Nature-Inspired Computing Project Report

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

Majid 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*—tbd**

***Index Terms*—multicollinearity, clustering, particle swarm optimization**

I. Introduction

Quality data preprocessing is an important prerequisite to creating accurate machine learning models. It solves a wide range of problems, from missing data and data inconsistency to required anonymization. In our project we have decided to focus primarily on one aspect of it, that is, **feature selection**.

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.

However, the main problem that can be solved by feature selection for multiple regression models (and will be addressed specifically in our project) is **multicollinearity**. Multicollinearity is an effect when multiple explanatory variables that are believed to me independent from each other, are in fact, closely interrelated. This can have damaging effects on the accuracy of prediction models, as even a small change in data will lead to unpredictable results. Ideally, feature selections would find such relationships and purge the dataset from them, our model aims to do so as well.

As to the practical applications of our project, we have decided to stick to the original project proposal and test it on a new dataset that would contain various countrywide statistics as features that will try to predict the population growth for a country in a given year. Since we have found no such dataset that would have suited our needs, it was created from scratch, using data provided by the World Bank Open Data site [2]. However, during our work on the project, we have decided to also test other similar datasets, that could be used for a multiple regression model, to figure out if our implemented feature selection process can be used more generally.

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.

II. Related Work

NEW INFO ABOUT SOLVING MULTICOLLINEARITY WILL GO HERE

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 extensively is NiaPy library [9]. 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. Linear model selection

We have designed and trained two separate tools for this task. Each is concerned with one of the two main branches of Supervised Learning problems (Classification and Regression). The model concerned with Regression problems was a choice among many models that were trained and tested on the dataset that was manually built by our team from the world data bank (Ransac and Linear Regression, Decision Tree, Random Forest, etc...). These models were fine-tuned using Cross-Validation Method in order to be customized so that they generate the most accurate predictions. After that, the model with the lowest mean squared error was chosen (Ransac Regressor).

The model concerned with classification was determined in a similar manner. We built and trained several models (KNN, Logistic Regression, Decision Tree, Random Forest, etc...) using a dataset generated specifically for this purpose. Then, the models were similarly tuned to give highly valuable insights into our dataset and the model with the highest accuracy was chosen.

B. Multicollinearity issue

1) Multicollinearity detection: First problem that we are going to address in our dataset would be multicollinearity. It 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. [6].

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

2) Clustering: When collinear features are present in a dataset, model can gather the same information from one feature, as it can from a group of features. This makes it possible to try and solve multicollinearity by applying clustering with a certain threshold on features, that have been proven to have a correlation [8]. In order to find those correlating features, we could use Spearman correlation, which measures the strength and direction of the monotonic relationship between two variables that can be ordered by transforming their values to ranking in their respective domains and comparing their relationship.

3) 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 [1]. 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.

Implementation of PSO that we used in our work has specific 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 [7].
- `max_iter` -

4) Implementing PSO for Clustering: PSO can be directly applied to our clustering problem. As was stated before, it can be based on the table containing variables that represent how similar one feature is to another. It is filled according to the formula, where $spr(i, j)$ is the Spearman coefficient between features numbered i and j:

$$x_{i,j} = 1 - spr(i, j) \tag{2}$$

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

C. Feature Transformation

Another important step in the preprocessing that we designed is feature transformation. The goal of that step is to improve linear correlation of different features with the target by applying various functions (e.g. polynomial, square root, quantile transformation) on predictors until the accuracy of prediction improves. This idea is compatible with Nature-Inspired Algorithms to produce a combined solution. In our case feature transformation was used in conjunction with the Genetic Algorithm, where the chromosome was represented by the list of specific transformations applied to each feature. The dataset resulting from applying the transformations described in the chromosome onto the initial dataset is evaluated, and new chromosomes are generated after randomly mutating the existing ones. Eventually we will reach a dataset, where features are capable of more effectively predicting the target variable.

D. 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) [3].

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. Because of that we have decided to select those algorithms that received positive feedback as optimizers in feature selection problems: Genetic Algorithm, Clonal Selection Algorithm, Harmony Search, Artificial Bee Colony Algorithm, Firefly Algorithm, Particle Swarm Algorithm [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

Test	Inform. range	Redund. range	Imp. acc.	Dec. acc.	Avg. rem. feats
1	6-9	4-6	3	0	5.5
2	6-9	4-6	7	0	5.44
3	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.

VI. ANALYSIS AND OBSERVATIONS

tbd

VII. CONCLUSION

tbd

REFERENCES

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

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

[3] 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.

[4] Iztok Fister Jr, Xin-She Yang, Iztok Fister, Janez Brest, and Dušan Fister. A brief review of nature-inspired algorithms for optimization. *arXiv preprint arXiv:1307.4186*, 2013.

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

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

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

[8] scikit-learn developers. scikit-learn. https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance_multicollinear.html. Accessed: 2022-12-04.

[9] 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.

VIII. FUNNI TABLE

TABLE III
TABLE TYPE STYLES

Table Head	Table Column Head		
	Table column subhead	*Subhead*	*Subhead*
copy	More table copy[a]		

[a]Sample of a Table footnote.

IX. FUNNI PICTURE

Fig. 1. Example of a figure caption.