imbalanced-learn

imbalanced-learn is a python package offering a number of re-sampling techniques commonly used in datasets showing strong between-class imbalance. It is compatible with scikit-learn and is part of scikit-learn-contrib projects.

imbalanced-learn

Re-sampling techniques are divided in two categories:

- **Under-sampling the majority class(es).**
- **Over-sampling the minority class.**
- Combining over- and under-sampling.
- Create ensemble balanced sets.

Under-sampling

Prototype generation (*prototype generation technique will reduce the number of samples in the targeted classes but the remaining samples are generated — and not selected — from the original set.*)

- ClusterCentroids

Prototype selection (On the contrary to prototype generation algorithms, prototype selection algorithms will select samples from the original set)

Controlled under-sampling techniques (allows for an under-sampling strategy in which the number of samples in S' is specified by the user.)

- RandomUnderSampler
- NearMiss

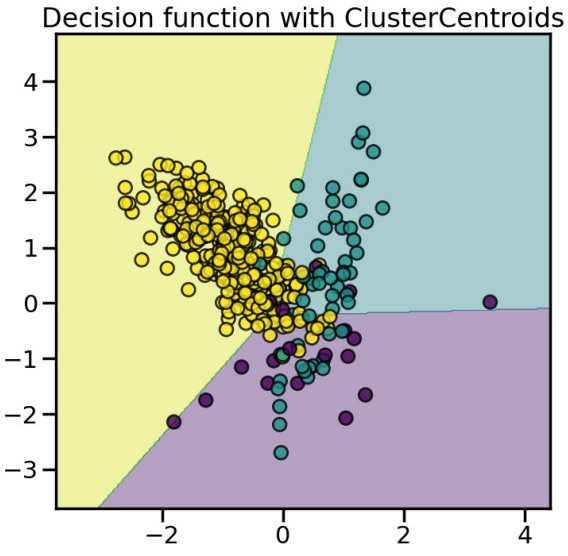
Cleaning under-sampling techniques (Cleaning under-sampling techniques do not allow to specify the number of samples to have in each class.)

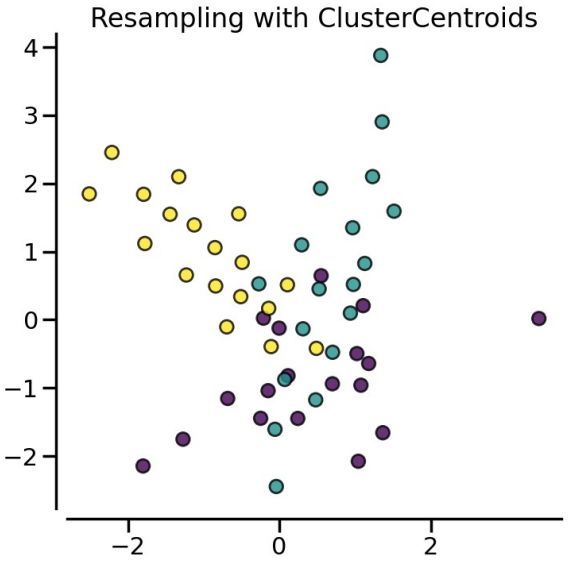
- TomekLinks
- CondensedNearestNeighbour
- OneSidedSelection
- Neighboorhood Cleaning Rule
- EditedNearestNeighbours
- RepeatedEditedNearestNeighbours
- AllKNN
- InstanceHardnessThreshold

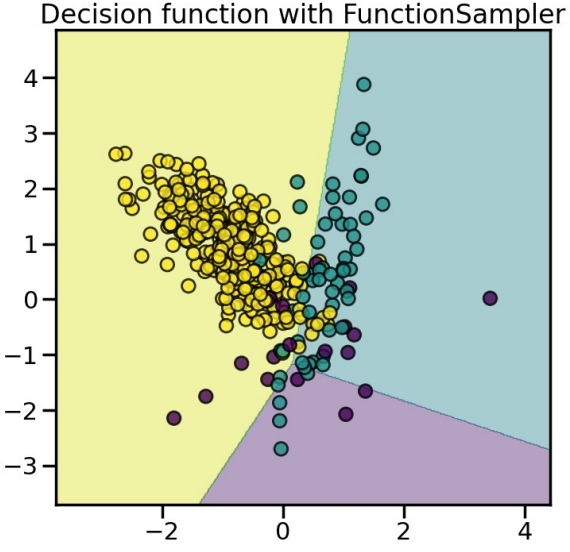
ClusterCentroids

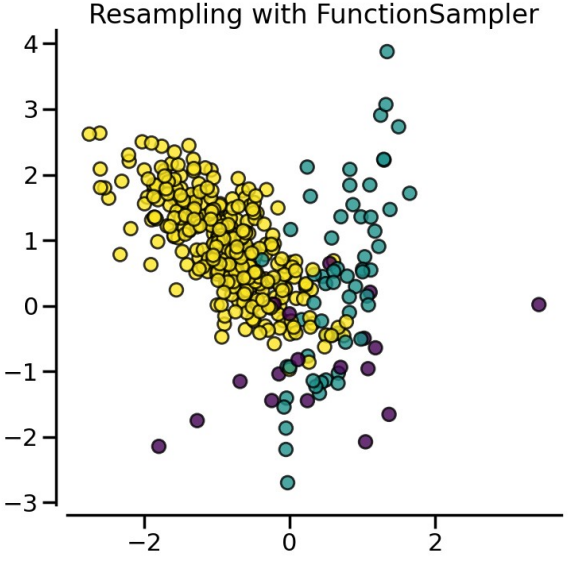
ClusterCentroids makes use of K-means to reduce the number of samples. Therefore, each class will be synthesized with the centroids of the **K-means** method instead of the original samples.

ClusterCentroids offers an efficient way to represent the data cluster with a reduced number of samples. Keep in mind that this method requires that your data are grouped into clusters. In addition, the number of centroids should be set such that the under-sampled clusters are representative of the original one.





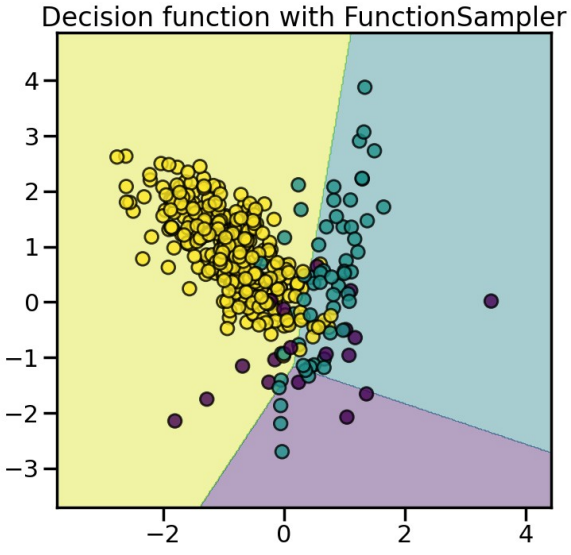


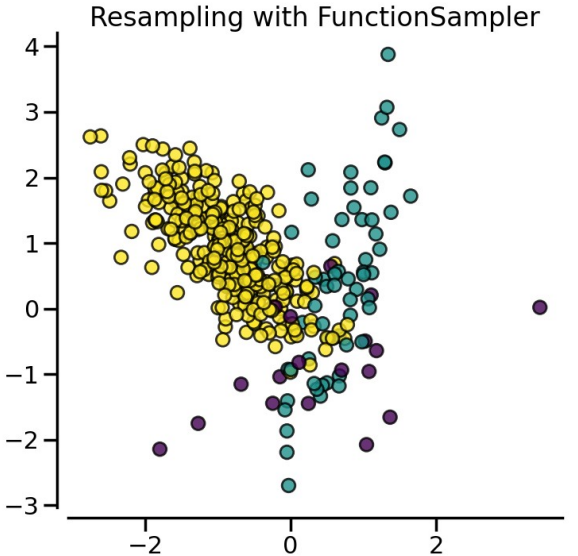


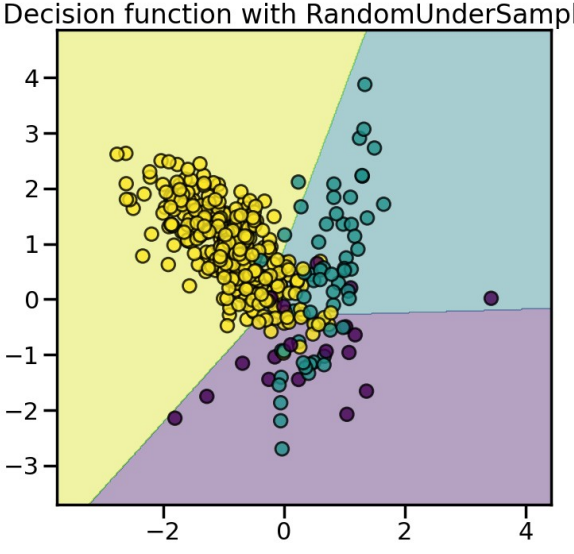
Random UnderSampler

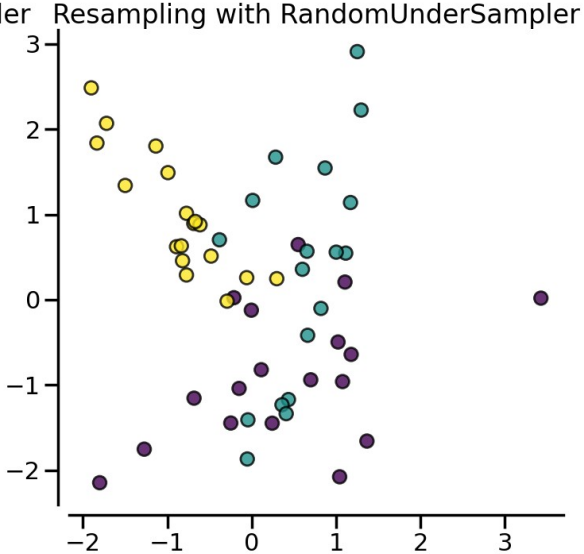
Under-sample the majority class(es) by randomly picking samples with or without replacement.

RandomUnderSampler is a fast and easy way to balance the data by randomly selecting a subset of data for the targeted classes.







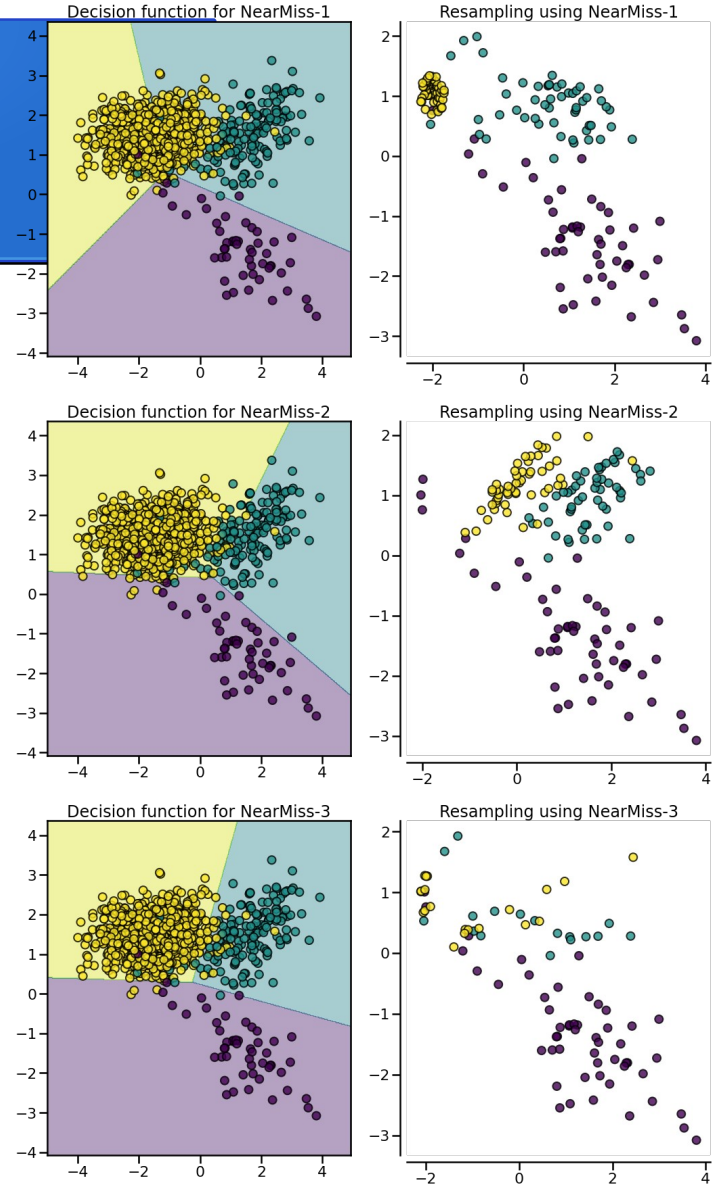


NearMiss

NearMiss-1 selects samples from the majority class for which the average distance of the k ` nearest samples of the minority class is the smallest.

NearMiss-2 selects the samples from the majority class for which the average distance to the farthest samples of the negative class is the smallest.

NearMiss-3 is a 2-step algorithm: first, for each minority sample, their m nearest-neighbors will be kept; then, the majority samples selected are the on for which the average distance to the k nearest neighbors is the largest.



Tomek Links

Tomek’s link exist if the two samples are the nearest neighbors of each other.

Tomeklink.py allows you to remove the Tomeklinks from your data.

Tomek’s algorithm looks for such pairs and removes the majority instance of the pair. The idea is to clarify the border between the minority and majority classes, making the minority region(s) more distinct.

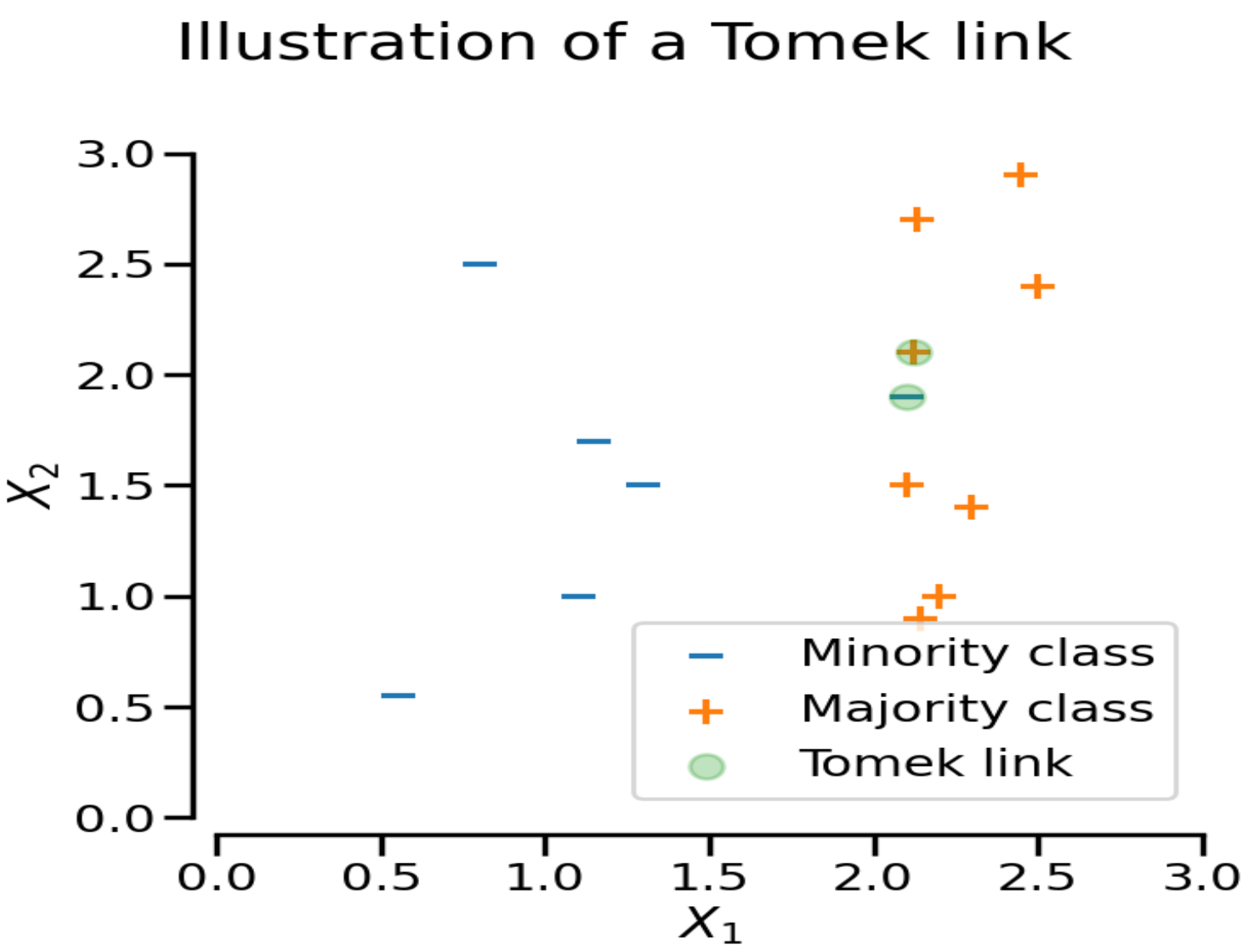
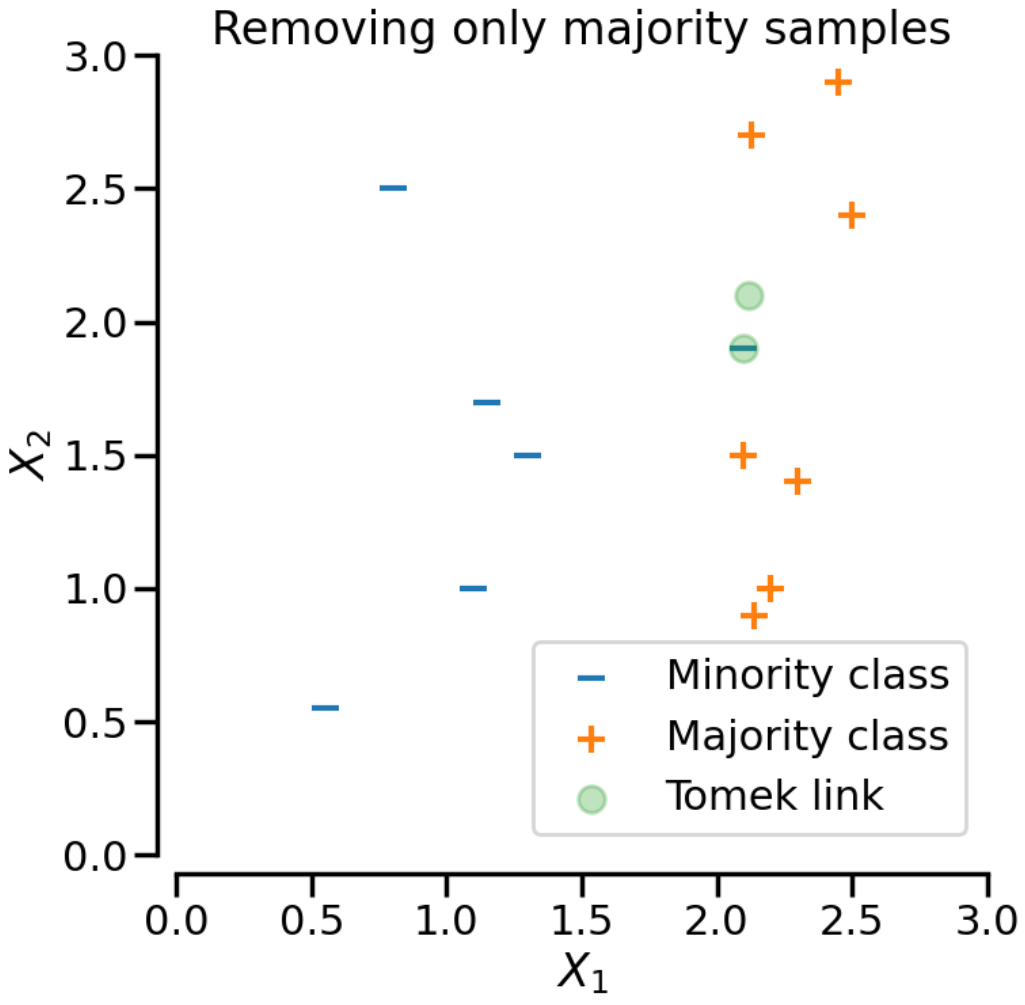
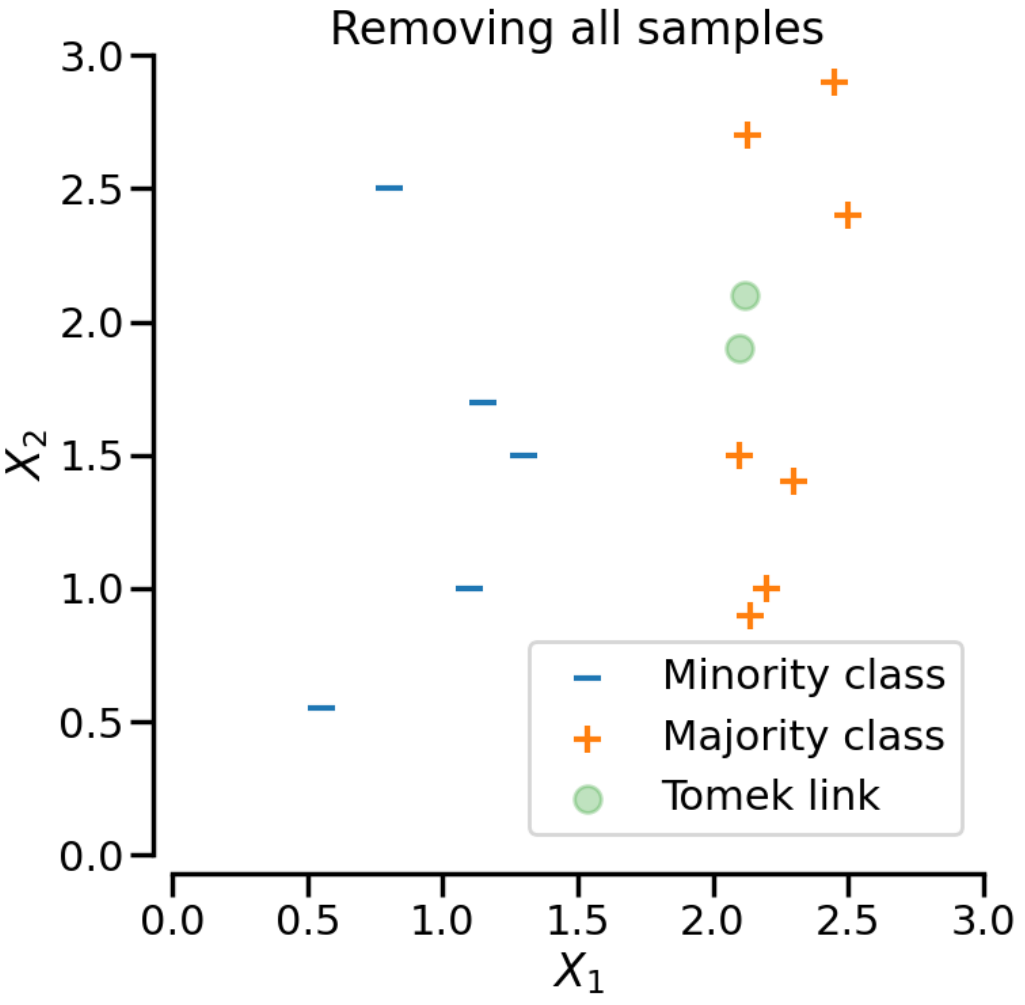


Illustration of a Tomek link
3.0
2.5
2.0
1.5
1.0
0.5
0.0
X_2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
X_1
Minority class
Majority class
Tomek link


Removing only majority samples
3.0
2.5
2.0
1.5
1.0
0.5
0.0
X_2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
X_1
Minority class
Majority class
Tomek link


Removing all samples
3.0
2.5
2.0
1.5
1.0
0.5
0.0
X_2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
X_1
Minority class
Majority class
Tomek link

Edited Nearest Neighbour

EditedNearestNeighbours applies a nearest-neighbors algorithm and “edit” the dataset by removing samples which do not agree “enough” with their neighboorhood.

This rule involves using k=3 nearest neighbors to locate those examples in a dataset that are misclassified and that are then removed before a k=1 classification rule is applied.

for each instance a in the dataset, its three nearest neighbors are computed. If a is a majority class instance and is misclassified by its three nearest neighbors, then a is removed from the dataset. Alternatively, if a is a minority class instance and is misclassified by its three nearest neighbors, then the majority class instances among a’s neighbors are removed.

Repeated Edited Nearest Neighbour

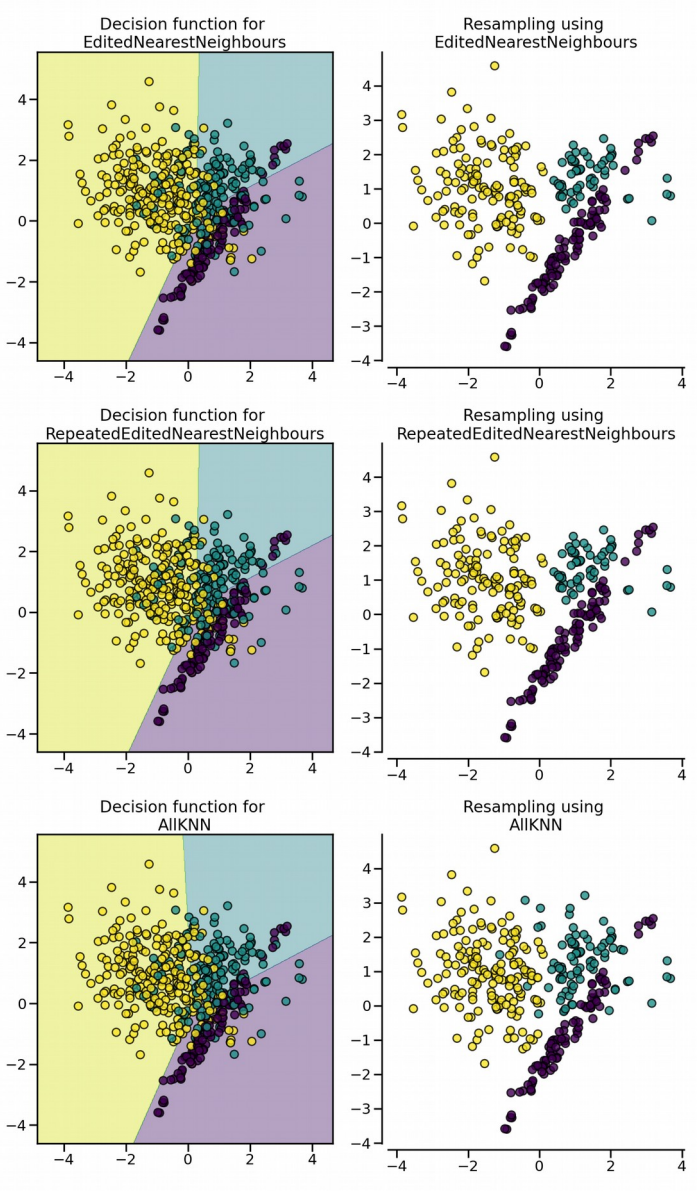
RepeatedEditedNearestNeighbours extends EditedNearestNeighbours by repeating the algorithm multiple times. Generally, repeating the algorithm will delete more data.

AllKNN

AllKNN differs from the previous RepeatedEditedNearestNeighbours since the number of neighbors of the internal nearest neighbors algorithm is increased at each iteration.

Nearest Neighbor Approach

In the example, it can be seen that the three algorithms have similar impact by cleaning noisy samples next to the boundaries of the classes.



CondensedNearestNeighbour

- Get all minority samples in a set C.
- Add a sample from the targeted class (class to be under-sampled) in C and all other samples of this class in a set S.
- Go through the set S, sample by sample, and classify each sample using a 1 nearest neighbor rule.
- If the sample is misclassified, add it to C, otherwise do nothing.
- Reiterate on S until there is no samples to be added.

CondensedNearestNeighbour???

Condensed Nearest Neighbors algorithm helps to reduce the dataset X for k-NN classification. It constructs a subset of examples which are able to correctly classify the original data set using a 1-NN algorithm.

- It is returning not the array of misclassified points, but a subset Z of the data set X.
- CNN works like that:
- 1) Scan all elements of X, looking for an element x whose nearest prototype from Z has a different label than x
- 2) Remove x from X and add it to Z
- 3) Repeat the scan until no more prototypes are added to Z
- Z used instead of X for kNN classification.
- An advantage of this method is decreasing of execution time, reducing a space complexity

CondensedNearestNeighbour

The condensed nearest-neighbor (CNN) method chooses samples randomly. This results in a) retention of unnecessary samples and b) occasional retention of internal rather than boundary samples.

Two modifications of CNN are presented which remove these disadvantages by considering only points close to the boundary. Performance is illustrated by an example.

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4309452

OneSidedSelection

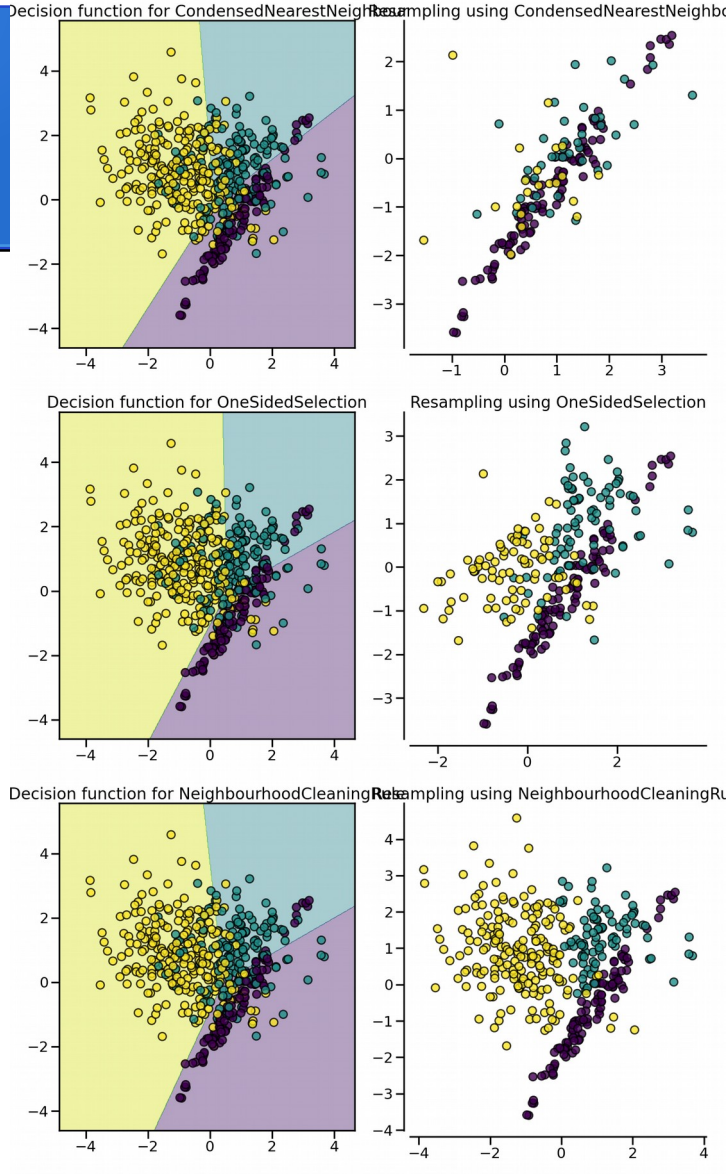
In the contrary, OneSidedSelection will use TomekLinks to remove noisy samples. In addition, the 1 nearest neighbor rule is applied to all samples and the one which are misclassified will be added to the set C. No iteration on the set S will take place.

Neighboorhood Cleaning Rule

NeighbourhoodCleaningRule will focus on cleaning the data than condensing them. Therefore, it will used the union of samples to be rejected between the EditedNearestNeighbours and the output a 3 nearest neighbors classifier.

Condensed Nearest Neighbor Approach

In the example below, it can be seen that the three algorithms have similar impact by cleaning noisy samples next to the boundaries of the classes.

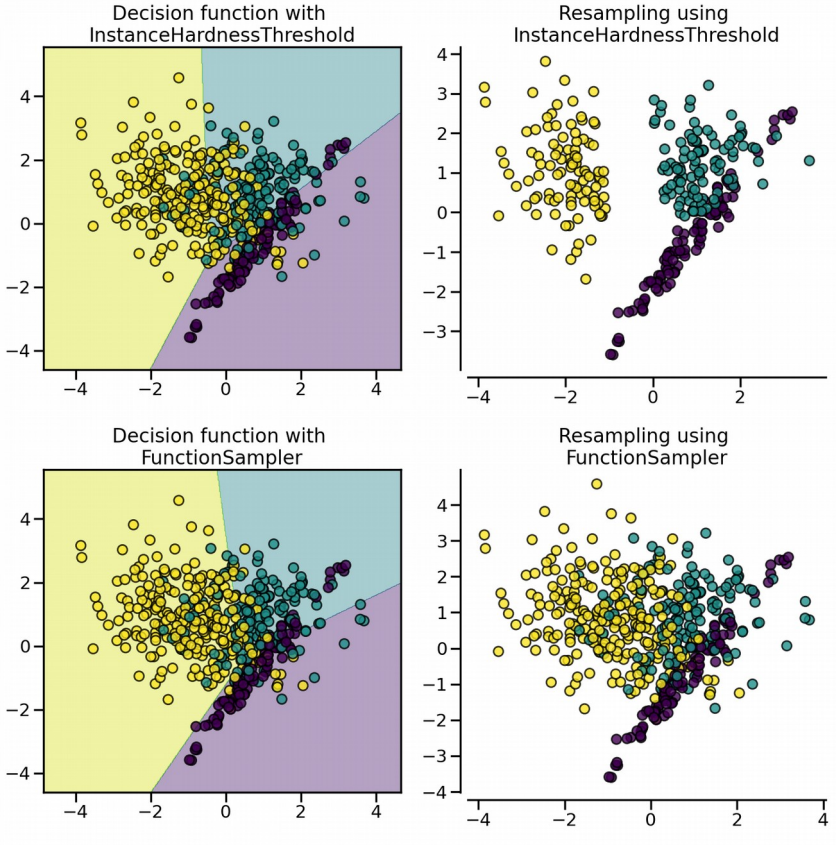


InstanceHardnessThreshold

InstanceHardnessThreshold is a specific algorithm in which a classifier is trained on the data and the samples with lower probabilities are removed

InstanceHardnessThreshold could almost be considered as a controlled under-sampling method. However, due to the probability outputs, it is not always possible to get a specific number of samples.

This class has 2 important parameters. estimator will accept any scikit-learn classifier which has a method predict_proba.



Over-sampling

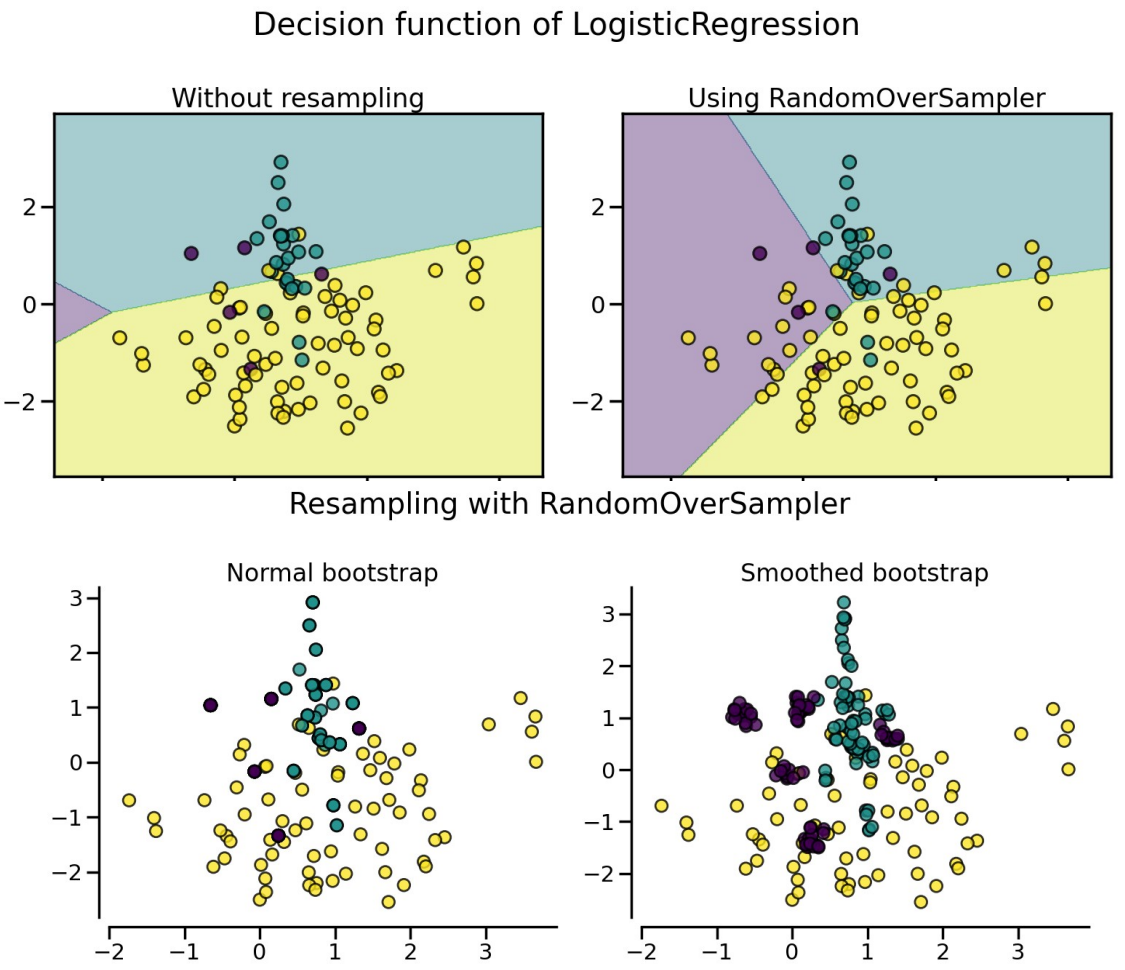
- RandomOverSampler
- SMOTE - Synthetic Minority Over-sampling Technique
- SMOTENC - SMOTE for Nominal and Continuous
- SMOTEN - SMOTE for Nominal
- BorderlineSMOTE
- SVMSMOTE - Support Vectors SMOTE
- KmeansSMOTE
- ADASYN - Adaptive synthetic sampling approach for imbalanced learning

RandomOverSampler

RandomOverSampler is to create new samples by randomly sampling existing samples.

RandomOverSampler is over-sampling by duplicating some of the original samples of the minority class.

We can avoid repeating samples with shrinkage parameter. Shrinkage controls the dispersion of the new generated samples.



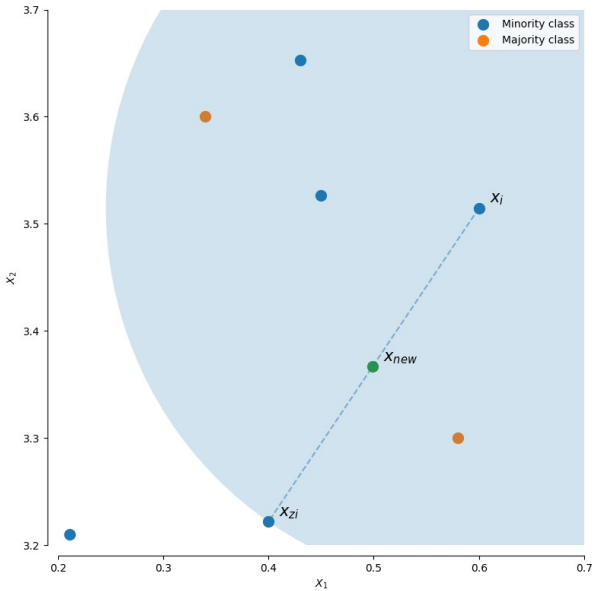
RandomOverSampler

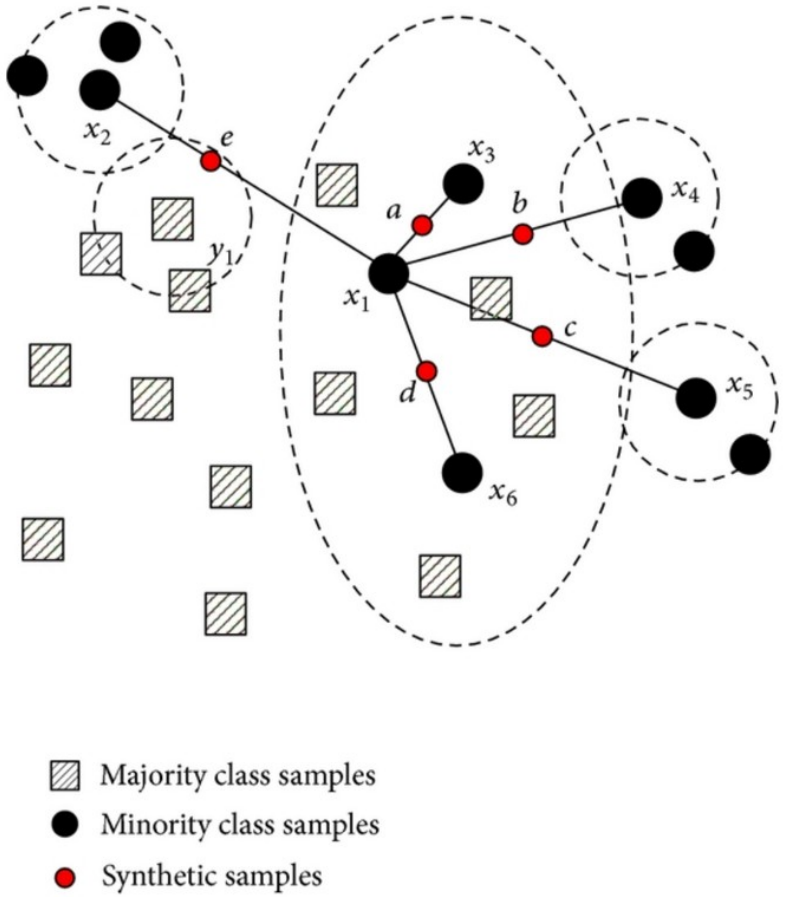
While the RandomOverSampler is over-sampling by duplicating some of the original samples of the minority class, SMOTE and ADASYN generate new samples in by interpolation.

SMOTE

Synthetic Minority Over-sampling Technique

What smote does is simple. First it finds the n-nearest neighbors in the minority class for each of the samples in the class . Then it draws a line between the the neighbors an generates random points on the lines.





ADASYN

Adaptive synthetic sampling approach for imbalanced learning

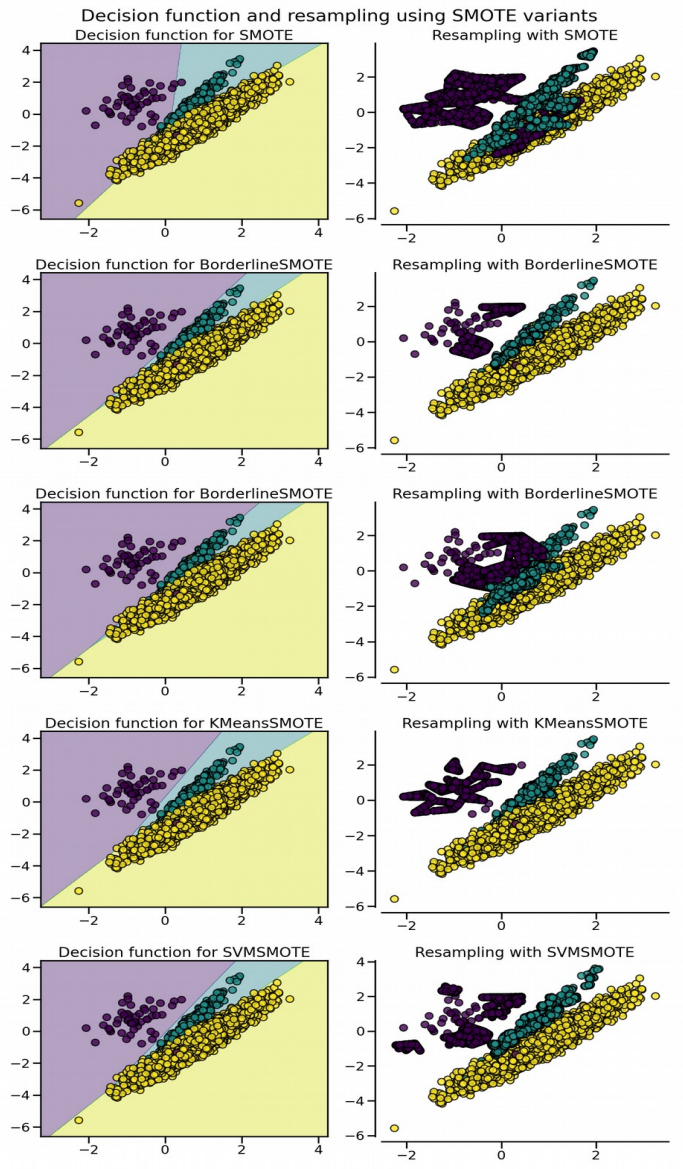
This method is similar to SMOTE but it generates different number of samples depending on an estimate of the local distribution of the class to be oversampled.

Its a improved version of Smote. What it does is same as SMOTE just with a minor improvement. After creating those sample it adds a random small values to the points thus making it more realistic. In other words instead of all the sample being linearly correlated to the parent they have a little more variance in them.

Minority Class	Minority Neighbours	Majority Neighbours	Impurity Ratio
Obs 1	3	2	.6
Obs 2	4	1	.4
Obs 3	1	4	.8
Obs 4	5	0	0

SMOTE Variants

- SMOTENC - SMOTE for Nominal and Continuous
- SMOTEN - SMOTE for Nominal
- BorderlineSMOTE
- SVMSMOTE - Support Vectors SMOTE
- KmeansSMOTE



SMOTENC

SMOTE for Nominal and Continuous

Unlike SMOTE, SMOTE-NC for dataset containing numerical and categorical features. However, it is not designed to work with only categorical features.

SMOTENC is only working when data is a mixed of numerical and categorical features. If data are made of only categorical data.

SMOTEN

SMOTE for Nominal

It expects that the data to resample are only made of categorical features.

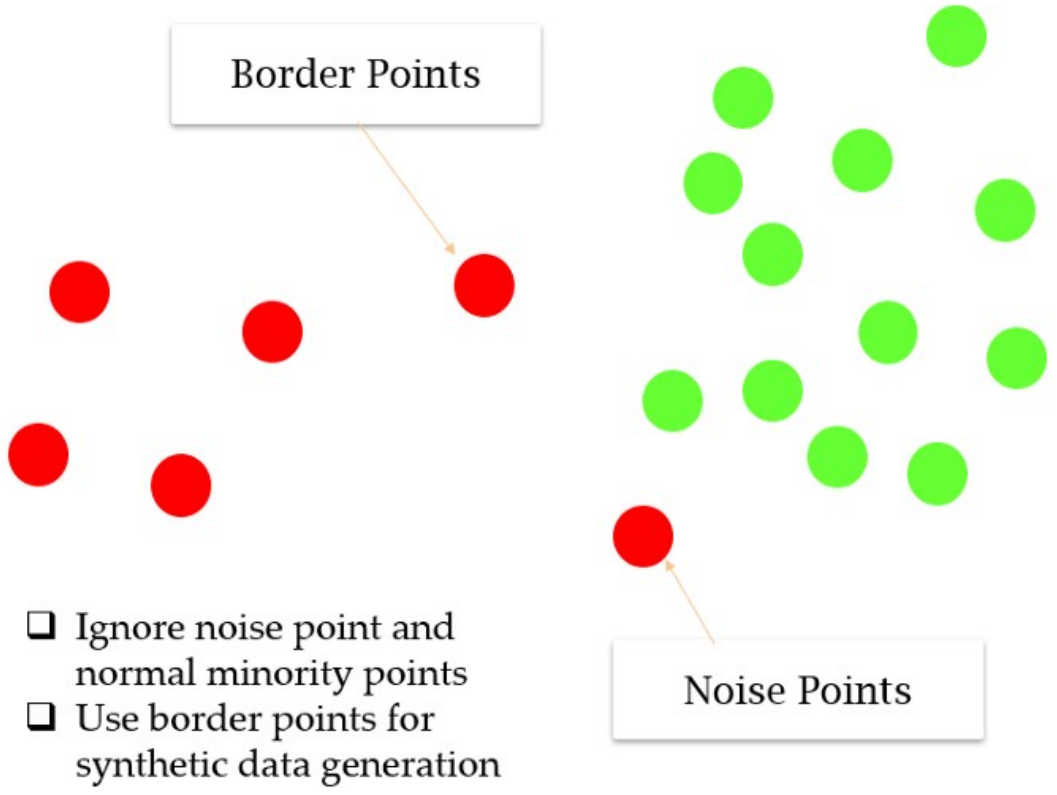
a new sample is generated where each feature value corresponds to the most common category seen in the neighbors samples belonging to the same class.

We had ‘IsActiveMember’ categorical feature with the data either 0 or 1. If we oversampled this data with SMOTE, we could end up with oversampled data such as 0.67 or 0.5, which does not make sense at all.

BorderlineSMOTE

This algorithm starts by classifying the minority class observations. It classifies any minority observation as a noise point if all the neighbors are the majority class and such an observation is ignored while creating synthetic data . Further, it classifies a few points as border points that have both majority and minority class as neighborhood and resample completely from these points.

Borderline-SMOTE only makes synthetic data along the decision boundary between the two classes



SVMSMOTE
Support Vectors SMOTE

SVM-SMOTE focuses on generating new minority class instances near borderlines with SVM so as to help establish boundary between classes.

In the SVM-SMOTE, the borderline area is approximated by the support vectors after training SVMs classifier on the original training set. Synthetic data will be randomly created along the lines joining each minority class support vector with a number of its nearest neighbors.

KMeansSMOTE

Cluster the entire data using the k-means clustering algorithm.

Select clusters that have a high number of minority class samples

Assign more synthetic samples to clusters where minority class samples are sparsely distributed.

Oversample each filtered cluster using SMOTE.

Over-sampling followed by under-sampling

- SMOTETomek - SMOTE + Tomek links
- SMOTEENN - SMOTE + ENN

Ensemble classifier using samplers internally

- EasyEnsembleClassifier
- BalancedRandomForestClassifier
- BalancedBaggingClassifier
- RUSBoostClassifier