applied sciences

MDPI

Article

An Oversampling Method for Class Imbalance Problems on Large Datasets

Fredy Rodríguez-Torres *, José F. Martínez-Trinidad and Jesús A. Carrasco-Ochoa

Instituto Nacional de Astrofísica, Óptica y Electrónica, Tonantzintla, Puebla, Mexico City C.P. 72840, Mexico; fmartine@inaoep.mx (J.F.M.-T.); ariel@inaoep.mx (J.A.C.-O.)
* Correspondence: frodriguez@inaoep.mx

Abstract: Several oversampling methods have been proposed for solving the class imbalance problem. However, most of them require searching the k-nearest neighbors to generate synthetic objects. This requirement makes them time-consuming and therefore unsuitable for large datasets. In this paper, an oversampling method for large class imbalance problems that do not require the k-nearest neighbors' search is proposed. According to our experiments on large datasets with different sizes of imbalance, the proposed method is at least twice as fast as 8 the fastest method reported in the literature while obtaining similar oversampling quality.

Keywords: class imbalance problem; fast oversampling; oversampling methods; large datasets

check for updates

Citation: Rodríguez-Torres, F.; Martínez-Trinidad, J.F.; Carrasco-Ochoa, J.A. An Oversampling Method for Class Imbalance Problems on Large Datasets. *Appl. Sci.* **2022**, *12*, 3424. https://doi.org/10.3390/app12073424

Academic Editors: Antonio Fernández-Caballero, Hugo Pedro Proença and Byung-Gyu Kim

Received: 14 February 2022
Accepted: 21 March 2022
Published: 28 March 2022

1. Introduction

One of the current research topics of interest in supervised classification is the class imbalance problem that frequently appears in several real-world datasets [1–5]. The class imbalance problem occurs when, in a dataset, one of the classes has fewer objects, usually called the minority class, than the other class, usually called the majority class. This problem produces a poor classification rate for the minority class, which is usually the most important. Consequently, it becomes difficult for a classifier to effectively discriminate between the minority and majority classes, especially if the class imbalance is extreme. For example, among several patients with a suspicious disease, very few are likely to have cancer, i.e., minority class, while most are likely not to have cancer, i.e., majority class. Here, a false negative means a patient with cancer is misclassified as not having the disease, which is a severe error. Another example of the class imbalance problem is distinguishing between 100 spam emails from 100,000, given that most emails are non-spam. Like these examples, the class imbalance problem is widespread and appears in many areas such as industrial, medical, and social, among others.

In the literature, we can find different types of solutions like class decomposition [6,7] or oversampling methods that are the most frequently used solutions reported in the literature for solving the class imbalance problem. SMOTE (Synthetic Minority Oversampling Technique) [8] is the most widely used and referenced method among the oversampling methods. SMOTE addresses the class imbalance problem by increasing the number of objects of the minority class by generating synthetic objects for this class.

Due to its success, many other oversampling methods based on SMOTE have been proposed in the literature [9–30]; these methods are characterized by modifications of some of the SMOTE steps and/or addition of new steps. Other oversampling methods that do not follow the SMOTE's approach have also been proposed [22,23,26,31–33]. However, most of the oversampling methods reported in the literature require computing the k-nearest neighbors of objects from the minority class or other slow steps, making them unsuitable for balancing large datasets. This paper will use the term large datasets to refer to those bigger than the conventional datasets used in the literature of oversampling, which have less than 10K objects.

Among those oversampling methods not based on the *k*-nearest neighbors' search, we can highlight LMCMO [34], ROS (Random OverSampling), and NDO [35]. However, LMCMO needs to find the minimum distance between the minority class and the majority class objects, becoming very slow for large datasets. On the other hand, although ROS is fast, it produces low oversampling quality because ROS only generates exact copies of the objects of the minority class [36–39]. Meanwhile, NDO is an oversampling method that is not based on the search for the *k*-nearest neighbors, but in a fast generation of synthetic objects; showing good oversampling results. Therefore, NDO can be applied to large datasets with imbalanced classes. The above shows that the development of fast oversampling methods not based on the *k*-nearest neighbors' search or other slow steps has been little studied in the literature. For his reason, this paper introduces a method that follows this approach, making it applicable to large datasets with imbalanced classes. Our experiments with large datasets show that the proposed method is at least twice as fast as the fastest method reported in the literature while obtaining similar oversampling quality.

The rest of the paper is organized as follows: In Section 2, we present the related work. In Section 3, our oversampling method for class imbalance problems on large datasets is introduced. Section 4 shows our experimental results and compares our proposed method against the most successful oversampling state-of-the-art methods. We also include experiments on real large datasets, as well as scalability experiments that confirm the good performance of the proposed method in terms of time and quality of oversampling. Finally, in Section 5, we present our conclusions and future work.

2. Related Work

The most successful and referenced oversampling method for class imbalance problems is SMOTE [8]. The main idea of SMOTE is to oversample the minority class by randomly generating synthetic objects between objects from the minority class and some of their *k*-nearest neighbors selected in random manner.

In SMOTE, to generate a synthetic object, the feature difference between an object of the minority class and one of its *k*-nearest neighbors, selected in random manner, is computed. Then, for each feature, this difference is multiplied by a random number between 0 and 1. Finally, the multiplied differences is added to the object of the minority class in order to obtain a synthetic object that is then added to the training set. In Algorithm 1, a pseudocode of SMOTE is shown.

Algorithm 1 SMOTE

```
Input: A minority class, $N$ - Percent of oversampling (integer multiple of 100), $k$ - number
  of nearest neighbors.
Output: An oversampled minority class.
  $n \leftarrow N/100$ Number of objects to be generated for each object of the minority class
  for each object $O$ in the minority class do
      Compute the $k$-nearest neighbors of $O$ in the minority class
      for $j = 1, ..., n$ do
          $Onn \leftarrow$ Randomly select one of the $k$-nearest neighbors of $O$
          $Syn \leftarrow$ Initialize an empty object
          for each feature $f$ do
              Compute the feature difference between $O_f$ and $Onn_f$
              $r \leftarrow Random(0,1)$
              $diff \leftarrow$ feature difference $*r$
              $Syn_f \leftarrow O_f + diff$
          end for
          Add $Syn$ to the minority class
      end for
  end for
```

Due to the high success of SMOTE, several oversampling methods based on SMOTE (SMOTE-based methods) have been proposed in the literature [9–29,40–44]. These methods modify SMOTE in different ways in order to improve the oversampling quality. In the next paragraphs, we will describe only some of the most widely cited and recent SMOTE-based methods.

Borderline-SMOTE [9] first labels each object of the minority class as a border. An object is labeled as a border if the number of its k-nearest neighbors computed from the whole training set that belong to the majority class is greater than or equal to $k/2$ and smaller than k. Then, Borderline-SMOTE generates synthetic objects between the border objects and their k-nearest neighbors in the minority class.

Safe Level SMOTE [11] computes, for each object of the minority class, a safe level (sl) value. This value depends on the number of nearest neighbors that the object has into the minority class among its k-nearest neighbors computed in the whole training set. Safe Level SMOTE generates synthetic objects between objects of the minority class with $sl > 0$ and their k-nearest neighbors in the minority class. If the safe level of the selected neighbor is 0, the object of the minority class is duplicated. Otherwise, Safe Level SMOTE computes a safe level ratio (slr) by dividing the sl value of the object in the minority class by the sl value of the selected neighbor. Then, to generate a synthetic object, as SMOTE does, the random number used for multiplying the feature difference between the object of the minority class and its selected neighbor is generated in a different interval, according to the slr value, as follows: if $slr = 1$ the random number is generated between 0 and 1; if $slr > 1$ the random number is generated between 0 and $1/slr$; and if $slr < 1$ the random number is generated between $(1 - slr)$ and 1.

LNE [12] is based on Safe Level SMOTE (SLS) [11], but in the generation of the sinthetic objects rather than computing the k-nearest neighbors in the minority class, LNE computes the k-nearest neighbors in the whole training set. Then, when a neighbor of the majority class is selected to generate a synthetic object, the random number (r) is computed as in SLS and adjusted as $r' = r(sl_n/k)$ where sl_n is the safe level of the selected neighbor.

NDO [35] generates synthetic objects around each object of the minority class by adding a fraction of the standard deviation of the values of each feature. This portion of the standard deviation is computed by dividing the standard deviation by the square root of the number of objects in the minority class, and multiplying the result by a random number. The random number is generated by a normal distribution with the mean of 0 and standard deviation of 1.

SMOTE-RSB* (S-RSB*) [13] first applies SMOTE, and then only keeps as synthetic objects the objects that belong to the lower approximation of the minority class, according to the Rough Set Theory.

K-means-SMOTE (K-means-S) [24] applies K-means for clustering the training set and then selects the clusters with an Imbalance Ratio (IR) lower than a given threshold. Finally, K-means-SMOTE applies SMOTE on those selected clusters. The amount of oversampling for each cluster is computed based on the sparsity and density values of the minority class objects. Following this idea, A-SUWO [25] first group the training set but apply Complete-Linkage Hierarchical Clustering and then apply SMOTE over each cluster.

Farthest SMOTE (Farthest-S) [27] uses a similar approach to that of SMOTE, but rather than using the k-nearest neighbors, it uses the k-farthest neighbors.

To generate a synthetic object, Geometric-SMOTE (G-SMOTE) [28] first randomly selects an object of the minority class as the center of a hyper-spherical region and, also randomly selects one of its nearest neighbors computed in the whole training set to define the size of the hyper-spherical region based on the distance between the two selected objects. Then, the synthetic object is randomly generated as a point inside of the hyper-spherical region; the generation of synthetic objects is repeated as many times as synthetic objects to be generated.

SMOTE-SF (S-SF) [29] selects a subset of features to handle with high-dimensional datasets (a high number of features). The selection of the subset of features is performed by applying a feature ranking method (many methods are studied in their work). Then, SMOTE-SF generates synthetic objects similar to SMOTE but only until it reaches the number specified by an input parameter.

In the literature, other methods that are not based on computing the k-nearest neighbors for generating synthetic objects have been also proposed. However, these methods use computationally expensive techniques such as particle swarm optimization [22], self-organizing maps [23], multi-objective swarm fusion [26], genetic algorithms [31], differential evolution [32], deep generative models and variational autoencoders [33], and feature correlation computing [29]; among others for generating synthetic objects. Therefore, these methods are time-consuming and are unsuitable for large datasets with imbalanced classes.

From the above review of oversampling methods, we can see that with the exception of NDO, most oversampling methods are based on the search for the k-nearest neighbors or other slow techniques. This makes them time-consuming when applied on large datasets with imbalanced classes. Therefore, in the next section, we propose a fast oversampling method for class imbalance problems on large datasets.

3. Proposed Method

SMOTE is the most widely cited method for addressing the class imbalance problems and has inspired and continues to inspire the development of many oversampling methods [45–51]. However, SMOTE as well as SMOTE-based methods have poor time performance for large class imbalance problems because all of these methods either require searching for the k-nearest neighbors or use other time-consuming methods for balancing the minority class.

If SMOTE avoided searching the k-nearest neighbors to generate synthetic objects, this would improve its performance. An alternative to searching for the k-nearest neighbors in the generation of synthetic objects is that instead of using an object of the minority class each time and selecting one of its k-nearest neighbors as is done in SMOTE, one object of the minority class is used and the nearest neighbor's selection is performed with a faster method for determining where in the minority class it is necessary to generate the new synthetic object.

To generate a new synthetic object, SMOTE uses an object of the minority class and one of its nearest neighbors. In some sense, this is done in order to guarantee that the synthetic object is generated in an object's neighborhood belonging to the minority class. Using the nearest neighbors also aims to guarantee that the synthetic object is located in a region in the minority class because it is computed by generating a value between the values of an object in the minority class and one of its nearest neighbors for each of the features of the synthetic object. Thus, in our proposed method for generating synthetic objects in the minority class, instead of using objects of the minority class, as is done in SMOTE, we will use the mode jointly with the minimum and maximum values in each feature. We tested other central tendencies such as the mean or the median and other statistical dispersion metrics such as the standard deviation or the variance in the proposed method. However, the mode jointly with the minimum and maximum values obtained the best oversampling quality results.

To compute the mode for numerical features, we followed [52] (allowing us to compute the mode rapidly) to obtain the minimum and maximum values from a feature and divide the feature's values in the sample in 10 equal-size bins. Then, the mode is computed as described by Equation (1):

$$mode = l + \frac{f - f_{-1}}{(f - f_{-1}) + (f - f_{+1})} * w \tag{1}$$

where l is the lower limit of the bin with the highest frequency (f), w is the bin size, f_{-1} is the frequency of the bin preceding the bin with the highest frequency, and f_{+1} is the frequency of the bin following to the bin with the highest frequency.

It is important to highlight that the mode, the minimum, and the maximum can be computed quickly since they can be computed in two scans of the minority class, i.e., computing them for a feature is $\mathcal{O}(n)$, where n is the number of objects in the minority class. Thus, the complexity of computing them for all features is $\mathcal{O}(mn)$, where m is the number of features.

In our method, we propose computing the absolute differences between the mode value and the minimum and maximum values for each feature in the minority class. Then, by subtracting or adding these differences multiplied by a certain random factor to each feature's mode value, it is possible to generate a new synthetic object into the minority class. Similar to SMOTE, our method generates a synthetic object located between the mode (a central tendency measure) of the class and the neighbor objects that are not farther from the mode than the minimum or maximum values. We want to point out that using the minimum and maximum values allows our method to keep the synthetic generated values in the observed value range of the features.

Following the idea mentioned above, it is possible to avoid searching for the k-nearest neighbors, which is the most time-consuming step of SMOTE. Consequently, we obtain an oversampling method that is faster than SMOTE and the SMOTE-based methods.

The first step of our method computes, for each feature the minimum, maximum and mode values in the minority class. Then, the absolute differences between the mode and the minimum and maximum values are computed. Then, a random number is generated for each feature. This random number is multiplied by one of the differences and subtracted or added to the corresponding feature mode. To decide if a difference is subtracted or added, we generate a random number between 0.0 and 1.0; then, if the generated random number is lower than 0.5, the difference is subtracted. Otherwise, the difference is added. Thus, the generated synthetic object's feature values will be clustered around the mode and in the region defined by each feature's difference values. In the proposed method, we use a random number between 0 and 1; this allows the generation of objects as far as the differences allow, but not too far so as to avoid the generation of feature value outside the observed feature value range.

We want to point out that if the random number that is multiplied by the difference is closer to 0, the synthetic objects are generated closer to the mode of the class on the corresponding feature, or, in the opposite case, if the random number is close to 1, the feature values of the synthetic object is generated farther from the mode of the class and closer to the minimum or maximum values.

Finally, the above-described procedure is repeated as many times as the number of synthetic objects to be generated N (an input parameter of our method). A pseudo-code of the proposed method is shown in Algorithm 2.

To determine the computational complexity of Fast SMOTE, we ill analyze each one of its steps (see Algorithm 2). Steps 1–3 of Fast-SMOTE are $\mathcal{O}(nf)$ (where n is the number of objects in the minority class and f is the number of features of the dataset), because as it is well-known, computing the minimum and maximum and the mode from grouped data of n values is $\mathcal{O}(n)$, and this must be done for each feature. Steps 4 and 5 are $\mathcal{O}(f)$, because they involves just a difference for each one of the f features. The FOR loop of step 6 is repeated n times, and inside of this loop, we have a nested FOR loop that is repeated f times. In the inner loop, the steps 9–15 are $\mathcal{O}(1)$. Therefore the complexity of the FOR loop of step 6 is $\mathcal{O}(nf)$, and hence the complexity of Fast-SMOTE is $\mathcal{O}(nf)$. This complexity is smaller than the complexity of SMOTE and SMOTE-based methods, which is at least $\mathcal{O}(n^2f)$ [22,53] due to the k-nearest neighbors search. Moreover, this complexity is similar to the complexity of NDO, which is the only oversampling method reported in the literature that is not based on computing the k-nearest neighbors and therefore that can be applied to large datasets.

Algorithm 2 Fast-SMOTE

```
Input: A minority class, $N$ - number of synthetic objects to be generated.
Output: An oversampled minority class.
  $MIN \leftarrow$ Minimum value for each feature into the minority class.
  $MAX \leftarrow$ Maximum value for each feature into the minority class.
  $MODE \leftarrow$ Mode value for each feature into the minority class.
  $MinDif \leftarrow MODE - MIN$ Absolute difference between the mode and the minimum
  value for each feature into the minority class.
  $MaxDif \leftarrow MAX - MODE$ Absolute difference between the mode and the maximum
  value for each feature into the minority class.
  for $j = 1, ..., N$ do
      $Syn \leftarrow$ Initialize an empty object
      for each feature $f$ do
          $r \leftarrow Random(0, 1)$
          $r2 \leftarrow Random(0, 1)$
          if $r2 < 0.5$ then
              $gap \leftarrow MinDif_f * r$
          else
              $gap \leftarrow MaxDif_f * r$
          end if
          $Syn_f \leftarrow MODE_f + gap$
      end for
      Add $Syn$ to the minority class.
  end for
```

4. Experimental Results

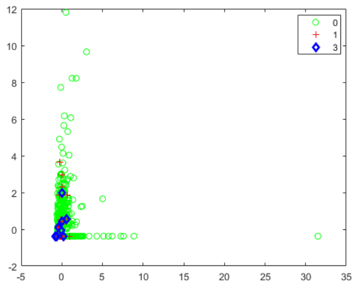
In this section, we first present an experiment to show the synthetic objects generated by SMOTE, NDO and our proposed method Fast-SMOTE. Then, in the second experiment, we evaluate the proposed method by comparing it to the methods described in the related work section using standard unbalanced public databases. Since these databases are relatively small, in the third experiment, we compared the proposed method to the fastest methods in the previous experiment but for larger datasets. Finally, in the last experiment, we demonstrate the scalability of the proposed method.

4.1. Synthetic Objects Generated by SMOTE, NDO and Fast-SMOTE

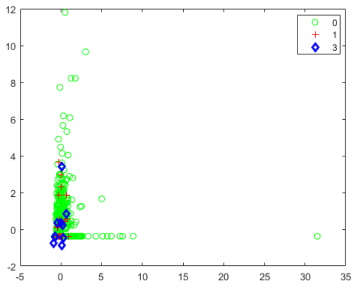
This experiment applies SMOTE, NDO, and Fast-SMOTE over three real datasets to show the synthetic objects generated by each method.

For this experiment, we used the mammography dataset that was also used in [54] for this purpose, and we included the glass6 and ecoli1 datasets from the KEEL repository [55]. For a better appreciation of the plotted objects, for this experiment, we selected only 10% of majority class objects and 10% of minority class objects. In this way, the IR of each dataset is preserved.

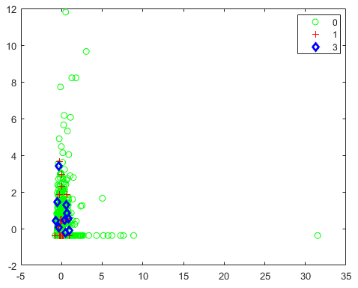
To show the synthetic objects generated by each method, in Figures 1–3 we plotted, in a 2-D projections, the mammography, glass, and ecoli datasets respectively, together with 10 synthetic objects generated by each method. The mammography dataset was projected on the fourth and fifth features, the glass6 dataset on the *aluminum* and *silicon* features, and the ecoli1 dataset on the *Mcg* and *Alm*2 features. In all three figures, green circles represent the objects of the majority class, and red crosses represent the objects of the minority class. Also, we show in Table 1 the mode values computed by the proposed method on each dataset (see rows) for axis x and y (see columns) for the Figures 1–3.



(**a**) SMOTE/mammography

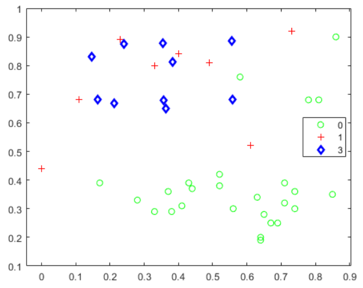


(**b**) NDO/mammography

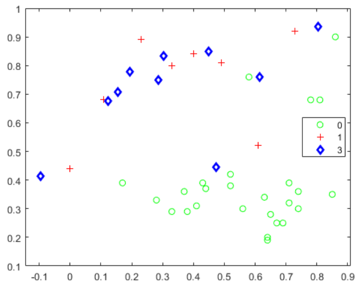


(**c**) Fast-SMOTE/mammography

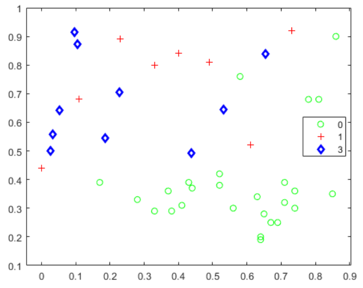
Figure 1. Synthetic objects (blue diamonds) generated by SMOTE (**a**), NDO (**b**), and Fast-SMOTE (**c**) over a 2-D projection of the dataset Mammography. Green circles represent objects of the majority class, and red crosses represent objects of the minority class.



(**a**) SMOTE/Glass

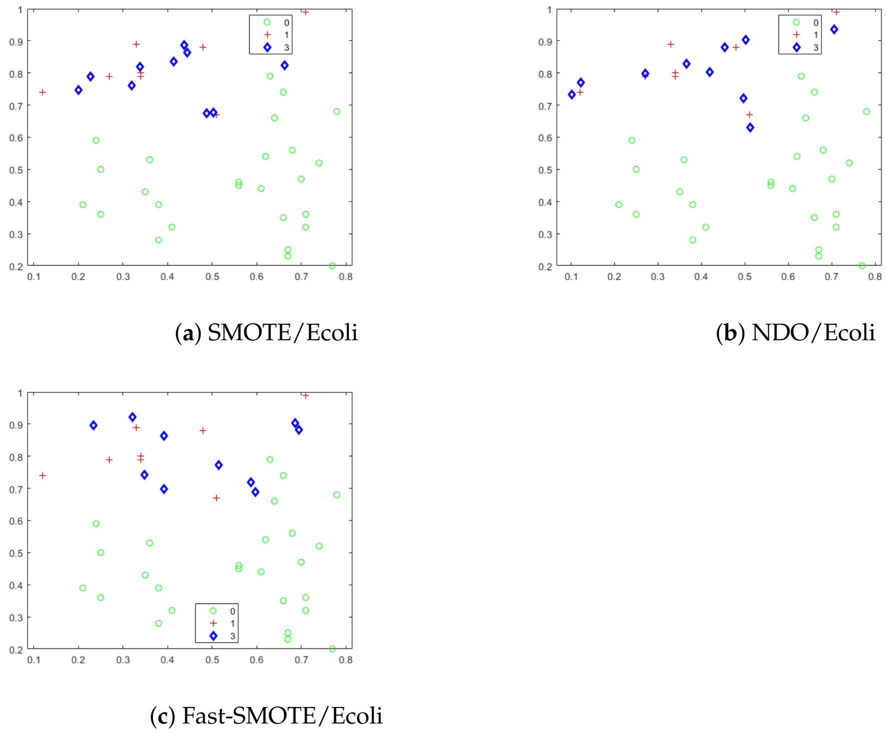


(**b**) NDO/Glass



(**c**) Fast-SMOTE/Glass

Figure 2. Synthetic objects (blue diamonds) generated by SMOTE (**a**), NDO (**b**), and Fast-SMOTE (**c**) over a 2-D projection of the dataset Glass. Green circles represent objects of the majority class, and red crosses represent objects of the minority class.



(**a**) SMOTE/Ecoli (**b**) NDO/Ecoli (**c**) Fast-SMOTE/Ecoli

Figure 3. Synthetic objects (blue diamonds) generated by SMOTE (**a**), NDO (**b**), and Fast-SMOTE (**c**) over a 2-D projection of the dataset Ecoli. Green circles represent objects of the majority class, and red crosses represent objects of the minority class.

Table 1. Mode values of x and y axis for for Figures 1–3.

	Features Mode	
	x	y
mamography	−0.55	0.06
glass	0.42	0.73
ecoli	0.32	0.79

In Figures 1–3, we can see that Fast-SMOTE, as well as SMOTE and NDO generate synthetic objects close to the objects of the minority class.

4.2. Comparison against State of the Art Oversampling Methods

In this experiment, we compare the proposed method Fast-SMOTE to the state-of-the-art oversampling methods in terms of oversampling quality and runtime. In this experiment, we used small-medium size datasets to allow applying SMOTE and other state of the art methods. We used all 66 imbalanced datasets from the KEEL repository [55] (shown in Table 2), and assessed the oversampling quality of the oversampling methods. This was done by oversampling a repository training dataset until reaching a full balance between classes and training a classifier with the oversampled dataset produced by an oversampling method. The testing set that is also available in the repository was used to evaluate the trained classifier. The quality of the classification result is interpreted as the oversampling quality of the employed oversampling method.

Table 2. Datasets from the KEEL repository used in our experiments.

Dataset	# Features	# Objects	IR	Dataset	# Features	# Objects	IR
glass1	9	214	1.82	glass-0-4_vs_5	9	92	9.22
ecoli-0_vs_1	7	220	1.86	ecoli-0-3-4-6_vs_5	7	205	9.25
wisconsin	9	683	1.86	ecoli-0-3-4-7_vs_5-6	7	257	9.28
pima	8	768	1.87	yeast-0-5-6-7-9_vs_4	8	528	9.35
iris0	4	150	2.00	vowel0	13	988	9.98
glass0	9	214	2.06	ecoli-0-6-7_vs_5	6	220	10.00
yeast1	8	1484	2.46	glass-0-1-6_vs_2	9	192	10.29
haberman	3	306	2.78	ecoli-0-1-4-7_vs_2-3-5-6	7	336	10.59
vehicle2	18	846	2.88	led7digit-0-2-4-5-6-7-8-9_vs_1	7	443	10.97
vehicle1	18	846	2.90	glass-0-6_vs_5	9	108	11.00
vehicle3	18	846	2.99	ecoli-0-1_vs_5	6	240	11.00
glass-0-1-2-3_vs_4-5-6	9	214	3.20	glass-0-1-4-6_vs_2	9	205	11.06
vehicle0	18	846	3.25	glass2	9	214	11.59
ecoli1	7	336	3.36	ecoli-0-1-4-7_vs_5-6	6	332	12.28
new-thyroid1	5	215	5.14	cleveland-0_vs_4	13	177	12.62
new-thyroid2	5	215	5.14	ecoli-0-1-4-6_vs_5	6	280	13.00
ecoli2	7	336	5.46	shuttle-c0-vs-c4	9	1829	13.87
segment0	19	2308	6.02	yeast-1_vs_7	7	459	14.30
glass6	9	214	6.38	glass4	9	214	15.47
yeast3	8	1484	8.10	ecoli4	7	336	15.80
ecoli3	7	336	8.60	page-blocks-1-3_vs_4	10	472	15.86
page-blocks0	10	5472	8.79	abalone9-18	8	731	16.40
ecoli-0-3-4_vs_5	7	200	9.00	glass-0-1-6_vs_5	9	184	19.44
yeast-2_vs_4	8	514	9.08	shuttle-c2-vs-c4	9	129	20.50
ecoli-0-6-7_vs_3-5	7	222	9.09	yeast-1-4-5-8_vs_7	8	693	22.10
ecoli-0-2-3-4_vs_5	7	202	9.10	glass5	9	214	22.78
glass-0-1-5_vs_2	9	172	9.12	yeast-2_vs_8	8	482	23.10
yeast-0-3-5-9_vs_7-8	8	506	9.12	yeast4	8	1484	28.10
yeast-0-2-5-7-9_vs_3-6-8	8	1004	9.14	yeast-1-2-8-9_vs_7	8	947	30.57
yeast-0-2-5-6_vs_3-7-8-9	8	1004	9.14	yeast5	8	1484	32.73
ecoli-0-4-6_vs_5	6	203	9.15	ecoli-0-1-3-7_vs_2-6	7	281	39.14
ecoli-0-1_vs_2-3-5	7	244	9.17	yeast6	8	1484	41.40
ecoli-0-2-6-7_vs_3-5	7	224	9.18	abalone19	8	4174	129.44

Since all oversampling methods generate random numbers when a synthetic object is built, the oversampling methods were applied 30 times for each one of the 66 imbalanced datasets using the 5-fold cross-validation partition available in the repository. We used the supervised classifiers CART (Classification and Regresion Tree), KNN (K-Nearest Neighbor Classifier) (K = 5), and Naïve Bayes. Additionally, as suggested in [56], we used AUC (Area Under Curve) to assess the classification results, the most used measure for evaluating classification quality on imbalance class problems.

Table 3 shows the average AUC of the 150 results (30 repetitions for each one of the 5 folds) obtained with each classifier (CART, KNN, and Naïve Bayes; see rows) over the 66 datasets for each oversampling method (see columns). As mentioned above, most of the compared oversampling methods are based on finding the *k*-nearest neighbors; it is observed from the results shown in Table 3 that these methods obtain the best AUC results. In particular, we can observe that SMOTE and S-RSB* obtained the best average oversampling quality. On the other hand, the methods that are not based on finding the nearest neighbors (NDO and Fast-SMOTE) do not obtain the lowest oversampling quality. The methods that obtain the worst AUC results were K-Means-S and G-SMOTE. The proposed method and NDO, obtain lower average AUC results than SMOTE and S-RSB* although unlike these methods, Fast-SMOTE and NDO can be applied on large datasets.

Additionally, in this experiment we assessed the runtime employed by the oversampling methods. This experiment was run in MATLAB 2020b, using a computer with a Ryzen 5300X 3.60 GHz processor with 32 GB DDR4 RAM, running 64-bit Windows 10.

Table 3. Average AUC results of CART, KNN and Naive Bayes classifiers for the 66 datasets from the KEEL repository.

Classifier	**Oversampling Method**										
	SMOTE	**Borderline**	**SafeLevel**	**LNE**	**NDO**	**S-RSB***	**K-Means-S**	**Farthest-S**	**G-SMOTE**	**S-SF**	**Fast-SMOTE**
CART	0.825	0.807	0.808	0.814	0.817	0.824	0.802	0.819	0.798	0.821	0.801
KNN	0.857	0.842	0.822	0.836	0.853	0.857	0.789	0.840	0.800	0.854	0.822
Naïve Bayes	0.808	0.775	0.776	0.802	0.786	0.807	0.755	0.801	0.731	0.808	0.772

Table 4 shows the runtime in seconds to fully balance each KEEL data set with all oversampling methods. An examination of the data presented in the table shows that S-RSB* was the most time-consiming oversampling method. By contrast, the proposed Fast-SMOTE method is clearly the fastest of all the oversampling methods under comparison. The closest method in runtime to our proposed method is NDO, but it was more than twice as time-consuming as the proposed method.

Table 4. Runtime in seconds spent by the oversampling methods revised in Section 2 for oversampling all datasets from Table 2.

Dataset	**Oversampling Method**										
	SMOTE	**Borderline**	**SafeLevel**	**LNE**	**NDO**	**S-RSB***	**K-Means-S**	**Farthest-S**	**G-SMOTE**	**S-SF**	**Fast-SMOTE**
glass-0-4_vs_5	0.0027	0.0006	0.0056	0.0040	0.0002	0.0045	0.0035	0.0025	0.0037	0.0003	0.0001
glass-0-6_vs_5	0.0033	0.0007	0.0069	0.0048	0.0003	0.0054	0.0040	0.0031	0.0044	0.0002	0.0001
shuttle-c2-vs-c4	0.0041	0.0004	0.0094	0.0058	0.0004	0.0067	0.0052	0.0040	0.0062	0.0003	0.0002
iris0	0.0017	0.0016	0.0039	0.0038	0.0002	0.0030	0.0028	0.0017	0.0024	0.0011	0.0001
glass-0-1-5_vs_2	0.0052	0.0011	0.0113	0.0076	0.0004	0.0083	0.0068	0.0049	0.0070	0.0004	0.0002
cleveland-0_vs_4	0.0054	0.0009	0.0124	0.0081	0.0005	0.0096	0.0078	0.0051	0.0072	0.0004	0.0002
glass-0-1-6_vs_5	0.0059	0.0007	0.0136	0.0083	0.0005	0.0101	0.0069	0.0057	0.0080	0.0004	0.0002
glass-0-1-6_vs_2	0.0056	0.0010	0.0132	0.0084	0.0005	0.0102	0.0070	0.0057	0.0076	0.0005	0.0002
ecoli-0-3-4_vs_5	0.0062	0.0009	0.0132	0.0083	0.0005	0.0091	0.0086	0.0055	0.0076	0.0005	0.0002
ecoli-0-2-3-4_vs_5	0.0062	0.0009	0.0136	0.0092	0.0005	0.0097	0.0088	0.0055	0.0082	0.0005	0.0002
ecoli-0-4-6_vs_5	0.0060	0.0009	0.0132	0.0091	0.0005	0.0091	0.0079	0.0055	0.0080	0.0005	0.0002
ecoli-0-3-4-6_vs_5	0.0061	0.0009	0.0136	0.0091	0.0005	0.0093	0.0081	0.0057	0.0082	0.0005	0.0002
glass-0-1-4-6_vs_2	0.0066	0.0012	0.0144	0.0092	0.0005	0.0107	0.0082	0.0062	0.0088	0.0004	0.0002
glass1	0.0023	0.0032	0.0052	0.0062	0.0002	0.0061	0.0035	0.0022	0.0031	0.0026	0.0001
glass2	0.0065	0.0010	0.0152	0.0099	0.0005	0.0113	0.0087	0.0065	0.0087	0.0005	0.0003
glass0	0.0026	0.0031	0.0059	0.0092	0.0002	0.0068	0.0038	0.0026	0.0036	0.0024	0.0001
glass4	0.0066	0.0008	0.0156	0.0101	0.0006	0.0117	0.0087	0.0065	0.0090	0.0005	0.0003
glass-0-1-2-3_vs_4-5-6	0.0041	0.0022	0.0093	0.0090	0.0003	0.0089	0.0051	0.0040	0.0056	0.0014	0.0002
glass5	0.0070	0.0006	0.0161	0.0100	0.0006	0.0125	0.0085	0.0067	0.0094	0.0004	0.0003
glass6	0.0055	0.0009	0.0125	0.0092	0.0005	0.0095	0.0067	0.0054	0.0076	0.0007	0.0002
new-thyroid1	0.0051	0.0017	0.0116	0.0091	0.0004	0.0074	0.0063	0.0050	0.0069	0.0008	0.0002
newthyroid2	0.0051	0.0016	0.0114	0.0091	0.0004	0.0073	0.0066	0.0050	0.0068	0.0008	0.0002
ecoli-0_vs_1	0.0109	0.0028	0.0069	0.0176	0.0002	0.0058	0.0047	0.0024	0.0049	0.0026	0.0001
ecoli-0-6-7_vs_5	0.0066	0.0010	0.0149	0.0090	0.0005	0.0098	0.0084	0.0061	0.0087	0.0005	0.0003
ecoli-0-6-7_vs_3-5	0.0066	0.0011	0.0150	0.0098	0.0005	0.0102	0.0085	0.0061	0.0088	0.0005	0.0003
ecoli-0-2-6-7_vs_3-5	0.0066	0.0012	0.0153	0.0099	0.0005	0.0105	0.0089	0.0063	0.0093	0.0006	0.0003
ecoli-0-1_vs_5	0.0077	0.0010	0.0170	0.0100	0.0006	0.0112	0.0093	0.0069	0.0094	0.0005	0.0003
ecoli-0-1_vs_2-3-5	0.0073	0.0012	0.0172	0.0111	0.0006	0.0113	0.0100	0.0068	0.0096	0.0006	0.0003
ecoli-0-3-4-7_vs_5-6	0.0082	0.0014	0.0177	0.0117	0.0006	0.0126	0.0104	0.0071	0.0099	0.0006	0.0003
ecoli-0-1-4-6_vs_5	0.0088	0.0010	0.0209	0.0118	0.0007	0.0137	0.0122	0.0082	0.0113	0.0006	0.0003
ecoli-0-1-3-7_vs_2-6	0.0096	0.0006	0.0224	0.0132	0.0008	0.0152	0.0111	0.0092	0.0127	0.0005	0.0004
haberman	0.0052	0.0042	0.0124	0.0099	0.0004	0.0186	0.0059	0.0050	0.0069	0.0027	0.0002
ecoli-0-1-4-7_vs_5-6	0.0104	0.0013	0.0257	0.0151	0.0008	0.0166	0.0139	0.0096	0.0137	0.0007	0.0004
ecoli4	0.0105	0.0012	0.0269	0.0150	0.0009	0.0179	0.0121	0.0101	0.0141	0.0007	0.0004
ecoli1	0.0072	0.0035	0.0170	0.0148	0.0005	0.0118	0.0079	0.0074	0.0096	0.0028	0.0003
ecoli-0-1-4-7_vs_2-3-5-6	0.0105	0.0015	0.0255	0.0147	0.0008	0.0172	0.0139	0.0096	0.0136	0.0008	0.0004
ecoli2	0.0086	0.0018	0.0207	0.0143	0.0007	0.0160	0.0096	0.0083	0.0115	0.0017	0.0003
ecoli3	0.0092	0.0021	0.0235	0.0147	0.0008	0.0164	0.0107	0.0092	0.0127	0.0010	0.0004
led7digit-0-2-4-5-6-7-8-9_vs_1	0.0138	0.0019	0.0369	0.0215	0.0011	33.1945	0.0162	0.0129	0.0189	0.0012	0.0005
yeast-1_vs_7	0.0142	0.0021	0.0424	0.0225	0.0012	0.0294	0.0171	0.0140	0.0192	0.0011	0.0006
page-blocks-1-3_vs_4	0.0153	0.0020	0.0443	0.0244	0.0013	0.0386	0.0171	0.0148	0.0203	0.0012	0.0007
yeast-2_vs_8	0.0167	0.0016	0.0471	0.0244	0.0013	0.0369	0.0199	0.0155	0.0218	0.0011	0.0007

Table 4. *Cont.*

Dataset	Oversampling Method										
	SMOTE	**Borderline**	**SafeLevel**	**LNE**	**NDO**	**S-RSB***	**K-Means-S**	**Farthest-S**	**G-SMOTE**	**S-SF**	**Fast-SMOTE**
yeast-0-3-5-9_vs_7-8	0.0154	0.0032	0.0431	0.0253	0.0012	0.0339	0.0180	0.0142	0.0197	0.0018	0.0006
yeast-2_vs_4	0.0155	0.0030	0.0451	0.0254	0.0013	0.0358	0.0196	0.0146	0.0226	0.0018	0.0006
yeast-0-5-6-7-9_vs_4	0.0159	0.0035	0.0471	0.0255	0.0013	0.0352	0.0178	0.0151	0.0225	0.0019	0.0006
wisconsin	0.0079	0.0119	0.0306	0.0271	0.0006	10.9796	0.0095	0.0077	0.0106	0.0241	0.0003
yeast-1-4-5-8_vs_7	0.0235	0.0032	0.0901	0.0428	0.0020	0.0630	0.0305	0.0227	0.0312	0.0017	0.0010
abalone9-18	0.0243	0.0037	0.0929	0.0436	0.0021	0.0700	0.0267	0.0233	0.0320	0.0021	0.0010
pima	0.0091	0.0170	0.0346	0.0338	0.0007	0.0332	0.0107	0.0087	0.0118	0.0289	0.0003
vehicle2	0.0175	0.0167	0.0745	0.0507	0.0016	0.0877	0.0205	0.0175	0.0232	0.0236	0.0008
vehicle1	0.0178	0.0184	0.0750	0.0501	0.0016	0.0877	0.0185	0.0174	0.0234	0.0230	0.0008
vehicle3	0.0181	0.0190	0.0767	0.0483	0.0016	0.0898	0.0202	0.0179	0.0238	0.0221	0.0008
vehicle0	0.0192	0.0143	0.0823	0.0625	0.0017	0.0912	0.0221	0.0188	0.0254	0.0199	0.0008
yeast-1-2-8-9_vs_7	0.0352	0.0039	0.1494	0.0618	0.0030	0.1203	0.0413	0.0324	0.0444	0.0025	0.0015
vowel0	0.0320	0.0076	0.1454	0.0656	0.0032	0.1441	0.0344	0.0310	0.0430	0.0061	0.0016
yeast-0-2-5-7-9_vs_3-6-8	0.0316	0.0070	0.1407	0.0654	0.0027	0.1101	0.0363	0.0300	0.0421	0.0059	0.0013
yeast-0-2-5-6_vs_3-7-8-9	0.0318	0.0074	0.1427	0.0677	0.0027	0.1092	0.0365	0.0300	0.0411	0.0059	0.0013
yeast1	0.0265	0.0347	0.1342	0.0896	0.0020	0.1277	0.0292	0.0260	0.0342	0.0747	0.0010
yeast4	0.0567	0.0080	0.2914	0.1046	0.0054	0.3084	0.0742	0.0523	0.0699	0.0052	0.0027
yeast5	0.0562	0.0071	0.2921	0.1067	0.0054	0.3100	0.0698	0.0525	0.0706	0.0050	0.0027
yeast3	0.0456	0.0136	0.2471	0.1047	0.0043	0.2384	0.0508	0.0448	0.0596	0.0141	0.0021
yeast6	0.0566	0.0062	0.3081	0.1069	0.0056	0.3173	0.0803	0.0528	0.0711	0.0048	0.0028
shuttle-c0-vs-c4	0.0634	0.0083	0.4022	0.1428	0.0068	0.4615	0.0018	0.0622	0.0864	0.0119	0.0033
segment0	0.0821	0.0466	0.5992	0.3141	0.0105	1.1618	0.0894	0.0807	0.1045	0.0597	0.0052
abalone19	0.1695	0.0255	1.8756	0.4817	0.0270	19.3115	0.1816	0.1679	0.2192	0.0262	0.0134
page-blocks0	0.2206	0.1040	2.5987	0.7628	0.0344	28.2546	0.1910	0.2177	0.2792	0.1586	0.0171
Sum	1.3955	0.4553	8.7356	3.3823	0.1522	96.2835	1.5021	1.3391	0.5366	0.5696	0.0754

Based on this experiment, we conclude that the methods that obtain the best oversampling quality in the previous experiment are among the slowest. By contrast, the fastest methods obtain lower oversampling quality, although it is similar to that of the methods that obtain the best oversampling results. However, the KEEL repository databases are relatively small and therefore do not provide a good assessment of the runtime advantage of the proposed method. Hence, in the following experiment, the proposed method is compared to NDO that is the second fastest method in the previous experiment but for larger datasets.

4.3. Assessing the Proposed Method in Large Datasets

In this experiment, we compared the proposed method to the fastest method in the previous experiment, namely NDO, but on larger datasets taken from the UCI repository [57] (see Table 5); these datasets are 5, 7, 10, 44, and 100 times larger than the largest dataset used in the previous experiments (page-blocks0).

Table 5. Large datasets from the UCI repository used in our experiments.

Dataset	# Features	# Objects	IR
Default of credit card clients	24	30,000	3.52
Online News Popularity	61	39,797	3.90
Statlog (Shuttle)	9	58,000	3.67
Skin segmentation	3	245,057	3.81
Buzz in social media (Twitter)	78	583,250	3.80

We followed the same evaluation scheme to measure the oversampling quality of the methods as in the previous experiment. However, given the datasets' size, the oversampling methods were applied only 10 times on each imbalanced dataset of Table 5 with a 5-fold cross-validation partition. Additionally, we only used the CART classifier because it can produce good results in a reasonable time for this size of the datasets compared with the other two supervised classifiers.

Table 6 shows the average AUC of the 50 results (10 repetitions for each one of the 5 folds) obtained from each dataset in Table 5 for NDO and Fast-SMOTE. As observed from the data presented in this table, the proposed method and NDO obtain quite similar oversampling quality results for all of the datasets. To validate these results, we applied the Wilcoxon test, which showed that Fast-SMOTE and NDO were statistically similar on the first three datasets (Default of credit card clients, Online News Popularity, and Statlog (Shuttle)). Meanwhile, for the Skin segmentation dataset, Fast-SMOTE was statistically better than NDO, and for the Buzz in social media (Twitter) dataset, NDO was statistically better than Fast-SMOTE. On the other hand, in Table 7, we show the runtime spent by NDO and Fast-SMOTE for fully balancing every dataset in Table 5. Table 7 shows that the proposed method's runtime is approximately 50% lower than NDO's runtime. Moreover, the difference grows with increasing the size of the dataset. For example, for the largest dataset, the dataset with 583,250 objects (153,487 objects in the minority class, a large dataset, although not so large) and 78 features, Fast-SMOTE only require 2860.4 s, while NDO require 8771.89 s, in this case, NDO spent more than three times longer time than our method.

Table 6. AUC results of Cart classifier for one of the 5 datasets from the UCI repository.

Dataset	**NDO**	**Fast-SMOTE**
Default of credit card clients	0.620	0.623
Online News Popularity	0.551	0.551
Statlog (Shuttle)	0.549	0.551
Skin segmentation	0.990	0.999
Buzz in social media (Twitter)	0.922	0.921

Although we could include larger datasets, the datasets of Table 5 are sufficient for showing that the proposed method is clearly faster than NDO on large datasets.

Table 7. Runtime in seconds spent by NDO and Fast-SMOTE on artificial datasets from Table 5.

Dataset	**Oversampling Method**	
	NDO	**Fast-SMOTE**
Default of credit card clients	0.81	0.40
Online News Popularity	4.94	1.95
Statlog (Shuttle)	1.61	0.62
Skin segmentation	16.47	5.80
Buzz in social media (Twitter)	8771.89	2860.40
Sum	8795.17	2869.17

4.4. Scalability

In this experiment, we evaluate the scalability of the proposed method Fast-SMOTE. Here, we randomly generated two artificial datasets with numeric features in [0,1], one with 10,000,000 objects and 10 features, and another with 10,000 features and 10,000 objects. Then, to evaluate the scalability of the proposed method regarding the number of objects, we measure the runtime required for oversampling minority classes from one million to 10 million of objects with increments of 2 million of objects, taken from the artificial dataset with 10 features. Additionally, to evaluate the scalability of Fast-SMOTE regarding the number of features, we measure the runtime required for oversampling minority classes with 1000 to 10,000 features with increments of 1000 features from the artificial dataset with 10,000 objects. Due to the size of the minority classes that makes it unmanageable in MATLAB, for this experiment, Fast-SMOTE and NDO were implemented in C++. The experiment was run on a computer with an Intel Core i7-3820 3.60 GHz processor with 64 GB DDR4 RAM, running 64-bit Ubuntu. Although we could use larger datasets, we selected these datasets because they are sufficient for demonstrating the scalability of Fast-SMOTE.

Figure 4 shows a graph of the scalability of the proposed method Fast-SMOTE with respect to the number of objects. For comparison purposes, we include NDO in the graph. In this graph, the x-axis shows the number of objects of the minority classes used in this experiment, while the y-axis shows the runtime in seconds for FAST-SMOTE (blue line) and NDO (red line) for oversampling the respective minority class. From this figure, we can observe that Fast-SMOTE scales better than NDO with increasing number of objects, allowing oversampling larger datasets in a much shorter runtime.

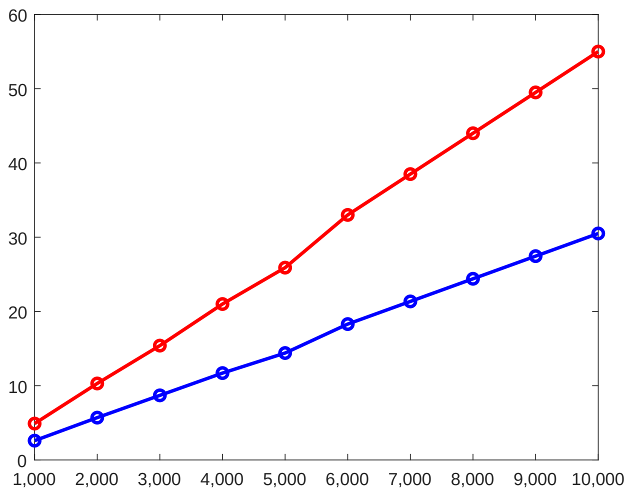


Figure 4. Runtime in seconds (y-axis) for oversampling different numbers of objects in the minority class (x-axis) by Fast-SMOTE (blue line) and NDO (red line).

Figure 5 shows a graph of the scalability of Fast-SMOTE with respect to the number of features. We also included NDO results in the graph. In this graph, the x-axis shows the number of features that describe the objects of the minority classes used in this experiment, while the y-axis shows the runtime in seconds required by FAST-SMOTE (blue line) and NDO (red line) for oversampling the respective minority class. It is observed from this figure that Fast-SMOTE scales better than NDO with increasing number of features.

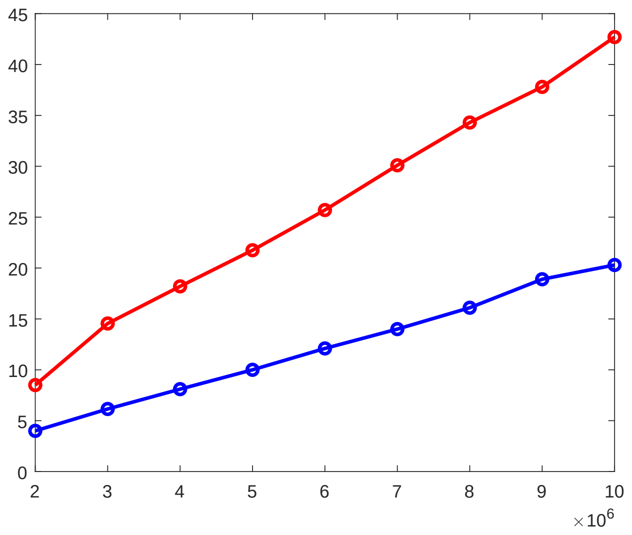


Figure 5. Runtime in seconds (y-axis) for oversampling a minority class with different number of features (x-axis) by Fast-SMOTE (blue line) and NDO (red line).

This experiment shows that when the number of objects and variables grows, Fast SMOTE has better scalability than NDO, making it more suitable than NDO for practical use.

5. Conclusions

In this paper, we introduced Fast-SMOTE, a fast oversampling method non based on the k-nearest neighbors search that is the most time-expensive step of SMOTE-based methods. For each feature, the proposed method uses the value that appears most often in the minority class (the mode) jointly with the minimum and maximum values (all three can be quickly computed), in an appropriate manner, for the generation of synthetic objects around the mode in the region of the minority class defined by the minimum and maximum values. This allowed us to obtain a linear complexity oversampling method that is much faster than SMOTE-based methods, which complexity is at least quadratic. Moreover, the proposed method has a complexity similar to the complexity of NDO, which is the only oversampling method reported in the literature that is not based on the k-nearest neighbors, however according to our experiments, the proposed method scales better when the number of objects and features increases.

Comparing the synthetic objects generated by the proposed method against those generated by SMOTE and NDO, we have shown that similar to SMOTE and NDO, Fast-SMOTE generate synthetic objects close to the objects of the minority class and near the decision region between the classes.

In small unbalanced datasets where state-of-the-art oversampling methods can oversample the minority class in a short time, the proposed method obtains an oversampling quality similar to these method but it is found to be the fastest method.

Our comparison between Fast-SMOTE and NDO for large datasets with regard to runtime and AUC shows that Fast-SMOTE produces oversampled datasets that allow training supervised classifiers to obtain classification results that are statistically similar to those classification results obtained by training the classifiers with datasets oversampled by NDO, but in much less runtime. From all of these experiments, we can conclude that Fast-SMOTE is the best method for oversampling large datasets with imbalanced classes.

As mentioned above, the development of oversampling methods not based on the search for nearest neighbors is a poorly explored research direction. Hence, in future work, we propose to continue developing oversampling methods following this approach. In particular, we are interested in developing oversampling methods that can work on large mixed datasets with imbalanced classes. Additionally, the implementation and development of oversampling methods on GPU, parallel CPU or distributed computing [58–62] is an active research area. Thus, in future work, we will develop an implementation of our oversampling method on these platforms to improve its runtime further. Also, we will face the imbalance problem with other problems in supervised classification as noise, missing data, or multiple minority class groups.

Author Contributions: Writing—original draft preparation, F.R.-T.; writing—review and editing, F.R.-T., J.F.M.-T. and J.A.C.-O.; supervision, J.F.M.-T. and J.A.C.-O. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partly funded by the National Council of Science and Technology of Mexico.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The first author gratefully acknowledges the National Council of Science and Technology of Mexico (CONACyT) for his Ph.D. fellowship through a scholarship.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Moscato, V.; Picariello, A.; Sperlí, G. A benchmark of machine learning approaches for credit score prediction. *Expert Syst. Appl.* **2021**, *165*, 113986. [CrossRef]
2. Du, G.; Zhang, J.; Li, S.; Li, C. Learning from class-imbalance and heterogeneous data for 30-day hospital readmission. *Neurocomputing* **2021**, *420*, 27–35. [CrossRef]
3. Eivazpour, Z.; Keyvanpour, M.R. CSSG: A cost-sensitive stacked generalization approach for software defect prediction. *Softw. Test. Verif. Reliab.* **2021**, *31*, e1761. [CrossRef]
4. Srinivasan, R.; Subalalitha, C. Sentimental analysis from imbalanced code-mixed data using machine learning approaches. *Distrib. Parallel Databases* **2021**, Volume 39, 1–16. [CrossRef]
5. Hussin, S.K.; Abdelmageid, S.M.; Alkhalil, A.; Omar, Y.M.; Marie, M.I.; Ramadan, R.A. Handling Imbalance Classification Virtual Screening Big Data Using Machine Learning Algorithms. *Complexity* **2021**, *2021*, 6675279. [CrossRef]
6. Vilalta, R.; Rish, I. A decomposition of classes via clustering to explain and improve naive bayes. In *European Conference on Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 444–455.
7. Elyan, E.; Gaber, M.M. A fine-grained random forests using class decomposition: An application to medical diagnosis. *Neural Comput. Appl.* **2016**, *27*, 2279–2288. [CrossRef]
8. Chawla, N.V.; Bowyer, K.W.; Hall, L.O.; Kegelmeyer, W.P. SMOTE: Synthetic minority over-sampling technique. *J. Artif. Intell. Res.* **2002**, *16*, 321–357. [CrossRef]
9. Han, H.; Wang, W.Y.; Mao, B.H. Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. In *International Conference on Intelligent Computing*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 878–887.
10. Hu, S.; Liang, Y.; Ma, L.; He, Y. MSMOTE: Improving classification performance when training data is imbalanced. In Proceedings of the 2009 Second International Workshop on Computer Science and Engineering, Qingdao, China, 28–30 October 2009; Volume 2, pp. 13–17.
11. Bunkhumpornpat, C.; Sinapiromsaran, K.; Lursinsap, C. Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In *Pacific-Asia Conference on Knowledge Discovery and data Mining*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 475–482.
12. Maciejewski, T.; Stefanowski, J. Local neighbourhood extension of SMOTE for mining imbalanced data. In Proceedings of the 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), Paris, France, 11–15 April 2011; pp. 104–111.
13. Ramentol, E.; Caballero, Y.; Bello, R.; Herrera, F. SMOTE-RSB*: A hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using SMOTE and rough sets theory. *Knowl. Inf. Syst.* **2012**, *33*, 245–265. [CrossRef]
14. Abdi, L.; Hashemi, S. To Combat Multi-Class Imbalanced Problems by Means of Over-Sampling Techniques. *IEEE Trans. Knowl. Data Eng.* **2016**, *28*, 238–251. [CrossRef]
15. Torres, F.R.; Carrasco-Ochoa, J.A.; Martínez-Trinidad, J.F. SMOTE-D a deterministic version of SMOTE. In *Mexican Conference on Pattern Recognition*; Springer: Cham, Switzerland, 2016; pp. 177–188.
16. Borowska, K.; Stepaniuk, J. Imbalanced data classification: A novel re-sampling approach combining versatile improved SMOTE and rough sets. In *IFIP International Conference on Computer Information Systems and Industrial Management*; Springer: Cham, Switzerland, 2016; pp. 31–42.
17. Gong, C.; Gu, L. A Novel SMOTE-Based Classification Approach to Online Data Imbalance Problem. *Math. Probl. Eng.* **2016**, *2016*, 5685970. [CrossRef]
18. Jiang, K.; Lu, J.; Xia, K. A novel algorithm for imbalance data classification based on genetic algorithm improved SMOTE. *Arab. J. Sci. Eng.* **2016**, *41*, 3255–3266. [CrossRef]
19. Yun, J.; Ha, J.; Lee, J.S. Automatic determination of neighborhood size in SMOTE. In Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication, Danang, Vietnam, 4–6 January 2016; p. 100.
20. Pérez-Ortiz, M.; Gutiérrez, P.A.; Tino, P.; Hervás-Martínez, C. Oversampling the minority class in the feature space. *IEEE Trans. Neural Netw. Learn. Syst.* **2016**, *27*, 1947–1961. [CrossRef] [PubMed]
21. Rivera, W.A.; Xanthopoulos, P. A priori synthetic over-sampling methods for increasing classification sensitivity in imbalanced data sets. *Expert Syst. Appl.* **2016**, *66*, 124–135. [CrossRef]
22. Cervantes, J.; Garcia-Lamont, F.; Rodriguez, L.; López, A.; Castilla, J.R.; Trueba, A. PSO-based method for SVM classification on skewed data sets. *Neurocomputing* **2017**, *228*, 187–197. [CrossRef]
23. Douzas, G.; Bacao, F. Self-Organizing Map Oversampling (SOMO) for imbalanced data set learning. *Expert Syst. Appl.* **2017**, *82*, 40–52. [CrossRef]
24. Douzas, G.; Bacao, F.; Last, F. Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE. *Inf. Sci.* **2018**, *465*, 1–20. [CrossRef]
25. Nekooeimehr, I.; Lai-Yuen, S.K. Adaptive semi-unsupervised weighted oversampling (A-SUWO) for imbalanced datasets. *Expert Syst. Appl.* **2016**, *46*, 405–416. [CrossRef]
26. Li, J.; Fong, S.; Wong, R.K.; Chu, V.W. Adaptive multi-objective swarm fusion for imbalanced data classification. *Inf. Fusion* **2018**, *39*, 1–24. [CrossRef]
27. Gosain, A.; Sardana, S. Farthest SMOTE: A Modified SMOTE Approach. In *Computational Intelligence in Data Mining*; Springer: Singapore, 2019; pp. 309–320.

28. Douzas, G.; Bacao, F. Geometric SMOTE: Effective oversampling for imbalanced learning through a geometric extension of SMOTE. *arXiv* **2017**, arXiv:1709.07377.
29. Maldonado, S.; López, J.; Vairetti, C. An alternative SMOTE oversampling strategy for high-dimensional datasets. *Appl. Soft Comput.* **2019**, *76*, 380–389. [CrossRef]
30. Elyan, E.; Moreno-Garcia, C.F.; Jayne, C. CDSMOTE: Class decomposition and synthetic minority class oversampling technique for imbalanced-data classification. *Neural Comput. Appl.* **2021**, *33*, 2839–2851. [CrossRef]
31. Zhang, Y.; Zuo, T.; Fang, L.; Li, J.; Xing, Z. An Improved MAHAKIL Oversampling Method for Imbalanced Dataset Classification. *IEEE Access* **2020**, *9*, 16030–16040. [CrossRef]
32. Kaya, E.; Korkmaz, S.; Sahman, M.A.; Cinar, A.C. DEBOHID: A differential evolution based oversampling approach for highly imbalanced datasets. *Expert Syst. Appl.* **2021**, *169*, 114482. [CrossRef]
33. Fajardo, V.A.; Findlay, D.; Jaiswal, C.; Yin, X.; Houmanfar, R.; Xie, H.; Liang, J.; She, X.; Emerson, D. On oversampling imbalanced data with deep conditional generative models. *Expert Syst. Appl.* **2021**, *169*, 114463. [CrossRef]
34. Sadhukhan, P. Learning minority class prior to minority oversampling. In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; pp. 1–8.
35. Zhang, H.; Wang, Z. A normal distribution-based over-sampling approach to imbalanced data classification. In *International Conference on Advanced Data Mining and Applications*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 83–96.
36. Rashu, R.I.; Haq, N.; Rahman, R.M. Data mining approaches to predict final grade by overcoming class imbalance problem. In Proceedings of the 2014 17th International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, 22–23 December 2014; pp. 14–19.
37. Bennin, K.E.; Keung, J.; Phannachitta, P.; Monden, A.; Mensah, S. Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. *IEEE Trans. Softw. Eng.* **2017**, *44*, 534–550. [CrossRef]
38. Gong, L.; Jiang, S.; Jiang, L. Tackling class imbalance problem in software defect prediction through cluster-based over-sampling with filtering. *IEEE Access* **2019**, *7*, 145725–145737. [CrossRef]
39. Huda, S.; Liu, K.; Abdelrazek, M.; Ibrahim, A.; Alyahya, S.; Al-Dossari, H.; Ahmad, S. An ensemble oversampling model for class imbalance problem in software defect prediction. *IEEE Access* **2018**, *6*, 24184–24195. [CrossRef]
40. García, V.; Sánchez, J.S.; Martín-Félez, R.; Mollineda, R.A. Surrounding neighborhood-based SMOTE for learning from imbalanced data sets. *Prog. Artif. Intell.* **2012**, *1*, 347–362. [CrossRef]
41. Ma, L.; Fan, S. CURE-SMOTE algorithm and hybrid algorithm for feature selection and parameter optimization based on random forests. *BMC Bioinform.* **2017**, *18*, 169. [CrossRef]
42. Guo, S.; Liu, Y.; Chen, R.; Sun, X.; Wang, X. Improved SMOTE algorithm to deal with imbalanced activity classes in smart homes. *Neural Process. Lett.* **2019**, *50*, 1503–1526. [CrossRef]
43. Liang, X.; Jiang, A.; Li, T.; Xue, Y.; Wang, G. LR-SMOTE—An improved unbalanced data set oversampling based on K-means and SVM. *Knowl.-Based Syst.* **2020**, *196*, 105845. [CrossRef]
44. Chen, B.; Xia, S.; Chen, Z.; Wang, B.; Wang, G. RSMOTE: A self-adaptive robust SMOTE for imbalanced problems with label noise. *Inf. Sci.* **2021**, *553*, 397–428. [CrossRef]
45. de Carvalho, A.M.; Prati, R.C. DTO-SMOTE: Delaunay Tessellation Oversampling for Imbalanced Data Sets. *Information* **2020**, *11*, 557. [CrossRef]
46. Wei, J.; Huang, H.; Yao, L.; Hu, Y.; Fan, Q.; Huang, D. NI-MWMOTE: An improving noise-immunity majority weighted minority oversampling technique for imbalanced classification problems. *Expert Syst. Appl.* **2020**, *158*, 113504. [CrossRef]
47. Wang, X.; Yang, Y.; Chen, M.; Wang, Q.; Qin, Q.; Jiang, H.; Wang, H. AGNES-SMOTE: An Oversampling Algorithm Based on Hierarchical Clustering and Improved SMOTE. *Sci. Program.* **2020**, *2020*, 8837357. [CrossRef]
48. Hemalatha, P.; Amalanathan, G.M. FG-SMOTE: Fuzzy-based Gaussian synthetic minority oversampling with deep belief networks classifier for skewed class distribution. *Int. J. Intell. Comput. Cybern.* **2021**, *14*, 270–287. [CrossRef]
49. Mukherjee, M.; Khushi, M. SMOTE-ENC: A novel SMOTE-based method to generate synthetic data for nominal and continuous features. *Appl. Syst. Innov.* **2021**, *4*, 18. [CrossRef]
50. Bej, S.; Davtyan, N.; Wolfien, M.; Nassar, M.; Wolkenhauer, O. Loras: An oversampling approach for imbalanced datasets. *Mach. Learn.* **2021**, *110*, 279–301. [CrossRef]
51. Guan, H.; Zhang, Y.; Xian, M.; Cheng, H.; Tang, X. SMOTE-WENN: Solving class imbalance and small sample problems by oversampling and distance scaling. *Appl. Intell.* **2021**, *51*, 1394–1409. [CrossRef]
52. Zheng, S.; Mogusu, E.; Veeranki, S.P.; Quinn, M.; Cao, Y. The relationship between the mean, median, and mode with grouped data. *Commun. Stat.-Theory Methods* **2017**, *46*, 4285–4295. [CrossRef]
53. Hu, F.; Li, H.; Lou, H.; Dai, J. A parallel oversampling algorithm based on NRSBoundary-SMOTE. *J. Inf. Comput. Sci.* **2014**, *11*, 4655–4665. [CrossRef]
54. Woods, K.S.; Doss, C.C.; Bowyer, K.W.; Solka, J.L.; Priebe, C.E.; Kegelmeyer, W.P., Jr. Comparative evaluation of pattern recognition techniques for detection of microcalcifications in mammography. *Int. J. Pattern Recognit. Artif. Intell.* **1993**, *7*, 1417–1436. [CrossRef]
55. Alcalá-Fdez, J.; Fernández, A.; Luengo, J.; Derrac, J.; García, S.; Sánchez, L.; Herrera, F. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *J. Mult.-Valued Log. Soft Comput.* **2011**, *17*, 255–287.

56. Luengo, J.; Fernández, A.; García, S.; Herrera, F. Addressing data complexity for imbalanced data sets: Analysis of SMOTE-based oversampling and evolutionary undersampling. *Soft Comput.* **2011**, *15*, 1909–1936. [CrossRef]
57. Dua, D.; Graff, C. UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences. Available online: http://archive.ics.uci.edu/ml (accessed on 20 March 2022).
58. Bhagat, R.C.; Patil, S.S. Enhanced SMOTE algorithm for classification of imbalanced big-data using random forest. In Proceedings of the 2015 IEEE International Advance Computing Conference (IACC), Banglore, India, 12–13 June 2015; pp. 403–408.
59. Gutiérrez, P.D.; Lastra, M.; Benítez, J.M.; Herrera, F. SMOTE-GPU: Big data preprocessing on commodity hardware for imbalanced classification. *Prog. Artif. Intell.* **2017**, *6*, 347–354. [CrossRef]
60. Fernández, A.; del Río, S.; Chawla, N.V.; Herrera, F. An insight into imbalanced big data classification: Outcomes and challenges. *Complex Intell. Syst.* **2017**, *3*, 105–120. [CrossRef]
61. Leevy, J.L.; Khoshgoftaar, T.M.; Bauder, R.A.; Seliya, N. A survey on addressing high-class imbalance in big data. *J. Big Data* **2018**, *5*, 1–30. [CrossRef]
62. Basgall, M.J.; Hasperué, W.; Naiouf, M.; Fernández, A.; Herrera, F. SMOTE-BD: An Exact and Scalable Oversampling Method for Imbalanced Classification in Big Data. In *VI Jornadas de Cloud Computing & Big Data (JCC & BD) (La Plata, Argentina, 2018)*; Universidad Nacional de La Plata: La Plata, Argentina, 2018.