Neurocomputing 464 (2021) 432–437

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Approx-SMOTE: Fast SMOTE for Big Data on Apache Spark

Mario Juez-Gil *, Álvar Arnaiz-González, Juan J. Rodríguez, Carlos López-Nozal, César García-Osorio

Departamento de Ingeniería Informática, Escuela Politécnica Superior, Universidad de Burgos, 09006 Burgos, Spain

ARTICLE INFO

Article history:
Received 2 March 2021
Revised 3 August 2021
Accepted 20 August 2021
Available online 24 August 2021
Communicated by Zidong Wang

Keywords:
SMOTE
Imbalance
Spark
Big data
Data mining

ABSTRACT

One of the main goals of Big Data research, is to find new data mining methods that are able to process large amounts of data in acceptable times. In Big Data classification, as in traditional classification, class imbalance is a common problem that must be addressed, in the case of Big Data also looking for a solution that can be applied in an acceptable execution time. In this paper we present Approx-SMOTE, a parallel implementation of the SMOTE algorithm for the Apache Spark framework. The key difference with the original SMOTE, besides parallelism, is that it uses an approximated version of *k*-Nearest Neighbor which makes it highly scalable. Although an implementation of SMOTE for Big Data already exists (SMOTE-BD), it uses an exact Nearest Neighbor search, which does not make it entirely scalable. Approx-SMOTE on the other hand is able to achieve up to 30 times faster run times without sacrificing the improved classification performance offered by the original SMOTE.

1. Introduction

In the era of Big Data, frameworks like Apache Spark [1] are growing in popularity and receiving a lot of attention. One of the strengths of this kind of open source projects is that they allow contributions from third party developers. Through Spark MLlib [2], this framework provides a machine learning library that contains a wide variety of algorithms for different tasks. Classification, regression, clustering, data extraction, data transformation, and data selection are some of them. Although powerful, its functionality is still limited compared to other machine learning frameworks like Scikit-Learn [3], and thus, many algorithms should be adapted and included into the Spark ecosystem. This is the case of the well known Synthetic Minority Over-sampling TEchnique (i.e., SMOTE) [4], which is based on *k*-Nearest Neighbor (i.e., *k*-NN) algorithm. This paper presents the Approx-SMOTE algorithm for Apache Spark, which provides a SMOTE implementation based on an approximated *k*-NN approach that uses hybrid spill trees [5] for achieving accurate and efficient distributed nearest neighbor search.

2. Problems and background

In datasets for classification, it is very common for the number of instances in the different classes to be very different from one another. This means that, for a binary classification problem, the majority of the examples belong to one class and only a minority to the other one. Furthermore, despite the fact that the minority class is usually the class of interest, having so few instances makes the classifiers to be biased in favor of the majority class, causing the instances of the minority class to be misclassified. This is known as imbalanced learning problem, and although it has been widely studied in the past with normal-sized datasets [6–8], it is still in an early research stage within Big Data scenarios [9].

One simple yet effective strategy to deal with imbalance, is to resample the datasets to obtain others in which the number of instances of each class are equal, or at least more similar. This eliminates the bias toward the majority class of the classifiers that are constructed, and favors the correct classification of the instances on the minority class.

One of the most popular resampling methods is SMOTE [4], which generates new synthetic examples in the neighborhood of small groups of nearby instances. To find those neighborhoods, *k*-NN is used. The main downside of *k*-NN is that it is computationally intense because for each instance in the data set, it calculates the distance with the rest of instances in order to find the *k* nearest ones. Thus, it has a complexity quadratic in the number of instances, $\mathcal{O}(n^2)$. Hence, when the number of instances of the dataset is huge, *k*-NN computa-

* Corresponding author.
E-mail addresses: mariojg@ubu.es (M. Juez-Gil), alvarag@ubu.es (Á. Arnaiz-González), jjrodriguez@ubu.es (J.J. Rodríguez), clopezno@ubu.es (C. López-Nozal), cgosorio@ubu.es (

https://doi.org/10.1016/j.neucom.2021.08.086
0925-2312/

tion becomes infeasible [10]. Fortunately, there are a few approaches to efficiently approximate the nearest neighbor search by taking advantage of parallel and distributed computing [5,11]. Specifically, hybrid spill trees, which is the approach used by spark-knn package[1] implemented by Forest Fang (saurfang on GitHub[2]), split the space in such a way that the instances that fall into the same leaf of the tree, are considered neighbors.

There exists an adaptation of SMOTE for Big Data, SMOTE-BD [12] implemented and published in the SparkPackages repository.[3] Unfortunately, it uses an iterative implementation of exact *k*-NN [13], which limits its applicability to solve real Big Data classification problems, as it requires a lot of computing power. As noted in [14], scalability is one of the issues present in large-scale parallel systems, so special attention must be paid to scalability when developing Big Data algorithms. The solution we are presenting, Approx-SMOTE, greatly reduces the required computing power by using an approximate nearest neighbor search approach which also offers high scalability. It should be highlighted that using an approximation to the nearest neighbors does not seem to negatively affect the final results.

3. Software framework

The Approx-SMOTE package and its documentation are publicly available at GitHub.[4] It is also published on SparkPackages repository,[5] so can be easily installed as a dependency using Maven or sbt.

3.1. Software architecture

Approx-SMOTE is built as an Apache Spark MLlib package. It has no dependencies since Saurfang's approximated *k*-NN[6] is bundled. Following the naming conventions used in other data mining frameworks, such as Weka, this implementation is provided inside a new package called `instance` in a class named `ASMOTE`, which inherits the Spark ML `Transformer`[7] class. The package `knn` contains Saurfang's implementation of the approximated *k*-NN used for synthesizing new instances.

3.2. Software functionalities

The Approx-SMOTE functionality consists in synthesizing new examples belonging to the minority class from an imbalanced binary classification dataset. New examples, along with the original examples, result in an oversampled dataset which, a priori, should contribute to the training of less biased classifiers towards the majority class. Approx-SMOTE, as an oversampling method, has a parameter (`percOver`) for defining the number of synthetic examples belonging to the minority class to be created. That parameter is a percentage of the number of minority class instances, and its default value is 100 (*i.e.*, the number of minority instances is doubled). As *k*-NN is used for synthesizing new examples, all the parameters of Saurfang's approximated *k*-NN can be adjusted, such as the number of nearest neighbors (`k`), the maximum distance between two neighbors (`maxDistance`), or the size of the top tree (`topTreeSize`) among others. The parameters meaning and functioning is explained in detail in [11]. The information about all the parameters default values is available at Saurfang's `spark-knn` GitHub repository.

[1] https://spark-packages.org/package/saurfang/spark-knn
[2] https://github.com/saurfang
[3] https://spark-packages.org/package/majobasgall/smote-bd
[4] https://github.com/mjuez/approx-smote
[5] https://spark-packages.org/package/mjuez/approx-smote
[6] https://github.com/saurfang/spark-knn
[7] https://spark.apache.org/docs/latest/api/scala/org/apache/spark/ml/Transformer.html

4. Implementation and empirical results

Approx-SMOTE is implemented in Scala 2.12 for Apache Spark 3.0.1 following the Apache Spark MLlib guidelines. A thorough validation of the algorithm was performed using cloud-based clusters. The objective of the experiments was to prove that Approx-SMOTE oversamples data equivalently as it does SMOTE-BD, but in a faster and more scalable way. We consider two oversampled datasets to be equivalent, if they both affect the classification performance of a classifier equally. In our experiments, a Spark ML Random Forest classifier with 100 trees and default parameters, was used. The characteristics of the six datasets used for all classification performance comparisons are described in Table 1.

4.1. Experimental framework

The experiments were launched on Google Cloud clusters composed of one master node (8 vCPU and 52 GB memory) and five different worker nodes configurations: 2, 4, 6, 8, and 10. Each worker node had 2 vCPU and 7.5 GB memory. Thus, the biggest cluster (1 master and 10 workers) consisted in 28 vCPUs and 127 GB memory. For comparing execution times all cluster configurations have been used. The classification performance comparison was executed on the biggest cluster, using 2-fold stratified cross-validation repeated 5 times.[8]

To ensure the repeatability of the experiments, a random seed was fixed to 46. The experiments consisted in reducing the imbalance ratio to 1 (i.e., balancing the dataset), thus, for each dataset, a specific `percOver` parameter was calculated. The number of neighbors (i.e., *k*) was fixed to 5. In the case of SMOTE-BD, the number of partitions was set to 8, as they recommended in [12]. All other parameters were kept as default.

To evaluate and compare statistical differences in performance, a Bayesian analysis was conducted using Bayesian hierarchical sign tests [17] (*baycomp*[9] library was used). The number of samples for all Bayesian comparisons was set to 50000. The graphical representation for this type of analysis, is a ternary plot [18] where the region of practical equivalence (*i.e.,* ROPE) appears on the top corner, and on the right and left corners are the regions of the methods under comparison. The ROPE was set to 0.01, which means that two algorithms with a difference in performance of less than 0.01 will be considered equivalent.

The performance metrics chosen were AUC and F_1-score, which are widely used in imbalanced learning [19].

4.2. Results and discussion

The comparative results about classification performance is shown in Table 2. The "No resampling" column shows how the Random Forest classifier performs trained with the original imbalanced dataset without any change (*i.e.,* without applying any kind of resampling). The blueness intensity of the cells depicts the results as a heatmap, the darker the blue, the better the result. The best result for each dataset and metric, is highlighted within a black box. According to the results, Approx-SMOTE achieves the best classification performance overall. However, as was to be expected, SMOTE-BD results were almost the same. The use of SMOTE, particularly for HIGGS IR4 and HIGGS IR16 datasets, benefited the classification performance. To check if Approx-SMOTE indeed performs better, bayesian statistical tests were performed and are shown in Fig. 1. The equivalence between SMOTE-BD and Approx-SMOTE is

[8] This cross-validation strategy is commonly used for imbalanced learning scenarios [16].
[9] The *baycomp* library is publicly available at https://baycomp.readthedocs.io/en/latest/.

Table 1
Main characteristics of the datasets used in the experiments: number of instances, number of features, number of classes of the minority and majority classes, imbalance ratio, and dataset size in libsvm [15] format.

Dataset	# Instances	# Attributes	# maj/min	IR	Size (GB)
SUSY IR4	3389320	18	2712173/677147	4.00	1.23
SUSY IR16	2881796	18	2712173/169623	15.99	1.04
HIGGS IR4	7284166	28	5829123/1455043	4.00	3.94
HIGGS IR16	6194093	28	5829123/364970	15.97	3.26
HEPMASS IR4	6561364	28	5250124/1311240	4.00	3.77
HEPMASS IR16	5578586	28	5250124/328462	15.98	3.20

Table 2
Classification performance on Random Forest with 100 trees. The best results appear within black boxes. The higher the blueness intensity, the better the performance. The value at the right of the ± sign, refers to the standard deviation between cross-validation folds.

Dataset	AUC			F_1-score		
	No resampling	SMOTE-BD	Appr-SMOTE	No resampling	SMOTE-BD	Appr-SMOTE
SUSY IR4	0.691 ± 0.002	0.771 ± 0.001	**0.772 ± 0.001**	0.542 ± 0.003	0.583 ± 0.003	**0.589 ± 0.003**
SUSY IR16	0.622 ± 0.009	0.771 ± 0.001	**0.772 ± 0.001**	**0.380 ± 0.019**	0.293 ± 0.003	0.300 ± 0.004
HIGGS IR4	0.507 ± 0.002	0.671 ± 0.002	**0.678 ± 0.002**	0.028 ± 0.008	0.452 ± 0.002	**0.459 ± 0.002**
HIGGS IR16	0.500 ± 0.000	0.660 ± 0.001	**0.672 ± 0.002**	0.000 ± 0.000	0.197 ± 0.001	**0.202 ± 0.001**
HEPMASS IR4	0.748 ± 0.004	**0.821 ± 0.001**	**0.821 ± 0.001**	**0.655 ± 0.007**	0.633 ± 0.003	0.630 ± 0.002
HEPMASS IR16	0.591 ± 0.014	**0.822 ± 0.001**	**0.822 ± 0.001**	0.306 ± 0.040	**0.338 ± 0.004**	0.329 ± 0.005

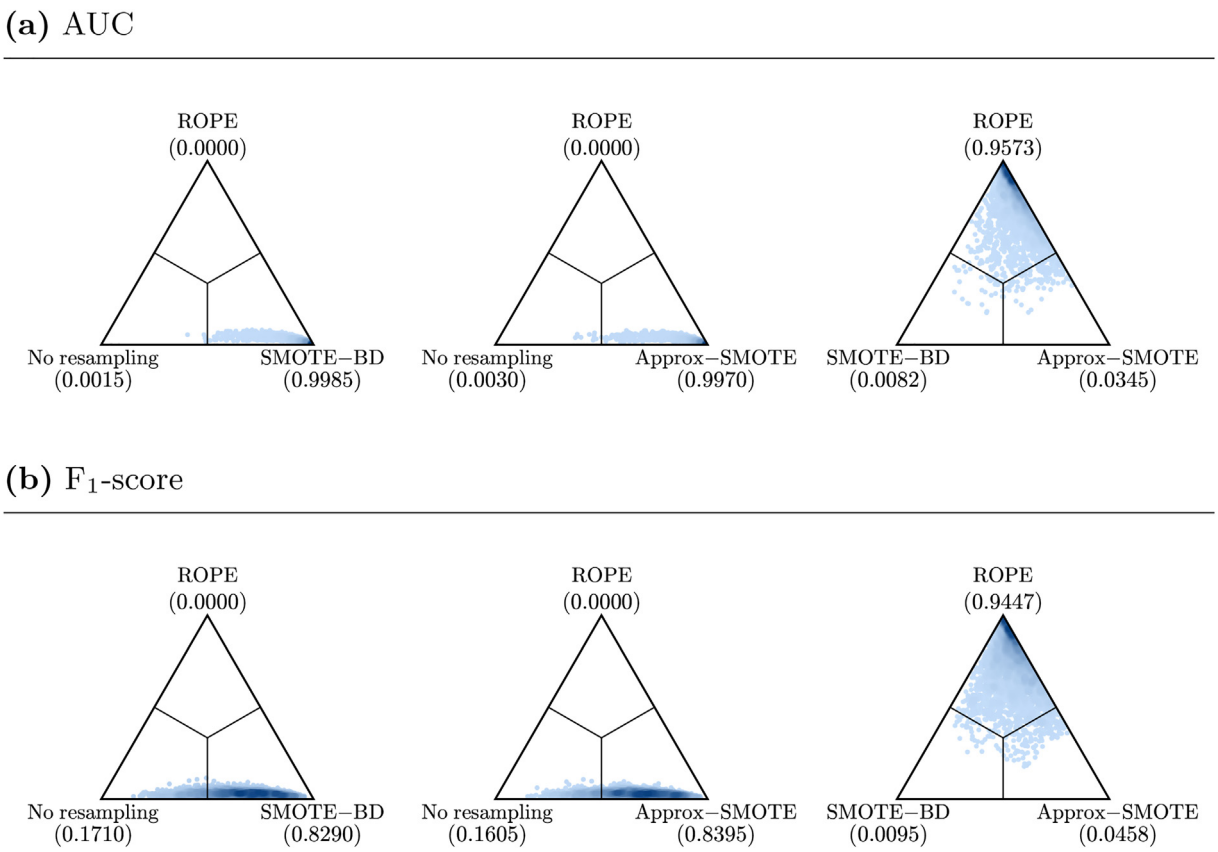


Fig. 1. Bayesian hierarchical sign tests comparing No resampling vs. SMOTE-BD (left column), No resampling vs. Approx-SMOTE (central column), and SMOTE-BD vs. Approx-SMOTE (right column). According to AUC (a) and F_1-score (b) metrics.

demonstrated because both approaches obtained almost the same supremacy compared to no resampling, and the direct comparison between them, resulted in an extremely high ROPE probability. Therefore, although at a first glance Approx-SMOTE seemed to be better than SMOTE-BD according to the Table 2, actually, the performance of both algorithms is not statistically different.

Finally, regarding execution times for SUSY IR16 dataset, Approx-SMOTE demonstrated to be between 7.52 (on the smallest cluster) and 28.15 (on the biggest cluster) times faster than SMOTE-BD. Table 3 shows the execution times in seconds of both approaches on clusters with 2, 4, 6, 8, and 10 workers. Speedup, which was calculated using the smallest cluster configuration as

Table 3
Execution times (in seconds) of SMOTE-BD and Approx-SMOTE on balancing SUSY IR16 task. Different cluster configurations (2, 4, 6, 8, and 10 workers) were tested. The number of times that Approx-SMOTE was faster than SMOTE-BD is shown in parentheses.

Approach	2w	4w	6w	8w	10w
SMOTE-BD	1321.39	2218.60	2103.68	1587.29	2172.05
Approx-SMOTE	175.70 (7.52×)	123.70 (17.94×)	113.89 (18.47×)	91.30 (17.39×)	77.15 (28.15×)

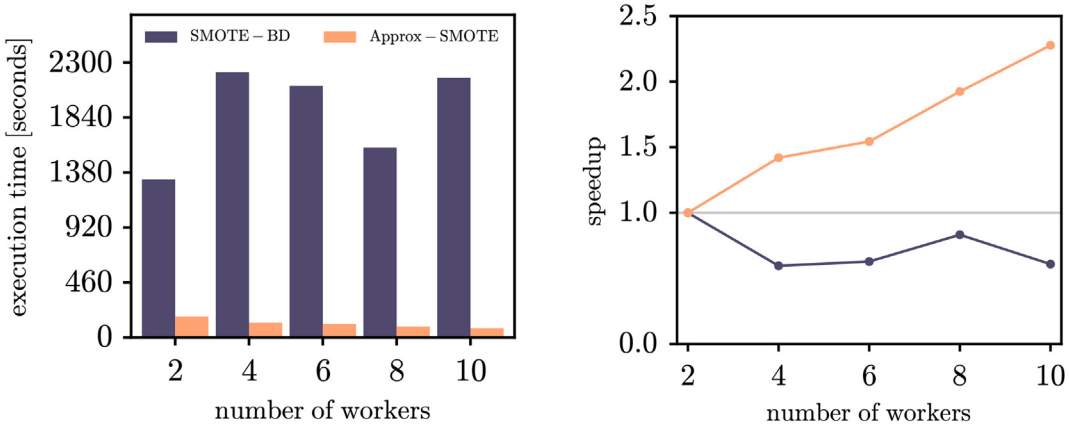


Fig. 2. Execution time (left) and speedup (right) of SMOTE-BD (purple) vs. Approx-SMOTE (orange) on balancing SUSY IR16 dataset task. Different cluster configurations (2, 4, 6, 8, and 10 workers) were tested.

baseline, revealed good scalability for Approx-SMOTE, and scalability issues for SMOTE-BD. Fig. 2 shows a graphical representation of the execution times and speedup comparisons where SMOTE-BD approach is depicted in purple, and Approx-SMOTE, in orange.

5. Illustrative examples

As Approx-SMOTE follows Spark-ML design guidelines, its use is similar to any other Transformer in the Spark API. Code 1 shows a basic example where a dataset is loaded and then oversampled doubling the minority class. The number of nearest neighbors for synthesizing new examples is fixed to 5.

6. Conclusions and future lines

In this paper, Approx-SMOTE, a novel approximated SMOTE adaptation for Big Data, is presented. It is implemented as an algorithm for Apache Spark framework, and as the original SMOTE does, it synthesizes new minority-class belonging examples for contributing to alleviate problems related to imbalanced learning in Big Data scenarios. Although there is currently available an implementation of SMOTE for Big Data (SMOTE-BD), it suffers from important deficiencies in terms of efficiency and scalability, as a consequence of the use of an exact search for nearest neighbors. In Approx-SMOTE, an approximated nearest neighbor search approach is used instead, resulting in an algorithm faster than SMOTE-BD. In particular, the new proposal is around 7 times faster

Code 1:

A basic example, written in Scala, showing how to use Approx-SMOTE for oversampling a dataset.

```
import org.apache.spark.ml.instance.ASMOTE

// reading dataset
val ds = session.read
                .format("libsvm")
                .option("inferSchema", "true")
                .load("binary.libsvm")

// using Approx-SMOTE
val asmote = new ASMOTE().setK(5)
                         .setPercOver(100)
                         .setSeed(46)
val resampledDF = asmote.transform(ds)
```

Table 4
Code metadata (mandatory).

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1.1.0
C2	Permanent link to code/repository used of this code version	https://github.com/mjuez/approx-smote
C3	Legal Code License	Apache-2.0
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	scala
C6	Compilation requirements, operating environments & dependencies	sbt 1.4.2, scala 2.12.10, Apache Spark 3.0.1
C8	Support email for questions	mariojg@ubu.es

when executed in a small cluster of 2 worker nodes, and almost 30 times faster when the number of workers is increased to 10. Therefore, the new proposal achieves great scalability. Regarding to classifiers performance trained with resampled datasets using SMOTE, it has been demonstrated that using approximated nearest neighbors within Big Data environments, is equivalent to use exact nearest neighbors. Approximating the neighbors does not affect the performance negatively because the aim of SMOTE is to generate synthetic instances in the minority class space. For this reason, the use of instances close to the neighborhood seems to be as good as using the exact neighbors.

As there exist other algorithms for fast approximate nearest neighbor search, by means of using hashing for example [20,21], an interesting future research line could be to use that approach within SMOTE and prove whether it also is equivalent to the classical non-approximated version of SMOTE.

Required metadata

Current code version

See Table 4.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The project leading to these results has received funding from "la Caixa" Foundation, under agreement LCF/PR/PR18/51130007. This work was supported by the *Junta de Castilla y León* under project BU055P20 and by the *Ministry of Science and Innovation of Spain* under project PID2020-119894 GB-I00, co-financed through European Union FEDER funds. It also was supported through *Consejería de Educación* of the *Junta de Castilla y León* and the European Social Fund through a pre-doctoral grant (EDU/1100/2017). This material is based upon work supported by Google Cloud.

References

[1] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M.J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, Apache spark: A unified engine for big data processing, Commun. ACM 59 (11) (2016) 56–65, https://doi.org/10.1145/2934664. url:https://doi.org/10.1145/2934664.

[2] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M.J. Franklin, R. Zadeh, M. Zaharia, A. Talwalkar, Mllib: Machine learning in apache spark, J. Mach. Learn. Res. 17 (1) (2016) 1235–1241.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[4] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, Smote: synthetic minority over-sampling technique, J. Artif. Intell. Res. 16 (2002) 321–357.

[5] T. Liu, A.W. Moore, A. Gray, K. Yang, An investigation of practical approximate nearest neighbor algorithms, in: Proceedings of the 17th International Conference on Neural Information Processing Systems, MIT Press, Cambridge, MA, USA, 2004, pp. 825–832.

[6] N.V. Chawla, N. Japkowicz, A. Kotcz, Editorial: Special issue on learning from imbalanced data sets, SIGKDD Explor. Newsl. 6 (1) (2004) 1–6, https://doi.org/10.1145/1007730.1007733.

[7] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1263–1284.

[8] J.F. Díez-Pastor, J.J. Rodríguez, C.I. García-Osorio, L.I. Kuncheva, Diversity techniques improve the performance of the best imbalance learning ensembles, Inf. Sci. 325 (2015) 98–117, https://doi.org/10.1016/j.ins.2015.07.025.

[9] W.C. Sleeman IV, B. Krawczyk, Multi-class imbalanced big data classification on spark, Knowl.-Based Syst. 212 (2021), https://doi.org/10.1016/j.knosys.2020.106598, url:http://www.sciencedirect.com/science/article/pii/S0950705120307279 106598.

[10] H. Neeb, C. Kurrus, Distributed k-nearest neighbors, 2016..

[11] T. Liu, C. Rosenberg, H.A. Rowley, Clustering billions of images with large scale nearest neighbor search, in: 2007 IEEE Workshop on Applications of Computer Vision (WACV '07), 2007, pp. 28–28. doi:10.1109/WACV.2007.18..

[12] M.J. Basgall, W. Hasperué, M. Naiouf, A. Fernández, F. Herrera, SMOTE-BD: An exact and scalable oversampling method for imbalanced classification in big data, J. Comput. Sci. Technol. 18 (03) (2018) 203–209, https://doi.org/10.24215/16666038.18.e23, url:https://journal.info.unlp.edu.ar/JCST/article/view/1122.

[13] J. Maillo, S. Ramírez, I. Triguero, F. Herrera, knn-is: An iterative spark-based design of the k-nearest neighbors classifier for big data, Knowl.-Based Syst. 117 (2017) 3 – 15, volume, Variety and Velocity in Data Science. doi:https://doi.org/10.1016/j.knosys.2016.06.012. url:http://www.sciencedirect.com/science/article/pii/S0950705116301757.

[14] H. Hu, Y. Wen, T.-S. Chua, X. Li, Toward scalable systems for big data analytics: A technology tutorial, IEEE Access 2 (2014) 652–687.

[15] C.-C. Chang, C.-J. Lin, LIBSVM: A library for support vector machines, ACM Trans. Intell. Syst. Technol. (TIST) 2 (3) (2011) 1–27, https://doi.org/10.1145/1961189.1961199.

[16] J.F. Díez-Pastor, J.J. Rodríguez, C. García-Osorio, L.I. Kuncheva, Random balance: Ensembles of variable priors classifiers for imbalanced data, Knowl.-Based Syst. 85 (2015) 96–111, https://doi.org/10.1016/j.knosys.2015.04.022, url:http://www.sciencedirect.com/science/article/pii/S0950705115001720.

[17] A. Benavoli, G. Corani, J. Demšar, M. Zaffalon, Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis, J. Mach. Learn. Res. 18 (77) (2017) 1–36. url:http://jmlr.org/papers/v18/16-305.html..

[18] M. Juez-Gil, mjuez/baycomp_plotting (Nov. 2020). doi:10.5281/zenodo.4244542. url:https://doi.org/10.5281/zenodo.4244542..

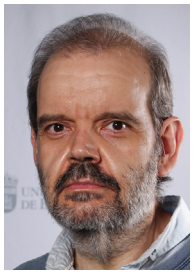
[19] Y. Sun, A.K. Wong, M.S. Kamel, Classification of imbalanced data: A review, Int. J. Pattern Recognit. Artif. Intell. 23 (4) (2009) 687–719, https://doi.org/10.1142/S0218001409007326.

[20] A. Gionis, P. Indyk, R. Motwani, et al., Similarity search in high dimensions via hashing, in: Vldb, vol. 99, 1999, pp. 518–529..

[21] J. Tchaye-Kondi, Y. Zhai, L. Zhu, A new hashing based nearest neighbors selection technique for big datasets (2021). arXiv:2004.02290..

Mario Juez-Gil received his BS, MS and Ph.D. degrees in Computer Science from Universidad de Burgos (Spain) in 2015, 2017 and 2021, respectively. His research interests are focused in parallelization, big data, data mining, and machine learning. As a researcher, he is a former member of the CIG research group at Universidad Politécnica de Madrid in 2017. Since 2018, he is part of the ADMIRABLE research group at Universidad de Burgos, where he is currently in a postdoctoral position.

Á**lvar Arnaiz-González** received his BS and ME degrees in Computer Science from the University of Burgos and a PhD, from the same university, in 2018. His research interests are focused on data mining and artificial intelligence, with a special interest in pre-processing techniques such as instance and prototype selection. He is a member of the ADMIRABLE research group (Advanced Data MIning Research And Business intelligence / Big data / Bioinformatics LEarning).



Juan J. Rodríguez received the BS, MS and Ph.D. degrees in Computer Science from the University of Valladolid, Spain, in 1994, 1998 and 2004, respectively. He worked with the Department of Computer Science, University of Valladolid from 1995 to 2000. Currently, he is working with the Department of Computer Science at Universidad de Burgos, Spain, where he is a Professor. His interests include data science, machine learning and pattern recognition. He has worked on methods for classifier and regression ensembles, time series, feature selection, instance selection, multi-output and big data; with industrial, health, bioinformatics and educational applications.

Carlos Lopez-Nozal received his BS and ME in Computer Science and his PhD from the University of Valladolid. He has been working since 1999 at the University of Burgos, teaching programming and conducting research in software maintenance and learning analytics. He is a member of the ADMIRABLE research group (Advanced Data MIning Research And Business intelligence / Big data / Bioinformatics LEarning).

César García-Osorio received his BS and ME degrees in Computing from Universidad de Valladolid (Spain) in 1994 and 1996, respectively, and his PhD from University of Paisley (now University of the West of Scotland) in 2005. His research interests include data mining, instance selection and big data. He is the coordinator of the ADMIRABLE research group and has been the principal investigator in several regional and national projects on fundamental and applied research in machine learning. Currently he is an associate professor at the University of Burgos (Spain) where he teaches "Intelligent Systems", "Languages Processors" and "Unsupervised Learning".