Trade Between Population Size and Mutation Rate for GAAM (Genetic Algorithm with Aggressive Mutation) for Feature Selection

Marc Chevallier, Nistor Grozavu, Faouzi Boufarès, Nicoleta Rogovschi, Charly Clairmont

Trade between population size and mutation rate for GAAM (genetic algorithm with aggressive mutation) for feature selection*

Marc Chevallier[1 [0000−0002−7983−6147]], Nistor Grozavu[1], Faouzi Boufarès[1], Nicoleta Rogovschi[1], and Charly Clairmont[2]

[1] LIPN Laboratory, Sorbonne Paris Nord University, Villetaneuse, France
https://lipn.univ-paris13.fr/
mchevallier@lipn.univ-paris13.fr
nistor.Grozavu@lipn.univ-paris13.fr
faouzi.boufares@lipn.univ-paris13.fr
nicoleta.rogovschi@lipn.univ-paris13.fr
[2] charly.clairmont@synaltic.fr
https://www.synaltic.fr/

Abstract. The "curse of dimensions" is a term that describes the many difficulties that arise in machine learning tasks as the number of features in the dataset increases. One way to solve this problem is to reduce the number of features to be provided to the model during the learning phase. This reduction in the number of dimensions can be done in two ways, either by merging dimensions together or by selecting a subset of dimensions. There are many methods to select the dimensions to be kept. One technique is to use a genetic algorithm to find a subset of dimensions that will maximize the accuracy of the classifier. A genetic algorithm specially created for this purpose is called genetic algorithm with aggressive mutation. This very efficient algorithm has several particularities compared to classical genetic algorithms. The main one is that its population is composed of a small number of individuals that are aggressively mutated. Our contribution consists in a modification of the algorithm. Indeed we propose a different version of the algorithm in which the number of mutated individuals is reduced in favor of a larger population. We have compared our method to the original one on 17 datasets, which allowed us to conclude that our method provides better results than the original algorithm while reducing the computation time.

Keywords: Features selection · Machine Learning · Genetic Algorithm · Metaheuristic

1 Introduction

In our post-digital transition world, more and more data is accumulated every year in companies. Much of this data is of poor quality [1]. Improving data

* Supported by Synaltic : www.synaltic.fr

quality is therefore becoming a task of great importance [2]. There are many methods to improve data quality. A large number of methods are based on the extraction of metadata on data: this field is called data profiling [3]. Our research in this area led us to study the features extracted by the Sherlock algorithm (an automatic semantic data type detection algorithm) [4]. We have reused similar features for several other data profiling tasks [5,6]. However, one obstacle to the industrial use of these methods is the large number of features extracted. Indeed, the time cost of extracting each feature increases with the volume of data. We are therefore confronted with the need to reduce the number of features extracted in order to be able to process larger volumes of data.
This is how we got interested in the possibilities of dimensional reduction. There are two main types of approaches, those which seek to merge the features together to create a smaller number of features (like ACP [7] or Autoencoder [8]). The best subset is the one that will give the best accuracy for a given classifier. And the methods that will try to select the best subset of the present features. Since the first possibility still requires the extraction of the initial set of characteristics, we turned to the second. This area is the field of feature selection [9]. This topic can be divided into two branches. The first one gathers the methods named "filters" which do not use a classifier to select the features, examples of algorithms of this type are : Correlation-based Feature Selection [10], Information Gain, ReliefF [11]. The advantage of these methods is that they are very fast.
The second grouping includes approaches using a classifier for feature selection. This field of research is itself subdivided into two types of methods, the methods called "Wrapper" and the methods called "Embedded". In Embedded methods the choice of features is done by evaluating the features at each iteration during the learning phase. These methods are not usable with all learning algorithms. An example of this type of algorithm is LASSO [12]. In wrapper methods the selection of features takes place after the classifier has been trained. Algorithms of this type for example: random selection, Recursive feature Elimination [13] ,genetic algorithms [14]. We have chosen to focus on genetic algorithms because although their computational cost is high, they can obtain excellent results with any classifier.

2 Related Work

2.1 Genetic algorithm

Genetic algorithms are bioinspired population-based metaheuristic algorithms popularized by J. Holland in the 1970s [16]. The goal of these algorithms is to minimize or maximize an objective function called fitness function. The parameters of this function are encoded in the form of a chromosome. Traditionally this encoding is done in a binary way, so a chromosome is a list of genes that can only take the values 0 and 1. The functioning of these algorithms is based on two basic operations. The first is the mutation which consists in altering a chromosome by randomly changing some of its genes. And the second one is the crossing which consists in mixing two chromosomes in order to create two new

ones. The classic method to do this is to break the two parent chromosomes into 1 point and then swap the pieces of chromosomes from the two parents to form two children. There are however several other methods to do this more efficiently [16].
Genetic algorithms generally have 5 parameters: the population size, the number of iterations, the chromosome size, the mutation probability and the crossover probability. The overall operation of the algorithm is as follows: a population (group of chromosomes) is randomly generated. Each individual generated in this way is evaluated using the fitness function. Depending on the results of this evaluation and the probability of crossover a daughter population is generated by crossing the initial population. Then a part of the daughter population is mutated according to the selected mutation probability. The daughter population is then evaluated using the fitness function. Finally a selection step takes place among the individuals of both populations (according to the fitness score) and a population of the same size as the initial population is kept. The algorithm then repeats this process of crossover, mutation, evaluation, selection until the number of iterations (generations) is reached. The chromosomes of the best individual of the last generation encode a good solution to the problem we are trying to solve.
For the feature selection problem, the encoding is done in the following way: if a dataset contains 10 features, the chromosomes will be of size 10, a 1 will represent a feature that can be used during the training of the classifier, a 0 an absent feature. The fitness function is the accuracy of the classifier. This method presents two problems. First, it is not possible to choose the number of features that we want to keep. Secondly, genetic algorithms tend to get trapped in local optimums. In the case of feature selection the algorithms tend to keep a number of features slightly less than half of the total number of features. This number can be reduced by modifying the fitness function to penalize the use of too many features. But this only reduces the number of selected features by 20% [17].

2.2 Genetic algorithm with aggressive mutation GAAM

The Genetic Algorithm with Aggressive Mutations (GAAM) is an algorithm that has been specifically designed for feature selection [18]. This algorithm has several major differences from classical genetic algorithms. First of all, the encoding of genes does not use a binary system but integers. If a dataset contains N features, the encoding uses N integers, each number representing 1 feature. In addition, the value 0 represents the absence of characteristics. Then the algorithm allows to choose the maximum number of features that we want to keep, this number corresponds to the size of the chromosome. For example if a dataset has 10 features and we want to keep a maximum of 5, examples of chromosomes would be: [2 7 9 10 8], [3 2 2 10 5] [0 7 8 9 5]. If a gene is present several times as in [3 2 2 10 5] the duplicates are eliminated and replaced by 0.
Then the mutation system is different from that of traditional algorithms, indeed each individual will allow to create a number of mutants equal to its size. Each mutant is a copy of the initial chromosome where only 1 gene has been

mutated (by random draw between 0 and the size of the chromosome), for the first mutant individual it is the first gene that is mutated, for the second the second gene and so on. For example if we have gene [2 3 4] the mutants will be for example [4 3 4], [2 0 4] and [2 3 2]. The crossover is a classical 1 point crossover, each individual is crossed in order to generate a daughter population of size equal to the size of the initial population. Thus the algorithm does not require a mutation probability and crossover probability parameter.
The last difference between the GAAM algorithm and the classical genetic algorithms lies in the order in which the operations are performed. Indeed, the randomly generated initial population is not evaluated at the beginning of the algorithm, it is simply used to generate the daughter population and the mutant population. So if we start with N individuals with chromosomes of size T we obtain N individuals in the daughter population and N*T individuals in the mutated population. We have thus 2*N + N*T individuals to evaluate at the end of the first generation. We keep then the N best individuals. GAAM is described in the algorithm 1.

Algorithm 1 : **GAAM**, G represents the number of generations, N the number of individuals, T the size of the chromosomes, L the size of the dataset.

INPUT : *G:int, N:int, T:int ,L:int*
g = 0
Step 1 : Build *N* individuals with *T* genes randomly picked in {0,1,2..L} in order to produce the initial population *Ip*
Step 2 : Aggressive mutation : Create *Mp* the mutate population
for j=1 to N **do**
  for x=1 to T **do**
    pick a random value *m* in *{0,1,2..L}*
    Assign to *D* a copy of **Ip(j)**
    D(x)=m
    add *D* to *Mp*
  end for
end for
Step 3 : Crossover : Apply a classical holland crossover on each individuals
Step 4 : Create *Tp=Ip+Mp+Cp* the total population evaluate each individual with the fitness function and rank them according to their fitness.
Step 5 : Drop the *N+N*T* individuals with the lowest fitness from *Tp* and replace *Ip* by the remaining,*g+=1* If *g = G* return *Ip(0)* else back to step 2

3 Modified genetic algorithm with aggressive mutation mGAAM

The GAAM algorithm is very efficient but the aggressive mutation principle forces to use only a small number of initial individuals in order not to have too

many individuals to evaluate after the mutation step. This low number of initial individuals will lead to a low genetic diversity within the population, which may cause the algorithm to remain trapped in a local optimum. We wanted to evaluate the impact of the initial population size on the results of the algorithm. We have therefore defined a version of GAAM, named mGAAM where the size of the initial population can be modified compared to a classical version of the GAAM algorithm. Thus our algorithm takes as parameters the size of a population and the number of iterations of a classical version of GAAM. Our goal is to keep approximately the same number of individuals in the population as the version of GAAM defined with these parameters while proposing a lower or higher number of individuals in the initial population while keeping the same number of crossed individuals as in the initial version. Thus the parameter that will be adjusted is the number of mutant chromosomes generated by a chromosome during the mutation step. If we use a larger population than the original version, for example 20 instead of 10, the number of mutated individuals will be reduced in order to obtain on average the same number of individuals per iteration as the original version of the algorithm.
Our method changes the steps 2 and 3 of the GAAM algorithm. For step 3, we keep only a number of crossed individuals equal to mGAAM_pop_seize. This number corresponds to the size of the population to which we compare ourselves. We cross all the individuals then we draw the desired number. Algorithm 2 replaces the mutation step of the original version. It is enough to add to the original version a parameter GAAM_pop_size which describes the size of the population to which one wishes to compare. Moreover the N corresponds to mGAAM_pop_size in algorithm 2. When the population size is smaller than the target population size, each gene is mutated several times. The overall mutation process depends on probabilities so the number of individuals generated at each generation is not fixed. However, the average size of the population at each iteration (before the selection step) corresponds to the size of the one of the classical GAAM algorithm (for the parameters we have chosen). In order to maintain this correspondence the mutation rate to be used must be calculated each time the experimental parameters change, this is done using the algorithm 3.

4 Comparisons between Gaam and mGaam

Our experimental setup is a Google Colab [23] instance with a Xeon 2.30GHz 4-core and 25GB Ram. The classifier used is a naive Bayesian classifier [19], we chose this classifier because it has the advantage of being extremely fast to train. The evaluation of the results is done using the average of the accuracy calculated on 3 cross validation.
The 17 used datasets are present and described in the UCI machine learning repository [20]. We used the versions of these datasets freely available on Open ML [21,22].The datasets used are described in Table 1. The datasets contain mostly numerical data, the categorical data are encoded.

Algorithm 2 : **Mutation mGAAM** pop the population, L the number of features in the dataset, mGAAM_pop_size : is the desired population size, GAAM_pop_size is the size of the population being compared to, T is the size of the chromosome

INPUT : *pop:list, L:int, mgaam_pop_size:int, gaam_pop_size:int, T:int*

INIT : tm ← Calc_tm_standard(mgaam_pop_size, gaam_pop_size,T)
offspring ← []

for *j=1 to size(pop)* **do**
  tmp_offspring ← []
  tmp_tm = copy(tm)
  while *tmp_tm>0* **do**
    for *i=1 to T* **do**
      ind *gets* copy(pop[j])
      $rho \leftarrow random\_uniform(0, 1)$
      if $rho < tmp\_tm$ **then**
        new_val ← random value between 0 and L
      ind[i]← new_val
      add ind to tmp_offspring
    tmp_tm ← tmp_tm - 1
  offspring ← offspring + tmp_offspring
return offspring

Algorithm 3 : **Calc_tm_standard** mgaam_pop_size : is the desired population size, GAAM_pop_size is the size of the population being compared to, T is the size of the chromosome

INPUT : *mGAAM_pop_size:int, GAAM_pop_size:int, T:int*

tm ← $\frac{gaam\_pop\_size*(T+1)-mgaam\_pop\_size)}{T*mgaam\_pop\_size}$
return *tm*

Each result presented is calculated from the results of 50 simulations. The mutation rates for mGaam are automatically calculated to keep the number of individuals equal (at each generation) to what would be used by the original Gaam algorithm with 10 individuals (N), and 100 iterations (G). We have chosen to use these parameters because they are always used in the original articles concerning GAAM [18,17]. The number of individuals to be evaluated and the number of characteristics must remain low in order to avoid an explosion of the number of individuals to evaluate at each iteration.

Table 1. Description of the datasets used

Name	Features	No. of classes	Examples	Source
Leaf [24]	15	30	340	openml.org/d/1482
Thoracic-surgery	16	2	470	openml.org/d/4329
Credit-g [27]	20	2	1000	openml.org/d/31
Climate-model [28]	20	2	540	openml.org/d/40994
Dermatology	34	6	358	openml.org/d/35
Ionosphere	34	2	351	openml.org/d/59
Audit [25]	36	2	1552	openml.org/d/42931
SPECTF	44	2	349	openml.org/d/1600
Hill-valley	100	2	1212	openml.org/d/1479
Spectrometer	101	48	531	openml.org/d/313
Musk	167	2	6598	openml.org/d/1116
Semeion	256	10	1593	openml.org/d/1501
Madelon	500	2	2600	openml.org/d/1485
Har [29]	561	6	10299	openml.org/d/1478
Isolet	617	26	7797	openml.org/d/300
Parkinson-speech-uci [26]	753	2	756	openml.org/d/42176
Micro-mass	1300	10	360	openml.org/d/1514

5 Results

The results of our experiments are shown in table 3 and 4. First of all the column corresponding exactly to the original algorithm is the one where the population size is 10. By observing the results we notice that in the majority of the datasets the results are better when the population size is higher than 10, very often a size of 60 with a low mutation rate gives better results. This is explained by the fact that using a larger starting population allows to obtain on average better individuals at the end of the first iteration. These better initial individuals then allow to obtain mostly better final results at the end of the 100 generations. This effect can be seen in the figure 1 which represents the average accuracy results at each iteration for the Credit g dataset. We can also see in the figure 1 that it

is useless to increase the population indefinitely, the population of size 90 having no advantage over the population of size 60.

We can then see that progressively when the number of features increases in the datasets, the best solutions are found for smaller population sizes. This is explained by the fact that our method tries to have the same number of individuals at each iteration as the initial algorithm. This constraint implies that at each generation when the population is greater than 10 we evaluate less new individuals than the initial GAAM algorithm. Indeed with the initial algorithm and the parameters of the experiment at each iteration (except the first one) 10 individuals from crossing and 100 individuals from mutations are evaluated. Whereas with the method we used in the experiment if the population is 30 there are on average only 90 new individuals evaluated in each generation. This has the effect of making the algorithm faster but also reduces the speed of convergence. But when the number of features becomes important the algorithm does not have time to converge completely because the problem becomes more difficult. This effect can be visualised in figure 2 on the Micro-mass dataset. We can thus explain the decrease in performance of the largest populations when the number of features increases.

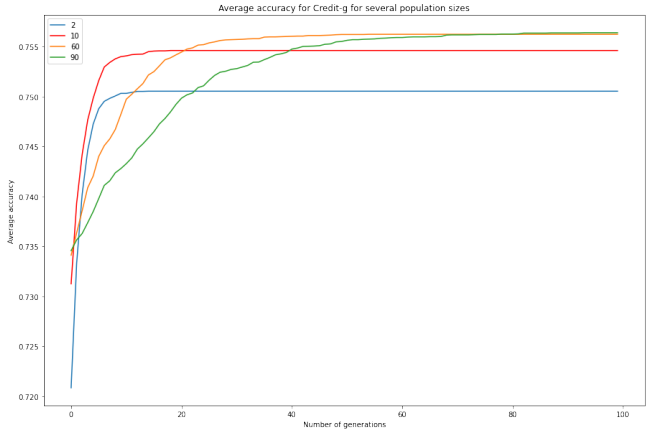

Fig. 1. Average accuracy for Leaf for several population sizes

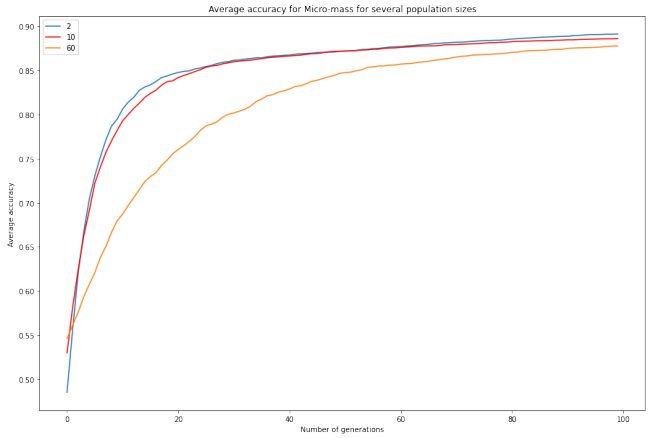

Fig. 2. Average accuracy for Micro-Mass for several population sizes

This problem can be alleviated by replacing the algorithm 3 with algorithm 4. The modification in the method of calculation allows to obtain a larger number of individuals evaluated at each iteration (equivalent to what we have with the classical GAAM algorithm). The change of calculation method increases the mutation rate in mGAAM (when the initial population is greater than the size of the population of the GAAM algorithm which is used as a comparison point) to compensate for the deficit of individuals evaluated at each iteration because of the non-evaluation of individuals of the previous generation during a new generation. We have tested (with the same parameters as before) the algorithm 4 on the micro-mass Dataset, the results are presented in the table 2. There is a clear improvement in the results for large populations. However, the drawback of this modification is the absence of gain in computation time compared to the original algorithm.

Algorithm 4 : **Calc_tm_imp** mgaam_pop_size : is the desired population size, gaam_pop_size is the size of the population being compared to, T is the size of the chromosome, G the number of generations

INPUT : *mgaam_pop_size:int, gaam_pop_size:int, T:int,G:int*

tm $\leftarrow \frac{gaam\_pop\_size - mgaam\_pop\_size}{G*T*mgaam\_pop\_size} + \frac{gaam\_pop\_size}{mgaam\_pop\_size}$

return *tm*

Table 2. Results on Micro-mass using algorithm 4 to generate the mutation rate

Dataset/Population size		10	20	30	40	60
Micro-mass	mean	88.588	88.861	**89.216**	89.155	89.155
	median	88.472	88.888	**89.444**	89.027	89.444
	max	88.588	91.944	**92.222**	91.666	**92.222**

Table 3. Results of the mGaam algorithm on several datasets defined against a version of gaam using 10 individuals, 10 and 100 iteration size chromosomes, calculated from the results of 50 simulations, part a.

Data/Pop		2	4	6	8	10	20	30	40	60
Leaf	$\overline{m}$	73.227	73.220	73.250	73.392	73.333	73.534	73.504	73.510	**73.552**
	Mdn	73.552	73.552	73.552	73.552	73.552	73.552	73.552	73.552	73.552
	max	73.552	73.552	73.552	73.552	73.552	73.552	73.552	73.552	73.552
Thora	$\overline{m}$	79.590	84.246	83.233	84.766	84.953	85.119	85.217	85.238	**85.285**
	Mdn	84.254	84.257	85.106	85.106	85.106	**85.319**	**85.319**	**85.319**	**85.319**
	max	**85.319**	**85.319**	**85.319**	**85.319**	**85.319**	**85.319**	**85.319**	**85.319**	**85.319**
Credit	$\overline{m}$	75.053	75.383	75.291	75.399	75.461	75.505	75.563	75.555	**75.623**
	Mdn	75.250	**75.599**	**75.599**	**75.599**	**75.599**	**75.599**	**75.599**	**75.599**	**75.599**
	max	**75.899**	**75.899**	**75.899**	**75.899**	**75.899**	**75.899**	**75.899**	**75.899**	**75.899**
Clima	$\overline{m}$	92.107	92.107	92.344	92.388	92.437	92.640	92.725	92.748	**92.862**
	Mdn	91.851	92.037	92.407	92.407	92.592	92.777	**92.962**	**92.962**	**92.962**
	max	**92.962**	**92.962**	**92.962**	**92.962**	**92.962**	**92.962**	**92.962**	**92.962**	**92.962**
Derma	$\overline{m}$	96.155	96.704	96.899	96.866	96.844	96.990	97.118	97.083	**97.207**
	Mdn	96.659	96.935	96.935	96.935	96.935	96.935	96.935	96.935	**97.207**
	max	**98.046**	**98.046**	**98.046**	**98.046**	**98.046**	**98.046**	**98.046**	**98.046**	**98.046**
Ionos	$\overline{m}$	87.054	88.501	88.632	88.689	88.552	88.957	88.991	89.065	**89.105**
	Mdn	88.319	88.603	88.888	88.888	88.603	88.888	88.888	**89.173**	**89.173**
	max	**89.458**	**89.458**	**89.458**	**89.458**	**89.458**	**89.458**	**89.458**	**89.458**	**89.458**
Audit	$\overline{m}$	87.722	88.307	87.951	88.858	89.664	89.745	89.784	89.802	**89.808**
	Mdn	89.432	89.559	89.496	89.559	89.689	89.818	**89.883**	**89.883**	89.819
	max	90.142	90.142	90.142	90.142	90.142	**90.142**	90.142	**90.142**	90.142
SPEC	$\overline{m}$	81.810	81.896	81.942	82.164	82.131	82.406	82.463	82.423	**82.531**
	Mdn	81.951	81.951	81.951	82.234	82.228	82.517	**82.520**	**82.520**	**82.520**
	max	**83.092**	**83.092**	**83.092**	**83.092**	**83.092**	**83.092**	**83.092**	**83.092**	**83.092**
Hill	$\overline{m}$	52.306	52.369	52.391	52.422	52.420	52.447	52.516	52.514	**52.534**
	Mdn	52.310	52.392	52.392	52.392	52.392	52.433	**52.557**	**52.557**	**52.557**
	max	**52.640**	**52.640**	**52.640**	**52.640**	**52.640**	**52.640**	**52.640**	**52.640**	**52.640**
Spect	$\overline{m}$	57.212	57.370	57.578	57.401	57.589	57.883	57.902	**58.131**	58.037
	Mdn	57.250	57.438	57.532	57.438	57.721	58.003	58.003	**58.192**	58.003
	max	58.568	**58.945**	**58.945**	**58.945**	**58.945**	**58.568**	**58.945**	**58.945**	**58.945**
Musk	$\overline{m}$	95.060	95.233	95.124	95.267	95.294	95.251	95.290	**95.338**	95.284
	Mdn	95.141	95.217	95.240	95.285	95.308	95.240	95.346	**95.369**	95.270
	max	**95.452**	**95.452**	**95.452**	**95.452**	**95.452**	**95.452**	**95.452**	**95.452**	**95.452**

Table 4. Results of the mGaam algorithm on several datasets defined against a version of gaam using 10 individuals, 10 and 100 iteration size chromosomes, calculated from the results of 50 simulations, part b.

Data/Pop		2	4	6	8	10	20	30	40	60
Musk	$\overline{m}$	95.060	95.233	95.124	95.267	95.294	95.251	95.290	**95.338**	95.284
	Mdn	95.141	95.217	95.240	95.285	95.308	95.240	95.346	**95.369**	95.270
	max	**95.452**	**95.452**	**95.452**	**95.452**	**95.452**	**95.452**	**95.452**	**95.452**	**95.452**
Seme	$\overline{m}$	66.367	66.178	66.204	**66.817**	66.726	66.514	66.661	66.572	65.546
	Mdn	66.823	66.478	66.603	66.917	**66.980**	66.698	66.917	66.792	65.599
	max	**68.926**	68.424	68.361	68.738	68.424	**68.926**	68.424	68.361	68.047
Made	$\overline{m}$	63.856	63.833	63.895	64.039	63.963	**64.353**	64.131	63.941	64.078
	Mdn	63.653	63.730	63.692	63.884	63.961	**64.212**	63.961	63.827	63.923
	max	65.038	65.577	65.347	**65.808**	65.731	65.615	65.577	65.308	65.346
Har	$\overline{m}$	90.949	**90.966**	90.957	90.842	90.937	90.853	90.761	90.742	90.411
	Mdn	90.994	90.989	**91.018**	90.965	91.013	90.940	90.756	90.804	90.435
	max	91.270	91.261	**91.280**	91.270	91.261	91.261	91.212	91.203	91.096
Isol	$\overline{m}$	74.846	74.979	4.971	74.805	**75.089**	74.722	74.339	74.296	73.241
	Mdn	74.746	75.073	5.099	74.996	**75.022**	74.868	74.586	74.284	73.252
	max	76.221	76.195	6.016	75.990	**76.208**	75.798	75.926	75.824	75.221
Parkin	$\overline{m}$	85.169	85.185	85.433	85.489	85.544	85.642	85.820	**85.899**	85.396
	Mdn	85.251	85.052	85.582	85.317	85.449	85.582	85.780	**85.978**	85.317
	max	87.301	87.566	87.169	86.904	87.566	87.037	87.301	**87.962**	87.169
Micro	$\overline{m}$	**89.111**	89.183	88.588	88.649	88.588	88.550	88.838	88.544	87.755
	Mdn	89.166	**89.444**	88.611	88.888	88.472	88.611	88.611	88.611	87.777
	max	91.944	**92.222**	91.944	91.944	91.111	91.666	91.666	91.388	90.833

6 Conclusion

In this paper we have presented a modification of the GAAM algorithm. The main modification is to decrease the number of mutations that each individual undergoes while using a larger population. It appears from the experiments that we have performed that our algorithm has two advantages over the original algorithm. First, the features selected by our algorithm lead to better accuracy results. Secondly, using a larger population reduces the computation time, by decreasing the total number of individuals to be re-evaluated at each iteration. In the case of datasets containing a large number of features, we have introduced a second possibility to calculate the mutation rate. This one does not have the advantage of a time saving but allows to maintain better results than the original algorithm at equal computation cost. Future studies should be conducted to combine our method with the techniques implemented in fGAAM [31] to speed up the algorithm. Moreover, we could be interested in combining our new method with seeding techniques of the initial population [30] to try to improve the results. Finally it would also be interesting to compare the method with other advanced feature selection techniques [32,33].

Acknowledgements I gratefully acknowledge Astrid Balick for her generous support. Supported by organization Synaltic

References

1. Redman, T. The impact of poor data quality on the typical enterprise. *Communications Of The ACM*. **41**, 79-82 (1998)
2. Ilyas, I. & Chu, X. Data Cleaning. (Association for Computing Machinery,2019)
3. Abedjan, Z., Golab, L., Naumann, F. & Papenbrock, T. Data Profiling. *Synthesis Lectures On Data Management*. **10**, 1-154 (2018,11), https://doi.org/10.2200/s00878ed1v01y201810dtm052
4. Hulsebos, M., Hu, K., Bakker, M., Zgraggen, E., Satyanarayan, A., Kraska, T., Demiralp, Ç. & Hidalgo, C. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. *Proceedings Of The 25th ACM SIGKDD International Conference On Knowledge Discovery And Data Mining*. pp. 1500-1508 (2019), https://doi.org/10.1145/3292500.3330993
5. Chevallier, M., Boufarès, F., Grozavu, N., Rogovschi, N. & Clairmont, C. Near duplicate column identification: a machine learning approach. *2021 IEEE Symposium Series On Computational Intelligence (SSCI)*. pp. 1-7 (2021), https://doi.org/10.1109/SSCI50451.2021.9659897
6. Chevallier, M., Rogovschi, N. Boufarès, F., Grozavu, N., & Clairmont, C. Detecting near duplicate dataset. *Proceedings of the 13th International Conference on Soft Computing and Pattern Recognition (SoCPaR) 2021)*, LNNS 417, pp. 1–10, 2022. https://doi.org/10.1007/978-3-030-96302-6_36
7. Karl Pearson F.R.S. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, And Dublin Philosophical Magazine And Journal Of Science*. **2**, 559-572 (1901), https://doi.org/10.1080/14786440109462720
8. Wang, Y., Yao, H. & Zhao, S. Auto-encoder based dimensionality reduction. *Neurocomputing*. **184** pp. 232-242 (2016), https://www.sciencedirect.com/science/article/pii/S0925231215017671, RoLoD: Robust Local Descriptors for Computer Vision 2014
9. Liu, H. & Motoda, H. Feature Selection for Knowledge Discovery and Data Mining. (Springer US,1998), https://doi.org/10.1007/978-1-4615-5689-3
10. Hall, M. Correlation-based feature selection of discrete and numeric class machine learning. (University of Waikato, Department of Computer Science,2000), https://hdl.handle.net/10289/1024, 00/08
11. Urbanowicz, R., Meeker, M., La Cava, W., Olson, R. & Moore, J. Relief-based feature selection: Introduction and review. *Journal Of Biomedical Informatics*. **85** pp. 189-203 (2018), https://www.sciencedirect.com/science/article/pii/S1532046418301400
12. Tibshirani, R. Regression Shrinkage and Selection Via the Lasso. *Journal Of The Royal Statistical Society: Series B (Methodological)*. **58**, 267-288 (1996), https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1996.tb02080.x
13. Guyon, I., Weston, J., Barnhill, S. & Vapnik, V. Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning*. **46**, 389-422 (2002,1), https://doi.org/10.1023/A:1012487302797
14. Reeves, C. Genetic algorithms. *Handbook Of Metaheuristics*. pp. 109-139 (2010)
15. Holland, J. Adaptation in natural and artificial systems. (University of Michigan Press,1975,12)

16. Rimcharoen, S. & Leelathakul, N. Ring-Based Crossovers in Genetic Algorithms: Characteristic Decomposition and Their Generalization. *IEEE Access.* **9** pp. 137902-137922 (2021)
17. Izabela RejerIzabela, R. Classic genetic algorithm vs. Genetic algorithm with aggressive mutation for feature selection for a brain-computer interface. (Przeglad Elektrotechniczny 1(2):100-104,2015)
18. Rejer, I. Genetic algorithm with aggressive mutation for feature selection in BCI feature space. *Pattern Analysis And Applications.* **18**, 485-492 (2015,8), https://doi.org/10.1007/s10044-014-0425-3
19. Zhang, H. The Optimality of Naive Bayes. (2004)
20. Dua, D. & Graff, C. UCI Machine Learning Repository. (University of California, Irvine, School of Information,2017), http://archive.ics.uci.edu/ml
21. Vanschoren, J., Rijn, J., Bischl, B. & Torgo, L. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations.* **15**, 49-60 (2013), http://doi.acm.org/10.1145/2641190.2641198
22. Matthias Feurer OpenML-Python: an extensible Python API for OpenML. *ArXiv.* **1911.0249**, https://arxiv.org/pdf/1911.02490.pdf
23. Bisong, E. Google Colaboratory. *Building Machine Learning And Deep Learning Models On Google Cloud Platform: A Comprehensive Guide For Beginners.* pp. 59-64 (2019), https://doi.org/10.1007/978-1-4842-4470-8_7
24. Silva, P., Marçal, A. & Silva, R. Evaluation of Features for Leaf Discrimination. *ICIAR.* (2013)
25. Hooda, N., Bawa, S. & Rana, P. Fraudulent Firm Classification: A Case Study of an External Audit. *Applied Artificial Intelligence.* **32**, 48-64 (2018), https://doi.org/10.1080/08839514.2018.1451032
26. Sakar, B., Isenkul, M., Sakar, C., Sertbaş, A., Gurgen, F., Delil, S., Apaydin, H. & Kursun, O. Collection and Analysis of a Parkinson Speech Dataset With Multiple Types of Sound Recordings. *Biomedical And Health Informatics, IEEE Journal Of.* **17** pp. 828-834 (2013,7)
27. Groemping, U. South German Credit Data: Correcting a Widely Used Data Set. *Reports In Mathematics.* (2019)
28. Lucas, D., Klein, R., Tannahill, J., Ivanova, D., Brandon, S., Domyancic, D. & Zhang, Y. Failure analysis of parameter-induced simulation crashes in climate models. *Geoscientific Model Development.* **6**, 1157-1171 (2013), https://gmd.copernicus.org/articles/6/1157/2013/
29. Anguita, D., Ghio, A., Oneto, L., Parra, X. & Reyes-Ortiz, J. A Public Domain Dataset for Human Activity Recognition using Smartphones. *ESANN.* (2013)
30. Chevallier, M., Rogovschi, N., Boufarès, F., Grozavu, N. & Clairmont, C. Seeding Initial Population, in Genetic Algorithm for Features Selection. *Proceedings Of The 12th International Conference On Soft Computing And Pattern Recognition (SoCPaR 2020).* pp. 572-582 (2021)
31. Rejer, I. & Jankowski, J. fGAAM: A fast and resizable genetic algorithm with aggressive mutation for feature selection. *Pattern Analysis And Applications.* (2021,6), https://doi.org/10.1007/s10044-021-01000-z
32. Eid, H. & Abraham, A. Adaptive feature selection and classification using modified whale optimization algorithm. *International Journal Of Computer Information Systems And Industrial Management Applications.* **10** pp. 174-182 (2018)
33. Chotchantarakun, K. & Sornil, O. An Adaptive Multi-levels Sequential Feature Selection. *International Journal Of Computer Information Systems And Industrial Management Applications.* **13** pp. 10-19 (2021)