cluster-over-sampling: A package for clustering-based oversampling

Georgios Douzas*, Fernando Bacao
NOVA Information Management School, Universidade Nova de Lisboa

Abstract

Learning from imbalanced data is a common and challenging problem in supervised learning. Standard classifiers are designed to handle balanced class distributions. While different strategies exist to tackle this problem, methods that generate artificial data to achieve a balanced class distribution, called oversampling algorithms, are more versatile than modifications to the classification algorithms. SMOTE algorithm, the most popular oversampler, as well as any other oversampling method based on it, generates synthetic samples along line segments that join minority class instances. SMOTE addresses only the issue of between-classes imbalance. On the other hand, by clustering the input space and applying any oversampling algorithm for each resulting cluster with appropriate resampling ratio, the within-classes imbalanced issue can be addressed. This approach, implemented in the `cluster-over-sampling` Python open source project, has been shown in multiple publications, using a variety of datasets, to outperform other standard oversamplers. In this paper we describe `cluster-over-sampling` in detail and make it available to the machine learning community. An important point is that the implementation integrates effortlessly with the Scikit-Learn ecosystem. Therefore, machine learning researchers and practitioners can integrate it directly to any pre-existing work.

Keywords: Machine learning, Classification, Imbalanced learning, Oversampling, Clustering

*Postal Address: NOVA Information Management School, Campus de Campolide, 1070-312 Lisboa, Portugal, Telephone: +351 21 382 8610

Email addresses: `gdouzas@novaims.unl.pt` (Georgios Douzas), `bacao@novaims.unl.pt` (Fernando Bacao)

Code metadata	
Current code version	v0.1.1
Permanent link to code/repository used for this code version	`https://github.com/AlgoWit/cluster-over-sampling`
Legal Code License	MIT
Code versioning system used	git
Software code languages, tools, and services used	Python, Travis CI, AppVeyor, Read the Docs, Codecov, CircleCI, zenodo, Anaconda Cloud
Compilation requirements, operating environments & dependencies	Linux, Mac OS, Windows
If available Link to developer documentation/manual	`https://cluster-over-sampling.readthedocs.io/`
Support email for questions	georgios.douzas@gmail.com

Table 1: Code metadata

1. Motivation and significance

1.1. Introduction

The imbalanced learning problem describes the case where in a machine learning classification task, using datasets with binary or multi-class targets, one of the classes, called the majority class, has a significantly higher number of samples compared to the remaining classes, called the minority class(es) [1]. Learning from imbalanced data is a non-trivial problem for both academic researchers and industry practitioners that can be frequently found in multiple domains such as chemical and biochemical engineering, financial management, information technology, security, business, agriculture or emergency management [2].

A bias towards the majority class is induced when imbalanced data are used to train standard machine learning algorithms. This results in low classification accuracy, especially for the minority class(es), when the classifier is evaluated on unseen data. An important measure for the degree of data imbalance is the Imbalance Ratio (IR), defined as the ratio between the number of samples of the majority class and each of the minority classes. Using a rare disease detection task as an example, with 1% of positive cases corresponding to an $IR = \frac{0.99}{0.01} = 99$, a trivial classifier that always labels a person as healthy will score a classification accuracy of 99%. However in this case, all positive cases remain undetected. The observed values of IR are often between 100 and 100.000 [3], [4]. Figure 1 presents an example of imbalanced data in two dimensions as well as the decision boundary identified by a typical classifier when they are used as a training set.

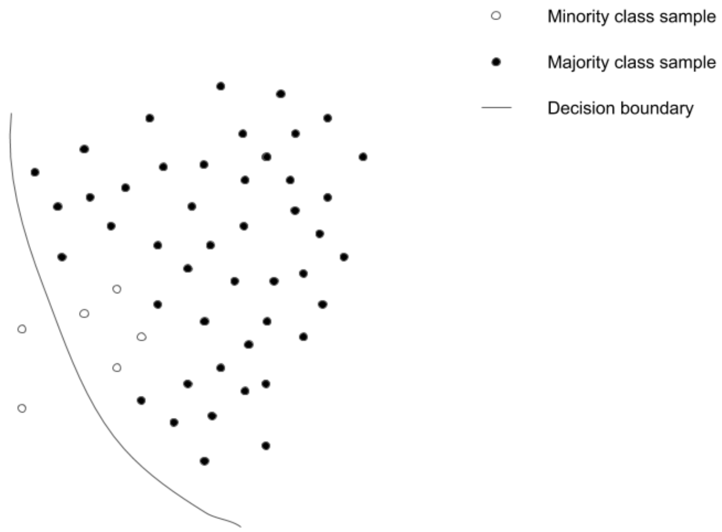

Figure 1: Imbalanced data in two dimensions. The decision boundary of a typical classifier shows a bias towards the majority class.

1.2. Oversampling algorithms

Various approaches have been proposed to improve classification results when the training data are imbalanced, a case also known as between-class imbalance. The most general approach, called oversampling, is the generation of artificial data for the minority class(es) [5]. Synthetic Minority Oversampling Technique (SMOTE) [3] was the first non-trivial oversampler proposed and remains the most popular one. Although SMOTE has been shown to be effective for generating artificial data, it also has some drawbacks [6]. In order to improve the quality of the artificial data many variants of SMOTE have been proposed. Nevertheless, they utilize the SMOTE data generation mechanism, which consists of a linear interpolation between minority class samples to generate synthetic instances as shown in figure 2.

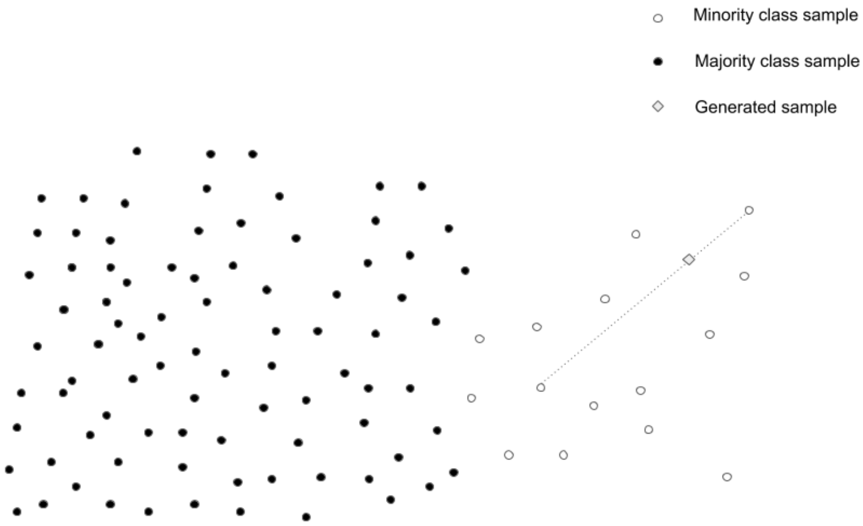

Figure 2: Visual representation of the SMOTE data generation mechanism.

A Python implementation of SMOTE and several of its variants is available in the Imbalanced-Learn [7] library, which is fully compatible with the popular machine learning toolbox Scikit-Learn [8].

1.3. Clustering-based oversampling

In addition to between-class imbalance, within-class imbalance refers to the case where areas of sparse and dense minority class instances exist. As the first step of generating synthetic samples, the SMOTE data generation mechanism selects randomly, with uniform probability, minority class instances. Consequently, dense minority class areas have a high probability of being inflated further, while the sparsely populated are likely to remain sparse. This allows to combat between-class imbalance, while the issue of within-class imbalance is ignored [9].

On the other hand, clustering-based oversampling, as presented in [10] and [11], aims to deal with both between-class and within-class imbalance problems. Initially a clustering algorithm is applied to the input space. The resulting clusters allow to identify sparse and dense minority class(es) areas. A small IR, relatively to a threshold, of a particular cluster is used as an indicator that it can be safely used as a data generation area, i.e. noise generation is avoided. Furthermore, sparse minority clusters are assigned more synthetic samples, which alleviates within-class imbalance.

Specific realizations of the above approach are SOMO [10] and KMeans-SMOTE [11] algorithms. Empirical studies have shown that both algorithms outperform SMOTE and its variants across multiple imbalanced datasets, classifiers and evaluation metrics. In this paper, we present a generic Python implementation of clustering-based oversampling, in the sense that any combination of a Scikit-Learn compatible clusterer and Imbalanced-Learn combatible oversampler can be selected to produce an algorithm that identifies clusters on the input space and applies oversampling on each one of them. In section 2, the software description is given while section 3 provides a demonstrative example of its functionalities.

2. Software description

The `cluster-over-sampling` software project is written in Python 3.7. It contains an object-oriented implementation of the clustering-based oversampling procedure as well as detailed online documentation. The implementation provides an API that is compatible with Imbalanced-Learn and Scikit-Learn libraries. Therefore, standard machine learning functionalities are supported while the generated clustering-based oversampling algorithm, composed by a clusterer and an oversampler, contains the initial oversampler functionality as a special case.

2.1. Software architecture

The `cluster-over-sampling` project contains the Python package `clover`. The main modules of `clover` are called `distribution` and `over_sampling`. The `distribution` module implements the functionality related to the distribution of the generated samples to the identified clusters, while `over_sampling` implements the functionality related to the generation of artificial samples. Both of them are presented in detail below.

2.1.1. Module `distribution`

The module `distribution` contains the files `base.py` and `density.py`. The former provides the implementation of the `BaseDistributor` class, the base class for distributors, while the later includes the `DensityDistributor` class, a generalization of the density based distributor presented in [10] and [11], that inherits from `BaseDistributor`. Following the Scikit-Learn API, `BaseDistributor` includes the public methods `fit` and `fit_distribute`. The `fit_distribute` method calls the `fit` method and returns two Python dictionaries that describe the distribution of generated samples inside each cluster and between clusters, respectively. Specifically, the `fit` method calculates various statistics related to the distribution process, while it calls `_fit` method to calculate the actual intra-cluster and inter-cluster distributions. This is achieved by invoking the `_intra_distribute` and `_inter_distribute` methods. The `BaseDistributor` class provides a trivial implementation of them, that should be overwritten when a realization of a distributor class is considered. Consequently, `DensityDistributor` overwrites both methods as well as the `_fit` method. The later calls the methods `_identify_filtered_clusters` and `_calculate_clusters_density` that identify the clusters used for data generation and calculate their density, respectively. Subsection 2.2 provides a detailed description of the initialization and functionality of the `DensityDistributor` class. Figure 3 shows a visual representation of the above classes and functions hierarchy.

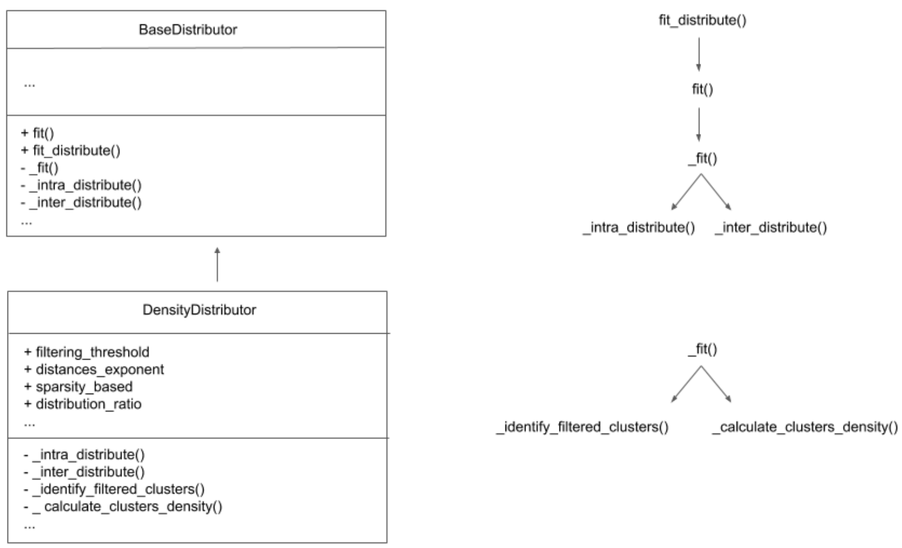

Figure 3: UML BaseDistributor and DensityDistributor class diagrams and callgraphs of main classes and methods.

2.1.2. Module `oversampling`

The module `over_sampling` contains the files `base.py` and `monkey_patching.py`. The former provides the implementation of the `BaseClusterOverSampler` class, an extension of the Imbalanced-Learn's `BaseOverSampler` class, while the later enhances the main oversamplers provided by Imbalanced-Learn with the functionality required by clustering-based oversampling. The initializer of `BaseClusterOverSampler`, compared to `BaseOverSampler`, includes the extra parameters `clusterer` and `distributor`. Also following the Imbalanced-Learn API, `BaseClusterOverSampler` includes the public methods `fit` and `fit_resample`. It also inherits from `BaseOverSampler`, the base class of oversamplers that is implemented in Imbalanced-Learn. The `fit` method calculates various statistics related to the resampling process, while the `fit_resample` method returns an enhanced version of the input data by appending the artificially generated samples. Specifically, `fit_resample` calls the `_fit_resample` method that in turn calls the `_intra_sample` and `_inter_sample` methods to generate the intra-cluster and inter-cluster artificial samples, respectively. This is achieved by invoking the `_fit_resample_cluster` method that implements the data generation mechanism. Therefore every oversampler that inherits from the `BaseClusterOverSampler` class should overwrite `_fit_resample_cluster`, providing a concrete implementation of the oversampling process. This has been done for the main oversamplers of Imbalanced-Learn through the `monkey_patching` module. Specifically, for each oversampler the `_fit_resample_cluster` method has been set equal to the `_fit_resample` method. Subsection 2.2 provides a detailed description of the initialization and functionality of the various oversamplers, enhanced by the clustering process. Figure 4 shows a visual representation of the above classes and functions hierarchy.

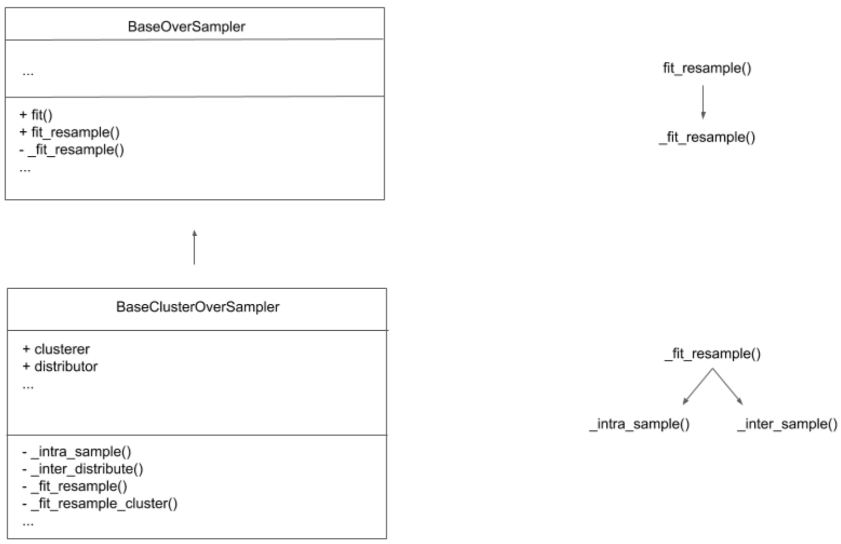

Figure 4: UML BaseOverSampler and BaseClusterOversampler class diagrams and callgraphs of main classes and methods.

2.2. Software Functionalities

As it was mentioned in subsection 2.1.2, `cluster-over-sampling` extends Imbalanced-Learn's functionality by clustering the input space before oversampling is applied. This is achieved through the implementation of the `BaseClusterOverSampler` class, an extension of Imbalanced-Learn's `BaseOverSampler` class. Oversamplers that inherit from `BaseClusterOverSampler`, compared to oversamplers inheriting from `BaseOverSampler`, require two additional initialization parameters: `clusterer` and `distributor`. Their default values are for both parameters equal to `None`, a case that corresponds to the usual oversampling procedure i.e. no clustering applied to the input space. On the other hand if the parameter `clusterer` is equal to any Scikit-Learn compatible clustering algorithm then clustering of the input space is initially applied, followed by oversampling in each cluster with the distribution of generated samples given by the `distributor` parameter. The default `distributor` value is an instance of `DensityDistributor` class as described in subsection 2.1.1. The initializer of `DensityDistributor` includes the following parameters: `filtering_threshold`, `distances_exponent`, `sparsity_based` and `distribution_factor`. The first parameter is used to identify the filtered clusters, i.e. clusters of samples that are included in the data generation process. The second parameter modifies the density calculation of the filtered clusters by increasing the effect of euclidean distances between samples. The third parameter selects whether generated samples are assigned to filtered clusters inversly proportonial to their density. Finally the last parameter adjusts the intra-cluster to inter-cluster proportion of generated samples, while it applies only to clusterers that support a neighborhood structure. Once the `DensityDistributor` object is initialized with a specific parametrization, it can be used to distribute the generated samples to the clusters identified by any clustering algorithm.

Resampling is achieved by using the two main methods of `fit` and `fit_resample` of any oversampler inheriting from `BaseClusterOverSampler`. More specifically, both of them take as input parameters the input matrix `X` and target labels `y`. Following the Scikit-Learn API, both `X`, `y` are array-like objects of appropriate shape. The first method computes various statistics which are used to resample `X`, while the second method does the same but additionally returns a resampled version of `X` and `y`.

The `cluster-over-sampling` project has been designed to integrate with the Imbalanced-Learn toolbox and the Scikit-Learn ecosystem. Therefore any oversampler that inherits from `BaseClusterOverSampler` can be used in a machine learning pipeline, through Imbalanced-Learn's class `Pipeline`, that automatically combines `samplers`, `transformers` and `estimators`. The next section provides examples of the above functionalities.

3. Illustrative examples

3.1. Basic example

An example of resampling an imbalanced dataset using the `fit_resample` method is presented in Listing 1. Initially, a binary-class imbalanced dataset is generated. Next, a `SMOTE` oversampler is initialized using `KMeans` as a clusterer. This effectively corresponds to the KMeans-SMOTE algorithm as presented in [11]. Finally, the oversampler's `fit_resample` method is used to resample the data. Printing the class distribution before and after resampling confirms that the resampled data `X_res`, `y_res` are perfectly balanced. `X_res`, `y_res` can be used as training data for any classifier in the place of `X`, `y`.

Listing 1: Resampling of imbalanced data using the `fit_resample` method of KMeans-SMOTE oversampling algorithm.

```
# Import classes and functions.
from collections import Counter
from clover.over_sampling import SMOTE
from sklearn.cluster import KMeans
from sklearn.datasets import make_classification

# Generate an imbalanced a binary class dataset.
X, y = make_classification(
    random_state=23,
        n_classes=2,
        n_features=5,
    n_samples=1000,
    weights=[0.8, 0.2]
)

# Create KMeans-SMOTE object with default hyperparameters.
kmeans_smote = SMOTE(clusterer=KMeans(random_state=4), random_state=10

# Resample the imbalanced dataset.
X_res, y_res = kmeans_smote.fit_resample(X, y)

# Print number of samples per class for initial and resampled data.
init_count = list(Counter(y).values())
resampled_count = list(Counter(y_res).values())

print(f'Initial class distribution: {init_count}.')
# Initial class distribution: [792, 208].

print(f'Resampled class distribution: {resampled_count}.')
# Resampled class distribution: [792, 792].
```

3.2. Machine learning pipeline

As mentioned before, any clustering-based oversampler can be used as a part of a machine learning pipeline. Listing 2 presents a pipeline composed by the combination of Borderline SMOTE oversampler and hierarchical clustering, a PCA tranformation and a decision tree classifier. The pipeline is trained on multi-class imbalanced data and evaluated on a hold-out set. The user applies the process in a simple way while the internal details of the calculations are hidden.

Listing 2: Training and evaluation of a machine learning pipeline that contains the AgglomerativeClustering-BorderlineSMOTE algorithm.

```python
# Import classes and functions.
from clover.over_sampling import BorderlineSMOTE
from sklearn.datasets import make_classification
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.cluster import AgglomerativeClustering
from imblearn.pipeline import make_pipeline

# Generate an imbalanced multi-class dataset.
X, y = make_classification(
        random_state=23,
        n_classes=3,
        n_informative=10,
        n_samples=500,
        weights=[0.8, 0.1, 0.1]
)

# Split the data to training and hold-out sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state

# Create the pipeline's objects with default hyperparameters.
hclusterer_bsmote = BorderlineSMOTE(clusterer=AgglomerativeClustering(
pca = PCA()
clf = DecisionTreeClassifier(random_state=3)

# Create the pipeline.
pip = make_pipeline(hclusterer_bsmote, pca, clf)
```

```
# Fit the pipeline to the training set.
pip.fit(X_train, y_train)

# Evaluate the pipeline on the hold-out set using the F-score.
test_score = f1_score(y_test, pip.predict(X_test), average='micro')

print(f'F-score on hold-out set: {test_score:.2f}.')
# F-score on hold-out set: 0.78.
```

4. Impact and conclusions

Classification of imbalanced datasets is a challenging task for standard machine learning algorithms. In addition to between-class imbalance, within-class imbalance refers to the case where areas of sparse and dense minority class instances exist. Clustering-based oversampling, aims to deal with both between-class and within-class imbalance problems.

The `cluster-over-sampling` project provides the only Python implementation, to the best of our knowledge, that provides a generic way to construct any clustering-based oversampler. A significant advantage of this implementation is that it is built on top of the Scikit-Learn's ecosystem and therefore it can be easily used in typical machine learning workflows. Also, the public API of any clustering-based oversampler is an extension of the one provided in Imbalanced-Learn. This means that users of Imbalanced-Learn and Scikit-Learn, that apply oversampling on imbalanced data, can integrate the `cluster-based-oversampler` package in their existing work in a straightforward manner.

5. Conflict of interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

References

[1] N. V. Chawla, A. Lazarevic, L. Hall, K. Boyer, SMOTEBoost: improving prediction of the minority class in boosting, Principles of Knowledge Discovery in Databases, PKDD-2003 (2003) 107–119doi:10.1007/b13634.
URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.80.1499

[2] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, G. Bing, Learning from class-imbalanced data: Review of methods and applications, Expert Systems with Applications 73 (2017) 220–239. doi:10.1016/j.eswa.2016.12.035.
URL https://doi.org/10.1016/j.eswa.2016.12.035

[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. arXiv:1106.1813, doi:10.1613/jair.953.

[4] S. Barua, M. M. Islam, X. Yao, K. Murase, MWMOTE - Majority weighted minority oversampling technique for imbalanced data set learning, IEEE Transactions on Knowledge and Data Engineering 26 (2) (2014) 405–425. doi:10.1109/TKDE.2012.232.

[5] A. Fernández, V. López, M. Galar, M. J. del Jesus, F. Herrera, Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches, Knowledge-Based Systems 42 (2013) 97–110. doi:http://dx.doi.org/10.1016/j.knosys.2013.01.018.
URL http://www.sciencedirect.com/science/article/pii/S0950705113000300

[6] H. He, E. A. Garcia, Learning from Imbalanced Data, IEEE Transactions on Knowledge and Data Engineering 21 (9) (2009) 1263–1284. arXiv:arXiv:1011.1669v3, doi:10.1109/TKDE.2008.239.

[7] G. Lemaitre, 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 (2016) 1–5. arXiv:1609.06570, doi:http://www.jmlr.org/papers/volume18/16-365/16-365.pdf.
URL http://arxiv.org/abs/1609.06570

[8] 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, Vol. 12, 2011. arXiv:arXiv:1201.0490v2, doi:10.1007/s13398-014-0173-7.2.
URL http://dl.acm.org/citation.cfm?id=2078195

[9] R. C. Prati, G. Batista, M. C. Monard, Learning with class skews and small disjuncts, in: SBIA, 2004, pp. 296–306. doi:10.1007/978-3-540-28645-5_30.

[10] G. Douzas, F. Bacao, Self-organizing map oversampling (somo) for imbalanced data set learning, Expert Systems with Applications 82 (2017) 40 – 52. doi:https://doi.org/10.1016/j.eswa.2017.03.073. URL http://www.sciencedirect.com/science/article/pii/S0957417417302324

[11] G. Douzas, F. Bacao, F. Last, Improving imbalanced learning through a heuristic oversampling method based on k-means and smote, Information Sciences 465 (2018) 1 – 20. doi:https://doi.org/10.1016/j.ins.2018.06.056. URL http://www.sciencedirect.com/science/article/pii/S0020025518304997