Parallel Processing of Very Many Textual Customers' Reviews Freely Written Down in Natural Languages

Jan Žižka and František Dařena
Department of Informatics
FBE, Mendel University in Brno
Brno, Czech Republic
Email: zizka@mendelu.cz, darena@mendelu.cz

***Abstract*—Text mining of hundreds of thousand or millions of documents written in a natural language is limited by the computational complexity (time and memory) and computer performance. Many applications can use only standard personal computers. In this case, the whole data set has to be divided into smaller subsets that can be processed in parallel. This article deals with the problem how to divide the original data set, which represents a typical collection containing two millions of customers' reviews written in English. The main goal is to mine information the quality of which is comparable with information obtained from the whole set despite the fact that the mining is carried out using subsets of the original large data set. The article suggests a method of dividing the set into subsets including a possibility of evaluating the mining results by comparing the unified outputs of individual subsets with the original set. The suggested method is illustrated with a task that searches for significant words expressing the customers' opinions on hotel services. It is shown that there is always a certain boundary under which the subset sizes cannot fall as well as how to experimentally find this border.**

***Keywords*-text mining; natural language; parallel processing; decision tree; data subset size; computational complexity.**

I. Introduction

Today, it is important to look for methods which speed up document search and reduce classifier training times and errors for very large text data collections [11]. Using a very large set of real data, this paper describes a parallelism-based procedure that improves the deficiency caused by rapidly increasing computational complexity. Collecting and subsequent processing of customer opinions that relate to a specific matter can usually present a valuable form of feedback. Many organizations and companies allow their users or customers to subsequently express opinions or sentiments, which can be later used for improving the provided services or any related activities with the intention to strengthen competitiveness. As a commonplace, the opinions are in many cases written down by way of the Internet as free unformatted (or with a very limited formatting), not very long text reviews using any natural language. Logically, the more reviews expressing various opinions, the better information can be mined and utilized from the data collection. Today's literature, like [9], describes a lot of different possibilities what we can mine from textual data, from clustering and classification to sentiment analysis to computational linguistic topics.

Looking at the high number of reviews as a naturally positive thing, it is necessary to see also the second coin side: Due to the nonlinear increase of computational complexity, the processing of very many textual items can take also very long time, including escalated memory demands.

In this article, we describe their experience with opinion mining from large textual data containing hundreds of thousands to millions of reviews written down by customers of on-line hotel services. The method itself of text mining was published, for example, in [2][13][14][15][16]. However, the mining in question had to face up to the high computational complexity caused by the big data volume. Altogether, there were more than five millions of customer reviews written in more than 50 natural languages. The most of reviews were written in English (almost two millions), following from more than 700,000 to more than 300,000 in French, Spanish, German, and Italian, to mention just the largest data sets. The original task was to mine significant words and phrases representing the customers' opinions concerning the hotel services booked on-line.

Among the main intentions, there was also the investigation how possible was the realistic text-mining using a common personal computer, PC, (as a 64-bits four-kernel processor 2.0 GHz, 8 GB RAM, 64-bits MS Windows 7 Professional) supposing that the hotel service provider had no access to a super-computer. The experiments quickly showed that it was not possible to process the big data sets en bloc, either because of insufficient memory or very long computational times (weeks), even if the mining procedure applied a professional implementation of the decision tree generator *c5/See5* that is based on the entropy minimization, see [7], that worked with RAM very well. However, in the beginning when *c5/See5* needs to read large data, it consumes a lot of memory, too.

After some experiments, the authors had to accept a natural solution based on dividing the whole data set into smaller subsets that could be processed in parallel using several common PC's. Different authors applied parallelism to various problems connected to very large data sets. For example, Ulmer et al. [12] created a text document-similarity classifier used to detect web attacks in HTTP data streams. They applied a parallel hardware approach because a sequential algorithm could not process a real-time data stream above certain data volumes. The parallel approach was also used for text feature selection – the process was parallelized and demonstrated using a cluster formed with several computers [6]. For data divided into several broad domains with many sub-category levels, separate classifiers of the same type could be trained on different subspaces in parallel. An improvement in subspace learning was accompanied by a very significant reduction in training times for all types of used classifiers [11]. Lertnattee and Theeramunkong [5] parallelized and distributed the process of text classification separately in each dimension. Classifiers learned from large training documents with a small number of classes on each dimension, and the best classifiers for each dimension were then combined. Both learning and classification phases run in parallel. Hao and Lu [3] developed a modular version of the k-nearest neighbor algorithm (k-NN) which was a faster and more efficient method for large-scale text categorization by direct modular classification without reducing the precision of the classification. The algorithm decomposed the large-scale text categorization problem into a number of smaller two-class subproblems and combined all of the individual modular k-NN classifiers info one classifier.

As the experiments described further showed, it was not negligible how large the subsets were (how many reviews they contained) because a dictionary of each subset was logically not identical, some significant words were not in all the data parts, or their significance – based on the frequency representation – markedly changed. Such a behavior of the textual data can be expected due to the high sparsity of vectors representing individual reviews.

In the following sections, a reader can find the English data description (Section II), the design of experiments (Section III), the results and their interpretation (Section IV) and, finally, conclusions (Section V).

II. Characteristics of the Examined Text Data

The investigated textual data represented usual customers' reviews written freely in natural languages, without following any specific structure or form. In all of the languages, the hotel service customers were satisfied or dissatisfied with the same or very similar, typical things (cleanness, price, personal willingness or helpfulness, noise, price, hotel position, food, and so like).

The results presented in this article come from the largest data set that contained the customers' reviews written in English, however, there were no significant differences in other 'big' languages (from the data volume point of view) mentioned in the Introduction section. It is necessary to emphasize the fact that not all authors of English reviews were English native speakers – the natural reason was that people all over the world use English, 'international English', as a universal communication means. As a result, the reviews contained many imperfections coming from lower knowledge of English or mistyping. This language incorrectness brings certain consequences like the artificial extension of the word list (dictionary) where a word can have many variations but only one is correct, for example, '*behavoir, behavior*', '*acomodation, accommodation, accomodation, acommodation*', '*noise, nois*', and so like. Such imperfections could be subsequently corrected by spell-checkers, however, without a human control (that could be for large data impossible) the result would not be guaranteed – fully automatic check-spelling can introduce additional errors. One possibility could be applying a spell-checker during writing a review but it would also need spell-checkers for all acceptable languages. Similarly, the customers used often also interjections like '*gooooood, goood*', '*aaarrrgh-hhh*', '*uuugly*', and so like, to express their dis/satisfaction with the service.

Sometimes, the English text contained also non-English terms when a customer could not remember a word, for example '*albergo*', which in Italian means '*hotel*'. In some cases, there were reviews written in two languages but they were assigned to English because customers wanted to express the opinion in their native language, however, if their native language belonged to a group of 'small' languages, as for example Czech, the opinion contained also its (not always correct) English version – one could not expect that hotel managers in, say, South America knew Czech. Here are some original examples of reviews without corrections:

- *breakfast and the closeness to the railwaystation were the only things that werent bad*
- *did not spend enogh time in hotel to assess*
- *it was somewhere to sleep*
- *very little !!!!!!!!!*
- *breakfast, supermarket in the same building, kitchen in the apartment (basic but better than none)*
- *no complaints on the hotel*

Overall, the English dictionary generated from the English group of reviews contained some 200,000 words in almost 2,000,000 reviews, which could be represented by matrix having two millions rows and 200,000 columns – a really large matrix containing ca 4×10^{11} numbers where each number meant the frequency of a word in a review.

The dictionary contained only *words*, which means that all numbers, punctuation symbols, or any special marks were excluded. Sometimes, the dictionary contained peculiar 'words' like '*t*' but it resulted from the preprocessing of the original words like *didn't* after using the apostrophe as one of delimiters, therefore *didn't* was transformed into two 'words' *didn* and *t*. In addition, all characters were transformed into the lower-case representation to avoid having more versions of the same term – even if there could be some loss of information, for example *Rose* (a hotel name) and *rose* (a plant); however, such cases were extremely rare, without influencing the results.

Each review was transformed into vector, which is a standard representation method. This representation contains all possible dimensions (that is, words in the dictionary), however, because of the word number per review and the word number in the dictionary, the vectors are extremely sparse, containing zeros in most of word positions because the minimum review length was one word (for example, *Excellent!!!*), the maximum was 167 words, and the average length of a review was 19 words. The vector sparsity was typically around 0.01%, that is, on average, a review contained only 0.01% of the words in the dictionary.

Comparing those word numbers with the dictionary size, it is clear that the vectors were very sparse, containing mostly zeroes for the word frequencies. Still, a human reader could say what reviews were positive, negative, mixed, neutral, or non-classable, and what terms were significant from the positive or negative standpoint, attitude, or sentiment: *noisy, quiet, smell, helpful personnel, good but small food portion, dirty rooms, nice hotel position*, and so like. As the previous research showed, see [15], an overwhelming majority of words were insignificant, only some 300 of terms played the significant role from the classification point of view (either a negative or positive review); the huge majority from almost 200,000 dictionary words had no function. That vector sparsity influenced the results of dividing the original data set into smaller subsets to decrease the computational complexity.

III. Experiments for Finding the Subset Size

The experiments were aimed at finding the optimal subset size for the main data set division. The *optimum* was defined as *obtaining the same results from the whole data set and the individual subsets*. Such a non-mathematical definition ideally meant that each subset should provide the same significant words that would have the same significance for categorizing reviews into correct classes; here, positive and negative opinions where the review positivity or negativity was given by a customer. As the subsets contained different, randomly selected reviews, the similarity of the subset results were defined as an average value.

The *word significance* was defined as *the number of times when a decision tree asked what was a word frequency in a review*. Obviously, the most significant words are tested every time for each classification query, which is, for example, quite typical for a word in the tree root, even if there could also be other words tested in 100% cases. Usually, the word frequency tests on lower tree levels do not check the words so often as on the higher levels due to the wide tree branching. The words included in the tree are in fact the relevant attributes from the classification point of view; other words are irrelevant and could be calmly omitted – but it is not how people create sentences understandable for them.

Therefore, if there is a word from the whole data set R in the root, most of the n subsets (ideally all) should have the same word in their roots. Similarly, the same rule can be applied to other words included in the trees on levels approaching the leaves. Then we could say that each subset represents the original set perfectly. In reality, the decision trees generated for each review subset r_i more or less mutually differ because they are created from different reviews. In addition, a tree generated from a subset r_i may contain also at least one word that is not in the tree generated from R. Each tree provides a set w_{r_i} of significant words. The union w_r of the sets of significant words w_{r_i} should give a resulting set that should ideally have the same words as in the whole review set R with the word set w_R provided by the tree generated for R:

$$r_i \subset R,\ w_{r_i} \subset w_R\ , \tag{1}$$

$$w_r = \bigcup_{i=1}^{n} w_{r_i}\ , \tag{2}$$

$$\text{therefore ideally, } w_R = w_r\ , \tag{3}$$

for $i = 1,\ \ldots,\ n$.

Thus, the question is: How many subsets should the whole review set R be divided into so that the unified results from all r_i's provide (almost) the same result as from R? Intuitively, if each r_i would contain just one review, the result can be bad because the individual reviews are typically very different even if they refer to the same thing: *bad accommodation, not good accommodation, horrible accommodation, we were not satisfied with the accommodation, excellent accommodation, relatively good accommodation*, and so like. The only shared word is *accommodation*, however, it itself is not either positive or negative, it is simply neutral. The adequate decision trees would be very different.

If those reviews would be grouped into one common set, the adequate tree would be also very different from the previous individual trees and, moreover, it would represent certain generalization, that is, knowledge. Provided that a computer cannot process the whole set R, the intuitively best way would be to create n as large subsets r_i as possible so that the computer could process its r_i as quickly as possible without the preliminary depletion of memory. Then, having n computers, the reviews could be processed during the time acceptable by a user.

Obviously, it is not easy to find a general solution because the result depends on particular data. The authors selected the data described above because it corresponded to many similar situations: a lot of short reviews concerning just one topic.

Firstly, the original set R was too big to be processed as a whole: two millions reviews. For a given PC model, the authors looked for the maximal size of R using a random selection from the whole original set. Selections containing more than 300,000 reviews crashed because of insufficient PC memory (8 GB RAM). The sets with 300,000 reviews (and more) were not ready after a week, therefore the computations had to be canceled.

In this place, it is worth to remark the computational complexity of the *c5/See5* decision tree type. In [10], the authors mention the time complexity of the c4.5 (a forerunner of c5) decision tree generator. The upper boundary is $O(m \cdot n^2)$, where m is the size of the training data (the number of matrix rows) and n is the number of attributes (the number of words in the dictionary).

The subset containing 200,000 reviews (10% of the whole original data) consumed almost 85,000 seconds of elapsed time (approximately 24 hours), therefore it was accepted as the largest processable R. Similarly, there were successively created smaller R's: 100,000, 50,000, and 20,000 (plus other sizes, but there were no big differences in the results between R's with similar sizes). Each of R was processed to obtain its particular significant words.

After that, each of the generated R's was randomly divided into smaller r's so that the individual sizes of each r_i represented 10%, 20%, 25%, 30%, 40%, and 50% of its adequate 'parent' R.

In the second step, every data set was preprocessed using the commonly known method called *bag-of-words*, see for example [8]. The reason was that linguistic preprocessing was impossible due to the too large data volume and not the same method for any language. In addition, all words appearing only once in the whole data set were removed which decreased the number of words, n, in the dictionary almost to a half, and the computational complexity even more because $O(m \cdot n^2)$ depends strongly on n^2.

The words were represented by their frequencies in reviews. As it was mentioned above, in each vector there were mostly zeros. The subsequent experiments tried the more advanced representation called *tf-idf* (term frequency times inverted document frequency, see for example [8]), however, the results were not better (maybe because the sizes of reviews were very similar – typically tens of word).

For each R and r_i, the third step gradually generated the decision trees to reveal the significant words as the relevant attributes for the classification to the positive or negative opinion class. Typically, the results looked similarly like this: *100% location, 80% friendly, 79% not, 73% excellent, 68% helpful, 63% closeness, 63% helpfulness, 63% friendliness, 62% comfortable, 62% spacious, ..., 5% facilities, 5% and, 4% nothing, 3% on, 2% door, 2% with, 2% to, 2% so, 1% in,* and so on (in this example, there were 167 significant/relevant words in the tree; in other cases, it was similar). The first word always represented the root – the tree asked the *location* frequency always, *friendly* frequency in 80%, and so on. The percentage value plays here the role of the *significance weight* because the frequencies of words that are closer to the root contribute more to the entropy decrease than frequencies of words on levels closer to leaves.

The basic result was always given by an R set. The lists of significant/relevant words generated for individual r_i's, where $r_i \subset R$, were compared with the basic result. The authors were interested in the fact how much each significant word in r_i corresponded to the same word in R from the percentage point of view, S_R – that is, a word in the R tree had its percentage equal to S_R. The sets of significant words generally contained a lot of the same words, even if there were also words that were not included in all r_i trees. For the r_i's common percentage of a given significant word, S_r, it was taken the average value:

$$S_r = \frac{1}{n} \sum_{i=1}^{n} S_{r_i} \ , \tag{4}$$

where n is the number of r's (subsets of R) and S_{r_i} is the percentage of the word in the i-th subset r_i.

Then, it was possible to compare S_R's with S_r's for each word and subset. As it was expected, dividing an R set into less but larger r_i subsets provided better results – the correspondence between R and its r's was closer to the ideal than in the case of more smaller subsets r. On the other hand, smaller subsets were processed noticeably faster than the larger ones. One of the reasons was the fact that each r_i contained only part of the total dictionary generated from R – consequently, smaller dictionaries of r_i's decreased also the computational complexity $O(.)$ The results of mining significant words are demonstrated in the following section.

IV. Results of Experiments

To compare results provided by the review sets and subsets having various number of items, the authors used a method that is illustrated in the following graphs Figure 1, Figure 2, and Figure 3.

On the horizontal axis, there are significant words generated by the trees. The graph does not show all significant words because of insufficient space; only the words having the higher percentage value are here used.

The vertical axis y shows the correspondence between the percentage of the significant words in the relative R set and the average percentage of the relevant r_i subsets. The whole set R contains all the significant words which means that the y value is always 1.0 (that is, 100%). In other words, the occurrence of significant words w_i in R is given by a simple equation:

$$y_R(w_i) = 1.0 \ . \tag{5}$$

On the other hand, some words in some r_i's could be missing. In the case of individual r_i's, the occurrence of

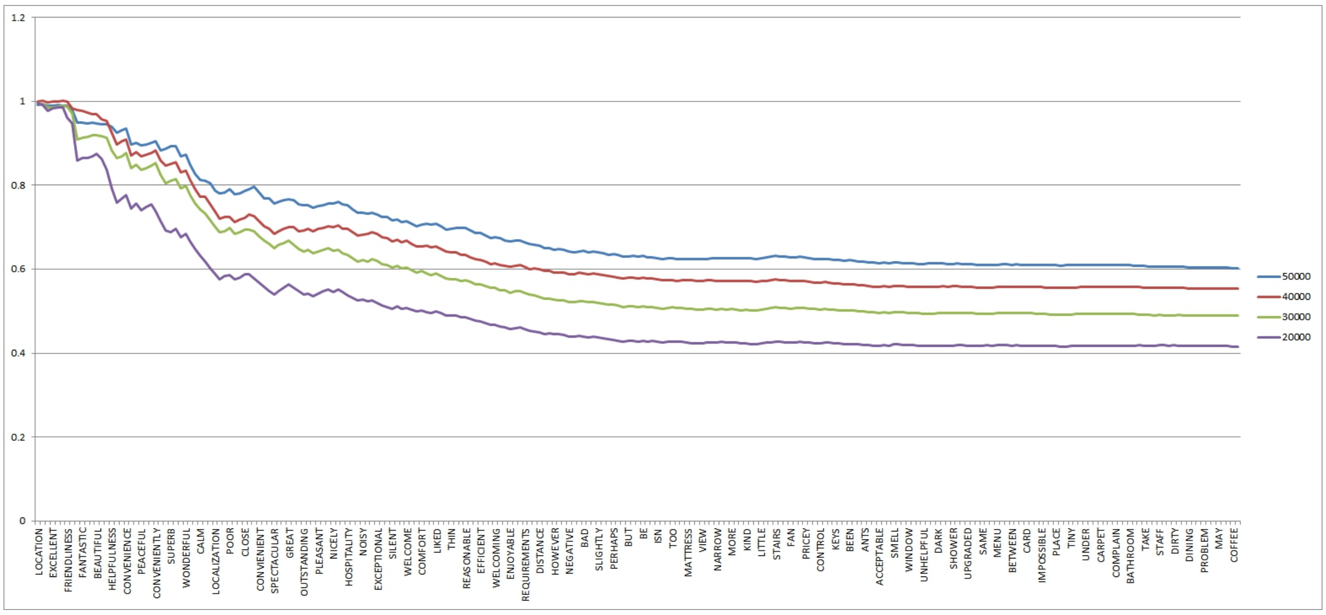

Figure 1. The whole set R with 200,000 reviews divided into subsets r_i having gradually 50,000, 40,000, 30,000, and 20,000 reviews

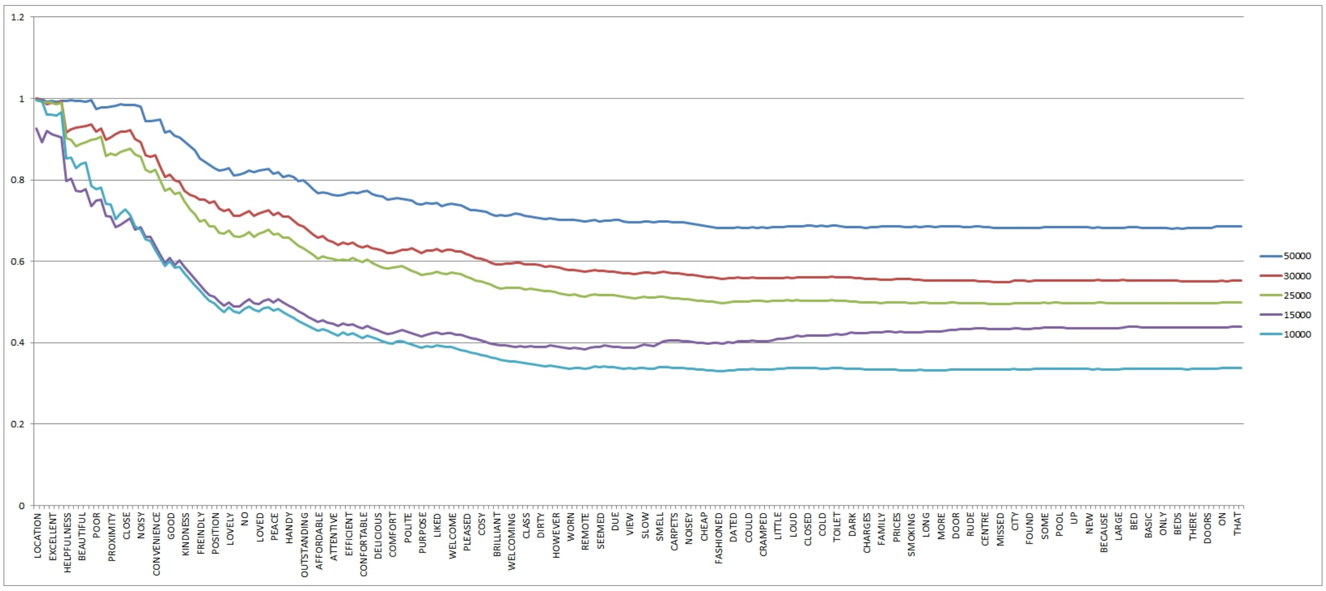

Figure 2. The whole set R with 100,000 reviews divided into subsets r_i having gradually 50,000, 30,000, 25,000, 15,000, and 10,000 reviews

significant words is expressed using the following formula, where for a word w_i the value on the y axis is calculated as:

$$y_r(w_i) = \frac{\sum_{j=1}^{i} S_r(w_j)}{\sum_{j=1}^{i} S_R(w_j)} \; , \tag{6}$$

where i is the serial number of a word, S_R is the percentage of usage of a word w_j given by the decision tree and created for the complete data set R, and S_r is the average percentage of usage of a word w_j by the decision tree created for every subset $r \subset R$. The same word can have different percentage values in different subsets r as well as in the relative set R, therefore $y_r(w_i) \neq y_R(w_i)$. Ideally, all significant words should be at the same tree position having the same weight; then, $\forall i, y_r(w_i) = y_R(w_i) = 1.0$.

Equation 6 measures the agreement between the percentage weight of a word w_j in the tree generated for R and the average value in the trees generated for all r_i's, where $r_i \subset R$. For example, if a whole set R would be randomly divided into $n = 5$ subsets, and a certain

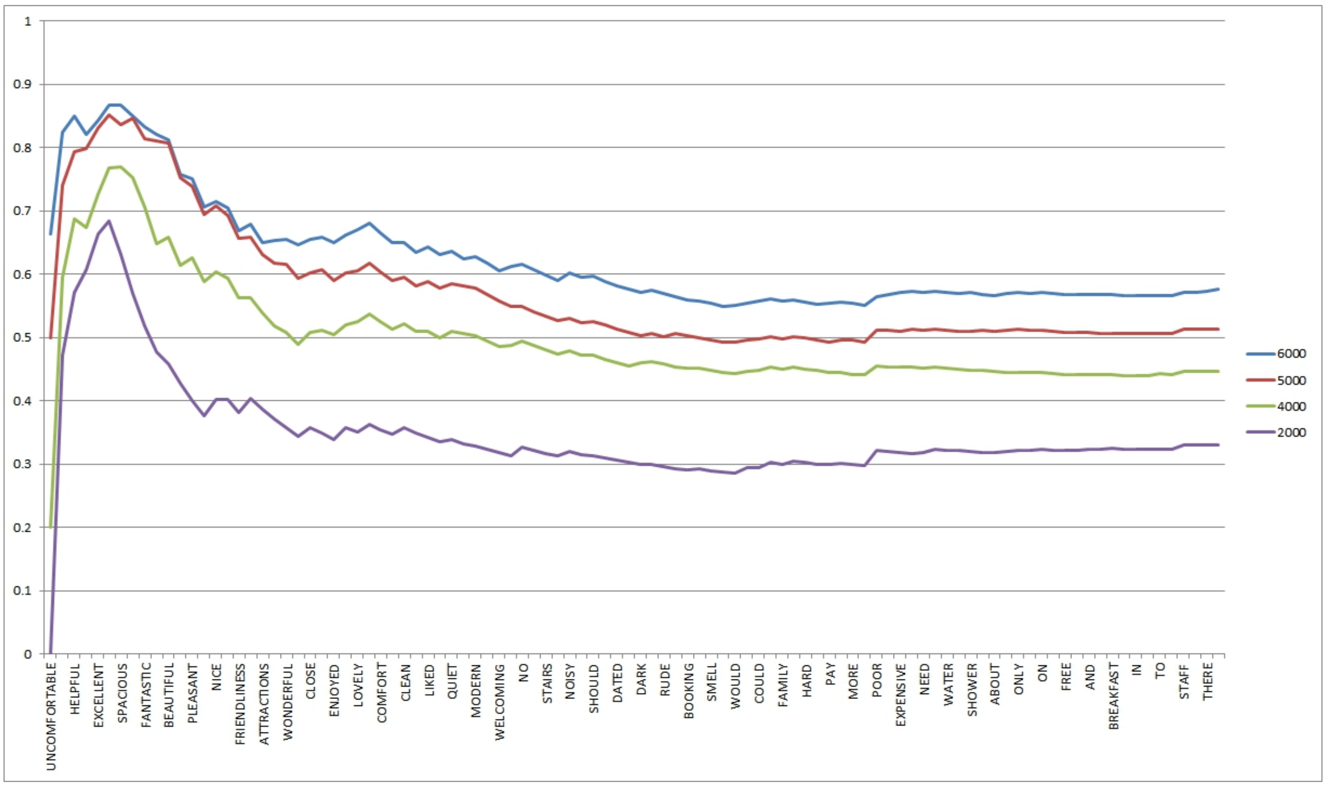

Figure 3. The whole set R with 20,000 reviews divided into subsets r_i having gradually 6,000, 5,000, 4,000, and 2,000 reviews

word $w_j = excellent$ would have its percentage weight $S_R(w_j) = 73\%$, then if all r_i's would have the same word weight $S_{r_i}(w_j) = 73\%$ for $i = 1, \ldots, 5$, the agreement is perfect, that is, $y(excellent) = 1.0$; otherwise, the results provided by the subsets may differ from the whole set.

The graphs in Figure 1, Figure 2, and Figure 3 show how the averaged results of subsets r_i agree with the results obtained from the complete review sets R for individual words that are at the top of the percentage list. Each individual curve represents an average result for r_i's that have a certain number of reviews (see also the graphs legends).

For example, Figure 1 illustrates the situation when R contains 200,000 reviews. After dividing R into four subsets $r_i, i = 1, 2, 3, 4$, where each r_i has 50,000 randomly selected reviews, it is possible to see that the correspondence (computed using Equation 6) is better than 80% for the first 13 significant words with high percentage weights. Then the similarity gradually decreases, but never under 40%. The curves also show that dividing R into 10 subsets r_i's (20,000 reviews per r_i) provides worse results than for the less number of larger r's.

Similarly, Figure 2 illustrates the situation for R containing 100,000 reviews, and Figure 3 for 20,000 reviews (here are the results markedly much worse – in addition, no r_i contained the R's root word *uncomfortable*). The experiments were carried out for various subset sizes and whole sets, however, the results were quite consistent, therefore they are not here illustrated all – only the three most characteristic ones.

V. Conclusion

Interestingly and predictably, the graphs illustrate the fact that the higher number of smaller subsets provide altogether worse results than the lower number of larger ones. Naturally, the complete review set R provides the best result as one extreme, and subsets (singletons) of R, containing just single reviews, give the worst results as the contrary extreme (not shown here because it is not interesting – at least, no one would process 2,000,000 reviews using 2,000,000 computers in parallel).

When the R data volume is too large to be processed using one PC, it has to be divided into smaller subsets r. It is probably not a big surprise that the smaller subsets should be as large as possible, however, the authors needed an empirical proof that randomly divided original sets R into subsets can provide similar (if not identical) results by unifying the results of all individual subsets using some large real-world data. Also, it was necessary to test what subset sizes could be used to obtain reliable results within a reasonable time (max. several hours, not many days).

The *result reliability* is a rather ‘fuzzy’ concept; it naturally depends on a user what he or she would accept as *reliable*. However, in reality, users mostly have no choice – standard PC’s do not enable processing of such large data volumes, thus it is very useful to know how the data having the similar properties as the one analyzed here should be prepared for the parallel processing that radically decreases the computation complexity (both time and memory).

ACKNOWLEDGMENT

The research work published in this paper was supported by the Research program of the Czech Ministry of Education VZ-MSM-6215648904.

REFERENCES

[1] http://www.rulequest.com/see5-info.html, September 2012.

[2] F. Dařena and J. Žižka, ”Text Mining-based Formation of Dictionaries Expressing Opinions in Natural Languages.” In: Proceedings of the 17th International Conference Mendel-2011, June 15-17, 2011, Brno, Czech Republic. No. 1, pp. 374–381, Brno Technical University Press, 2011.

[3] H. Zhao and B. L. Lu, ”A modular k-nearest neighbor classification method for massively parallel text categorization.” Lecture Notes in Computer Science, Vol. 3314, 2004, pp. 867-872.

[4] M. Hu and B. Liu, ”Mining and Summarizing Customer Reviews.” In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD’04, August 22-25, 2004, Seattle, Washington, USA. ACM, 2004.

[5] V. Lertnattee and T. Theeramunkong, ”Parallel text categorization for multi-dimensional data.” Lecture Notes in Computer Science, Vol. 3320, 2004, pp. 38-41.

[6] M. J. Meena, K. R. Chandran, A. Karthik, and A. V. Samuel, ”An enhanced ACO algorithm to select features for text categorization and its parallelization.” Expert Systems with Applications, Vol. 39, No. 5, 2012, pp. 5861-5871.

[7] J. R. Quinlan, C4.5: Programs for Machine Learning. Morgan Kaufmann Series in Machine Learning, 1992.

[8] F. Sebastiani, ”Machine learning in automated text categorization.” ACM Computation Survey 34, 1, March 2002, pp. 1-47.

[9] A. N. Srivastava and M. Sahamimph, Text Miming: Classification, Clustering, and Applications. Chapmann and Hall/CRC, New York, USA, 2009.

[10] J. Su and H. Zhang, ”A Fast Decision Tree Learning Algorithm.” In: Proceedings of the 21st National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference, July 16-20, Boston (MA), USA. AAAI Press, 2006.

[11] N. Tripathi, M. Oakes and S. Wermter, ”A fast subspace text categorization method using parallel classifiers.” Lecture Notes in Computer Science, Vol. 7182, No. 2, 2012, pp. 132-143.

[12] C. Ulmer, M. Gokhale, B. Gallagher, P. Top, and T. Eliassi-Rad, ”Massively parallel acceleration of a document-similarity classifier to detect web attacks.” Journal of Parallel and Distributed Computing, Vol. 71, No. 2, 2011, pp. 225-235.

[13] J. Žižka and F. Dařena, ”Automatic Sentiment Analysis Using the Textual Pattern Content Similarity in Natural Language.” Lecture Notes in Computer Science No. 6231, Springer, Heidelberg, Germany, 2010, pp. 224-231.

[14] J. Žižka and F. Dařena, ”Mining Textual Significant Expressions Reflecting Opinions in Natural Languages.” In: Proceedings of the 11th International Conference Intelligent Systems Design and Applications. Cordoba, Spain, November 22-24, 2011, pp. 136-141.

[15] J. Žižka and F. Dařena, ”Mining Significant Words from Customer Opinions Written in Different Natural Languages.” Lecture Notes in Computer Science No. 6836, Springer, Heidelberg, Germany, 2011, pp. 211–218.

[16] J. Žižka and V. Rukavitsyn, ”Automatic Categorization of Reviews and Opinions of Internet E-Shopping Customers.” International Journal of Online Marketing, Vol. 1 No. 2, IGI Global, USA, 2011, pp. 68-77.