Comment on "Solving the Class Imbalance Problem Using a Counterfactual Method for Data Augmentation"

1st Fryderyk Mantiuk
University of Tübingen
Tübingen, Germany
fryderyk.mantiuk@student.uni-tuebingen.de

2nd Luisa Kurth
University of Tübingen
Tübingen, Germany
luisa.kurth@student.uni-tuebingen.de

***Abstract*—The problem of class imbalance is a well-known issue in machine learning, where the number of samples in one class is significantly larger than the number of samples in the other class. In this report, we re-implemented the counterfactual data augmentation algorithm presented by Temraz and Keane in their paper "Solving the Class Imbalance Problem Using a Counterfactual Method for Data Augmentation". The algorithm proposes a novel approach to augment the minority class by creating counterfactual samples. Our experiments were conducted on five datasets from the UCI repository and compared the results obtained by the authors in their paper with the results obtained by our re-implementation of the algorithm. We found that the method introduced in the paper did not yield improved results in our experiments. Additionally, we encountered difficulties in re-implementing the algorithm from the paper alone. Despite these challenges, the report provides an insight on the current state of the research in the field of class imbalance and counterfactual data augmentation and highlights the potentials and limitations of the proposed method.**

***Index Terms*—machine learning, class imbalance, data augmentation, counterfactual explanations**

I. Introduction

Class imbalance is a well-known issue in machine learning where the number of samples in one class (the *majority* class) is significantly larger than the number of samples in the other class (the *minority* class). This can lead to a bias towards the majority class and poor performance on the minority class. Various techniques have been proposed to address this problem, such as oversampling, undersampling and synthetic data generation.

In recent years, there has been a series of works on synthetic data generation which use *counterfactual explanations*, a concept from research on explainable AI: Counterfactual explanations are a way of understanding why a machine learning model made a certain decision or prediction by identifying a set of changes to the input data that would lead to a different outcome. For example, in a binary classification problem, a counterfactual explanation would identify a set of changes to a negative example that would cause the model to predict it as positive. Thus, by actually applying these identified changes to the negative instance, we arrive at a synthetic positive instance that did not exist beforehand. Repeating this process with additional instances, a dataset can be augmented with "counterfactual" data.

One approach to synthetic data generation which uses this concept is "Solving the Class Imbalance Problem Using a Counterfactual Method for Data Augmentation" [1]. In this paper, the authors follow up on an earlier paper [2], where they introduced a method for generating counterfactual instances and found that it improved results when they used it to augment a farming dataset for crop-growth prediction. The main goal behind their more recent paper was then to systematically evaluate the usefulness of their counterfactual data augmentation method over a wide range of classifiers and datasets, in order to answer the question whether the synthetic instances can help models to better generalize to the minority class and improve predictive performance of ML methods in general.

In their experiments, the authors report that their proposed method leads to a significant improvement in performance on a wide range of imbalanced datasets. However, the method has not been widely adopted yet, and the code has not been made public. In this report, we re-implemented the counterfactual data augmentation algorithm presented by Temraz and Keane in their paper and conducted experiments on five datasets from the UCI repository. Our goal was to evaluate the performance of the proposed method and to understand the potentials and limitations of the algorithm.

In the following sections, we will describe the counterfactual data augmentation algorithm in detail and present the results of our experiments. Additionally, we will discuss the difficulties we encountered when re-implementing the algorithm and processing the data, and give possible reasons for our inability to reproduce the improvements reported by the authors.

II. Methodology

The underlying idea of the algorithm is visualized in Fig. 1 and can be summarized briefly in three steps: (1) Find pairs of majority and minority instances that are similar in most features except one or two (2) Use the information from these "native" pairs to find the features that change an

instance's class (push it over the decision boundary) (3) Apply this change to a nearby majority instance to produce a new, synthetic minority instance, and repeat this for all unpaired majority instances.

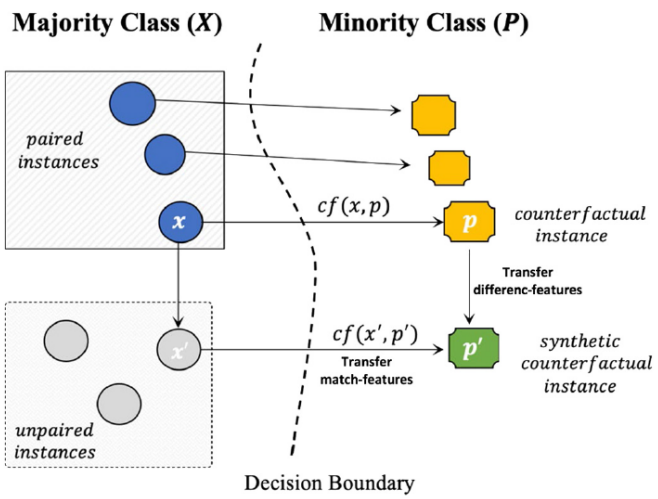


Fig. 1: The idea behind CFA.

In more detail, the algorithm **as we understood and implemented it**, is as follows (see Fig. 2):

We have a dataset with a minority and majority class.

1) For each majority class instance, find the closest minority instance using a 1-Nearest-Neighbor[1] algorithm.
2) From these pairs, select "good" native counterfactual pairs, e.g. pairs of majority and minority instances that are similar in most features except one or two:
   a) Calculate a tolerance limit by estimating the standard deviation for each feature.
   b) For each majority instance, for each feature, calculate if the difference between the feature value in the majority class instance and the feature value in its corresponding paired minority instance (determined by the 1-NN search in step 1) is within $\pm 10\%$ of the standard deviation for that feature. Features that differ by an amount larger than this threshold are noted as a "difference feature" for this specific pair. Features that differ by less than the tolerance are denoted as "match features".
   c) If a paired majority instance has less than or equal to 2 difference features, it is kept as a "good" native counterfactual pair for the next step and all other majority class instances are kept as unpaired instances.
3) For each unpaired majority instance, find the nearest paired majority instance using 1-Nearest-Neighbor[2], e.g. the closest majority instance taking part in a good native pair.
4) Generate synthetic counterfactual instances: For each unpaired majority instance, ...
   a) Identify which features of its closest paired neighbor were considered difference features in step 2, and which were considered match features.
   b) Create a synthetic instance that is counterfactually related to the current majority instance by simply copying the current majority instance and:
      - Keeping those feature values of the *unpaired majority* instance if that feature was found to be a match feature.
      - Transferring those feature values of the *minority* instance in the closest native pair if that feature was found to be a match feature.
5) Validate the generated samples:
   a) Use a classifier that was trained on the original data to predict the classes for each new minority instance.
   b) Keep only those generated samples that are in fact labelled as the minority class by the classification model. These are then the final synthetic minority instances and can be added to the data in order to create a counterfactually augmented dataset.

The whole algorithm is then ideally[3] repeated until we have as many minority instances as majority instances, e.g. a fully balanced dataset.

III. Results

We selected 3 of the UCI datasets (Yeast, Abalone and Wine Quality) on which the authors reported that their method worked better than any other compared approach. Following their approach, we turned each dataset into a specific binary classification problem by selecting a subset of all classes in a dataset as the majority class, and another subset as the minority class. In this way, we end up with 5 different binary classification datasets that the authors measured their results on (which they refer to as D8, D9, D12, D14, and D22), each with a different amount of majority and minority instances, and varying imbalance ratios.

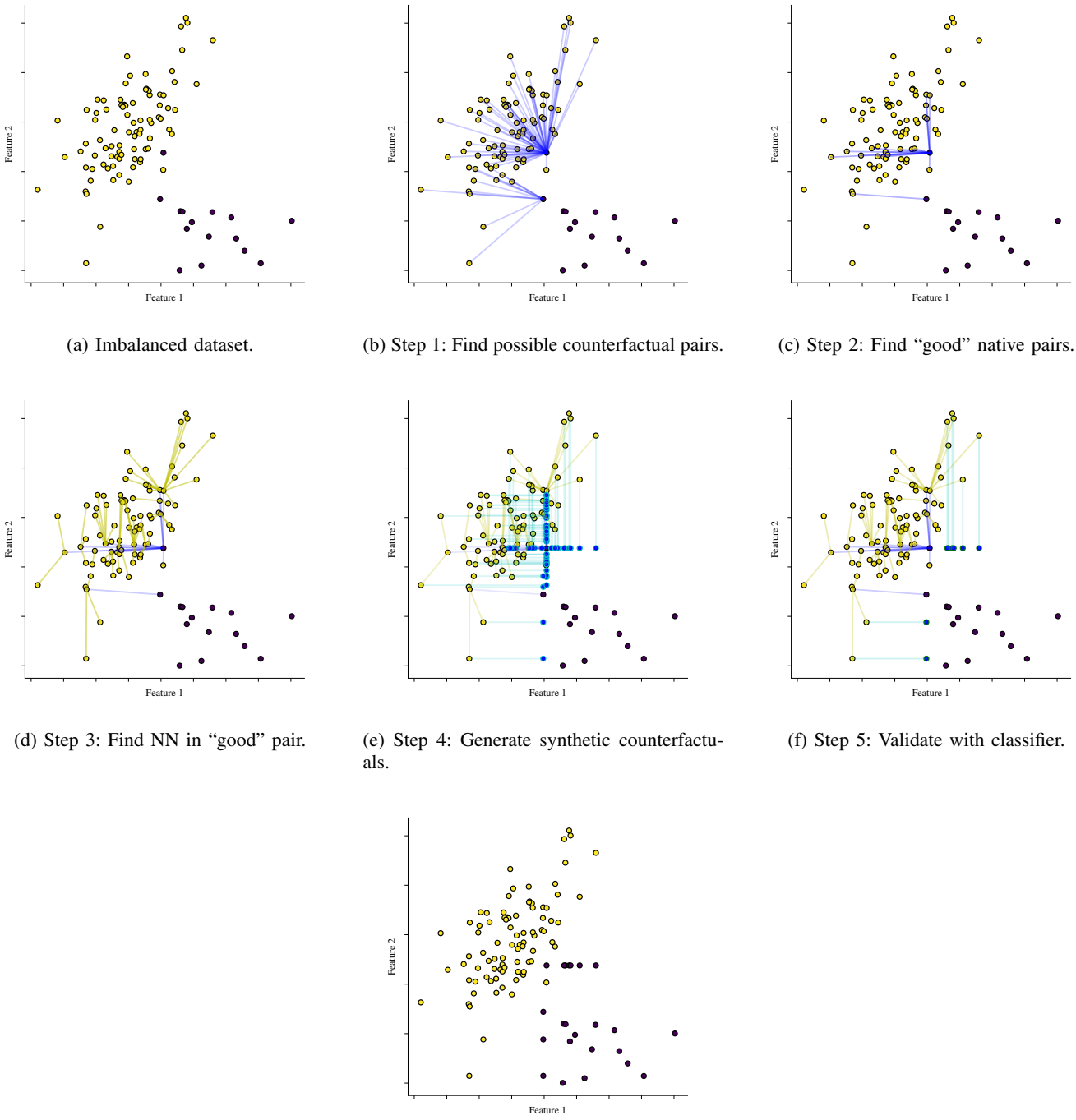
We divide each dataset into a training and test set, run a data augmentation method to oversample the minority class on the training set, train a classifier on the resulting augmented data and measure its performance on the non-augmented test set using the ROC AUC score, the same metric the authors are using.

For augmentation, we compare three approaches: (1) No augmentation as a baseline, (2) ADASYN, a state-of-the-art data augmentation method for handling class imbalance, (3) our implementation of the CFA algorithm. **Importantly, we use a threshold of 50% of the standard deviation instead of the 10%-threshold given by the authors**, because the original threshold resulted in no "good" native counterfactuals, making the subsequent steps of the algorithm impossible.

[1]Based on Euclidean distance

[2]Again, based on Euclidean distance

[3]The classifier may predict **all** new instances to be in the majority dataset, meaning that no new data is added and it is not possible to arrive at a fully balanced dataset.



(a) Imbalanced dataset.

(b) Step 1: Find possible counterfactual pairs.

(c) Step 2: Find "good" native pairs.

(d) Step 3: Find NN in "good" pair.

(e) Step 4: Generate synthetic counterfactuals.

(f) Step 5: Validate with classifier.

(g) Less imbalanced dataset.

Fig. 2: CFA visualized on a toy 2D dataset. Note that in this toy example, we select pairs with only **one** difference features as "good" native pairs (instead of the **two** difference features from the original algorithm). This is needed because our toy dataset is two-dimensional, meaning that (1) **all** pairs have less than or equal to 2 difference-features, and even if not, (2) using 2 difference-features here would reduce to simple oversampling of minority instances in step 4.

For classification, we use two of the classifiers that the authors evaluated, namely a logistic regression and a random forest classifier. For each of them, we perform a grid search with 5-fold cross-validation on the training set to find the best hyperparameters from the same grid of values that the authors used. The classifier with best hyperparameters found by the

TABLE I: Comparison of experimental results from our work with results reported in [1]. Values shown are ROC AUC scores.

Classifier	Dataset	Baseline (Paper)	ADASYN (Paper)	CFA (Paper)	Baseline (Ours)	ADASYN (Ours)	CFA (Ours)	Improvement Over Baseline Through CFA (Paper)	Improvement Over Baseline Through CFA (Ours)
Logistic Regression	D8-Yeast-0-3-5-9-vs-7-8	0.792	**0.796**	0.788	0.811	0.806	**0.812**	-0.004	+0.001
	D9-Abalone-9-vs-16	0.945	0.951	**0.990**	0.938	0.939	**0.939**	+0.044	+0.001
	D12-Yeast-1-vs-7	0.862	0.863	**0.922**	**0.874**	0.868	0.867	+0.060	-0.006
	D14-Abalone-13-vs-R	0.737	0.747	**0.838**	0.734	**0.738**	0.734	+0.101	0.000
	D22-Wine-Quality-White-3-9-vs-5	**0.742**	0.677	0.692	**0.767**	0.747	0.763	-0.050	-0.004
Random Forest	D8-Yeast-0-3-5-9-vs-7-8	0.787	0.800	**0.937**	**0.823**	0.779	0.820	+0.150	-0.003
	D9-Abalone-9-vs-16	0.890	0.917	**0.994**	**0.895**	0.890	0.890	+0.104	-0.006
	D12-Yeast-1-vs-7	0.849	0.853	**0.983**	**0.853**	0.792	0.844	+0.134	-0.009
	D14-Abalone-13-vs-R	0.750	0.763	**0.980**	**0.740**	0.717	0.740	+0.230	0.000
	D22-Wine-Quality-White-3-9-vs-5	0.806	0.799	**0.999**	**0.832**	0.779	0.832	+0.194	0.000

grid search is then used to make the predictions on the test set, from which the final ROC AUC scores are calculated.

Table I summarizes the results of our experiments and compares them to the results reported in the paper: While we were able to roughly reproduce the results of their experiments for the Baseline (no augmentation) and ADASYN, our implementation of CFA was unable to reproduce their scores, and **did not yield any significant improvements over the baseline** (see last two columns).

Another observation is that in the results reported in the paper, ADASYN was able to improve slightly over the baseline quite a few times, while in our experiments ADASYN at best achieved only results that were as good as the baseline.

IV. Discussion

In contrary to the results reported by the authors, augmenting data with CFA did not lead to performance improvements in our experiments. One possible reason for the discrepancy in results is that we encountered several ambiguities and problems while trying to implement the method.

A first ambiguity we found was in the definition of "match features" and "difference features": Here, the authors stated *"CFA computes a tolerance by finding the mean (μ) and standard deviation (σ) for each feature. Then it allows features to match if their values are within $\pm 10\%$ of the standard deviation from the mean [of] all the values for that feature."* Following this definition from the paper, it seems that match features are those that are close to the mean, and difference features are those that differ from the mean - **independent from the actual difference between a majority and minority sample in each good native pair**. From our understanding, this cannot be what the authors meant[4], which is why we reformulated the definition to state: *"CFA computes a tolerance by finding the mean and standard deviation for each feature. Then it allows features to match if their values differ from each other by less than 10% of the standard deviation for that feature."* However, we cannot know if our interpretation is correct, and the authors could still have meant something entirely different.

[4]To make sure, we actually tried using their definition, but it did not result in **any** native counterfactual pairs, making subsequent steps of the algorithm impossible.

Another source of confusion was the fact that the authors only mentioned the fact that an ML model should be used to validate generated instances once, and notably neither in the pseudo-code nor in the detailed description of each step in the algorithm accompanying it.

Also, at another point, the authors state the following:*"For our experiments, we oversample the minority class using each of the data augmentation methods until we have the same number of instances in each class. As a result, fully balanced datasets were created."*, but this was not always possible in our implementation. It may happen that the classifier predicts all generated counterfactuals to be in the majority class during the validation step, meaning that no new counterfactuals can be added, and the algorithm terminates early, which would contradict the claim that fully balanced datasets are created.

Because of these ambiguities, it seems likely that we did not correctly implement the method in the way the authors did, and our results should be interpreted with caution until the authors provide public code.

Another notable finding was that in their experiments, ADASYN seemed to improve over the baseline more consistently than it did for us. A reason for this discrepancy could simply be that both the Random Forest classifier and Logistic Regression have inherent randomness, and without knowing the seed of the authors for the classifiers as well as for their train-test-splits, we cannot arrive at the exact same results.

Furthermore, we noticed a statement from the authors that could point to a methodical error on their side. The authors stated that *"For CFA, the native counterfactuals in the training data were computed (CF-Set) and then all the remaining unpaired majority-instances were run through CFA to create the synthetic counterfactual instances in the minority class; this augmented dataset was then used for testing."* If they actually used the augmented data in the *test* set like this statement suggests, this could lead to data leakage, meaning that the generated counterfactuals might be similar enough in both the training and test set that a classifier could have simply memorized them. However, we hope that this is simply erroneous or misleading wording, and not actually a methodical error of the authors.

V. Conclusion

In this study, we aimed to re-implement and evaluate a counterfactual data augmentation algorithm. While the investigated CFA algorithm could have potential as a method for handling class imbalance, our results did not match the performance reported by the original authors. There are several possible explanations for this discrepancy, including ambiguities and problems encountered while trying to implement the method, inherent randomness in the classifiers and folds used, as well as potential methodical errors in the original study. Further research and replication studies – this time, ideally with the original code of the authors – are needed to fully understand the performance of the CFA algorithm and to improve its implementation.

References

[1] M. Temraz and M. T. Keane, "Solving the class imbalance problem using a counterfactual method for data augmentation", Machine Learning with Applications, vol. 9, p. 100375, 2022.

[2] M. Temraz, E. M. Kenny, E. Ruelle, L. Shalloo, B. Smyth, and M. T. Keane, "Handling climate change using counterfactuals: Using counterfactuals in data augmentation to predict crop growth in an uncertain climate future," Case-Based Reasoning Research and Development, pp. 216–231, Sep. 2021.