https://doi.org/10.31891/csit-2023-2-1
UDC 519.6

Lesia MOCHURAD
Lviv Polytechnic National University

CUDA-BASED PARALLELIZATION OF GRADIENT BOOSTING AND BAGGING ALGORITHM FOR DIAGNOSING DIABETES

Data, its volume, structure, and form of presentation are among the most significant problems in working in the medical field. The probability of error is very high without innovative high-tech data analysis tools. It is easy to miss an important factor that is critical but lost among other, less important information. This work aims to study the proposed parallel gradient boosting algorithm in combination with the Bagging algorithm in the classification of diabetes to achieve greater stability and higher accuracy, reduce computational complexity and improve performance in medicine. The methods of parallelization of the Gradient Boosting algorithm in combination with the Bagging algorithm are investigated in the paper. Performance scores were obtained: approximately 7 using ThreadPoolExecutor and an eight-core computer system and 9.5 based on CUDA technology. Performance indicators that go to the unit are calculated. This, in turn, confirms the effectiveness of the proposed parallel algorithm. Another significant result of the study is improving algorithm accuracy by increasing the number of algorithms in the composition. The problem of diagnosing a patient's diabetes based on specific measurements included in the data set is considered. Detailed analysis and pre-processing of the selected dataset were performed. The parallelization of the proposed algorithm is implemented using the multi-core architecture of modern computers and CUDA technology. The process of learning models and training samples was parallelized. The theoretical estimation of the computational complexity of the offered parallel algorithm is given. A comparison of serial and parallel algorithm execution time using ThreadPoolExecutor when varying the number of threads and algorithms in the composition is presented. And also, the comparative analysis of time expenses at consecutive and parallel execution based on CPU and GPU is carried out.

Key words: ensemble of models, acceleration, graphic processor, CUDA technology, machine learning, classification problem.

Леся МОЧУРАД
Національний університет «Львівська політехніка»

РОЗПАРАЛЕЛЮВАННЯ АЛГОРИТМУ ГРАДІЄНТНОГО БУСТИНГУ ТА БЕГГІНГУ ДЛЯ ДІАГНОСТИКИ ДІАБЕТУ НА ОСНОВІ CUDA

Важливим аспектом, який відрізняє медичні дані від більшості інших, є те, що об'єктивність, точність, якість і своєчасність результатів є критично важливими і повинні постійно ставитися під сумнів.

Одну з найбільших складнощів в процесі роботи в медичній сфері становлять дані: їх об'єм, структура та форма представлення. Очевидно, що без інноваційних високотехнологічних інструментів аналізу даних, ймовірність помилки є дуже високою, адже легко пропустити якийсь важливий фактор, що є критичним, але загубленим серед іншої, менш важливої інформації. У роботі досліджено методи паралелізації алгоритму Градієнтного Бустингу у поєднанні з алгоритмом Беггінгу. Розглянуто задачу діагностичного прогнозування наявності у пацієнта діабету на основі певних вимірювань, включених до набору даних. Проведено детальний аналіз та попередню обробку обраного датасету. Розпаралелювання запропонованого алгоритму реалізовано за допомогою модуля Python – concurrent. Futures, а саме його класу – ThreadPoolExecutor та технології CUDA. Здійснено паралелізацію процесу навчання моделей та навчальної вибірки. Наведено теоретичну оцінку обчислювальної складності запропонованого паралельного алгоритму. Проведено порівняння часу виконання послідовного та паралельного алгоритмів при використанням ThreadPoolExecutor при варіації кількості потоків та алгоритмів в композиції. А також здійснено порівняльний аналіз часових витрат при послідовному та паралельному виконанні на основі CPU і GPU. Отримано показники прикорення: приблизно 7 при використанні ThreadPoolExecutor та восьмиядерної обчислювальної системи та 9,5 на основі технології CUDA. Обчислено показники ефективності, які прямують до одиниці. Що, в свою чергу, підтверджує ефективність запропонованого паралельного алгоритму. Ще одним важливим результатом дослідження є досягнення кращої точності алгоритму за рахунок збільшення кількості алгоритмів у композиції.

Ключові слова: ансамбль моделей, прискорення, графічний процесор, технологія CUDA, машинне навчання, задача класифікації.

Introduction

Today, one of the most important factors influencing the development of human society is artificial intelligence (AI). The latter is used in various fields of health and medicine [1-3]. An important aspect that distinguishes medical data from most others is that results' objectivity, accuracy, quality, and timeliness are critical and should be constantly questioned.

Various methods of machine learning for disease diagnosis have attracted attention in recent years [4]. With the increase in the number of people coming to hospitals, conventional traditional methods may not be enough. Thus, various forms of AI, such as natural language processing [5, 6], data processing algorithms [7, 8], clustering, and classification [9, 10], should be used to improve the efficiency of the medical system significantly.

The use of AI has some benefits: it reduces human error, saves time and money, and improves service delivery. Moreover, machine learning methods have repeatedly shown better accuracy compared to medical personnel.

The basis of medical diagnosis is the problem of classification. The diagnosis can be reduced to a problem with displaying data to one of the different results. In some cases, the task is only to determine based on data (such as radiographs or electrocardiography) whether the patient has a specific disease – (1) or not – (0).

Over the past few years, the growth of diabetes among people has become exponential [11, 12]. A health report shows that about 347 million people worldwide suffer from diabetes. Diabetes affects not only the elderly but also the younger generation. Detecting diabetes at an early stage is a big problem. Therefore, diagnosing diabetes at an early stage will be helpful in the decision-making process of the medical system and will help save lives. After all, prolonged diabetes leads to the risk of damage to vital organs of the human body.

Globally, the incidence of diabetes is snowballing, even if we consider the increase in population in recent decades. This, in turn, leads to rapid growth in the amount of medical data. This increase in health data means that AI will be increasingly used in this area. In this case, for the effective operation of AI methods, their accuracy and the speed of execution must be high, which makes the topic of the study relevant.

The healthcare information technology industry is changing daily. As a result of this rapid development, new scientific advances in machine learning have enabled the healthcare industry to utilize several revolutionary tools that use natural language processing, pattern recognition, and "deep" learning. Of course, enterprises still have work to do before machine learning and AI can meet the needs of modern medicine. Still, it is worth noting that innovative technologies are already leaving their mark in the environment of medical data analysis [13-16].

The main contribution of this article can be summarized as follows:

1. We have proposed an accurate and computationally efficient approach to the parallelization of the combination of two algorithms – Gradient Boosting and Bagging – to solve the diagnostic prediction of the presence of diabetes in patients based on specific measurements. This provides an opportunity to significantly optimize the computational process by parallelizing it, solving significant data processing in medicine;

2. We have developed an algorithmic implementation of the proposed approach. The relevance of the development of the multi-core architecture of modern CPUs and the use of CUDA technology for graphics processors was taken into account;

3. We have demonstrated the reduction of computational complexity using the proposed algorithm; we have an acceleration of approximately seven times for the octa-core architecture of the corresponding computing system when using ThreadPoolExecutor (this result can be significantly improved by choosing a CPU with more than eight cores) and 9.5 times for GPUs based on CUDA technology; the efficiency indicator which goes to the unit is calculated; achieving better algorithm accuracy by increasing the number of algorithms in the composition, which is an excellent opportunity to increase the efficiency of the model.

Related works

Since the problems investigated in this paper are relevant, an analysis of the literature related to this range of tasks was conducted. This provided an opportunity to study the studied algorithms in detail, determine the achievements in their use for the diagnosis of diabetes, and explore existing approaches to improve the Gradient Boosting algorithm.

In [17], the author used boosting and running to enhance the decision tree algorithm in predicting diabetes risk. An accuracy of 89.65% was achieved. Despite the high precision, the question of the execution time of the algorithms remains open. After all, small amounts of data (768 instances) [18] were used by the author, which suggests that the time will be too extensive with more data. So, in contrast to the previous one, in this paper, the dataset [19] that provides information about 10000 patients was chosen in order to find and test methods of accelerating the proposed algorithms.

Many machine learning methods have been discussed, starting from different basic algorithms such as the logistic regression, a modified support vector machine, a decision trees, to further classification including the Iterative Dichotomiser 3, C4.5, C5.0, J48 and Classification and Regression Tree and the naïve Bayes on diabetes detection [20]. Ensemble methods, such as Bagging, Boosting, and Random Forest (RF) regressors, are further used to enhance the accuracy and performance of models. These techniques have been implemented on all types of platforms such as Python or MATLAB, and the models have been analyzed using different parameters such as area under curve or confusion matrices or mathematical terms such as the Root-mean-square error or Mean absolute error. However, the work does not provide an analysis of parallel algorithms for solving the problem of big data analysis on diabetes detection. And today, the possibility of speeding up decision-making in medicine is extremely important.

A comparison of gradient boosting with two other machine learning algorithms: RF and neural networks on a dataset for diagnosing diabetes was carried out in [21]. Studies have shown the benefits of boosting. This article provided a better understanding of the benefits of the practical application of this method.

The research paper [22] presents a methodology forclassification of diabetic and normal heart rate variability signals using deep learning architectures. The classification system proposed can help the clinicians to diagnose diabetes using electrocardiogram signals with a very high accuracy of 95.7%. As the authors note, the highest value published for the automated diabetes detection with HRV as input data was obtained in the work. And further improvement in accuracy can be obtained using a very large-sized input dataset. However, this, in turn, requires modification of existing algorithms in order to obtain a solution in real-time.

In [23], the authors improved the parallel efficiency of the decision tree by proposing a new GBDT system – HarpGBDT. This approach includes a block strategy of parallelism and extension of the TopK tree growth method (which selects the best K candidates of tree nodes to allow more levels of parallelism without sacrificing the

algorithm's accuracy). In addition to the description of this approach, the paper presents a comparative analysis with other parallel implementations, which states that despite the many advantages over other approaches, the operating time is longer. This, in turn, indicates the need to find a method of parallelization, which will give a significant acceleration among the many advantages presented in the paper.

The authors paralleled Boosting and Bagging [24]. The research was conducted using decision trees and two standard data sets. The main results are that the sample size limits the achieved accuracy, regardless of the computational time. Therefore, this work encourages experiments on larger data and attempts to improve accuracy. The paper also demonstrated the parallelization of Boosting and Bagging methods separately, confirming our proposed method's novelty.

In [25], the authors developed the idea of ensemble learning, combining adaptive Boosting and Bagging for binary classification. The algorithms were tested on different data sets, showing improved accuracy and reduced error rates. However, this algorithm has increased the computation time. This study further actualizes the problem of parallelization of the combination of Boosting and Bagging algorithms.

In the work [26] a stacked ensemble-based deep neural network approach is proposed for diabetes possibility assessment in the early stages. As a result, the proposed method achieved the highest success rate with 99.36% accuracy and 99.19% AUC. But the proposed approach was tested on a dataset of 520 patients. In the work, the authors did not investigate the problem of speed of decision-making and changes in accuracy when the number of patients increases.

Thus, an analysis of the literature has shown that the use of the Gradient Boosting algorithm to diagnose diabetes is not sufficiently studied. Researches usually use an insignificant volume of data, and the realized parallelization marches do not provide the expected acceleration. In addition, no works have been found that describe the development of a similar algorithm to what will be implemented and investigated in the following sections. This once again confirms the value and relevance of this work.

Methodology

Combining Gradient Boosting and Bagging

As is known [27], Begging classification technology uses compositions of algorithms, each of which learn independently. In this case, each model is studied on a separate sample formed from the original data set. The output of the ensemble is determined by averaging the outputs of the basic models. The method allows to improve the accuracy and stability of machine learning algorithms, reduce error variance and reduce the effect of retraining. The method was initially developed for classifiers based on decision trees, but now it can be used for any model. At any given time, the model results are weighted based on previous results. Correctly predicted results are given less weight, and those that do not meet the classification – more. Therefore, several subsets are randomly selected from the set of source data, containing the number of objects corresponding to the number of objects in the original set. Remember that because the selection is random, the set of objects will always be different: some examples fall into several subsets and some into none. A classifier is built based on each sample. The results of the classifiers are aggregated. Aggregation of results usually occurs by averaging or voting. Moreover, the first option is used in the regression problem, and the second – classification.

Advantages:

- a significant increase in the accuracy of the prediction of the ensemble relative to the basic classifiers (from 10% to 40%). It is achieved by reducing the variance of the predictions of the basic models in averaging. Scatter is the variance of the answers of our models. The scatter shows how sensitive to small changes in the sample the algorithms are;
- small offset. Offset – the deviation of the average answers of all models from the answers of the most optimal model. The offset characterizes how complex the family of algorithms is, how much it can restore complex patterns;
- the use of bagging reduces deviations.

Disadvantages:

- insufficient mathematical justification for improving the accuracy of predictions.

Like bagging, the main task of this method of aggregation is to convert a set of weak classifiers (i.e., those that assume many errors in the test sample) into a stronger one. But, unlike the previous one, the training takes place sequentially, and each subsequent classifier aims to compensate for the shortcomings of the previous one. The result is usually a weighted linear combination of responses of all algorithms used.

The general idea of Boosting algorithms is to consistently apply predictors so that each subsequent model minimizes the error of the previous one [26].

Advantages:

- as an ensemble model, boosting is an easy-to-read and interpretive algorithm that makes predictive interpretations easy to use;
- the ability to predict is effective through the use of cloning techniques, such as bags or RF, and decision trees;
- boosting is an elastic method that easily curbs re-equipment.

Disadvantages:

- emission sensitive, as each classifier is obliged to correct the errors of the predecessors;
- the method is almost impossible to increase. This is due to the fact that each evaluator bases his correctness on previous predictors, which complicates the ordering procedure.

The gradient boosting method works consistently by adding new ones to past models to correct the mistakes made by previous predictors.

This method changes the weights with each iteration, teaching new models on the residual error of the past (moving to a minimum loss function).

Here are the steps to implement gradient Boosting:

1. The model is based on data collection.
2. This model makes predictions for the entire data set.
3. Errors are calculated according to forecasts and true value.
4. The new model is built, taking into account errors as target variables. At the same time, we strive to find a better separation to minimize error.
5. The predictions made with this new model are combined with the previous ones.
6. Errors are recalculated using these predicted values and true values.
7. This process is repeated until the error function stops changing or until the maximum number of predictors is reached.

Basic algorithm learning is consistent. Suppose that at some point $N-1$ algorithms $b_1(x),...,b_{N-1}(x)$ are trained, i.e. the composition has the form: $a_{N-1}(x)=\sum_{n=1}^{N-1} b_n(x)$.

Now another algorithm $b_N(x)$ is added to the current composition. This algorithm is trained to minimize the composition error in the training sample: $\sum_{i=1}^{l} L(y, a(x)+b(x)) \to \min_b$.

First, it makes sense to solve a simpler problem: to determine what values of $s_1,...,s_l$ should take the algorithm $b_N(x_i)=s_i$ on the objects of the training sample, so that the error in the training sample is minimal: $F(S)=\sum_{i=1}^{l} L(y, a(x)+s) \to \min_s$, where $s=(s_1,...,s_l)$ – vector of shifts.

In other words, it is necessary to find a shift vector s that will minimize the function $F(S)$. Since the direction of the greatest decrease of the function is given by the direction of the antigradient, it can be taken as a vector s: $s=-\nabla F=(-L_z(y_1, a_{N-1}(x_1)), ..., -L_z(y_l, a_{N-1}(x_l)))$.

The components of the shift vector s are, in fact, the values that the new algorithm $b_N(x)$ must take on the objects of the training sample to minimize the composition error. Learning $b_N(x)$, thus, is a learning task on marked data, in which $\{(x, s)\}$ is a training sample.

It should be noted that the information about the initial loss function $L(y, z)$, which is not necessarily quadratic, is in the optimal shift vector s. Therefore, for most tasks in learning $b_N(x)$ we can use the quadratic loss function.

When it is necessary to solve a complex computational problem and no algorithm is ideal, ensembles are used [28]. Ensembles make it possible to combine several algorithms that learn simultaneously and correct each other's mistakes. To date, they give the most accurate results.

A total sample of elements is first taken, then divided into sub-samples. Then the sample data are fed parallelly to the input of the basic algorithms, as in the bagging. The difference is that the basic algorithms are gradient boosting algorithms. After completing the process of independent learning, the algorithms are combined into an ensemble model. By inputting test characteristics to the algorithms, we can obtain the final prediction of the model, taking the average value of all predictions.

Description of the proposed algorithm

1. Specify the required number of gradient boosting models that we want to build. Let it be n.
2. Divide the initial data N by n subsamples, where n is the number of models of gradient boosting, which is specified in step 1. We obtain N_1, ..., N_n models of size m, where $m=N/n$ is the amount of data in each model of gradient boosting.
3. Next, we implement the Bagging algorithm, running the training for each model of gradient boosting separately. In this case, each model is given a corresponding sub-sample for training (the first model is the first sub-sample, etc.). We get $w_1(\cdot)$, $w_2(\cdot)$, ..., $w_n(\cdot)$ independent weak learners (one for each subsample).

4. At each iteration, we fit the weak learner to the gradient of the current selection error relative to the current ensemble model. $s_n(\cdot) = s_{n-1}(\cdot) - c_n * \nabla_{s_{n-1}} E(s_{n-1})$, where $E(\cdot)$ is the fit error of this model, c_n is the coefficient corresponding to the step size, $-\nabla_{s_{n-1}} E(s_{n-1})$ is the gradient of the fit error relative to the ensemble model.

5. At the output of the Bagging algorithm, we obtain n trained models (classifiers) of the Gradient Boosting algorithm.

6. To perform the prediction (assigning an object to a certain class), we pass the object to each of the models, namely a number of its features that need to be classified.

7. As a result, each model of Gradient Boosting assigns an object to the class to which the probability of belonging of the object according to its calculations is the highest. Belonging of object x to each of the classes:

$$P(y = 1 | x) = \frac{1}{1 + \exp(-a(x))}, \quad P(y = -1 | x) = \frac{1}{1 + \exp(a(x))}.$$

8. Then find the average value between all predictions obtained from n models.

9. Round off the determined average value to the nearest integer, which will be the result of the algorithm of combining Bagging and Gradient Boosting, and, consequently, the class to which the transferred object belongs.

It is possible to parallelize the combination of algorithms in step 3 of the proposed algorithm, during which the models are studied sequentially, independently of each other.

Since the models learn independently, we can run their training parallelly. This is one of the main reasons why we used the idea of combining the Bagging algorithm with Gradient Boosting.

For example, when classifying the Gradient Boosting model, several classifiers are also trained. Still, this process cannot be parallelized, because here each subsequent classifier uses the results of the previous one and does not work independently, as in Bagging.

Therefore, when parallelizing, we change step 3 in the algorithm for combining Begging with Gradient Boosting, running parallel training for each model of Gradient Boosting separately. In this case, each thread will work with its subsample and its model.

Parallelization

Gradient boosting is a sequential algorithm for learning trees because the result of the previous weak algorithm is the input for another. That is why it is impossible to perform training parallelly. However, using gradient boosting as a basic algorithm for bagging will allow us to perform training parallelly.

The parallelization of bagging is quite simple. The training set is divided equally among the available processors (streams). Each processor (thread) executes a sequential algorithm until the appropriate predictions are made. In general, it is a good idea to divide the training set randomly to make sure that the predictions created by each processor (thread) do not contain unnecessary biases.

Parameters to select for the parallel ensemble technique: sample size used and number of iterations.

We will implement parallelization using the Python module – concurrent Futures, namely its class – ThreadPoolExecutor and CUDA technology [29].

We will parallel the learning process. The training sample will also be parallelized. In a single-threaded system, the training vectors will be sent to the classifier one by one. In a parallel system, these learning vectors will be separated between streams.

Threads in Python are a form of parallel programming that allows a program to perform multiple procedures simultaneously. Flow-based parallelization is especially well suited for accelerating applications that work with large amounts of data.

ThreadPoolExecutor is a utility that is built into Python 3, is located in the concurrent Futures module [30] and is designed to distribute code execution among threads (a pool of threads is formed). You must first import it from the specified module, then initialize the ThreadPoolExecutor() object.

The map function was also used, the syntax of which is as follows: *map(func, *iterables).*

The map method applies the func function to one or more iterating objects; in this case, the function is to build and train a model of the Gradient Boosting algorithm, and iterating objects are the parts into which the total data sample is divided. In this case, each function call is started in a separate thread. The map method returns an iterator with the function results for each element of the iterating object. The number of threads in which the code will run is specified in the max_workers parameter when declaring the ThreadPoolExecutor() object, applied to the map function.

So, on input ThreadPoolExecutor().map accepts a function that needs to be parallelized and arguments that need to be transferred to it. On an output, we receive an array that consists of the models trained on subsamples.

Next, a quality check will be performed independently for each model, and their quality metrics will be calculated (score and f1_score). In the end, when we leave the parallel area of our program, we will re-check the quality for a set of all models. The prediction of such a composition will be averaged and rounded.

As is known [31], the GPU speeds up programs running on the CPU by unloading some computing and time-consuming parts of the code. The rest of the program is still running on the CPU. From the user's point of view, the

program runs faster because it uses the powerful parallel processing power of the graphics processor to increase performance.

The CPU consists of several processor cores, while the GPU consists of hundreds of smaller cores. Together they work to process data. This highly parallel architecture is what gives the GPU high computing performance.

The system that executes the software implementations presented in this paper supports CUDA technology, which provides the ability to use GPUs for parallelization and acceleration of program execution.

We will use LightGBM to call CUDA technology [32]. This Python framework has been used to improve gradient boosting. It is designed to increase efficiency with the following benefits:

- Faster training speed and higher efficiency.
- Less memory usage.
- Better accuracy.
- Support of parallel, distributed and GPU learning.
- Able to process large-scale data.

Classification will be done by calling the LGBMClassifier() method from the LightGBM library described above.

The complexity of the Boosting algorithm: the complexity of the training time: $O(M*n*\log(n)*d)$, where M is the number of trees, d is the number of signs, $\log(n)$ is the depth of the tree; n is the total amount of data.

The complexity of Boosting in conjunction with Bagging: $O(n)+O(K*M*m*\log(m)*d)$, where m is the sample size of the data used, $O(n)$ is the difficulty of sampling, n is the total data size. $O(K*M*m*\log(m)*d)$ – Boosting complexity for K iterations (K – number of Gradient Boosting algorithms).

The difficulty of one round of parallel learning for Boosting in conjunction with Bagging can be expressed in the formula: $O\left(\frac{n}{P}, m\right)+O(M*m*\log(m)*d)$, where m is the size of the data subsample used, $O\left(\frac{n}{P}, m\right)$ is the complexity of sampling (which depends on both the size of the local data set in the round and the size of the selected set according to the specified number of threads), n is total data size, P – number of threads. $O(M*m*\log(m)*d)$ – Boosting complexity in one round.

Data review and analysis

The paper uses a data set based on a long-term survey conducted among residents of Framingham, Massachusetts [19]. The purpose of the classification is to predict whether a patient is at risk of developing diabetes. The dataset provides information about 10000 patients and contains 15 attributes.

Each attribute is one of the potential risk factors: demographic, behavioral, and medical risk factors.

Demographic attributes:

- Sex: man or woman (nominal value);
- Age: age of the patient (continuous values).

Behavioral attributes:

- Current Smoker: whether the patient was a smoker at the time of the examination (nominal value);
- Cigs Per Day: the number of cigarettes a person smoked on average in one day (continuous value).

Medical attributes:

- BP Meds: whether the patient was receiving medication for blood pressure (nominal value);
- Prelevant Stroke: whether the patient has suffered a stroke (nominal value);
- Prevalent Hyp: whether the patient had a hypertensive crisis (nominal value);
- Risk of coronary heart disease (CHD): whether the patient was at risk of coronary heart disease (nominal value);
- Tot Chol: total cholesterol (continuous);
- Systolic blood pressure (Sys BP) (continuous);
- Diastolic blood pressure (Dia BP) (continuous);
- Body Mass Index (BMI) (continuous value);
- Heart Rate (continuous value – in medical research, variables such as heart rate, although discrete, are considered continuous due to a large number of possible values);
- Glucose: Glucose level (continuous).

Variable for prediction:

- Diabetes: a binary value of "1" means that diabetes has been detected and "0" has not been detected.

We performed the pre-processing and analysis of data to achieve the best quality of the model and detailed acquaintance with the data.

Experiments

Time costs of sequential and parallel implementations

Let's present the execution time results of the sequential and the proposed parallel algorithm using the module ThreadPoolExecutor and analyze them, where K – the number of algorithms in the composition.

Table 1

Program execution time for sequential and parallel processing (ThreadPoolExecutor), s

K	Sequential	Parallel (threads)			
		2	4	8	16
1	0.8053	0.8185	0.8109	0.8523	0.7743
2	1.6932	0.9073	0.8598	0.9075	0.8426
4	3.1604	1.7761	0.9813	1.1275	1.0676
8	6.5957	3.3760	1.9017	1.5271	1.6136
16	12.8843	6.5524	4.0012	3.4891	3.1071
30	23.9632	12.7330	9.4793	6.1386	5.9924

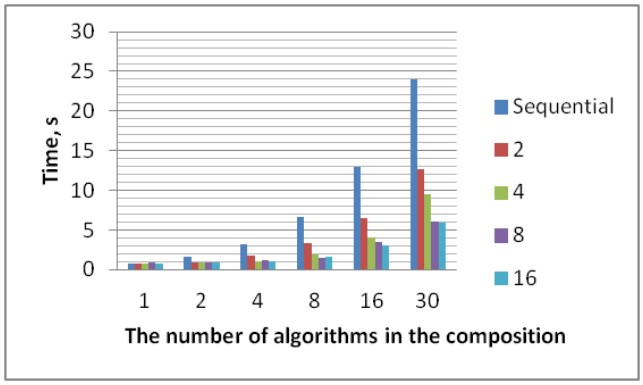

Fig. 1. Visualization of comparison of execution time of sequential and parallel algorithm (using ThreadPoolExecutor) with variation of flows and number of algorithms in a composition

From the graph shown in Figure 1 and Table 1, it can be seen that in comparison with the sequential execution of the program, run time is significantly reduced by parallelization. The greater is the number of threads, the lower is the time. Only when using more than eight threads, the time value hardly changes. This is due to the capabilities of the system's architecture on which this program was implemented (only four cores and eight logical processors, i.e., the maximum efficiency can be obtained by using eight threads).

Now let's perform a comparative analysis of the execution time of the sequential and the proposed parallel algorithm based on CPU and GPU.

Table 2

Program execution time for sequential and parallel processing on the CPU and GPU, s

K	Sequential	Parallel (CPU)	Parallel (GPU)
1	0.8053	0.7743	0.0842
2	1.6932	0.8426	0.2444
4	3.1604	1.0676	0.3494
8	6.5957	1.5271	0.7865
16	12.8843	3.1071	1.4916
30	23.9632	5.9924	2.8142

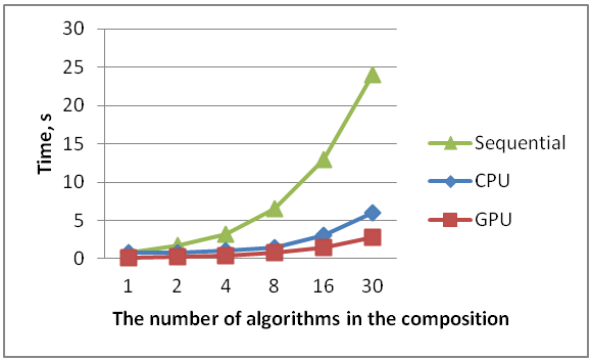

Fig. 2. Visualization comparing the execution time of sequential and parallel algorithms (using CPU and GPU) with variation in the number of algorithms in the composition

Figure 2 and Table 2 shows how rapidly the program execution time decreases when using GPU-based parallelization. To compare the time spent working with the CPU and GPU, we used the best results achieved on the CPU and still got a fairly significant acceleration on the GPU. This confirms that the use of GPUs in parallelization is very efficient and can provide impressive results. In this case, with 24 seconds of sequential execution, the graphics processor provided the ability to speed up the program to 3 seconds, which is a significant improvement.

Accuracy of the model

Now let's evaluate the quality of the model. For a more accurate estimate, we use two metrics – Accuracy and F1-score:

- Accuracy – an indicator that describes the overall accuracy of model predictions for all classes.
- F1-score is the harmonic mean value of Precision and Recall metrics, normalized between 0 and 1. If F1 score = 1, it indicates an ideal balance.

Table 3

Values of metrics Accuracy and F1-score for different number of algorithms studied parallelly

Metrics	K					
	1	2	4	8	16	30
Accuracy	0.986	0.987	0.991	0.925	0.992	0.998
F1-score	0.711	0.778	0.795	0.838	0.873	0.881

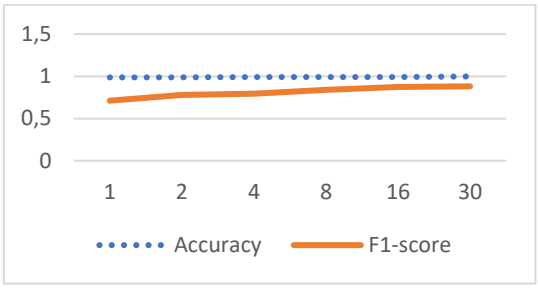

Fig. 3. Visualization of Accuracy and F1-score metrics depending on the number of algorithms

Analyzing Figure 3 and Table 3, we conclude that the accuracy is very high even when using only one algorithm but still increases slightly with an increasing number of algorithms. The value of the F1-score metric is smaller than the accuracy. Still, here we see more clearly the influence of the number of algorithms on the model's accuracy – the more algorithms in the composition, the higher the value of the metric.

Thus, the use of a parallel algorithm for combining Gradient Boosting with Begging tends to increase the accuracy of the constructed model. And the accuracy results at different values of 10 fold cross validation of the proposed algorithm were given in Table 4.

Table 4

Cross Validation Results

Fold	Loss	Accuracy(%)
1	0.015	99.685
2	0.016	99.789
3	0.011	99.899
4	0.019	98.987
5	0.013	99.843
6	0.091	97.345
7	0.026	99.341
8	0.051	98.562
9	0.016	99.876
10	0.065	97.560
Average	0.0323	99.0887

According to the results obtained (see Table 4), the proposed method reached 99.09% accuracy with 10 fold cross validation and also showed higher performance with an accuracy rate of 99.80%, although the percentage split (75:25).

Discussion of research results

Now we calculate the experimental indicators of acceleration and efficiency of parallel algorithms with different numbers of boosting algorithms in the bagging composition.

To calculate these indicators of acceleration and efficiency, we will use the following formulas: $S_p(n) = T_1(n)/T_p(n)$, $E_p(n) = S_p(n)/p$, respectively, where $T_1(n)$ is the time complexity of the sequential execution of the algorithm, $T_p(n)$ is the time complexity of the parallel execution of the algorithm for p processors (threads).

Table 5

Indicators of acceleration of the parallel algorithm with different number of threads and variation in the number of algorithms in the composition (CPU)

K	Number of threads			
	2	4	8	16
1	1.0164	1.0069	1.0583	0.9615
2	0.5358	1.9693	1.8658	2.0095
4	1.7794	3.2206	2.8030	2.9603
8	1.9537	3.4683	4.3191	4.0876
16	1.9663	3.2201	5.6927	5.1467
30	1.9819	3.5280	6.9041	6.9989

Table 5 shows the acceleration rates by parallel execution for different numbers of threads and algorithms in the composition when working on the CPU.

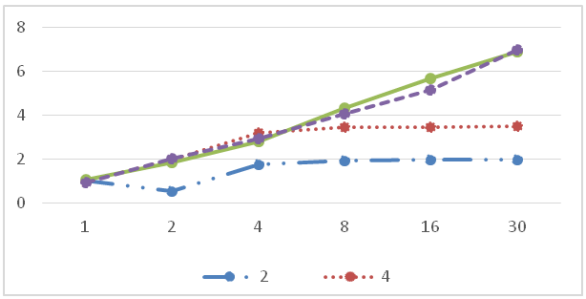

Fig. 4. Visualization of acceleration values for different number of threads and different number of algorithms in the composition (CPU)

From Figures 4, we can see that the value of the acceleration index increases with an increasing number of threads and increasing the number of algorithms in the composition. With a few algorithms, the results are ambiguous and do not reflect a stable and significant increase in acceleration. When a small number of algorithms are used, more time is allocated to the distribution of data between streams, i.e., to parallelization, than to the operation of the algorithms themselves. In addition, the acceleration goes to the number of streams used, which is in line with the basic idea of the acceleration rate. The highest acceleration was recorded when using 16 threads and 30 boosting algorithms in the bagging composition.

Table 6

Indicators of the efficiency of the parallel algorithm with different number of threads and variations in the number of algorithms in the composition

K	Number of threads			
	2	4	8	16
1	0.5082	0.2517	0.1323	0.1202
2	0.2679	0.4923	0.2332	0.2512
4	0.8897	0.8052	0.3504	0.3700
8	0.9769	0.8671	0.5399	0.5112
16	0.9832	0.8050	0.7116	0.6433
30	0.9911	0.8820	0.8630	0.8749

Table 6 shows the performance indicators using parallel execution for different numbers of threads and algorithms in the composition when working on the CPU.

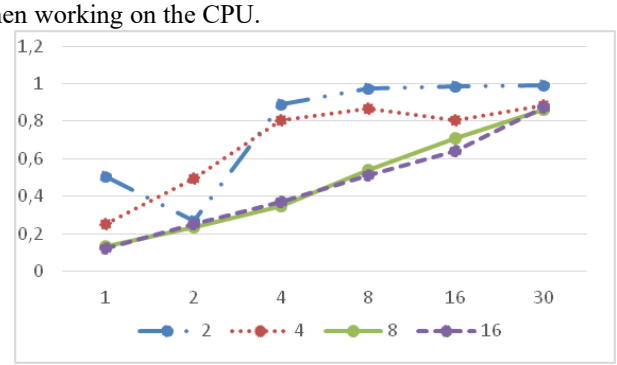

Fig. 5. Visualization of performance indicators for different number of threads and different number of algorithms in the composition

Analyzing Table 6 and Figures 5, we can say that the efficiency decreases with an increasing number of threads in contrast to the acceleration. This decrease in efficiency is due to an increase in the load on the system when

calling more threads. However, with the increase in the number of algorithms, the efficiency also increases and, at the same time, goes to 1, which indicates the quality of the parallelization method.

Let's compare the work of parallel execution of the program using the CPU and GPU.

Table 7

Acceleration rates when running a program parallelly using the CPU and GPU

K	CPU	GPU
1	1.0583	9.1641
2	1.8658	6.9279
4	2.8030	8.0452
8	4.3191	8.3861
16	5.6927	8.6379
30	6.9041	9.5151

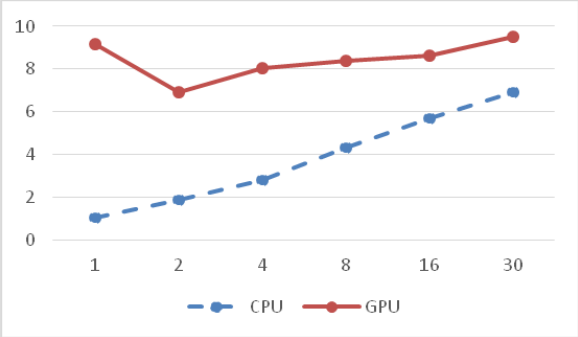

Fig. 6. Graphs of acceleration when using CPU and GPU with different number of algorithms in the composition

From Table 7 and Figure 6 we see that as the number of algorithms in the composition increases, the value of acceleration increases both when working on the CPU and the GPU, but the use of GPU gives higher values of the acceleration index, which indicates the high efficiency of CUDA technology.

Therefore, the results of experiments showed that the parallel implementation of the algorithm is most effective when using graphics processors based on CUDA technology.

Conclusions

Modern technologies can improve the standard of living of humanity in various fields, and medicine is no exception. The paper considered the relevance of the research topic: the use of data mining methods for diagnosing the disease in a patient on a set of indicators, such as symptoms, test results, and more.

A pre-processed Framingham dataset (with the number of patients equal to 10 000) was used for the study. The choice of this data set is primarily related to the need to test the proposed parallel algorithm on big data sets and obtain the best performance indicators. A search and analysis of significant features and patterns between different factors influencing the disease were conducted.

In addition, this paper proposed a parallel Gradient Boosting algorithm with Bagging to improve accuracy in predicting disease risk and significantly speed up execution time by using a multi-core architecture of modern CPUs and GPUs. So, the accuracy of the model with 10 cross validation is equal 99.09%, and for percentage split (75:25) – 99.80%. Parallelization is performed using two technologies – a pool of threads through the Python ThreadPoolExecutor utility on the CPU and CUDA on the GPU. High rates of acceleration and efficiency were achieved. Thus, with eight threads, an acceleration of approximately seven times is obtained, which indicates the reliability of the obtained results and the possibility of significant improvement of this indicator by choosing the architecture of a computer with more cores. The latter is especially relevant in modern trends in multi-core architecture. When using CUDA technology on GPUs, the acceleration is approximately 9.5 times. As for efficiency indicators, with the increase in the number of algorithms, the efficiency goes to 1, indicating the parallelization method's quality.

After analyzing the results, we concluded that CUDA works much more efficiently and prevails over the pool of threads for the selected method and data set. This is because GPUs are designed with thousands of processor cores running simultaneously and thus provide massive parallelism when each core is focused on efficient computing. Another significant result of the study is the achievement of better algorithm accuracy by increasing the number of algorithms in the composition. Using more algorithms that are learned and whose values are then averaged, we get even more accuracy values. Therefore this is a great opportunity to increase the efficiency of the model.

So, using several training algorithms to get the best forecasting efficiency, namely combining Boosting with Begging, is a great solution. This ensemble allows using different methods of accelerating and improving algorithms, which is very important today to make real-time decisions.

References

1. Buch V., Ahmed I., Maruthappu M. Artificial intelligence in medicine: current trends and future possibilities. *The British journal of general practice: the journal of the Royal College of General Practitioners*, 2018; 68(668):143–144.
2. Briganti G., Le Moine O. Artificial Intelligence in Medicine: Today and Tomorrow. *Frontiers in Medicine.* 2020; 7(27):1-6.
3. Chan Y.K., Chen Y.F., Pham T., Chang W., Hsieh M.Y. Artificial Intelligence in Medical Applications. *J Healthc Eng.*, 2018 Jul 15;2018:4827875.
4. Richens J.G., Lee C.M. & Johri S. Improving the accuracy of medical diagnosis with causal machine learning. *Nat Commun*, 2020, 11(3923):1-9.
5. Chary M., Parikh S., Manini A.F., et al. A Review of Natural Language Processing in Medical Education. West J. Emerg. Med. 2019, 20(1):78-86.
6. Le Glaz A, Haralambous Y, Kim-Dufor DH, Lenca P, Billot R, Ryan TC, Marsh J, DeVylder J, Walter M, Berrouiguet S, Lemey C. Machine Learning and Natural Language Processing in Mental Health: Systematic Review. *J. Med. Internet Res.*, 2021 May 4;23(5):e15708.
7. Dash S., Shakyawar S.K., Sharma M., et al. Big data in healthcare: management, analysis and future prospects. *J. Big Data*, 2019, 6(54):1-25.
8. Melnykova N., Shakhovska N., Gregušml M., Melnykov V. Using big data for formalization the patient's personalized data. *Paper presented at the Procedia Computer Science*, 2019, 155:624-629.
9. Mochurad L., Hladun Ya.. Modeling of Psychomotor Reactions of a Person Based on Modification of the Tapping Test. *International Journal of Computing*, 2021, 20(2):190-200.
10. Izonin I., Trostianchyn A., Duriagina Z., et al. The combined use of the wiener polynomial and SVM for material classification task in medical implants production. *International Journal of Intelligent Systems and Applications*, 2018, 10 (9):40-47.
11. Colás A., Vigil L., Vargas B., et al. Detrended Fluctuation Analysis in the prediction of type 2 diabetes mellitus in patients at risk: Model optimization and comparison with other metrics. *PLoS ONE*, 2019, 14(12):1-15.
12. Revathi Addala, Chandra Sekhar Vasamsetty. Diabetes Prediction with improved Data Preprocessing and using Linear Support Vector Classifier. *International Journal of All Research Education and Scientific Methods (IJARESM)*, 2021, 9(6): 33-40.
13. Alsuliman T., Humaidann D., Sliman L. Machine learning and artificial intelligence in the service of medicine: Necessity or potentiality? *Current Research in Translational Medicine*, 2020, 68(4):245-251.
14. Mukherjee S., Yadav G., Kumar R. Recent trends in stem cell-based therapies and applications of artificial intelligence in regenerative medicine. *World journal of stem cells*, 2021, 13(6):521–541.
15. Hussain A., Choi H.E., Kim H.J., et al. Forecast the Exacerbation in Patients of Chronic Obstructive Pulmonary Disease with Clinical Indicators Using Machine Learning Techniques. *Diagnostics (Basel, Switzerland)*, 2021, 11(5):829.
16. Melnykova N., Shakhovska N., Gregus M., et. al. Data-driven analytics for personalized medical decision making. *Mathematics*, 2020, 8(8):1211.
17. Taser P.Y. Application of Bagging and Boosting Approaches Using Decision Tree-Based Algorithms in Diabetes Risk Prediction. *Proceedings of the 7th International Management Information Systems Conference*, 2021, 74(1):6.
18. Diabetes Database. Available online: https://www.kaggle.com/uciml/pima-indians-diabetes-database (accessed on 9 April 2023).
19. Dataset Framingham. Available online: https://www.researchgate.net/publication/ 366485790_Diabetes (accessed on 9 April 2023).
20. Sharma, T., Shah, M. A comprehensive review of machine learning techniques on diabetes detection. *Vis. Comput. Ind. Biomed*., 2021, 4(30):1-16.
21. Beschi R., Anitha R., Sujatha R., et al. Diabetics Prediction using Gradient Boosted Classifier. *International Journal of Engineering and Advanced Technology (IJEAT),* 2019, 9(1): 3181-3183.
22. Swapna G., Vinayakumar R. and Soman K. Diabetes detection using deep learning algorithms. ICT Express. 2018, 4(4): 243-246.
23. Peng B., et al. HarpGBDT: Optimizing Gradient Boosting Decision Tree for Parallel Efficiency. 2019 *IEEE International Conference on Cluster Computing (CLUSTER),* 2019:1-11.
24. Yu C., Skillicorn D.B. Parallelizing boosting and bagging. Queen's University, Kingston, Canada, *Tech. Rep.*; 2001, 22 p.
25. Arsov N., Pavlovski M., Basnarkov L., Kocarev L. Generating highly accurate prediction hypotheses through collaborative ensemble learning. *Sci Rep.,* 2017, 7:44649.
26. Yurttakal A.H., Baş H. Possibility Prediction Of Diabetes Mellitus At Early Stage Via Stacked Ensemble Deep Neural Network. *Afyon Kocatepe Üniversitesi Fen Ve Mühendislik Bilimleri Dergisi*, 2021, 21(4):812-819.
27. González S., García S., Del Ser J., Rokach L., Herrera F. A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities, *Information Fusion*, 2020, 64:205-237.
28. Ensemble methods: bagging, boosting and stacking. Available from: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205 (accessed on 9 April 2023).
29. Mochurad L, Hladun Y, Tkachenko R. An Obstacle-Finding Approach for Autonomous Mobile Robots Using 2D LiDAR Data. *Big Data and Cognitive Computing*, 2023; 7(1):43.
30. Mochurad, L., Kryvinska, N. Parallelization of Finding the Current Coordinates of the Lidar Based on the Genetic Algorithm and OpenMP Technology. *Symmetry*, 2021, 13(666).
31. Felipe A. Quezada, Cristobal A. Navarro, Miguel Romero, Cristhian Aguilera. Modeling GPU Dynamic Parallelism for self similar density workloads. *Future Generation Computer Systems*, 2023, 145:239-253.
32. Shi Yu et al. Quantized Training of Gradient Boosting Decision Trees. *36th Conference on Neural Information Processing Systems (NeurIPS 2022)*, 2022:1-24, https://arxiv.org/pdf/2207.09682.pdf.

Lesia Mochurad Леся Мочурад	PhD, Associate Professor of Department of Artificial Intelligence, Lviv Polytechnic National University, Lviv, Ukraine, e-mail: lesia.i.mochurad@lpnu.ua. orsid.org/0000-0002-4957-1512, Scopus Author ID: 57210286606, ResearcherID: AAZ-9150-2020	кандидат технічних наук, доцент, доцент кафедри систем штучного інтелекту національного університету "Львівська політехніка", Львів, Україна.