Improving Effectiveness of SVM Classifier for Large Scale Data

Jerzy Balicki, Julian Szymański(✉), Marcin Kępa,
Karol Draszawka, and Waldemar Korłub

Department of Computer Systems Architecture,
Faculty of Electronics, Telecommunications and Informatics
Gdańsk University of Technology, Gdańsk, Poland
{jerzy.balicki,julian.szymanski,kadr,
waldemar.korlub}@eti.pg.gda.pl, marc.kepa@gmail.com

Abstract. The paper presents our approach to SVM implementation in parallel environment. We describe how classification learning and prediction phases were pararellised. We also propose a method for limiting the number of necessary computations during classifier construction. Our method, named *one-vs-near*, is an extension of typical *one-vs-all* approach that is used for binary classifiers to work with multiclass problems. We perform experiments of scalability and quality of the implementation. The results show that the proposed solution allows to scale up SVM that gives reasonable quality results. The proposed *one-vs-near* method significantly improves effectiveness of the classifier construction.

Keywords: SVM · Wikipedia · Documents categorization · Parallel classification

1 Introduction

The size of the internet and globally stored data is growing with every year. Today the estimated number of indexed web pages is somewhere between 20 and 50 billion pages [1]. Automatic categorization of this evergrowing data becomes a real challenge. Even smaller text documents repositories, such as the Wikipedia reaching 4.5 million articles organized with hundreds of thousands of categories [2], require the aid of automatic categorization. The building of accurate text classifiers is a hard task by itself and the huge size of the data makes this problem even more challenging. There are many existing approaches to this problem, with different results both in terms of accuracy and performance [3] [4] [5] [6] [7] [8], but there is still need for improvements in this area.

The aim of the work presented here was to create a classifier capable of automatic categorization of text documents from repositories containing over 100k categories with acceptable performance and quality. The experiments, aimed at evaluating our classification solution, have been performed using Wikipedia data, created with our application that allows to construct its machine-processable representation [9].

The structure of this article is as follows. The next section briefly describes SVM classifiers and the way they are incorporated to solve multiclass classification problems. Section 3 describes our proposition to speed up and boost performance of SVM

L. Rutkowski et al. (Eds.): ICAISC 2015, Part I, LNAI 9119, pp. 675–686, 2015.
DOI: 10.1007/978-3-319-19324-3_60

in highly multiclass classification tasks. Then, in section 4 we present details of our parallel implementation of the proposed method. The experiments using this implementation, along with empirical results on Wikipedia datasets, are given in section 5. Last section concludes the paper and gives the ideas for further research in this area.

2 SVM Classifier in a Typical Multiclass Setting

One of the most effective methods of text classification is Support Vector Machines [10]. SVM in its base form is a binary classifier. Mathematically, SVM classifier is a hyperplane $h()$ in high dimensional feature space (examples are typically projected into that space by a kernel function), which is convex-optimized during training so that it separates the classes leaving maximal possible space (margins) between them. Prediction step can be summarized in a simple equation $a = h(x)$, where a is the activation of the hyperplane, and x is the feature vector (possibly transformed by a kernel) of a testing object. The sign of a decides which class is predicted, whereas the absolute value of a indicates the confidence of this decision. With advanced optimization algorithms used by SVM, time complexity of training such a hyperplane is $O\left(m_{\text{train}}\right)$, where m_{train} is number of training examples.

Although there are attempts to directly deal with multiclass problems using reformulated SVMs [11,12], most often such problems are decomposed into binary classifications and incorporate typical SVM classifiers summarized above.

In a popular *one–vs–all* scheme (also more correctly referred to as *one–vs–rest*), for each class a separate hyperplane is trained by treating examples from that class as positives and all the rest examples in the dataset as negatives. During prediction a test object is assigned to a class which hyperplane's activation a is the highest (*winner takes all* strategy). Complexity of calculating the whole model in this setting is $O\left(m_{\text{train}} \cdot N\right)$, where N denotes the number of classes. In case of prediction, assigning labels to m_{test} test objects can be performed in a $O\left(m_{\text{test}} \cdot N\right)$ time. For large datasets comprising lots of classes and objects, both training and prediction is computationally expensive to the extent where it becomes impractical to perform training and prediction sequentially and thus parallel techniques are necessary. Moreover, for very big datasets, the requirement that the whole dataset is needed for training a single hyperplane, can lead to memory problems. Another important issue with this approach is that it divides data into positive and negative classes which are very imbalanced, especially for highly multiclass problems. For example, assuming that there are 100k classes of equal size, the proportion of positive to negative examples would be 1:99999. This high imbalance can lead to poor quality of predictions.

The second popular scheme is called *one–vs–one*. Here, a separate SVM classifier is trained for each pair of classes, yielding $N(N-1)/2$ hyperplanes. The prediction in such a system is most often done with *Max Wins* approach, in which the class with the biggest number of *votes* from all classifiers is chosen, although more advanced techniques are possible [13]. Although the number of hyperplanes grows quadratically with N, each classifier requires examples only from two classes and not from the whole dataset, therefore the learning phase theoretically could still be $O\left(m_{\text{train}} \cdot N\right)$, while there are no problems with fitting data into memory and imbalanced data. Unfortunately,

this complexity estimation does not hold for datasets, where objects can belong to more than one class (multi-label classification tasks). Also, the testing phase, assuming standard *Max Wins* approach, is $O\left(N^2\right)$. Therefore, for large-scale multi-label problems (and categorizing Wikipedia articles belongs to this family), *one–vs–one* scheme becomes impractical. The comparative study of these multiclass SVM settings, as well as less popular ones, can be found in [14].

It is important to note, that the Wikipedia classification belongs to a multi-label family of problems, where a given document can be (and almost always is) associated with more that one category label. In such cases, the *winner takes all* algorithm of *one–vs–all* strategy is replaced with the following procedure. Each article is tested against every category in the dataset and the final result consists of categories with activation scores that exceed a specified threshold value. The number of categories returned as well as the minimum acceptable level of activation are parametrized. It should be noticed that changing these parameters has great impact on accuracy of the classifier. Having large computational resources (as training has to be repeated many times) this task can be optimized to select the parameters giving the best results.

3 *One–vs–near* Method

Since the number of categories has a crucial influence on classifier performance, we propose a solution to limit the number of necessary comparisons by modifying standard *one–vs–all* scheme. During SVM training instead of comparing every class with set of all others we compare it *only to the most similar ones*.

This scheme we call *one–vs–near*. It allows to limit the number of articles m_{train}, required to train during a single binary classifier construction, by reducing the dataset only to the most similar categories. This solution makes certain assumptions that need to be met for it to work properly.

- The dataset should contain many distinguishable categories.
- It should be possible to find nearest neighbors of each category in a short time.
- The neighboring categories should allow to create an accurate classifier.

All these conditions should be easily met in a sparse dataset such as the Wikipedia machine processable representation based on bag of words [15] and its extensions [16]. Because of the huge number of categories and articles, it should be possible to limit the training dataset size for each binary classifier. This solution should give us the following advantages:

- The memory consumption should be limited.
- The training performance should be better due to smaller size of the training set.
- The accuracy of the resulting classifier should be comparable to the one trained on the entire dataset, if above assumptions are satisfied, or even better due to noise reduction.
- The problem of highly unbalanced datasets should be mitigated.

Since the SVM uses the support vectors to create the hyperplane only the datapoints at the border between categories are significant to the result. This means that most of

the dataset should be redundant and only the points in the neighboring categories are needed to create an accurate classifier. This situation is symbolically represented in Fig. 1.

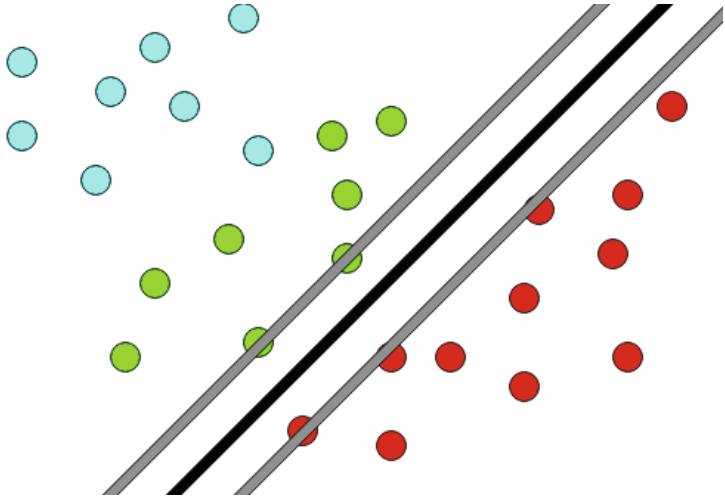

Fig. 1. Schematic example motivating *one–vs–near* approach

In this example we can see that only the border points of categories represented as red and green dots are needed in order to create the support vectors and plot the optimal hyperplane. The points of the category represented with blue dots are therefore redundant, and have no impact at the resulting classifier.

The neighboring classes are computed by a kNN classifier using cosine distance, typical in text processing [17]. To obtain nearest categories, instead of comparing distances of a test document feature vector to training feature vectors, the kNN classifier computes the distances between every category centroid – we model distances between categories by distances between their centroids.

4 Parallel Implementation Details

Besides the speedups obtained due to our *one–vs–near* multiclass scheme, the construction of scalable text classifier requires the use of a parallel computations environment. As presented before, both the training and the prediction task related to each SVM hyperplane are intrinsically independent, therefore the job of decomposing the problem between parallel computational nodes is straightforward. In fact, each task is either a category to train (in the training phase) or an article, for which classes are to be predicted (in the prediction phase). Each computational node picks up tasks from the task queue.

In our implementation each of computed hyperplanes is saved into a separate file and for each category prediction this file should be read. SVM training for large data can be parallelized by running a single class training procedure in one thread. Managing to distribute training procedures related to all classes over different computational nodes allows to construct a scalable classifier. Our implementation uses a file queue in order to distribute training tasks between processes. This allows us to run the classifier in parallel on many computing nodes using common network file system.

The prediction phase of the classifier requires to test each article from the test set against every hyperplane. Just as in the training phase, this problem can also be parallelized – the procedure is run in parallel threads and the use of file queue allows to run the program on many computing nodes.

In our implementation computations were parallelized in two ways. First, the jobs are spread across multiple machines. Then, on each machine multiple processor cores are used to speed up the whole process. Synchronization between computing nodes was obtained using Network File System (NFS) and file locking. With NFS, every machine works in the same directory, having access to the same files and saving results in the same folder. All the jobs to be done are stored in a single TODO file. The TODO file contains names of hyperplanes to train in case of training and list of objects to predict labels for in case of prediction. Every computing node can obtain certain number of jobs from the TODO file and run these jobs using available cores. Having done that it can receive new jobs and so on.

In addition to machine level parallelization the architecture allows for each node to run its computations in parallel threads. This means that eg. having 5 nodes each with 4 logical processors gives us 20 parallel threads in total. Both the number of nodes and threads running on each node is parametrized and can be changed depending on available hardware.

As manually starting the software on each node is time consuming and prone to errors we did MPI Message Passing Interface implementation [18] to run and initialize the program on specified nodes with a single command as well as to assign ranks to each process in order to identify them. The ranks are used for logging the computations of each node in a separate file and in some cases to assign certain parts of the problem to separate threads on separate nodes. Since the filesystem queue performed with acceptable results there was no need for any additional tasks division scheme like master-slave.

5 Experiments

To evaluate effectiveness of our approach we perform series of tests. They were planned to check performance, scalability and accuracy of the classifier. Initial tests have been performed with smaller size data and without cross validation in order to make them feasible to run with limited time and computing power. The final tests have been performed using large scale datasets.

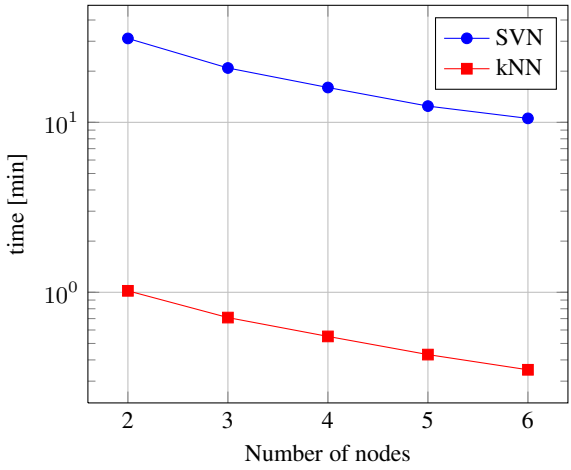

Fig. 2. SVM and kNN performance

5.1 SVM and kNN Training Scalability

For evaluation of SVM and kNN scalability we use data prepared from Wikipedia that contained 530 categories and 853283 articles. The average density of feature vectors equals 0.004%, which gives 876 MB file. The results were computed on our computer cluster with usage of different number of nodes. Each node consists of four logical processors, the time results are shown in Fig. 2.

The aim of this test was to approximate the performance of SVM classifier construction in relation to the number of computational nodes. As we can see the classifier scales quite well as it was expected. Whats more, we can see that the kNN classifier is faster by a magnitude, which confirms the most important assumption of the one–vs–near scheme, that we can find neighbours of each category in a short time.

5.2 Tests of Accuracy on Small Data

In order to assess the quality of the SVM classifier we perform a test of its accuracy. To limit the time of the experiment we run this test on the dataset with limited number of elements, same as in the previous experiment. The tests were performed using 10-fold cross-validation. The measured values contains precision, recall and the F-score. These values are presented in Tab. 1.

- $Precision = true\_positive/(true\_positive + false\_positive)$
- $Recall = true\_positive/(true\_positive + false\_negative)$
- $F - score = 2 * Precision * Recall/(Precision + Recall)$

Big number of examples per category allowed to train an accurate classifier. The results of the experiments presented in sections 5.1 and 5.2 indicate the classifier will scale up well and can reach acceptable results of classification quality.

Table 1. SVM classification results

True positives	False positives	False negatives	Precision	Recall	F-score
888306	203669	339974	81.34%	72.32%	76.56%

5.3 Scalability in the Function of Datasets Size

To check how the classifiers architecture works on large data we perform experiments with different sizes of big data packages. Beside computational effectiveness, the main concern here was the memory required on each computing node to load a dataset. To overcome that problem it was required to implement additional parameter that specifies the size of processed data block. For this experiment different sets of data were created with different numbers of articles.

Five various datasets were created from the full Wikipedia using Matrix'u application [9] based on 8th March 2013 [19] dump. They differ in the number of articles filtered out from them, the number of small categories merged with their parents and finally the number of the remaining small categories removed (categories with not enough examples to train a general classifier). Their description is shown in Tab. 2.

Table 2. Large scale datasets

Name	File size	Num. of cat.	Num. of art.	Vector density
Dataset1	2.1 GB	127402	2331707	0,0017%
Dataset2	2.4 GB	125573	2675198	0,0017%
Dataset3	2.7 GB	146444	3067138	0,0017%
Dataset4	3.0 GB	156829	3517048	0,0017%
Dataset5	4.4 GB	163986	3520309	0,0024%

The tests were run on two different clusters of computers with different hardware configurations:

Department Cluster: processor: Intel(R) Xeon(TM) CPU 2.80GHz, 4 physical, 8 logical cores, L2 Cache size: 2048KB, Memory: 4054340 kB, Swap 3076436 kB.

Lab 527: processor: Intel(R) Core(TM) i7-2600K CPU 3.40GHz, 4 physical, 8 logical cores, L2 cache size: 256K, L3 cache size, Memory: 8172568 kB, Swap 2111484 kB.

Each dataset was run on each cluster in order to test the memory consumption as well as to compare the performance between different hardware configurations. Both clusters consisted of 8 computing nodes with four threads per node, giving a total of 32 threads. The results of these tests are presented in Fig. 3

As expected, the execution times in this test are bigger for datasets containing more articles and categories and having bigger density. The lab527 cluster shows better performance than the department cluster for all datasets, most likely due to the hardware specification. The most interesting result of this test is the scaling of the classifier on the department cluster. The execution times grew nonlineary with the size of the dataset. This can be explained by the memory limitations of each node on this cluster. The times

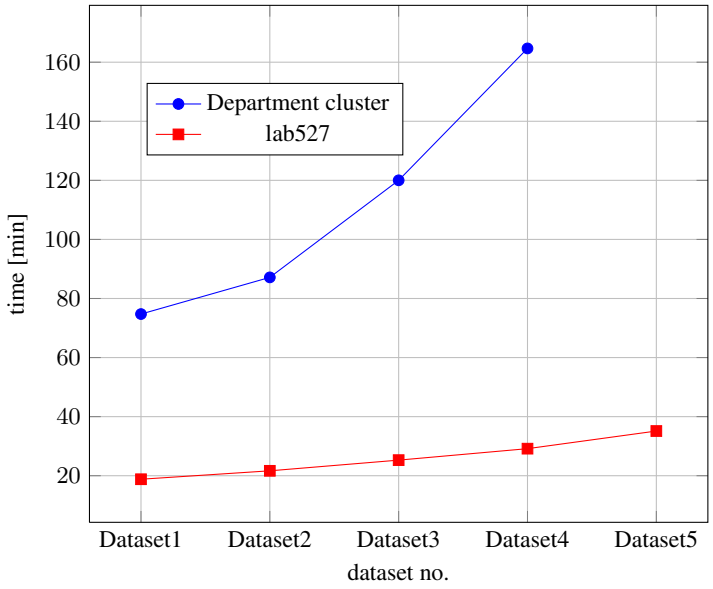

Fig. 3. kNN training scalability for large data

used for memory initialization by the system grew rapidly with dataset sizes closing to the maximum size of system memory.

5.4 *One–vs–near* Accuracy in Relation to the Number of Neighbors

This test was designed in order to measure the accuracy of the *one–vs–near* strategy in relation to the number of category neighbours and to compare it with the accuracy of the *one–vs–all* scheme. The test was conducted for a predefined set of parameters and the only variable parameter was the number of neighbours. As an evaluation dataset, here we use the same dataset as in section 5.2. The assessment of the *one–vs–near* accuracy was performed with use of 10-fold cross-validation. The results of this test are shown in Fig. 4.

As we can see in Fig. 4 the *one–vs–near* strategy starts to perform quite well above particular treshold of k neighbors. The results for the *one–vs–near* strategy were worse than the *one–vs–all* scheme if smaller number of class neighbors has been selected. Increase of that parameter leads to improvement of the results. Increasing the number of k neighbors over 400 makes the strategy practically equivalent to *one–vs–all*. This is intuitive as the number of category's nearest categories limits in the total number of all the other categories. It should be noticed proposed approach consumes additional computation time required to calculate neighbors. In next section we show how this addition processing is compensated by more effective classifier learning.

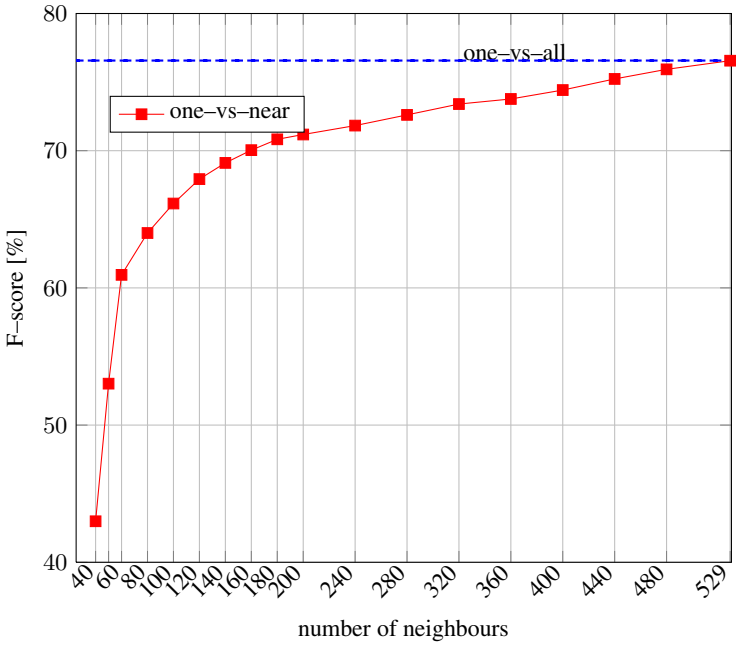

Fig. 4. *One-vs-near* accuracy compared to *one-vs-all*

5.5 *One–vs–near* Performance

Another factor we measure is the performance of the *one–vs–near* approach in relation to the number of neighbors. The results performed on lab527 cluster and dataset #5 are shown in Fig. 5.

What can be observed from the graphs given in Fig. 5 is that the decreasing number of classes used for classifier construction significantly reduces the computation time. The gain is significant up to using 500 neighboring classes. This fact together with previously observed improvement of the classifier quality constructed with smaller number of classes (see Fig. 4) constitute a strong argument that *one–vs–near* scheme can be used as a general method for SVM improvement.

However, it should be noticed that the *one–vs–near* method adds a fixed factor to the computations that comes from computing the neighbouring classes. This additional time can be observed in the graph when the total number of neighbouring classes exceeds 1000. Above that point *one-vs-all* strategy starts to perform better than one-vs-near. In our application we use a simple comparison of one class with others, but this time can be significantly reduced incorporating dedicated indexes [20].

Despite increasing computation cost, the memory requirements of the strategy proposed by us, interestingly, are still below requirements of *one–vs–all* approach. Our solution allows us to limit the memory consumption using the cost of additional computations. If the number of neighbouring classes would contain all examples from the

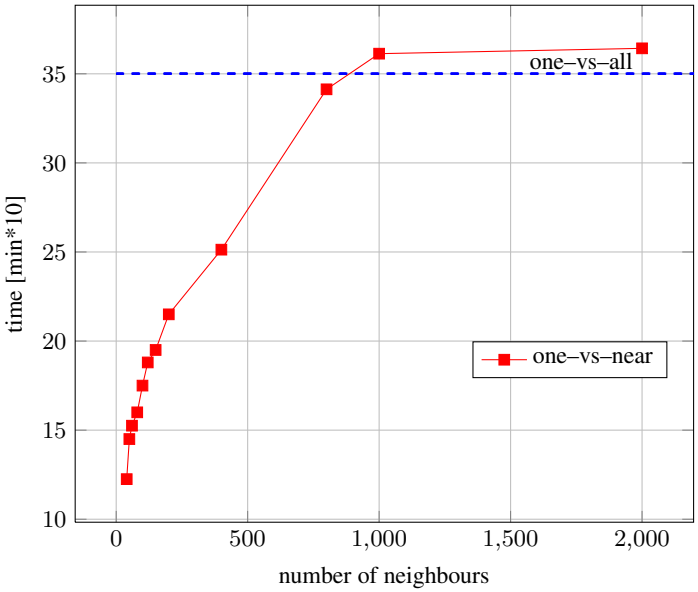

Fig. 5. *One–vs–near* performance in relation to the number of neighbors

dataset, it would cause maximum memory requirements that would be as big as for the *one–vs–all* scheme.

6 Conclusions and Future Work

In our research we developed and evaluated a parallel approach to classification of text documents with SVM. The algorithm was designed to be used with large scale text document repositories in mind, such as the Wikipedia. The proposed solution scales up well and gave acceptable accuracy results. Additionally, a new approach to improve effectiveness of classification – *one–vs–near* scheme – was implemented and tested. Proposed scheme provides accuracy comparable, to typical *one–vs–all* scheme while significantly improving time needed to classifier construction.

Although the problem of text documents classification was extensively tested in many works (eg. [5] [3] [21]), there is still some room for further research and improvements in this area. There are many yet untested approaches to this problem that might be worth testing.

One of such possible approaches is an application of different methods for neighbors search, such as the one proposed by Holloway et al [22]. It would be valuable to compare its quality and performance with the centroid based approach. Another idea would be to seek for neighboring articles instead of whole categories although this could be very demanding performance–wise. It would be also interesting to test the *one–vs–near* classifier with different kinds of SVM solvers and their parameters. Proposed approach can also be useful for identifying inner categories relations [23].

One thing that did not present satisfying results in our research was the memory usage reduction. Storage of large matrixes in the form of neighbor list can be possibly improved using some sort of array data DBMS, such as SciDB [24], to store the feature vectors, instead of plain text files. This would further improve the performance of the classifier and allow to limit the batch size and thus the memory requirements.

Another potentially interesting approach would be to make the initial classification on some reduced set of higher level categories (only if categories are organized in a hierarchical structure, as in case of Wikipedia categories) and then continue the more detailed classification only on the set of neighboring categories. However, the accuracy of such solution could be greatly decreased since errors from each classification stage would multiply.

As mentioned before, the developed classifier presents promising results in the Wikipedia classification task. There are many different approaches to automatic classification that have not been tested yet or at least not in the context of text documents classification. Some of which were mentioned in this section. It is an interesting topic and there is still a lot of potentially valuable research to be done in this area of computer science. A big challenge will be to develop some classifiers using a quantum-inspired algorithms [25] as well as some immune algorithms.

Acknowledgements. The work was supported by founds of Department of Computer Systems Architecture of Faculty of Electronics, Telecommunications and Informatics Gdańsk University of Technology.

References

1. de Kunder, M.: The size of the world wide web (2014), http://www.worldwidewebsize.com/ (Online; accessed May 22, 2014)
2. Wikipedia: Size of wikipedia (2014), http://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia (Online; accessed January 25, 2014)
3. Gantner, Z., Schmidt-Thieme, L.: Automatic content-based categorization of wikipedia articles. In: Proceedings of the 2009 Workshop on The People's Web Meets NLP: Collaboratively Constructed Semantic Resources, People's Web 2009, pp. 32–37. Association for Computational Linguistics, Stroudsburg (2009)
4. Han, E.-H., Karypis, G.: Centroid-based document classification: Analysis and experimental results. In: Zighed, D.A., Komorowski, J., Żytkow, J.M. (eds.) PKDD 2000. LNCS (LNAI), vol. 1910, pp. 424–431. Springer, Heidelberg (2000)
5. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: Liblinear: A library for large linear classification. Journal of Machine Learning Research 9, 1871–1874 (2008)
6. Miao, Y., Qiu, X.: Hierarchical centroid-based classifier for large scale text classification. In: Large Scale Hierarchical Text Classification (2009)
7. Chu, C.T., Kim, S.K., Lin, Y.A., Yu, Y., Bradski, G.R., Ng, A.Y., Olukotun, K.: Map-reduce for machine learning on multicore. In: Schölkopf, B., Platt, J.C., Hoffman, T. (eds.) NIPS, pp. 281–288. MIT Press (2006)
8. Balicki, J., Korłub, W., Szymanski, J., Zakidalski, M.: Big data paradigm developed in volunteer grid system with genetic programming scheduler. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2014, Part I. LNCS (LNAI), vol. 8467, pp. 771–782. Springer, Heidelberg (2014)

9. Szymański, J.: Wikipedia Articles Representation with Matrix'u. In: Hota, C., Srimani, P.K. (eds.) ICDCIT 2013. LNCS, vol. 7753, pp. 500–510. Springer, Heidelberg (2013)
10. Joachims, T.: Text categorization with support vector machines: Learning with many relevant features. Springer (1998)
11. Crammer, K., Singer, Y.: On the algorithmic implementation of multiclass kernel-based vector machines. The Journal of Machine Learning Research 2, 265–292 (2002)
12. Lee, Y., Lin, Y., Wahba, G.: Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. Journal of the American Statistical Association 99, 67–81 (2004)
13. Platt, J.C., Cristianini, N., Shawe-Taylor, J.: Large margin dags for multiclass classification. In: Advances in Neural Information Processing Systems, vol. 12, pp. 547–553. MIT Press (2000)
14. Duan, K.-B., Keerthi, S.S.: Which is the best multiclass SVM method? An empirical study. In: Oza, N.C., Polikar, R., Kittler, J., Roli, F. (eds.) MCS 2005. LNCS, vol. 3541, pp. 278–285. Springer, Heidelberg (2005)
15. Szymański, J.: Comparative analysis of text representation methods using classification. Cybernetics and Systems 45, 180–199 (2014)
16. Szymański, J.: Words context analysis for improvement of information retrieval. In: Nguyen, N.-T., Hoang, K., Jędrzejowicz, P. (eds.) ICCCI 2012, Part I. LNCS, vol. 7653, pp. 318–325. Springer, Heidelberg (2012)
17. Baeza-Yates, R., Ribeiro-Neto, B., et al.: Modern information retrieval, vol. 463. ACM Press, New York (1999)
18. Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L., Woodall, T.S.: Open MPI: Goals, concept, and design of a next generation MPI implementation. In: Proceedings of the 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, pp. 97–104 (2004)
19. Wikipedia: Wikipedia database dump (2014), http://dumps.wikimedia.org/enwiki/20140102/ (Online; accessed January 25, 2014)
20. Kryszkiewicz, M., Skonieczny, L.: Faster clustering with DBSCAN. In: Intelligent Information Processing and Web Mining, Proceedings of the International IIS: IIPWM 2005 Conference held in Gdansk, Poland, June 13-16, pp. 605–614 (2005)
21. Hsu, C.W., Chang, C.C., Lin, C.J.: A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University (2003)
22. Holloway, T., Bozicevic, M., Börner, K.: Analyzing and visualizing the semantic coverage of wikipedia and its authors: Research articles. Complex 12, 30–40 (2007)
23. Szymański, J.: Mining relations between wikipedia categories. In: Zavoral, F., Yaghob, J., Pichappan, P., El-Qawasmeh, E. (eds.) NDT 2010. CCIS, vol. 88, pp. 248–255. Springer, Heidelberg (2010)
24. Cudré-Mauroux, P., Kimura, H., Lim, K.T., Rogers, J., Simakov, R., Soroush, E., Velikhov, P., Wang, D.L., Balazinska, M., Becla, J., et al.: A demonstration of scidb: a science-oriented dbms. Proceedings of the VLDB Endowment 2, 1534–1537 (2009)
25. Balicki, J.: An adaptive quantum-based multiobjective evolutionary algorithm for efficient task assignment in distributed systems. In: Proceedings of the WSEAES 13th International Conference on Computers, ICCOMP 2009, Stevens Point, Wisconsin, USA, pp. 417–422. World Scientific and Engineering Academy and Society (WSEAS) (2009)