smote_variants Documentation

Release 0.1.0

György Kovács

Feb 03, 2020
1.1 What is smote_variants?

Imbalanced datasets are around. In fact, the synthetic oversampling of the minority class is able to improve classification performance in imbalanced learning scenarios. There are more than 85 variants of the classical Synthetic Minority Oversampling Technique (SMOTE) published, but source codes are available for only a handful of techniques. In this package we have implemented 85 variants of SMOTE in a common framework, and also supplied some model selection and evaluation codes.

In order to get an impression on what to expect, an ordinary, imbalanced, 2D dataset can be seen in the left hand side, and the oversampled dataset on the right hand side:

If you use this package, please consider citing the following papers.


BibTex for the package:
Preprint of the comparative study: https://www.researchgate.net/publication/334732374_An_empirical_comparison_and_evaluation_of_minority_oversampling_techniques_on_a_large_number_of_imbalanced_datasets

BibTex for the comparison and evaluation:

```bibtex
@article{smote-comparison,  
  author={György Kovács},  
  title={An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets},  
  journal={Applied Soft Computing},  
  note={(IF-2019=4.873)},  
  year={2019},  
  volume={83},  
  pages={105662},  
  group={journal},  
  code={https://github.com/analyticalmindsLtd/smote_variants},  
  doi={10.1016/j.asoc.2019.105662}  
}
```

1.2 Comparison and evaluation

For a thorough comparison and evaluation see https://www.researchgate.net/publication/334732374_An_empirical_comparison_and_evaluation_of_minority_oversampling_techniques_on_a_large_number_of_imbalanced_datasets

1.3 Why oversampling?

At first glance, oversampling seems to be an empirical heuristic technique to improve classification performance. After all, generating samples somehow and using them for training a classifier seems to be fairly contracitory. However, there are some reasons why the contrary is the case:

1) **Data augmentation**: sample generation is widely used on nowadays’ flagship field deep learning, but it is called data augmentation. In data augmentation additional images are generated to drive deep learning by applying various geometrical and statistical distortions like skewing or adding noise. In the image domain, it is known that these transformations shouldn’t change the useful content of the image much, but increase the variability of the training set, thus, better models can be expected. Using a general dataset, the set of transformations that keep the content unchanged is unclear, but one can expect that applying some interpolation between close data
points of a class is likely to remain in the same class. This is exactly what oversampling does, thus, it can be considered as a general analogy of the widely accepted data augmentation process.

2) *Improved regularization:* to avoid overfitting, most of the classifiers have some hyperparameters for regularization. Whether it is the coefficient of the L1 or L2 norm of weights in the objective function or it is an upper bound on the depth of decision trees, it’s goal is to reduce overfitting by making a compromise between bias and variance. Training an imbalanced dataset, most of the general purpose classifiers tend to overfit the majority class (since it has a larger contribution to the loss function, or its samples are more dense and dominate some regions in the feature space, etc.). Even though the overfitting caused by imbalance is a decent effect in the training process, we have only the regularization parameters developed for balanced datasets. It becomes extremely hard to prevent overfitting majority classes using the standard regularization parameters, especially if the degree of imbalance varies spatially. One of the basic principles of machine learning is that the goal of regularization is to fix the lack of data. If we had enough training data, there would be no need for regularization, at all. Thus, generating training data is closely related to the root of the problem, it is a kind of regularization, in which we put artificial sample points to certain positions in the feature space to articulate what we expect about the distribution of the minority class(es).

3) *Regularization by samples - a Bayesian thought:* finally, Bayesian people usually say that one important reason why Bayesian statistics is better than frequentist statistics is that in the lack of infinite data (which is usually the case), all we can derive is a distribution on any parameter of interest. And, arbitrary distributions can be represented by a bunch of samples. Analogously, in our interpretation, there is no better way to do regularization than generating artificial training sampels according to our expectations on the distribution of the data, thus, regularizing by a set of properly positioned training samples.
2.1 Prerequisites

The following packages are requirements:

- joblib
- numpy
- pandas
- sklearn
- scipy
- minisom
- keras (with any backend)

Optionally, consider installing the package `imbalanced_databases` for evaluation.

2.2 Installation

2.2.1 Install from PyPi

```bash
> pip install smote_variants
```

For testing purposes, it is recommended to install the `imbalanced_databases` package:

```bash
> pip install imbalanced_databases
```

2.2.2 Clone from GitHub

```bash
> git clone git@github.com:gykovacs/smote_variants.git
> cd smote_variants
> pip install .
```

For out of box imbalanced databases consider installing the `imbalanced_databases` package:

```bash
> git clone git@github.com:gykovacs/imbalanced_databases.git
> cd imbalanced_databases
> pip install .
```
2.2.3 Install directly from GitHub

```bash
> pip install git+https://github.com:gykovacs/smote_variants.git
```

For out of box imbalanced databases consider installing the `imbalanced_databases` package, as well:

```bash
> pip install git+https://github.com:gykovacs/imbalanced_databases.git
```
3.1 SMOTE

3.1.1 API

3.1.2 Example

```python
>>> oversampler = smote_variants.SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
3.2 SMOTE_TomekLinks

3.2.1 API

3.2.2 Example

```python
>>> oversampler = smote_variants.SMOTE_TomekLinks()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
3.2. SMOTE_TomekLinks

References:

• BibTex:

```latex
@article{smote_tomeklinks_enn,
  author = {Batista, Gustavo E. A. P. A. and Prati, Ronaldo C. and Monard, Maria Carolina},
  title = {A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data},
  journal = {SIGKDD Explor. Newsl.},
  issue_date = {June 2004},
  volume = {6},
  number = {1},
  month = jun,
  year = {2004},
  issn = {1931-0145},
  pages = {20--29},
  numpages = {10},
}
```

(continues on next page)
3.3 SMOTE_ENN

3.3.1 API

3.3.2 Example

```python
>>> oversampler= smote_variants.SMOTE_ENN()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
References:

- BibTex:

```bibtex
@article{smote_tomeklinks_enn,
  author = {Batista, Gustavo E. A. P. A. and Prati, Ronaldo C. and Monard, Maria Carolina},
  title = {A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data},
  journal = {SIGKDD Explor. Newsl.},
  issue_date = {June 2004},
  volume = {6},
  number = {1},
  month = jun,
  year = {2004},
  issn = {1931-0145},
  pages = {20--29},
  numpages = {10},
  url = {http://doi.acm.org/10.1145/1007730.1007735},
  doi = {10.1145/1007730.1007735},
  acmid = {1007735},
  publisher = {ACM},
  address = {New York, NY, USA},
}
```

Notes:

- Can remove too many of minority samples.
3.4 Borderline_SMOTE1

3.4.1 API

3.4.2 Example

```python
>>> oversampler= smote_variants.Borderline_SMOTE1()
>>> X_samp, y_samp= oversampler.sample(X, y)
```

References:

- BibTex:

```
@InProceedings{borderlineSMOTE,
    author="Han, Hui",
    ...}
```
3.5 Borderline_SMOTE2

3.5.1 API

3.5.2 Example

```python
>>> oversampler = smote_variants.Borderline_SMOTE2()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@InProceedings{borderlineSMOTE,
  author="Han, Hui and Wang, Wen-Yuan and Mao, Bing-Huan",
  editor="Huang, De-Shuang and Zhang, Xiao-Ping and Huang, Guang-Bin",
  title="Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning",
  booktitle="Advances in Intelligent Computing",
  year="2005",
  publisher="Springer Berlin Heidelberg",
  address="Berlin, Heidelberg",
  pages="878--887",
  isbn="978-3-540-31902-3"
}
```

### 3.6 ADASYN

#### 3.6.1 API

#### 3.6.2 Example

```python
>>> oversampler = smote_variants.ADA SYN()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```bibtex
@inproceedings{adasyn,
  author={He, H. and Bai, Y. and Garcia, E. A. and Li, S.},
  title={ADASYN: adaptive synthetic sampling approach for imbalanced learning},
  booktitle={Proceedings of IJCNN},
  year={2008},
  pages={1322--1328}
}
```

3.6. ADASYN
3.7 AHC

3.7.1 API

3.7.2 Example

```python
>>> oversampler = smote_variants.AHC()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

References:
- BibTex:

```latex
@article{AHC,
  title = "Learning from imbalanced data in surveillance of nosocomial infection",
}
```

(continues on next page)
3.8 LLE_SMOTE

3.8.1 API

3.8.2 Example

```python
>>> oversampler= smote_variants.LLE_SMOTE()
>>> X_samp, y_samp= oversampler.sample(X, y)

Original data sample

---

feature 0

---

feature 1

---

majority

minority
References:

- BibTex:

```bibtex
@INPROCEEDINGS{lle_smote,
  author={Wang, J. and Xu, M. and Wang, H. and Zhang, J.},
  booktitle={2006 8th international Conference on Signal Processing},
  title={Classification of Imbalanced Data by Using the SMOTE Algorithm and Locally Linear Embedding},
  year={2006},
  volume={3},
  number={},
  pages={},
  keywords={artificial intelligence;biomedical imaging;medical computing;imbalanced data classification;SMOTE algorithm;locally linear embedding;medical imaging intelligence;synthetic minority oversampling technique;high-dimensional data;low-dimensional space;Biomedical imaging;Back;Training data;Data mining;Biomedical engineering;Research and development;Electronic mail;Pattern recognition;Performance analysis;Classification algorithms},
  doi={10.1109/ICOSP.2006.345752},
  ISSN={2164-5221},
  month={Nov}}
```

Notes:

- There might be numerical issues if the nearest neighbors contain some element multiple times.
3.9 distance_SMOTE

3.9.1 API

3.9.2 Example

```python
>>> oversampler = smote_variants.distance_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

References:

- BibTex:

```latex
@INPROCEEDINGS{distance_smote,
    author={de la Calleja, J. and Fuentes, O.,}
}
```
3.10 SMMO

3.10.1 API

3.10.2 Example

```python
>>> oversampler = smote_variants.SMMO()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

• BibTex:

```latex
@InProceedings{smmo,
  author = {de la Calleja, Jorge and Puentes, Olac and González, Jesús},
  booktitle = {Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference},
  year = {2008},
  month = {01},
  pages = {276–281},
  title = {Selecting Minority Examples from Misclassified Data for Over-Sampling.}
}
```

Notes:

• In this implementation the ensemble is not specified. I have selected some very fast, basic classifiers.

• Also, it is not clear what the authors mean by “weighted distance”.

• The original technique is not prepared for the case when no minority samples are classified correctly by the ensemble.

### 3.11 polynom_fit_SMOTE

#### 3.11.1 API

#### 3.11.2 Example

```python
>>> oversampler = smote_variants.polynom_fit_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```
@INPROCEEDINGS{polynomial_fit_smote,
  author={Gazzah, S. and Amara, N. E. B.},
  booktitle={2008 The Eighth IAPR International Workshop on Document Analysis Systems},
  title={New Oversampling Approaches Based on Polynomial Fitting for Imbalanced Data Sets},
  year={2008},
  volume={},
  number={},
  pages={677-684},
  keywords={curve fitting;learning (artificial intelligence);mesh generation;pattern classification;polynomials;sampling methods;support vector machines;oversampling approach;polynomial fitting function;imbalanced data set;pattern classification task;class-modular strategy;support vector machine;true negative rate;true positive rate;bus topology;polynomial curve topology;mesh topology;Polynomials;Topology;Support vector machines;Support vector machine classification;Pattern classification;Performance evaluation;Training data;Text analysis;Data engineering;Convergence;writer identification system;majority class;minority class;imbalance;data sets;polynomial fitting functions;class-modular strategy},
```
3.12 Stefanowski

3.12.1 API

3.12.2 Example

```python
>>> oversampler = smote_variants.Stefanowski()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

Original data sample

Stefanowski: CM, NR, SCpy, BL
3.13 ADOMS

3.13.1 API

3.13.2 Example

```python
>>> oversampler = smote_variants.ADOMS()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```bibtex
@INPROCEEDINGS{adoms,
    author={Tang, S. and Chen, S.},
    booktitle={2008 International Conference on Information Technology and Applications in Biomedicine},
    title={The generation mechanism of synthetic minority class examples},
    year={2008},
    volume={},
    number={},
    pages={444-447},
    keywords={medical image processing;generation mechanism; synthetic minority class examples;class imbalance problem;medical image analysis;oversampling algorithm;Principal component analysis;Biomedical imaging;Medical diagnostic imaging;Information technology;Biomedical engineering;Noise generators;Concrete;Nearest neighbor searches;Data analysis;Image analysis},
    doi={10.1109/ITAB.2008.4570642},
    ISSN={2168-2194},
    month={May})
```

### 3.14 Safe_Level_SMOTE

#### 3.14.1 API

#### 3.14.2 Example

```python
>>> oversampler= smote_variants.Safe_Level_SMOTE()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@inproceedings{safe_level_smote,
  author = {Bunkhumpornpat, Chumphol and Sinapiromsaran, Krung and Lursinsap, Chidchanok},
  title = {Safe-Level-SMOTE: Safe-Level-Synthetic Minority Over-Sampling Technique for Handling the Class Imbalanced Problem},
  booktitle = {Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining},
  series = {PAKDD '09},
  year = {2009},
  isbn = {978-3-642-01306-5},
  location = {Bangkok, Thailand},
  pages = {475--482},
  numpages = {8},
  url = {http://dx.doi.org/10.1007/978-3-642-01307-2_43},
} (continues on next page)
```
• The original method was not prepared for the case when no minority sample has minority neighbors.

### 3.15 MSMOTE

#### 3.15.1 API

#### 3.15.2 Example

```python
>>> oversampler = smote_variants.MSMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@inproceedings{msmote,
    author = {Hu, Shengguo and Liang, Yanfeng and Ma, Lintao and He, Ying},
    title = {MSMOTE: Improving Classification Performance When Training Data is Imbalanced},
    booktitle = {Proceedings of the 2009 Second International Workshop on Computer Science and Engineering - Volume 02},
    series = {IWCSE '09},
    year = {2009},
    isbn = {978-0-7695-3881-5},
    pages = {13--17},
    numpages = {5},
    url = {https://doi.org/10.1109/WCSE.2009.756},
    doi = {10.1109/WCSE.2009.756},
    acmid = {1682710},
    publisher = {IEEE Computer Society},
    address = {Washington, DC, USA},
    keywords = {imbalanced data, over-sampling, SMOTE, AdaBoost, samples groups, SMOTEBoost},
}
```

Notes:

- The original method was not prepared for the case when all minority samples are noise.
3.16 DE_oversampling

3.16.1 API

3.16.2 Example

```python
>>> oversampler= smote_variants.DE_oversampling()
>>> X_samp, y_samp= oversampler.sample(X, y)
```

References:

- BibTex:
  ```
  @INPROCEEDINGS{de_oversampling,
  author={Chen, L. and Cai, Z. and Chen, L. and Gu, Q.},
  ```

(continues on next page)
3.17 SMOBD

3.17.1 API

3.17.2 Example

```python
>>> oversampler = smote_variants.SMOBD()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@INPROCEEDINGS{smobd,
    author={Cao, Q. and Wang, S.},
    booktitle={2011 International Conference on Information Management, Innovation Management and Industrial Engineering},
    title={Applying Over-sampling Technique Based on Data Density and Cost-sensitive SVM to Imbalanced Learning},
    year={2011},
    volume={2},
    number={},
    pages={543-548},
    keywords={data handling;learning (artificial intelligence);support vector machines;oversampling technique application;data density;cost sensitive SVM;imbalanced learning;SMOTE algorithm;data distribution;density information;Support vector machines;Classification algorithms;Noise measurement;Arrays;Noise;Algorithm design and analysis;Training;imbalanced learning;cost-sensitive SVM;SMOTE;data density;SMOBD},
    doi={10.1109/ICIII.2011.276},
    ISSN={2155-1456},
    month={Nov},}
```

3.18 SUNDO

3.18.1 API

3.18.2 Example

```python
>>> oversampler = smote_variants.SUNDO()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```bibtex
@INPROCEEDINGS{sundo,
  author={Cateni, S. and Colla, V. and Vannucci, M.},
  booktitle={2011 11th International Conference on Intelligent Systems Design and Applications},
  title={Novel resampling method for the classification of imbalanced datasets for industrial and other real-world problems},
  year={2011},
  volume={},
  number={},
  pages={402-407},
  keywords={decision trees;pattern classification;sampling;methods;support vector machines;resampling method;imbalanced dataset;classification;industrial problem;real world problem;oversampling technique;undersampling technique;support vector machine;decision tree;binary;classification;synthetic dataset;public dataset;industrial data;data;vector machines;Training;Accuracy;Databases;Intelligent systems;Breast cancer;Decision trees;oversampling;undersampling;imbalanced dataset},
} (continues on next page)
```
3.19 MSYN

3.19.1 API

3.19.2 Example

```python
>>> oversampler = smote_variants.MSYN()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

![Original data sample](image1)

![MSYN: Ex](image2)
3.20 SVM_balance

3.20.1 API

3.20.2 Example

```python
>>> oversampler= smote_variants.SVM_balance()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
References:

- BibTex:

```
@article{svm_balance,
    author = {Farquad, M.A.H. and Bose, Indranil},
    title = {Preprocessing Unbalanced Data Using Support Vector Machine},
    journal = {Decis. Support Syst.},
    issue_date = {April, 2012},
    volume = {53},
    number = {1},
    month = apr,
    year = {2012},
    issn = {0167-9236},
    pages = {226--233},
    numpages = {8},
    url = {http://dx.doi.org/10.1016/j.dss.2012.01.016},
    doi = {10.1016/j.dss.2012.01.016},
}
```
3.21 TRIM_SMOTE

3.21.1 API

3.21.2 Example

```python
>>> oversampler = smote_variants.TRIM_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@InProceedings{trim_smote,
  author="Puntumapon, Kamthorn 
  and Waiyamai, Kitsana",
  editor="Tan, Pang-Ning 
  and Chawla, Sanjay 
  and Ho, Chin Kuan 
  and Bailey, James",
  title="A Pruning-Based Approach for Searching Precise and Generalized Region for Synthetic Minority Over-Sampling",
  booktitle="Advances in Knowledge Discovery and Data Mining",
  year="2012",
  publisher="Springer Berlin Heidelberg",
  address="Berlin, Heidelberg",
  pages="371--382",
  abstract="One solution to deal with class imbalance is to modify its class distribution. Synthetic over-sampling is a well-known method to modify class distribution by generating new synthetic minority data. Synthetic Minority Over-sampling TECnique (SMOTE) is a state-of-the-art synthetic over-sampling algorithm that generates new synthetic data along the line between the minority data and their selected nearest neighbors. Advantages of SMOTE is to have decision regions larger and less specific to original data. However, its drawback is the over-generalization problem where synthetic data is generated into majority class region. Over-generalization leads to misclassify non-minority class region into minority class. To overcome the over-generalization problem, we propose an algorithm, called TRIM, to search for precise minority region while maintaining its generalization. TRIM iteratively filters out irrelevant majority data from the precise minority region. Output of the algorithm is the multiple set of seed minority data, and each individual set will be used for generating new synthetic data. Compared with state-of-the-art over-sampling algorithms, experimental results show significant performance improvement in terms of F-measure and AUC. This suggests over-generalization has a significant impact on the performance of the synthetic over-sampling method.",
  isbn="978-3-642-30220-6"
}
```

(continues on next page)
Notes:

- It is not described precisely how the filtered data is used for sample generation. The method is proposed to be a preprocessing step, and it states that it applies sample generation to each group extracted.

3.22 SMOTE_RSB

3.22.1 API

3.22.2 Example

```python
>>> oversampler = smote_variants.SMOTE_RSB()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@Article{smote_rsb,
  author="Ramentol, Enislay and Caballero, Yail'e and Bello, Rafael and Herrera, Francisco",
  title="SMOTE-RSB*: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using SMOTE and rough sets theory",
  journal="Knowledge and Information Systems",
  year="2012",
  month="Nov",
  day="01",
  volume="33",
  number="2",
  pages="245--265",
  abstract="Imbalanced data is a common problem in classification. This phenomenon is growing in importance since it appears in most real domains. It has special relevance to highly imbalanced data-sets (when the ratio between classes is high). Many techniques have been developed to tackle the problem of imbalanced training sets in supervised learning. Such techniques have been divided into two large groups: those at the algorithm level and those at the data level. Data level groups that have been emphasized are those that try to balance the training sets by reducing the larger class through the elimination of samples or increasing the smaller one by constructing new samples, known as undersampling and oversampling, respectively. This paper proposes a new hybrid method for preprocessing imbalanced data-sets through the construction of new samples, using the Synthetic Minority Oversampling Technique together with the application of an editing technique based on the Rough Set Theory and the lower approximation of a subset. The proposed method has been validated by an experimental study showing good results using C4.5 as the learning algorithm.",
  issn="0219-3116",
  doi="10.1007/s10115-011-0465-6",
}
```

(continues on next page)
Notes:

- I think the description of the algorithm in Fig 5 of the paper is not correct. The set “resultSet” is initialized with the original instances, and then the While loop in the Algorithm run until resultSet is empty, which never holds. Also, the resultSet is only extended in the loop. Our implementation is changed in the following way: we generate twice as many instances are required to balance the dataset, and repeat the loop until the number of new samples added to the training set is enough to balance the dataset.

### 3.23 ProWSyn

#### 3.23.1 API

#### 3.23.2 Example

```python
>>> oversampler= smote_variants.ProWSyn()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
References:

- BibTeX:

```latex
@InProceedings{prowsyn,
  author="Barua, Sukarna and Islam, Md. Monirul and Murase, Kazuyuki",
  editor="Pei, Jian and Tseng, Vincent S. and Cao, Longbing and Motoda, Hiroshi and Xu, Guandong",
  title="ProWSyn: Proximity Weighted Synthetic Oversampling Technique for Imbalanced Data Set Learning",
  booktitle="Advances in Knowledge Discovery and Data Mining",
  year="2013",
  publisher="Springer Berlin Heidelberg",
  address="Berlin, Heidelberg",
  pages="317--328",
  abstract="An imbalanced data set creates severe problems for the classifier as number of samples of one class (majority) is much higher than the other class (minority). Synthetic oversampling methods address this problem by generating new synthetic minority class samples. To distribute the synthetic samples effectively, recent approaches create weight values for original minority samples based on their importance and distribute synthetic samples according to weight values. However, most of the existing algorithms create inappropriate weights and in many cases, they cannot generate the required weight values for the minority samples. This results in a poor distribution of generated synthetic samples. In this respect, this paper presents a new synthetic oversampling algorithm, Proximity Weighted Synthetic Oversampling Technique (ProWSyn). Our proposed algorithm generates effective weight values for the minority data samples based on sample’s proximity information, i.e., distance from boundary which results in a proper distribution of generated synthetic samples across the minority data set. Simulation results on some real world datasets shows the effectiveness of the proposed method showing improvements in various assessment metrics such as AUC, F-measure, and G-mean.",
}
```

(continues on next page)
3.24 SL_graph_SMOTE

3.24.1 API

3.24.2 Example

```python
>>> oversampler = smote_variants.SL_graph_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

References:
3.25 NRSBoundary_SMOTE

3.25.1 API

3.25.2 Example

```python
>>> oversampler= smote_variants.NRSBoundary_SMOTE()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
3.26 LVQ_SMOTE

3.26.1 API

3.26.2 Example

```python
>>> oversampler = smote_variants.LVQ_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

• BibTex:

```latex
@inproceedings{lvq_smote,
  title={LVQ-SMOTE - Learning Vector Quantization based Synthetic Minority Over-sampling Technique for biomedical data},
  author={Munehiro Nakamura and Yusuke Kajiwara and Atsushi Otsuka and Haruhiko Kimura},
  booktitle={BioData Mining},
  year={2013}
}
```

Notes:

• This implementation is only a rough estimation of the method described in the paper. The main problem is that the paper uses many datasets to find similar patterns in the codebooks and replicate patterns appearing in other datasets to the imbalanced datasets based on their relative position compared to the codebook.
elements. What we do is clustering the minority class to extract a codebook as kmeans cluster means, then, find pairs of codebook elements which have the most similar relative position to a randomly selected pair of codebook elements, and translate nearby minority samples from the neighborhood one pair of codebook elements to the neighborhood of another pair of codebook elements.

3.27 SOI_CJ

3.27.1 API

3.27.2 Example

```python
>>> oversampler = smote_variants.SOI_CJ()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

![Original data sample](image1)

![SOI_CJ: Ex, Clus, SCmp](image2)
References:

- BibTex:

```latex
@article{soi_cj,
  author = {Sánchez, Atlántida I. and Morales, Eduardo and Gonzalez, Jesus},
  year = {2013},
  month = {01},
  pages = {},
  title = {Synthetic Oversampling of Instances Using Clustering},
  volume = {22},
  booktitle = {International Journal of Artificial Intelligence Tools}
}
```

3.28 ROSE

3.28.1 API

3.28.2 Example

```python
>>> oversampler= smote_variants.ROSE()
>>> X_samp, y_samp= oversampler.sample(X, y)
```

![Original data sample](image)

- majority
- minority
References:

- BibTex:

```biblatex
@Article{rose,
  author="Menardi, Giovanna and Torelli, Nicola",
  title="Training and assessing classification rules with imbalanced data",
  journal="Data Mining and Knowledge Discovery",
  year="2014",
  month="Jan",
  day="01",
  volume="28",
  number="1",
  pages="92--122",
  abstract="The problem of modeling binary responses by using cross-sectional data has been addressed with a number of satisfying solutions that draw on both parametric and nonparametric methods. However, there exist many real situations where one of the two responses (usually the most interesting for the analysis) is rare. It has been largely reported that this class imbalance heavily compromises the process of learning, because the model tends to focus on the prevalent class and to ignore the rare events. However, not only the estimation of the classification model is affected by a skewed distribution of the classes, but also the evaluation of its accuracy is jeopardized, because the scarcity of data leads to poor estimates of the model's accuracy. In this work, the effects of class imbalance on model training and model assessing are discussed. Moreover, a unified and systematic framework for dealing with the problem of imbalanced classification is proposed, based on a smoothed bootstrap re-sampling technique. The proposed technique is founded on a sound theoretical basis and an extensive empirical study shows that it outperforms the main other remedies to face imbalanced learning problems.",
  issn="1573-756X",
  doi="10.1007/s10618-012-0295-5",
  url="https://doi.org/10.1007/s10618-012-0295-5"
}
```

Notes:
• It is not entirely clear if the authors propose kernel density estimation or the fitting of simple multivariate Gaussians on the minority samples. The latter seems to be more likely, I implement that approach.

3.29 SMOTE_OUT

3.29.1 API

3.29.2 Example

```python
>>> oversampler = smote_variants.SMOTE_OUT()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

References:

• BibTex:
3.30 SMOTE_Cosine

3.30.1 API

3.30.2 Example

```python
>>> oversampler = smote_variants.SMOTE_Cosine()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```bibtex
@article{smote_out_smote_cosine_selected_smote,
  title={SMOTE-Out, SMOTE-Cosine, and Selected-SMOTE: An enhancement strategy to handle imbalance in data level},
  author={Fajri Koto},
  journal={2014 International Conference on Advanced Computer Science and Information System},
  year={2014},
  pages={280-284}
}
```

### 3.31 Selected_SMOTE

#### 3.31.1 API

#### 3.31.2 Example

```python
>>> oversampler= smote_variants.Selected_SMOTE()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@article{smote_out_smote_cosine_selected_smote,
  title={SMOTE-Out, SMOTE-Cosine, and Selected-SMOTE: An enhancement strategy to handle imbalance in data level},
  author={Fajri Koto},
  journal={2014 International Conference on Advanced Computer Science and Information System},
  year={2014},
  pages={280-284}
}
```

Notes:

- Significant attribute selection was not described in the paper, therefore we have implemented something meaningful.
### 3.32 LN_SMOTE

#### 3.32.1 API

#### 3.32.2 Example

```python
>>> oversampler = smote_variants.LN_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

![Original data sample](image1)

![LN_SMOTE: Ex, SCmp](image2)

References:
- BibTex:
  ```
  @INPROCEEDINGS{ln_smote,
  author={Maciejewski, T. and Stefanowski, J.},
  ```
3.33 MWMOTE

3.33.1 API

3.33.2 Example

```python
>>> oversampler = smote_variants.MWMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```bib
@ARTICLE{mwmote,
  author={Barua, S. and Islam, M. M. and Yao, X. and Murase, K.},
  journal={IEEE Transactions on Knowledge and Data Engineering},
  title={MWMOTE--Majority Weighted Minority Oversampling Technique for Imbalanced Data Set Learning},
  year={2014},
  volume={26},
  number={2},
  pages={405-425},
  keywords={learning (artificial intelligence);pattern clustering;geometric mean; minority class cluster; clustering approach; weighted; informative minority class samples; Euclidean distance; hard-to-learn; informative minority class samples; majority class; synthetic minority class; samples; synthetic oversampling methods; imbalanced learning problems; imbalanced data set learning; MWMOTE--majority weighted minority oversampling technique; Sampling methods; Noise measurement; Boosting; Simulation; Complexity; theory; Interpolation; Abstracts; Imbalanced learning; undersampling; oversampling; synthetic sample generation; clustering},
  doi={10.1109/TKDE.2012.232},
  ISSN={1041-4347},
  month={Feb})
```

Notes:

- The original method was not prepared for the case of having clusters of 1 elements.
3.34 PDFOS

3.34.1 API

3.34.2 Example

```python
>>> oversampler = smote_variants.PDFOS()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

---

References:

- BibTex:

```latex
@article{pdfos,
  title = "PDFOS: PDF estimation based over-sampling for imbalanced two-class problems",
}
```

(continues on next page)
Notes:

- Not prepared for low-rank data.

### 3.35 IPADE_ID

#### 3.35.1 API

#### 3.35.2 Example

```python
>>> oversampler= smote_variants.IPADE_ID()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@article{ipade_id,
  title = "Addressing imbalanced classification with instance generation techniques: IPADE-ID",
  journal = "Neurocomputing",
  volume = "126",
  pages = "15 - 28",
  year = "2014",
  note = "Recent trends in Intelligent Data Analysis Online Data Processing",
  issn = "0925-2312",
  doi = "https://doi.org/10.1016/j.neucom.2013.01.050",
  author = "Victoria López and Isaac Triguero and Cristóbal J. Carmona and Salvador García and Francisco Herrera",
  keywords = "Differential evolution, Instance generation, Nearest neighbor, Decision tree, Imbalanced datasets"
}
```

Notes:

- According to the algorithm, if the addition of a majority sample doesn’t improve the AUC during the DE optimization process, the addition of no further majority points is tried.

- In the differential evolution the multiplication by a random number seems have a deteriorating effect, new scaling parameter added to fix this.

- It is not specified how to do the evaluation.
3.36 RWO_sampling

3.36.1 API

3.36.2 Example

```python
>>> oversampler = smote_variants.RWO_sampling()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

![Original data sample](image1)

![RWO_sampling: Ex](image2)

References:

- BibTex:

```latex
@article{rwo_sampling,
  author = {Zhang, Huaxzhang and Li, Mingfang},
  (continues on next page)
```
3.37 NEATER

3.37.1 API

3.37.2 Example

```python
>>> oversampler = smote_variants.NEATER()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

• BibTex:

```latex
@INPROCEEDINGS{neater,
  author={Almogahed, B. A. and Kakadiaris, I. A.},
  booktitle={2014 22nd International Conference on Pattern Recognition},
  title={NEATER: Filtering of Over-sampled Data Using Non-cooperative Game Theory},
  year={2014},
  volume={},
  number={},
  pages={1371-1376},
  keywords={data handling;game theory;information filtering;NEATER;imbalanced data problem;synthetic data;filtering of over-sampled data using non-cooperative game theory;Games;Game theory;Vectors;Sociology;Statistics;Silicon;Mathematical model},
  doi={10.1109/ICPR.2014.245},
  ISSN={1051-4651},
  month={Aug}}
```

Notes:

• Evolving both majority and minority probabilities as nothing ensures that the probabilities remain in the range [0,1], and they need to be normalized.

• The inversely weighted function needs to be cut at some value (like the alpha level), otherwise it will overemphasize the utility of having differing neighbors next to each other.
3.38 DEAGO

3.38.1 API

3.38.2 Example

```python
>>> oversampler = smote_variants.DEAGO()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

References:

- BibTex:
  ```
  @INPROCEEDINGS{deago,
  author={Bellinger, C. and Japkowicz, N. and Drummond, C.},
  ```
Notes:

- There is no hint on the activation functions and amounts of noise.

### 3.39 Gazzah

#### 3.39.1 API

#### 3.39.2 Example

```python
>>> oversampler = smote_variants.Gazzah()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

![Original data sample](image)
References:

- BibTex:

```bibtex
@INPROCEEDINGS{gazzah,
  author={Gazzah, S. and Heckel, A. and Essoukri Ben Amara, N.},
  booktitle={2015 IEEE 12th International Multi-Conference on Systems, Signals Devices (SSD15)},
  title={A hybrid sampling method for imbalanced data},
  year={2015},
  volume={},
  number={},
  pages={1-6},
  keywords={computer vision;image classification;learning (artificial intelligence);sampling methods;hybrid sampling method; imbalanced data;diversification;computer vision domain;classical machine learning systems;intraclass variations;system performances;classification accuracy;imbanced training data;training data set;over-sampling;minority class;SMOTE star topology;feature vector deletion;intraclass variations; distribution criterion;biometric data;true positive rate;Training data; Principal component analysis;Databases;Support vector machines;Training; Feature extraction;Correlation;Imbalanced data sets;Intraclass variations; Data analysis;Principal component analysis;One-against-all SVM),
  doi={10.1109/SSD.2015.7348093},
  ISSN={},
  month={March}}
```
3.40 MCT

3.40.1 API

3.40.2 Example

```python
>>> oversampler = smote_variants.MCT()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

References:

- BibTex:

```latex
@article{mct,
  author = {Jiang, Liangxiao \textbf{and} Qiu, Chen \textbf{and} Li, Chaoqun},
  (continues on next page)
```

---

3.40. MCT 65
Notes:

- Mode is changed to median, distance is changed to Euclidean to support continuous features, and normalized.

### 3.41 ADG

#### 3.41.1 API

#### 3.41.2 Example

```python
>>> oversampler = smote_variants.ADG()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@article{adg,
    author = {Pourhabib, A. and Mallick, Bani K. and Ding, Yu},
    year = {2015},
    month = {16},
    pages = {2695--2724},
    title = {A Novel Minority Cloning Technique for Cost-Sensitive Learning},
    volume = {16},
    journal = {Journal of Machine Learning Research}
}
```

Notes:

- This method has a lot of parameters, it becomes fairly hard to cross-validate thoroughly.
- Fails if matrix is singular when computing alpha_star, fixed by PCA.
- Singularity might be caused by repeating samples.
- Maintaining the kernel matrix becomes unfeasible above a couple of thousand vectors.

### 3.42 SMOTE_IPF

#### 3.42.1 API

#### 3.42.2 Example

```python
>>> oversampler = smote_variants.SMOTE_IPF()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```bibtex
@article{smote_ipf,
  title = "SMOTE-IPF: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering",
  journal = "Information Sciences",
  volume = "291",
  pages = "184 - 203",
  year = "2015",
  issn = "0020-0255",
  doi = "https://doi.org/10.1016/j.ins.2014.08.051",
  author = "José A. Sáez and Julián Luengo and Jerzy Stefanowski and Francisco Herrera",
}
```

(continues on next page)
3.43 KernelADASYN

3.43.1 API

3.43.2 Example

```python
>>> oversampler = smote_variants.KernelADASYN()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

![Original data sample](image1)

![KernelADASYN: DE, Ex, BL](image2)
3.44 MOT2LD

3.44.1 API

3.44.2 Example

```python
>>> oversampler = smote_variants.MOT2LD()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```bibtex
@InProceedings{mot2ld,
  author = "Xie, Zhipeng \and Jiang, Liyang \and Ye, Tengju \and Li, Xiaoli",
  editor = "Renz, Matthias \and Shahabi, Cyrus \and Zhou, Xiaofang \and Cheema, Muhammad Aamir",
  title = "A Synthetic Minority Oversampling Method Based on Local Densities in Low-Dimensional Space for Imbalanced Learning",
  booktitle = "Database Systems for Advanced Applications",
  year = "2015",
  publisher = "Springer International Publishing",
}
```

(continues on next page)
Imbalanced class distribution is a challenging problem in many real-life classification problems. Existing synthetic oversampling do suffer from the curse of dimensionality because they rely heavily on Euclidean distance. This paper proposed a new method, called Minority Oversampling Technique based on Local Densities in Low-Dimensional Space (or MOT2LD in short). MOT2LD first maps each training sample into a low-dimensional space, and makes clustering of their low-dimensional representations. It then assigns weight to each minority sample as the product of two quantities: local minority density and local majority count, indicating its importance of sampling. The synthetic minority class samples are generated inside some minority cluster. MOT2LD has been evaluated on 15 real-world data sets. The experimental results have shown that our method outperforms some other existing methods including SMOTE, Borderline-SMOTE, ADASYN, and MWMOTE, in terms of G-mean and F-measure.

Notes:

- Clusters might contain 1 elements, and all points can be filtered as noise.
- Clusters might contain 0 elements as well, if all points are filtered as noise.
- The entire clustering can become empty.
- TSNE is very slow when the number of instances is over a couple of 1000

### 3.45 V_SYNTH

#### 3.45.1 API

#### 3.45.2 Example

```python
>>> oversampler = smote_variants.V_SYNTH()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@article{v_synth,
    author = {Young, Ii, William A. and Nykl, Scott L. and Weckman, Gary and Chelberg, David M.},
    title = {Using Voronoi Diagrams to Improve Classification Performances when Modeling Imbalanced Datasets},
    journal = {Neural Comput. Appl.},
    issue_date = {July 2015},
    volume = {26},
    number = {5},
    month = jul,
    year = {2015},
    issn = {0941-0643},
    pages = {1041--1054},
    numpages = {14},
}
```

(continues on next page)
Notes:

- The proposed encompassing bounding box generation is incorrect.
- Voronoi diagram generation in high dimensional spaces is unstable

3.46 OUPS

3.46.1 API

3.46.2 Example

```python
>>> oversampler = smote_variants.OUPS()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```bibtex
@article{oups,
    title = "A priori synthetic over-sampling methods for increasing classification sensitivity in imbalanced data sets",
    journal = "Expert Systems with Applications",
    volume = "66",
    pages = "124 - 135",
    year = "2016",
    issn = "0957-4174",
    doi = "https://doi.org/10.1016/j.eswa.2016.09.010",
    author = "William A. Rivera and Petros Xanthopoulos",
    keywords = "SMOTE, OUPS, Class imbalance, Classification"
}
```

Notes:

- In the description of the algorithm a fractional number $p(j)$ is used to index a vector.

### 3.47 SMOTE_D

#### 3.47.1 API

#### 3.47.2 Example

```python
>>> oversampler= smote_variants.SMOTE_D()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
References:

- BibTex:

```
@InProceedings{smote_d,
    author="Torres, Fredy Rodriguez and Carrasco-Ochoa, Jesus A. and Martinez-Trinidad, Jose Fco.,",
    editor="Martinez-Trinidad, Jesus Ariel and Ayala Ramirez, Victor and Olivera-Lopez, Jose Arturo and Jiang, Xiaoyi",
    title="SMOTE-D: a Deterministic Version of SMOTE",
    booktitle="Pattern Recognition",
    year="2016",
    publisher="Springer International Publishing",
    address="Cham",
}
```
Imbalanced data is a problem of current research interest. This problem arises when the number of objects in a class is much lower than in other classes. In order to address this problem several methods for oversampling the minority class have been proposed. Oversampling methods generate synthetic objects for the minority class in order to balance the amount of objects between classes, among them, SMOTE is one of the most successful and well-known methods. In this paper, we introduce a modification of SMOTE which deterministically generates synthetic objects for the minority class. Our proposed method eliminates the random component of SMOTE and generates different amount of synthetic objects for each object of the minority class. An experimental comparison of the proposed method against SMOTE in standard imbalanced datasets is provided. The experimental results show an improvement of our proposed method regarding SMOTE, in terms of F-measure.

Notes:
- Copying happens if two points are the neighbors of each other.

### 3.48 SMOTE_PSO

#### 3.48.1 API

#### 3.48.2 Example

```python
>>> oversampler = smote_variants.SMOTE_PSO()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

![Original data sample](image-url)
References:

- BibTex:

```latex
@article{smote_pso,
  title = "PSO-based method for SVM classification on skewed data sets",
  journal = "Neurocomputing",
  volume = "228",
  pages = "187 - 197",
  year = "2017",
  note = "Advanced Intelligent Computing: Theory and Applications",
  issn = "0925-2312",
  doi = "https://doi.org/10.1016/j.neucom.2016.10.041",
  author = "Jair Cervantes and Farid Garcia-Lamont and Lisbeth Rodriguez and Asdrúbal López and José Ruiz Castilla and Adrian Trueba",
  keywords = "Skew data sets, SVM, Hybrid algorithms"
}
```

Notes:

- I find the description of the technique a bit confusing, especially on the bounds of the search space of velocities and positions. Equations 15 and 16 specify the lower and upper bounds, the lower bound is in fact a vector while the upper bound is a distance. I tried to implement something meaningful.

- I also find the setting of accelerating constant 2.0 strange, most of the time the velocity will be bounded due to this choice.

- Also, training and predicting probabilities with a non-linear SVM as the evaluation function becomes fairly expensive when the number of training vectors reaches a couple of thousands. To reduce computational burden, minority and majority vectors far from the other class are removed to reduce the size of both classes to a maximum of 500 samples. Generally, this shouldn’t really affect the results as the technique focuses on the samples near the class boundaries.
3.49 CURE_SMOTE

3.49.1 API

3.49.2 Example

```python
>>> oversampler= smote_variants.CURE_SMOTE()
>>> X_samp, y_samp= oversampler.sample(X, y)
```

References:

- BibTex:

```latex
@Article{cure_smote,
  author="Ma, Li",
  (continues on next page)
}
```
and Fan, Suohai", 
title="CURE-SMOTE algorithm and hybrid algorithm for feature selection and parameter optimization based on random forests",
journal="BMC Bioinformatics",
year="2017",
month="Mar",
day="14",
volume="18",
number="1",
pages="169",
abstract="The random forests algorithm is a type of classifier with prominent universality, a wide application range, and robustness for avoiding overfitting. But there are still some drawbacks to random forests. Therefore, to improve the performance of random forests, this paper seeks to improve imbalanced data processing, feature selection and parameter optimization.",
issn="1471-2105",
doi="10.1186/s12859-017-1578-z",
url="https://doi.org/10.1186/s12859-017-1578-z"
}

Notes:

- It is not specified how to determine the cluster with the “slowest growth rate”
- All clusters can be removed as noise.

3.50 SOMO

3.50.1 API

3.50.2 Example

```python
>>> oversampler= smote_variants.SOMO()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@article{somo,
  title = "Self-Organizing Map Oversampling (SOMO) for imbalanced data set learning",
  journal = "Expert Systems with Applications",
  volume = "82",
  pages = "40 - 52",
  year = "2017",
  issn = "0957-4174",
  doi = "https://doi.org/10.1016/j.eswa.2017.03.073",
  author = "Georgios Douzas and Fernando Bacao"
}
```
Notes:

- It is not specified how to handle those cases when a cluster contains 1 minority samples, the mean of within-cluster distances is set to 100 in these cases.

### 3.51 ISOMAP_Hybrid

#### 3.51.1 API

#### 3.51.2 Example

```python
>>> oversampler = smote_variants.ISOMAP_Hybrid()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

Original data sample

![Original data sample](image)

ISOMAP_Hybrid: Ex, NR, DR, CM

![ISOMAP_Hybrid: Ex, NR, DR, CM](image)

References:
3.52 CE_SMOTE

3.52.1 API

3.52.2 Example

```python
>>> oversampler = smote_variants.CE_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

![Original data sample](image-url)

- majority
- minority
References:

- BibTex:

```bibtex
@INPROCEEDINGS{ce_smote,
  author={Chen, S. and Guo, G. and Chen, L.},
  booktitle={2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops},
  title={A New Over-Sampling Method Based on Cluster Ensembles},
  year={2010},
  volume={},
  number={},
  pages={599-604},
  keywords={data mining;Internet;pattern classification;pattern clustering;over sampling method;cluster ensembles;classification;method;imbalance data handling;CE-SMOTE;clustering consistency index;cluster boundary minority samples;imbalance public data set;Mathematics;Computer science;Electronic mail;Accuracy;Nearest neighbor searches;Application software;Data mining;Conferences;Web sites;Information retrieval;classification;imbalance data sets;cluster ensembles;over-sampling},
  doi={10.1109/WAINA.2010.40},
  ISSN={},
  month={April})
```

## 3.53 Edge_Det_SMOTE

### 3.53.1 API

### 3.53.2 Example

```python
>>> oversampler = smote_variants.Edge_Det_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```
@INPROCEEDINGS{Edge_Det_SMOTE,
    author={Kang, Y. and Won, S.},
    booktitle={ICCAS 2010},
    title={Weight decision algorithm for oversampling technique on class-imbalanced learning},
    year={2010},
    volume={},
    number={},
    pages={182-186},
    keywords={edge detection;learning (artificial intelligence);weight decision algorithm;oversampling technique;class-imbalanced learning;class imbalanced data problem;edge detection algorithm;spatial space;representation;Classification algorithms;Image edge detection;Training;Noise measurement;Glass;Training data;Machine learning;Imbalanced learning;Classification;Weight decision;Oversampling;Edge detection),
```

(continues on next page)
Notes:

- This technique is very loosely specified.

### 3.54 CBSO

#### 3.54.1 API

#### 3.54.2 Example

```python
>>> oversampler = smote_variants.CBSO()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@InProceedings{cbso,
  author="Barua, Sukarna and Islam, Md. Monirul and Murase, Kazuyuki",
  editor="Lu, Bao-Liang and Zhang, Liqing and Kwok, James",
  title="A Novel Synthetic Minority Oversampling Technique for Imbalanced Data Set Learning",
  booktitle="Neural Information Processing",
  year="2011",
  publisher="Springer Berlin Heidelberg",
  address="Berlin, Heidelberg",
  pages="735--744",
  abstract="Imbalanced data sets contain an unequal distribution of data samples among the classes and pose a challenge to the learning algorithms as it becomes hard to learn the minority class concepts. Synthetic oversampling techniques address this problem by creating synthetic minority samples to balance the data set. However, most of these techniques may create wrong synthetic minority samples which fall inside majority regions. In this respect, this paper presents a novel Cluster-Based Synthetic Oversampling (CBSO) algorithm. CBSO adopts its basic idea from existing synthetic oversampling techniques and incorporates unsupervised clustering in its synthetic data generation mechanism. CBSO ensures that synthetic samples created via this method always lie inside minority regions and thus, avoids any wrong synthetic sample creation. Simulation analyses on some real world datasets show the effectiveness of CBSO showing improvements in various assessment metrics such as overall accuracy, F-measure, and G-mean.",
  isbn="978-3-642-24958-7"
}
```

Notes:

- Clusters containing 1 element induce cloning of samples.
### 3.55 E_SMOTE

#### 3.55.1 API

#### 3.55.2 Example

```python
>>> oversampler = smote_variants.E_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

![Original data sample](image1.png)

![E_SMOTE: Ex, DR, M, CM](image2.png)

References:

- BibTex:

```latex
@INPROCEEDINGS{e_smote,
    author={Deepa, T. and Punithavalli, M.},
    (continues on next page)
```
Notes:

- This technique is basically unreproducible. I try to implement something following the idea of applying some simple genetic algorithm for optimization.
- In my best understanding, the technique uses evolutionary algorithms to for feature selection and then applies vanilla SMOTE on the selected features only.

### 3.56 DBSMOTE

#### 3.56.1 API

#### 3.56.2 Example

```python
>>> oversampler = smote_variants.DBSMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
A dataset exhibits the class imbalance problem when a target class has a very small number of instances relative to other classes. A trivial classifier typically fails to detect a minority class due to its extremely low incidence rate. In this paper, a new oversampling technique called DBSMOTE is proposed. Our technique relies on a density-based notion of clusters and is designed to oversample an arbitrarily shaped cluster discovered by DBSCAN. DBSMOTE generates synthetic instances along a shortest path from each positive instance to a pseudo-centroid of a minority-class cluster. Consequently, these synthetic instances are dense near this centroid and are sparse far from this centroid. Our experimental results show that DBSMOTE improves precision, F-value, and AUC more effectively than SMOTE, Borderline-SMOTE, and Safe-Level-SMOTE for imbalanced datasets.
Notes:

- Standardization is needed to use absolute eps values.
- The clustering is likely to identify all instances as noise, fixed by recursive call with increasing eps.

### 3.57 ASMOBD

#### 3.57.1 API

#### 3.57.2 Example

```python
>>> oversampler= smote_variants.ASMOBD()
>>> X_samp, y_samp= oversampler.sample(X, y)
```

![Original data sample](image.png)
References:

- BibTex:

```latex
@INPROCEEDINGS{asmobd,
  author={Senzhang Wang and Zhoujun Li and Wenhan Chao and Qinghua Cao},
  booktitle={The 2012 International Joint Conference on Neural Networks (IJCNN)},
  title={Applying adaptive over-sampling technique based on data density and cost-sensitive SVM to imbalanced learning},
  year={2012},
  volume={},
  number={},
  pages={1-8},
  keywords={data analysis;learning (artificial intelligence);sampling methods;smoothing methods;support vector machines;adaptive over-sampling technique;cost-sensitive SVM;imbalanced learning;resampling method;data density information;overfitting;minority sample;learning difficulty;decision region;over generalization;smoothing method;cost-sensitive learning;UCI dataset;G-mean of;receiver operation curve;Smoothing methods;Noise;Support vector machines;Classification algorithms;Interpolation;Measurement;Algorithm design and analysis;over-sampling;Cost-sensitive SVM;imbalanced learning},
  doi={10.1109/IJCNN.2012.6252696},
  ISSN={2161-4407},
  month={June}}
```

Notes:

- In order to use absolute thresholds, the data is standardized.
- The technique has many parameters, not easy to find the right combination.
3.58 Assembled_SMOTE

3.58.1 API

3.58.2 Example

```python
>>> oversampler = smote_variants.Assembled_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

References:
- BibTex:
  ```
  @INPROCEEDINGS{assembled_smote,
  author={Zhou, B. and Yang, C. and Guo, H. and Hu, J.},
  (continues on next page)
  ```
3.59 SDSMOTE

3.59.1 API

3.59.2 Example

```python
>>> oversampler = smote_variants.SDSMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@INPROCEEDINGS{sdsmote,
  author={Li, K. and Zhang, W. and Lu, Q. and Fang, X.},
  booktitle={2014 International Conference on Identification, Information and Knowledge in the Internet of Things},
  title={An Improved SMOTE Imbalanced Data Classification Method Based on Support Degree},
  year={2014},
  volume={},
  number={},
  pages={34-38},
  keywords={data mining;pattern classification;sampling methods;improved SMOTE imbalanced data classification method;support degree;data mining;class distribution;imbalanced data-set classification;over sampling;method;minority class sample generation;minority class sample selection;training;bagging;computers;testing;algorithm design and analysis;data mining;imbalanced data-sets;classification;boundary sample;support degree;SMOTE},
}
```
3.60 DSMOTE

3.60.1 API

3.60.2 Example

```python
>>> oversampler = smote_variants.DSMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```
@INPROCEEDINGS{dsmote,
    author={Mahmoudi, S. and Moradi, P. and Akhlaghian, F. and Moradi, R.},
    booktitle={2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)},
    title={Diversity and separable metrics in over-sampling technique for imbalanced data classification},
    year={2014},
    volume={},
    number={},
    pages={152-158},
    keywords={learning (artificial intelligence);pattern classification;sampling methods;diversity metric;separable metric;over-sampling technique;imbalanced data classification;class distribution;techniques;under-sampling technique;DSMOTE method;imbalanced learning problem;diversity measure;separable measure;Iran University of Medical Science;UCI dataset;Accuracy;Classification algorithms;Vectors;Educational institutions;Euclidean distance;Data mining;Diversity measure;Separable Measure;Over-Sampling;Imbalanced Data;Classification problems},
    doi={10.1109/ICCKE.2014.6993409},
    ISSN={},
    month={Oct}}
```

Notes:

- The method is highly inefficient when the number of minority samples is high, time complexity is $O(n^3)$, with 1000 minority samples it takes about $1e9$ objective function evaluations to find 1 new sample point. Adding 1000 samples would take about $1e12$ evaluations of the objective function, which is unfeasible. We introduce a new parameter, $n_{\text{step}}$, and during the search for the new sample at most $n_{\text{step}}$ combinations of minority samples are tried.

- Abnormality of minority points is defined in the paper as $D_{\text{maj}}/D_{\text{min}}$, high abnormality means that the minority point is close to other minority points and very far from majority points. This is definitely not abnormality, I have implemented the opposite.

- Nothing ensures that the fisher statistics and the variance from the geometric mean remain comparable, which might skew the optimization towards one of the sub-objectives.

- MinMax normalization doesn’t work, each attribute will have a 0 value, which will make the geometric mean of all attribute 0.

## 3.61 G_SMOTE

### 3.61.1 API

### 3.61.2 Example

```python
>>> oversampler= smote_variants.G_SMOTE()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
References:

- BibTex:

```bibtex
@INPROCEEDINGS{g_smote,
    author={Sandhan, T. and Choi, J. Y.},
    booktitle={2014 22nd International Conference on Pattern Recognition},
    title={Handling Imbalanced Datasets by Partially Guided Hybrid Sampling for Pattern Recognition},
    year={2014},
    volume={},
    number={},
    pages={1449-1453},
    keywords={Gaussian processes;learning (artificial intelligence);pattern classification;regression analysis;sampling methods;support vector machines;imbalanced datasets;partially guided hybrid;rebalancing;learning algorithm;extremely low minority class samples;classification tasks;extracted hidden patterns;support vector machine;logistic regression;nearest neighbor;Gaussian process classifier;Support vector machines;Proteins;Pattern recognition;Kernel;Databases;Gaussian processes;Vectors;Imbalanced dataset;protein classification;ensemble;classifier;bootstrapping;Sat-image classification;medical diagnoses},
} (continues on next page)
```
Notes:

- the non-linear approach is inefficient

3.62 NT_SMOTE

3.62.1 API

3.62.2 Example

```python
>>> oversampler = smote_variants.NT_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

Original data sample

![Image of original data sample]
References:

- BibTex:

```bibtex
@INPROCEEDINGS{nt_smote,
  author={Xu, Y. H. \textbf{and} Li, H. \textbf{and} Le, L. P. \textbf{and} Tian, X. Y.},
  booktitle={2014 Seventh International Joint Conference on Computational Sciences \textbf{and} Optimization},
  title={Neighborhood Triangular Synthetic Minority Over-sampling Technique \textbf{for} Imbalanced Prediction on Small Samples of Chinese Tourism \textbf{and} Hospitality Firms},
  year={2014},
  volume={},
  number={},
  pages={534-538},
  keywords={financial management;pattern classification;risk management;sampling methods;travel industry;Chinese tourism;hospitality firms;imbalance risk prediction;minority \textbf{class samples};up-sampling approach;neighborhood triangular synthetic minority over-sampling technique;NT-SMOTE;nearest neighbor idea;triangular area sampling idea;single classifiers;data excavation principles;hospitality industry;missing financial indicators;financial data filtering;financial risk prediction;MDA;DT;LSVM;logit;probit;firm risk prediction;Joints;Optimization;imbalanced datasets;NT-SMOTE;neighborhood triangular;random sampling},
  doi={10.1109/CSO.2014.104},
  ISSN={},
  month={July}}
```
3.63 Lee

3.63.1 API

3.63.2 Example

```python
>>> oversampler = smote_variants.Lee()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

References:
- BibTex:
  ```
  @inproceedings{lee,
  author = {Lee, Jaedong and Kim, Noo-ri and Lee, Jee-Hyong},
  (continues on next page)
  ```
3.64 SPY

3.64.1 API

3.64.2 Example

```python
>>> oversampler = smote_variants.SPY()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```plaintext
@INPROCEEDINGS{spy,
    author={Dang, X. T. and Tran, D. H. and Hirose, O. and Satou, K.},
    booktitle={2015 Seventh International Conference on Knowledge and Systems Engineering (KSE)},
    title={SPY: A Novel Resampling Method for Improving Classification Performance in Imbalanced Data},
    year={2015},
    volume={},
    number={},
    pages={280-285},
    keywords={decision making;learning (artificial intelligence);pattern classification;sampling methods;SPY;resampling method;decision-making process;biomedical data classification;class imbalance learning;method;SMOTE;oversampling method;UCI machine learning repository;G-mean;value;borderline-SMOTE;safe-level-SMOTE;Support vector machines;Training;Bioinformatics;Proteins;Protein engineering;Radio frequency;Sensitivity;Imbalanced dataset;Over-sampling;Under-sampling;SMOTE;borderline-SMOTE},
    doi={10.1109/KSE.2015.24},
    ISSN={},
    month={Oct})
```
References:

- BibTex:

```latex
@INPROCEEDINGS{smote_psobat,  
  author={Li, J. and Fong, S. and Zhuang, Y.},  
  booktitle={2015 3rd International Symposium on Computational},  
  and Business Intelligence (ISCBI)},  
  title={Optimizing SMOTE by Metaheuristics with Neural Network},  
  and Decision Tree},  
  year={2015},  
  volume={},  
  number={},  
  pages={26-32},  
  keywords={data mining;particle swarm optimisation;pattern,  
  classification;data mining;classifier;metaheuristics;SMOTE parameters;  
  performance indicators;selection optimization;PSO;particle swarm,  
  optimization algorithm;BAT;bat-inspired algorithm;metaheuristic,  
  optimization algorithms;nearest neighbors;imbalanced dataset;smote,  
  synthetic minority over-sampling technique;decision tree;neural network;  
  classification algorithms;neural networks;decision tree;training;Optimization;Particle swarm optimization;Data mining;SMOTE;Swarm,  
  Intelligence;parameter selection optimization},
```

(continues on next page)
3.66 MDO

3.66.1 API

3.66.2 Example

```python
g>>> oversampler = smote_variants.MDO()
g>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@ARTICLE{mdo,
    author={Abdi, L. and Hashemi, S.},
    journal={IEEE Transactions on Knowledge and Data Engineering},
    title={To Combat Multi-Class Imbalanced Problems by Means of Over-
    \rightarrow Sampling Techniques},
    year={2016},
    volume={28},
    number={1},
    pages={238-251},
    keywords={covariance analysis;learning (artificial intelligence);-
    \rightarrow modelling;pattern classification;sampling methods;statistical distributions;-
    \rightarrow minority class instance modelling;probability contour;covariance structure;-
    \rightarrow MDO;Mahalanobis distance-based oversampling technique;data-oriented_
    \rightarrow technique;model-oriented solution;machine learning algorithm;data skewness;-
    \rightarrow multiclass imbalanced problem;Mathematical model;Training;Accuracy;-
    \rightarrow Eigenvalues and eigenfunctions;Machine learning algorithms;Algorithm design;-
    \rightarrow and analysis;Benchmark testing;Multi-class imbalance problems;over-sampling_
    \rightarrow techniques;Mahalanobis distance;Multi-class imbalance problems;over-
    \rightarrow sampling techniques;Mahalanobis distance},
    doi={10.1109/TKDE.2015.2458858},
    ISSN={1041-4347},
    month={Jan}}
```
3.67 Random_SMOTE

3.67.1 API

3.67.2 Example

```python
>>> oversampler = smote_variants.Random_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

References:
- BibTex:

```bibtex
@InProceedings{random_smote,
  author = "Dong, Yanjie",
  title = "Random SMOTE",
  year = "2020",
  organization = "IEEE",
  keywords = "oversampling",
  doi = "10.1109/CBMS.2020.00012",
}
```
3.68 ISMOTE

3.68.1 API

3.68.2 Example

```python
>>> oversampler = smote_variants.ISMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
Improvement on SMOTE (SMOTE) is used to do over-sampling on minority class, while distance-based under-sampling (DUS) method is used to do under-sampling on majority class. We adjust the sampling times to search for the optimal results while maintain the dataset size unchanged. Experiments on UCI datasets show that the proposed method performs better than using single over-sampling or under-sampling method.

BibTex:

@InProceedings{ismote,
  author="Li, Hu and Zou, Peng and Wang, Xiang and Xia, Rongze",
  editor="Sun, Zengqi and Deng, Zhidong",
  title="A New Combination Sampling Method for Imbalanced Data",
  booktitle="Proceedings of 2013 Chinese Intelligent Automation Conference",
  year="2013",
  publisher="Springer Berlin Heidelberg",
  address="Berlin, Heidelberg",
  pages="547--554",
  abstract="Imbalanced data is commonly in the real world and brings a lot of challenges. In this paper, we propose a combination sampling method which resamples both minority class and majority class. Improved SMOTE (ISMOTE) is used to do over-sampling on minority class, while distance-based under-sampling (DUS) method is used to do under-sampling on majority class. We adjust the sampling times to search for the optimal results while maintain the dataset size unchanged. Experiments on UCI datasets show that the proposed method performs better than using single over-sampling or under-sampling method.",
  isbn="978-3-642-38466-0"
}
## 3.69 VIS_RST

### 3.69.1 API

### 3.69.2 Example

```python
>>> oversampler = smote_variants.VIS_RST()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

References:

- BibTex:

```tex
@InProceedings{vis_rst,
    author="Borowska, Katarzyna",
}
```
and Stepaniuk, Jarosław, editor=Saeed, Khalid and Homenda, Władysław, title="Imbalanced Data Classification: A Novel Re-sampling Approach Combining Versatile Improved SMOTE and Rough Sets", booktitle="Computer Information Systems and Industrial Management", year="2016", publisher="Springer International Publishing", address="Cham", pages="31--42", abstract="In recent years, the problem of learning from imbalanced data has emerged as important and challenging. The fact that one of the classes is underrepresented in the data set is not the only reason of difficulties. The complex distribution of data, especially small disjuncts, noise and class overlapping, contributes to the significant depletion of classifier’s performance. Hence, the numerous solutions were proposed. They are categorized into three groups: data-level techniques, algorithm-level methods and cost-sensitive approaches. This paper presents a novel data-level method combining Versatile Improved SMOTE and rough sets. The algorithm was applied to the two-class problems, data sets were characterized by the nominal attributes. We evaluated the proposed technique in comparison with other preprocessing methods. The impact of the additional cleaning phase was specifically verified.", isbn="978-3-319-45378-1" }

Notes:

• Replication of DANGER samples will be removed by the last step of noise filtering.

### 3.70 GASMOTE

#### 3.70.1 API

#### 3.70.2 Example

```python
>>> oversampler= smote_variants.GASMOTE()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@Article{gasmote,
  author="Jiang, Kun and Lu, Jing and Xia, Kuiliang",
  title="A Novel Algorithm for Imbalance Data Classification Based on Genetic Algorithm Improved SMOTE",
  journal="Arabian Journal for Science and Engineering",
  year="2016",
  month="Aug",
  day="01",
  volume="41",
  number="8",
  pages="3255--3266",
  abstract="The classification of imbalanced data has been recognized as a crucial problem in machine learning and data analysis. In an imbalanced dataset, there are significantly fewer training instances of one class compared to another class. Hence, the minority class instances are much more likely to be misclassified. In the literature, the synthetic minority over-sampling technique (SMOTE) has been developed to deal with the classification of imbalanced datasets. It synthesizes new samples of the minority class to balance the dataset, by re-sampling the instances of the majority class. Nevertheless, the existing algorithms-based SMOTE uses the same sampling rate for all instances of the minority class. This results in sub-optimal performance. To address this issue, we propose a novel genetic algorithm-based SMOTE (GASMOTE) algorithm. The GASMOTE algorithm uses different sampling rates for different minority class instances and finds the combination of optimal sampling rates. The experimental results on ten typical imbalance datasets show that, compared with SMOTE algorithm, GASMOTE can increase 5.9% on F-measure value and 1.6% on G-mean value, and compared with Borderline-SMOTE algorithm, GASMOTE can increase 3.7% on F-measure value and 2.3% on G-mean value. GASMOTE can be used as a new over-sampling technique to deal with imbalance dataset classification problem. We have particularly applied the GASMOTE algorithm to a practical engineering application: prediction of rockburst in the VCR rockburst datasets. The experiment results indicate that the GASMOTE algorithm can accurately predict the rockburst occurrence and hence provides guidance to the design and construction of safe deep mining engineering structures."
}
```
3.71 A_SUWO

3.71.1 API

3.71.2 Example

```python
>>> oversampler = smote_variants.A_SUWO()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

![Original data sample](image)
References:

- BibTeX:

```bibtex
@article{a_suwo,
  title = "Adaptive semi-unsupervised weighted oversampling (A-SUWO) for imbalanced datasets",
  journal = "Expert Systems with Applications",
  volume = "46",
  pages = "405 - 416",
  year = "2016",
  issn = "0957-4174",
  doi = "https://doi.org/10.1016/j.eswa.2015.10.031",
  author = "Iman Nekoeimehr and Susana K. Lai-Yuen",
  keywords = "Imbalanced dataset, Classification, Clustering, Oversampling"
}
```

Notes:

- Equation (7) misses a division by \( R_j \).
- It is not specified how to sample from clusters with 1 instances.

### 3.72 SMOTE_FRST_2T

#### 3.72.1 API

#### 3.72.2 Example

```python
>>> oversampler= smote_variants.SMOTE_FRST_2T()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
References:

- BibTex:

```bibtex
@article{smote_frst_2t,
  title = "Fuzzy-rough imbalanced learning for the diagnosis of High Voltage Circuit Breaker maintenance: The SMOTE-FRST-2T algorithm",
  journal = "Engineering Applications of Artificial Intelligence",
  volume = "48",
  pages = "134 - 139",
  year = "2016",
  issn = "0952-1976",
  doi = "https://doi.org/10.1016/j.engappai.2015.10.009",
  keywords = "High Voltage Circuit Breaker (HVCB), Imbalanced learning, Fuzzy rough set theory, Resampling methods"
}
```

(continues on next page)
Notes:

- Unlucky setting of parameters might result 0 points added, we have fixed this by increasing the gamma_S threshold if the number of samples accepted is low.

- Similarly, unlucky setting of parameters might result all majority samples turned into minority.

- In my opinion, in the algorithm presented in the paper the relations are incorrect. The authors talk about accepting samples having POS score below a threshold, and in the algorithm in both places POS >= gamma is used.

### 3.73 AND_SMOTE

#### 3.73.1 API

#### 3.73.2 Example

```python
>>> oversampler= smote_variants.AND_SMOTE()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
References:

- BibTex:

```bibtex
@inproceedings{and_smote,
    author = {Yun, Jaesub and Ha, Jihyun and Lee, Jong-Seok},
    title = {Automatic Determination of Neighborhood Size in SMOTE},
    booktitle = {Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication},
    series = {IMCOM '16},
    year = {2016},
    isbn = {978-1-4503-4142-4},
    location = {Danang, Viet Nam},
    pages = {100:1--100:8},
    articleno = {100},
    numpages = {8},
    url = {http://doi.acm.org/10.1145/2857546.2857648},
    doi = {10.1145/2857546.2857648},
    acmid = {2857648},
    publisher = {ACM},
    address = {New York, NY, USA},
    keywords = {SMOTE, imbalanced learning, synthetic data},
}
```

3.74 NRAS

3.74.1 API

3.74.2 Example

```python
>>> oversampler = smote_variants.NRAS()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```biblatex
@article{nras,
  title = "Noise Reduction A Priori Synthetic Over-Sampling for class imbalanced data sets",
  journal = "Information Sciences",
  volume = "408",
  pages = "146 - 161",
  year = "2017",
  issn = "0020-0255",
  doi = "https://doi.org/10.1016/j.ins.2017.04.046",
  author = "William A. Rivera",
  keywords = "NRAS, SMOTE, OUPS, Class imbalance, Classification"
}
```
3.75 AMSCO

3.75.1 API

3.75.2 Example

```python
>>> oversampler = smote_variants.AMSCO()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

References:

- BibTex:

```latex
@article{amsco,
  title = "Adaptive multi-objective swarm fusion for imbalanced data classification",
  \end{document}
```
Notes:

- It is not clear how the kappa threshold is used, I do use the RA score to drive all the evolution. Particularly:
  
  “In the last phase of each iteration, the average Kappa value in current non-inferior set is compare
  with the latest threshold value, the threshold is then increase further if the average value increases,
  and vice versa. By doing so, the non-inferior region will be progressively reduced as the Kappa
  threshold lifts up.”

  I don’t see why would the Kappa threshold lift up if the kappa thresholds are decreased if the average Kappa
  decreases (“vice versa”).

- Due to the interpretation of kappa threshold and the lack of detailed description of the SIS process, the
  implementation is not exactly what is described in the paper, but something very similar.

3.76 SSO

3.76.1 API

3.76.2 Example

```python
>>> oversampler= smote_variants.SSO()
>>> X_samp, y_samp= oversampler.sample(X, y)
```
References:

- BibTex:

```bibtex
@InProceedings{sso,
    author="Rong, Tongwen and Gong, Huachang and Ng, Wing W. Y.",
    editor="Wang, Xizhao and Pedrycz, Witold and Chan, Patrick and He, Qiang",
    title="Stochastic Sensitivity Oversampling Technique for Imbalanced Data",
    booktitle="Machine Learning and Cybernetics",
    year="2014",
    publisher="Springer Berlin Heidelberg",
    address="Berlin, Heidelberg",
}
```

(continues on next page)
Data level technique is proved to be effective in imbalance learning. The SMOTE is a famous oversampling technique generating synthetic minority samples by linear interpolation between adjacent minorities. However, it becomes inefficient for datasets with sparse distributions. In this paper, we propose the Stochastic Sensitivity Oversampling (SSO) which generates synthetic samples following Gaussian distributions in the Q-union of minority samples. The Q-union is the union of Q-neighborhoods (hypercubes centered at minority samples) and such that new samples are synthesized around minority samples. Experimental results show that the proposed algorithm performs well on most of datasets, especially those with a sparse distribution.

Notes:
- In the algorithm step 2d adds a constant to a vector. I have changed it to a componentwise adjustment, and also used the normalized STSM as I don’t see any reason why it would be some reasonable, bounded value.

### 3.77 NDO_sampling

#### 3.77.1 API

#### 3.77.2 Example

```python
>>> oversampler = smote_variants.NDO_sampling()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```{bibtex}
@INPROCEEDINGS{ndo_sampling,
  author={Zhang, L. and Wang, W.},
  booktitle={2011 International Conference of Information Technology, Computer Engineering and Management Sciences},
  title={A Re-sampling Method for Class Imbalance Learning with Credit Data},
  year={2011},
  volume={1},
  number={},
  pages={393-397},
  keywords={data handling;sampling methods;resampling method; class imbalance learning;credit rating;imbalance problem;synthetic minority oversampling technique;sample distribution;synthetic samples;credit data set;Training;Measurement;Support vector machines;Logistics;Testing;Noise; Classification algorithms;class imbalance;credit rating;SMOTE;sample distribution},
  doi={10.1109/ICM.2011.34},
  ISSN={},
  month={Sept}}
```

### 3.78 DSRBF

#### 3.78.1 API

#### 3.78.2 Example

```python
>>> oversampler = smote_variants.DSRBF()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```bibtex
@article{dsrbf,
  title = "A dynamic over-sampling procedure based on sensitivity for multi-class problems",
  journal = "Pattern Recognition",
  volume = "44",
  number = "8",
  pages = "1821 - 1833",
  year = "2011",
  issn = "0031-3203",
  doi = "https://doi.org/10.1016/j.patcog.2011.02.019",
  author = "Francisco Fernández-Navarro and César Hervás-Martínez and Pedro Antonio Gutiérrez",
}
```

(continues on next page)
Notes:

- It is not entirely clear why J-1 output is supposed where J is the number of classes.
- The fitness function is changed to a balanced mean loss, as I found that it just ignores classification on minority samples (class label +1) in the binary case.
- The iRprop+ optimization is not implemented.
- The original paper proposes using SMOTE incrementally. Instead of that, this implementation applies SMOTE to generate all samples needed in the sampling epochs and the evolution of RBF networks is used to select the sampling providing the best results.

3.79 Gaussian_SMOTE

3.79.1 API

3.79.2 Example

```python
>>> oversampler = smote_variants.Gaussian_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
### Gaussian_SMOTE: Ex

![Gaussian SMOTE Example](image)

#### References:

- BibTex:

```bibtex
@article{gaussian_smote,
  title={Gaussian-Based SMOTE Algorithm for Solving Skewed Class Distributions},
  author={Hansoo Lee and Jonggeun Kim and Sungshin Kim},
  journal={Int. J. Fuzzy Logic and Intelligent Systems},
  year={2017},
  volume={17},
  pages={229-234}
}
```

### 3.80 kmeans_SMOTE

#### 3.80.1 API

#### 3.80.2 Example

```python
>>> oversampler = smote_variants.kmeans_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@article{kmeans_smote,
  title = "Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE",
  journal = "Information Sciences",
  volume = "465",
  pages = "1 - 20",
  year = "2018",
  issn = "0020-0255",
  doi = "https://doi.org/10.1016/j.ins.2018.06.056",
  author = "Georgios Douzas and Fernando Bacao and Felix Last",
  keywords = "Class-imbalanced learning, Oversampling, Classification, Clustering, Supervised learning, Within-class imbalance"
}
```

(continues on next page)
3.81 Supervised_SMOTE

3.81.1 API

3.81.2 Example

```python
>>> oversampler = smote_variants.Supervised_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

References:
3.82 SN_SMOTE

3.82.1 API

3.82.2 Example
```python
>>> oversampler = smote_variants.SN_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

---

**Original data sample**

![Original data sample](image)

**SN_SMOTE: Ex, SO**

![SN_SMOTE: Ex, SO](image)

---

**References:**

- BibTex:

```latex
@Article{sn_smote,
  author="Garc{'i}a, V. and S{'a}nchez, J. S. and Mart{'i}n-F{'e}lez, R. and Mollineda, R. A.",
  title="Surrounding neighborhood-based SMOTE for learning from \rightarrow imbalanced data sets",
  journal="Progress in Artificial Intelligence",
  year="2012",
  month="Dec",
}
```

(continues on next page)
Many traditional approaches to pattern classification assume that the problem classes share similar prior probabilities. However, in many real-life applications, this assumption is grossly violated. Often, the ratios of prior probabilities between classes are extremely skewed. This situation is known as the class imbalance problem. One of the strategies to tackle this problem consists of balancing the classes by resampling the original data set. The SMOTE algorithm is probably the most popular technique to increase the size of the minority class by generating synthetic instances. From the idea of the original SMOTE, we here propose the use of three approaches to surrounding neighborhood with the aim of generating artificial minority instances, but taking into account both the proximity and the spatial distribution of the examples. Experiments over a large collection of databases and using three different classifiers demonstrate that the new surrounding neighborhood-based SMOTE procedures significantly outperform other existing over-sampling algorithms.

### 3.83 CCR

#### 3.83.1 API

#### 3.83.2 Example

```python
>>> oversampler = smote_variants.CCR()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@article{ccr,
author = {Koziarski, Michał and Wozniak, Michal},
year = {2017},
month = {12},
pages = {727-736},
title = {CCR: A combined cleaning and resampling algorithm for imbalanced data classification},
volume = {27},
journal = {International Journal of Applied Mathematics and Computer Science},
}
```

Notes:

- Adapted from https://github.com/michalkoziarski/CCR

### 3.84 ANS

#### 3.84.1 API

#### 3.84.2 Example

```python
>>> oversampler = smote_variants.ANS()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
References:

- BibTex:

```latex
@article{ans,
  author = {Siriseriwan, W and Sinapiromsaran, Krung},
  year = {2017},
  month = {09},
  pages = {565-576},
  title = {Adaptive neighbor synthetic minority oversampling technique under 1NN outcast handling},
  volume = {39},
  booktitle = {Songklanakarin Journal of Science and Technology}
}
```

Notes:

- The method is not prepared for the case when there is no \( c \) satisfying the condition in line 25 of the algorithm, fixed.
• The method is not prepared for empty Pused sets, fixed.

3.85 cluster_SMOTE

3.85.1 API

3.85.2 Example

```python
>>> oversampler = smote_variants.cluster_SMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```

References:

- BibTex:
3.86 NoSMOTE

3.86.1 API

3.86.2 Example

```python
>>> oversampler = smote_variants.NoSMOTE()
>>> X_samp, y_samp = oversampler.sample(X, y)
```
The goal of this class is to provide a functionality to send data through on any model selection/evaluation pipeline with no oversampling carried out. It can be used to get baseline estimates on performance.
CHAPTER FOUR

NOISE FILTERS AND PROTOTYPE SELECTION

4.1 TomekLinkRemoval

4.1.1 API

4.1.2 Example

```python
>>> noise_filter = smote_variants.TomekLinkRemoval()
>>> X_samp, y_samp = noise_filter.remove_noise(X, y)
```

![Original data sample](image)
Tomek link removal

References:

- BibTex:

```latex
@article{smoteNoise0,
  author = {Batista, Gustavo E. A. P. A. and Prati, Ronaldo C. and Monard, Maria Carolina},
  title = {A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data},
  journal = {SIGKDD Explor. News1.},
  issue_date = {June 2004},
  volume = {6},
  number = {1},
  month = jun,
  year = {2004},
  issn = {1931-0145},
  pages = {20--29},
  numpages = {10},
  url = {http://doi.acm.org/10.1145/1007730.1007735},
  doi = {10.1145/1007730.1007735},
  acmid = {1007735},
  publisher = {ACM},
  address = {New York, NY, USA}
}
```

4.2 CondensedNearestNeighbors

4.2.1 API

4.2.2 Example

```python
>>> noise_filter= smote_variants.CondensedNearestNeighbors()
>>> X_samp, y_samp= noise_filter.remove_noise(X, y)
```
Condensed nearest neighbors

References:

- BibTex:

```latex
@ARTICLE{condensed_nn,
author={Hart, P.},
journal={IEEE Transactions on Information Theory},
title={The condensed nearest neighbor rule (Corresp.)},
year={1968},
volume={14},
number={3},
pages={515-516},
keywords={Pattern classification},
doi={10.1109/TIT.1968.1054155},
ISSN={0018-9448},
month={May})
```
4.3 OneSidedSelection

4.3.1 API

4.3.2 Example

```python
>>> noise_filter = smote_variants.OneSidedSelection()
>>> X_samp, y_samp = noise_filter.remove_noise(X, y)
```

References:

- BibTex:

```bibtex
@article{smoteNoise0,
  author = {Batista, Gustavo E. A. P. A. and Prati, Ronaldo C. and Monard, Maria Carolina},
  (continues on next page)
```
4.4 CNNTomekLinks

4.4.1 API

4.4.2 Example

```python
>>> noise_filter = smote_variants.CNNTomekLinks()
>>> X_samp, y_samp = noise_filter.remove_noise(X, y)
```
4.5 NeighborhoodCleaningRule

4.5.1 API

4.5.2 Example

```python
>>> noise_filter = smote_variants.NeighborhoodCleaningRule()
>>> X_samp, y_samp = noise_filter.remove_noise(X, y)
```
References:

- BibTex:

```latex
@article{smoteNoise0,
  author = {Batista, Gustavo E. A. P. A. and Prati, Ronaldo C. and Monard, Maria Carolina},
  title = {A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data},
  journal = {SIGKDD Explor. News1.},
  issue_date = {June 2004},
  volume = {6},
  number = {1},
  month = jun,
  year = {2004},
  issn = {1931-0145},
  pages = {20--29},
  numpages = {10},
}
```

(continues on next page)
4.6 EditedNearestNeighbors

4.6.1 API

4.6.2 Example

```python
>>> noise_filter = smote_variants.EditedNearestNeighbors()
>>> X_samp, y_samp = noise_filter.remove_noise(X, y)
```
References:

- BibTex:

```latex
@article{smoteNoise0,
  author = {Batista, Gustavo E. A. P. A. and Prati, Ronaldo C. and Monard, Maria Carolina},
  title = {A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data},
  journal = {SIGKDD Explor. Newsl.},
  issue_date = {June 2004},
  volume = {6},
  number = {1},
  month = jun,
  year = {2004},
  issn = {1931-0145},
  pages = {20--29},
  numpages = {10},
  url = {http://doi.acm.org/10.1145/1007730.1007735},
  doi = {10.1145/1007730.1007735},
  acmid = {1007735},
  publisher = {ACM},
  address = {New York, NY, USA}
}
```
Besides the oversampler implementation, we have prepared some codes for model selection compatible with `sklearn` classifier interfaces.

Having a dataset, a bunch of candidate oversamplers and classifiers, the tools below enable customizable model selection.

### 5.1 Caching

The evaluation and comparison of oversampling techniques on many datasets might take enormous time. In order to increase the reliability of an evaluation process, make it stoppable and restartable and let the oversampling techniques utilize results already computed, we have implemented some model selection and evaluation scripts, both using some hard-disk cache directory to store partial and final results. These functions cannot be used without specifying some cache directory.

### 5.2 Parallelization

The evaluation and model selection scripts are executing oversampling and classification jobs in parallel. If the number of jobs specified is 1, they will call the sklearn algorithms to run in parallel, otherwise the sklearn implementations run in sequential, and the oversampling and classification jobs will be executed in parallel, using `n_jobs` processes.

### 5.3 Querying and filtering oversamplers

### 5.4 Cross validation

### 5.5 Evaluation and validation

### 5.6 Model selection
Multiclass oversampling is highly ambiguous task, as balancing various classes might be optimal with various oversampling techniques. The multiclass oversampling goes on by selecting minority classes one-by-one and oversampling them to the same cardinality as the original majority class, using the union of the original majority class and all already oversampled classes as the majority class in the binary oversampling process. This technique works only with those binary oversampling techniques which do not change the majority class and have a \texttt{proportion} parameter to explicitly specify the number of samples to be generated. Suitable oversampling techniques can be queried by the \texttt{get_all_oversamplers_multiclass} function:
7.1 Simple oversampling

Oversampling can be carried out by importing any oversampler from the `smote_variants` package, instantiating and calling its `sample` function:

```python
import smote_variants as sv
oversampler= sv.SMOTE_ENN()
# supposing that X and y contain some the feature and target data of some dataset
X_samp, y_samp= oversampler.sample(X, y)
```

Using the `datasets` package of `sklearn` to import some data:

```python
import smote_variants as sv
import sklearn.datasets as datasets

dataset= datasets.load_breast_cancer()
oversampler= sv.KernelADASYN()
X_samp, y_samp= oversampler.sample(dataset['data'], dataset['target'])
```

Using the imbalanced datasets available in the `imbalanced_datasets` package:

```python
import smote_variants as sv
import imbalanced_datasets as imbd

data= imbd.load_iris0()
oversamplers= sv.SMOTE_OUT()
X_samp, y_samp= oversampler.sample(dataset['data'], dataset['target'])
```
7.2 Oversampling with random, reasonable parameters

In order to facilitate model selection, each oversampler class is able to generate a set of reasonable parameter combinations. Running an oversampler using a reasonable parameter combination:

```python
import numpy as np
import smote_variants as sv
import imbalanced.datasets as imbd

dataset= imbd.load_yeast1()
par_combs= SMOTE_Cosine.parameter_combinations()
oversampler= SMOTE_Cosine(**np.random.choice(par_combs))
X_samp, y_samp= oversampler.sample(dataset['data'], dataset['target'])
```

7.3 Multiclass oversampling

Multiclass oversampling is highly ambiguous task, as balancing various classes might be optimal with various oversampling techniques. Currently, we have support for multiclass oversampling with one specific oversampler, and only those oversamplers can be used which do not change the majority class and have a proportion parameter to explicitly specify the number of samples to be generated. Suitable oversampling techniques can be queried by the `get_all_oversamplers_multiclass` function. In the below example the `wine` dataset is balanced by multiclass oversampling:

```python
import smote_variants as sv
import sklearn.datasets as datasets

dataset= datasets.load_wine()
oversampler= sv.MulticlassOversampling(sv.distance_SMOTE())
X_samp, y_samp= oversampler.sample(dataset['data'], dataset['target'])
```

7.4 Model selection

When facing an imbalanced dataset, model selection is crucial to find the right oversampling approach and the right classifier. It is obvious that the best performing oversampling technique depends on the subsequent classification, thus, the model selection of oversampler and classifier needs to be carried out hand in hand. This is facilitated by the `model_selection` function of the package. One must specify a set of oversamplers and a set of classifiers, a score function (in this case ‘AUC’) to optimize in cross validation and the `model_selection` function does all the job:

```python
import smote_variants as sv
import imbalanced.datasets as imbd

from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

datasets= [imbd.load_glass2]
```
oversamplers= [sv.SMOTE_ENN, sv.NEATER, sv.Lee]
classifiers= [KNeighborsClassifier(n_neighbors= 3),
             KNeighborsClassifier(n_neighbors= 5),
             DecisionTreeClassifier()]
cache_path= '/home/<user>/smote_validation/'
sampler, classifier= model_selection(datasets,
                                   oversamplers,
                                   classifiers,
                                   cache_path,
                                   'auc',
                                   n_jobs= 10,
                                   max_n_sampler_parameters= 15)

Note, that we have also supplied a cache path, it is used to store partial results, samplings and cross validation scores. The n_jobs parameter specifies the number of oversampling and classification jobs to be executed in parallel, and `max_n_sampler_parameters` specifies the maximum number of reasonable parameter combinations tested for each oversampler. The function call returns the best performing oversampling object and the corresponding, best performing classifier object, respecting the 'glass2' dataset.

7.5 Thorough evaluation involving multiple datasets

Another scenario is the comparison and evaluation of a new oversampler to conventional ones involving a set of imbalance datasets. This scenario is facilitated by the `evaluate_oversamplers` function, which is parameterized similarly to `model_selection`, but returns all the raw results of the numerous cross-validation scenarios (all datasets times (all oversamplers with `max_n_sampler_parameters` parameter combinations) times (all supplied classifiers)):

```python
import smote_variants as sv
import imbalanced_datasets as imbd
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
datasets= [imbd.load_glass2, imbd.load_ecoli4]
oversamplers= [sv.SMOTE_ENN, sv.NEATER, sv.Lee]
classifiers= [KNeighborsClassifier(n_neighbors= 3),
             KNeighborsClassifier(n_neighbors= 5),
             DecisionTreeClassifier()]
cache_path= '/home/<user>/smote_validation/'
results= evaluate_oversamplers(datasets,
                               oversamplers,
                               classifiers,
                               cache_path,
                               n_jobs= 10,
                               max_n_sampler_parameters= 10)
```

Again, the function uses 10 parallel jobs to execute oversampling and classification. In the example above, 2 datasets, 3 classifiers and maximum 10 oversampler parameter combinations are specified for 3 oversampling objects, which requires 2x3x10x3 180 cross-validations altogether. In the resulting pandas DataFrame, for each classifier type (KNeighborsClassifier and DecisionTreeClassifier), and for each oversampler the highest performance measures and the corre-
sponding classifier and oversampler parameters are returned. The structure of the DataFrame is self-explaining.

### 7.6 Reproducing the results in the comparative study

Although a 5-fold 3 times repeated stratified k-fold cross validation was executed, one might expect that the results still depend slightly on the foldings being used. In order to fully reproduce the results of the comparative study, download the foldings we use, and execute the following script by setting the cache_path to the path containing the downloaded foldings. The folding generator will pick-up and use the foldings supplied:

```python
import os, pickle, itertools

# import classifiers
from sklearn.calibration import CalibratedClassifierCV
from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from smote_variants import MLPClassifierWrapper

# import SMOTE variants
import smote_variants as sv

# itertools to derive imbalanced databases
import imbalanced_databases as imbd

# global variables
folding_path= '/home/<user>/smote_foldings/'
max_sampler_parameter_combinations= 35
n_jobs= 5

# instantiate classifiers
sv_classifiers= [CalibratedClassifierCV(LinearSVC(C=1.0, penalty='l1', loss= 'squared_hinge', dual=False)),
                 CalibratedClassifierCV(LinearSVC(C=1.0, penalty='l2', loss= 'hinge', dual=True)),
                 CalibratedClassifierCV(LinearSVC(C=1.0, penalty='l2', loss= 'squared_hinge', dual=False)),
                 CalibratedClassifierCV(LinearSVC(C=10.0, penalty='l1', loss= 'squared_hinge', dual=False)),
                 CalibratedClassifierCV(LinearSVC(C=10.0, penalty='l2', loss= 'hinge', dual=True)),
                 CalibratedClassifierCV(LinearSVC(C=10.0, penalty='l2', loss= 'squared_hinge', dual=False))]

mlp_classifiers= []
for x in itertools.product(['relu', 'logistic'], [1.0, 0.5, 0.1]):
    mlp_classifiers.append(MLPClassifierWrapper(activation= x[0], hidden_layer_fraction= x[1]))

nn_classifiers= []
for x in itertools.product([3, 5, 7], ['uniform', 'distance'], [1, 2, 3]):
    nn_classifiers.append(KNeighborsClassifier(n_neighbors= x[0], weights= x[1], p=x[2]))

dt_classifiers= []
for x in itertools.product(['gini', 'entropy'], [None, 3, 5]):
    dt_classifiers.append(DecisionTreeClassifier(criterion= x[0], max_depth= x[1]))
```

(continues on next page)
classifiers = []
classifiers.extend(sv_classifiers)
classifiers.extend(mlp_classifiers)
classifiers.extend(nn_classifiers)
classifiers.extend(dt_classifiers)

datasets = imbd.get_data_loaders('study')

# instantiate the validation object
results = sv.evaluate_oversamplers(datasets,
    samplers= sv.get_all_oversamplers(),
    classifiers= classifiers,
    cache_path= folding_path,
    n_jobs= n_jobs,
    remove_sampling_cache= True,
    max_n_sampler_parameters= max_sampler_parameter_˓
    combinations)
Implementing a new oversampling logic which can be used in the model selection framework is easy:

- It should inherit from the `smote_variants.OverSampling` class
- Implement the class-level method `parameter_combinations`, which returns a list of reasonable parameter combinations compatible with the constructor. A parameter combination in the list needs to be a dictionary which can be passed to the constructor of the object using the asterisk-operator.
- It needs to implement the `sample` function, which takes a feature array and a target array.
- Finally, it needs to implement the `get_params` function to return the parameters of an oversampling instance as a dictionary.

Below can be found a template for adding new oversamplers:

```python
class New_SMOTE_Variant(smote_variants.OverSampling):
    def __init__(self, param1, param2):
        super().__init__()
        self.param1 = param1
        self.param2 = param2

    @classmethod
    def parameter_combinations(cls):
        return [{'param1': 1, 'param2': 'a'},
                {'param1': 2, 'param2': 'b'},
                {'param1': 3, 'param2': 'c'}]

    def sample(self, X, y):
        # implement sampling logic here
        return X_samp, y_samp

    def get_params(self):
        return {'param1': self.param1, 'param2': self.param2}
```

An oversampler like this should work flawlessly with the model selection and evaluation scripts provided.
In this page, we demonstrate the output of various oversampling and noise removal techniques, using default parameters.

For binary oversampling and noise removal, an artificial database was used, available in the `utils` directory of the github repository.

For multiclass oversampling we have used the ‘wine’ dataset from `sklearn.datasets`, which has 3 classes and many features, out which the first two coordinates have been used for visualization.

### 9.1 Oversampling sample results

In the captions of the images some abbreviations referring to the operating principles are placed. Namely:

- NR: noise removal is involved
- DR: dimension reduction is applied
- Clas: some supervised classifier is used
- SCmp: sampling is carried out componentwise (attributewise)
- SCpy: sampling is carried out by copying instances
- SO: ordinary sampling (just like in SMOTE)
- M: memetic optimization is used
- DE: density estimation is used
- DB: density based - the sampling is based on a density of importance assigned to the instances
- Ex: the sampling is extensive - samples are added successively, not optimizing the holistic distribution of a given number of samples
- CM: changes majority - even majority samples can change
- Clus: uses some clustering technique
- BL: identifies and samples the neighborhoods of borderline samples
- A: developed for a specific application
9.1. Oversampling sample results
9.1. Oversampling sample results
9.1. Oversampling sample results
9.1. Oversampling sample results
9.1. Oversampling sample results

SUNDO: CM, A

MSYN: Ex

9.1. Oversampling sample results
SVM_balance: Ex, Clas, CM

TRIM_SMOTE: Ex, Clus
9.1. Oversampling sample results
9.1. Oversampling sample results
9.1. Oversampling sample results
9.1. Oversampling sample results
RWO_sampling: Ex

NEATER: Ex, BL, CM
9.1. Oversampling sample results
9.1. Oversampling sample results

**SMOTE_IPF: CM, Clas**

**KernelADASYN: DE, Ex, BL**
9.1. Oversampling sample results
9.1. Oversampling sample results

SOMO: Ex, Clus

ISOMAP_Hybrid: Ex, NR, DR, CM
9.1. Oversampling sample results
9.1. Oversampling sample results
9.1. Oversampling sample results

![NT_SMOTE: Ex, A](image1)

![Lee: Ex, SO](image2)
9.1. Oversampling sample results

MDO: Ex, DR

Random_SMOTE: Ex, SCmp
GASMOTE: Ex, M, SO

A_SUWO: Ex, Clus, DB, NR

9.1. Oversampling sample results
9.1. Oversampling sample results

![NRAS: SO, NR](image)

- Majority (black +)
- Minority (red •)

![AMSCO: CM, M, Clas](image)

- Majority (black +)
- Minority (red •)
Chapter 9. Gallery
9.1. Oversampling sample results
9.1. Oversampling sample results

SN_SMOTE: Ex, SO

CCR: Ex
9.2 Noise removal sample results
9.2. Noise removal sample results
9.3 Multiclass sample results
9.3. Multiclass sample results

wine dataset, LLE_SMOTE

wine dataset, distance_SMOTE
9.3. Multiclass sample results
Chapter 9. Gallery

wine dataset, MSMOTE

wine dataset, SMOBD
9.3. Multiclass sample results
9.3. Multiclass sample results
wine dataset, SOI_CJ

wine dataset, ROSE
9.3. Multiclass sample results
9.3. Multiclass sample results
9.3. Multiclass sample results

wine dataset, MCT

wine dataset, ADG
9.3. Multiclass sample results
9.3. Multiclass sample results
### 9.3. Multiclass sample results

![wine dataset, DBSMOTE](image)

![wine dataset, ASMOBD](image)
9.3. Multiclass sample results
9.3. Multiclass sample results
wine dataset, AND_SMOTE

wine dataset, NRAS
9.3. Multiclass sample results
**wine dataset, DSRBF**

**wine dataset, Gaussian_SMOTE**
9.3. Multiclass sample results
9.3. Multiclass sample results
All the implemented oversampling techniques can be called from R using the *reticulate* package. It needs a distinct, working Python installation, which then takes care about the conversion of data back and forth. Supposing that an Anaconda installation is available in the home directory of the user, with *smote_variants* and *imbalanced_databases* (to load imbalanced datasets easily) installed, the following R code works flawlessly.

```r
library(reticulate)

python_path <- file.path(file.expand('~'), 'anaconda3', 'bin', 'python')
virtualenv_name <- 'base'

use_python(python_path)
use_virtualenv(virtualenv_name)

imbd <- import("imbalanced_databases")
sv <- import("smote_variants")

data <- imbalanced_databases$load_iris0()
sv$SMOTE()$sample(data$data, data$target)
```
Similarly to R using reticulate, Python packages can be called from Julia using the package PyCall given that some python installation with smote_variants is available.

Suppose, there is an Anaconda3 install available at `/home/<user>/anaconda3` and smote_variants and imbalanced_databases are installed on the base conda environment.

The following steps are needed to run oversampler codes from the smote_variants package:

- Start the Julia interpreter:

  ```
  julia
  ```

- In the Julia prompt, set the Python path to that of the Anaconda install:

  ```
  ENV["PYTHON"]= "/home/<user>/anaconda3/bin/python3"
  ```

- Add the PyCall package:

  ```
  import Pkg
  Pkg.add("PyCall")
  Pkg.build("PyCall")
  ```

- Restart the Julia interpreter.
- The following code should work:

  ```
  using PyCall
  @pyimport imbalanced_databases as imbd
  @pyimport smote_variants as sv
  dataset= imbd.load_iris0()
  oversampler= sv.SMOTE_ENN()
  X_samp, y_samp= oversampler[:sample](dataset["data"], dataset["target"])
  ```
We are running a volunteery competition, there is no prize except topping the ranking below.

Implement your oversampling technique in terms of the \texttt{smote\_variants} package (check the page \textit{Adding a new oversampler}), evaluate locally against the best performers, and if the results are convincing, send us or merge your code into the package. We’ll repeat the testing and rank your oversampler according to the results.
Based on a thorough evaluation using 104 imbalanced datasets, the following 10 techniques provide the highest performance in terms of the AUC, GAcc, F1 and P20 scores, in nearest neighbors, support vector machine, decision tree and multilayer perceptron based classification scenarios. For more details on the evaluation methodology, see our paper on the comparative study.

<table>
<thead>
<tr>
<th>sampler</th>
<th>overall</th>
<th>auc</th>
<th>rank_auc</th>
<th>gacc</th>
<th>rank_gacc</th>
<th>f1</th>
<th>rank_f1</th>
<th>p_top20</th>
<th>rank_p_top20</th>
</tr>
</thead>
<tbody>
<tr>
<td>polynom-fit-SMOTE</td>
<td>2.5</td>
<td>0.902538</td>
<td>6</td>
<td>0.870753</td>
<td>1</td>
<td>0.695154</td>
<td>1</td>
<td>0.992496</td>
<td>2</td>
</tr>
<tr>
<td>ProWSyn</td>
<td>4.5</td>
<td>0.904389</td>
<td>1</td>
<td>0.868449</td>
<td>4</td>
<td>0.690284</td>
<td>3</td>
<td>0.991112</td>
<td>10</td>
</tr>
<tr>
<td>SMOTE-IPF</td>
<td>7.5</td>
<td>0.902565</td>
<td>5</td>
<td>0.868715</td>
<td>3</td>
<td>0.687935</td>
<td>9</td>
<td>0.990906</td>
<td>13</td>
</tr>
<tr>
<td>Lee</td>
<td>8</td>
<td>0.902318</td>
<td>7</td>
<td>0.868324</td>
<td>5</td>
<td>0.688086</td>
<td>8</td>
<td>0.991008</td>
<td>12</td>
</tr>
<tr>
<td>SMOBD</td>
<td>9.25</td>
<td>0.902247</td>
<td>8</td>
<td>0.86766</td>
<td>6</td>
<td>0.688885</td>
<td>4</td>
<td>0.990583</td>
<td>19</td>
</tr>
<tr>
<td>G-SMOTE</td>
<td>13.5</td>
<td>0.901916</td>
<td>10</td>
<td>0.865103</td>
<td>18</td>
<td>0.686613</td>
<td>12</td>
<td>0.990846</td>
<td>14</td>
</tr>
<tr>
<td>CCR</td>
<td>14.25</td>
<td>0.902112</td>
<td>9</td>
<td>0.861994</td>
<td>30</td>
<td>0.687886</td>
<td>10</td>
<td>0.991254</td>
<td>8</td>
</tr>
<tr>
<td>LVQ-SMOTE</td>
<td>14.75</td>
<td>0.902799</td>
<td>3</td>
<td>0.862295</td>
<td>29</td>
<td>0.683646</td>
<td>24</td>
<td>0.992211</td>
<td>3</td>
</tr>
<tr>
<td>Assembled-SMOTE</td>
<td>15.5</td>
<td>0.902691</td>
<td>4</td>
<td>0.866914</td>
<td>7</td>
<td>0.688614</td>
<td>5</td>
<td>0.982685</td>
<td>46</td>
</tr>
<tr>
<td>SMOTE-TomekLinks</td>
<td>15.75</td>
<td>0.901016</td>
<td>14</td>
<td>0.866174</td>
<td>9</td>
<td>0.684708</td>
<td>20</td>
<td>0.990573</td>
<td>20</td>
</tr>
</tbody>
</table>
• Database foldings: https://drive.google.com/open?id=1PKw1vETVUzaToomio1-RGzJ9-_buYjOW
• Raw results: https://drive.google.com/open?id=12CfB3184nchLiwStaHhrjcKQ7Art18Mo
• Aggregated results: https://drive.google.com/open?id=19JGikRYQ6-eOxaFVrqkF64zOCiSdT-j
CONTRIBUTE

Any contribution is welcome! Feel free to add further oversampling techniques, functionalities, features or fix bugs:

- fork the GitHub repository at http://github.com/analyticalmindsld/smote_variants,
- open a discussion on GitHub,
- or contact me at gyuriofkovacs@gmail.com.
16.1 Version 3.2 (2019.07.28.)

- `OversamplingClassifier` added for compatibility with `sklearn` `Pipeline`.
- New example code added illustrating the use of `OversamplingClassifier`.
- `get_params` fixed by adding the `deep` option.

16.2 Version 3.0 (2019.01.10.)

Stable release containing 85 oversamplers and model selection features.

16.3 Version 0.1 (2018.12.06.)

Initial release.
CHAPTER
SEVENTEEN

INDICES AND TABLES

• genindex
• modindex
• search