Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 207 (2022) 887–896

Procedia Computer Science

www.elsevier.com/locate/procedia

26th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2022)

Reduced precision discretization based on information theory

Brais Ares[a], Laura Morán-Fernández[b], Verónica Bolón-Canedo[b,*]

[a]*Gradiant, Estrada do Vilar 56 (Vigo), Pontevedra, Spain*
[b]*CITIC, Universidade da Coruña, A Coruña, Spain*

Abstract

In recent years, new technological areas have emerged and proliferated, such as the Internet of Things or embedded systems in drones, which are usually characterized by making use of devices with strict requirements of weight, size, cost and power consumption. As a consequence, there has been a growing interest in the implementation of machine learning algorithms with reduced precision that can be embedded in these constrained devices. These algorithms cover not only learning, but they can also be applied to other stages such as feature selection or data discretization. In this work we study the behavior of the Minimum Description Length Principle (MDLP) discretizer, proposed by Fayyad and Irani, when reduced precision is used, and how much it affects to a typical machine learning pipeline. Experimental results show that the use of fixed-point format is sufficient to achieve performances similar to those obtained when using double-precision format, which opens the door to the use of reduced-precision discretizers in embedded systems, minimizing energy consumption and carbon emissions.

Keywords: Reduced precision; discretization; preprocessing; mutual information; machine learning

1. Introduction

Discretization of numerical data is one of the most influential data preprocessing tasks in knowledge discovery and data mining. The discretization process aims to find a concise representation of the input data into categories that are better suited for the learning task, while retaining as much information as possible from the original continuous attributes. One of the most famous discretization methods is the discretizer based on the *Minimum Description Length Principle* (MDLP) proposed by Fayyad and Irani [3]. This method uses information from the class to control the partitioning process, taking into account the entropy gain of each possible partition.

Usually, the discretization process is run on machines that fully support double-precision floating-point format (64 bits), trying to make use of all available resources (powerful processors, large amount of memory, hardware support, etc.). However, this kind of resources may not be present when working with embedded systems, low-power devices

* Corresponding author. Tel.: +34-981167000.
E-mail address: bares@gradiant.org; laura.moranf@udc.es; veronica.bolon@udc.es

1877-0509
10.1016/j.procs.2022.09.144

or ad-hoc solutions that need to minimize and optimize the use of hardware components. The objective of this work is to redesign the MDLP algorithm so that it uses reduced precision (i.e. limiting the number of bits) to run the discretization process. The reduced precision affects both the way the numerical data is represented as well as the math operations themselves when computing the algorithm.

Taking an algorithm that was originally intended to work with floating point arithmetic and adapting it to fixed-point format, in which the number of bits is reduced, is not something new. In many scenarios, the potential loss of precision that this adaptation may entail is negligible, yet a series of substantial advantages are obtained in return, such as: reduction of computational requirements, lower power consumption, faster execution, etc. In much of the existing literature one can find several studies on the effect of using reduced precision for learning algorithms, most of them based on neural networks [11]. Han et al. [6] propose an energy-efficient engine that performs inference in deep neural networks and accelerates the resulting sparse matrix-vector multiplication with weight sharing. Hubara et al. [7] introduce a method for training Quantized Neural Networks (QNN) in which weights and activations have very low precision. Jacob et al. [8] propose a quantization scheme for the approximation of floating-point computations in a neural network. The authors built on previous research by Gupta et al. [4], which proposed the use of fixed-point arithmetic with reduced precision to speed up the training process of convolutional neural networks. In the area of Bayesian networks, the work of Tschiatschek and Pernkopf [15] proposes to perform online learning with reduced-precision parameters. All these authors have managed to obtain very similar results to Bayesian network classifiers that use traditional algorithms for parameter learning with double-precision floating-point representation.

Although the use of reduced precision in other stages of machine learning is not as popular as in classification, several experiments can be found that show the applicability of these techniques in previous stages. For example, Sharma et al. [14] propose a fixed-point version of the Principal Component Analysis (PCA) algorithm to perform feature extraction of a set of DNA gene microarrays, in the field of human cancer, obtaining similar results to the original version. Morán-Fernández et al. [10] also proposes the simplification of certain feature selection methods by means of a fixed-point version for calculating the mutual information. Experimental results show that 16 bits are sufficient for the algorithm to retrieve the same features as a 64-bit representation does. However, no work has been found in the state-of-the-art that attempts to perform a discretization process using an algorithm with fixed-point arithmetic. It is known that applying a discretization process on the input data can greatly relax the computational requirements of the following learning stages, and in many cases even improve the results obtained in classification accuracy [9, 5]. Moreover, there are certain algorithms which simply do not support continuous variables and require prior discretization of the input data. For all these reasons, the goal of this work is to adapt a widely-known discretization algorithm to fixed-point format and then compare the obtained results with the original double-precision (64-bit) floating-point version. In order to test how this discretization process affects other steps of a typical machine learning pipeline, the use of two classifiers and one feature selection method are proposed for the experiments.

The reminder of this document is organized as follows: Section 2 describes in detail the concepts of discretization and fixed-point representation. Section 3 enumerates the different datasets and algorithms used during this work. Section 4 details how the adaptation of the MDLP algorithm was implemented and describes the experiments. Section 5 shows and analyzes the results of the experimentation. Finally, section 6 draws the conclusions of this study and outlines possible future lines of work.

2. Background

2.1. Discretization process

Continuous numerical variables may sometimes exhibit large biases or non-standard distributions, which can be caused by anomalies in the data (*outliers*), multimodal or exponential distributions, etc. These conditions can degrade the performance of machine learning algorithms, which tend to perform better when the input data follows a Gaussian probability distribution. Thus, the discretization process transforms continuous variables into discrete values, creating groups of contiguous intervals (partitions or *bins*) over which the input data is mapped. Although part of the information present in the data is lost in the discretization process, it provides several important advantages: it simplifies the problem, reduces the effect of *outliers*, limits the degrees of freedom of the input data, and filters the noise of the samples, among others. In addition, there are some algorithms that are incompatible with continuous variables and

require discrete input data, as in the case of a Naive Bayes classifier or the Mutual Information Maximization feature selection method.

There are several popular discretization techniques, but they all work under the same premise: to find the best *cutpoints*, based on a certain metric, that divide the range of a continuous variable into multiple partitions. A typical discretization process has three steps: sort the continuous values of the variable to be discretized, evaluate different cutpoints according to a given criterion and finally terminate the search when some stop condition is met. Among the different discretization algorithms in the literature, entropy-based algorithms have usually achieved remarkable results [13]. One of the most popular methods is the one proposed by Fayyad and Irani [3] that uses a stopping criterion based on the *Minimum Description Length Principle* (MDLP). This method provides good results especially for environments where Naive Bayes classifiers or decision trees are used.

In this work, the aforementioned MDLP-based discretization method has been selected to evaluate how using a fixed-point representation may affect the results, also considering different formats (number of bits) in this study. Given a set S of instances, an attribute A, and a cutpoint T (responsible of dividing the set into two subsets), the class information entropy of the partition induced by T, denoted as $E(A,T;S)$, is defined as:

$$E(A,T;S) = \frac{|S_1|}{|S|} \cdot Ent(S_1) + \frac{|S_2|}{|S|} \cdot Ent(S_2) \tag{1}$$

where $Ent(S_i)$ is the class entropy of the subset S_i defined as:

$$Ent(S_i) = -\sum_{j=1}^{k} P(C_j, S_i) \cdot log(P(C_j, S_i)) \tag{2}$$

where there are k clases $C_1, ..., C_k$ and $P(C_j, S_i)$ is the proportion of examples in S_i that belong to class C_j. For an attribute A, the MDLP method selects a cutpoint T_A for which $E(A,T_A;S)$ is minimal among all the boundary points. The training set is then split into two subsets by the cutpoint. Subsequent cutpoints are selected by recursively applying the same binary discretization method to each of the newly generated subsets until the following condition is achieved:

$$Gain(A,T;S) \leq \frac{log_2(N-1)}{N} + \frac{\Delta(A,T;S)}{N} \tag{3}$$

where:

- N is the number of examples in S
- $Gain(A,T;S) = Ent(S) - E(A,T;S)$
- $\Delta(A,T;S) = log_2(3^k - 2) - [k \cdot Ent(S) - k_1 \cdot Ent(S_1) - k_2 \cdot Ent(S_2)]$
- k, k_1 and k_2 are the number of classes represented in the sets S, S_1 and S_2 respectively

2.2. Fixed-point representation

Nowadays, floating point is the most widely used format in practically any application, whether or not they are related to the area of artificial intelligence, and is commonly used to represent real values. But sometimes there are applications that have strict requirements of size, power consumption, memory, weight, etc. In this kind of context, it may be necessary to make use of a lighter format.

The **fixed-point** format consists of a signed mantissa and a scaling factor that defines the placement of the point that separates the integer part of the decimal part, known as *radix-point*. Reducing the scaling factor would be equivalent to moving decimal point to the right, which would increase the number of bits dedicated to the integer part and decrease the bits for the decimal part. This increases the range at the cost of reducing precision. There are several ways to represent a fixed-point format with a given *radix-point*. One of the most widespread notations is $< IL, FL >$, where IL is the number of bits for the integer part and FL is the number of bits for the fractional part. Thus the range would be defined as $[-2^{IL-1}, 2^{IL-1} - 2^{-FL}]$ and the precision, or *epsilon*, would be equal to 2^{-FL}.

Fixed-point format is typically found in embedded systems that do not have an FPU (Floating Point Unit), such as microcontrollers, microprocessors or certain models of FPGAs[1]. In addition, there are also other situations in which, even with the possibility of using floating-point operations, it may be preferred to use fixed-point format because of the advantages it offers. These advantages are mainly the lower use of memory and a higher speed of calculations, from which other advantages arise as a consequence: lower consumption, lower hardware cost, reduction of carbon emissions, etc. Sometimes these circumstances can even be a requirement, as in Internet of Things applications, embedded systems (drones, satellites, ...), aeronautics or industrial sector, to give a few examples.

3. Materials and methods

This section introduces the different materials used during the development and the experimentation. These materials are: the datasets, the discretization library and some complementary algorithms. All experiments were conducted on a laptop with the following technical characteristics: Intel i7-10510U CPU @ 2.30 GHz, 16 GB of RAM and Windows 10 Pro OS.

3.1. Datasets

In this work, a total of five datasets have been used, where three of them are binary and the other two are multiclass. Also, three of them are microarrays, characterized by having a very high number of features with respect to the number of samples. Table 1 displays an overview of these datasets.

Table 1. Characteristics of the datasets

Dataset	#Features	#Samples	#Classes	Reference
breast-cancer	30	569	2	https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
madelon	500	2400	2	https://archive.ics.uci.edu/ml/datasets/madelon
colon-cancer	2000	62	2	https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html
leukemia1	5327	72	3	https://portals.broadinstitute.org/cgi-bin/cancer/datasets.cgi
TOX-171	5748	171	4	https://jundongl.github.io/scikit-feature/datasets.html

3.2. Algorithms and libraries

The present state of the art has been reviewed in pursuit of any open-source library of MDLP algorithm and any other auxiliary library with the goal of reusing and adapting their code, in order to fasten the development. Different libraries have been identified and tested, and eventually the following libraries were chosen for the development of this study:

- ***navicto/Discretization-MDLPC*** was implemented in 2018 and based on Fayyad-Iranis's publication on MDLP discretizer [3]. It is capable of calculate the optimal cutpoints for a given dataset and also obtain the corresponding partitions. The behaviour of the library was validated against the existing R package for MDLP algorithm (*R/mdlp.R*[2]).
- ***Naive Bayes*** [1]: Probabilistic classifier based on the application of Bayes' theorem assuming conditional independence between each pair of predictor variables. This assumption makes the algorithm very fast at the cost of losing accuracy when such independence is not given. This algorithm supports discrete variables and works really well on high dimensionality data, such as microarrays.

[1] Acronym for Field Programmable Gate Array. FPGAs are semiconductor devices based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing.
[2] https://rdrr.io/cran/discretization/src/R/mdlp.R

- ***Decision tree*** [2]: Non-parametric supervised learning method for classification and regression. The algorithm creates a model that is able to predict the class based on simple decision rules inferred during training. This algorithm is fast and low complexity, and also supports discrete variables.
- ***mRMR*** [12]: Feature selection method that selects those features that have the highest relevance with respect to the target class and also show a higher intervariable dependency. The optimization criterion is called *minimum-redundancy-maximum-relevance* (hence the name mRMR) and is based on mutual information. The intention behind using this algorithm is to analyze how fixed-point discretization affects other algorithm apart from the classification ones.
- ***francof2a/fxpmath***[3]: Among the different open source libraries available online for the implementation of fixed-point operations, this library was found to be the best suited for this study. It supports various arithmetic operations, bit-level manipulation and configuration of the radix-point. It gives the necessary support to convert all arithmetic operations that appear in the MDLP algorithm to fixed-point precision.

All these libraries, aside from ***navicto*** and ***fxpmath***, can be found under the scikit-learn and PymRMR packages.

4. Methodology

This section summarizes the main contributions that have been made during the development of this work. The first subsection presents the key features of the selected MDLP library, as well as what has been done to adapt it to fixed-point format. The second subsection describes the testing setup and the experiments.

4.1. Adapting MDLP algorithm to fixed-point format

Different types of operations can be found within the source code of the *navicto* library, but only a few of them affects the accuracy of the results: arithmetic and logarithmic operations. All other operations make use of the data, but neither modify it nor have any implication on the accuracy.

4.1.1. Adapting arithmetic operations

The process of adapting arithmetic operations to fixed-point is a simple but tedious process. The first thing to do is to define a fixed-point format template, which will be used to adapt all operations to the selected format, named *FMT* from here onwards.

Once this template is defined, the adaptation process consists in taking each line of code in which an arithmetic operation is performed and replacing it with the fixed-point version from the *fxpmath* library. In those cases where there are more than two elements in an arithmetic operation, it is necessary to always convert the intermediate operations, one by one, to the selected format, to prevent the library from automatically adjusting the format during partial calculations. For example, the `X = A + B + C` operation would be converted as:

- `A_fxp = A.like(FMT); B_fxp = B.like(FMT); C_fxp = C.like(FMT)`
- `partial = (A_fxp + B_fxp).like(FMT)`
- `X = (partial + C_fxp).like(FMT)`

4.1.2. Adapting logarithmic operations

Three operations with logarithms can be found in the code, but *fxpmath* library does not offer support for logarithmic operations, so it is necessary to find an alternative solution. Two of these logarithmic operations appear during the calculation of the MDLP criterion, by which a cutpoint is accepted or discarded:

$$\log_2(3^k) \tag{4}$$

$$\log_2(size(partition) - 1) \tag{5}$$

[3] https://github.com/francof2a/fxpmath

While the third appears during entropy calculation:

$$\log_2\left(\frac{size(partition)}{\#total\_examples}\right) \tag{6}$$

In the Equation (4) the variable k represents the number of total classes, which is part of the equation (3) shown in section 2.1. This equation has an algebraic identity that transforms it into an arithmetic operation:

$$\log_2(3^k) = k \cdot \log_2(3) = k \cdot 1.585 \tag{7}$$

Whereas to implement the Equations (5) and (6) in fixed-point the most typical solution is to make use of lookup tables that have been precomputed and stored in memory. Thus to obtain the result of an operation, such as $log(x)$, the calculation is not performed, but the result is taken directly from memory using the value of x for indexing. Equation (5) is the simplest since the argument only takes integer values, so it is sufficient to compute and store in memory the logarithms of integers from 0 to the total number of data samples. In Equation (6) the argument is the division of two integer values, the result being a decimal number. In this case the table has to be two-dimensional, and it will be indexed based on the number of occurrences of an event (partition size) and the total number of events (total number of samples). Again, the values are precomputed and stored in memory in reduced precision.

4.2. Experimental setup

The workflow used during the experiments is presented in Figure 1. The figure shows only the classification scenario, where an algorithm is trained with the training set and the resulting model is used for classification of the test set. In the case of feature selection the diagram would be similar, but using the mRMR algorithm right after the discretization process, instead of a classifier.

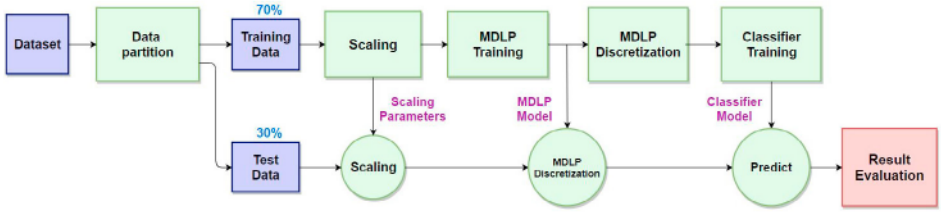

Fig. 1. Workflow during the experiments (only the classification scenario is represented)

The pipeline starts from a dataset (with no missing values), which is partitioned in a ratio of 70% and 30% for training and test respectively. To ensure reproducibility of the results, the same static seed is used for the partitioner in all executions. Next, the input data is standardized so that it has zero mean and unit variance. This is especially important when using fixed-point format to take better advantage of the range and precision offered by the number of bits being used. The mean and scaling values obtained on the training set are used for the standardization of the test set.

In the next step, the MDLP algorithm is trained on the training set. The output of this block is the model with the optimal cutpoints found by the algorithm. These cutpoints are then used to discretize both the training set and the test set. Lastly, one of the algorithms proposed in the 3.2 subsection (Naive Bayes classifier, decision tree or mRMR feature selection) is applied in order to evaluate the results of the discretization process.

Based on this workflow, the following experiments are performed:

- **Cutpoints**: The aim is to analyze how the results differ at the output of the fixed-point version of MDLP compared to the floating-point version. Once the number of cutpoints is known, the number of total partitions can be calculated as $\#partitions = \#cutpoints + \#features$. Unlike the following experiments this is a direct measure of the performance of the MDLP algorithm.
- **Classification accuracy**: Analysis of the accuracy results obtained by Naive Bayes and decision tree classifiers when the data has been previously discretized by either the MDLP original version (double floating-point) or its fixed-point adaptation.

- **Similarity rate in feature selection**: This experiment uses the mRMR algorithm to identify the 10 most relevant features. The mRMR algorithm is applied right after MDLP discretization for both the fixed-point and the original version (double-precision). The **similarity rate**, in percentage, is defined as the number of features selected by mRMR that match in both scenarios.
- **Training time and carbon emissions**: It measures the MDLP algorithm training time and how much is the carbon emission (in kilograms) of this process. This last measurement is obtained by using *codecarbon*[4] library.

Each of the experiments is repeated for different fixed-point formats: sizes of 12, 16 and 32 bits. For each given size, half of the bits are assigned to the fractional part (that is, 6, 8 and 16 bits respectively), while the other half is allocated to the integer part and the sign. Given the *Fxp* naming convention, these formats can be represented as *fxp-s32/16*, *fxp-s16/8* and *fxp-s12/6*.

5. Experimental results

This section presents the results of the experiments that have been defined in the subsection 4.2. The initial goal in conducting these experiments was to be able to perform exhaustive tests with different scenarios: bit number and *raxid-point* sweeps, cross-validation, statistical tests, etc. However, the *fxpmath* library is far from optimized and it was only viable to run a single execution of each experiment within the scope of this work.

5.1. Cutpoints and number of bins

Table 2 shows the resulting number of cutpoints and partitions after running both the original MDLP algorithm and the fixed-point version when using different bit sizes (12, 16 and 32 bits). As mentioned in subsection 4.2, there is a direct relationship between cutpoints and partitions, so conclusions can be drawn by the results shown in this table.

Table 2. Number of cutpoints and partitions returned by the MDLP algorithm

	Number of cutpoints				Number of partitions			
Dataset	64-bit	fixed point (#bits)			64-bit	fixed point (#bits)		
	floating point	32	16	12	floating point	32	16	12
breast-cancer	27	26	19	11	57	56	49	41
madelon	12	12	0	0	512	512	500	500
colon-cancer	89	89	88	25	2089	2089	2088	2025
leukemia1	626	626	627	181	5953	5953	5954	5508
TOX-171	1023	1023	1047	9	6771	6771	6795	5757

At first glance, the results for 32 bits (fixed-point) are particularly noteworthy since they are practically identical to the original version. There is only one case, *breast-cancer*, where the number of cutpoints differs by one unit. This result alone can be considered highly relevant, since it reflects that the original algorithm (double precision) can be adapted to fixed-point barely affecting the resulting partitions, with all the benefits this format provides.

When 16 bits are used, the results start to show minor differences. For some datasets the number of resulting cutpoints is reduced, as in *breast-cancer* or *madelon*, even not finding any cutpoint in the latter case (notice that this is a very challenging dataset for machine learning researchers used in competitions). For *TOX-171* the opposite effect occurs, where the fixed-point algorithm obtains more cutpoints, a mere side effect of the loss of precision due to the use of fewer bits. Finally, when the bits are limited to 12, the number of cutpoints are drastically reduced for all datasets. This, per se, is not necessarily a negative result, as there may be classifiers that perform better with simpler partitions, as will be seen next.

[4] https://github.com/mlco2/codecarbon

5.2. Classification accuracy for Naive Bayes and Decision Tree algorithms

Table 3 shows the accuracy results for the Naive Bayes and Decision Tree classifiers when the data has been previously discretized by the MDLP algorithm, using either its original version or the fixed-point adaptation (12, 16 and 32 bits).

Table 3. Classification accuracy (%) for Naive Bayes and Decision Tree algorithms (best result for each dataset is highlighted in bold)

	Naive Bayes				Decision Tree			
Dataset	64-bit	fixed point (#bits)			64-bit	fixed point (#bits)		
	floating point	32	16	12	floating point	32	16	12
breast-cancer	95.91	**98.25**	**98.25**	95.91	96.49	96.49	**97.08**	94.74
madelon	**71.81**	**71.81**	49.58	49.58	80.00	**80.14**	49.58	49.58
colon-cancer	57.89	57.89	57.89	**63.16**	52.63	52.63	52.63	**73.68**
leukemia1	77.27	77.27	77.27	**81.82**	81.82	**90.91**	86.36	86.36
TOX-171	**75.00**	**75.00**	**75.00**	44.23	**55.77**	51.92	51.92	48.08

As in the previous section, the accuracy results using 32 bits are practically identical to those of double precision. In fact, for *breast-cancer*, using one cutpoint fewer gives more accuracy for Naive Bayes, increasing from 95.91% to 98.25%. Since these results correspond to a single run of the experiment, this difference might be due to the fact that the random distribution of the partitions has in this case benefited the fixed-point version of the algorithm.

When the Decision Tree classifier is used, the most notable difference appears for *leukemia1* using 32 bits, which obtains a much higher precision value than the original (90.91% vs. 81.82%). The cause of this effect is that this dataset is a microarray, with only 72 samples (50 for training, 22 for testing). This means that a successful classification increases the accuracy by $1/22 = 4.545\%$. Therefore, the difference observed for 32 bits with respect to the original version (+9.09%) implies that the former has two more correct results than the latter. This effect can occur when the samples are extremely close to a certain cutpoint found by MDLP, so that any perturbation, however small, can cause them to fall into a contiguous partition. In this case, the use of reduced precision is the cause of this small perturbation.

The results for 16 bits are interesting, since although the number of cutpoints varied slightly compared to the 32 bits version, the accuracy values remain quite the same. The only notable exception occurs for *madelon*, which, as it did not find any cutpoint, the discretization process does not generate any partition, making it innefective to use. In the case of using 12 bits the results vary greatly, given that MDLP algorithm in this case returns drastically fewer cutpoints for all datasets.

5.3. Similarity rate in feature selection

The 10 variables selected by the mRMR algorithm after discretizing the input data with the fixed-point MDLP algorithm are compared with those obtained when the original MDLP was used instead. The similarity rate of the selected features between both cases is given in Table 4.

Table 4. Similarity rate (%) in mRMR feature selection method

Dataset	64-bit	fixed point (#bits)		
	floating point	32	16	12
breast-cancer	baseline	100	70	70
madelon	baseline	100	100	100
colon-cancer	baseline	100	100	100
leukemia1	baseline	100	100	100
TOX-171	baseline	100	70	90

Once more, it can be seen that the 32-bit fixed-point version gets identical results to the original algorithm. For 16-bit and 12-bit, depending on the dataset, some minor differences show up. In particular, while 32-bit and 12-bit versions for *TOX-171* practically match the original version, there are three selected features that differ with respect

to the 16-bit version, what may seem counterintuitive. Here, again, something similiar to the previous subsections occurs: this dataset is a microarray with very few examples (171) and too many features (5748). Different fixed-point formats can generate slightly different partitions causing the features to climb up or down within the ranker. Thus, any other of the many features may be detected as more relevant. In addition, microarray datasets usually have many redundant features and they can be swapped without altering the classification results.

5.4. Training time and carbon emissions

Training times were measured for the double-precision and fixed-point (32-bit) versions for all datasets. The results showed that the fixed-point version takes between 2 and 3 orders of magnitude longer than the original version, which can be counterintuitive. The main issue is that *fxpmath* library emulates the behavior of fixed-point arithmetic by using complex structures and updating some state information after each operation, which is costly both in memory and computational resources. It is important to note that implementing the fixed-point algorithm for a real environment, such as in a microcontroller, would deliver an opposite outcome: memory used would be proportional to the bitwidth used (e.g. 16-bit fixed-point format would need four times less memory than double precision format) and also the training time would be reduced in a similar ratio.

Although comparing the training time between the two versions does not yield useful results in this case, what may be enlightening is to measure how much the computation time decreases as the number of bits used for fixed-point operations is reduced. To do this, *breast-cancer* dataset is taken as an example and the training times for different bit sizes are measured. Figure 2 shows these results, where it can be seen that the training time grows almost linearly with the number of bits. This information can be extrapolated, for example, to a microcontroller supporting 8, 16 and 32 bit operations, in which a similar behaviour could be expected.

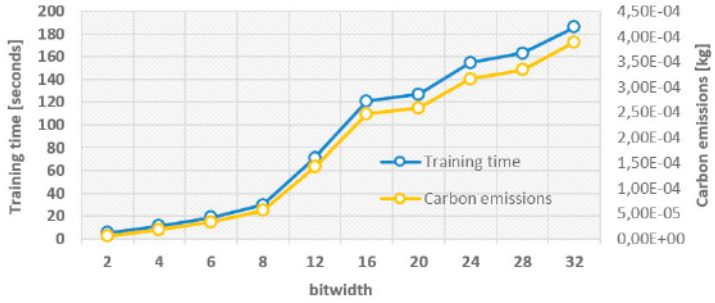

Fig. 2. Training time and carbon emissions for the *breast-cancer* dataset

In Figure 2, the vertical axis on the right represents the carbon emissions in kilograms as reported by the *codecarbon* library. This library uses processor computation time to draw these results, so they are expected to show a high correlation with regard to the training time. In any case, it can be observed that carbon emissions also decrease as the number of bits is reduced. For the same reasons discussed above, in a real environment these emissions would be reduced by several orders of magnitude. For reference, using the double precision discretizer for this same data, the value of carbon emissions is $1.58 \cdot 10^{-6}$ kilograms.

6. Conclusions and future work

This work constitutes a first milestone in the use of fixed-point arithmetic in the discretization stage within a machine learning pipeline. The results show that it is totally plausible to replace algorithms that operate in double precision with reduced precision. The following conclusions can be derived from the different experiments:

- A double precision version of the MDLP discretization algorithm was found online and its code was adapted to work with reduced precision. This implementation serves as a point of reference for any experiment that seeks to emulate the behavior of the fixed-point MDLP algorithm on low-performance devices.

- It has been found that, at least for the MDLP algorithm and for the datasets chosen for this study, the results are practically the same either using the original version of the algorithm, in double precision, or using the 32-bit fixed-point version. Furthermore, depending on the input dataset 16-bit and 12-bit versions can also return acceptable results.
- By emulating fixed-point operations with the *fxpmath* library, it has been observed that reducing the number of bits for arithmetic operations has a direct implication on the training time and, as a consequence, also in the amount of carbon emissions generated by the device which runs the algorithm.

Certainly, the use of reduced precision in discretization algorithms is an area of research in which there is still a long way to go. As future work, we plan to use look-up tables for logarithmic operations [15] in order to reduce memory space, and to migrate our algorithm to a real environment, such as a microcontroller or FPGA, so that actual accurate measures can be obtained.

Acknowledgements

This work has been supported by the grant *Machine Learning on the Edge - Ayudas Fundación BBVA a Equipos de Investigación Científica 2019*. It has also been possible thanks to the support received by the National Plan for Scientific and Technical Research and Innovation of the Spanish Government (Grant PID2019-109238GB-C2), and by the Xunta de Galicia (Grant ED431C 2018/34) with the European Union ERDF funds. CITIC, as Research Center accredited by Galician University System, is funded by "Consellería de Cultura, Educación e Universidades from Xunta de Galicia", supported in an 80% through ERDF Funds, ERDF Operational Programme Galicia 2014-2020, and the remaining 20% by "Secretaría Xeral de Universidades" (Grant ED431G 2019/01).

References

[1] Bayes, T., 1763. An essay towards solving a problem in the doctrine of chances. Philosophical transactions of the Royal Society of London , 370–418.
[2] Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J., 2017. Classification and regression trees. Routledge.
[3] Fayyad, U., Irani, K., 1993. Multi-interval discretization of continuous-valued attributes for classification learning .
[4] Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P., 2015. Deep learning with limited numerical precision, in: International conference on machine learning, PMLR. pp. 1737–1746.
[5] Hacibeyoglu, M., Arslan, A., Kahramanli, S., 2011. Improving classification accuracy with discretization on data sets including continuous valued features. Ionosphere 34, 2.
[6] Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M.A., Dally, W.J., 2016. Eie: Efficient inference engine on compressed deep neural network. ACM SIGARCH Computer Architecture News 44, 243–254.
[7] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y., 2017. Quantized neural networks: Training neural networks with low precision weights and activations. The Journal of Machine Learning Research 18, 6869–6898.
[8] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D., 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2704–2713.
[9] Lustgarten, J.L., Gopalakrishnan, V., Grover, H., Visweswaran, S., 2008. Improving classification performance with discretization on biomedical datasets, in: AMIA annual symposium proceedings, American Medical Informatics Association. p. 445.
[10] Morán-Fernández, L., Sechidis, K., Bolón-Canedo, V., Alonso-Betanzos, A., Brown, G., 2020. Feature selection with limited bit depth mutual information for portable embedded systems. Knowledge-Based Systems 197, 105885.
[11] Murshed, M., Murphy, C., Hou, D., Khan, N., Ananthanarayanan, G., Hussain, F., 2019. Machine learning at the network edge: A survey. arXiv preprint arXiv:1908.00080 .
[12] Peng, H., Long, F., Ding, C., 2005. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on pattern analysis and machine intelligence 27, 1226–1238.
[13] Rabaséda-Loudcher, S., Sebban, M., Rakotomalala, R., 1996. Discretization of continuous attributes: a survey of methods, in: Proceedings of the 2nd annual Joint Conference on Information Sciences, pp. 164–166.
[14] Sharma, A., Paliwal, K.K., Imoto, S., Miyano, S., Sharma, V., Ananthanarayanan, R., 2014. A feature selection method using fixed-point algorithm for dna microarray gene expression data. International Journal of Knowledge-Based and Intelligent Engineering Systems 18, 55–59.
[15] Tschiatschek, S., Pernkopf, F., 2015. Parameter learning of bayesian network classifiers under computational constraints, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer. pp. 86–101.