Proceedings of Machine Learning Research 183:38–52, 2022 Full Paper - LIDTA 2022

DistSMOGN: Distributed SMOGN for Imbalanced Regression Problems

Xin Yue Song xsong019@uottawa.ca
Nam Dao ndao104@uottawa.ca
Paula Branco pbranco@uottawa.ca
School of Electrical Engineering and computer Science, Faculty of Engineering
University of Ottawa, Ottawa, Ontario, Canada

Editor: Nuno Moniz, Paula Branco, Luís Torgo, Nathalie Japkowicz, Michał Woźniak and Shuo Wang.

Abstract

Imbalanced domains pose important challenges to learning systems and multiple resampling solutions have been put forward in the past two decades. More recently, it became clear that the imbalance problem arises in several other tasks including regression. Although several resampling solutions were proposed to tackle the imbalanced regression problem, with the emergence of big data this problem has become more difficult as these solutions become unfeasible due to the large volumes of data. In this paper, we propose the first distributed resampling solution for imbalanced regression that is applicable to large amounts of data. Our algorithm, DistSMOGN, is a resampling solution based on SMOGN that addresses simultaneously the imbalanced regression problem and the challenge of dealing with high volumes of data. We apply Scalable KMeans++ as way to obtain coherent cluster that maintain the spatial relationships between the rare cases. Then, we apply the well-known SMOGN method in each cluster to obtain the new synthetic examples. This method allows to generate high quality synthetic examples while dealing with the large volumes of data. Our solution is based on the MapReduce paradigm and we propose an efficient implementation on Apache Spark. The experimental evaluation carried out shows the advantages of DistSMOGN. All the code implementing DistSMOGN is freely available and can be downloaded at https://github.com/ndao1104/distributed-resampling.

Keywords: Imbalanced Regression, Big Data, Spark, MapReduce

1. Introduction

Imbalanced domains are a challenge predominant in many real-world application where the most relevant cases for the end-user are not frequent in the data. This problem has been studied for over two decades with a particular focus on classification tasks. Still, imbalanced domain problem spans multiple tasks including regression, data streams, or time series (Branco et al., 2016; Krawczyk, 2016).

Imbalanced regression started to be addressed more recently. A multitude of solutions have been proposed to alleviate the imbalanced regression problem with a special focus on resampling techniques. Resampling acts by changing the original data distribution of the training data to force the learning algorithm to focus on the most important cases which would otherwise be neglected. The popularity of resampling methods to tackle imbalanced domains can be associated to their easy implementation and efficiency and to the capability of using any user-preferred learning algorithm after the modification of the training data. Resampling techniques revolve around under-sampling, over-sampling or hybrid solutions that combine both under and over-sampling.

When dealing with binary class imbalance problem, the decision of which class is the most important one is straightforward: the least represented (minority) class is considered the class of interest. However, to obtain this information for a regression tasks is more difficult. Either the end-user is able to provide this, which is a time consuming and expensive task, or an automatic method can be applied. This automatic method has several assumptions regarding the end-user interests but is an effective way to estimate a the relevance (or importance) of the target variable values across the problem domain. Imbalanced regression entails further challenges in order to be addressed successfully. Big data is an additional relevant challenge that, together with imbalanced regression, poses serious issues.

In this paper, we provide the first resampling solution for imbalanced regression in a big data environment. We propose DistSMOGN, a distributed implementation of SMOGN to deal with imbalanced regression in an efficient way making it possible to deal with large volumes of data. Our solution is based on SMOGN (Branco et al., 2017), a SMOTE-based (Chawla et al., 2002) algorithm that combines both SmoteR (Torgo et al., 2013) and Introduction of Gaussian Noise techniques to generate new instances. SmoteR is the extension to regression tasks of SMOTE, a well-known resampling algorithm initially developed for classification tasks that generates new synthetic cases by interpolating one minority class case and one of its nearest neighbors. However, SMOTE-based algorithms depend on the calculation of nearest neighbors and are thus designed for standard-sized datasets and cannot be directly applied when it comes to Big Data. To overcome this issue, DistSMOGN re-implements the SMOGN algorithm in a distributed fashion in order to be suitable in a big data context while seeking to keep the spatial coherence of the rare cases.

Our main contributions are as follows: (i) a solution for imbalanced regression in big data environment: we propose DistSMOGN, the first resampling solution that is able to deal with the imbalanced regression problem while also being applicable in environments with large volumes of data implemented on Spark; (ii) keeping the rare cases spatial coherence: our proposed solution maintains the spatial relationships between the rare examples which ensures that high quality synthetic examples are generated; and (iii) code and data repository: we made all code and data used freely available to the research community to allow the reproducibility of our work and the usage of our solution.

This paper is organized as follows. Section 2 presents the related work associated with imbalanced regression and distributed resampling. In Section 3, details of DistSMOGN, our proposed distributed SMOGN implementation, are presented. Section 4 describes the experimental framework and dataset used to evaluate the performance of DistSMOGN algorithm and the results obtained. Finally, in Section 5, the main conclusions of our paper are provided.

2. Background and Related Work

This section provides an overview of the problem of imbalanced domains and reviews the main existing works to deal with this problem with a special focus on distributed solutions.

2.1. Learning from Imbalanced Domains

The problem of imbalanced domains is well-known and multiple solutions have been put forward to tackle it. Most of the existing solutions focus on the imbalanced classification problem. Among the most frequently used approaches are the resampling (or pre-processing) methods which act by changing the original dataset in order to make it easier for the learning algorithm to focus on the most rare and important cases. Resampling methods can be clustered into under-sampling, over-sampling or hybrid approaches. Under-sampling methods tackle the problem by removing cases from the majority (negative) class, while over-sampling methods do not discard any cases but instead add more examples form the minority (positive) class. Under-sampling and over-sampling methods can be random or more informed in the sense that they use instances characteristics to inform the removal/introduction of examples. Random under-sampling, Near Miss (Mani and Zhang, 2003), or the Condensed Nearest Neighbor (Hart, 1968) are examples of under-sampling approaches. Over-sampling approaches can rely on the addition of exact copies of examples present in the available data or the generation of new synthetic examples based on the real examples available. The first method is used, for instance, by the random over-sampling strategy, while the second method is used, for instance, by SMOTE (Chawla et al., 2002). SMOTE is one of the most well-known and used resampling approaches for the class imbalance problem. This approach is based on the interpolation of two minority class cases to generate a new synthetic minority class case. Since its development many alternative approaches were proposed to deal with some of the risks that SMOTE strategy entails.

The class imbalance is the most extensively studied predictive task among the possible tasks in the context of imbalanced domains. Regression tasks, data streams, time series are examples of other tasks for which the presence of imbalanced domains also poses significant challenges (Krawczyk, 2016). In this paper, we focus on regression problems, for which several solutions have been proposed recently. In order to tackle an imbalanced regression problem we first must understand which examples are the most important ones. This is straight forward in binary classification where the minority class is assumed to be the class of interest, i.e., the most relevant class for the end-user. In regression, we need to have a similar notion of important and unimportant cases. To address this problem, Torgo and Ribeiro (Torgo and Ribeiro, 2007) proposed the definition and use of a relevance function, $\phi()$, that expresses the importance assigned by the end-user to the range of the target variable of the problem being tackled. The relevance function helps in determining the rare (important) and normal (unimportant) cases by mapping the target variable into a scale of relevance between 0 and 1. However, it is difficult for the end-user to define the relevance function for a continuous target variable. Moreover, this information is should ideally be provided by domain experts, but in that case this can become a time consuming and expensive task. An automatic way for estimating the relevance function, was proposed to deal with this challenge (Ribeiro, 2011). This method allows us to obtain the relevance function information using the target variable density while assuming that low density regions in the extremes of the distribution will be the most interesting cases for the end-user.

Existing resampling approaches for imbalanced regression use the notion of the relevance function and a relevance threshold that is used to build the rare/important and normal/unimportant ranges. Multiple under-sampling and over-sampling solutions were proposed for imbalanced regression using this framework. In particular, and adaption of SMOTE named SmoteR (Torgo et al., 2013) was proposed as well as multiple alternatives. SMOGN (Branco et al., 2017) is an evolution of SmoteR that combines SmoteR with the introduction of Gaussian Noise. When the risk estimated for generating a new example through SmoteR is high, then a the more conservative alternative of generating the new case through the introduction of Gaussian Noise is used instead. More precisely, the choice of which method to use is determined by the distance between the base case and the neighbor case. If the neighbor case is close enough, SmoteR is used, otherwise the new case is generated by Gaussian Noise. Moreover, SMOGN also incorporates both over-sampling and under-sampling to resample the dataset.

In the context of data streams, some solutions were also proposed to tackle the imbalanced regression problem (e.g., Aminian et al. (2021)). Although dealing with imbalanced regression, these methods are not built in a distributed fashion.

2.2. Distributed Solutions for dealing with Imbalanced Domains

The study of distributed resampling techniques for tackling the imbalance in classification tasks is fairly recent. In a distributed context, we only found three works that address binary class imbalance problems and one work that tackles multi-class imbalanced problems. Moreover, as far as we know, no work exists that proposes a distributed resampling solution for imbalanced regression problems.

In 2018, Rastogi et al. (2018b) proposed the first distributed implementation of SMOTE for classification problems. The authors extend SMOTE to distributed environments under Spark using Locality Sensitivity Hashing (LSH) (Indyk and Motwani, 1998). LSH is used to identify the nearest neighbors of the minority class samples and then SMOTE is applied to generate minority class synthetic samples. LSH method allows to obtain equal size partitions of the space, providing a fast clustering solution. The intuition behind LSH is that two points that are close to each other will continue to be close after a projection operation. The main idea of the solution proposed by Rastogi et al. (2018b) is to use LSH to cluster the data points and then apply SMOTE to each generated cluster. Another solution for adapting SMOTE to a distributed setting was proposed by Rastogi et al. (2018a). The authors used distributed K-Means and M-Trees as the base methodology to cluster the data and search for the nearest neighbors inside each cluster. After having the clusters and nearest neighbors the standard SMOTE algorithm is applied. A similar solution was presented by Hooda and Mann (2019). Sleeman IV and Krawczyk (2021) proposed the first framework for dealing with multi-class imbalanced problems in a distributed way and presented an extensive set of experiments.

Still, we must highlight that, as far as we know, there is no such distributed solution for resampling strategies in a imbalanced regression context. The advent of big data as an important impact in these tasks frequently exacerbating the imbalance problem. Regression problems are heavily affected by these issues, thus it is necessary and important to explore and test solutions in this particular context.

3. DistSMOGN: Distributed SMOGN for Imbalanced Regression

SMOTE-based algorithms rely on the neighbourhood information when generating new samples. The computational complexity of searching nearest neighbor grows with the size and dimensionality of the dataset. The intuition behind our algorithm is generate clusters of data and to search for the nearest neighbors within each formed cluster so that parallel resampling is supported. Applying resampling techniques individually on randomly partitioned data may seem to be an option. However, this usually fails since randomly partitioned data are usually spatially disconnected. Our goal is to overcome this issue by partitioning the dataset effectively such that similar data are always available in the same node of the cluster which allows safe nearest neighbor search within each partition.

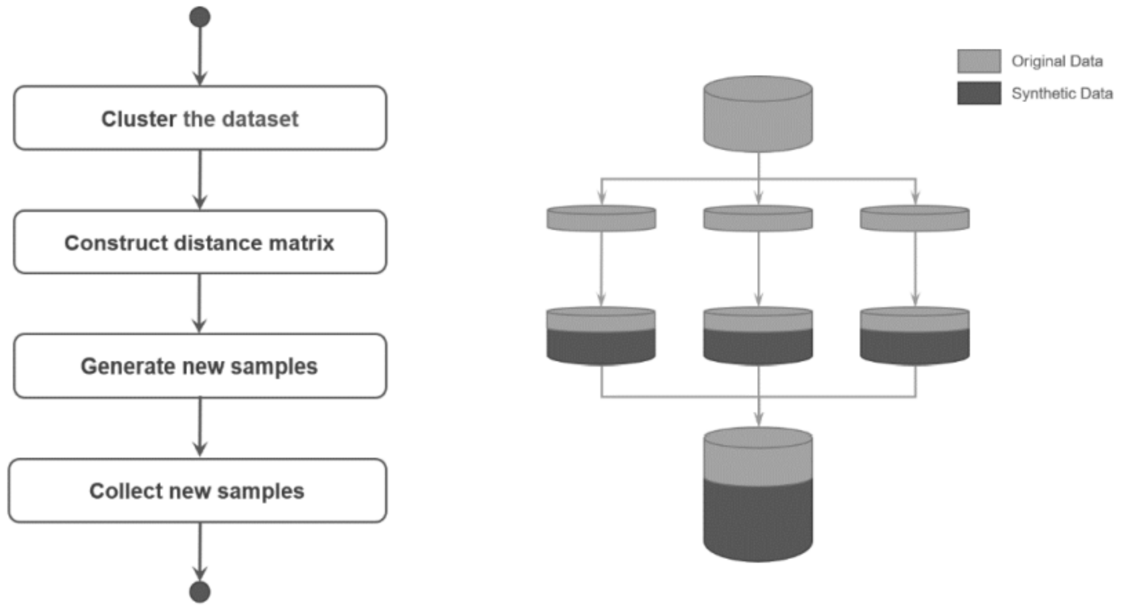

Figure 1: The DistSMOGN overall structure.

In this paper, we present DistSMOGN, a distributed implementation of SMOGN in Apache Spark. The general structure of the algorithm is shown in Figure 1. Overall, the algorithm includes four key steps: (i) clustering the dataset; (ii) building a distance matrix; (iii) generate new samples and/or undersampling the dataset cluster; and (iv) update the modified training dataset. We will discuss each step with more details next.

The pseudo code for the DistSMOGN Algorithm is provided in Algorithm 1. In any imbalanced regression task we need to obtain the relevance function, $\phi()$, and a use a relevance threshold to define the important and unimportant ranges of the target variable. We assume the end-user is able to provide this information or is able to use an automatic method to estimate it from the data distribution. More details on $\phi()$ and the automatic method to obtain it were provided in Section 2.1. Our first step is then to identify the rare and normal samples using $\phi()$ and the user-defined threshold of the relevance function. Then, we conduct distributed random under-sampling for normal samples and distributed over-sampling using SMOGN strategy for rare samples. The implementation of distributed random under-sampling is trivial as it only implies the parallel computation of a random sample of each bin containing normal cases (cf. Algorithm 1 lines 5-7). For this reason, we will only focus on the algorithm for distributed SMOGN Algorithm that applies over-sampling with SMOTER and introduction of Gaussian Noise.

We first cluster the dataset into different partitions using Scalable KMeans++ (Bahmani et al., 2012) to preserve the spatial relationship among data points within each partition (cf. lines 10-11 in Algorithm 1). We selected Scalable Kmeans++ as this solution overcomes important limitations of the KMeans clustering. The well-known KMeans clustering has two main steps: initialization and convergence. The original KMeans algorithm first initialize centroids randomly which affects the final results if the initial centroids are chosen incorrectly. The KMeans++ algorithm improves on KMean by sampling a single point in each pass to remove the dependence on centroid initialization. Scalable K-Means++ (Bahmani et al., 2012) takes it one step further by sampling multiple points in each pass and repeating the preprocess multiple times which allows parallel implementation of the KMeans algorithm. For this reason we selected this algorithm for our implementation.

As shown in Figure 1, our algorithm employs the MapReduce philosophy to achieve distributed processing. After creating the dataset partitions we enter the map phase. For each partition, we construct the distance matrix for nearest neighbor search (cf. line 14 in Algorithm 1) and for each sample in the partition, we generate new synthetic samples using SMOGN (cf. lines 17-27 in Algorithm 1). At the synthetic data generation stage, either SmoteR strategy or introduction of Gaussian Noise strategy is applied depending on the SMOGN safe distance requirement between the base case and its neighbor.

Finally, we enter the reduce phase by collecting all the newly generated samples in each partition to obtain the balanced dataset (cf. line 30 in Algorithm 1).

Our DistSMOGN solution tries to maintain the spatial relationship between the dataset cases in each partition built and relies on the MapReduce paradigm to effectively resample the dataset. This allows us to deal with large volumes of data in an efficient way while simultaneously maintaining the spatial coherence of the data instances in each partition built. For reproducibility purposes, the source code for DistSMOGN is freely available to the research community in the following link: `https://github.com/ndao1104/distributed-resampling`.

4. Experimental Evaluation

4.1. Experimental Settings

We selected seven datasets from different imbalanced domains to evaluate the effectiveness of our implementation. The selected datasets include two datasets that have a larger number of samples and features which we use to assess and compare the improvement on execution time between the original sequential implementation (Branco et al., 2017) and DistSMOGN, our distributed implementation of SMOGN. The main characteristics of the datasets used are shown in Table 1. All datasets were normalized. The relevance function is automatically obtained based on the label density of both low and high values and a relevance threshold of 0.8 is used to determine the rare and normal cases. The selected datasets have a total number of instances ranging from 506 to 21263. This will allow us to observe how the

Algorithm 1 DistSMOGN: Distributed SMOGN Algorithm

```
Input: $Data$: dataset;
       $y$: target variable
       $thresh$: relevance threshold
       $\%under$: percentage of undersampling
       $\%over$: percentage of oversampling
       $k\_partition$: number of partitions
       $k\_neigh$: number of nearest neighbors
       $pert$: perturbation
Output: $new\_data$: the resampled dataset
1  $y\_sorted \leftarrow sort(y)$
2  $\phi() \leftarrow$ relevance score obtained based on the $y$ distribution
3  $bins\_norm, bins\_rare \leftarrow$ normal and rare bins obtained using $\phi()$ and $thresh$
4  $new\_data \leftarrow \{\}$
5  for $bin \in bin\_norm$ do
6      $new\_samples \leftarrow$ randomly select $\%under \times |bin|$ samples from bin
7      $new\_data \leftarrow new\_data \bigcup new\_samples$
8  end
9  for $bin \in bin\_rare$ do
10     $centroids \leftarrow KMeans(bin, k\_partition)$ ;    // build partitions of bins with rare cases
11     $partitions \leftarrow$ partitions of bin obtained based on $centroids$
12     $n \leftarrow \%over \times |bin|$
13     for $partition \in partitions$ do
           // apply SMOGN in each obtained data partition
14         $dist\_mat \leftarrow$ pairwise distance matrix between all samples in $partition$
15         $new\_samples \leftarrow partition$
16         for $sample \in partition$ do
17             $neighbours \leftarrow KNN(k\_neigh, sample, dist\_mat)$
18             $safe\_dist \leftarrow median(dist_mat(samples, neighbours))/2$ ;  // SMOGN safe distance
19             for $i \leftarrow 1$ to $n$ do
20                 $sel\_neigh \leftarrow$ randomly select one neighbour from $neighbours$
                   // apply SmoteR or GN
21                 if $dist(sample, sel\_neigh) < safe\_dist$ then
22                     $new\_sample \leftarrow SmoteR(sample, sel\_neigh)$
23                 else
24                     $new\_sample \leftarrow GN(sample, sel\_neigh)$
25                 end
26                 $new\_samples \leftarrow new\_samples \bigcup new\_sample$
27             end
28         end
29     end
30     $new\_data \leftarrow new\_data \bigcup new\_samples$
31 end
32 return $new\_data$
```

dimension of the datasets impacts the performance and the time to carry out the resampling. Moreover, we are also considering datasets with imbalance ratios varying between 4.56 and 14.6, which will allow to analyze the impact of DistSMOGN in difference IR settings.

Dataset	#Instances	#Attr	#Cat Attr	#Num Attr	Normal:Rare	IR
Boston	506	13	0	13	415:91	4.560
Abalone	4117	8	1	7	3498:679	5.152
Bank8FM	4499	8	0	8	4211:288	14.622
heat	7400	11	3	8	6736:664	10.145
cpuSM	8192	12	0	12	7479:713	10.489
energy	19735	27	0	27	17070:2665	6.405
superconductivity	21263	81	0	81	19726:1537	12.834

Table 1: Main characteristics of the used datasets (#Attr: No. of attributes; #Cat Attr: No. categorical attributes; #Num Attr: No. numeric attributes).

The infrastructure used for the experiments was a Databrick cluster with one driver node and two worker nodes. The driver node has six cores and 16 GB of memory. Each worker node has two cores and 8 GB of memory. The cluster was configured with Apache Spark 3.2.1.

We tested DistSMOGN using different number of partitions (k = 2, k = 4 and k = 8). We also included in our tests the initial sequential versions of SMOGN, Random Undersampling (RUS) and Random Over-sampling (ROS). We included RUS and ROS solutions as they allow to modify the original dataset in a simple and fast way.

With the setting described we carried out two key experiments that evaluated: (i) the impact of the different resampling solutions on the execution time; and (ii) the impact in the performance of the different resampling solutions. For the first experiment we simply evaluated the time required to modify each dataset using the different resampling techniques.

For the second experiment, we used four learning algorithms to evaluate the performance of the four resampling techniques (RUS, ROS, SMOGN and DistSMOGN) and the use of the original (unchanged) dataset. The experiments were conducted in the Python environment using implementations of regression algorithms from the scikit-learn package. The four regression algorithms selected are: Linear Regression (LR), Support Vector Machine (SVM), Random Forest (RF) and Neural Network (NN). For each selected algorithm, we tested six parameter variants (except for Linear Regression). Table 2 shows the learning algorithms and corresponding parameter variants. Overall we carried out tests with 19 learners variants.

Each dataset is splitted into two sets of data: the training set (80%) and the test set (20%) and for each training set, we created 6 balanced training sets using the following resampling methods and variants: RUS, ROS, SMOGN, DistSMOGN with k = 2, 4 and 8. We also tested the performance of the learners when using the original imbalanced training set. Overall, we carried out a total of 931 tests ($7 \times 7 \times 19 = 931$) using 7 datasets, 7 resampling variants and 19 learner variants.

For the performance evaluation we used a variant of the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) metrics with the relevance values of the target variable

Learning Algorithms	Parameters
Linear Regression (LR)	No parameter
Support Vector Machine (SVM)	C = {10, 150, 300}; gamma = {0.01, 0.001}
Random Forest (RF)	min_samples_leaf = {1, 2, 4}; min_samples_split={2, 5}
Neural Network (NN)	hidden_layer_sizes = {1, 5, 10}; max_iter = {500, 1000}

Table 2: Used learning algorithms and respective parameters.

being used as the sample weights. We refer to these metrics as $MAE\phi$ and $RMSE\phi$ which are defined in Equation 1 and Equation 2, respectively. The intuition behind using the relevance values as weights for the different errors is to penalize more heavily errors that occur in the most relevant regions of the target variable.

$$MAE\phi = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\phi(y_i) \times |y_i - \hat{y}_i|} \tag{1}$$

$$RMSE\phi = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\phi(y_i) \times (y_i - \hat{y}_i)^2} \tag{2}$$

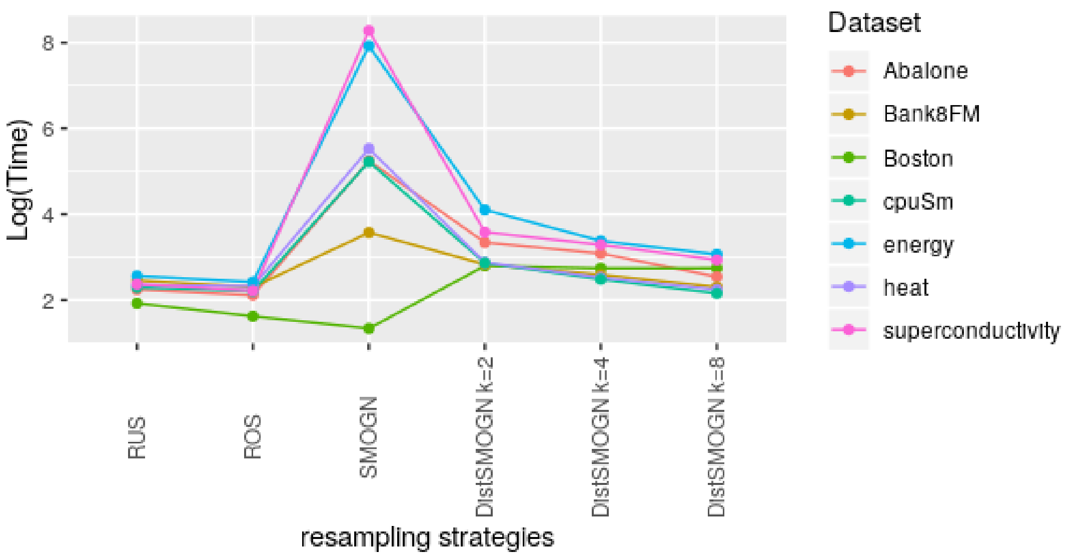

Figure 2: Execution time (measured in seconds) in a logarithmic scale of RUS, ROS, SMOGN and DistSMOGN with different number of partitions (k=2, 4, 8) for each dataset. (Bold: lowest execution time; underlined: highest execution time)

	RUS	ROS	SMOGN	DistSMOGN		
				k = 2	k = 4	k = 8
Boston	6.85	5.077	**3.817**	16.439	15.526	15.563
Abalone	9.456	**8.276**	189.563	28.336	21.896	12.725
Bank8FM	11.673	**9.983**	35.68	16.951	13.32	10.012
heat	10.458	10.209	251.871	17.738	12.48	**9.437**
cpuSm	9.94	9.105	185.718	17.556	11.985	**8.643**
energy	12.983	**11.276**	2758.835	60.497	29.373	21.45
superconductivity	10.705	**9.187**	3951.75	35.988	26.844	18.789

Table 3: Execution time (measured in seconds) of RUS, ROS, SMOGN and DistSMOGN with different number of partitions for each dataset. (k: the number of partitions.)

	LR		SVM		RF		NN	
	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$
No Sampling	4.849	7.003	6.329	9.034	2.476	4.138	8.483	10.818
RUS	4.394	6.189	8.128	11.235	3.444	6.075	15.94	16.206
ROS	4.313	**5.947**	5.425	8.505	2.702	3.499	**4.474**	**6.353**
SMOGN	**4.246**	6.221	5.921	**7.219**	**2.332**	3.801	5.615	7.766
DistSMOGN (k = 2)	4.459	6.214	5.265	7.618	2.638	**3.59**	5.849	7.682
DistSMOGN (k = 4)	4.453	6.295	5.25	7.781	2.581	3.876	5.206	7.056
DistSMOGN (k = 8)	4.297	6.134	**5.174**	7.494	2.9	3.658	5.981	7.968

Table 4: $MAE\phi$ and $RMSE\phi$ of the best model variant for each regression algorithm with different resampling strategies for the Boston dataset. (Best results for each regression algorithm and resampling strategy are highlighted.)

4.2. Results and Discussion

4.2.1. Execution Time

To evaluate the impact on the execution time, we compared the resampling time taken using our distributed SMOGN (DistSMOGN) implementation with different number of partitions (partitions = 2, 4 or 8) against the sequential implementations of SMOGN, RUS and ROS.

Table 3 summarizes the execution time obtained for each dataset and Figure 2 displays the resampling time in a logarithmic scale. We decided to visualize of the results in a transformed scale due to the extremely large values observed for some datasets in some resampling strategies, namely the SMOGN strategy for the energy and superconductivity datasets, which would skew the graph.

We observe that globally the sequential version of SMOGN is the most time consuming approach. Only for the smaller dataset (Boston) SMOGN does not provide the longer execution time. RUS and ROS exhibit very low execution times as expected even for the larger datasets. In fact, these resampling strategies are very efficient and easy to apply.

	LR		SVM		RF		NN	
	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$
No Sampling	2.331	3.128	2.851	3.895	2.194	3.061	2.405	3.302
RUS	2.123	2.808	2.45	3.339	2.072	2.803	3.817	4.684
ROS	2.08	**2.761**	**2.173**	**2.976**	2.145	2.978	**2.022**	**2.719**
SMOGN	2.235	2.821	2.516	3.316	2.123	2.881	3.069	3.863
DistSMOGN (k = 2)	2.097	2.78	2.223	3.062	1.981	2.805	2.057	2.786
DistSMOGN (k = 4)	**2.078**	2.776	2.223	3.062	2.004	2.81	2.878	3.676
DistSMOGN (k = 8)	2.087	2.793	2.236	3.075	**1.975**	**2.796**	2.703	3.554

Table 5: $MAE\phi$ and $RMSE\phi$ of the best model variant for each regression algorithm with different resampling strategies for the Abalone dataset. (Best results for each regression algorithm and resampling strategy are highlighted.)

When comparing SMOGN and the DistSMOGN variants we observe that DistSMOGN is always faster irrespectively of the number of partitions considered, except for the smaller dataset used in our experiments.

4.2.2. Performance

Regarding the experiments for measuring the predictive performance of our proposed DistSMOGN algorithm, Tables 4, 5, 6, 7, 8, 9, and 10 show the aggregated benchmarking results for each dataset considered.

Overall we observe that DistSMOGN presents an overall competitive performance. The clustering applied does not seem to be affecting the quality of the newly generated instances. We conclude this due to the number of times DistSMOGN exhibits the best overall performance but also becasue when comparing DistSMOGN performance against SMOGN they do not differ much for both performance metrics considered. We also noticed that for some datasets, such as Abalone, there is another strategy that performs better (ROS) than SMOGN. In these cases, we observe that DistSMOGN is able to beat that resampling strategy for some learners. There are also datasets for which SMOGN is overlla the best performing strategy, such as Bank8FM. Still, we must highlight that in these cases DistSMOGN performance is only slightly worst that SMOGN but DistSMOGN results are much faster to compute.

Our results show that DistSMOGN is an effective strategy for tackling the imbalanced regression problem in a distributed fashion making it fats and easy to resample the training data while achieving a good performance. DistSMOGN is able to preserve the spatial relationships between the rare examples allowing the generation of high quality synthetic cases for the rare ranges of the target variable.

5. Conclusion

This paper presented DistSMOGN, the first distributed resampling method for tackling imbalanced regression problems. DistSMOGN uses scalable KMeans++ to partition the dataset and then applies SMOGN to each cluster formed. Our Spark-based algorithm relies

	LR		SVM		RF		NN	
	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$
No Sampling	0.046	0.058	0.049	0.06	0.038	0.046	0.056	0.068
RUS	**0.044**	0.056	0.046	0.057	0.042	0.053	0.054	0.066
ROS	0.045	0.057	**0.041**	**0.051**	0.041	0.052	**0.049**	0.057
SMOGN	**0.044**	**0.055**	0.045	0.055	**0.037**	**0.045**	**0.049**	**0.055**
DistSMOGN (k = 2)	0.045	0.056	**0.041**	**0.051**	0.039	0.049	0.051	0.066
DistSMOGN (k = 4)	0.045	0.057	0.042	0.052	0.039	0.048	0.051	0.067
DistSMOGN (k = 8)	0.045	0.057	0.042	0.052	0.042	0.052	0.051	0.067

Table 6: $MAE\phi$ and $RMSE\phi$ of the best model variant for each regression algorithm with different resampling strategies for the Bank8FM dataset. (Best results for each regression algorithm and resampling strategy are highlighted.)

	LR		SVM		RF		NN	
	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$
No Sampling	16.583	23.391	25.999	33.306	**2.681**	3.463	10.112	14.843
RUS	12.221	15.902	21.225	27.93	4.72	6.121	14.599	20.596
ROS	12.508	16.223	**9.426**	**12.969**	2.687	**3.443**	9.585	**14.055**
SMOGN	**12.014**	16.168	13.926	20.159	3.142	4.135	9.54	14.581
DistSMOGN (k = 2)	16.649	21.173	13.553	18.607	4.405	5.89	9.179	14.82
DistSMOGN (k = 4)	12.124	**15.755**	13.201	17.974	4.222	5.821	9.6	14.315
DistSMOGN (k = 8)	12.744	16.241	12.951	17.621	4.503	5.948	**9.144**	14.89

Table 7: $MAE\phi$ and $RMSE\phi$ of the best model variant for each regression algorithm with different resampling strategies for the heat dataset. (Best results for each regression algorithm and resampling strategy are highlighted.)

	LR		SVM		RF		NN	
	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$
No Sampling	11.002	17.245	4.937	6.945	2.833	4.264	**9.247**	**14.627**
RUS	10.064	**11.974**	4.74	6.471	3.142	4.52	36.196	41.295
ROS	10.028	12.019	**3.494**	**4.926**	**2.827**	**4.175**	15.524	19.405
SMOGN	**9.766**	12.019	3.896	5.351	3.263	5.073	42.675	46.611
DistSMOGN (k = 2)	10.119	12.034	3.732	5.178	2.868	4.274	39.584	43.134
DistSMOGN (k = 4)	10.104	12.043	3.801	5.325	2.963	4.324	23.281	24.696
DistSMOGN (k = 8)	10.081	11.999	3.818	5.352	2.868	4.207	41.823	44.921

Table 8: $MAE\phi$ and $RMSE\phi$ of the best model variant for each regression algorithm with different resampling strategies for the cpuSm dataset. (Best results for each regression algorithm and resampling strategy are highlighted.)

	LR		SVM		RF		NN	
	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$
No Sampling	**130.37**	192.78	**122.279**	189.287	92.343	143.363	**130.142**	192.642
RUS	145.583	176.061	152.833	176.316	145.808	176.074	136.94	165.441
ROS	143.783	174.478	123.782	164.308	91.84	**130.273**	143.337	173.009
SMOGN	133.621	174.571	123.249	**163.144**	91.064	133.345	144.917	173.577
DistSMOGN (k = 2)	133.291	174.079	124.528	163.684	92.329	135.685	142.774	172.28
DistSMOGN (k = 4)	133.158	**173.949**	123.843	163.388	**90.962**	135.697	142.416	171.75
DistSMOGN (k = 8)	134.362	174.509	124.882	164.229	92.786	136.964	142.433	**171.637**

Table 9: $MAE\phi$ and $RMSE\phi$ of the best model variant for each regression algorithm with different resampling strategies for the energy dataset. (Best results for each regression algorithm and resampling strategy are highlighted.)

	LR		SVM		RF		NN	
	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$	$MAE\phi$	$RMSE\phi$
No Sampling	**12.022**	**16.62**	12.29	17.704	**5.734**	**11.222**	**9.793**	**15.65**
RUS	29.775	44.719	44.11	47.595	17.878	23.146	27.161	33.384
ROS	14.348	19.526	**12.174**	**17.673**	5.786	11.423	10.84	17.203
SMOGN	14.275	19.305	13.727	18.927	7.201	13.281	11.836	17.299
DistSMOGN (k = 2)	14.224	19.372	13.403	18.652	7.281	13.997	11.09	16.484
DistSMOGN (k = 4)	13.871	18.873	13.085	18.391	7.046	13.365	11.223	16.24
DistSMOGN (k = 8)	14.491	19.631	13.236	18.532	7.025	13.788	11.465	17.064

Table 10: $MAE\phi$ and $RMSE\phi$ of the best model variant for each regression algorithm with different resampling strategies for the superconductivity dataset. (Best results for each regression algorithm and resampling strategy are highlighted.)

on the MapReduce paradigm and is significantly faster than the original sequential implementation when it comes dataset with a large number of samples. Moreover, DistSMOGN is able to achieve better or comparable results when considering SMOGN strategy in most scenarios. This happens because DistSMOGN allows to effectively distribute the samples across multiple machines while preserving the spatial relationships between the examples which allows the fast generation of high quality synthetic cases. The execution time of DistSMOGN as well as its ability to keep the spatial relationships between rare examples are two of its key advantages.

The time complexity of our algorithm is still currently constrained by the KNN algorithm that we used which is based on a brute-force implementation for exact KNN search. We believe that exploring the embedding of approximate KNN solutions is a promising future research direction that may provide more interesting results. Furthermore, we are also considering embedding in our algorithm an automatic method for determining the best number of partitions to use for the given dataset. This could make the end-user task easier. We are planning to carry out more experiments using multiple runs for each dataset to allow to observe the variability of the results and to overcome the potential bias of only using one train/test split. Finally, we consider that extending the experiments to more and larger datasets and extending the DistSMOGN algorithm to other resampling techniques are interesting research avenues.

References

Ehsan Aminian, Rita P Ribeiro, and João Gama. Chebyshev approaches for imbalanced data streams regression models. *Data Mining and Knowledge Discovery*, 35(6):2389–2466, 2021.

Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *arXiv preprint arXiv:1203.6402*, 2012.

Paula Branco, Luís Torgo, and Rita P Ribeiro. A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys (CSUR)*, 49(2):1–50, 2016.

Paula Branco, Luís Torgo, and Rita P Ribeiro. Smogn: a pre-processing approach for imbalanced regression. In *First international workshop on learning with imbalanced domains: Theory and applications*, pages 36–50. PMLR, 2017.

Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16: 321–357, 2002.

Peter Hart. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3):515–516, 1968.

Sakshi Hooda and Suman Mann. Distributed synthetic minority oversampling technique. *Int. J. Comput. Intell. Syst.*, 12(2):929–936, 2019.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

Bartosz Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232, 2016.

Inderjeet Mani and I Zhang. knn approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of workshop on learning from imbalanced datasets*, volume 126, pages 1–7. ICML, 2003.

Avnish Kumar Rastogi, Nitin Narang, and Mohammad Ajmal. Distributed synthetic minority oversampling technique. In *International Workshop on Algorithms and Architectures for Distributed Data Analytics*, 2018a.

Avnish Kumar Rastogi, Nitin Narang, and Zamir Ahmad Siddiqui. Imbalanced big data classification: a distributed implementation of smote. In *Proceedings of the workshop program of the 19th international conference on distributed computing and networking*, pages 1–6, 2018b.

Rita P. Ribeiro. Utility-based regression. PhD thesis, Dep. Computer Science, Faculty of Sciences - University of Porto, 2011.

William C Sleeman IV and Bartosz Krawczyk. Multi-class imbalanced big data classification on spark. *Knowledge-Based Systems*, 212:106598, 2021.

Luis Torgo and Rita Ribeiro. Utility-based regression. In *European conference on principles of data mining and knowledge discovery*, pages 597–604. Springer, 2007.

Luís Torgo, Rita P Ribeiro, Bernhard Pfahringer, and Paula Branco. Smote for regression. In *Portuguese conference on artificial intelligence*, pages 378–389. Springer, 2013.