smote-variants: a Python Implementation of 85 Minority Oversampling Techniques

Article *in* Neurocomputing · June 2019

DOI: 10.1016/j.neucom.2019.06.100

1 author:

György Kovács
Analytical Minds Ltd
69 PUBLICATIONS **108** CITATIONS

SEE PROFILE

smote-variants: a Python Implementation of 85 Minority Oversampling Techniques

György Kovács

Analytical Minds Ltd., Hungary

Abstract

Imbalanced classification problems are definitely around [1], and a successful approach to avoid the overfitting of majority classes is the synthetic generation of minority training samples [2]. Despite the large number of minority oversampling algorithms proposed, open source implementations are available for only a handful of techniques. The package `smote-variants` provides a Python implementation of 85 oversampling techniques to boost the applications and development in the field of imbalanced learning. The source code, documentation and examples are available in the GitHub repository `http://github.com/gykovacs/smote_variants/`.

Keywords: imbalanced learning, SMOTE, synthetic minority oversampling, Python, smote-variants

1. Introduction

Imbalanced classification problems gained increased interest in the recent years [1, 2]. Literature [1] distinguishes three main approaches to handle imbalance in classification problems: cost-sensitive training, classifier-specific solutions, and oversampling the minority class(es). One of the first oversampling techniques was the Synthetic Minority Oversampling TEchnique (SMOTE) [3], and a recent review [2] of its numerous variants and applications clearly shows the popularity and interest in oversampling techniques.

Despite the success of oversampling, only a handful of techniques are available in open source software: the `imblearn` [4] Python package implements 3 oversampling techniques; the R package `smotefamily` offers 4 further oversampling methods, and a limited number of techniques, like CCR [5], have public, standalone implementations. Given that more than 85 variants of

SMOTE have been proposed so far [2] (out of which 9 methods in the past two years), the lack of available implementations clearly draws back the development in the field, making it difficult to properly evaluate new techniques or select the best performing technique for a particular dataset. The goal of the Python package `smote-variants` is to boost research and applications in the field by implementing 85 oversampling techniques in a comprehensive framework. According to our best knowledge, this is the first public, open source implementation for 76 oversamplers. As the current versions of `imblearn` and `smotefamily` implement 7 oversampling techniques together, we do not treat them as competitors of the proposed `smote-variants` package in scope or volume.

Besides binary oversampling, `smote-variants` offers multiclass compatible with 61 of the implemented binary oversamplers and a model selection framework to easily find the appropriate oversampler for a particular dataset. The list of all techniques implemented, citations, documentations, galleries, sample codes (Python, R and Julia) and a ranking of top-performing oversamplers based on their performance on 104 imbalanced datasets are available at the GitHub repository `http://github.com/gykovacs/smote_variants` and at the documentations page `http://smote-variants.readthedocs.io`.

The main contributions of the package to the field: (a) theorists can easily compare existing techniques and discover the most successful operating principles to drive further research in the field; (b) practitioners working with imbalanced data can select the most suitable oversamplers to improve classification performance in real applications.

The paper is organized as follows. In Section 2 a more detailed introduction to oversampling is given; in Section 3 the organization of the package and implementation details are discussed; empirical results and illustrations are presented in Section 4; and conclusions are drawn in Section 5.

2. Problems and Background

An imbalanced binary classification problem has a training set $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-, +\}, i = 1, \dots, (N_+ + N_-)$ consisting of N_+ minority and N_- majority samples with $N_+ << N_-$. Conventional classification techniques naturally overfit the majority class as the majority samples are overly represented in the loss function. Since conventional regularization techniques are prepared to balance bias and variance, they fail to regularize one-sided degeneracies like the overfitting of one class to the detriment of the other. One possible way to improve classification performance is to make assumptions on the local spatial distribution of the minority class and generate further minority training samples to balance the dataset. For example, the assumption made by SMOTE is that the line segments between neighboring minority samples belong to the minority class. Accordingly, SMOTE generates minority instances by randomly sampling these line segments: let $\mathbf{x}^1$ and $\mathbf{x}^2$ be neighboring minority samples, a new minority instance is generated as $\mathbf{x}^{\text{new}} = \mathbf{x}^1 + r \cdot (\mathbf{x}^2 - \mathbf{x}^1)$, where $r \in [0, 1]$ is a uniformly distributed random number. In the last decade, numerous oversampling techniques have been proposed making various assumptions on the local distribution of the data, and utilizing various mathematical concepts from Voronoi-diagrams to game theory (for a detailed overview see [2]), this variety of techniques is offered by the proposed `smote-variants` package.

3. Software Framework and Implementation Details

The software is designed as a standalone Python 3.5+ package, mainly built on the machine learning functionalities of `sklearn` [6]. Oversampling techniques are implemented as separate classes providing the `sample` function as a common interface, carrying out the oversampling of datasets. As a public package, code quality is ensured by the TravisCI continuous integration system set up and unit tests covering more than 95% of the code.

The package offers two main functionalities: (a) binary and multi-class oversampling – each of the implemented oversamplers can be used with binary datasets, and 61 of them are compatible with the multi-class oversampling approach implemented; (b) model selection – the package provides high-level functions to facilitate evaluation, comparison and oversampler selection.

In the implementation, we used the same acronyms as [2] and tried to stick to the variable names and algorithm structures presented in the original papers. As many of the oversampling techniques were not specified at pseudocode level, minor algorithmic details were filled by the most meaningful steps to make the algorithms resistant to unexpected configurations of data, all these changes are documented in the sources. Finally, we mention that the package supports distributed execution in terms of the `joblib` package.

4. Illustrative Examples

In order to illustrate the use of the package, we have selected the *libras_move* dataset from the `imblearn` package, and the OUPS [2] oversam-

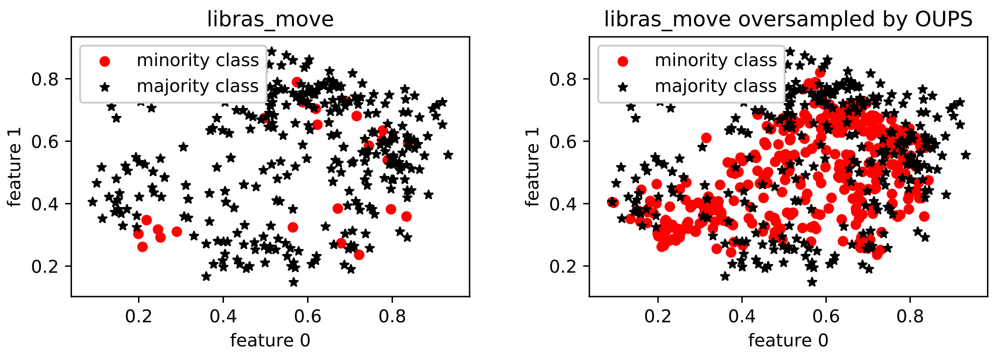

Figure 1: The first two features of the original (`libras['data']`) and oversampled (`X`) datasets – the dataset became balanced and the new minority samples are dense near the original ones

pler. By the following code segment, we illustrate the ease of oversampling using `smote-variants`, the results are plotted in Figure 1:

Listing 1: examples/007_paper_examples.py:96-101

```
import smote_variants as sv
import imblearn.datasets as imb_datasets

libras= imb_datasets.fetch_datasets()['libras_move']

X, y= sv.OUPS().sample(libras['data'], libras['target'])
```

To evaluate the performance of an oversampler on a dataset using a specific classifier, the `smote_variants.cross_validate` function carries out oversampling in each cross-validation step:

Listing 2: examples/007_paper_examples.py:120-124

```
from sklearn.neighbors import KNeighborsClassifier

results= sv.cross_validate(dataset= libras, sampler= sv.OUPS(),
                           classifier= KNeighborsClassifier())
print(results.loc['auc'])
```

The AUC (Area Under receiver-operating characteristic Curve) score is a commonly accepted measure of performance in imbalanced learning [5], the resulting score of 0.9789 compared to the score of 0.9628 without oversampling illustrates the benefits of oversampling in imbalanced learning.

5. Conclusions

The `smote-variants` package provides Python implementation for 85 binary oversampling techniques, a multi-class oversampling approach compatible with 61 of the implemented binary oversamplers, and offers various cross-validation and evaluation functionalities to facilitate the use of the package. According to our best knowledge, for 76 oversampling techniques this is the first open source implementation. The cited list of implemented algorithms, the documented source code, galleries and examples are available in the GitHub repository `http://github.com/gykovacs/smote_variants`.

6. References

[1] H. He, E. A. Gracia, Learning from imbalanced data, IEEE Transactions on Knowledge Discovery 21 (9) (2009) 1263–1284.

[2] A. Fernandez, S. Garcia, F. Herrera, N. V. Chawla, SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary, Journal of Artificial Intelligence Research 61 (2018) 863–905.

[3] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, SMOTE: Synthetic minority over-sampling technique, Journal of Artificial Intelligence Research 16 (2002) 321–357. `doi:10.1613/jair.953`.

[4] G. Lemaître, F. Nogueira, C. K. Aridas, Imbalanced-learn: A Python toolbox to tackle the curse of imbalanced datasets in machine learning, Journal of Machine Learning Research 18 (17) (2017) 1–5.

[5] M. Koziarski, M. Wozniak, CCR: A combined cleaning and resampling algorithm for imbalanced data classification, International Journal of Applied Mathematics and Computer Science 27 (2017) 727736. `doi:10.1515/amcs-2017-0050`.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

Current executable software version

Ancillary data table of the executable is provided in Table 1.

Nr.	(executable) Software metadata description	Please fill in this column
S1	Current software version	0.2.4
S2	Permanent link to executables of this version	example: `http://github.com/gykovacs/smote_variants`
S3	Legal Software License	MIT
S4	Computing platform/Operating System	Linux, OS X, Microsoft Windows, Unix-like
S5	Installation requirements & dependencies	Python 3.5+, Python packages: numpy, pandas, scikit-learn, minisom, keras, joblib
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	`http://smote-variants.readthedocs.io`
S7	Support email for questions	`gyuriofkovacs@gmail.com`

Table 1: Software metadata (optional)

Current code version

Ancillary data table of the codebase is provided in Table 2.

Nr.	**Code metadata description**	**Please fill in this column**
C1	Current code version	0.2.4
C2	Permanent link to code/repository used of this code version	`http://github.com/gykovacs/smote_variants`
C3	Legal Code License	MIT
C4	Code versioning system used	`git`
C5	Software code languages, tools, and services used	Python, scikit-learn
C6	Compilation requirements, operating environments & dependencies	Python 3.5+, Python packages: numpy, pandas, scikit-learn, minisom, keras, joblib
C7	If available Link to developer documentation/manual	`http://smote-variants.readthedocs.io`
C8	Support email for questions	`gyuriofkovacs@gmail.com`

Table 2: Code metadata (mandatory)