Nature-Inspired Computing Project Report

Ayhem Bouabid
DS-01
Innopolis University
Innopolis, Russian Federation
a.bouabid@innopolis.university

Majed Naser
DS-01
Innopolis University
Innopolis, Russian Federation
m.naser@innopolis.university

Nikolay Pavlenko
DS-01
Innopolis University
Innopolis, Russian Federation
n.pavlenko@innopolis.university

***Abstract*—Preprocessing and dataset optimization play an important role in machine learning models. Nature-inspired algorithms such as particle swarm optimization, genetic algorithm and cuckoo search algorithm can be creatively combined into a preprocessing pipeline that will rid the dataset of uninformative features and transform the data without harming the predictive capacity of the model.**

***Index Terms*—multicollinearity, clustering, nature-inspired algorithms, particle swarm optimization**

I. Introduction

The initial goals that we have set in this project was applying nature-inspired algorithms as optimizers to the dataset that contained various features that were supposed to predict population growth. The resulting dataset would be evaluated by regression Machine Learning (referred to as ML) models, and results of our preprocessing would be evaluated. However, after starting working on the project, it very soon became evident that the project would have had a very small scope, and few opportunities to explore interesting applications of Nature Inspired Algorithms (referred to as NIA).

Therefore, our final project represents a significant extension of the initial proposal. The latter constitutes of predicting the population's growth using a combination of Machine Learning and Nature Inspired Computing techniques (referred to as NIC). The final project takes this first ***experimental*** step into a more complex and sophisticated direction. Our team attempts to build a ML preprocessing pipeline almost fully-based on NIC techniques and evaluates its performance with the considered dataset as per with the initial proposal. Such direction was motivated by a number of considerations including:

1) The performance of ML models is highly correlated with the quality of the data fed to the algorithms. Performing feature selection allows us to solve a great number of problems, being especially effective in large datasets containing many features. It improves accuracy of predictions by finding and eliminating spurious relationships, as well as reducing the chances of overfitting. Other effects of feature selection are the improvement of training time for the model through cutting down on unnecessary data, and increase in interpretability, as fewer features have to be analyzed to figure out the dependencies. Therefore, Feature engineering as well as data preprocessing represent pillars of a high-performing end-to-end ML pipeline.
2) Although data preprocessing, manifested mostly in feature selection, represents one of the main applications of NIC algorithms, **feature engineering** on the other hand has not received its fair share of the scientific community's attention. The *art* of feature engineering, as mentioned above, is a crucial step of Machine Learning pipelines and an extremely valuable skill for each and every ML practitioner. Our project attempts to build an NIC-based framework for two major "feature engineering techniques":
   - Eliminating multicollinearity
   - Feature transformations
3) If we succeed in developing preprocessing that will efficiently reduce the number of features in the initial dataset, the pipeline that we developed will have to be tested on other similar datasets in order to demonstrate its effectiveness. The problem of predicting population growth covers a lot of the same variables that would be useful in predicting other statistics important in banking, finance, or government, so if our model proves its effectiveness on a more generic dataset, its area of applicability will expand significantly.

With that goal in mind, during the testing process we have resorted several times to generating new datasets (both for regression and classification problems) with the help of tools provided by the sklearn libraries, with randomly assigned or manually set parameters.

Our project will have the following structure:

1) Data Preparation
2) Multicollinearity Reduction using **Particle Swarm Optimization** algorithm
3) Feature selection with **Cuckoo Search** algorithm
4) Feature transformations with **Genetic Algorithms**
5) Additional minimalistic preprocessing of the dataset and

comparison of a number of ML models performances with and without our experimental pipeline.

II. Related Work

The idea of feature selection using Nature-Inspired Algorithms is not new - we have based our project around already existent frameworks that implement those algorithms and apply them as optimizers in preprocessing tasks.

One framework that we have used in our project is NiaPy library [14]. They have implemented a large collection of nature-inspired algorithms, and provided a simple interface to use them as optimizers. Their library is also used by the Transaction Fraud Detection project, which is based around the same idea of feature selection with the nature-inspired algorithms and has served as an inspiration behind our own project, so we have decided to follow in their footsteps.

Later, in the process of searching for another library that is compatible with NiaPy implementation of various nature-insipred algorithms, and would provide a friendly and extendable Python interface to use them as optimizers in combination with Pandas and sklearn libraries, we have found the evopreprocess library [5].

That library contains several main modules: `data_sampling`, `data_weighting`, and `feature_selection`. Howeveer, only the last module is relevant for our task, so we will only describe its functionality. The task class for `feature_selection` in evopreprocess is `EvoSampling`, which extends `_BaseFilter` class from scikit-learn. It is important to list the parameters of that class, as they are going to play a big role in the feature selection process:

- `random_seed` - seed state, by default is equal to system time in milliseconds.
- `evaluator` - ML approach used to evaluate the data, expecting a scikit-learn-compatible regressor or classifier.
- `optimizer` - NI optimizer, expecting a NiaPy-copatible method, by default Genetic algorithm
- `n_folds` - number of folds for the cross-validation split into training and validation sets
- `n_runs` - number of runs of the optimizer on each fold
- `benchmark` - evaluation class that measures the quality of data sampling. By default optimizes error rate and F-score for classification problems and mean square error for regression problems
- `n_jobs` - number of optimizers to be run in parallel, by default equal to the number of your CPU's cores

III. Methodology

A. Data preparation

The population's growth dataset was built upon the different datasets made available by the "world bank of data" [3]. In this regression problem, models are predicting the annual demographic growth based on features covering a country's significant statistics [7] [13], such as:

1) Population: age, dependency ratio, etc.
2) Agriculture: annual food production, etc.
3) Health: mortality rate, etc.
4) Education: percentage of school enrollment, etc.

Furthermore, some inputting and interpolation methods were applied, such as:

- Spline - efficient for inputting piece-wise changeing of variables, it preserves the seasonality in data.
- Linear - based on the assumption that the datapoints are linearly spaced in time, as is likely for the type of data that we collect.

B. Multicollinearity issue

1) Problem description: Before describing our approach to solve the **multicollinearity** problem, it is of significance to better describe the statistical phenomena. Mutlicollinearity manifests itself when one or more predictors (features) can be predicted with high accuracy using the other predictors. This statistical phenomena leads to several issues:

- The coefficients computed by linear models can be sensitive even to small changes in the data, leading to significant generalization error
- The model's interpretability and estimation of features' importance suffers, as is the case with one of the most robust ML models: random forest [11].

2) Multicollinearity detection: Multicollinearity can be detected in a dataset by finding the VIF - variance inflation factor, calculated for each feature separately, according to the formula below, where R^2 is the coefficient of determination:

$$VIF = \frac{1}{1 - R^2} \tag{1}$$

VIF can normally take positive values, and by checking its value for each column, one could determine whether or not a serious issue is detected. According to some authors, value of VIF greater than 1 identifies existence of some collinearity, greater than 5 is already a cause for concern, and greater than 10 must be addressed. However, in our solution we have selected 2.5 as a boundary for VIF, as a low boundary allows us to include more features in clusters and make the results of application of our method more evident. [8].

As our solution to multicollinearity we have decided to combine 2 techniques: clustering and Particle Swarm Algorithm.

3) Clustering: Our solution is inspired by the following approach that is presented in the sklearn documentation [11]:

1) Consider the Spearman correlation as a distance measurement.
2) Build hierarchical clustering using Ward's clustering algorithm.

3) Determine a threshold for cutting the resulting dendogram.
4) Group the features according to the remaining clusters and keeping a single feature out of every group.

4) Particle Swarm Optimization Clustering: Particle Swarm Optimization is one of the most influential Nature-Inspired algorithms that can optimize a problem by iteratively improving a chosen solution, depending on the elected performance measure [2]. This property makes it possible to use PSO for our clustering problem, where the final choice of solution will be determined by the algorithm and defined fitness function. The solutions to our optimization problem will be represented as particles, and the aim of the algorithm will be to adjust those particles' position according to the best one found so far, and the best position in the neighborhood of that particle.

where N_c is umber of clusters in the particle, C_i is set of features belonging to i-th cluster, f_i is i-th feature as a vector.

Implementation of PSO that we used in our work has specific important parameters, the values we assigned to them as well and the reasoning behind those assignments are listed below:

- `func` - contains the function that PSO will be trying to optimize, in our case is equal to the cluster scores
- `n_dim` - contains the number of parameters of func, in our case is equal to the number of features
- `pop` - contains the number of particles, we have set it to 15, as it is the smallest number of birds in a flock that were in zoological papers that Kennedy and Eberhart referred to in the first paper on the PSO [9].
- `max_iter` - contains the maximum number of iterations of the algorithm, is set to 200

5) Implementing PSO for Clustering: In [2] the applications of PSO for clustering purposes are considered in details. However, this task is generally applied to datapoints/sample. A change in perspective was needed. We denote PSO for clustering as CPSO. CPSO can be determined by 3 points:

- the number of clusters
- The problem representation
- The PSO performance evaluation

In our solution:

- We run CPSO for a range value determined as [min_fraction *n, max_fraction * n], the final result is the best result out of the multiple runs
- For a predetermined number of clusters, denoted by K, we define a particle as a sequence of length n where the value at each cell is bounded by 0 and K exclusively
- Denoting by particle[i]. the value at the i-th cell of a particle, we consider the floor(particle[i]) as the cluster to which the i-th feature belongs
- A particle's score (representing a potential clustering) is then evaluated according to the formula below:

$$score(particle) = \sum_{i=1}^{N_c} \frac{1 + e^{\sum_{k,j \in C_i} dis(f_k, f_j)}}{\log(e + |C_i|)} \tag{2}$$

The upper term directs the algorithm towards having similar points in the same cluster. The lower term directs the algorithm towards having larger clusters as a term with only one element is optimal with total accumulative distance of zero. The following illustrations demonstrate the results of a the clustering process applied to a toy regression dataset [12]:

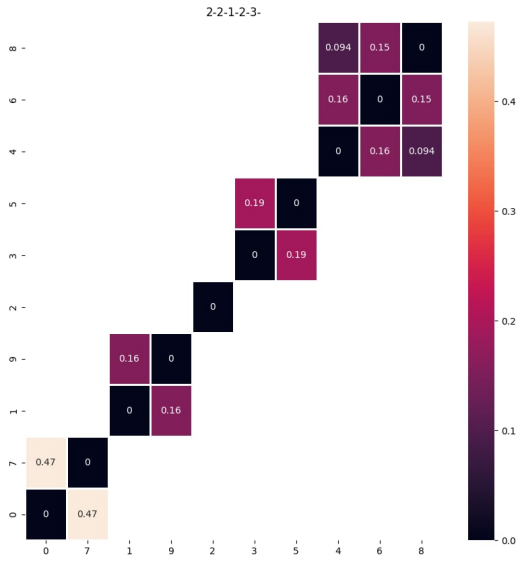

Fig. 1. Heatmap with clustered features.

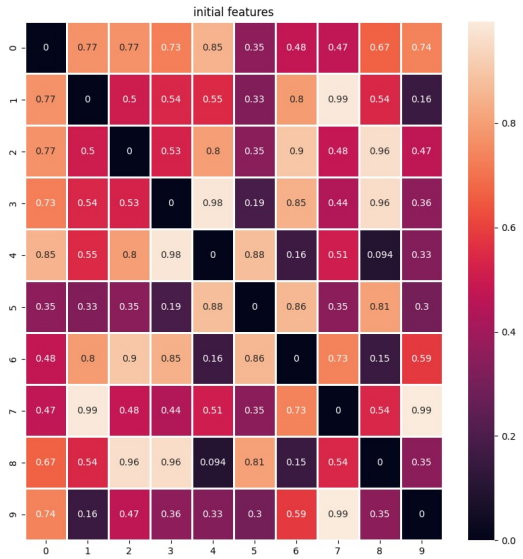

Fig. 2. Full heatmap of distances.

$$x_{i,j} = 1 - spr(i,j) \tag{3}$$

The algorithm will be checking the table to find features that are most closely correlated, and grouping them in clusters

(Fig. 1), later applying Principal Component Analysis to select one of the features from the cluster to explain the influence of all of them onto the target. Ideally, that would remove the multicollinearity problem, and allow us to reduce the number of features in the dataset without significantly harming the predictive capacity of the model.

```
for feature in features:
        if feature.VIF > 2.5:
                n.add(feature)
n.compute_correlation_matrix()
n.get_dist_matrix(n.correlation_matrix)
clusters = n.create_clusters(n.dist_matrix)
for cluster in clusters:
        final_feats.add(cluster.apply_pca())
return final_feats
```

C. Feature Selection

In our research, we were able to identify three categories of feature selection methods. Firstly, it is the filter method, which ranks each feature on some statistical metric, and evaluates the ranks afterwards, picking the ones that score the highest. Secondly, it is the wrapper method, which takes a subset of features and trains a model using them. Depending on results of the testing, it adds or removes features from the subset, incrementally improving the performance until user-defined stopping criteria is achieved. Thirdly, it is the embedded method, which is in-built into models themselves - it add a penalizing term to the regression equation (en example of such a model would be Lasso Regression) [4].

Among all categories mentioned, wrapper method has the highest computational costs, but can provide the best dataset that would provide the most accurate results for our model. It can also include nature-inspired algorithms as its estimators, making **wrapped** category of feature selection models our preferred choice.

NiaPy library provides many implementations of different nature-inspired algorithms, and using all of them in our project would have been very inefficient. Since we already do most of the preprocessing ourselves and without aid from the evopreprocess library, and use some nature-inspired algorithms in our work (that is, particle swarm optimization and genetic algorithm), we have decided to use another algorithm as an optimizer for the evopreprocess library. The cuckoo search algorithm, that was taught on the most recent lecture, caught our attention, and after finding out that it was already used for feature selection by researchers [1] [10], we have decided to select it as the optimizer in `EvoFeatureSelection` class. Levy Flights used in Cuckoo Search are also usually far more efficient than other randomization techniques based on random-walks. It also preserves a high performance while also ranking as the fastest NIC algorithm in feature selection tasks. [6]

D. Feature Transformation

Another important step in the preprocessing that we designed is feature transformation. It is undeniable that relations between target variables and predictors in real-world data rarely posses two of the most desirable statistical properties:

- Data having a normal distribution
- Linear relation between target variable and predictors

The main goal of this feature engineering step is to apply a number of functions on the given features to map it as much as possible to a normal distribution. Such procedure it likely to improve performance by possibly increase the linear correlation between the independent and dependent variables. Our implementation of this step is highly inspired by TPOT as the it makes use of Genetic Algorithm to find the best ML pipeline.

Our solution uses the following transformations:

- cox-box
- yeo-johnson
- Reciprocal
- Square root
- Quantile transformation (mapping to a uniform distribution)
- Polynomial: (from degree 1 up to degree 5)

A genetic algorithm is generally defined by:

1) Problem presentation (interpretation of the chromosome data structure)
2) Genetic operators: crossover, mutation, selection
3) Performance evaluation

Denoting by n the number of features,

- Our chromosome is a sequence of n genes, where each gene is associated with an integer value. The latter maps to one of the predefined transformations mentioned above. The latter will be applied on the corresponding feature
- Mutation operator: under a predetermined probability (mutation probability), a gene undergoes the following depending on the initial associated transformation:
  - It mapped to a different non-polynomial transformation if it is initially associated with a non-polynomial transformation
  - If it is initially associated with a polynomial transformation, then its degree either increments or decrements with 0.5 probability
- Provided with the target variable, a particle is scored as follows:

$$score = \frac{\sum_{i=1}^{n} |corr(y, f_i(x_i))|}{n} \tag{4}$$

IV. GITHUB LINK

https://github.com/Daru1914/NIC_Project

V. Experiments and Evaluation

While initially the goal of the project was to evaluate the population growth dataset, the goals were expanded to include testing for any unoptimized generic dataset, which is why our testing not only covers the regression models, but also the classification ones.

A. Testing for Elimination of Multicollinearity

First part of preprocessing that we tested was the removal of multicollinearity with the help of clustering and PSO algorithm. Using modules imported from sklearn, we generated a batch of 100 datasets. Each one was used for classification task with logistic regression, and contained 25 features with 8 informative ones and 6 redundant ones. However, initial implementation of the PSO contained mistakes, and some useful features where removed by the algorithm, because of which the accuracy of predictions in general became significantly worse after preprocessing.

After fixing those mistakes, new tests were run. In general model performed much better, as the accuracy no longer fell by 20-30% percent after the preprocessing, as was the case before. In order to interpret the results of testing more effectively, we have modified the output to include counters of the number of times performance was improved, and the number of times performance has decreased, and ran the output again. The results of that testing showed that:

TABLE I
Testing results

Tests number	Increased accuracy	Decreased accuracy
100	16	49

From those results it could be concluded that our preprocessing was ineffective and produced too many results with decreased accuracy. However, our testing was also somewhat flawed. One of the main positive consequences of dealing with multicollinearity is removal of useless features, and it wasn't tracked at all during testing. Moreover, manual observation of results of each test showed that in most of the cases where accuracy suffered, it was only reduced by a very small fraction. Keeping those observations in mind, we have redesigned the testing process.

We have added the option for the number of informative and redundant features to be generated randomly, and also started tracking the number of features removed with each iteration of the loop. Also, new criteria have been chosen to classify preprocessing as degrading accuracy of predictions - it will only be classified so, if the difference between the accuracy score before and after application of PSO clustering is greater than 0.001. The number of iterations of the testing loop was also reduced from 100 to 25 (execution time of 100 iterations approached 1.5 hours). Results received after the testing was reworked are as follows:

TABLE II
Testing for Multicollinearity

Inf. range	Redund. range	Imp. acc.	Dec. acc.	Avg. rem. feats
6-9	4-6	3	0	5.5
6-9	4-6	7	0	5.44
6-10	4-7	7	0	5.44

Results of the conducted tests confirm our earlier assessment of the effect that PSO clustering has had on the dataset and the multicollinearity problem in particular. It never significantly harms the accuracy of the model and in rare cases even improves it, while also removing more than 20% of features that are unnecessary and thereby simplifying the work during future preprocessing and prediction. While the increase in performance is not very evident on a fairly small dataset, such as ours, it would be very significant in real-world tasks related to statistical aspects of banking, finance and government.

B. Transformations testing

The effect of feature transformations was tested on 3 generated datasets for classification. Each one of them was tested on several classification models, both before and after the observations, and their accuracy score was compared. Said classification models are as follows:

1) K neighbors classifier with 3 neighbors
2) Linear SVM with $C = 0.025$
3) RBF SVM with $gamma = 2$, $C = 1$
4) Decision Tree Classifier with $max_depth = 5$
5) Random Forest Classifier with 10 estimators, $max_depth = 5$, $max_features = 1$

The results of that testing are as follows:

TABLE III
Transformations for dataset 0

Model	Accuracy before transf.	Accuracy after transf.
Nearest Neighbors	0.905	0.908125
Linear SVM	0.8675	0.885
RBF SVM	0.918125	0.92125
Decision Tree	0.908125	0.90875
Random Forest	0.91875	0.915625

TABLE IV
Transformations for dataset 1

Model	Accuracy before transf.	Accuracy after transf.
Nearest Neighbors	0.85625	0.868125
Linear SVM	0.5375	0.886875
RBF SVM	0.885625	0.889375
Decision Tree	0.8575	0.870625
Random Forest	0.864375	0.8775

As we can see, feature transformation produced small improvements in accuracy, when accuracy for the initial dataset was high enough, and it produced a major increase

TABLE V
TRANSFORMATIONS FOR DATASET 2

Model	Accuracy before transf.	Accuracy after transf.
Nearest Neighbors	0.894375	0.888125
Linear SVM	0.906875	0.90875
RBF SVM	0.915625	0.903125
Decision Tree	0.901875	0.89875
Random Forest	0.91625	0.90625

in it when accuracy was low (Linear SVM for dataset 1). However, in the testing of the last dataset (which was the most complicated), the feature transformation actually decreased the accuracy by a small amount. Generally this type of dataset can be separated by applying kernel transformations and functions such as RBF.

C. Testing the preprocessing pipeline

Testing of the population growth dataset on the entire preprocessing pipeline was performed using several selected regression models, that is: Ridge, Lasso, Decision Tree, KNN, Random Forest regressions. We have given the initial dataset to all aforementioned models and done the same thing after putting the dataset through our preprocessing pipeline. Though our implementation allows for a choice of many different metrics to compare the performance of the models, we have decided to stick to the mean square error (MSE). Results of our testing are given below:

TABLE VI
PIPELINE TESTING

Model	Pre-pipeline MSE	Post-pipeline MSE
Ridge	0.179	0.818
Lasso	0.179	1.264
Decision Tree	0.676	0.791
K Neighbors	0.133	0.298
Random Forest	0.448	0.580

VI. ANALYSIS AND OBSERVATIONS

The previous section covers in details the results of our experimental attempt. The different pipeline components achieved reasonable performance on the toy datasets generated by tools such as scikit-learn. It is crucial to bear in mind that those datasets are indeed of limited complexity and do not introduce complex and non-linear interactions between the predictors and the target variable.

The comparison of the performance of Machine Learning models before and after applying the transformation further justifies this claim. (refer to Fig. 3).

The current implementation is indeed limited and not suitable to tackle such a difficult challenge (building a NIC-based preprocessing pipeline) with complex and high-dimentional real-world data.

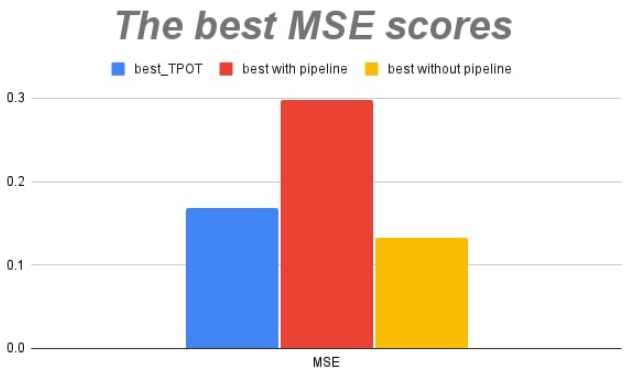

Fig. 3. Best MSE scores.

Nevertheless, we believe that the overall performance, in the light of the limited experience, the experimental nature of the project and the time constraints, is quite satisfactory.

VII. CONCLUSION

In the end, while our preprocessing pipeline has not managed to improve accuracy of the predictions in comparison to just operating on the given dataset, the drop in accuracy was not significant, and we have managed to rid the dataset of several mostly redundant features in the process.

However, the room for improvement is quite large with the current implementation. The following upgrades deserve consideration:

- Building more mathematically-robust scoring functions instead of determining it on experimental basis
- Conceive better techniques to link between the discrete nature of the problem and the continuous nature of the optimization algorithms
- Applying more rigorous statistical analysis, as well as considering additional datasets

REFERENCES

[1] Mohamed Abd El Aziz and Aboul Ella Hassanien. Modified cuckoo search algorithm with rough sets for feature selection. *Neural Computing and Applications*, 29(4):925–934, 2018.

[2] Augusto Luis Ballardini. A tutorial on particle swarm optimization clustering. *arXiv preprint arXiv:1809.01942*, 2018.

[3] The World Bank. World bank open data. https://data.worldbank.org/. Accessed: 2022-12-02.

[4] Indraneel Dutta Baruah. Deep-dive on ml techniques for feature selection in python - part 1. https://towardsdatascience.com/deep-dive-on-ml-techniques-for-feature-selection-in-python-part-1-3574269d5c69. Accessed: 2022-12-02.

[5] Sašo Karakatič. EvoPreprocess—Data Preprocessing Framework with Nature-Inspired Optimization Algorithms. *Mathematics*, 8(6), 2020.

[6] Sašo Karakatič. Evopreprocess—data preprocessing framework with nature-inspired optimization algorithms. *Mathematics*, 8(6):900, 2020.

[7] Allen C Kelley. Population growth, the dependency rate, and the pace of economic development. *Population Studies*, 27(3):405–414, 1973.

[8] Scott Menard. *Applied logistic regression analysis*. Number 106. Sage, 2002.

[9] Adam P Piotrowski, Jaroslaw J Napiorkowski, and Agnieszka E Piotrowska. Population size in particle swarm optimization. *Swarm and Evolutionary Computation*, 58:100718, 2020.

[10] Douglas Rodrigues, Luis AM Pereira, TNS Almeida, João Paulo Papa, AN Souza, Caio CO Ramos, and Xin-She Yang. Bcs: A binary cuckoo search algorithm for feature selection. In *2013 IEEE International symposium on circuits and systems (ISCAS)*, pages 465–468. IEEE, 2013.

[11] scikit-learn developers. Permutation importance with multicollinear or correlated features. https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance_multicollinear.html. Accessed: 2022-12-04.

[12] scikit-learn developers. sklearn.datasets.make_regression. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_regression.html. Accessed: 2022-12-06.

[13] Vijayakumar Sinnathurai. An empirical study on the nexus of poverty, gdp growth, dependency ratio and employment in developing countries. *Journal of competitiveness*, 2013.

[14] Grega Vrbančič, Lucija Brezočnik, Uroš Mlakar, Dušan Fister, and Iztok Fister Jr. NiaPy: Python microframework for building nature-inspired algorithms. *Journal of Open Source Software*, 3, 2018.