Artificial Intelligence Research and Development
E. Armengol et al. (Eds.)
IOS Press, 2015

doi:10.3233/978-1-61499-578-4-100

A Windowing based GPU optimized strategy for the induction of Decision Trees in JaCa-DDM

Xavier Limón [a], Alejandro Guerra-Hernández [a], Nicandro Cruz-Ramírez [a], Héctor-Gabriel Acosta-Mesa [a], and Francisco Grimaldo [b]

[a] *Universidad Veracruzana, Centro de Investigación en Inteligencia Artificial, Sebastián Camacho No 5, Xalapa, Ver., México 91000*

[b] *Universitat de València, Departament d'Informàtica, Avigunda de la Universitat, s/n, Burjassot-València, España 46100*

Abstract. When inducing Decision Trees, Windowing consists in selecting a random subset of the available training instances (the window) to induce a tree, and then enhance it by adding counter examples, i.e., instances not covered by the tree, to the window for inducing a new tree. The process iterates until all instances are well classified or no accuracy is gained. In favorable domains, the technique is known to speed up the induction process, and to enhance the accuracy of the induced tree; while reducing the number of training instances used. In this paper, a Windowing based strategy exploiting an optimized search of counter examples through the use of GPUs is introduced to cope with Distributed Data Mining (DDM) scenarios. The strategy is defined and implemented in JaCa-DDM, a novel system founded on the Agents & Artifacts paradigm. Our approach is well suited for DDM problems generating large amounts of training instances. Some experiments in diverse domains compare our strategy with the traditional centralized approach, including an exploratory case study on pixel-based segmentation for the detection of precancerous cervical lesions on colposcopic images.

Keywords. Windowing, Decision Trees, GPU computation, Multi-Agent Systems, Distributed Data Mining

1. Introduction

The Windowing technique was originally designed to cope with memory limitations in the C4.5 [8] system. It consists in inducing a tree from a small random subset of the available training instances (the window). The tree is then used to classify the remaining training instances, searching for counter examples, i.e., instances not covered by the current tree. The window is extended with the counter examples found and a new tree is induced. The process iterates until a stop criterion is met, e.g., all examples are covered; or the accuracy of the new tree does not enhance anymore.

Windowing is expected to obtain an accuracy similar to that obtained using all the available training instances, while reducing considerably the number of examples used to induce a tree. In favorable domains, i.e., free of noise and indeterminism, it is also expected to speed up the inductive process; but in the general case, it slows down the process since convergence requires many iterations. Windowing based strategies [4] for Distributed Data Mining (DDM) seem to inherit these properties: The accuracy of the induced trees is close to, or even slightly better than that obtained without windowing; The number of examples used to induce the tree is reduced up to 60%; But the processing time is much worse when using Windowing, 90 times slower in the worst case.

Searching for counter examples seems to be in part responsible for the poor time performance of the Windowing based strategies. In this work, CUDA [7] enabled GPUs are used to improve the gathering of counter examples, seeking a performance improvement of the overall induction processes. Although some frameworks have been proposed to boost time efficiency of the data mining process through GPUs [12,6], including the induction of Decision Trees [11,5], our work focuses on DDM scenarios, using JaCa-DDM [4] to further enhance the performance of the processes and to overcome GPU memory limitations.

JaCa-DDM is an Agents & Artifacts [9] based DDM system, conceived to design, implement, deploy and evaluate distributed learning strategies. A strategy is a description of the interactions among a set of agents, exploiting artifacts deployed in a distributed system, that provide data mining tools. The proposed strategy concerns the induction of Decision Trees [8], using the J48 algorithm provided by Weka [14].

The organization of the paper is as follows: Section 2 presents a brief description of JaCa-DDM, introducing the notions of strategy and deployment system. Section 3 describes the implementation of the Windowing based strategy proposed in this paper, detailing the GPU based optimizations. Section 4 defines the experimental methodology to evaluate the proposed strategy. The results and discussion of the experiments are presented in section 5. Finally, this paper closes with some conclusions and insights of future work in section 6.

2. JaCa-DDM

JaCa-DDM is a system based on the Agents & Artifacts paradigm as implemented by Jason [3], the well known agent oriented programming language, and CArtAgO [9], an agent infrastructure to define environments based on the concept of artifacts. The main interest of JaCa-DDM is to provide a platform to execute and test data mining processes over a distributed environment. A novelty of JaCa-DDM is the way in which the DDM processes are conceived as strategies.

Strategies are descriptions of workflows in terms of agents and artifact interactions, allowing the implementation of truly sophisticated processes that can exploit BDI reasoning and representations, as well as speech acts based communications; while using already existing data mining tools provided by Weka, wrapped in the form of artifacts. Strategies are by definition encapsulated, allowing standarized ways to define, configure, deploy, and test them.

The JaCa-DDM model is built on the concepts of strategy and its deployment. While a strategy defines a DDM workflow, its deployment deals with configuration issues.

Definition 2.1 *A tuple* $\langle Ags, Arts, Params, ag_1 \rangle$ *denotes a JaCa-DDM strategy, where:*

- $Ags = \{ag_1, \ldots, ag_n\}$ *is the set of user defined Jason agent programs.*
- $Arts = \{art_1, \ldots, art_m\}$ *is the set of user defined artifact types.*
- $Params = \{param_1 : type_1, \ldots, param_k : type_k, \}$ *is a set of paramenters and their associated data types, where* $type_{1,\ldots,k} \in \{int, bool, double, string\}$.
- $ag_1 \in Ags$ *is a special agent program playing the role of contact person between the agents in* Ags *and the deployment system. This agent launches and finishes the strategy, and can be programmed to do any other task.*

Definition 2.2 *A tuple* $\langle Nodes, DS, Arts, Strat, Config, ag_0 \rangle$ *is a JaCa-DDM deployment system, where:*

- $Nodes = \{node_0, node_1 \ldots, node_j\}$, *is a set of computational nodes, usually, but not necessarily, distributed in a network, where:* $node_0$ *is running Jason and CArtAgO, while* $node_{1,\ldots,j}$ *are running only CArtAgO. Each node defines a single CArtAgO workspace, where artifacts are to be created, but all agents run in* $node_0$. *Each node is denoted by a pair* $\langle nodeName, IPaddress : port \rangle$.
- $DS = \{ds_1, \ldots, ds_j\}$ *is a set of data sources associated to each node, not including* $node_0$. *Data sources can be created dinamically at run time; or be statically defined in each node.*
- $Arts = \{art_1, \ldots, art_i\}$ *is a set of artifact types, used to deploy the system.*
- *Strat is a learning strategy as stated in Definition 2.1.*
- $Config = \langle \delta, \pi \rangle$ *is a configuration for a strategy deployment. It has two components:*
  - $\delta = \{(ag, node, i), \ldots\}$, *is a set of allocations, i.e., an agent distribution specifying how many copies of a given agent program will be focusing on a given node. Where* $ag \in Strat_{Ags}$ *is an agent program in the strategy, that will be cloned* $i \geq 1$ *times, and assigned to focus on* $node \in Nodes \backslash \{node_0\}$.
  - $\pi = \{(p, v), \ldots\}$ *is a set of pairs: strategy parameter, initialization value; where for all* $param : type \in Strat_{Params}$, p *is a parameter of the strategy and* v *is its value of type* t.
- ag_0 *is the agent setting up the deployment system.*

3. Implementation

The implementation of the proposed Windowing based GPU optimized strategy comprehends a GPU based counter examples filtering processes, and the Parallel Counter GPU strategy itself.

3.1. GPU based counter examples filtering processes

The windowing process can be split into two main subprocesses repeated iteratively: counter examples filtering and model induction. In this work we implement the filtering of counter examples on GPUs, trying to achieve a negligible time cost for this subprocess. As mentioned, we do not deal with the induction subprocess, which allows for other methods to be applied, for example ensemble techniques, or GPU based induction algorithms.

A decision tree has two kinds of nodes: internal nodes, and leaf nodes. Internal nodes represent a given attribute, and leaf nodes class values. Arcs contain a boolean function over the node attribute values, and each function over the same node is mutually exclusive. There are three kinds of arc functions, each one bound to a boolean operator: $\leq$, $>$, $=$. The first two operators are for numerical attributes, and the last one for nominal ones. Given a Decision Tree, and an unclassified instance, a classification process consist off traversing arcs yielding true values on its function, from the root node to a leaf node.

On GPUs, it is a good practice to avoid irregular and complex data structures, in order to improve performance. Scattered memory access is not efficient, and affects the performance of the GPU cache memories. It is better to read large blocks of memory in order to exploit coalesced memory access (combining multiple memory accesses into a single transaction). With these ideas in mind, a plain representation based on one dimensional arrays was adopted for the GPU Decision Trees. The structure consists on various properties, some of them are related to node and arc information:

- int NUM_NODES : how many nodes (including leaves) has the tree.
- int MAX_NUM_ARCS: each node can have a variable number of arcs, but a constant value is necessary to reserve memory.
- int attributes[NUM_NODES]: contains the attribute index for each node. On the case of a leaf node, it contains the index of the class value.
- int isLeaf[NUM_NODES]: a leaf node contains the value 1, otherwise 0.
- int numbersOfArcs [NUM_NODES]: the actual number of arcs of each node.
- int NUM_ARCS: the sum of all the arcs of all nodes.
- int evalTypes[NUM_ARCS]: the evaluation type of the arc: $\leq$, $>$, $=$.
- float vals[NUM_ARCS]: evaluation value of the arc.
- int nodeIndices[NUM_ARCS]: the index of the node pointed by the arc.

A method that takes a Weka J48 Tree and transforms it to a GPU Decision Tree was implemented in the J48 artifact. A kernel (in CUDA terms) is a function that executes on a device (GPU). Kernels are intended to be executed in parallel, receiving parameters to define the number of threads to be used. The implemented kernels include:

- classify : Return the index value of the predicted class of an instance.
- searchCounter: Classifies each instance within the instance set in GPU, and if the predicted class is different from the actual class, then it saves the index of the instance in an array as big as the instance set. Each thread receives the number of instances that will process. At the end of its work, each thread also saves the number of counter examples found.

- countSize: Computes a sum over each thread result of the searchCounter kernel to yield the total number of counter examples found.
- genResult: "Shrinks" the array that contains the counter examples indices found by the searchCounter kernel, saving the indices on a new array that is the exact size of the counter examples found.
- filterParallel: Filters the counter examples from the dataset in the GPU.

The workflow of the counter examples filtering process requires to load the instance set into the GPU at the beginning of the general process. A copy of the instance set is held in the CPU. It is also necessary to determine the number of multi-processors, and the maximum number of parallel threads of each multi-processor, in order to define an ideal number of working threads. The filtering process, from the host's (CPU) point of view, can be summarized in the following steps:

1. Transform the J48 Tree into a GPU Decision Tree.
2. Load the GPU Decision Tree in the GPU.
3. Invoke the searchCounter kernel on the ideal number of threads.
4. Invoke the countSize kernel on one thread.
5. Use the result from countSize to reserve enough memory on the GPU to save all the counter example indices on an array.
6. Invoke genResult on one thread to fill the array created previously.
7. Invoke filterParallel on the ideal number of threads to erase the counter examples found from the instance set in the GPU.
8. Use the array with counter examples indices, and filter all the counter examples in the CPU to obtain a counter examples instance set.
9. Free the memory not needed anymore on the GPU.

Note that the search process on the GPU only finds index values, the actual filtering is done on the CPU. This design choice was made to reduce data transmission between the host and the devices.

3.2. Parallel Counter GPU strategy

The proposed strategy consists on an agent bulding an initial Decision Tree with a subset of its training instances, in a central classifier artifact. Asking to other agents in the system to gather all the counter examples found in the deployment system, and sending them to the central classifier artifact for trying to enhance the Decision Tree. The process iterates for a number of rounds. Following definition 2.1, its components are:

- $Ags = \{contactPerson, worker, roundController\}$, where:
  - *contactPerson* controls the rounds and induces the learned model.
  - *worker* gathers counter examples.
  - *roundController* determines the stop condition.
- $Arts = \{ClassifierJ48, InstancesBase, Evaluator\}$, where:
  - $ClassifierJ48$. Induces models.

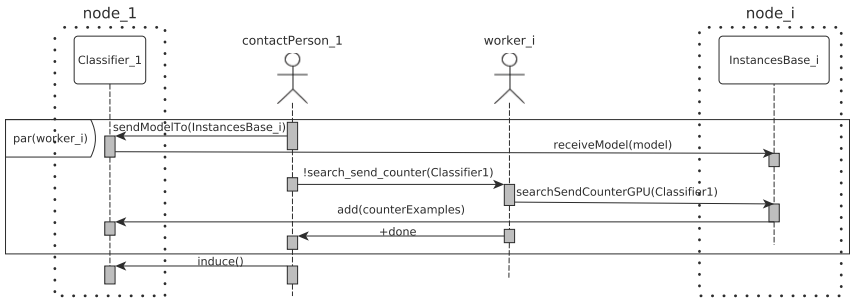

Figure 1. Parallel Counter GPU strategy sequence diagram for counter examples filtering workflow. worker_i represents any worker, i.e $i = 1, ..., j$ (the same goes for node_i). $contacPerson_1$ sends the current model to the $InstancesBase_i$ artifact of each $worker_i$. Then it asks all the workers to search for counter examples in their $InstancesBase_i$ using the GPU, and send them to $Classifier_1$, where a new model is induced.

  - $*$ *InstancesBase*. Used to store and manipulate the learning examples. The GPU search is launched on this artifact.
  - $*$ *Evaluator*. Used to compute the accuracy of a model given a testing set, for the auto-adjust stop procedure.

- *Params* include:

  - $*$ $Prunning : Bool$ if true, forces the J48 to use post pruning.
  - $*$ $InitPercentage : Double$ defines the size of the initial training window.
  - $*$ $TestPercentageForRounds : Double$ defines the size of the testing set for the auto-adjust stop procedure.
  - $*$ $ChangeStep : Double$ defines a threshold of minimum change between two consecutive rounds. Used by the auto-adjusted stop procedure.
  - $*$ $MaxRounds : Int$ defines the maximum number of rounds.

Figure 1 shows the workflow for one round of the Parallel Counter GPU strategy. The stop criterion computing is not show for the sake of clarity, but at the end of every round, the induced Decision Tree is tested to obtain its accuracy and decide if the process continues or not.

4. Experimental Methodology

A set of datasets were selected to compare the perfomance of the proposed Parallel Counter GPU strategy with the usual centralized approach. The measured parameters for all the experiments are the following: accuracy, percentage of training examples used, time in seconds, number of leaves, and tree size. The experiments were executed on a cluster consisting of three computers with the same characteristics: Two Xeon processors at 2.40 GHz with four cores, and two threads each; 24 GB of RAM; Two GPU Nvidia Tesla C2050.

Table 1 shows some datasets used for this purpose. They were selected from the MOA [2] and TunedIT [15] projects, because they vary in the number of

instances, attributes, and classes. Evaluation was done with a 10-fold stratified cross validation, and also the training instances were stratified and split evenly among the 3 computer nodes.

Table 1. Used MOA/TunedIT datasets.

DS	#Instances	#Attributes	#Classes
airlines	539383	8	2
covtypeNorm	581012	55	7
KDDCup99	4898431	42	23
poker-lsn	829201	11	10

Strategy distribution δ was $\{(contactPerson, node_1, 1), (roundController, node_1, 1), (worker, node_1, 1), (worker, node_2, 1), (worker, node_3, 1)\}$. The parameters initialization π for all datasets was $\{(Pruning, true), (InitPercentage, 0.15), TestPercentageForRounds, 0.15), (ChangeStep, 0.004), (MaxRounds, 10)\}$. The InitPercentage and TestPercentageForRounds parameters only take data from one node, and each node has one third of the training data (JaCa-DDM splits and shares the dataset in a stratified fashion before beginning the process), thus the parameter value considers this fact. On the centralized case, pruning was also active.

An exploratory case study for pixel-based segmentation was also considered. Pixel-based segmentation consists on extracting features from labeled pixels to create a training dataset, which is used to construct a model to predict the class of unseen pixels, and in this way achieve image segmentation [13]. Our case study deals with sequences of colposcopic images presenting possible precancerous cervical lesions. The image data was extracted from 38 patients, for each patient a range between 300 and 600 images were obtained. A medical expert labeled some of the pixels of each image (figure 2 shows an example), from which two classes can be drawn: precancerous lesion, and no precancerous lesion. For a more detailed account of this case study see [1]. From the images for each patient, we selected 30 images evenly spread in the series. Using FIJI [10] we extracted the pixels of interest from the images to create a Weka ARFF file, selecting the default pixel parameters: gaussian blur, sobel filter, hessian, membrane projections, and difference of gaussians. The obtained dataset has the following characteristics: Total number of examples: 1016600; No precancerous lesions: 213819; Precancerous lesions: 802781; Number of attributes: 80. We compare our strategy with the centralized approach using leave-one-out, where the test data in each case is the extracted pixels from a single patient. For our strategy, the training data was stratified and evenly split in our 3 computers available. The distribution δ and parameters π of the Parallel Counter GPU was the same as in the general experiments. It is worth mentioning that we did not apply any preprossessing to the dataset, being an exploratory experiment, we are more interested in testing the behavior of our approach in this kind of setting.

5. Results and discussion

Table 2 shows the results obtained by our strategy and the centralized approach for the MOA/TunedIT datasets. As expected, accuracy is similar in all cases, and

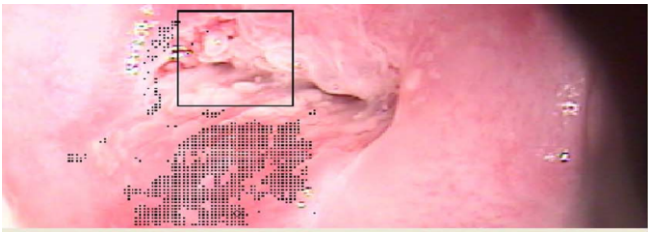

Figure 2. Example of colposcopic image. Black dots represent pixels labeled by a medical expert.

our strategy reduced the number of examples used for training up to a 90%. The number of leafs and tree size are also reduced by our strategy in all cases. GPU based counter examples filtering speeds up the whole process. For KDDCup99 and poker-lsn, our approach is up to 8 times faster than the centralized one. For airlines and covtypeNorm, in the wort case, our approach is 0.78 times slower than the centralized. This enhances the results of the Windowing based strategies previously reported by Limón [4], where such strategies were up to 200 times slower than the centralized approach.

The rate of instances used to induce a Decision Tree is indicative of how difficult is for the strategy to converge. Higher rates suggest that the strategy iterates more times, inducing more trees, and using more instances before attaining convergence. Interestingly, in these and previous experiments, the rate of used instances seems to correlate with the accuracy: Low accuracy demands more instances, while high accuracies demands much fewer instances. In these cases, the GPU based counter example fintering is not responsible of the decreased time performance, but the number of iterations executed by the strategy. The airlines dataset shows an extreme case in this regard, the overall problem is too difficult for the J48 algorithm.

In the observation of the evolution of the learning process of covtypeNorm we found that the majority of examples used for learning were found during the fist search of counter examples. This means that the initial model was too simple, and adding all the counter examples found could not be the best choice in the long run. Possibly, increasing the size of the initial training set, and/or further filtering counter examples on initial phases of the process, would help to obtain a better time performance.

Table 2. Results for the MOA/TunedIT datasets.

DS	Strategy	Accuracy	Used instances	Time (seconds)	#Leaves	Tree Size
airlines	Centralized	66.34 ± 0.11	100.00 ± 0.00	1164.66 ± 211.76	137470	142081
airlines	Parallel Counter	66.26 ± 0.12	94.95 ± 0.01	1810.78 ± 446.47	132767	137210
covtypeNorm	Centralized	94.59 ± 0.04	100.00 ± 0.00	855.41 ± 97.88	14158	28314
covtypeNorm	Parallel Counter	93.10 ± 0.34	48.44 ± 0.01	1089.03 ± 277.06	12679	25265
KDDCup99	Centralized	99.99 ± 0.01	100.00 ± 0.00	1688.91 ± 363.89	968	1147
KDDCup99	Parallel Counter	99.96 ± 0.01	9.28 ± 0.01	199.72 ± 45.62	667	855
poker-lsn	Centralized	99.78 ± 0.01	100.00 ± 0.00	174.26 ± 28.55	2212	4408
poker-lsn	Parallel Counter	98.67 ± 0.46	9.56 ± 0.01	24.90 ± 8.05	1831	3552

For the pixel-based segmentation case study, Table 3 shows a comparison of the results obtained by our strategy, the centralized approach, and the results

reported in [1]. The sensibility (Sen) is the rate of true positive observations (precancerous lesion) against the sum of true positive plus false negatives, and the specificity (Spe) is the rate of true negative (no precancerous lesion) observations against the sum of true negative plus false positives.

Observe that the results are similar to those obtained for the covtypenorm dataset, with similar explanations. Anyway, observe that the proposed approach obtained a similar accuracy and sensibility than the results reported by Acosta-Mesa et al. [1], where a time series approach was adopted, and other normalizations were applied, that yielded a more balanced dataset, which in turns explains the difference in specificity that we obtained. Our experiment was done with a unbalanced dataset with no preprocessing applied (as this is an exploratory case study), favoring the precancerous lesion class, which also may explain the sensibility obtained. Being identified the probable cause of no time performance improvement in our strategy (i.e too many counter examples are added on the first round), and given the results obtained, we are optimistic that with a proper prepossessing of the dataset, and with an enhancement of our current strategy, we can achieve much better results.

Table 3. Results for the case study

Strategy	Accuracy	Used instances	Time (seconds)	#Leaves	Tree Size	Sen	Spe
Centralized	66.32 ± 29.50	100.00 ± 0.00	4806.50 ± 455.28	32436	64871	79.04	18.53
Parallel Counter	65.87 ± 26.89	46.80 ± 0.02	6317.36 ± 605.46	30633	61265	77.73	21.37
Results from [1]	67.00	n/a	n/a	n/a	n/a	71.00	59.00

6. Conclusions and future work

The Windowing based GPU optimized strategy proposed in this work demonstrates that Windowing based approaches can be applied to large datasets, having clear time performance improvements in some cases, in comparison with the centralized approach, while preserving a similar accuracy. Even on no favorable cases, our strategy was acceptably slower, and always reduced the number of leaves and tree size by using less counter examples. We believe that the time performance that our strategy can achieve is bound by two factors: i) The complexity of the problem represented by the dataset; and ii) The dataset redundancy. The first factor is an open problem, and for the time being, we do not plan to direct our efforts toward it, while the second factor is of interest for our future work. In practical terms, redundancy means that some examples from the dataset are not needed for the learning process, and that discarding such examples is paramount to improve time performance. We plan to so research on the nature of redundancy on tree model induction to create an improved version of the Windowing algorithm that further filters redundant examples.

References

[1] Héctor-Gabriel Acosta-Mesa, Nicandro Cruz-Ramírez, and Rodolfo Hernández-Jiménez. Aceto-white temporal pattern classification using k-nn to identify precancerous cervical lesion in colposcopic images. *Computers in biology and medicine*, 39(9):778–784, 2009.

[2] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *The Journal of Machine Learning Research*, 11:1601–1604, 2010.

[3] Rafael H. Bordini, Jomi F. Hübner, and Mike Wooldridge. *Programming Multi-Agent Systems in Agent-Speak using Jason*. John Wiley & Sons Ltd, 2007.

[4] Xavier Limón, Alejandro Guerra-Hernández, Nicandro Cruz-Ramírez, and Francisco Grimaldo. An agents & artifacts approach to distributed data mining. In F. Castro, Alexander Gelbukh, and M. G Mendoza, editors, *11th MICAI*, volume 8266 of *LNAI*, pages 338–349, Berlin Heidelberg, 2013. Springer Verlag.

[5] Win-Tsung Lo, Yue-Shan Chang, Ruey-Kai Sheu, Chun-Chieh Chiu, and Shyan-Ming Yuan. Cudt: a cuda based decision tree algorithm. *The Scientific World Journal*, 2014, 2014.

[6] Wenjing Ma and Gagan Agrawal. A translation system for enabling data mining applications on gpus. In *Proceedings of the 23rd international conference on Supercomputing*, pages 400–409. ACM, 2009.

[7] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.

[8] John Ross Quinlan. *C4. 5: programs for machine learning*, volume 1. Morgan kaufmann, 1993.

[9] Alessandro Ricci, Michele Piunti, and Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, 2011.

[10] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, et al. Fiji: an open-source platform for biological-image analysis. *Nature methods*, 9(7):676–682, 2012.

[11] Toby Sharp. Implementing decision trees and forests on a gpu. In *Computer Vision–ECCV 2008*, pages 595–608. Springer, 2008.

[12] Nam-Luc Tran, Quentin Dugauthier, and Sabri Skhiri. A distributed data mining framework accelerated with graphics processing units. In *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*, pages 366–372. IEEE, 2013.

[13] Xiang-Yang Wang, Xian-Jin Zhang, Hong-Ying Yang, and Juan Bu. A pixel-based color image segmentation using support vector machine and fuzzy c-means. *Neural Networks*, 33:148–159, 2012.

[14] Ian H. Witten and Eibe Frank. *Data mining: Practical machine learning tools and techniques.* Morgan Kaufmann, San Francisco, CA., USA, second edition, 2005.

[15] Marcin Wojnarski, Sebastian Stawicki, and Piotr Wojnarowski. TunedIT.org: System for automated evaluation of algorithms in repeatable experiments. In *Rough Sets and Current Trends in Computing (RSCTC)*, volume 6086 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 20–29. Springer, 2010.