The Effect of Parallelism on Data Reduction

Pavlos Ponos
pponos@uom.edu.gr
Dept. of Applied Informatics
School of Information Sciences
University of Macedonia
Thessaloniki, Greece

Stefanos Ougiaroglou*
stoug@uom.edu.gr
Dept. of Information Technology
Alexander TEI of Thessaloniki
Sindos, Greece

Georgios Evangelidis
gevan@uom.gr
Dept. of Applied Informatics
School of Information Sciences
University of Macedonia
Thessaloniki, Greece

ABSTRACT

In this paper, we investigate the effect of parallelism on two data reduction algorithms that use k-Means clustering in order to find homogeneous clusters in the training set. By homogeneous, we refer to clusters where all instances belong to the same class label. Our approach divides the training set into subsets and applies the data reduction algorithm on each separate subset in parallel. Then, the reduced subsets are merged back to the final reduced set. In our experimental study, we split the datasets into 8, 16, 32 and 64 subsets. The results obtained reveal that parallelism can achieve very low preprocessing costs. Also, when the number of subsets is high, in some datasets the accuracy of k-NN classification is almost equal (if not better) to the one achieved when using the standard execution of the reduction algorithms, with a small loss in the reduction rate.

CCS CONCEPTS

• **Information systems → Nearest-neighbor search**; • **Theory of computation → Parallel algorithms**; • **Computing methodologies → Feature selection**.

KEYWORDS

k-NN Classification, Data Reduction, Prototype Merging, Parallel Implementation, Clustering

ACM Reference Format:
Pavlos Ponos, Stefanos Ougiaroglou, and Georgios Evangelidis. 2019. The Effect of Parallelism on Data Reduction. In *9th Balkan Conference in Informatics (BCI'19), September 26–28, 2019, Sofia, Bulgaria.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3351556.3351584

1 INTRODUCTION

The k Nearest Neighbor (k-NN) is an extensively used lazy classifier. It classifies a new item x by searching in the training set (TS) for the k nearest items (neighbors) to x according to a distance metric (e.g. Euclidean distance). Then, x is assigned to the most common class determined via a majority vote of the retrieved k nearest neighbors. Ties are resolved either randomly or by the single nearest neighbor.

k-NN classifier has two major drawbacks when it comes to datasets of very large size: (i) training data have to be maintained, which leads to high storage requirements, and, (ii) the computational cost is high, since between every new (unclassified) item and the training data, all distances have to be calculated.

Many data reduction techniques have been proposed in the literature, which can cope with both weaknesses by building a small representative training dataset, called the condensed set. All these methods reduce datasets with labeled examples. The advantage of applying k-NN to a smaller set is two-fold: (i) lower computational cost is involved and (ii) less storage is required. Data reduction techniques can be distinguished to prototype selection algorithms [2] and to prototype generation algorithms [10]. The former choose some representatives from the training set and put them into the condensed set, while the latter generate representatives by summarizing similar training set items. Another point worth mentioning is that most of these algorithms are sensitive to noise [6]. Hence, for noisy datasets an editing algorithm [11] for noise removal must be applied beforehand.

According to [8] and [7], Reduction through finding Homogeneous Clusters (RHC) and Editing and Reduction through finding Homogeneous Clusters (ERHC) prototype generation algorithms run very fast by achieving high reduction rates, as well as high accuracy levels. The motivation of the present work was to test whether parallel and distributed execution of RHC and ERHC on subsets of the original data, as well as merging of the resulting reduced sets to a final reduced set is a feasible approach. In other words, we apply a Map Reduce paradigm approach and we want to investigate whether, in addition to the speed up that we expect to be very high, the quality of the merged reduced set will be similar to the reduced set that results from the application of RHC and ERHC on the full original training set. We tested the effect of parallelism using a predefined number of subsets (8, 16, 32 and 64). Each subset was assigned to a different processor/thread and RHC or ERHC algorithm was executed in all subsets in parallel. Finally, k-NN was applied both on the reduced set that is derived from the original training set and the reduced set that is produced by merging all the reduced subsets that are created in parallel. The experimental analysis shown in Section 5.2 reveals that the parallel execution of RHC and ERHC speeds-up the process, while maintaining accuracy and reduction rate at almost the same levels.

The rest of the paper is structured as follows: Section 2 discusses the background knowledge and Section 3 reviews RHC and ERHC. Section 4 considers in detail the proposed algorithms. In Section 5, the parallel implementations of RHC and ERHC are

*Stefanos Ougiaroglou is also a Postdoctoral Researcher at the University of Macedonia.

BCI'19, September 26–28, 2019, Sofia, Bulgaria

ACM ISBN 978-1-4503-7193-3/19/09…$15.00
https://doi.org/10.1145/3351556.3351584

compared against RHC and ERHC on fifteen datasets. Section 6 concludes the paper and proposes directions for future work.

2 BACKGROUND KNOWLEDGE

The problem of efficiently handling Big Data has attracted the interest of the research community in the recent years. The already collected historical data and the vast amounts of data that are continuously collected by modern big data systems need to be reduced in size in order to be appropriate for analysis.

[9] contains a detailed survey on Big Data reduction methods that addresses the full spectrum of challenges specific to Big Data, namely, dimension reduction techniques, data reduction methods and algorithms for preprocessing, cluster-level data deduplication, redundancy elimination, and implementation of network (graph) theory concepts. The survey also differentiates between methods that are applicable at the post-data collection phases and methods that reduce big data during the collection process [5].

The idea of reducing the size of big data is crucial in lazy classification tasks, like k-NN. Such classifiers benefit a lot when running by using relatively small training datasets. The problem can be addressed both at the post-data collection phase, i.e., by applying any popular data reduction technique on the collected data, and during the collection process, i.e., when data comes from independent streams (see surveys at [2], [10]).

Because of the enormous size of the collected data, parallelization and distributed computation is a natural approach to data reduction, using technologies like Hadoop, Pig, Hive and Spark ([12], [4]).

Our approach is applicable to any such framework and deals with data reduction at the post-data collection phase.

3 THE RHC AND ERHC ALGORITHMS

Reduction through Homogeneous Clusters (RHC) [8] is a prototype generation algorithm for data (instance) condensing, whereas, ERHC [7] is an extension of the aforementioned algorithm that simultaneously performs data condensing and editing. Both algorithms are based on a procedure that forms clusters containing items of a specific class only, i.e., homogeneous clusters. Initially, the whole training set is considered to be a single non-homogeneous cluster and the algorithm recursively splits it in as many clusters as the existing class labels in the cluster. For both RHC and ERHC, when a cluster becomes homogeneous, it is replaced by its centroid. For ERHC, in addition, single item clusters are discarded.

Algorithm 1 presents the pseudo-code of a possible implementation of RHC and ERHC. The only difference between them (see line 7) is how they treat the homogeneous clusters [7].

For a given non-homogeneous cluster C, the class means for all the existing class labels in the cluster are computed by averaging the corresponding items of the cluster and are put in R (lines 9–12). Then, k-means clustering is applied on the cluster, with initial cluster centroids being the computed class means R and the outcome is $|R|$ clusters, which replace the cluster under processing (lines 13–16). This procedure is applied on all non-homogeneous resulting clusters (lines 4–18). Finally, RHC replaces each homogeneous cluster by its cluster centroid, whereas, ERHC in addition removes clusters that contain a single item, since they most probably correspond to noise and outliers (line 7).

Algorithm 1 RHC/ERHC

```
   Input: TS                                              ▷ Training Set
   Output: RS                                             ▷ Reduced Set
1: RS ← ∅                                                 ▷ Initialize the Reduced Set
2: Q ← ∅      ▷ Initialize a queue for the clusters to be processed
3: Enqueue(Q, TS)  ▷ the entire training set constitutes a cluster
4: repeat
5:    C ← Dequeue(Q)                                      ▷ Get the cluster from the queue
6:    if C is homogeneous then ▷ C holds items of the same class
7:       RS ← Reduce_Edit(RS, C) ▷ For (E)RHC: C is replaced by
   its class centroid. For ERHC: if |C|=1, C is removed
8:    else                                                ▷ C is non-homogeneous
9:       R ← ∅                                            ▷ R is the set of class means
10:      for each class label M in C do
11:         R ← R ∪ class_mean_of (M)      ▷ Compute the class
   mean or representative for each class label and put it in R
12:      end for
13:      clusters ← k-Means (C, R)  ▷ The elements of R are used
   as the initial cluster centroids for clustering C
14:      for each cluster r ∈ clusters do
15:         Enqueue(Q, r)  ▷ C is replaced by the |R| new clusters
16:      end for
17:   end if
18: until IsEmpty(Q)    ▷ until all clusters become homogeneous
19: return RS                                             ▷ The Reduced Set is returned
```

4 THE PROPOSED ALGORITHMS

The main idea of this paper is the parallel execution of prototype generation algorithms. We apply RHC and ERHC on the training part of the datasets presented in Table 1 after splitting it in smaller equally sized subsets, and then, we merge the resulting reduced subsets. The application of the algorithms on the subsets can in practice be performed in parallel, assuming that we have as many processing units as the number of subsets. We call these algorithms pmRHC and pmERHC, with the letter "p" denoting the parallel execution of the algorithms on the subsets and the letter "m" denoting the merging of the resulting reduced subsets back into a single merged reduced set.

In order to split the datasets we used the Stratified Remove Folds method from the WEKA data mining tool [3]. Stratified sampling is a technique where the entire dataset is segregated into different strata and then the final subjects are proportionally selected from them. It is worth mentioning that the strata must be non-overlapping, otherwise overlapping subjects have higher chances of being selected, which contradicts the concept of stratified sampling as a type of probability sampling [3].

The training part of the datasets was fed into the Stratified Remove Folds method, with the output being n subsets of the initial dataset. As an example, let us consider a dataset that consists of 20000 items. We form 5 folds, so the training part of the dataset consists of 16000 items. When this training set is split into, for example, 64 subsets, each subset will accommodate 250 items. In turn, the newly created 64 subsets will be the input to either RHC or ERHC algorithm (Algorithm 1). The 64 resulting reduced subsets are then merged to a single reduced set that is used as the training set. Figure 1 schematically depicts this process.

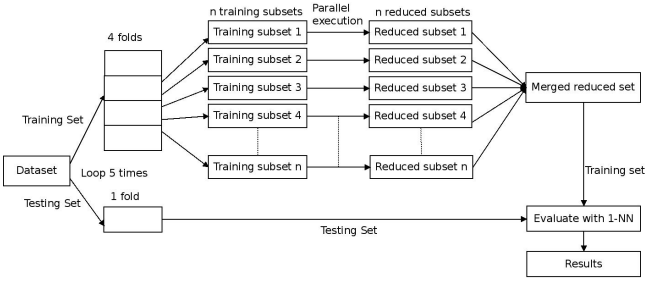

Figure 1: Schematic representation of pmRHC/pmERHC: the training part of the datasets is split into subsets, the reduction process is applied on each subset in parallel, and the reduced subsets are merged back into the final training set to be used in the cross validation.

5 PERFORMANCE EVALUATION

5.1 Experimental Setup

The performance of the new variations was tested against RHC and ERHC by using fifteen datasets distributed by the KEEL dataset repository[1] [1]. Table 1 summarizes the datasets used in this paper.

Table 1: Dataset description

Dataset	Size	Attributes	Classes
Letter Image Recognition (LIR)	20000	16	26
Magic G. Telescope (MGT)	19020	10	2
Pen-Digits (PD)	10992	16	10
Landsat Satellite (LS)	6435	36	6
Shuttle (SH)	58000	9	7
Texture (TXR)	5500	40	11
Phoneme (PH)	5404	5	2
Balance (BL)	625	4	3
Pima (PM)	768	8	2
Ecoli (ECL)	336	7	8
Yeast (YS)	1484	8	10
Twonorm (TN)	7400	20	2
MONK 2 (MN2)	432	6	2
KddCup (KDD)	141481	36	23
Banana (BN)	5300	2	2

The chosen distance metric was the Euclidean distance. All algorithms were implemented in C. All datasets except KDD were not normalized. Three average values were measured for each algorithm, via 5-fold cross validation. These values are *Accuracy (Acc)*, *Reduction Rate (RR)* and *Preprocessing cost (PC)*. *Acc* and *RR* measured as percentages (%) while *PC* in million distance computations.

A point worth mentioning is that due to the parallel execution of the reduction algorithms, the preprocessing costs are calculated using the equation below:

$$PC = \frac{PC_1 + PC_2 + \ldots + PC_n}{n} \quad (1)$$

where n is the number of subsets to which the initial training dataset is split and PC_i is the preprocessing cost for subset i.

Acc was estimated by running k-NN classification with $k = 1$ and *PC* was measured in terms of distance computations.

[1]http://sci2s.ugr.es/keel/datasets.php

5.2 Results and Discussion

The performance of RHC was compared against four variations of the parallel version of the algorithm (pmRHC), using 8, 16, 32 and 64 subsets respectively. The results can be found in Table 2, with the prevalent values highlighted in bold.

Due to the fact that the k-Means clustering, as presented in Algorithm 1, is executed in parallel to as many processors/threads as the available subsets, the preprocessing cost is significantly lower, especially in pmRHC64. As an example, in LIR dataset the *PC* is 41.844 while in pmRHC8, pmRHC16, pmRHC32 and pmRHC64 the *PC* is 1.7478, 0.6021, 0.1919 and 0.0629 respectively. In other words, the data reduction procedure is executed from 24 up to 665 times faster. This was expected since the repeated applications of k-Means on the full dataset start with a large dataset, whereas, the parallel executions start on much smaller subsets and converge much faster.

In addition, the lower the number of subsets to which the initial training set is split, the higher the loss in *Acc* in all datasets except BL and ECL. In turn, from Table 2, we can ascertain that *RR* deteriorates (apart from YS) when the number of subsets increases. For instance, in MN2 and LIR there is a drop in *RR* of almost 50%.

Similarly, in Table 2, one can compare the performance of ERHC against four variations of the parallel version of the algorithm (pmERHC). The parallelized variations have significantly lower preprocessing costs compared to ERHC, which means that there is speed-up in the process from 25 up to 650 times, taking into consideration the LIR dataset.

Alike in RHC, *Acc* declines when a small number of subsets is used and it is almost the same as in ERHC when the initial datasets are split into 64 subsets, if not better for BL, TXR, ECL, BN and YS datasets. Contrary to the RHC's variations, we noticed that notwithstanding the fact that *RR* is worse (except ECL), the proportion of the reduction is about 20% for MN2 and LIR, compared to the 50% in RHC's variations. A possible explanation could be the editing step that removes the noise from the datasets.

6 CONCLUSIONS AND FUTURE WORK

This paper introduces the parallel execution of RHC and ERHC where the initial training datasets are split into subsets that are then sent for processing to different processors/threads in a Map Reduce fashion. In theory, we would expect lower preprocessing costs, thus faster execution of the algorithm, compared to the execution of RHC and ERHC on the original training set, while keeping the accuracy and the reduction rate at decent levels.

From the experimental study it is clear that with the parallelism we can achieve faster execution up to 650 times in certain datasets, compared to the execution of RHC and ERHC on the original training set. On the other hand, in pmRHC there is significant loss in reduction rate (up to 50%) and a relative small loss in accuracy in most of the datasets that are tested. For the pmERHC algorithm, reduction rate values are deteriorating, but less than in the case of pmRHC, which is justified by the editing step that is introduced in the former. Last but not least, when the number of subsets is high, the accuracy is almost the same compared to the one obtained when applying ERHC on the original training set, while in BL, SH, TXR, BN and YS datasets the accuracy improves.

Table 2: Comparison of RHC, ERHC, and pmRHC/pmERHC 8 through 64 in terms of Reduction Rate (RR(%)), Accuracy (ACC(%)) and Preprocessing Cost (PC (millions of distance computations))

Data		RHC	pmRHC 8 subsets	pmRHC 16 subsets	pmRHC 32 subsets	pmRHC 64 subsets	ERHC	pmERHC 8 subsets	pmERHC 16 subsets	pmERHC 32 subsets	pmERHC 64 subsets
BL	Acc	68.64	63.28	67.73	71.30	**76.53**	67.62	81.42	85.65	89.97	**94.08**
	RR	**78.00**	71.12	67.44	65.64	59.92	**86.68**	83.44	82.36	80.80	73.80
	PC	0.05	0.002	0.001	0.00016	**0.00005**	0.273	0.002	0.001	0.00016	**0.00005**
KDD	Acc	**99.39**	84.49	87.66	90.17	92.08	**99.45**	95.51	96.79	97.86	98.10
	RR	**99.19**	98.32	97.80	97.15	96.34	**99.46**	98.83	98.48	98.09	97.58
	PC	81.59	6.741	2.287	0.880	**0.315**	81.59	6.741	2.287	0.880	**0.315**
LS	Acc	**88.95**	66.41	68.73	72.11	75.03	**89.01**	75.43	78.28	81.70	85.38
	RR	**89.84**	84.41	82.04	78.87	74.06	**92.95**	90.52	89.10	87.22	84.38
	PC	1.693	0.0848	0.0286	0.0097	**0.0031**	1.693	0.0848	0.0286	0.0097	**0.0031**
LIR	Acc	**93.59**	82.86	83.90	85.27	86.02	**92.69**	89.34	89.54	89.94	90.00
	RR	**88.08**	71.87	64.77	56.31	46.83	**92.03**	83.38	80.10	77.86	77.74
	PC	41.844	1.7478	0.6021	0.1919	**0.0629**	41.844	1.7478	0.6021	0.1919	**0.0629**
MGT	Acc	**71.97**	62.13	62.49	63.85	64.89	**77.01**	71.61	71.65	72.58	73.74
	RR	**73.76**	70.79	69.77	68.32	66.40	**84.46**	82.71	81.88	81.26	80.25
	PC	4.082	0.2668	0.1073	0.0417	**0.0159**	4.082	0.2668	0.1073	0.0417	**0.0159**
MN2	Acc	**94.68**	92.39	89.75	88.10	83.76	**95.14**	94.46	91.56	89.59	84.11
	RR	**96.47**	81.19	73.61	66.32	57.41	**96.76**	85.07	80.21	76.85	67.83
	PC	0.007	0.00079	0.00027	0.00008	**0.00002**	0.007	0.00079	0.00027	0.00008	**0.00002**
PD	Acc	**98.30**	92.61	93.79	94.92	96.25	**98.63**	97.57	97.82	98.01	98.60
	RR	**96.52**	91.84	88.77	84.89	79.19	**97.45**	94.22	92.07	89.63	86.52
	PC	2.882	0.1708	0.0591	0.0203	**0.0075**	2.882	0.1708	0.0591	0.0203	**0.0075**
PH	Acc	**85.59**	69.51	70.90	71.49	71.97	**86.57**	78.55	78.60	77.99	76.59
	RR	**80.71**	74.10	71.82	69.75	68.06	**88.05**	85.14	84.42	83.65	82.76
	PC	0.658	0.1708	0.0149	0.0055	**0.0019**	0.658	0.1708	0.0149	0.0055	**0.0019**
SH	Acc	**98.10**	88.44	91.15	93.95	95.43	98.04	96.15	96.84	98.25	**98.82**
	RR	**99.55**	98.64	98.00	97.07	95.70	**99.69**	99.02	98.50	97.78	96.70
	PC	16.827	1.5323	0.6121	0.2473	**0.0880**	16.827	1.5323	0.6121	0.2473	**0.0880**
TXR	Acc	**97.04**	90.58	92.56	93.75	94.88	97.36	95.09	96.93	97.66	**98.54**
	RR	**94.71**	86.81	82.19	75.96	66.55	**95.94**	90.75	87.82	84.31	79.53
	PC	3.629	0.0961	0.0321	0.0111	**0.0038**	3.629	0.0961	0.0321	0.0111	**0.0038**
TN	Acc	**88.69**	73.58	75.90	75.89	77.74	**91.53**	82.93	84.40	85.45	87.43
	RR	**96.63**	96.48	96.47	96.15	95.56	**97.58**	97.44	97.31	97.16	96.69
	PC	1.642	0.0506	0.0148	0.0034	**0.0008**	1.642	0.0506	0.0148	0.0034	**0.0008**
ECL	Acc	68.76	61.16	64.56	71.37	**72.44**	73.48	78.20	78.70	77.44	**84.21**
	RR	**67.58**	59.30	60.49	67.34	65.72	84.83	82.66	79.84	74.93	**87.05**
	PC	0.03	0.00081	0.00024	0.00007	**0.00006**	0.175	0.00081	0.00024	0.00007	**0.00006**
PM	Acc	**63.28**	59.94	59.27	60.89	60.58	69.79	64.35	67.56	**70.97**	66.54
	RR	**63.58**	57.94	57.33	51.40	62.73	80.07	79.00	77.77	77.15	**80.14**
	PC	0.062	0.003	0.001	0.00034	**0.00008**	0.062	0.003	0.001	0.00034	**0.00008**
YS	Acc	**48.85**	41.68	42.60	44.03	46.24	50.33	54.48	54.89	55.47	**59.25**
	RR	49.83	45.08	43.35	46.75	**53.42**	**79.34**	78.12	78.52	77.98	77.88
	PC	0.84	0.0171	0.0054	0.0015	**0.0004**	4.215	0.0171	0.0054	0.0015	**0.0004**
BN	Acc	**83.28**	68.15	71.27	75.45	79.98	88.00	82.15	85.90	88.69	**91.18**
	RR	**79.68**	75.93	74.06	70.12	65.12	**90.33**	87.53	85.59	83.17	79.93
	PC	0.562	0.0375	0.0146	0.0053	**0.0019**	0.562	0.0375	0.0146	0.0053	**0.0019**

We plan to further investigate the effect of parallelism on more state of the art data reduction algorithms, and also examine whether the phase of evaluation using the testing sets could be applied on the reduced subsets in a majority vote fashion instead of merging the reduced sets back to a single reduced set.

7 ACKNOWLEDGMENTS

Research funded by the University of Macedonia Research Committee as part of the "Principal Research 2019" funding program.

REFERENCES

[1] Jesús Alcalá-Fdez, Alberto Fernández, Julián Luengo, Joaquín Derrac, and Salvador García. 2011. KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. *Multiple-Valued Logic and Soft Computing* 17, 2-3 (2011), 255–287. http://www.oldcitypublishing.com/journals/mvlsc-home/mvlsc-issue-contents/mvlsc-volume-17-number-2-3-2011/mvlsc-17-2-3-p-255-287/

[2] Salvador García, Joaquín Derrac, José Ramón Cano, and Francisco Herrera. 2012. Prototype Selection for Nearest Neighbor Classification: Taxonomy and Empirical Study. *IEEE Trans. Pattern Anal. Mach. Intell.* 34, 3 (2012), 417–435. https://doi.org/10.1109/TPAMI.2011.142

[3] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: an update. *SIGKDD Explorations* 11, 1 (2009), 10–18.

[4] Ishwarappa and J. Anuradha. 2015. A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology. *Procedia Computer Science* 48 (2015), 319–324. https://doi.org/10.1016/j.procs.2015.04.188

[5] Beniamino Di Martino, Rocco Aversa, Giuseppina Cretella, Antonio Esposito, and Joanna Kolodziej. 2014. Big data (lost) in the cloud. *IJBDI* 1, 1/2 (2014), 3–17. https://doi.org/10.1504/IJBDI.2014.063840

[6] Stefanos Ougiaroglou, Georgios Arampatzis, Dimitris A. Dervos, and Georgios Evangelidis. 2017. Generating Fixed-Size Training Sets for Large and Streaming Datasets. In *Advances in Databases and Information Systems - 21st European Conference, ADBIS 2017, Nicosia, Cyprus, September 24-27, 2017, Proceedings*. 88–102. https://doi.org/10.1007/978-3-319-66917-5_7

[7] Stefanos Ougiaroglou and Georgios Evangelidis. 2016. Efficient editing and data abstraction by finding homogeneous clusters. *Ann. Math. Artif. Intell.* 76, 3-4 (2016), 327–349. https://doi.org/10.1007/s10472-015-9472-8

[8] Stefanos Ougiaroglou and Georgios Evangelidis. 2016. RHC: a non-parametric cluster-based data reduction for efficient k-NN classification. *Pattern Anal. Appl.* 19, 1 (2016), 93–109. https://doi.org/10.1007/s10044-014-0393-7

[9] Muhammad Habib Ur Rehman, Chee Sun Liew, Assad Abbas, Prem Prakash Jayaraman, Teh Ying Wah, and Samee U. Khan. 2016. Big Data Reduction Methods: A Survey. *Data Science and Engineering* 1, 4 (2016), 265–284. https://doi.org/10.1007/s41019-016-0022-0

[10] Isaac Triguero, Joaquín Derrac, Salvador García, and Francisco Herrera. 2012. A Taxonomy and Experimental Study on Prototype Generation for Nearest Neighbor Classification. *IEEE Trans. Systems, Man, and Cybernetics, Part C* 42, 1 (2012), 86–100. https://doi.org/10.1109/TSMCC.2010.2103939

[11] Dennis L. Wilson. 1972. Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. *IEEE trans. on systems, man, and cybernetics* 2, 3 (July 1972), 408–421.

[12] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010.*