attribute and generates the clusters. Then in the test phase it assigns classes to the clusters based on the majority value of the class attribute within each cluster. After that the classification error can be computed, based on this assignment.

Because we have only one class in the dataset, we use the default mode used by Weka to evaluate the algorithm. In this case we evaluate looking at the number of elements that remain outside of the clusters (numbers of samples which the algorithm didn't put into a cluster (these are considered noise in the dataset).

4 Experimental results

We intend to evaluate the algorithm regarding the number of attributes used in the learning part and regarding the influence of the algorithm parameters. Because we use a clustering algorithm, we don't use the class attribute present in the dataset. That class is used only in the feature selection method, for select the best attributes. This algorithm generates a different number of classes, depending on the parameters. We evaluate the algorithm based on the number of samples that are considered to be noise and are not grouped.

4.1 Influence of input parameters

For all the features from the initial file (1000 features) we evaluate the number of elements that are left out in the learning step and in the testing step. We have 542 samples that were randomly split by Weka in a training set with 358 samples and in the testing set with 184 samples. The algorithm generates between 3 and 7 clusters.

For the 1000 attributes we vary the ε between 0.9 and 0.1 and keep the *MinPts* constant. The number of documents in the resulted clusters for the training set are presented in Table 1 (358 samples and 1000 attributes)

Table 1. Learning rate for training dataset

ε	MinPts	Nr. clusters	C1	C2	C3	C4	C5	C6	C7	Noise	Learning Rate
0,9	6	3	102	61	19					176	50,84%
0,8	6	3	100	56	19					183	48,88%
0,7	6	3	94	49	19					196	45,25%

0,6	6	3	94	45	19					200	44,13%
0,5	6	6	41	7	27	7	11	8		257	28,21%
0,4	6	6	41	7	27	7	11	8		257	28,21%
0,3	6	6	40	7	25	7	11	8		260	27,37%
0,2	6	7	25	6	16	6	10	6	8	281	21,51%
0,1	6	6	10	6	15	6	13	7		301	15,92%

The column named 'Learning Rate' presents the percent of elements that were grouped by the algorithm into clusters. When this value is small it means a higher number of elements that were not grouped and were considered noise (these are presented also as number in column 'Noise'). Form this point of view we see that a small value for ε leads to an increased number of noise elements in the results.

For the testing set we present the results in the Table 2 (184 samples and 1000 attributes). In the testing part the algorithm will cluster the samples in learned groups. For the training and testing part we use the Euclidean distance as measure for computing the distance.

Table 2. Learning rate for testing dataset

ε	MinPts	Nr. clusters	C1	C2	C3	C4	C5	C6	C7	Noise	Learning Rate
0,9	6	3	48	27	10					99	46,20%
0,8	6	3	47	26	10					101	45,11%
0,7	6	3	57	24	10					105	46,43%
0,6	6	3	45	22	10					107	41,85%
0,5	6	6	18	4	16	5	4	4		133	27,72%
0,4	6	6	18	4	16	5	4	4		133	27,72%
0,3	6	6	18	4	16	5	4	4		133	27,72%
0,2	6	7	13	3	11	4	4	2	4	143	22,28%
0,1	6	6	5	3	10	4	6	4		152	17,39%

When we decrease the ε value the number of elements that are considered noise will increase in the same way for the training and for the testing dataset. It's interesting that when we create more clusters the number of documents that remain outside increases too.

In table 3 we present for the training dataset the influence of *MinPts* (number of points that are needed to be taken into consideration to create a core object and a cluster). Because we have obtained best values for ε=0.9 we will present the experiments only for this value.

Table 3. Influence of *MinPts* for training dataset

ε	MinPts	Nr. clusters	C1	C2	C3	C4	C5	C6	C7	…	Noise	Learning Rate
0,9	2	11	2	102	61	19	2	2	2	…	158	54,60%
0,9	3	3	104	62	19						173	51,68%
0,9	4	4	102	61	19	4					172	51,96%
0,9	5	3	102	61	19						176	50,84%
0,9	6	3	102	61	19						176	50,84%

0,9	7	3	102	59	19						178	50,28%
0,9	8	3	103	61	19						175	51,12%
0,9	10	3	101	59	19						179	50,00%
0,9	15	3	101	59	19						179	50,00%
0,9	20	2	101	59							198	44,69%

When we have a small number of *MinPts* the algorithm creates a large number of clusters. In table 4 we present only the values for the first 7 clusters. When the number of *MinPts* increases the number of clusters decrease but the number of elements that are considered noise increases too.

Table 4. Influence of MinPts for test dataset

ε	MinPts	Nr. clusters	C1	C2	C3	C4	C5	C6	C7	…	Noise	Learning Rate
0,9	2	11	1	48	27	10	1	2	2	…	88	50,84%
0,9	3	3	49	27	10						98	46,74%
0,9	4	4	48	27	10	1					98	46,74%
0,9	5	3	48	27	10						99	46,20%
0,9	6	3	48	27	10						99	46,20%
0,9	7	3	48	26	10						100	45,65%
0,9	8	3	48	27	10						99	46,20%
0,9	10	3	47	26	10						101	45,11%
0,9	15	3	47	26	10						101	45,11%
0,9	20	2	47	26							111	39,67%

From the 'Noise' point of view we can observe that, when the number of *MinPts* increases, for the same value for ε, the number of considered noise samples increases also. This means that in the file the samples are not uniformly distributed, and this becomes a problem to group such type of samples. Also, because we can have a relatively a small number of samples, we can't increase the *MinPts* value. The conclusion is that for text documents we need to use a large value for ε and a small number for *MinPts*.

5 Conclusions

The purpose of this paper was to analyze the performance of the DBScan algorithm in the context of clustering text documents. For this we have used the Weka implementation for the DBScan algorithm and algorithm evaluation, and we have used our own implementation for processing the Reuters files and create the dataset that respects the input Weka format.

We have evaluated the clustering algorithm from the point of view of the number of samples (Noise) that are left outside in the training and testing dataset. We have considered that if this value increases, the quality of the learning algorithm decreases. As it can be observed, we obtain the best values, in case of text document clustering, for a small number of MinPts (points that need to be taken in consideration for creating a core object) and

for a high value for Epsilon (radius of a neighborhood that is considered for each object).

6 References

[1] S. Chakrabarti, *Mining the Web- Discovering Knowledge from Hypertext Data*, Morgan Kaufmann Press, 2003;

[2] Cretulescu, R., Morariu,D. *Text Mining. Tehnici de clasificare si clustering al documentelor*, Published at Editura Albastra, Cluj Napoca, 2012, ISBN 978-973-650-289-7

[3] Radu Cretulescu, Daniel Morariu, Macarie Breazu - *Using WEKA framework in document classification,* Int. Journal of Advanced Statistics and IT&C for Economics and Life Sciences, Vol 6, No 2, ISSN 2067-354X, 2016

[4] Han, J., Kamber, M., *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, 2001;

[5] Mitchell T. *Machine Learning*, McGraw Hill Publishers, 1997.

[6] Mitkov R., *The Oxford Handbook of Computational Linguistics*, Oxford University Press, 2005;

[7] Daniel Morariu, Radu Cretulescu, Macarie Breazu - *The WEKA Multilayer Perceptron Classifier*, Int. Journal of Advanced Statistics and IT&C for Economics and Life Sciences, Vol 7, No 1, ISSN 2067-354X, 2017

[8] Reuters Corpus: http://about.reuters.com/researchandstandards/corpus/. Released in November 2000

[9] WEKA package - http://www.cs.waikato.ac.nz/ml/weka/index.html (accessed March 2015)