Bio-inspired technique for improving machine learning speed and big data processing

Andronicus A Akinyelu
Department of Computer Science and Informatics
University of the Free State
Bloemfontein, South Africa
akinyeluaa@ufs.ac.za

***Abstract*— Big data analytics (BDA) is progressively becoming a popular practice implemented by many organizations, because of its potential to discover treasured insights for improved decision-making. Machine Learning (ML) algorithms are one of the effective tools used for BDA, however, their computational complexity increases with an increase in data size. Therefore, this paper introduces a boundary detection and instance selection technique for improving the speed of ML-based big data classification models. The proposed technique (called ACOISA_ML) is inspired by edge selection in ant colony optimization. ACOISA_ML is evaluated on five ML algorithms and ten large- or medium-scale datasets, and the results show that it has the potential to reduce the training speed of ML algorithms by over 94% without significantly affecting their prediction accuracy. Moreover, the results show that it reduced the storage size of big datasets by over 55% (in most cases), thus improving the speed of big data processing.**

Keywords*— *Big data analytics, Machine learning, Instance selection, Data reduction, Speed optimization.

Introduction

Big Data Analytics (BDA) is progressively becoming a popular practice that many organizations are implementing intending to obtain valued insights for enhanced decision making. The volume of data generated by numerous data sources has increased immensely. These data sources include textual content (structured, semi-structured, and unstructured), multimedia content (such as videos, images, audio, etc.), sensor networks, social media, and internet of things. Every day, the world produces about 2.5 quintillion bytes of data [1]. Moreover, in the year 2020, over 40 trillion gigabytes of data will be generated [1]. We are in an era of big data, and the rapid rate of data growth is alarming. Given this, numerous researchers are developing improved, fast, and effective techniques suitable for speeding up BDA, including feature selection techniques, instance selection techniques, sampling, and distributed computing. Leyva *et al*. [2] introduced three instance selection techniques using the local set concept [3]. The first technique (LSSM) was designed to eliminate irrelevant and overlapping instances from datasets, and the second technique (LSCO) was designed to choose a centroid from various clusters. The third technique (LSBO) was designed to select border instances from datasets. LSBO uses LSSM to eliminate irrelevant instances and then use the local set concept to choose border instances. The authors evaluated the three algorithms and they produced promising results. In another study, Carbonera and Abel introduced two instance selection techniques (LDIS [4] and XLDIS [5]) for selecting instances with high density. The techniques were designed to assess instances of various classes independently and select instances with high density in a specific neighbourhood. Furthermore, in [6], the same authors designed another data reduction algorithm (called ISDSP) based on the concept of spatial partition. They used the algorithm to split a dataset into various partitions of comparable instances and then selected representative instances from each of the partitions. The technique was tested on various datasets and it achieved satisfactory storage reduction. In [7], Rathee *et al*. solved the data reduction problem for multi-objective frameworks. They introduced a hybrid technique by combining the CHC genetic algorithm and the non-dominated sorting and replacement strategy of NSGA-II [8]. The combined technique (called MOCHC) was evaluated on different datasets and it produced good storage reduction percentage and prediction accuracy. Venkatasalam *et al*. [9] proposed an improved feature selection for BDA. They used Accelerated Flower Pollination (AFP) algorithm to select optimal features from big datasets, and the selected features were used to build fast models. Many other data reduction techniques have been introduced in the literature; however, some of them failed to achieve a balanced trade-off between storage reduction and predictive accuracy [2]. Moreover, most of them centered on feature selection and parameter optimization, while a few focused on instance selection. Nevertheless, instance selection is one of the most efficient data reduction optimization technique [10]. Therefore, this study proposes an effective instance selection technique for reducing the training speed of Machine Learning (ML)-based big data classification models. The key highlights of this study are as follows:

i. This study introduces a fast and efficient boundary detection and instance selection technique. The proposed technique is used to significantly reduce the size of big datasets (by removing irrelevant instances) and improve the training speed of ML algorithms. It is worthy to note the difference between ACOISA_ML and other ACO-based techniques. ACOISA_ML uses two algorithms to perform data reduction: ACO and *k*-NN. Unlike other techniques that use ACO to directly select instances for training, ACOISA_ML use ACO to only identify potential boundaries (not to select instances). Afterwards, it uses *k*-NN to select instances close to the identified boundary.

ii. The proposed technique is applied to five ML algorithms (Artificial Neural Network (ANN), Random Forest (RF), Naïve Bayes (NB), *k*-NN, and Logistic Regression (LR)), and ten large or medium scale datasets. The results show that the proposed technique significantly reduced the storage size of big datasets and the training speed of ML algorithms without significantly affecting their prediction accuracy or dataset quality.

I. PROPOSED TECHNIQUE

As mentioned above, this study introduces a boundary detection and instance selection technique called Ant Colony Optimization Instance Selection Algorithm for Machine Learning (ACOISA_ML). The pseudocode for ACOISA_ML is shown in Algorithm 1. Given a training dataset, D, ACOISA_ML is designed to select a reduced set d that produces the highest possible predictive accuracy, where ($d \subseteq D$). ACOISA_ML is inspired by edge selection in Ant Colony Optimization (ACO). The algorithm uses ACO to construct a pheromone matrix, where each entry in the matrix signifies the boundary information for each instance

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

in a dataset. Typically, ants move arbitrarily in search for food, and the instant a food source is identified, the ant deposits pheromone trails on the ground when travelling back to its nest. The pheromone trail guides other ants to the identified food source, thus reducing the time taken for food search. Trails with the highest pheromone concentration lead to the best food source. Inspired by this concept, ACOISA_ML searches a binary space for various boundary instances. Afterwards, ACOISA_ML selects the best boundary instance, that is, the boundary instance with the maximum pheromone value. Subsequently, the *k*-NN algorithm is used to select *k* instances close to the boundary instance. The pseudocode for ACOISA_ML is presented in Algorithm 1. ACOISA_ML is divided into three stages: initialization stage, heuristic value computation stage, and construction stage. A discussion on these stages is provided next.

A. Initialization Stage

The algorithm begins by initializing the binary search space where the ant-like agents will evolve (line 4). Furthermore, *M* ants are randomly assigned to every node in the search space, where each node represents an instance (line 5). Each node consists of different values, including pheromone and heuristic value. More information is provided in Section I.C. The dimension of the search space is $S_1 \times S_2$, where S_1 represents the number of instances and S_2 represents the number of features. Each ant moves in the search space along a chosen trail, thereby creating a solution or constructing a new trail (maybe better trail) based on the pheromone value and heuristic information of each node it passes.

B. Heuristic Value Computation Stage

The boundary instances used to build improved ML models are selected in this stage. At the beginning of the search, an initial pheromone value is assigned to each ant (line 5). Moreover, each ant decides to move to an unvisited node based on a certain probability that depends on the pheromone level and heuristic value of the node. The heuristic value for each node is computed in lines 6 – 17, as shown in the pseudocode. The heuristic value is designed to reflect the boundary information of each instance. After all the ants have completed their tour, each ant deposits an amount of pheromone on the edge of each node it visited. Moreover, evaporation takes place by reducing the number of pheromones on each edge. Finally, the node with the highest pheromone value is selected as the boundary instance. Given a training subset *s* with *I* instances and *c* classes, the heuristic value for each training $instance_i$ in subset *s* is computed as follows:

i. Using equation (1), the first step is to compute the Euclidean distance between $instance_i$ and $n-1$ instances in subset *s,* where $instance_i$ is the current instance in subset *s*, and n is the number of instances in subset *s* (line 7 – 10). After computing the Euclidean distance for $instance_i$, store the computed value in $distance\_mask_i$. Note that $distance\_mask_i$ is of dimension $n-1$.

If $x = x_1, x_2, \dots x_n$ and $y = y_1, y_2, \dots y_k$, then the Euclidean distance between two instances *x* and *y* is given as:

$$d(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_k - y_k)^2} = \sqrt{\sum_{i=1}^{k}(x_k - y_k)^2} \quad (1)$$

ii. The second step is to select the nearest neighbours of $instance_i$. Using the calculated Euclidean distances in $distance\_mask_i$ as reference points, select the k nearest neighbours of $instance_i$ using the k-NN algorithm, and save them in $neighbourhood\_list_i$ (line 11).

iii. The third step is to search $neighbourhood\_list_i$, and select the class ($class_i$) of $instance_i$. Also, select the instances belonging to the opposite class. Instances belonging to $class_i$ are taken as positive class, while instances belonging to other classes are taken as the opposite classes. For example, if there are three classes in subset *s* (*classes x, y* and z), and $instance_j$ belongs to *class y*, then all the instances (in subset *s)* belonging to *class y* are treated as the positive class, while the other instances that belong to *classes x and z* are treated as the opposite class. After selecting the instances belonging to the opposite class in $neighbourhood\_list_i$, select their corresponding Euclidean distances in $distance\_mask_i$.

iv. The fourth step is to calculate the heuristic value for $instance_i$. Calculate the sum (sum_i) of the selected Euclidean distances in step iii (line 12 – 16). Finally, save the sum in $heuristic\_value_i$. The sum is the heuristic value for $instance_i$.

v. Do steps i – iv for all the *n* instance in subset *s* until the heuristic values for the entire instances in subset *s* are computed.

vi. The final step is to select the boundary instances. That is, select the $instance_j$ with the maximum $heuristic\_value_i$ and use it as a reference point to select *k* instances close to the boundary instance. The instance with the maximum heuristic value is the instance with the maximum number of opposite classes in its neighbourhood list. If the heuristic value of $instance_i$ is very high, then it indicates that $instance_i$ has several opposite class instances in its neighbourhood list. Moreover, it indicates that the opposite classes are far from $instance_i$. The farther they are from $instance_i$, the bigger the margin. Besides, the bigger the margin, the better the performance, especially for instance-based classifiers like Support Vector Machine (SVM).

C. Construction stage

Sequel to the heuristic value computation for all the training instances, *n* ants are moved from one node to another (line 18-27). At first, an ant is arbitrarily selected and moved for *k* number of steps, where *k* is pre-defined (line 19-24). Furthermore, the same ant is moved from one node ($node\ x$) to another node ($node\ y$) in its neighbourhood list, using the probability specified in equation (2).

$$p_{x,y}^{(n)} \frac{(\tau_y^{n-1})^{\alpha}(\eta_y)^{\beta}}{\sum_{y \in \Omega_x}(\tau_y^{n-1})^{\alpha}(\eta_y)^{\beta}}, \quad (2)$$

where τ_y^{n-1} is the pheromone value of $node\ y$, Ω_x refers to the neighbourhood list of $node\ x$, η_y is the heuristic information at node y. $\alpha\ and\ \beta$ are user-defined constants. α regulates the pheromone matrix and β regulates the heuristic matrix. Each node consists of five values, including (i) initial pheromone value, which is set to a constant value (ii) current pheromone value, (iii) heuristic value, computed using the *k*-NN algorithm as previously explained (iv) position for each ant, defined by equation (2) (v) neighbourhood nodes, which contain the list of possible nodes that a given ant can go to. In addition, the current pheromone value is updated twice. The first update is done in line 26, using equation (3). This update is carried out each time an ant is moved. The second update is done after all the ants have been moved, using equation (4) (line 29).

$$\tau_y^{n-1} \begin{cases} (1-\rho) * \tau_y^{n-1} + \rho * \Delta_y^{(z)}, & if\ node\ y\ is\ currently\ visted \\ \tau_y^{n-1}, & otherwise \end{cases} \quad (3)$$

where ρ refers to the evaporation rate, $\Delta_y^{(z)}$ is the heuristic information at the z[th] node.

$$\tau^{(n)} = (1-\psi) * \tau^{(n-1)} + \psi * \tau^{(0)} \quad (4)$$

where ψ is the pheromone decay coefficient.

In line 28, the best node (with the maximum pheromone value) is selected, and in line 29, k-NN data reduction algorithm is used to select boundary instances. Boundary instances are the instances close to the best node. Finally, boundary instances are used to build fast and improved ML models.

II. EXPERIMENTS

A different set of experiments were performed to evaluate the performance of the proposed technique. The experiments are divided into two stages: data reduction stage and training stage. In the first stage, ACOISA_ML was used to remove irrelevant instances from the datasets. After data reduction, the reduced datasets (and the entire datasets) were used to train five ML algorithms, namely: ANN, RF, NB, LR, and k-NN. In this study, we refer to the models produced by the entire dataset as standard models, and the models produced by the reduced subset as hybrid models. Finally, we compare the hybrid models to the standard models based on the following performance measures: (i) the ability to preserve prediction accuracy (ii) training speed (iii) storage reduction percentage, and (iv) algorithm time (or instance selection time). Moreover, we compare the performance of ACOISA_ML to an existing instance selection algorithms, MCIS [11], also implemented in this study for comparison. Besides, the effectiveness of ACOISA_ML was further demonstrated by comparing it to five other instance selection techniques, namely MOCHC [7], LDIS [4], LSSM [2], LSBO [2], and ISDSP [6]. The authors of MOCHC evaluated it on the *k*-NN algorithm and the following datasets: Yeast, Segment, Page-blocks, Optdigit, Mushroom, Letter, and Shuttle. Moreover, LDIS, LSSM, LSBO, and ISDSP were evaluated on SVM and the following datasets: Cardiotocography, Ecoli, Heart-statlog, Ionosphere, Landsat, Letter, Optdigits, Page-blocks, Parkinson, Segment, and Wine. Hence, to ensure a fair comparison, we also implemented ACOISA_ML on SVM, *k*-NN and the same datasets before comparing its performance to the techniques.

A. Experimental Setup

As shown in Table I, the proposed technique is evaluated on ten medium or large-scale datasets. Nine of the datasets are obtained from the UCI dataset repository [12], while the USPS (US Postal Service) dataset can be obtained from Kaggle [13]. Some of the datasets were originally separated into training and test set by their providers, including Optdigit, Pendigit, USPS, Letter, Shuttle, and Landstat. Besides, in this study, the Twitter dataset was divided into training and test subsets. Twenty percent of the dataset was used for testing and the remaining 80% for training. Due to the stochastic nature of ACO, data reduction was performed on each dataset ten times (using ACOISA_ML) and the average performance for the 10 runs was calculated. However, because Mushroom, Waveform, and Pageblock datasets did not have test subset, 10-times, 10-fold cross-validation was performed for the three datasets. All the experiments were carried out on the WEKA platform [14] – a reliable open source software for ML tasks. Moreover, the experiments were performed on a laptop computer with the following specifications: Windows 10, 8GB RAM, 64-bit, Intel Core i5, 1.70GHz (4CPUs). The parameters used for the ACO were selected experimentally and reported in Tables II.

TABLE I. DATASETS

Dataset Name	Instance Size	#Features	#Classes	#Training Samples	#Test Samples
Landstat	6435	36	7	4435	2000
Letter	20000	26	2	16000	4000
Mushroom	8124	22	2	7308	816
Optdigit	5620	64	10	3823	1797
Page-blocks	5473	10	5	4923	550
Shuttle	58000	9	7	43500	14500
Twitter	140707	77	2	112566	28141
USPS	9298	256	10	7291	2007
Pentdigit	10992	16	10	7494	3498
Waveform	5000	40	3	4500	500

TABLE II. PARAMETERS USED FOR ACOISA_ML

α	β	NN	Evaporation Rate	Total Ant Movement	Decay Coefficient	Initial Heuristic Value	Iteration
1	2	8	0.1	40	0.05	0.01	10

Key: $\alpha = alpha, \beta = beta, NN = nearest\ neighbours$

B. Result and Discussion

This section reports the average prediction accuracy, storage percentage, algorithm time, and training time produced (over ten runs) by the hybrid algorithms (denoted as ACOISA_ML) and MCIS. Furthermore, this section reports the training speed improvement (in percentage) achieved by ACOISA_ML (donated as Speed Imp). The speed improvement is calculated using equation (5).

$$((\alpha - \beta)/\alpha) * 100 \quad (5)$$

where $\alpha\ and\ \beta$ refers to the training speed produced by the standard and hybrid models, respectively. Moreover, this section reports the results produced by the standard algorithms (denoted as *Standard*). Specifically, Tables III - VIII shows the results for *k*-NN, ANN, LR, RF, and NB respectively. Moreover, Table VIII shows the average algorithm time and storage percentage produced by

ACOISA_ML and MCIS. The storage reduction percentage represents the fraction of instances selected after data reduction.

As shown in Table VIII, the proposed technique significantly reduced the storage size of the evaluated datasets without meaningfully affecting the quality of the datasets. Moreover, as shown in Tables III-VIIII, the proposed technique improved the training speed of the five evaluated ML algorithms (in most cases), without significantly affecting their prediction accuracy. For example, as shown in Tables IV, the proposed technique improved the training speed of ANN (for twitter dataset) by over 94%, without significantly affecting its prediction accuracy. The speed improvement is particularly obvious in algorithms with computational complexity problems, such as ANN and SVM. Moreover, as shown in Table VIII, the proposed technique reduced the storage size of the evaluated datasets by over 55% (in most cases). The storage reduction is predominantly obvious in larger datasets, such as the Twitter dataset. These improved results show the ability of ACOISA_ML in enhancing the speed and preserving the prediction accuracy of ML algorithms. It also shows the ability of ACOISA_ML in improving the speed of big data processing. The reduced dataset and improved speed should lead to simplified, better, and fast decision making.

Furthermore, the average algorithm time (or speed) used by ACOISA_ML is in few seconds. As shown in Table VIII, ACOISA_ML produced an average algorithm speed of 144 seconds; that is, it used an average speed of 144 seconds to select relevant instances from all the datasets evaluated in this study. Specifically, for some datasets (such as Landstat and Waveform), it used less than 8 seconds to perform instance selection, while for some datasets (such as Mushroom and Pentdigit), it used less than 20 seconds. This shows the effectiveness of ACOISA_ML in dataset processing. Besides, the proposed technique was compared to MCIS, also adopted in this study for the sake of comparison. As shown in Table III - VIII, for all the algorithms, ACOISA_ML outperform MCIS in some cases and produced comparable results in other cases. Specifically, for some algorithms (such as ANN, LR, RF, and NB) and datasets, the proposed technique produced better training speed than MCIS. Besides, in the average case, the proposed technique produced better storage reduction percentage than MCIS.

In addition, the robustness of ACOISA_ML was further demonstrated by comparing it to four recent instance selection techniques, namely: LDIS, LSSM, LSBO, and ISDSP. The techniques were evaluated on SVM, hence ACOISA_ML was first evaluated on SVM before it was compared to the techniques. Table IX shows the prediction accuracy (denoted as accr) and storage reduction percentage (denoted as stor) for the four techniques. The best prediction accuracy for each dataset is underlined. As shown in Table IX and Figure 1, ACOISA_ML outperformed LSSM in prediction accuracy in 6 out of 11 datasets and outperformed LSBO in 7 out of 11 datasets. Moreover, the results show that ACOISA_ML outperformed LDIS and ISDSP in 9 out of 11 datasets. Overall, ACOISA_ML outperformed the compared techniques in most cases and simultaneously produced comparable storage reduction percentage. Another experiment was performed to compare ACOISA_ML to a recent data reduction technique, called MOCHC. The authors of MOCHC evaluated MOCHC on the k-NN algorithm; hence, ACOISA_ML was first evaluated on the *k*-NN algorithm and then compared to MOCHC. Table X and Figure 2 shows the evaluation between ACOISA_ML and MOCHC on seven datasets. As shown in the results, ACOISA_ML produced better storage reduction percentage in 6 of the 7 datasets and simultaneously produced better prediction accuracy in 4 out of 7 of the datasets evaluated. Moreover, T-test statistical analysis was performed to further evaluate the speed improving capacity of ACOISA_ML. The test was performed on five datasets and five ML algorithms. The P-values for the algorithms are reported. Besides, the P-values that are above 0.05 are underlined. As shown in Table XI, most of the P-values are over 0.05, hence we can conclude with 95% confidence level that ACOISA_ML is significantly faster, in terms of training speed than the analyzed standard algorithms.

III. CONCLUSION AND FUTURE RESEARCH

This study presents a fast and effective hybrid technique for BDA and ML speed optimization. The technique was evaluated on ten large or medium-scale datasets and five ML algorithms, and the results show that it has the potential to efficiently reduced big datasets by over 75%, thus improving the response time of BDA. Moreover, the result shows that the proposed technique can improve the training speed of ML algorithms by over 94% without significantly affecting their prediction accuracy. Besides, the hybrid technique was compared to six existing instance selection techniques, and it produced very competitive results. Besides, T-test statistical analysis was performed, and the test result shows that the proposed technique significantly improved the training speed of ML algorithms. The improved performance of the proposed technique comes with the following benefits: (i) improved speed and computational complexities for ML-based big data classification, (ii) improved dataset quality, reduced dataset size and storage requirements. The proposed technique is iterative in structure; therefore, future research can attempt developing non-iterative speed optimization approaches for ML algorithms. In addition, future research can attempt implementing data reduction operations on a distributed computer system to improve its performance (such as Hadoop).

REFERENCES

[1] U. Sivarajah, M. M. Kamal, Z. Irani, and V. Weerakkody, "Critical analysis of Big Data challenges and analytical methods," *Journal of Business Research,* vol. 70, pp. 263-286, 2017.

[2] E. Leyva, A. González, and R. Pérez, "Three new instance selection methods based on local sets: A comparative study with several approaches from a bi-objective perspective," *Pattern Recognition,* vol. 48, no. 4, pp. 1523-1537, 2015/04/01/ 2015.

[3] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," *Data mining and knowledge discovery,* vol. 6, no. 2, pp. 153-172, 2002.

[4] !!! INVALID CITATION !!! [4].

[5] J. L. Carbonera, "An Efficient Approach for Instance Selection," in *Big Data Analytics and Knowledge Discovery*, Cham, 2017, pp. 228-243.

[6] J. L. Carbonera and M. Abel, "Efficient instance selection based on spatial abstraction," in *2018 IEEE 30th International*

Conference on Tools with Artificial Intelligence (ICTAI), 2018, pp. 286-292.

[7] S. Rathee, S. Ratnoo, and J. Ahuja, "Instance Selection Using Multi-objective CHC Evolutionary Algorithm," in Information and Communication Technology for Competitive Strategies, ed: Springer, 2019, pp. 475-484.

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation,* vol. 6, no. 2, pp. 182-197, 2002.

[9] K. Venkatasalam, P. Rajendran, and M. Thangavel, "Improving the Accuracy of Feature Selection in Big Data Mining Using Accelerated Flower Pollination (AFP) Algorithm," *Journal of Medical Systems,* vol. 43, no. 4, p. 96, 2019/03/09 2019.

[10] J. A. Olvera-López, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, and J. Kittler, "A review of instance selection methods," *Artificial Intelligence Review,* vol. 34, no. 2, pp. 133-143, 2010.

[11] J. Chen, C. Zhang, X. Xue, and C.-L. Liu, "Fast instance selection for speeding up support vector machines," *Knowledge-Based Systems,* vol. 45, pp. 1-7, 2013.

[12] K. Bache and M. Lichman. (2013), "UCI machine learning repository". available at: http://archive.ics.uci.edu/ml (accessed 12-May-2017).

[13] Andrea. (2016), "Credit Card Fraud Detection". available at: https://www.kaggle.com/dalpozz/creditcardfraud (accessed 27-October-2016).

[14] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*: Morgan Kaufmann, 2016.

TABLE III. PREDICTION ACCURACY AND TRAINING TIME FOR K-NN

	Average Prediction Accuracy			**Average Training Time**			**Speed Imp (%)**
Datasets	***Standard***	***ACOISA_ML***	***MCIS***	***Standard***	***ACOISA_ML***	***MCIS***	
Landstat	90.550	86.860	86.180	0	0.001	0.001	0
Letter	95.725	90.995	91.565	0.020	0.047	0.012	0
Mushroom	100	99.910	100	0.010	0.001	0.030	90
Optdigit	97.830	93.784	96.906	0.030	0.004	0.001	86.667
Page-blocks	96.017	96.916	97.807	0.020	0.004	0	80
Shuttle	99.910	99.775	99.832	0.040	0.017	0.015	57.500
Twitter	96.091	94.694	95.519	0.060	0.015	0.124	75
USPS	95.117	93.737	93.249	0.030	0.001	0.003	96.667
Pentdigit	97.742	92.885	97.133	0.020	0.002	0.004	90
Waveform	80.240	81.800	80.072	0	0	0	0

Key: Speed Imp - Speed Improvement calculated using equation (5)

TABLE IV. PREDICTION ACCURACY AND TRAINING TIME FOR ANN

	Average Prediction Accuracy			**Average Training Time**			**Speed Imp (%)**
Datasets	***Standard***	***ACOISA_ML***	***MCIS***	***Standard***	***ACOISA_ML***	***MCIS***	
Landstat	88.500	85.165	85.150	115.160	40.811	38.620	64.561
Letter	80.975	79.390	79.615	365.310	177.991	167.691	51.276
Mushroom	98.966	99.015	100	79.440	19.277	24.556	75.734
Optdigit	96.550	93.461	94.969	278.160	157.821	101.350	43.263
Page-blocks	96.236	97.328	97.075	25.610	10.482	12.514	59.071
Shuttle	99.752	99.706	99.710	255.040	109.026	88.086	57.251
Twitter	96.411	94.7831	96.218	8859.440	485.939	3607.944	94.515
USPS	94.320	93.284	92.930	5047.810	2420.235	1879.554	52.054
Pentdigit	89.823	89.137	90.795	74.250	32.380	42.310	56.391
Waveform	83.840	85.267	86.871	50.330	11.299	15.116	77.550

TABLE V. PREDICTION ACCURACY AND TRAINING TIME FOR LR

	Average Prediction Accuracy			**Average Training Time**			**Speed Imp (%)**
Datasets	***Standard***	***ACOISA_ML***	***MCIS***	***Standard***	***ACOISA_ML***	***MCIS***	
Landstat	83.750	82.100	80.215	4.820	8.458	102.001	0
Letter	77.375	75.968	75.600	174.530	77.836	93.824	55.403
Mushroom	95.483	99.970	100	1.240	0.211	0.110	82.984
Optdigit	92.321	86.900	87.757	26.640	18.089	6.327	32.098
Page-blocks	96.455	97.408	97.320	3.020	1.793	2.535	40.629
Shuttle	96.835	96.760	96.794	1757.580	27.482	34.142	98.436
Twitter	96.556	95.393	96.580	69.840	2.902	24.882	95.845
USPS	89.536	86.871	82.995	509.160	227.328	139.309	55.352

Pentdigit	89.823	89.137	91.850	108.860	39.143	188.232	64.043
Waveform	87.080	87.517	85.557	1.100	0.206	0.297	81.273

TABLE VI. PREDICTION ACCURACY AND TRAINING TIME FOR RF

	Average Prediction Accuracy			**Average Training Time**			**Speed Imp (%)**
Datasets	***Standard***	***ACOISA_ML***	***MCIS***	***Standard***	***ACOISA_ML***	***MCIS***	
Landstat	91.050	88.085	88.330	4.790	1.212	1.287	74.697
Letter	96.175	91.913	92.280	11.290	7.671	15.303	32.055
Mushroom	100	99.950	100	1.870	0.458	0.089	75.508
Optdigit	97.385	90.384	96.188	2.600	2.289	1.305	11.961
Page-blocks	97.533	97.948	98.439	4.160	1.243	0.959	70.120
Shuttle	99.993	99.903	99.952	18.750	7.617	7.131	59.376
Twitter	96.688	95.185	96.638	275.060	6.987	86.103	97.460
USPS	93.373	92.312	91.928	19.060	8.839	4.449	53.625
Pentdigit	96.598	90.792	95.360	4.460	1.600	1.783	64.126
Waveform	85.240	85.517	88.186	6.290	1.178	1.669	81.272

TABLE VII. PREDICTION ACCURACY AND TRAINING TIME FOR NB

	Average Prediction Accuracy			**Average Training Time**			**Speed Imp (%)**
Datasets	***Standard***	***ACOISA_ML***	***MCIS***	***Standard***	***ACOISA_ML***	***MCIS***	
Landstat	79.600	78.705	78.890	0.130	0.036	0.027	72.308
Letter	62.300	62.273	60.680	0.100	0.058	0.079	42
Mushroom	90.829	91.945	100	0.120	0.025	0.030	79.167
Optdigit	89.427	83.923	88.798	0.080	0.050	0.036	37.500
Page-blocks	90.846	92.764	93.697	0.040	0.014	0.013	65
Shuttle	92.207	92.530	90.464	0.300	0.077	0.078	74.333
Twitter	94.961	93.247	95.493	4.460	0.208	1.666	95.336
USPS	76.781	75.102	74.873	0.650	0.319	0.191	50.923
Pentdigit	82.133	81.635	81.704	0.070	0.027	0.024	61.429
Waveform	**81.020**	80.608	80.461	**0.080**	0.011	0.022	86.250

TABLE VIII. ALGORITHM TIME AND STORAGE PERCENTAGE

Average Algorithm Time (Seconds)			**Average Storage Percentage**		
Dataset	***ACOISA_ML***	***MCIS***	***Standard***	***ACOISA_ML***	***MCIS***
Landstat	6.791	0.362	100	31.567	41.669
Letter	101.420	1.329	100	43.750	41.669
Mushroom	14.758	0.761	100	35.435	41.667
Optdigit	32.614	0.946	100	52.315	41.669
Page-blocks	19.839	0.212	100	45.678	41.659
Shuttle	645.186	4.444	100	43.678	41.667
Twitter	503.838	36.839	100	6.219	41.666
USPS	97.992	2.895	100	43.890	41.668
Pentdigit	15.175	0.469	100	33.360	41.660
Waveform	4.056	0.333	100	24	41.680

TABLE IX. COMPARISONS BETWEEN ACOISA_ML, LDIS, LSSM, LSBO AND ISDSP (SVM CLASSIFIER)

Dataset Name	**ACOISA_ML**		**LDIS**		**LSSM**		**LSBO**		**ISDSP**	
	Accr	***Stor***	***Accr***	***Stor***	***Accr***	***Stor***	***Accr***	***Stor***	***Accr***	***Stor***
Cardiotocography	71.25	73.87	62	86	67	14	62	69	59	90
Ecoli	84.97	32.79	77	92	83	9	74	83	78	90
Heart-statlog	82.44	38.27	81	93	84	15	81	67	78	90
Ionosphere	92.51	68.25	84	91	88	4	45	81	86	90
Landsat	84.81	54.90	84	92	87	5	85	88	84	90

Letter	89.10	75.00	75	82	84	4	73	84	74	90
Optdigits	92.13	47.69	96	92	99	2	98	92	97	90
Page-blocks	95.12	79.69	94	87	94	3	92	96	91	90
Parkinson	80.95	41.69	82	83	87	11	82	87	85	90
Segment	94.63	85.78	89	82	90	10	90	82	87	90
Wine	94.94	34.97	94	88	97	11	96	75	93	90

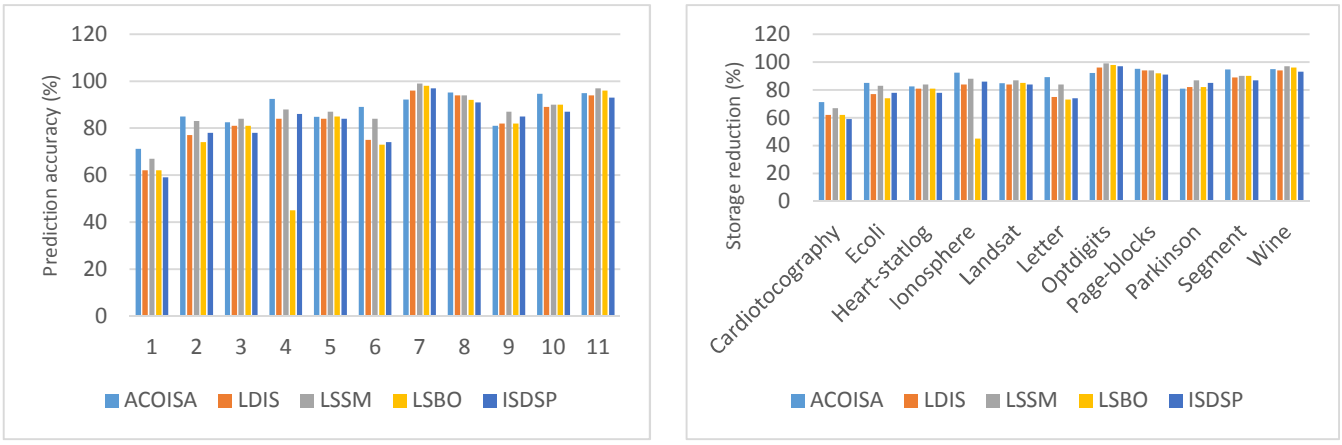

Fig 1. Comparison between ACOISA_ML and LDIS, LSSM, LSBO, ISDP

TABLE X. COMPARISON OF ACOISA_ML (K-NN CLASSIFIER) AND MOCHC

Dataset Name	**ACOISA_ML**		**MOCHC**	
	Accr	***Stor***	***Accr***	***Stor***
Yeast	51.13	52.83	96.20	53.38
Segment	94.83	79.70	80.50	54.50
Page-blocks	95.90	63.46	94.82	54.77
Optdigit	92.50	60.76	97.60	54.02
Mushroom	99.91	64.56	94.98	54.01
Letter	91.00	56.25	98.06	55.03
Shuttle	99.78	56.33	99.64	54.57

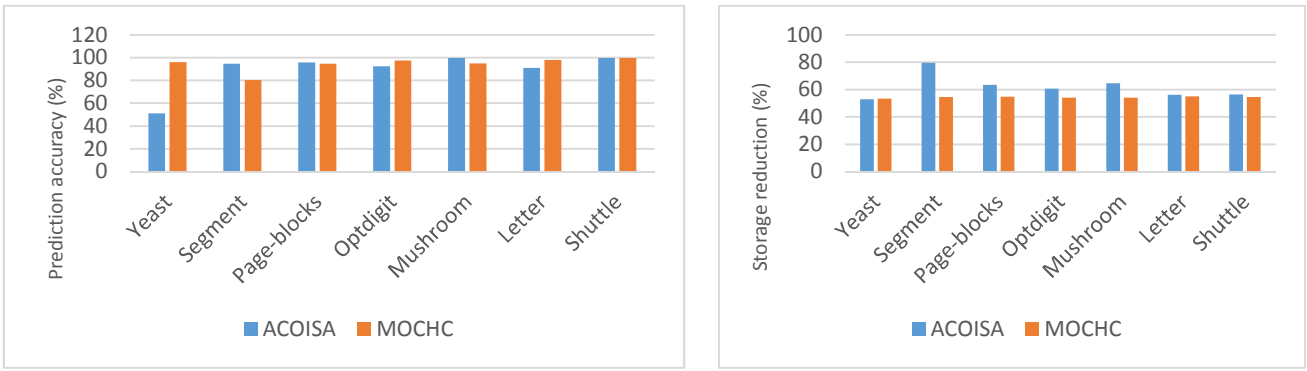

Fig 2. Comparison between ACOISA_ML AND MOCHC

TABLE XI. P-VALUES FOR T-TEST STATISTICS

Datasets	**ANN**	**RF**	**LR**	**NB**	**k-NN**
Letter	2.11104E-17	1.98738E-06	2.87349E-09	0.000411357	0.346373058
Landstat	3.90011E-10	2.35831E-06	0.282637755	0.092211804	1
Pentdigit	2.16469E-10	8.2398E-11	1.16369E-05	0.000664543	0.219861381
Shuttle	1.71188E-06	8.03282E-07	8.75876E-13	0.000279903	0.419029571
Mushroom	3.45101E-10	5.89417E-07	1.20705E-05	0.000602646	0.057731897

Algorithm 1: Ant Colony Optimization Instance Selection Algorithm

Input: $D, MaxG, N, NSub, NR, NRuns, K$
Output: T_s, the reduced training subset

```
1  Start ACOISA_ML
2        TrainingSet ← ExtractTrainingSet()
3        ACOISA(TrainingSubset) /*Pass training subset to ACOISA_ML for boundary and instance selection*/
4            Randomly select M training instances from TrainingDataset, where M = size of training subset
5            Randomly assign ants to instances and initialize pheromone value for all ants
6            for a = 1 to N /* Compute heuristic value for all instances in dataset */
7                for b = 1 to N
8                    Compute distance between instance_a and instance_b, where a ≠ b
9                    dist[a, b] = distance
10               end b
11               NL[a] ← ComputeKNN(dist[a]) /*Compute k nearest neighbours for instance_a */
12               for b = 1 to K /* Compute heuristic value for each instance */
13                   if Class of instance_a ≠ Class of NL[a, b]
14                       HV[a]+= dist[a, b] /*compute the heuristic value for instance_a */
15                   end if
16               end b
17           end a
18           while (p < MaxG) /* Start moving ants */
19               for k = 1 to N
20                   for m = 1 to NR
21                       Move k^th ant to m^th neighbouring node using equation (2)
22                       Perform update using equation (3)
23                   end m
24               end k
25               Perform update using equation (4)
26               p++
27           end while
28           E ← SelectBoundaryInstance(HV, D) /*Select instance with the highest heuristic value*/
29           T_s ← ComputeKNN(E) /*Select k nearest neighbours to the Boundary Instance*/
30       end
31 end ACOISA_ML
```