JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 21 (2021) 279–286
Received: 29 August 2021
Accepted: 30 September 2021

Comparison of classical machine learning algorithms in the task of handwritten digits classification

Porównanie klasycznych algorytmów uczenia maszynowego w zadaniu klasyfikacji liczb pisanych odręcznie

Oleksandr Voloshchenko*, Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The purpose of this paper is to compare classical machine learning algorithms for handwritten number classification. The following algorithms were chosen for comparison: Logistic Regression, SVM, Decision Tree, Random Forest and k-NN. MNIST handwritten digit database is used in the task of training and testing the above algorithms. The dataset consists of 70,000 images of numbers from 0 to 9. The algorithms are compared considering such criteria as the learning speed, prediction construction speed, host machine load, and classification accuracy. Each algorithm went through the training and testing phases 100 times, with the desired metrics retained at each iteration. The results were averaged to reach the reliable outcomes.

Keywords: machine learning; classification; MNIST; classical algorithms

Streszczenie
Celem niniejszej pracy jest porównanie klasycznych algorytmów uczenia maszynowego do klasyfikacji liczb pisanych odręcznie. Do porównania wybrano następujące algorytmy: Logistic Regression, SVM, Decision Tree, Random Forest oraz k-NN. Do szkolenia i testowania powyższych algorytmów wykorzystano zbiór danych MNIST. Zbiór danych składa się z 70 000 obrazów cyfr od 0 do 9. Algorytmy porównywane są z uwzględnieniem takich kryteriów jak szybkość uczenia, szybkość budowania predykcji, obciążenie maszyny głównej oraz dokładność klasyfikacji. Każdy algorytm przeszedł przez fazy szkolenia i testowania 100 razy, z zachowaniem pożądanych metryk przy każdej iteracji. Wyniki zostały uśrednione w celu uzyskania wiarygodnych rezultatów.

Słowa kluczowe: uczenie maszynowe; klasyfikacja; MNIST; algorytmy klasyczne

*Corresponding author
Email address: **oleksandr.voloshchenko@pollub.edu.pl** (O. Voloshchenko)

1. Introduction

Machine learning is one of the most popular technologies these days, evolving incredibly fast and having a huge impact on the world today. The definition of machine learning dates back to 1959, when Arthur Samuel, an American pioneer in machine learning and artificial intelligence, said that machine learning is a field of learning that allows computers to learn without explicit programming [1]. In turn, in 1997, American professor Thomas Mitchell gave a more modern definition. He said that machine learning is when a program is told to learn from experience E with respect to some class of tasks T that measures performance P, if its performance on tasks T, which is measured as P, improves with each experience E.

This paper considers one of the main tasks of machine learning – the classification, and more specifically, the task of classifying handwritten digits.

Classification is one of the problems of machine learning, which refers to the supervised learning approach. Supervised learning is an approach in machine learning in which there is a set of data and expected results for that data, and the goal is to find a dependency rule between input and output parameters. In turn, image classification is a fairly commonly used task. For example, in classifying crops using satellite images, for agricultural purposes [2], determining benign and malignant tumors in medicine [3], gesture languages [4], in manufacturing, and in other modern tasks [5-8].

One of the problems of classification is the problem of classifying handwritten numbers [9-15]. The oldest papers and their conference discussions were held in 1994-1995. One of the examples of such works is *Comparison of classifier methods: a case study in handwritten digit recognition* [10], written by a large group of authoritative scientists in the field of machine learning. Despite the fact that the first mentions are quite old – the problem continues to be studied even now. For example, the paper [14] gives examples of using such recognition on documents, photos, or, for example, text analysis after touch screen input. The article compares three common machine learning algorithms (SVM, k-NN, NN) for text classification after optical character recognition (OCR). Or, for example, the paper [13] explores the capabilities of SVM, k-NN and neural network tools. The recognition time, error rate, number of misclassified images, and computation time were studied. In general, the following algorithms dominate in the literature among classical algorithms, those that do not rely on neural networks: Logistic Regression,

Support Vector Machine (SVM), k-Nearest Neighbours (k-NN) classifier, Decision Tree and Random Forest.

In this paper, the handwritten digit classification problem is solved using classical machine learning algorithms described above. These algorithms were chosen because they are among the most popular classification algorithms and are fairly common in the literature on such topics.

2. Purpose of the experiment

The purpose of this paper is to compare classical machine learning algorithms in the handwritten number classification problem. The experiment covers the comparison of several machine learning algorithms, such as logistic regression, support vector machine (SVM), k-nearest neighbor (k-NN) classifier, decision tree and random forest. The MNIST dataset was used in the analysis. The thesis defined for this paper is as follows: all the investigated algorithms will show different results in the handwritten digit classification problem.

2.1. Accuracy part

The primary purpose of this article is to compare the above algorithms in terms of overall accuracy and individual number recognition accuracy. This part of the work involves comparing algorithms in terms of the accuracy of handwritten number classification classes and the accuracy of classification in the case of each individual class.

The following research questions are defined for this part of the experiment:

- Which algorithm more accurately identifies classes of handwritten digits?
- Which numbers are better or worse defined for each algorithm?

2.2. Performance part

In addition to accuracy during the experiment, additional algorithm parameters, such as performance metrics, can also be explored. The following research questions are defined for the performance part of the experiment:

- Which algorithm learns faster?
- Which algorithm takes longer to learn?
- Which algorithm makes predictions faster?
- Which algorithm makes the prediction slower?
- Which algorithm requires more computer resources to learn?

3. Experiment description

To compare the algorithms, it is necessary to provide a plan for testing all of the mentioned algorithms under the study. All selected algorithms will be compared in the handwritten number classification task. MNIST handwritten number dataset was chosen as a ready-made dataset. This set consists of 70,000 pictures of numbers from 0 to 9, 28 by 28 pixels (a total of 784 pixels per picture). The parameters by which each of the algorithms will be evaluated are defined. These parameters were chosen so that the values obtained would help to answer the questions presented in section 2. In the case of the accuracy part (section 2.1), the following metrics were determined:

- accuracy – accuracy of algorithms when classifying handwritten numbers, which is determined as the ratio of successful classifications to the total number of tests;
- accuracy within numbers classes – accuracy of algorithms in determining each individual class, where classes are numbers from 0 to 9. Metric shows the percentage of correctly recognized numbers of a particular class (from 0 to 9) out of the total amount of numbers of that class in the test data set.

In the case of the performance part of the article (section 2.2), the following metrics were defined for the study:

- learning time – amount of time it takes for the algorithm to learn on the training data set;
- prediction time – amount of time needed for the algorithm to build the prediction for the whole test dataset;
- CPU load – percentage of CPU load during the processing of the training data set;
- RAM load – amount of used MiB RAM while processing the training dataset;

All of the above parameters must be measured during the execution of the overall plan of the experiment. This plan is as follows:

1. Randomly divide 70,000 handwritten number pictures into training and test datasets.
2. Start measuring the training parameters.
3. Train the learning algorithm on the training dataset.
4. End of training parameters measurement.
5. Beginning of measurement of the test parameters.
6. Testing of the trained algorithm on the test dataset.
7. End of the testing parameters measurement.
8. Saving of all measured parameters.
9. Repeat steps 1-8 N-number of times.
10. Results analysis.

Following this plan, steps 2 and 4 obtain the values of the parameters related to the training of the model, namely learning time, CPU load and RAM load. In steps 5 and 7 the data of the parameters calculated during model testing are obtained, namely prediction time, accuracy and accuracy within numbers classes. In step 8, the parameters obtained during iteration are saved for further analysis. To improve the accuracy of the obtained results, each algorithm went through the steps of the above plan 100 times, and the final results were averaged.

At each iteration, the set of pictures described above was randomly divided into a training and a test dataset according to the recommendations of its creators. The training dataset at each iteration contained 60,000 pictures (85.7% of the total dataset), while the test dataset contained 10,000 pictures (14.3% of the total dataset).

For the experiment was used the python language and the library scikit-learn, which contains ready-made implementations of the studied algorithms. The numerical values of the pixels of the pictures were taken as the parameters of the models. This means that each input

algorithm had 784 numeric parameters on the basis of which it must classify the number in the picture.

The computer used for the experiment is an Ubuntu 18.04 system, a 2.8GHz Intel Core i7-7700HQ processor and 16GB of DDR4 RAM. During the experiment the algorithms were run with a CPU limit of 1 thread.

4. Algorithms

4.1. Logistic Regression

Logistic regression is a statistical model that uses a logistic function to model the relationship between an output variable and input parameters. Binary logistic regression assumes answers at the model output as 0 or 1. Logistic regression can be used to estimate the probability of occurrence of an event for a particular test.

The logistic regression problem uses a linear regression function for estimating prediction. However, the linear regression equation does not fit the definition of logistic regression, since the model's responses may differ from 0 and 1. Some kind of transformation, also called a logit-transformation, must be performed to solve the problem. This transformation looks like this (1):

$$P = \frac{1}{1 + e^{-y}} \tag{1}$$

where P is the probability of occurrence of the event of interest, e is the Euler number, and y is the standard regression equation.

During the experiment, the *LogisticRegression* class from the *linear_model* module of the used library was used as a logistic regression in the experiment. Since the algorithm needs to classify many classes, in this case the algorithm with a multinomial parameter was used. There were also used parameters for L2 regularization and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimizing algorithm, which fit multiclass classification, are quite robust, and together should converge faster. When it comes to the number of iterations for convergence for the optimizing algorithm, a value of 100 iterations was chosen.

4.2. Decision Tree

Decision trees are a method for data analysis and predictive analytics. It is clear from the name that the tool is a hierarchical tree structure in which nodes are rules and leaves are decision points.

The rules are generated automatically by training the algorithm on a test dataset. These nodes test whether an example matches a given rule on a particular attribute. As a result of the check, the set of examples in a node is split into two subsets, one of which contains examples that satisfy the rule and one that does not. The sets then descend the tree to the next nodes, where further conditions are checked. This process is repeated until some stopping condition is satisfied. The last node performs no more checks, such a node is called a tree leaf. The leaf defines a solution for each example it contains. However, unlike a node, a leaf does not contain a rule, but a subset of objects that satisfy all branch rules.

The attribute that is used for partitioning in a node must partition the set so that the resulting subsets contain objects with the same class labels, or are as close to this result as possible. In the experiment, the Gini index, which is calculated by formula (2), is used to solve this problem:

$$Gini(Q) = 1 - \sum_{i=1}^{n} p_i^2 \tag{2}$$

where Q – the final data set, n – the number of classes in that set, p_i – probability of the i-th class. This index takes values from 0 to 1. If the index takes the value 0, then all elements of the dataset Q belong to one class, and 1 if all classes are equally likely. Obviously, the smaller the index value, the better the partition will be.

After the rules are formed on the branches of the tree on the basis of the training dataset – the branch pruning stage is performed. This approach determines the accuracy and error of the full tree. After evaluation, leaves and nodes are removed from the tree, the removal of which will not lead to a significant loss of accuracy and increase the error.

During the experiment, the *DecisionTreeClassifier* class from the *tree* module was chosen as the Decision Tree algorithm. The Gini index was used as a criterion for evaluating data partitioning in the tree, and for the partitioning itself all the parameters of the input objects were taken into account. The classification tree was split until the outermost nodes (leaves) had objects of the same class, or until the number of objects in the leaf was equal to 1. The number of leaves was not limited.

4.3. Random Forest

This model is an improvement of the previous one – the decision tree. The improvement is made by the fact that the random forest consists of an ensemble of decision trees [16].

During training, each tree is trained on random data sets, each of which is taken from the main training set. These sets are generated using a statistical method called bootstrapping. Although each tree may be highly variable with respect to a particular training dataset, training trees on different sample sets reduces the overall variability of the forest without sacrificing accuracy.

The random forest result is derived by averaging the predictions obtained from each tree in the ensemble of decision trees. This bootstrapping approach for each tree, and after averaging, is called bagging.

During the experiment, the *RandomForestClassifier* class from the *ensemble* module of the library was used as Random Forest algorithm. The used forest had 100 tree units, which were built using bootstrap samples. The trees used the same parameters as the Decision Tree algorithm above. This means that each tree used a Gini index to evaluate the quality of the separation. All input object parameters were used in the node for splitting. The tree was not limited in growth and grew as long as the leaves did not consist of objects of the same class, or as long as the leaf did not count 1 element.

4.4. Support Vector Machine

Support vector machine is a supervised machine learning algorithm. It is used for both classification and regression tasks. In this algorithm, each sample of the training dataset is represented as a point in n-dimensional space, where n is the number of features of an element in the dataset. The task of the algorithm is to construct a hyperplane that would optimally divide by classes the elements in this space. This hyperplane is constructed so that the distance from it to the elements of the classes was the largest. In other words, the principle of the algorithm is to increase the gap between the dividing hyperplane and the vectors of classes that are closest to it. These vectors are called support vectors. Obviously, the best hyperplane is the one with the largest gap.

During the experiment, the *SVC* class from the *svm* module of the used library was used as Support Vector Machine. Multi-class classification in the algorithm is supported by the one-vs-one scheme. As a kernel, the Radial Basis Function was used with an adjusting parameter with value 1 and with value gamma, which was calculated using the (3) formula:

$$\gamma = \frac{1}{n\_features \times \sigma^2} \quad (3)$$

where $n\_features$ is the number of model parameters, which in the case of this experiment is the number 784, and σ^2 – dispersion of test parameters values. The average value of the γ parameter during the experiment was 2.06.

4.5. k – Nearest Neighbors

The algorithm works according to the following principle: the object under study gets the class that most of its neighbors have in space. During training, the algorithm remembers the feature vectors of incoming observations and their corresponding class labels. At the classification stage, k nearest neighbors that already have a class are determined for the example under study.

In the simplest case, a class is defined by simply choosing the most frequent class among k examples, but this approach is not always successful. For example, in cases where the frequency of occurrence of all classes is the same. It is also important that not all neighbors are equally significant for the studied example. The usual case is simple unweighted voting. In this strategy, all k neighbors have the same weight regardless of their distance to the examined instance.

However, it would be reasonable to assume that the further away the sample is, the less influence it has. To do this, the model includes a weighting of the parameters depending on their distance to the sample under study. This method is called weighted voting.

Also, an important aspect of the algorithm is the choice of coefficient k. If k is small enough, there is an overfitting phenomenon, and the decision is made on the basis of a rather small number of examples and is generally of low significance. When k is too large, too many examples from different classes are involved in the classification process, which poorly reflects the local features of the data.

During the experiment, the *KNeighborsClassifier* class from the *neighbors* module of the used library was used as the k-NN algorithm. To determine the class of an object, five of its nearest neighbors were studied. Each object in the space had the same weight.

5. Results

During the experiment described in section 3, each algorithm under study went through the training and testing stages 100 times. At each stage the parameters described in section 3 were recorded. All results were averaged and displayed in this section.

5.1. Accuracy results

When evaluating the overall accuracy of the algorithms on the test data set, the results were obtained as shown in Figure 1.

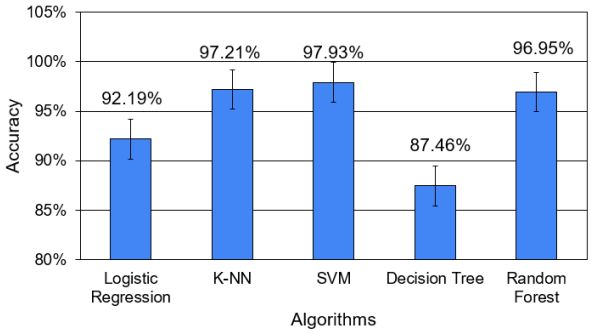

Figure 1: Accuracy of the algorithms

As can be seen from the graph, the best results were shown by the SVM, k-NN and Random Forest algorithms, the Logistic Regression algorithm comes next and the Decision Tree algorithm showed the worst performance relative to other algorithms in this task.

The next accuracy metric is accuracy within numbers classes. Along with the usual bar charts, confusion matrices are also used to display the results, which will show what errors the algorithm made during training. For each algorithm, the matrices were obtained on one of the 100 iterations of the overall experiment plan.

Figure 2 presents the results for the Logistic Regression algorithm.

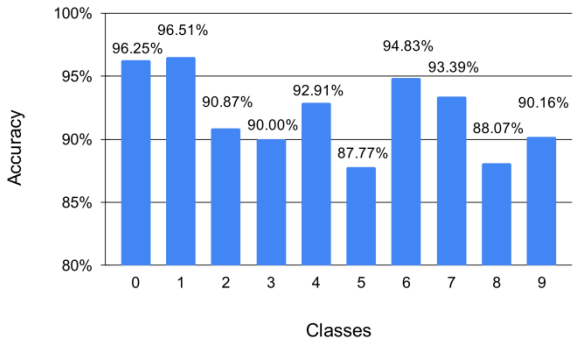

Figure 2: Accuracy of determining each individual number by Logistic Regression

The algorithm is best at determining the numbers 0 and 1, in turn determining the numbers 5 and 8 is relatively worse than other numbers. Figure 3 shows the confusion matrix of the Logistic Regression algorithm. From the matrix, it can be seen that the number 5 was mostly mistaken for the numbers 3, 6, and 8. In the case of number 8, false predictions were made in favor of numbers 1, 3, and 5.

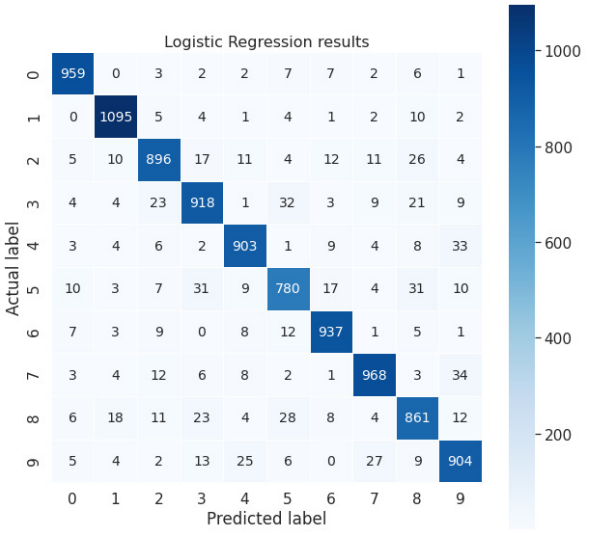

Figure 3: Logistic Regression confusion matrix

In the case of the k-NN algorithm the following results were obtained, shown in Figure 4.

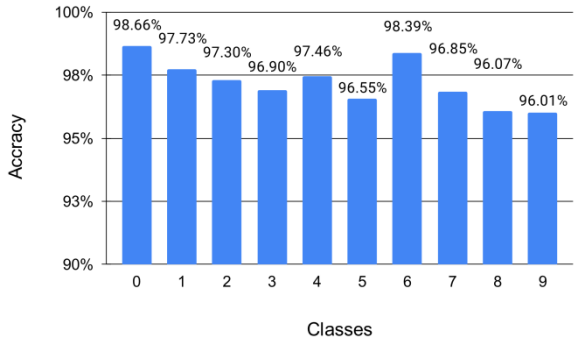

Figure 4: Accuracy of determining each individual number by k-NN

The algorithm performed well in the task of classifying numbers in general, but showed great accuracy for the numbers 0 and 6, but did a little worse for the numbers 8 and 9. The confusion matrix for the k-NN algorithm is shown in Figure 5. The matrix shows that in the case of number 8, the largest number of false predictions are assigned to numbers 1, 3, and 5. In the case of number 9 most of the errors fell on number 7 and a few on number 4.

The results of the SVM algorithm are shown in Figure 6. The algorithm did best with the numbers 0, 1 and 6, the algorithm did relatively worst with the classification of the number 9. The confusion matrix of the algorithm results is shown in Figure 7. The matrix shows that in the case of number 9, most of the false assumptions were made in favor of numbers 4 and 7.

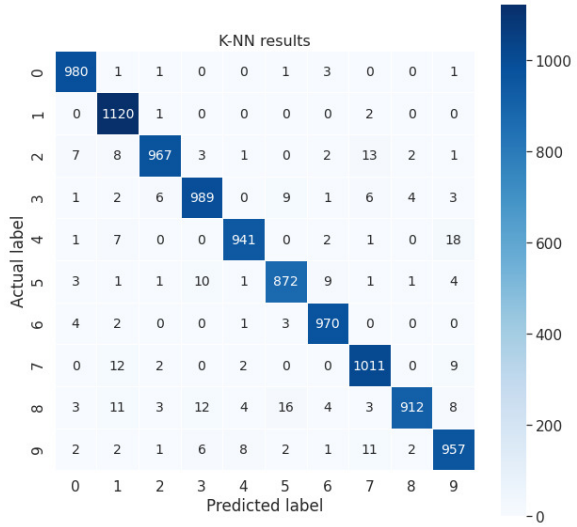

Figure 5: k-NN confusion matrix

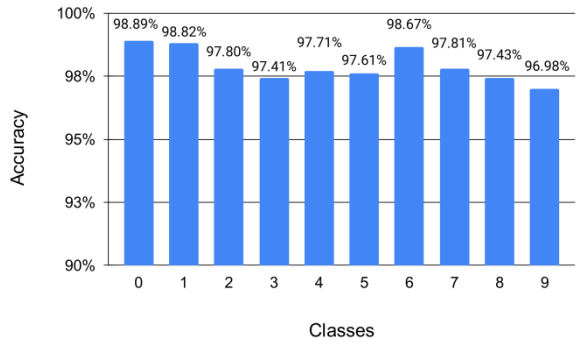

Figure 6: Accuracy of determining each individual number by SVM

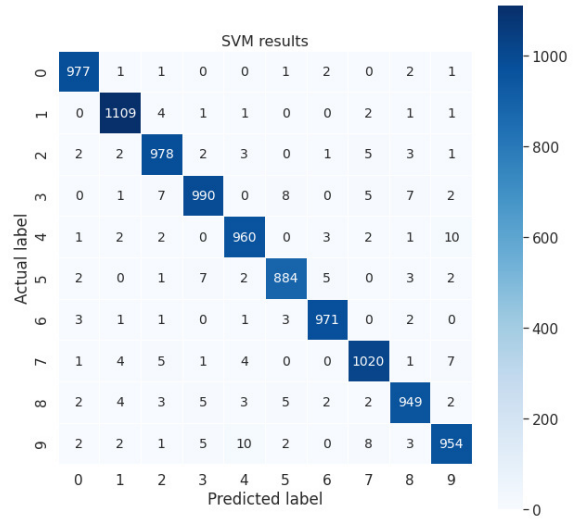

Figure 7: SVM confusion matrix

The Decision Tree algorithm got the results shown in Figure 8. The algorithm showed the best classification results for numbers 0 and 1. In the case of numbers 3, 5, 8, and 9, the algorithm showed the worst result, which, in general, is the lowest result among all the algorithms studied. The confusion matrix for the Decision Tree algorithm is shown in Figure 9. From the data in the matrix, it can be said that in the case of number 3,

a lot of false predictions were made in favor of classes 2, 5, 8, and 9. In the case of number 5, false predictions were made in favor of numbers 3, 6, 8, and 9. The number 8 was taken by the algorithm as the numbers 2, 3, 4, 5, and 9 in most false cases. In the case of number 9, most false predictions were made in favor of numbers 3, 4, 5, 7, and 8.

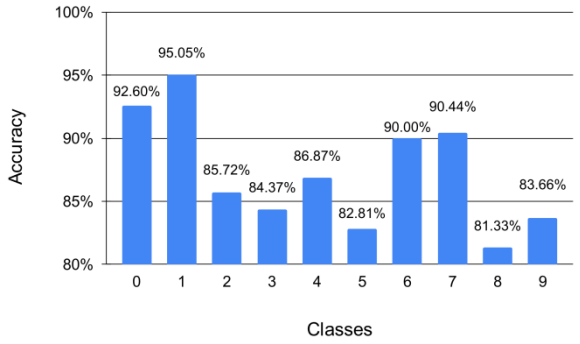

Figure 8: Accuracy of determining each individual number by Decision Tree

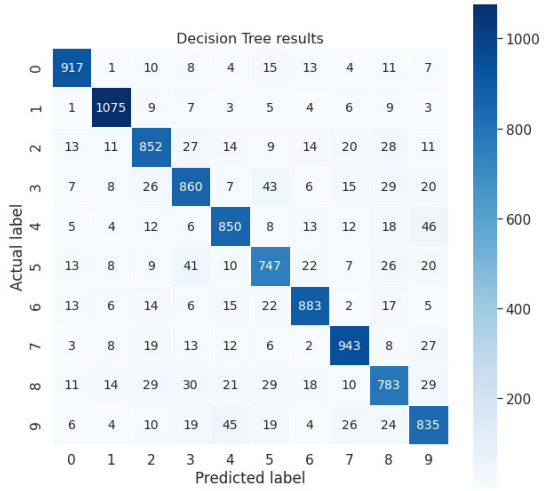

Figure 9: Decision Tree confusion matrix

In the case of the Random Forest algorithm the results are shown in Figure 10.

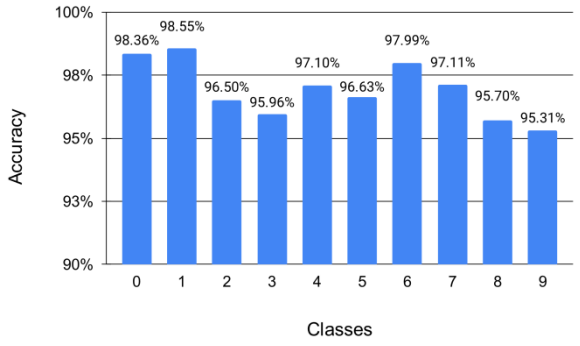

Figure 10: Accuracy of determining each individual number by Random Forest

The algorithm showed the best results in numbers 0, 1 and 6, but in the case of classes 3, 8 and 9 the results are relatively worse. The confusion matrix in the case of this algorithm is shown in Figure 11. From the matrix it can be seen that the algorithm confused the number 3 with the numbers 2 and 8. In the case of number 8, false predictions were in favor of numbers 3, 5 and 9. In the case of number 9, numbers 3, 4, and a little bit of 7 caused problems for the algorithm.

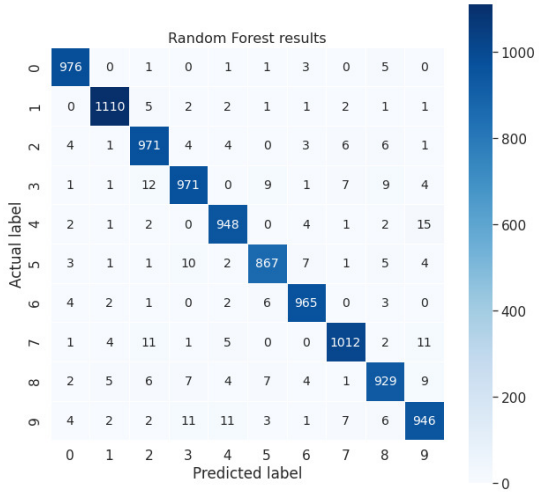

Figure 11: Random Forest confusion matrix

For a more convenient comparison of the accuracy of the class classification is built Figure 12, which can be used to compare the accuracy of determining the individual classes of numbers by algorithms.

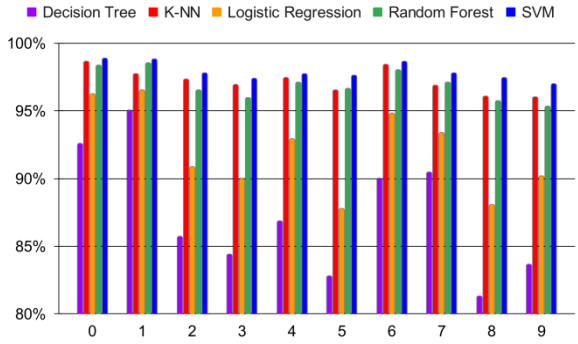

Figure 12: Accuracy of classes classification by algorithms

5.2. Performance results

In terms of learning time, the algorithms showed the following results shown in Figure 13.

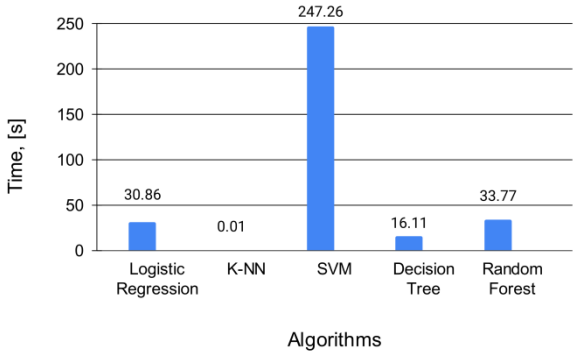

Figure 13: Learning time of algorithms

As can be seen, the fastest algorithm in terms of learning was k-NN. Such a fast result is due to the specifics of the algorithm during training, the main calculations and time costs occur during the construction of predictions, which will be shown further in this section of the article. The next fastest algorithms are Decision Tree, Logistic Regression and Random Forest. The slowest algorithm in the training phase was SVM.

The next evaluation criterion is the speed of building the predictions for the test dataset with the results shown in Figure 14. As can be seen, the fastest algorithms during the prediction stage for the test dataset were Decision Tree, Logistic Regression and Random Forest. The slowest algorithm at this stage was the SVM algorithm.

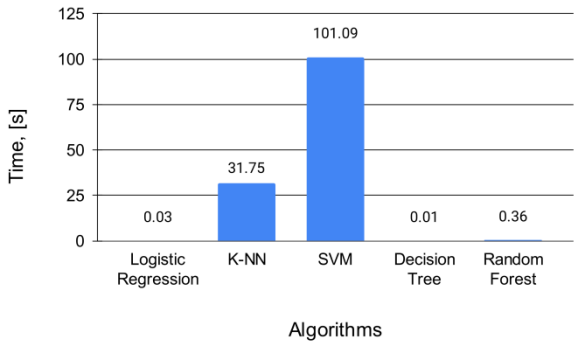

Figure 14: Algorithms prediction time

The next metric to measure was CPU load. The results are shown in Figure 15. SVM, Decision Tree, and Random Forest had the highest CPU load. The k-NN algorithm showed the lowest load.

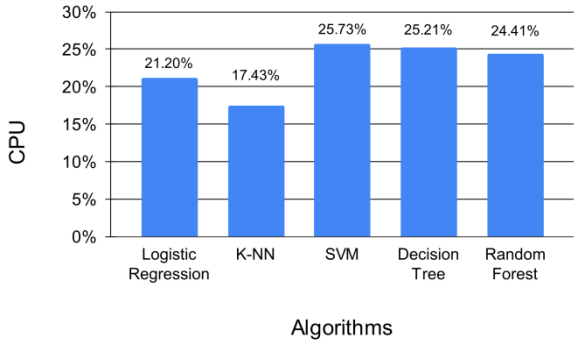

Figure 15: CPU load during learning

On the RAM usage side, the following results were obtained, shown in Figure 16. The highest level of memory usage is observed for the SVM and Logistic Regression algorithms. The Random Forest and Decision Tree algorithms are relatively average and the lowest memory usage is observed for the k-NN algorithm.

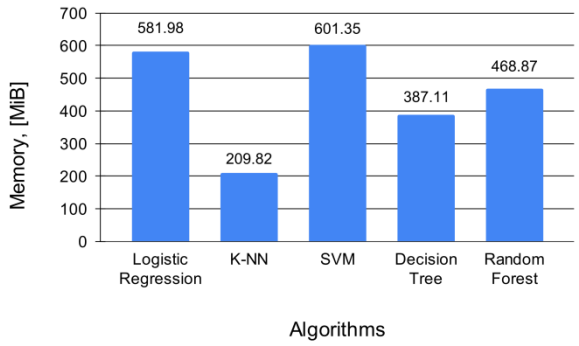

Figure 16: Memory usage during learning

6. Conclusions

As already described in section 2, the aim of the experiment was to compare classical algorithms in the problem of classifying handwritten numbers. First of all, as described in section 2.2 of this paper, we analyzed the accuracy metrics of the algorithms.

The accuracy analysis showed that the most accurate algorithms in this problem were SVM, k-NN and Random Forest algorithms with 97.93%, 97.21% and 96.95% accuracy. The results for this metric can also be seen in more detail in Figure 1.

An accuracy metric within number classes, the overall results of which are shown in Figure 12, showed that the algorithms classified numbers 0, 1, and 6 well, but had some problems classifying numbers 3, 5, 8, and 9. Based on the confusion matrices for each of the algorithms (figures 3, 5, 7, 9, and 11), we can find that in most of the incorrect cases the numbers 3, 5, 8, and 9 were confused with each other. These results are generally plausible, because the spelling of these numbers is quite similar.

The second part of the research questions (section 2.2) concerned the study of the performance metrics, which were also measured according to the experimental plan in section 3. The conclusions of the results of these metrics are described below.

The k-NN algorithm was the fastest on the training dataset with a result of 0.01 second. This speed is caused by the specifics of the algorithm, the main calculations of which take place at the prediction stage. The next fastest was the Decision Tree algorithm with a result of 16.11 seconds.

The slowest algorithm was SVM, which took an average of 247.26 seconds to train.

In terms of prediction construction time, the Decision Tree, Logistic Regression, and Random Forest algorithms were the fastest to process the test data set with corresponding results of 0.01, 0.03 and 0.36 seconds. The slowest in this aspect was the SVM algorithm with a result of 101.09 seconds.

The CPU load and memory consumption metrics show that the k-NN algorithm was the least stressful on the machine during training with a CPU load of 17.43% and used memory of 209.8 MiB. The SVM algorithm, in turn, had the highest load on the host machine among all

the algorithms studied with a CPU load of 25.73% and used memory of 601.35 MiB.

The experiment confirmed the thesis described in section 2: all the algorithms under study showed different results in the task of classifying handwritten numbers. In the future, this experiment could be conducted using deep learning tools and compare different types of neural networks in this task.

References

[1] A.L. Samuel, Some studies in machine learning using the game of checkers, IBM Journal of Research and Development 44 (2000) 206-226.

[2] J.M. Peña-Barragán, P.A. Gutiérrez, C. Martínez, J. Six, R.E. Plant, F. López-Granados, Object-Based Image Classification of Summer Crops with Machine Learning Methods, Remote Sensing 6 (2014) 5019-5041.

[3] P. Mohapatra, B. Panda, S. Swain, Enhancing histopathological breast cancer image classification using deep learning, The International Journal of Innovative Technology and Exploring Engineering 8 (2019) 2024-2032.

[4] N.H. Aung, Y.K. Thu, S.S. Maung, Feature Based Myanmar Fingerspelling Image Classification Using SIFT, SURF and BRIEF, Proceedings of the 17th International Conference on Computer Applications (ICCA 2019) (2019) 245-253.

[5] I.H. Sarker, A.S. Kayes, P. Watters, Effectiveness analysis of machine learning classification models for predicting personalized context-aware smartphone usage, Journal of Big Data 6 (2019) 1-28.

[6] R. Razavi, A. Gharipour, M. Gharipour, Depression screening using mobile phone usage metadata: a machine learning approach, Journal of the American Medical Informatics Association 27 (2020) 522-530.

[7] M. Pennacchiotti, A.-M. Popescu, A Machine Learning Approach to Twitter User Classification, Proceedings of the International AAAI Conference on Web and Social Media 5 (2021) 281-288.

[8] Y. Nieto, V. Gacía-Díaz, C. Montenegro, C.C. González, R.G. Crespo, Usage of machine learning for strategic decision making at higher educational institutions, IEEE Access 7 (2019) 75007-75017.

[9] L. Bottou, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, L.D. Jackel, Y. LeCun, U.A. Muller, E. Sackinger, P. Simard, V. Vapnik, Comparison of classifier methods: a case study in handwritten digit recognition, Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3-Conference C: Signal Processing (Cat. No. 94CH3440-5) 2 (1994) 77-82.

[10] Y. LeCun, L.D. Jackel, L. Bottou, A. Brunot, C.Cortes, J. Denker, H. Drucker, I. Guyon, U.A. Muller, E. Sackinger, P. Simard, Comparison of learning algorithms for handwritten digit recognition, International conference on artificial neural networks 60 (1995) 53-60.

[11] B. El Kessab, C. Daoui, B. Bouikhalene, R. Salouan, A Comparative Study between the Support Vectors Machines and the K-Nearest Neighbors in the Handwritten Latin Numerals Recognition, International Journal of Signal Processing, Image Processing and Pattern Recognition 8 (2015) 325-336.

[12] K. Zhao, Handwritten digit recognition and classification using machine learning, M.Sc. in Computing (Data Analytics), Technological University Dublin (2018).

[13] C. Kaensar, A comparative study on handwriting digit recognition classifier using neural network, support vector machine and k-nearest neighbor, The 9th International Conference on Computing and InformationTechnology (IC2IT2013) (2013) 155-163.

[14] N.A. Hamid, N.N. Sjarif, Handwritten recognition using SVM, KNN and neural network, arXiv preprint arXiv:1702.00723 (2017).

[15] T.A. Assegie, P.S. Nair, Handwritten digits recognition with decision tree classification: a machine learning approach, International Journal of Electrical and Computer Engineering (IJECE) 9 (2019) 4446-4451.

[16] L. Breiman, Random forests, UC Berkeley TR567 (1999).