Distributed Fuzzy Rough Prototype Selection for Big Data Regression

Sarah Vluymans*†, Hasan Asfoor ‡, Yvan Saeys†§, Chris Cornelis*¶,
Matthew Tolentino‡, Ankur Teredesai‡ and Martine De Cock‡*
*Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Gent, Belgium
Email: {Sarah.Vluymans, Chris.Cornelis, Martine.DeCock}@ugent.be
† VIB Inflammation Research Center, Zwijnaarde, Belgium
Email: {Sarah.Vluymans, Yvan.Saeys}@irc.vib-ugent.be
‡Center for Data Science, Institute of Technology, University of Washington Tacoma, Tacoma, USA
Email: {asfoorhm, metolent, ankurt, mdecock}@uw.edu
§Department of Respiratory Medicine, Ghent University, Gent, Belgium
Email: Yvan.Saeys@ugent.be
¶Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain
Email: Chris.Cornelis@decsai.ugr.es

***Abstract*—Size and complexity of Big Data requires advances in machine learning algorithms to adequately learn from such data. While distributed shared-nothing architectures (Hadoop/Spark) are becoming increasingly popular to develop such new algorithms, it is quite challenging to adapt existing machine learning algorithms. In this paper, we propose a solution for big data regression, where the aim is to learn the regression model over large high-dimensional datasets. First, a new distributed implementation of the weighted kNN regression method is presented followed by a novel distributed prototype selection method based on fuzzy rough set theory. Experiments demonstrate that our implementations in Apache Spark for the proposed distributed algorithms handle the size and complexity of modern real-world datasets well. We furthermore show that application of our prototype selection method improves the regression accuracy.**

I. Introduction

The k-nearest neighbor classifier (kNN, [1]) is a simple and popular classification algorithm, assigning newly presented instances to the most prevalent class among their k nearest neighbors. This method stores its training set in full instead of constructing a classification model. The stored instances are denoted as *prototypes* and form the selection pool for the classification of unlabeled elements. This setup has several disadvantages. The storage of the full dataset can require a considerable amount of memory. A complete pass through it when detecting the nearest neighbors for each test instance also results in high execution times. Finally, the possible presence of noise is ignored, as all elements can act as neighbors and receive an equal weight in the classification process.

Prototype Selection (PS, [2]) has been proposed to deal with these issues. PS methods reduce the training set to a subset, aiming at noise removal or high storage reduction, or both. The application of PS is performed in a preprocessing step. Afterwards, only the reduced prototype set is used by kNN.

In this paper we focus on a PS technique based on fuzzy rough set theory. Rough set theory [3] was introduced as a framework for dealing with inconsistencies in information systems, i.e. the presence of instances with the same values for the input attributes but different outcomes. Fuzzy rough set theory [4] is an extension of rough set theory that allows both input and output attributes to be continuous without imposing a need for discretization like traditional rough set theory does. In addition, in fuzzy rough set theory instances can be inconsistent to a degree between 0 and 1.

Verbiest et. al. [5] proposed a prototype selection method based on fuzzy rough set theory (FRPS) with a good performance for kNN classification. In the experimental comparison with the state-of-the-art prototype selection algorithms, FRPS came out as the best method of all in terms of prediction accuracy. Their technique uses a wrapper framework. In a first step, the method evaluates the quality of all instances, based on a given measure. Secondly, it determines an ideal threshold on the quality value of instances to be included, that is, it aims to include only high-quality elements and searches for the threshold above which instances are deemed to be so, avoiding the need of a user-specified value. The final prototype set consists of instances for which the quality exceeds the determined threshold. This method has only been developed and tested on small-to-medium size datasets. Its scalability is limited due to the need to evaluate all calculated quality values as candidate thresholds, which requires a kNN classification process of the entire training set for each candidate, a significant limitation that we address. Furthermore, the proposed quality measure uses the fuzzy rough lower approximation operator. It has been noted that rough approximations do not scale well to big data (e.g. [6]). Their fuzzy rough extensions experience similar (and more complex) issues [7].

Since the current need for big data scenarios is not addressed in the original FRPS implementation, nor does the algorithm lend itself to a straightforward extension to large datasets, we make these contributions in this paper. It is now accepted that standard machine learning techniques are unable to handle large datasets, as a result of their high volume and complexity

978-1-4673-7248-0/15/$31.00 ©2015 IEEE

and this is often termed the big data problem [8]. A popular distributed approach to handle big data is the MapReduce scheme [9]. The procedure splits the dataset in a number of data partitions, which are processed separately in parallel (map-phase). A reduce-phase aggregates the results.

In this paper, we propose a prototype selection algorithm with a similar setting as FRPS. However, to further distinguish our work, our focus will be on regression instead of classification; implying that we deal with a real-valued rather than discrete outcome. This is taken into account in the development of a suitable quality measure. We thus provide a distributed approach to compute the fuzzy rough lower approximation for regression. We develop a distributed implementation of our algorithm and kNN within the MapReduce technique, using the open source framework Apache Spark [10]. This allows these methods to deal with large datasets, which the sequential versions of corresponding algorithms cannot process in reasonable time. As an example application, we evaluate our proposal on healthcare data which has both scale and complexity issues typical of a big data workload.

In this paper, Section II describes the kNN regression method on which we focus. Section III introduces our prototype selection method. Distributed implementations of the fuzzy lower approximation, the PS algorithm and the kNN regression method are presented in Section IV. In Section V we experimentally evaluate our proposal. We discuss our findings and outline future work directions in Section VI.

II. Regression problems

We focus on regression problems. This is a supervised learning task aimed at predicting a real-valued outcome $d(y)$ of a target y. The prediction is denoted as $\widehat{d(y)}$. In this work, we use a weighted kNN regression algorithm [11]. When estimating the outcome of a target y, its k nearest prototypes are determined and are assigned different weights according to their distance to y. The closer a neighbor, the higher its weight. As such, more distant neighbors have a smaller influence on the outcome estimation. We follow [12] and set the weight of the ith neighbor to

$$w_i = \frac{D_{max} - D_i}{D_{max} - D_{min}},$$

where D_i is the distance of the ith neighbor to y and D_{max} and D_{min} are the distance of the target to its furthest, respectively closest, neighbor. As noted in [12], this results in a zero-weight for the kth neighbor, such that we are effectively performing $(k-1)$NN. This should be taken into account in the applications: when 7 neighbors should be used, we should set $k = 8$. The value $\widehat{d(y)}$ is predicted as the weighted average of the outcomes of its neighbors.

A common way to compute the distance between two instances, which are defined by their attribute values, is by averaging their per-attribute distances. For numeric attributes, which we assume to be rescaled internally to the range [0,1], we use the Manhattan distance:

$$d_{num}(x,y) = \sum_{a \in A_{num}} |a(x) - a(y)|,$$

where A_{num} is the set of numeric input attributes. The distance $d_{nom}(x,y)$ between x and y based on their nominal attributes is set to the number of these attributes for which their values differ. The overall distance $d(x,y)$ between two elements x and y can therefore be given as

$$d(x,y) = \frac{d_{num}(x,y) + d_{nom}(x,y)}{m},$$

where m is the number of input attributes. To stress the flexibility of our proposal, we point out that our presented distributed approach however does not rely on being able to compute the distance between instances on a per-attribute basis. If the application domain requires so, a set of attributes can be grouped and similarity can be computed over a set of attributes as a whole instead of over each attribute individually. For example, while Boolean attributes can be treated as nominal attributes in the way described above, one might also opt to interpret an entire group of Boolean attributes as a set B: if the value is `true`, the attribute is present in the set, if the value is `false`, it is not. These sets B can be compared using Jaccard similarity [13] and the distance based on Boolean attributes set to

$$d_{bool}(x,y) = 1 - \frac{|B(x) \cap B(y)|}{|B(x) \cup B(y)|},$$

where $B(x)$ is the set corresponding to instance x. In this case, the overall distance is given by

$$d(x,y) = \frac{d_{num}(x,y) + d_{nom}(x,y) + d_{bool}(x,y)}{m+1}$$

in which m is the total number of numerical and nominal (non-Boolean) attributes. This distance measure is used in the experimental section.

III. High-Mid-Low prototype selection

In this section, we introduce the new proposed PS approach to improve the performance of kNN. As stated earlier, the FRPS algorithm of [5] involves two important steps:

1) Determine quality $Q(x)$ of every training instance x.
2) Find a threshold τ such that the corresponding set $S_\tau = \{x \,|\, Q(x) \geq \tau\}$ results in the highest classification accuracy of kNN in the search space. In case of ties, the minimum τ value is used.

Our proposed method follows the same scheme.

A. Instance quality measure

In [5], the quality of an instance equals its membership degree to the fuzzy rough lower approximation of its decision class. In regression problems, the outcome is real-valued, so a modification is required. We divide the range of decision values and define three fuzzy sets, which can be interpreted as three fuzzy classes. They correspond to the high, medium

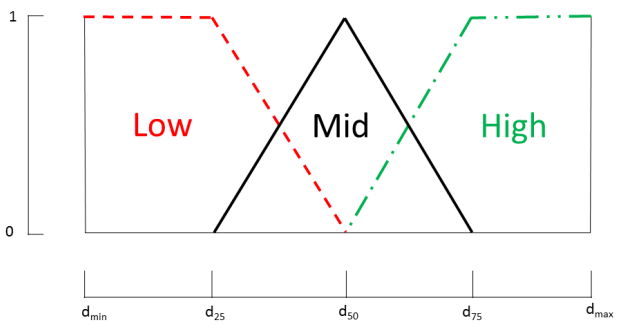

Fig. 1. Membership functions of elements to the HML sets, based on the value for the decision attribute.

and low (HML) ranges of this value. For each instance, we compute its membership degree to the lower approximation of the three fuzzy sets. The final quality is computed as the maximum of these three values, which is the intuitive generalization of the membership degree of the instance to its own decision class.

1) HML approach: Let d be the numeric decision attribute. We set d_{range} to the difference between the highest and lowest decision value observed in the training set. Based on this value, we determine the points d_{25}, d_{50} and d_{75} within the observed interval as

$$\begin{aligned} d_{25} &= d_{min} + 0.25 \cdot d_{range} \\ d_{50} &= d_{min} + 0.50 \cdot d_{range} \\ d_{75} &= d_{min} + 0.75 \cdot d_{range}, \end{aligned}$$

where d_{min} is the minimum observed decision value. These points allow us to define the three fuzzy sets and the membership degrees of instances to them. The membership value of an instance x to the fuzzy set $low(\cdot)$ is defined as follows:

$$low(x) = \begin{cases} 1 & \text{if } d(x) \leq d_{25} \\ \frac{d(x)-d_{50}}{d_{25}-d_{50}} & \text{if } d_{25} < d(x) < d_{50} \\ 0 & \text{if } d(x) \geq d_{50}. \end{cases}$$

The piecewise linear membership functions of $mid(\cdot)$ and $high(\cdot)$ are defined similarly, as presented in Fig. 1.

2) Fuzzy rough lower approximation: In the next step, we compute the fuzzy rough lower approximations la^{low}, la^{mid} and la^{high} of the three fuzzy sets. An instance x belongs to the fuzzy rough lower approximation of a fuzzy set C to the degree to which all elements similar to x belong to C. The lower approximation is a fuzzy set, to which the membership degree is defined as

$$(R \downarrow C)(x) = \min_{y \in T} \mathcal{I}(R(x,y), C(y)), \tag{1}$$

where T represents the training set, $\mathcal{I}$ is a fuzzy implication and the function $R(\cdot,\cdot)$ computes the similarity degree between two instances in T. We measure the similarity between two instances based on the distance between them, i.e.

$$R(x,y) = 1 - d(x,y). \tag{2}$$

3) Quality vector: As a final step, we determine the quality measure $Q(x)$ of an instance x as its maximum membership degree to the lower approximation of one of these sets, that is, $Q(x) = \max(la^{low}(x), la^{mid}(x), la^{high}(x))$.

B. Prototype selection

The instance quality values $Q(\cdot)$ are used to obtain a prototype set with high-quality instances. A threshold is determined by evaluating 10 candidates. This differs from the setup of [5], in order to avoid the time-consuming step of testing all computed quality values as candidates. The first value is set to the overall highest quality. Each next one is taken by decrementing the current value by $(\text{highest quality} - \text{lowest quality})/10$.

For each threshold τ, the subset $S_\tau \subseteq T$ is constructed, consisting of instances for which the quality is not lower than τ. The elements of S_τ are used as prototypes in kNN. We apply 10-fold cross-validation to calculate an estimation error made by using this threshold. As error measure, we use the root mean squared error (RMSE), which, when predicting the outcome of elements in a set X, is defined as

$$RMSE = \sqrt{\frac{\sum_{x \in X} (\widehat{d(x)} - d(x))^2}{n}},$$

where $d(x)$ is the real outcome of x. The final threshold is chosen as the one with the minimal RMSE.

IV. Distributed Implementation

In this section, we provide the details on the distributed implementations of all components of our proposed approach. The computations of both the PS algorithm and the regression method are distributed. In a first stage, we compute the quality measures needed for PS in a distributed way. Next, determining the optimal threshold, which results in the reduced prototype set, as well as the final prediction step is handled by a distributed implementation of the kNN regression method. We develop the algorithm and demonstrate its scalability on Apache Spark [10]. Spark is installed on a cluster of compute nodes as a distributed processing system that implements the MapReduce model. To process a typically large dataset, Spark splits the dataset into smaller partitions with error recovery and other main-memory management components all encapsulated in a concept termed: resilient distributed dataset (RDD) [14] over multiple nodes. One important feature about RDDs is that they can be cached in memory so that they can span multiple map and reduce steps. Spark can transform the data in the RDD from one form to another through Map calls. The data in an RDD can also be aggregated through a call to a Reduce function. When the output is a list of elements, the alternative Grouping function is used.

A. Distributed quality measure computation

To the best of our knowledge, we were the first to present a distributed approach to calculate fuzzy rough approximations in [7]. It provides a distributed approach for calculating these approximations in the Message Passing Interface. We refer the

interested reader to this paper for a more detailed description of the workload distribution, which cannot be included here due to space limitations. Other previous efforts in this area either do not calculate the (fuzzy) rough lower and upper approximations explicitly [15], [16], [17], [18] or are limited to the non-fuzzy setting [19], [20], [21], [22]; a significant distinction. The main difference between the crisp and fuzzy cases, is that the crisp one involves a compact representation of similarity by means of equivalence classes. The fuzzy case allows degrees of similarity and does not have that advantage.

Below, we assume the availability of a cluster with a master node and e slave nodes, as well as the accessibility of the whole dataset to every slave node (by simply keeping a copy on each node's disk storage). In addition, it is needed that every slave node has enough memory to store all proposed data structures. We equally distribute the computational load over the available slave nodes such that each performs a part of the computation: each slave node processes $\lceil \frac{n}{e} \rceil$ of the rows (elements) in the dataset.

In the similarity calculation (2), we assume the numerical attribute values to be rescaled to the range [0,1]. Rescaling of an attribute a can be achieved by determining its minimal and maximal values a_{min} and a_{max} in the dataset and setting

$$a(x) \leftarrow \frac{a(x) - a_{min}}{a_{max} - a_{min}} \tag{3}$$

for all elements x. This is done in a preprocessing MapReduce phase. For each attribute, the minimal and maximal values need to be determined. Each node calculates these values within its partition. The overall minimal and maximal values are determined as the minimum and maximum over all nodes. In a second stage of the process further modification (3) is applied in all partitions.

A further preliminary step requires the computation of the three class membership vectors low, $medium$ and $high$. To this end, the range of the decision attribute needs to be computed. The maximal and minimal value of this attribute can again be determined within each partition, setting the overall results to the maximum and minimum over all nodes. These two values are the only requirements to calculate the membership degrees to low, $medium$ and $high$ for all instances in each partition.

Each node computes $\lceil \frac{n}{e} \rceil$ columns of the $n \times n$ similarity matrix, namely the submatrix corresponding to its assigned instances. Using these computed columns and the class membership degrees, each node generates a partial lower approximation: it restricts the minimum in Eq. (1) to the available instances. The final lower approximation is determined as the element-wise minimum over each node. The final step is the calculation of the quality measure. This is a straightforward operation, as it consists of taking the maximum of the three lower approximation membership degrees for each instance. No interaction between instances is required.

B. Distributed kNN

We use the kNN method both within the PS algorithm to evaluate candidate thresholds as well as in the final regression step. Previous work on distributed kNN includes the use of an approximate kNN graph [23] and a tree-based structure [24], [25]. We can find a kNN implementation in Apache Spark in [26]. However, it requires a large amount of memory as it stores the entire distance matrix. Zhang et al [27] proposed a Hadoop-based implementation that divides both training and test sets into partitions and applying a join operation on them to compute distances. This operation has a certain overhead, which we avoid in our implementation by partitioning only the training set similar to the brute-force approaches [28]. These approaches also end up applying a sort operation to an already sorted nearest neighbor list. Instead we use a fixed-size heap data structure to maintain a kNN list for every instance in the test set implemented on a shared-nothing big data architecture previously described for a shared-memory multi-GPUs architecture [29].

Our distributed implementation of the weighted kNN regression algorithm works as follows. The input to the program are a training set and a test set. The test set contains the instances for which we want to find k nearest neighbors and compute prediction accuracy. The training set contains instances from which to select the neighbors. For a given test instance, we distribute the computation of the distance matrix over multiple nodes, which each contain a partition of the training set. Within each node, we determine the k nearest neighbors to the given target. If we have e nodes, we end up with a set of $e \times k$ computed neighbors. A grouping stage combines all of the $e \times k$ nearest neighbors of a test instance, sorts them and returns the top k as its final neighbors. The weighted average of these k neighbors is computed and stored as the final result.

V. Experimental results

The experimental study is divided into two main parts. First, we assess the scalability of our approach. Next, we consider its strength in improving the performance of the weighted kNN regression method. All experimental results were obtained using the Kleene-Dienes implicator. It can easily be substituted with other implicators, without significant effect on the observed runtimes.

A. Scalability

The scalability of our approach is assessed by means of synthetic datasets with varying sizes and dimensions. The attribute values of these datasets are randomly generated numbers between 0 and 1000. The decision value is taken as a random number between 0 and 1. This implies no prior correspondence between the attributes and decision value, which is why we solely use these datasets in the scalability experiments. The experiments are performed on a cluster with 1 master node and 8 slave nodes, using 46 computing threads. We use a multi-threaded implementation reading multiple lines from the test set at once and equally distributing them over the available threads.

In a first stage, we tested the scalability of our distributed implementation of kNN on these synthetic datasets. In Fig. 2,

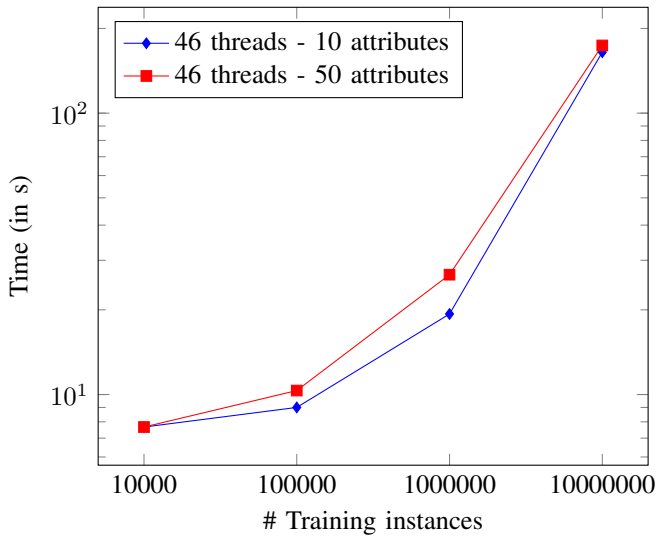

Fig. 2. Execution time (in sec) of for our kNN implementation ($k = 10$) with a varying number of instances in the training set. The test set always contains 10000 instances. The results are averages over 3 runs, each time generating a new dataset.

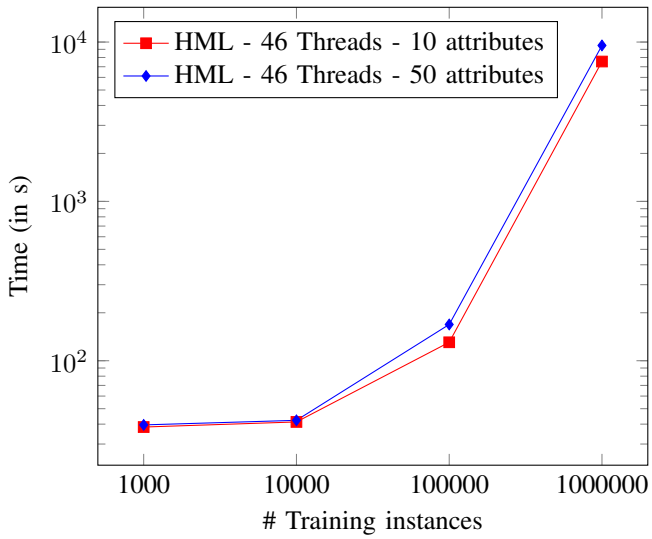

Fig. 3. Execution time (in sec) for the HML PS method with a varying number of instances in the training set. The time includes the calculation of all quality values and the procedure to find the optimal threshold.

we graphically display the execution time of 10NN on synthetic datasets with 10000, 100000, 1 million and 10 million instances and 10 and 50 attributes. The execution time grows approximately quadratically in terms of the number of instances in the dataset. Clearly, the proposed method is able to handle large datasets, processing a dataset of 10 million instances in 165 seconds (10 attributes) and 174 seconds (50 attributes) respectively.

In Fig. 3 we show the effect of a change in dataset size on the execution time of the distributed HML PS algorithm, which involves all steps to compute the quality values and the procedure to find the optimal threshold. Experiments were conducted for datasets with 10 and 50 attributes and sizes ranging from a thousand to a million instances. Again, a change in the number of instances in the dataset is shown to have an approximately quadratic effect on the runtime. Datasets of one million instances were processed in 7548 seconds (10 attributes) and 9520 (50 attributes) seconds.

B. Accuracy

We use two large healthcare datasets: the Washington State Inpatient Database[1] (SID) and the Medical Expenditure Panel Survey[2] (MEPS). SID is a dataset of 320395 instances provided by the Healthcare Cost and Utilization Project (HCUP) which contains clinical and claims data, collected through a partnership between HCUP and the Washington State Department of Health. The preprocessed dataset contains age (numeric), gender (nominal), race (nominal), 773 numeric diagnosis attributes indicating the number of times a diagnosis was made, and 29 Boolean comorbidity (other related diseases) indicators (T or F). The 29 Boolean comorbidity indicators are represented as a string of bits to allow for an efficient computation of the Jaccard distance. A final numeric attribute is the aggregated patient charge over the first three quarters (January through September) of 2010. The objective is the prediction of the cost for the fourth quarter (October through December). MEPS consists of data extracted from responses to panel surveys given to households and their employers, medical providers and insurance providers over two year periods (2011-2012). The preprocessed version of the MEPS dataset contains 14039 instances with 569 numeric attributes representing the number of visits related to a certain medical condition, and the total cost of a patient for 2011. The prediction goal is the total cost of the same patient in 2012. All numeric attributes in both datasets are rescaled to [0,1] internally. The preprocessing of these datasets and their wide applicability for healthcare cost prediction as a regression problem is further described in [30].

In order to evaluate the performance of our method in the presence of an increasing amount of noise, we create noisy versions of both datasets: by selecting a predefined percentage of instances and perturbing their outcome by adding a number drawn from standard Gaussian distribution multiplied with the standard deviation of the outcome in the entire dataset.

Table I presents our experimental results. PS is traditionally combined with 1NN [2], but we include results for 10NN as well. The table computes the ratio of the RMSE of kNN over the RMSE of kNN after PS. The RMSE values were computed by 10-fold cross-validation. Values strictly larger than 1 indicate an improvement after the application of PS. For both values of k and all datasets, an improvement after PS is evident, showing the strength of our proposed algorithm. The improvement generally increases with the noise level, which shows that our method is able to adequately handle the imposed noise. We further present the reduction obtained after PS. For 1NN, this value is very high all round, implying that our method is able to considerably reduce storage require-

[1]http://www.hcup-us.ahrq.gov/sidoverview.jsp
[2]http://meps.ahrq.gov/mepsweb

TABLE I
RATIO (RAT.) OF THE RMSE OBTAINED BY kNN WITHOUT PS OVER THE RMSE OF kNN WITH PS. STRICT IMPROVEMENTS OF THE USE OF HML+kNN OVER kNN ARE PRINTED IN BOLD. THE COLUMN RED. PRESENTS THE REDUCTION AFTER PS. COLUMN TIME LISTS THE EXECUTION TIME IN SECONDS OF PS+kNN.

		$k=1$			$k=10$		
Dataset	% noise	Red.(%)	Rat.	Time	Red.(%)	Rat.	Time
SID	0	99.9963	**1.2872**	1092.18	70.2196	**1.0006**	1147.18
SID	10	99.9963	**1.2983**	1065.59	98.7768	**1.0055**	1133.59
SID	20	99.9963	**1.3093**	1059.96	98.7768	**1.0080**	1099.96
SID	30	99.9963	**1.3099**	1089.28	99.9913	**1.0097**	1097.28
SID	40	99.9959	**1.3324**	1853.81	46.1739	**1.0004**	1885.81
SID	50	99.8817	**1.2583**	1084.65	98.7768	**1.0125**	1104.65
MEPS	0	99.9929	**1.1682**	39.74	0.0142	**1.0032**	40.74
MEPS	10	99.9929	**1.1948**	32.17	0.0142	**1.0028**	33.17
MEPS	20	99.9929	**1.2235**	31.94	0.0142	**1.0025**	33.94
MEPS	30	99.9929	**1.2565**	32.12	0.0142	**1.0021**	34.12
MEPS	40	99.6367	**1.0348**	31.96	0.0142	**1.0017**	33.96
MEPS	50	99.9929	**1.2966**	32.26	99.9929	**1.0127**	33.26

ments. For 10NN, the reduction is still quite high for the SID data, but small for most MEPS datasets. The lower reduction rate is explained by the lower susceptibility of kNN to noise for higher values of k. As a result, fewer instances are removed to increase the performance of the regression method. Only for the version in which 50% of the instances were perturbed, does the reduction on the MEPS data attain a high level.

VI. CONCLUSION

In this paper, we presented a distributed implementation of weighted kNN regression method and paired it with a new distributed PS algorithm based on fuzzy rough set theory specifically for regression problems. We experimentally validated their scalability and demonstrated that our PS method is able to improve the performance of kNN. In the future we plan to extend the original quality measure of the FRPS method [5] to enable regression over very large datasets and compare it with algorithms described in this paper.

REFERENCES

[1] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[2] S. Garcia, J. Derrac, J. Cano, and F. Herrera, "Prototype selection for nearest neighbor classification: Taxonomy and empirical study," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 3, pp. 417–435, 2012.

[3] Z. Pawlak, "Rough sets," *International Journal of Computer Information Science*, vol. 11, pp. 341–356, 1982.

[4] D. Dubois and H. Prade, "Rough fuzzy sets and fuzzy rough sets," *International Journal of General Systems*, vol. 17, pp. 191–209, 1990.

[5] N. Verbiest, C. Cornelis, and F. Herrera, "A prototype selection method based on ordered weighted average fuzzy rough set theory," in *Proceedings of the 14th International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, 2013, pp. 180–190.

[6] J. Zhang, T. Li, D. Ruan, Z. Gao, and C. Zhao, "A parallel method for computing rough set approximations," *Information Sciences*, vol. 194, pp. 209–223, 2012.

[7] H. Asfoor, R. Srinivasan, G. Vasudevan, N. Verbiest, C. Cornelis, M. Tolentino, A. Teredesai, and M. De Cock, "Computing fuzzy rough approximations in large scale information systems," in *2014 IEEE International Conference on Big Data (Big Data)*. IEEE, 2014, pp. 9–16.

[8] P. Zikopoulos and C. Eaton, *Understanding big data: Analytics for enterprise class Hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[9] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[10] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010, pp. 10–10.

[11] T. Mitchell, "Machine learning. WCB," 1997.

[12] A. Troncoso Lora, J. Riquelme Santos, A. Gómez Expósito, J. Martínez Ramos, and J. Riquelme Santos, "Electricity market price forecasting based on weighted nearest neighbors techniques," *IEEE Transactions on Power Systems*, vol. 22, no. 3, pp. 1294–1301, 2007.

[13] A. Rajaraman and J. Ullman, *Mining of massive datasets*. Cambridge University Press, 2011.

[14] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[15] F. Xu, L. Wei, Z. Bi, and L. Zhu, "Research on fuzzy rough parallel reduction based on mutual information," *Journal of Computational Information Systems*, vol. 10, no. 12, pp. 5391–5401, 2014.

[16] Y. Fan and C. Chern, "An agent model for incremental rough set-based rule induction: a big data analysis in sales promotion," in *Proceedings of the 46th Hawaii International Conference on System Sciences*, 2013, pp. 985–994.

[17] Y. Yang, Z. Chen, Z. Liang, and G. Wang, "Attribute reduction for massive data based on rough set theory and MapReduce," in *Proceedings of the Conference on Rough Sets and Knowledge Technology*, 2010, pp. 672–678.

[18] J. Zhang, T. Li, and Y. Pan, "Parallel rough set based knowledge acquisition using MapReduce from Big Data," in *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, 2012, pp. 20–27.

[19] J. Zhang, T. Li, D. Ruan, Z. Gao, and C. Zhao, "A parallel method for computing rough set approximations," *Information Sciences*, vol. 194, pp. 209–223, 2012.

[20] J. Zhang, J. Wong, T. Li, and Y. Pan, "A comparison of parallel large-scale knowledge acquisition using rough set theory on different MapReduce runtime systems," *International Journal of Approximate Reasoning*, vol. 55, pp. 896–907, 2014.

[21] Y. Yang and Z. Chen, "Parallelized computing of attribute core based on rough set theory and MapReduce," in *Proceedings of the Conference on Rough Sets and Knowledge Technology*, 2012, pp. 155–160.

[22] J. Zhang, Y. Zhu, Y. Pan, and T. Li, "A parallel implementation of computing composite rough set approximations on GPUs," in *Proceedings of the Conference on Rough Sets and Knowledge Technology*, 2013, pp. 240–250.

[23] R. Mall, V. Jumutc, R. Langone, and J. Suykens, "Representative subsets for big data learning using k-NN graphs," *Proc. of IEEE Big Data 2014*, no. accepted, 2014.

[24] F. Gieseke, J. Heinermann, C. Oancea, and C. Igel, "Buffer kd trees: processing massive nearest neighbor queries on GPUs," in *Proceedings of The 31st International Conference on Machine Learning*, 2014, pp. 172–180.

[25] J. Pan and D. Manocha, "Fast GPU-based locality sensitive hashing for k-nearest neighbor computation," in *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2011, pp. 211–220.

[26] M. Parsian, *Data Algorithms: Recipes for Scaling up with Hadoop and Spark*. O'Reilly Media, 2014.

[27] C. Zhang, F. Li, and J. Jestes, "Efficient parallel kNN joins for large data in mapreduce," in *Proceedings of the 15th International Conference on Extending Database Technology*. ACM, 2012, pp. 38–49.

[28] B. Aydin, "Parallel algorithms on nearest neighbor search."

[29] K. Kato and T. Hosino, "Solving k-nearest neighbor problem on multiple graphics processors," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 769–773.

[30] S. Sushmita, S. Newman, J. Marquardt, P. Ram, V. Prasad, M. De Cock, and A. Teredesai, "Population cost prediction on public healthcare datasets," to appear in: Proceedings of ACM Digital Health 2015 (5th International Conference on Digital Health), 2015.