Paralelización y adaptación a plataformas de cómputo en la nube de algoritmos de mantenimiento y detección de fallos

Programa de Doctorado «Tecnologías Industriales e Ingeniería Civil»

Por
Mario Juez Gil

UNIVERSIDAD
DE BURGOS

Memoria presentada para optar al título de doctor por la Universidad de Burgos

28 de Mayo de 2021

La investigación realizada para el desarrollo de la presente tesis doctoral ha sido parcialmente subvencionada por el Ministerio de Economía, Industria y Competitividad (proyecto TIN2015-67534-P MINECO/FEDER, UE), la Junta de Castilla y León (proyectos BU085P17 y BU055P20 JCyL/FEDER, UE), la Consejería de Educación de la Junta de Castilla y León junto con el Fondo Social Europeo (ayuda pre-doctoral EDU/1100/2017), la Fundación «la Caixa» (acuerdo LCF/PR/PR18/51130007), y Google Cloud.

A quienes me habéis acompañado a lo largo de este camino,
en todo o en parte.

A mí, por haber caminado incluso cuando no tenía fuerzas.

RESUMEN

El *big data* es uno de los temas del momento. Su popularidad dentro del mundo de la ciencia, especialmente en campos relacionados con las ciencias de la computación como pueden ser la inteligencia artificial y el aprendizaje máquina, es indiscutible. Además, esa popularidad también existe fuera del mundo de la ciencia, pues es cada vez más común la aparición de noticias donde se habla de una nueva aplicación revolucionaria del *big data*, ya sea en medicina, en industria, en agricultura, en economía, o incluso en deporte. Se puede decir por tanto, que el *big data* ya forma parte de nuestra vida cotidiana, y su presencia e importancia va a ser cada vez mayor.

El foco de la presente tesis se centra en el papel que tiene el *big data* dentro de la nueva revolución industrial que está teniendo lugar actualmente. Comúnmente nos referiremos a ella con el término Industria 4.0. La característica que más nos interesa de esta nueva industria, es el creciente uso de sensores capaces de monitorizar y registrar de forma continua el funcionamiento de su maquinaria. Gracias a ello surgen nuevas oportunidades para optimizar procesos como el mantenimiento, avanzando hacia nuevas estrategias más eficaces que contribuyan a abaratar costes y maximizar los beneficios. Es el caso del mantenimiento predictivo, el cual, a través de la detección temprana de fallos en todo tipo de maquinaria, como motores de inducción por ejemplo, se pueden programar mantenimientos que ayuden a evitar paradas inesperadas en el proceso de producción. Fruto de ello surgen líneas de investigación sobre el desarrollo de nuevos algoritmos predictivos, o la adaptación de los existentes para hacerlos capaces de trabajar con las grandes cantidades de datos que se generan en estos problemas. Para este último caso, el tipo de adaptación escogida ha sido la paralelización algorítmica para su ejecución en plataformas de cómputo en la nube.

La investigación realizada en esta tesis tiene como resultado cuatro publicaciones científicas. La primera aborda la detección extremadamente temprana de fallos, aislados o simultáneos, en motores de inducción a través de una estrategia basada en PCA y árboles de decisión multi-etiqueta. La segunda propone una implementación paralela del conocido algoritmo de ensembles *Rotation Forest*. La tercera es un estudio experimental sobre el impacto que tiene el desequilibro de datos en *big data*. Y la última es una nueva implementación del conocido algoritmo de sobre-muestreo SMOTE, especialmente diseñada para *big data*.

ABSTRACT

Big data is a topic of our day and age. Its popularity in the scientific world, and especially in the fields of computer science such as artificial intelligence and machine learning, is beyond discussion. Moreover, that popularity is also present outside the scientific world so that it is increasingly common to hear news of a revolutionary new application of big data, be it in medicine or industry, agriculture or economics, and even in sport. It can therefore be said that big data is already part of our daily routine, and its presence and importance can only increase.

The focus of this thesis is on the role of big data in the new industrial revolution, commonly referred to as Industry 4.0, that is currently taking place. The most interesting aspect of Industry 4.0 is the increasing use of sensors that can continuously monitor and record the operating conditions of machinery. Thanks to this innovation, new possibilities arise that allow processes such as maintenance to be optimized, moving towards new, more efficient strategies that help to reduce costs and to maximize profits. Predictive maintenance is one good example, that assists the early detection of faults in all sorts of machinery, such as induction motors, so that maintenance can be scheduled and unexpected outages can be prevented in the production process. As a result, new research lines are emerging about the development of new predictive algorithms or the adaptation of existing ones, so that they can process large volumes of data generated in these sorts of problems. Algorithmic parallelization for execution on cloud computing platforms, was the type of adaptation chosen.

The research conducted in this thesis has resulted in four scientific publications. In the first one, the extremely early detection of isolated or simultaneous faults in induction motors was studied, by employing a strategy based on PCA and multi-label decision trees. In the second publication, a parallel implementation of the well-known Rotation Forest ensemble algorithm was proposed. The third reports an experimental study on the impact of data imbalance on big data. Finally, the fourth and last paper presented a new implementation of the well-known SMOTE over-sampling algorithm, specially designed for big data.

AGRADECIMIENTOS

Creo que no me confundo al afirmar que los agradecimientos son esa parte de la tesis donde todo cuanto decimos a quienes nos han apoyado incondicionalmente a lo largo de estos años, se queda corto. No obstante, si te sientes parte de ello, te invito a complementarlo con el recuerdo de una sonrisa, un abrazo, una broma, o una canción que hayamos compartido en multitud de momentos de esta etapa de mi vida. Ese gesto también era un «Gracias».

Quiero empezar agradeciendo a César y Álvar, mis directores, toda la confianza, cariño, ayuda, y saber que han compartido conmigo en todo momento. También me habría gustado incluir a Carlos López entre mis directores; él supo ver (y pulir) mi brillo incluso antes de terminar el Grado, y desde entonces siempre se ha preocupado por guiarme y darme consejo en todo cuanto he necesitado. Tras ellos, quiero dar las gracias a Juanjo, hombre de pocas palabras pero infinitud de preciosos gestos. Trabajar contigo y aprender de tí siempre es un placer y un lujo extraordinario. Tampoco puedo dejar pasar la oportunidad de agradecer a todos mis compañeros del grupo ADMIRABLE el excelente trato que tienen conmigo, así como su preocupación por mi éxito tanto personal como laboral. Finalmente quiero terminar los agradecimientos académicos acordándome de David Lowe y Lucy Kuncheva, mis dos anfitriones de estancias investigadoras, los cuales me trataron con gran aprecio y haciéndome sentir como si fuese uno de sus propios doctorandos. Todos me habéis acompañado en este viaje y estas palabras nunca podrían haberse escrito aquí sin vuestra cercanía y humanidad.

En lo personal quiero dar las gracias a mi círculo de amigos más íntimo: Manu, Ponce, Loreto, Álvaro, y Ángel. Vosotros habéis vivido (y sufrido) esta tesis de cerca, y aun así, vuestra amistad siempre ha permanecido inalterable. Os quiero sobremanera. Tampoco me quiero olvidar de mostrar mi eterna gratitud a Pedro, mi entrenador, quien cada vez que piso las pistas me hace sentir seguro. Eres una de esas personas que hacen del mundo un lugar un poquito menos hostil.

Para terminar, este último párrafo se lo quiero dedicar a mi familia. Especialmente a mis padres, Mª Jesús y Óscar, así como a mi hermana Laura, quienes, aunque no tienen del todo claro a qué me he estado dedicando a lo largo de estos años, saben que es importante para mí y lo han apoyado con fé ciega. Soy consciente de que muchas de las horas de sueño que la tesis y otras circunstancias me han quitado, también os las han robado a vosotros. Supongo que no hay mayor expresión del amor que esa. Gracias, desde lo más profundo de mi corazón.

ACKNOWLEDGEMENTS

I believe I am not wrong when I say that the acknowledgements are that part of a thesis where everything we say to those who have supported us unconditionally throughout the years somehow falls short of the mark. However, if you feel you were part of it, I would invite you to complement it with the memory of a smile, a hug, laughter, or a song that we may have shared during the many moments of this stage of my life. That gesture also meant "Thank you".

I would like to start by thanking César and Álvar, my supervisors, for all their trust, affection, help, and wisdom that they have shared with me at all times. I would have also liked to include Carlos López among my supervisors; even before I finished my college degree, he has known how to perceive (and even to improve) my talents, and has since then always been concerned about guiding me and giving me useful advice in relation to everything I might ever need to know. After my two supervisors and guide in life, I would like to thank Juanjo, a man of few words yet infinite precious gestures. Working with you and learning from you has always been a great pleasure and a extraordinary luxury. Nor can I miss the opportunity to thank all my colleagues in the ADMIRABLE research group for their excellent treatment towards me, as well as their concern for my personal and professional success. Finally, I would like to end my academic acknowledgements by thanking my two research stay hosts, David Lowe and Lucy Kuncheva, who treated me with great appreciation and made me feel as if I were one of their own PhD students. All of you have accompanied me on this journey and these words could never have been written here without your warmth and humanity.

On a personal note, I would like to thank my closest friends: Manu, Ponce, Loreto, Álvaro, and Ángel. You have lived alongside (and suffered) the work of this thesis closely. Your friendship has been unwavering and I hope it will continue to be so. I love you all very much. I also do not want to miss this opportunity to show my eternal gratitude to Pedro, my coach, who makes me feel safe every time I step onto the athletics track. You are one of those people who make the world a little less hostile.

Finally, I would like to dedicate this last paragraph to my family. Especially to my parents, Mª Jesús and Óscar, as well as to my sister Laura. Although you were not entirely clear about what I have dedicated myself to over these years, you knew that it was important to me and you have supported me with blind faith. I am aware that the very many hours of sleep that the thesis and other related circumstances have wrested from me have also somehow affected you. I guess that there is no greater expression of love than your continued support. Thank you most warmly.

La tesis «Paralelización y adaptación a plataformas de cómputo en la nube de algoritmos de mantenimiento y detección de fallos», que presenta D. Mario Juez Gil para optar al título de doctor, ha sido realizada dentro del programa «Tecnologías Industriales e Ingeniería Civil», en el área de Lenguajes y Sistemas Informáticos, perteneciente al departamento de Ingeniería Informática de la Universidad de Burgos, bajo la dirección de los doctores D. César Ignacio García Osorio y D. Álvar Arnaiz González. Los directores autorizan la presentación del presente documento como memoria para optar al grado de Doctor por la Universidad de Burgos.

V.B. del Director:	V.B. del Director:	El doctorando:
Dr. César Ignacio García Osorio	Dr. Álvar Arnaiz González	D. Mario Juez Gil

En Burgos, a 28 de Mayo de 2021

TABLE OF CONTENTS

LIST OF TABLES

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

LIST OF FIGURES

List of figures

Part I

•

PhD dissertation

INTRODUCTION

1

This introductory chapter explains which ideas, concepts, and research topics have been addressed throughout the development of the thesis. If we had to choose the main, cross-cutting topic for all the research carried out, it has to be big data; undoubtedly one of the most fashionable and interesting topics nowadays. Although big data can be related to almost anything, its relationship with data mining and machine learning has been studied in this thesis, always with a view towards possible applications within the industrial world.

"Information is the oil of the 21st century, and analytics is the combustion engine"

– Peter Sondergaard

1.1 BIG DATA

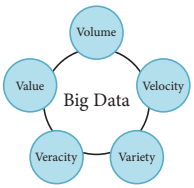
Ever-larger volumes of data are now generated and stored. In (Fernández et al., 2014), the emergence of big data is ascribed to three reasons: (1) more and more of the devices and services we use are continuously recording information, thanks to the advances in sensors and connectivity; (2) the development of storage technologies is advancing by leaps and bounds, so that data collection is less expensive than ever; (3) constant advances in machine learning are enabling the acquisition of a higher degree of knowledge from data. Generally, larger data volumes make it increasingly difficult to extract useful information and knowledge from the data. However, the complexity of the task hardly makes it any less necessary or useful. On the contrary, thanks to the emergence of large data sets, problems that not so many years ago were considered as bordering on science fiction, can nowadays be tackled and solved. Problems such as colouring black-and-white images (Zhang et al., 2016), improving industrial manufacturing processes (Qi and Tao, 2018), implementing predictive maintenance in industry (Yan et al., 2017), predicting protein structures (Callaway, 2020), spotting medical diseases before the appearance of symptoms (Cukier and Mayer-Schoenberger, 2013), and developing strategies for effective prevention and management of epidemics such as COVID-19 (Wang et al., 2020), among many others. Although each problem is solved using different types of techniques, the

common ground is that they are all trained and tuned using large volumes of data.

When discussing big data, mention must be made to the five Vs (Luengo et al., 2020): Volume, Velocity, Variety, Veracity, and Value. *Volume* refers to large volumes of data; *velocity* refers to the high speed at which the data is transmitted; a wide *variety* of formats *i.e*, structured, semi-structured, and unstructured, are usually found in big data; *veracity* is how accurate, precise, and trusted the data are; finally, the usefulness of the data is linked to the *value* of the data.

Big Data

The main issue with big data is the scalability (Hu et al., 2014b). Traditional data collection, processing, and analysis techniques become inadequate when the amount of data they have to work with increases notably. As a solution, distributed computing paradigms are emerging that are able to scale up or down dynamically in a way that depends on the growth of the data.



The five V's of big data

1.1.1 *Cloud computing*

Cloud computing is very most closely related to big data. It refers to a set of hardware and software resources that are delivered as services over the Internet. The services have been referred to as Software as a Service (SaaS), Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and lately, Big Data as a Service (BDaaS). One of the most characteristic aspects of cloud computing, is the appearance of infinite computing resources available on demand (Hashem et al., 2015), which is undoubtedly a game changer for addressing the new problems associated with big data.

The tech giants of our World such as Google, Amazon, and Microsoft have huge data centres containing the infrastructures they offer to their clients in a flexible manner, charging only for the use of certain resources (*e.g.*, networks, computing units, storage, applications, and ad services) over certain periods of time. Both the resources and the time required will depend on the quantity of data processing, as well as the type of processing or analysis that is applied. The companies can therefore use fewer resources at the beginning and will increase them only when their needs increase. This radical advantage eliminates the need to acquire and to maintain expensive proprietary hardware and software licenses in the context of planning for a 'worst requirements' scenario.

In addition to flexibility and "*infinite*" computing resources, the other fundamental advantage of cloud computing is its reliability. Cloud com-

puting providers have certain Service Level Agreements (SLA) where they ensure high availability of all their services. The software that is provided is also specifically configured and carefully tested under their infrastructure, to ensure the best possible performance.

It is worth noting that, although cloud computing is revolutionising the world of the Internet, data science, and even industry, it will not always be the best possible solution, due to certain intrinsic drawbacks. As a general rule, and although cloud providers offer tools for pricing (*e.g.*, Google Cloud pricing calculator[1]), it is very difficult to predict the costs associated with each specific use case. Moreover, costs are always invoiced after the services have been used, so it is not possible to avoid cost overruns by contracting the service before using it. It is therefore necessary to have a very detailed knowledge of the problem to be solved in a cloud environment, as well as its optimal requirements, in order to achieve good cost forecasting. Another scenario where cloud computing may not be the best solution, is for infrastructures that will support the same workload for long periods of time where hardware and software flexibility is not necessary. For example, an industrial predictive maintenance application that will be used to monitor and to process the same amount of data all the time. Having therefore a proprietary infrastructure for that purpose, will in the medium and long term be much cheaper than using cloud-based solutions. However, cloud computing is the perfect solution for training the predictive model used by that application, because it is only done once, despite its high computational requirements. In this case, setting up a proprietary infrastructure, that will be underutilised most of the time, is much more expensive than using a cloud-based service for a certain period of time.

1.2 DATA MINING AND MACHINE LEARNING

In data science, a distinction is often made between data, information, and knowledge. As explained above, increasingly greater volumes of data are generated and stored every day. However, such raw data are of no value until converted through proper pre-processing and cleaning into what is called information, which in turn can be used to search for underlying patterns from which to obtain what is really valuable: knowledge. In the commercial world, knowledge leads to insights that offer potential competitive advantages in economic terms. Data mining and machine learning are the

[1] See the GCloud pricing calculator at: https://cloud.google.com/products/calculator.

two concepts that make all of the above feasible. Although there is no consensus on the definition and boundaries of the two terms, one of the most widespread formulations is described in (Witten et al., 2011), where data mining is defined as the process of finding and describing patterns in data, and machine learning, as the set of techniques used to achieve it. In other words, data mining comprises the set of information discovery tasks that we wish to undertake, and machine learning is the set of tools that we have to solve these tasks.

Data mining and machine learning

In the following, various concepts related to data mining and machine learning will be explained within the framework of this thesis.

1.2.1 *Supervised learning*

Supervised learning is the most common type of machine learning task. It consists of using a labelled data set to train or to fit a model that will be able to label new examples that were not used during training (Mohri et al., 2018). Therefore, obtaining machine learning models in a supervised manner is about discovering the underlying relationships between the features of the training examples and the labels to be predicted. Data labels can be of two types: categorical or numerical, so depending on their type, we will use the term "classification problem" when they are of the first, or "regression problem" when they are of the second. For example, in an industrial scenario, when machinery is susceptible to a particular type of failure, predicting that failure using historical data from past failures is a classification task. In contrast, predicting the economic losses as a result of that failure is a regression task.

Multi-label problems

Normally, when we talk about either classification or regression, we are thinking of the prediction of a single value (either categorical or numerical). However, there are a variety of real-world problems where more than one value, either categorical or numerical, needs to be predicted. Returning to the example of industrial machinery, it is common for several failures of a different nature to occur simultaneously. The prediction of these simultaneous failures cannot be done using classical methods that only predict a single label, so it is necessary to use extended learning methods with which several values may be predicted at the same time. We refer to such problems as multi-label, multi-target, or multi-output problems, and there are a multitude of approaches for dealing with each one (Tsoumakas and Katakis,

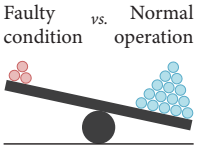
2007). Generally, these approaches fall into two categories: problem transformation methods and algorithm adaptation methods.

On the one hand, problem transformation methods transform multi-label data in such a way that single-label algorithms can be used (Tsoumakas et al., 2009). For example, Label Powerset processes each combination of labels in a data set as one single label; and Binary Relevance creates as many data sets as targets and builds a single-label model for each one.

On the other hand, algorithm adaptation methods modify the behaviour of machine learning methods, so as to provide many outputs at the same time. For example, the C4.5 algorithm was adapted in (Clare and King, 2001) leaving multiple labels at the leaves of the tree and modifying the entropy calculation. Other algorithms such as neural networks or radial basis function networks (Lowe, 2015) can very easily be adapted, by adding as many output neurons as there are labels to be predicted.

The imbalanced problem

The quality of machine learning models is usually strongly influenced by the quality of the data used in the training step. For example, some algorithms have difficulty finding relationships between attributes that do not share the same scale, *i.e.*, they tend to be less accurate if trained on non-normalised data. In such cases, the solution is trivial: data normalisation. However, other types of shortcomings may be present in the data, such as imbalance. In this case the solution is not so simple, in fact, there is no perfect solution because, although continuous development has continued within this field for over two decades, it is still a focus of intense research (Krawczyk, 2016). We say that a data set is imbalanced when the number of examples for each of the labels is unevenly distributed. In a binary classification problem, where there are examples belonging to one of two classes, a data set is imbalanced when the number of examples of one class is much less than that of the other class. Moreover, the minority class is usually the one of interest. For example, in industry, most of the time, all machinery operates without any issues and breakdowns occur only sporadically. However, in the worst-case scenario, such an unforeseen event could bring a production line to a complete halt. A data set recorded in that scenario will be imbalanced, because most of the examples belong to an expected behaviour of the machinery, while only a very small part corresponds to breakdowns, which are also crucial.



An imbalanced data set

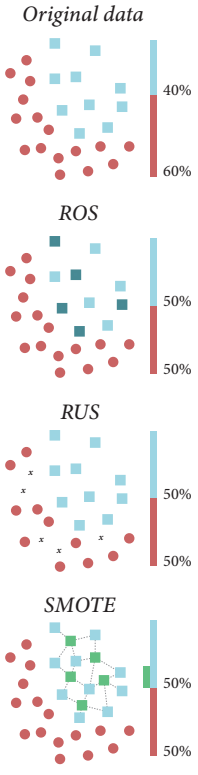
Machine learning algorithms commonly assume by default that the number of examples belonging to each of the labels under consideration, is almost the same. So, when using imbalanced data sets in training, the generated models are biased in favour of the most representative label. In a similar way to multi-label, there are two main families of approaches for tackling imbalanced problems: data-level methods and algorithm-level methods.

Data mining and machine learning

Data-level methods transform the data set to alter its imbalance ratio. Usually the desired output is a data set with a balanced distribution of examples for each label, but depending on the specific problem, other strategies such as reversing the imbalance ratio may be more effective. The most popular approaches are Random Under-sampling (RUS), which randomly removes examples until the desired imbalance ratio is achieved; Random Over-sampling (ROS), which follows the same idea, but duplicates examples instead of removing them; and Synthetic Minority Over-sampling TEchnique (SMOTE), which synthesises and adds new examples to the data set.

Algorithm-level methods modify machine learning algorithms for tackling imbalance. For example, some tree-based methods can be modified to assign different weights to different examples depending on their label. These weights are then taken into account when adjusting the decision boundaries of the model.

On a final note, it should be pointed out that the emergence of big data has underlined a need to advance further down this line of research.



Data-level methods

1.2.2 *Unsupervised learning*

In unsupervised learning, the data used for training are unlabelled. The possible outputs of the model are therefore unknown before the training process, and the algorithm learns the patterns, relationships, and structure of the data without any assistance. Arguably, while the goal of supervised learning was predictive, that of unsupervised learning is descriptive. Data clustering or dimensionality reduction are among the most popular unsupervised learning tasks.

In data clustering, similarities are found between training data. Similar examples are grouped together while different examples belong to different groups (Rokach, 2009). Distance and similarity measures are used to determine whether two examples are similar or dissimilar.

Dimensionality reduction consists of extracting the most valuable information from all the features of the examples, in such a way that a new set of features may be obtained. The new set must be smaller but still capable of

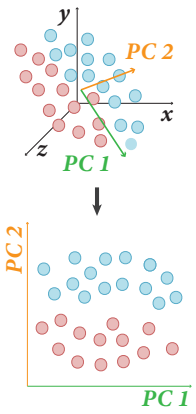
describing each of the examples with equal or even higher fidelity than the complete set. On some occasions, dimensionality reduction is also used for visualisation purposes.

Principal Component Analysis

Introduction

Principal Component Analysis (PCA) is one of the most well-known and popular methods for dimensionality reduction (Jolliffe and Cadima, 2016). However, it should be noted that it can also be used to obtain more representative projections of data without reducing its dimensionality. PCA seeks for the mutually orthogonal directions of greatest variance of the data. The output of PCA is therefore a set of principal components ordered from highest to lowest variance. Up to n principal components can be computed, where n is the number of features that the examples in the data set have. If some of the features in a data set have some kind of linear relationship with each other, *i.e.*, they are correlated, the quality of the models trained on these data may be negatively affected. With this in mind, another strength of PCA is that the generated principal components are uncorrelated, *i.e.*, a principal component cannot be linearly predicted from another principal component.

From a more mathematical point of view, PCA is defined as the eigendecomposition of a covariance matrix. Having a centred data set $\mathbf{X}$[2] composed of m features and n observations, its covariance matrix is calculated as:



PCA 3D to 2D reduction

$$\mathbf{C_X} = \frac{\mathbf{X^T X}}{n-1} \tag{1.1}$$

which is a $m \times m$ square matrix. That covariance matrix can be decomposed into the following product of matrices:

$$\mathbf{C_X} = \mathbf{W} \Lambda \mathbf{W^{-1}} \tag{1.2}$$

where Λ is a diagonal matrix containing the eigenvalues of $\mathbf{C_X}$, and $\mathbf{W}$ is a matrix whose columns are the associated eigenvectors. Using matrix algebra, the previous formula can be solved for computing both, Λ and $\mathbf{W}$. The eigenvectors present as columns of $\mathbf{W}$ are called principal components. Using the eigenvalues of Λ, the order of the principal components can be

[2] That is the data set obtained after subtracting the mean.

determined, such that the larger the eigenvalue, the larger the variance explained by its corresponding principal component. Rotating the data to the new PCA coordinate system, is as simple as multiplying $\mathbf{X}$ by $\mathbf{W}$.

Data mining and machine learning

However, it has to be taken into account that multiplying $\mathbf{X}^T\mathbf{X}$ for calculating the covariance matrix and then computing its eigendecomposition, is not the most computationally efficient way to calculate PCA. Singular Value Decomposition (SVD) is a computationally more efficient and widely used alternative for that purpose. For example, the PCA implementation available in the Apache Spark big data framework takes advantage of SVD when computing the principal components in a parallel efficient way. The linear algebra behind SVD works by decomposing the real-valued data set $\mathbf{X}$ matrix into three matrices:

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T \tag{1.3}$$

where $\mathbf{U}$ and $\mathbf{V}$ are the left and right singular vectors respectively, and Σ is a diagonal matrix containing the singular values. In relation to the calculation of PCA, the matrix $\mathbf{V}$ contains the same values as the matrix $\mathbf{W}$ described before, *i.e.*, the right singular vectors correspond to the principal components. One of the properties of this decomposition is that the singular vectors in both, $\mathbf{U}$ and $\mathbf{V}$, are unitary. It means that $\mathbf{U}^T\mathbf{U}$ and $\mathbf{V}^T\mathbf{V}$ result in the identity matrix. Thus, rotating the data to the PCA coordinate system can be done as follows:

$$\mathbf{X}_R = \mathbf{X}\mathbf{V} = \mathbf{U}\Sigma\mathbf{V}^T\mathbf{V} = \mathbf{U}\Sigma \tag{1.4}$$

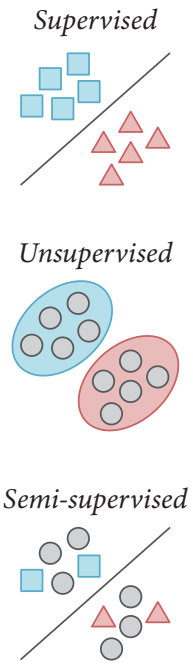
SVD can be rapidly computed, especially when data matrices are large, so it has been commonly used in big data scenarios instead of the classical approach, based on the eigendecomposition of the covariance matrix.

1.2.3 *Semi-supervised learning*

There is a third type of machine learning called semi-supervised learning. It is mainly supervised learning aided with additional unlabelled data to improve its performance (Seeger, 2002). While using fully labelled data sets may appear to be the best option for building the best possible learning models, in practice it is often almost impossible to obtain such data sets. Data labelling processes are normally costly because, for example, the intervention of a human expert might be indispensable. It might involve a lot of

time-consuming work, in order to label a data set consisting of a large number of examples. One strategy to alleviate the workload of the expert may be to label only a small part of the examples corresponding to those that are most representative. The rest of the examples will remain unlabelled but will also be used in the learning process.

In semi-supervised learning, two main groups of methods can be distinguished (van Engelen and Hoos, 2020): inductive, and transductive. Relating these methods to what has been seen so far, inductive methods have a greater similarity to supervised learning than to any other methods. They involve the construction of a model with both labelled and unlabelled data that will process data as yet unseen in the future. The idea behind transductive methods, however, may prove to be groundbreaking, because a model is constructed in such a way that it will only be able to predict those labels of the unlabelled examples used for its training. Thus, there is no distinction between training and prediction when using transductive methods; if there were a need to predict new unlabelled data, then a new model would have to be trained over that data.



Semi-supervised learning has been attracting extra attention from the scientific community over recent years. It is becoming a clear trend that is addressed from various fields related to machine learning, such as neural networks, ensemble learning, and generative learning among others.

1.2.4 *Time series*

A time series is a set of data where all observations have a temporal correlation with each other, *i.e.*, the value of one observation depends on the previous value and will influence the subsequent value. This particular characteristic of this type of data requires the use of machine learning and/or statistical methods specifically designed for them, which are commonly referred to as time series analysis. The values describing a time series are usually numerical, but discrete-valued time series also exist.

The nature of time series data has certain similarities with big data, as (Fu, 2011) states that some of its characteristics include: large data sizes, high dimensionality and continuous updating. Therefore the use of certain pre-processing techniques commonly applied within big data environments, such as dimensionality reduction, could also be beneficial for some time-series problems. Industry-related time series problems may be a good example in that case. Let us consider the aforementioned example where industrial machinery is continuously monitored for inconsistent measure-

ments that can help predict a possible future failure. Such recorded data is considered time series data for which there is also a clear need for efficient processing to gain the highest possible degree of anticipation of unexpected events.

Data mining and machine learning

Time-series problems are usually approached as forecasting problems where historical data are used to predict the next values of the series. Methods based on exponential smoothing or Auto-Regressive Integrated Moving Average (ARIMA) models (De Gooijer and Hyndman, 2006), are among the most widely used and well-known methods. However, alternatives based on more modern techniques are still emerging and becoming more popular, as may be the case of Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997). Despite the popularity of time-series forecasting, the problems related with this particular kind of data could be solved from other perspectives such as classification (supervised learning), clustering (unsupervised learning), and semi-supervised classification (semi-supervised learning).

Although research in the field of time series started several decades ago, it is still one of the most fashionable topics, especially in the machine learning community. This popularity is probably due to its presence within diverse areas such as industry, healthcare or finance, where new problems that require solutions are continuously emerging.

1.2.5 *Ensemble learning*

Ensemble learning refers to the combination of several machine learning models in such a way that the output of the ensemble is somehow "agreed" by all models (*i.e.*, base models) in the ensemble. For example, a popular "agreement" for classification is to give the class which is predicted by most of the base classifiers as the final prediction (*i.e.*, majority voting); whereas for regression, it is to provide the mean of the values predicted by each of the base regressors as the output value. The idea behind ensemble learning is that training several base models and combining their outputs could improve, or at least equal the performance of a single model.

When all the base models forming an ensemble are constructed using the same algorithm, we call it an homogeneous ensemble. A Random Forest model is an example of an homogeneous ensemble, as all the base models are constructed using the random tree algorithm. In contrast, an heterogeneous ensemble is composed of models of a different nature, *e.g.*, decision trees, and neural networks.

Diversity

A great explanation about diversity can be found in (Kuncheva, 2014) where Kuncheva highlights the vital importance of diversity within ensemble learning, because without it there is nothing to combine (*i.e.*, if there is no diversity, the models obtained with each use of the algorithm will be the same). Imagine an ensemble combining several base models whose estimates have no errors, *i.e.*, all of them are perfect models. The combined output will therefore be equal to the output provided by a single perfect model, so that the ensemble will merely waste computational power. Obviously, the perfect model does not exist, because there will always be some error, however small. With this in mind, one of the fundamental requirements to make ensembles worthwhile is to be able to ensure that each of the base models makes errors on different instances, *i.e.*, the models in the ensemble are diverse. In this way, the combination of all the models will mean individual errors may be ignored and overall accurate estimations will be increased.

When talking about diversity, we usually refer to the terms strong model and weak model. We say that a model is strong when it is highly accurate, *i.e.*, it makes a low number of errors. In contrast, if a model barely improves the performance of a random model, we refer to it as a weak model. Hence, in ensemble learning, the use of weak base models usually contribute to more diverse ensembles. Given the above, one might think that the best ensembles have to be composed of very weak models to maximise diversity, but there is a point where diversity can be counterproductive. Therefore, the main challenge of ensemble learning is to find an optimal amount of diversity that will contribute to the accuracy of the ensemble estimates.

Bagging and boosting

There are several ensemble methods, of which two of the most popular are bagging (Breiman, 1996) and boosting (Schapire, 1999). In these methods, N base models are trained using N random sub-samples from the training data set. Usually, data are sampled with replacement, *i.e.*, the same example can be present several times in the same sub-sample. The aim of this idea is to obtain diversity through specialisation of the different base models in different parts of the hypothesis space. In bagging, all base models are independently trained, while in boosting, the errors made by the previously trained model are used to weight the examples in the following sub-sample, so that the next model will focus on correctly estimating the previously misestimated examples. Thus, the bagging training process is considered to be

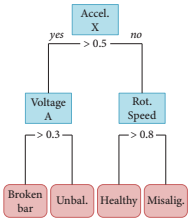
parallel because there are no dependencies between the base models, and the boosting training process is considered sequential. Adapting bagging to be exploited in big data scenarios is straightforward and high-performance, while adapting boosting is a more challenging task. In any case, adaptations are currently available for both.

Data mining and machine learning

Decision tree-based ensembles

Decision tree-based algorithms (Breiman, 1984) are popular in machine learning and data mining, mainly due to their simplicity and explainability. A decision tree is formed of nodes and branches, it typically starts with a single node, commonly referred to as the root node. Each node in the tree indicates a decision to be made which, unless the node is a leaf node, will indicate how the data should be divided according to the value of one of its features. Branches serve to establish the path between decisions until a leaf node is reached where no more branches appear and a final choice (*i.e.*, model output) is decided upon.

The way a tree chooses which decision to make at each node and how to split data into two branches is determined by the split criteria. There are several criteria, the choice of which will depend on the type of problem (*e.g.*, classification or regression) and on the nature of the data. Two of the most popular split criteria, used mainly in classification, are Gini impurity and Information gain (based on entropy). The CART algorithm uses the former, and other well-known algorithms, such as ID3 or C4.5, use the latter. Equations 1.5 and 1.6 show how Gini impurity and Entropy are calculated, respectively, for a classification problem:



A decision tree for predicting motor faults

$$\text{Gini} = 1 - \sum_{i=1}^{C} {p_i}^2 \tag{1.5}$$

$$\text{Entropy} = \sum_{i=1}^{C} -p_i \log_2(p_i) \tag{1.6}$$

where, C is the number of classes and p_i is the probability of a class in a certain branch. Since both criteria are probability-based, for imbalanced data sets, split criteria will be biased in favour of the most represented class. Weighted variants of both (Gini impurity and Entropy), could help to overcome that issue because they assign (often unequal) weights to the different

classes. In Equations 1.7 (Casquilho, 2020) and 1.8 (Singer et al., 2020), w_i refers to the weight to be applied to the i-th probability.

$$\text{WGini} = \sum_{i=1}^{C} w_i p_i - \sum_{i=1}^{C} w_i {p_i}^2 \tag{1.7}$$

$$\text{WEntropy} = \sum_{i=1}^{C} -w_i p_i \log_2(p_i) \tag{1.8}$$

One of the major shortcomings of decision trees is that they are unstable. So, small variations in the training data will result in completely different tree models. Instability, however, is a desirable property for building ensembles based on decision trees, as it contributes to the generation of diversity. Hence, this type of algorithm is very attractive for ensemble learning, all the more so since decision-tree training is computationally inexpensive, relatively speaking, and potentially parallelisable.

The most popular tree-based ensemble is Random Forest, which is a simple, yet powerful method, that trains several random trees on several bootstraps of the data (*i.e.*, bagging of random trees) (Breiman, 2001). The difference between a random tree and a decision tree and therefore between Random Forest and bagging of decision trees, is that a random tree is trained using only a random subset of features. The combination of both (bootstrapping and feature sub-sampling) leads to very different and diverse base models.

Rotation Forest (Rodriguez et al., 2006) is another well-known and successful tree-based ensemble. Unlike Random-Forest training, Rotation-Forest training uses the entire feature space, but different data set rotations to train each of the base models in the ensemble. The rotations are computed using PCA, which, although computationally expensive, helps to obtain a diversity that enhances the overall performance of the ensemble.

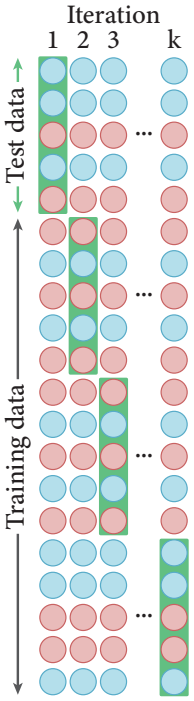
Finally, decision tree-based algorithms can be particularly powerful, in the sense that they provide extra information on how decisions are taken (*i.e.*, explainability). Thanks to this, an expert can acquire added knowledge on a certain problem and how to solve it, which can be of interest in some environments such as industry. In addition to predicting certain malfunctions, knowing their causes can contribute to more efficient maintenance.

1.2.6 *Evaluating and comparing machine learning models*

Data mining and machine learning

Measuring the performance of a machine learning model requires the use of an evaluation strategy and a certain performance metric (or set of metrics) must be chosen to measure the performance of a model. Once the performance has been quantitatively characterised, it will be possible to make comparisons that allow us to choose between several algorithms, or to adjust the parameters of an algorithm. The cross-validation strategy will be described below, as well as some of the most popular performance metrics and how the comparisons are usually carried out using statistical tests.

Cross-validation



k-fold cross-validation

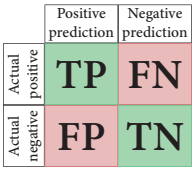
The most straightforward way to evaluate a machine learning method is to divide the data set into two parts: a training set and a test set. The learning method is used to obtain a model trained with the data from the training set, and then the testing data set is used to estimate the error of the model on new unseen instances. The differences between the actual and predicted labels will be used to calculate the model performance. It is essential that the test set examples are not used in the training phase, otherwise the performance estimation will be too optimistic. The main drawback of this strategy is that the evaluation is strongly conditioned by the division of the data set, which is usually drawn at random.

Cross validation is an alternative approach to splitting the data set into training and testing. In this technique all data instances are used in both training and testing. For this purpose, the set is divided into k parts, named folds, which are used to build k models. $k-1$ folds are used to train each of the models and the remaining fold is used to estimate the performance. Each model is evaluated with a different fold, and the final evaluation result is the average of the k evaluations. Cross-validation therefore makes it possible to obtain performance estimations that are less conditioned by the random division of the data set. In addition, several repetitions of cross-validation (i.e., repeated k-fold cross-validation) can be used, in order to achieve even more reliable performance estimations. Finally, it must be noted that this technique is not perfect and it has some drawbacks. The major one is its computational expense, because the number of models to be trained can be very high depending on the value of k and the number of repetitions chosen. This consideration has to be carefully taken into account, especially in big data, where training a single model may require a lot of time and computational resources.

Performance metrics

A quantitative characterisation of the performance of machine learning methods is necessary for their evaluation. This is done through the use of performance metrics that summarise in a single numerical value how good or bad the model predictions are. There are a wide variety of metrics that can be used, some of which are suitable for classification problems, while others are intended for regression, clustering, *etc*. Moreover, depending on the domain, a certain metric or set of metrics will be preferred over others, because different performance metrics measure different trade-offs in the predictions made by a machine learning model, and therefore, according to some metrics the performance of a model could be good, while for other metrics the same performance could be worse (Caruana and Niculescu-Mizil, 2006). Since all the publications forming part of the present thesis are focused on classification problems, for the sake of simplicity, only classification metrics, and specifically those related to binary classification problems are discussed below.

In binary data sets there are examples belonging to two possible classes: positive (P) and negative (N). While predicting binary data sets, four basic indicators can be derived: True Positive (TP), which refers to the number of positive examples correctly classified as positive; True Negative (TN), which refers to the number of negative examples correctly classified as negative; False Positive (FP), which refers to negative examples wrongly classified as positive; and False Negative (FN) which refers to positive examples wrongly classified as negative. From the aforementioned indicators, several different metrics could be calculated. Some of them such as accuracy (equation 1.9) take no account of data imbalance, while others such as precision (equation 1.10) or recall (equation 1.11) do take imbalanced data into account.



Confusion matrix showing binary prediction indicators

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.9}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1.10}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{1.11}$$

There are many other popular metrics such as Area Under the Curve (AUC), F_1 Score, Matthews Correlation Coefficient (MCC), and Geometric Mean, which may be very useful to use for certain problems such as imbalance. As each metric reflects a certain type of behaviour of the machine learning method under evaluation, it is important to use a broad set of performance metrics, in order to obtain as accurate a picture as possible of the overall performance of the algorithm.

Statistical tests

Part of the process of evaluating and validating machine learning algorithms is the comparison of differently parameterised variants, or even with other state-of-the-art algorithms. The aim of these comparisons is to demonstrate that the proposed algorithm yields an improved performance over the rest of the alternatives. The standard procedure for these sorts of comparative studies is to choose several data sets (commonly a well-known benchmark), for each of the algorithms under comparison. The performance of the models were therefore evaluated using the appropriate metrics, and finally statistical tests were applied to decide upon whether the improved performance result from the proposal was really significant.

In statistics, there are two broad families of tests: parametric and non-parametric. For parametric tests, the data to be compared must meet certain restrictions, some of which are often not met when comparing algorithms. For example, one is the independence of the samples being compared, which is not the case if cross-validation is applied. Therefore, non-parametric tests are preferable. In (Demšar, 2006), a set of non-parametric statistical tests for comparing algorithms over multiple data sets were proposed. This way of testing, although with certain shortcomings, has been adopted by the machine learning scientific community and can be found in several state-of-the-art studies. Demšar proposed the Wilcoxon Signed-Rank test for comparing two algorithms on several data sets, while he proposed the Friedman test for comparing many algorithms on several data sets.

The Wilcoxon Signed-Rank test ranks the absolute value of the differences in performance of two classifiers for each data set, and compares the ranks for the positive and the negative differences. Let R^+ be the sum of ranks for the positive differences (*i.e.*, the first algorithm outperforms the second), and let R^- be the sum of the opposite. When the difference is 0,

the sum of ranks is split between R^+ and R^-:

$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \tag{1.12}$$

$$R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \tag{1.13}$$

The Friedman test ranks the algorithms for each data set separately, assigning 1 to the best performing algorithm, 2 to the second best, and so on. In case of ties, average ranks are assigned. Then, this test compares the average ranks of the algorithms, calculated as $R_j = \frac{1}{N}\sum_i r_i^j$, where r_i^j is the rank of the j-th (of k algorithms) on the i-th of N data sets. The next step is to compute a Friedman statistic proposed in (Iman and Davenport, 1980) for testing whether the null-hypothesis (all algorithms are equivalent) is rejected. Rejection allows us to proceed with the Nemenyi post-hoc test, which states that the performance of two classifiers is significantly different, if the corresponding average ranks differ by at least the critical difference

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{1.14}$$

where, q_α is the critical value based on the Studentized range statistic divided by $\sqrt{2}$.

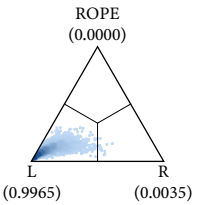
Bayesian analysis

The tests described above, known as frequentist null-hypotesis testing, are still in fairly widespread use. Nevertheless, they have several shortcomings that make them unsuitable for comparing classifiers (Demšar, 2008; Benavoli et al., 2017). Alternatives have therefore been emerging that employ Bayesian analysis which, among other aspects, permit a reliable estimation of the probability of one classifier performing better than another, or the probability that both classifiers may be considered equivalent, if their performance difference is not over a certain threshold (*e.g.*, 1%).

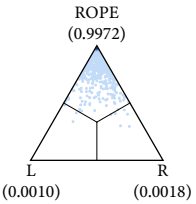
Bayesian analysis can be used to compare two algorithms on a single data set, or on multiple data sets. For the first type of comparison, cross-validation must have been used, because the results of each fold are used to perform a Bayesian correlated *t* test (Corani and Benavoli, 2015), which

accounts for the correlation due to the overlapping training sets. As a result, a Student's t-distribution shows how the probabilities are distributed. On the other hand, for comparing two algorithms on multiple data sets there are two types of test. The first type is a Bayesian Wilcoxon Signed-Rank test based on the Dirichlet process (Benavoli et al., 2014), which computes rankings in a similar way to that seen on the frequentist Wilcoxon Signed-Rank test. Then, these rankings are used to estimate the subsequent probabilities using a Bayesian approach. The second type is a hierarchical test for using the cross-validation evaluations in pair-wise comparisons (Corani and Benavoli, 2015). The results of the aforementioned tests can be depicted through histograms or density plots. It is also possible to use ternary plots (simplex). Graphical representations are usually divided into three areas of interest (L, R and ROPE), although in density plots they can be reduced to two (L and R):

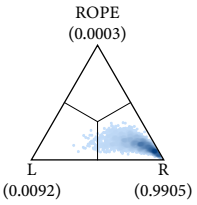
Parallel computing



The model at the left side of the comparison is statistically better than the model at the right.



Both models are statistically equivalents.



The model at the right side of the comparison is statistically better than the model at the left.

- Right probability (R): The probability of the classifier on the right-hand side of the comparison outperforming the classifier on the left-hand side.
- Left probability (L): The probability of the classifier on the left-hand side of the comparison outperforming the classifier on the right-hand side.
- Region of Practical Equivalence (ROPE): The probability that the performance of both classifiers will be equivalent. A threshold value setting the width of the ROPE to 0 means that it is not considered possible for both classifiers to have equivalent performance.

Finally, it should be noted that the Friedman test for comparing multiple algorithms on multiple data sets also has its corresponding Bayesian approach (Benavoli et al., 2015), but it is not widely used and is not yet implemented in the most popular libraries for these types of tests (*e.g.*, Baycomp[3]).

1.3 PARALLEL COMPUTING

In parallel computing, several instructions are executed simultaneously. It arises as an alternative to sequential computing where one instruction must wait for the previous one to finish before starting its execution. Many of the

[3]The Baycomp library is publicly available at: https://baycomp.readthedocs.io/.

problems solved by computers can be divided into smaller problems that are easier to compute, which can be executed in parallel. A very typical example is matrix multiplication, where each element of the resulting matrix can be computed independently of the rest. While parallel computing is not new, the emergence of big data is driving the development of parallel infrastructures and algorithms specially designed and optimised to run on such infrastructures. In the following, data and task parallelism will be detailed; existing parallel frameworks such as Apache Spark will be introduced; and finally, MapReduce, the most popular parallel computing paradigm, will be described.

1.3.1 *Data and task parallelism*

When talking about parallelism, a distinction has been made between data parallelism and task parallelism. Recently, along with deep learning, a new form of parallelism called model parallelism (Jia et al., 2018) has been emerging, but only the previous two are within the scope of this thesis.

Data parallelism consists of decomposing a data set into smaller parts, before applying the same operation or function to each of them. The parts are distributed across different parallel processor nodes, so that the operations are simultaneously performed. The amount of parallelism that can be achieved depends on the size of the data set and the number of parts that may be derived from it, which makes it a very effective type of parallelism for big-data problems. The parallel matrix multiplication example is a case of data parallelism: two matrices are decomposed into rows and columns which are then distributed. Each processor node, which has only one row of one matrix and one column of the other, will compute a single position of the resulting matrix.

In task parallelism, each processor node performs a different operation to the same data. The operations are independent of each other, so they can be computed at the same time. The amount of parallelism that can be achieved is limited by the number of different independent operations (McCormick and Ahrens, 2005), which does not necessarily have to be increased if the amount of data increases.

On a final note, when different types of operations on different parts of the data are performed simultaneously, it is referred to as mixed data and task parallelism.

1.3.2 *Parallel frameworks*

Parallel computing

The development of parallel algorithms is not a new topic. Libraries such as OpenMP are specifically designed to exploit the full potential of parallel architectures and have been available for years (Dagum and Menon, 1998). Thanks to these tools, it is possible to obtain deeply optimized implementations. However, they require a high degree of knowledge of many aspects such as the specific library, the parallelisation strategy designed for the particular algorithm, and the parallel architecture where the algorithm will be executed. The emergence of cloud computing, with dynamic and flexible configurations of distributed environments, has resulted in developers having no control over where and how their algorithms will run. There is therefore a need to extract from the algorithm, the way in which the different parallel operations will be distributed. For this reason, parallel frameworks such as Apache Hadoop or Apache Spark have emerged. They are responsible for orchestrating where and when each of the parallel instructions should be executed, thus allowing the developer to worry only about which operations to parallelise.

Apache Hadoop is a framework for processing large amounts of data by implementing the MapReduce programming model. One of its main features and strengths is the Hadoop Distributed File System (HDFS), designed for reliable storage of very large data sets in clusters (Shvachko et al., 2010). Despite its great popularity, Hadoop has some shortcomings that slow it down in certain scenarios such as iterative jobs that repeatedly apply a function to the same data set. The way Hadoop uses the disk to ensure fault tolerance, significantly penalises its performance because it requires the data set to be read from disk many times. In (Zaharia et al., 2010), this issue was identified and a new framework called Apache Spark was proposed (Zaharia et al., 2012, 2016). Spark introduces the Resilient Distributed Dataset (RDD) concept, which is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. It means that an RDD as an in-memory object, is not stored on disk, but is still fault-tolerant. This makes Spark almost 100 times faster than Hadoop[4]. Another strength of Apache Spark over Hadoop, is its built-in machine learning library (mllib) (Meng et al., 2016), which comes with scalable algorithms for a broad variety of machine learning tasks like classification, regression, and clustering. It also defines useful interfaces for growing the library with new parallel algorithms. As a result, Apache Spark

[4]Source: Apache Spark official website (https://spark.apache.org/).

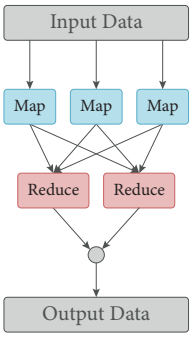
has become one of the most important analytic frameworks for large-scale data processing.

1.3.3 *MapReduce*

Introduction

MapReduce (Dean and Ghemawat, 2008) is a successful programming model for processing large data sets in a parallel manner. The computation is defined by the user through the functions *map* and *reduce*. The input data is divided into several distributed parts formed by key/value pairs. The *map* function is applied to each key/value pair for computing a list of intermediate values that also consists of key/value pairs. Then, as many *reduce* functions as intermediate keys are executed for merging intermediate values together for a specific key, forming a possibly smaller set of values. Zero or one output values are typically returned after invoking the *reduce* function.

It should be noted that not all algorithms can be implemented following a "mapreduced" strategy. There are methods such as bagging ensembles that can be easily adapted to MapReduce, because each base classifier is independently trained with a bag of the data. In a very basic approximation, one base model is trained in each map invocation. Then, all the base models are combined into the resulting ensemble model, by means of one or more reducers. However, the same cannot be said for boosting ensembles, because the training of one base model is conditioned by the errors of the base model trained beforehand. This is inherently sequential and therefore, there is no mapping option.



MapReduce execution flow

Although MapReduce is not perfect, its efficiency, fault tolerance, and scalability, has turned it into one of the most widely used programming models in big-data scenarios nowadays (Fernández et al., 2017; Ramírez-Gallego et al., 2018a).

1.4 INDUSTRY 4.0

Throughout history, technological advances have been exploited by industry to transform, to evolve, and to improve technological processes. If mechanization and the use of electrical energy gave a strong boost to the industrialization process, we are now witnessing a new paradigm shift driven by digitalization and the use of information technologies in certain industrial processes. This is the so-called Industry 4.0 (Lasi et al., 2014). Thanks to the growing use of sensors in all kinds of industrial machinery and the advent of the Internet of Things (IoT), their operation is starting to be monitored in

depth. Sensor measurements are continuously recorded and it is possible to apply data mining and machine learning techniques to provide intelligence to various processes, thus moving towards a more productive and sustainable industry (Ge et al., 2017).

1.4.1 *Predictive maintenance*

Industry 4.0

Over time and with use, it is inevitable that the components of industrial machinery will gradually degrade. When degradation is severe, breakdowns occur that will have to be repaired, thus leading to a forced shutdown of a machine in the best-case scenario, or the shutdown of an entire production line in the worst-case scenario. Another aspect that should be taken into account is that a breakdown in one component can cause unexpected breakdowns in other related components, increasing the costs of the repair in both time and money. This strategy based on fixing the machine when it breaks down, is known as run-to-failure management (Mobley, 2002). An alternative is preventive maintenance, which is time-driven, *i.e.*, maintenance work is based on elapsed time or hours of operation. However, it has been increasingly demonstrated that carrying out scheduled maintenance is flawed and unreliable, because of random failure patterns that are not dependent on operating times. Nowadays, the most widely accepted maintenance technique, which uses the actual operating condition of industrial machinery, is called predictive maintenance.

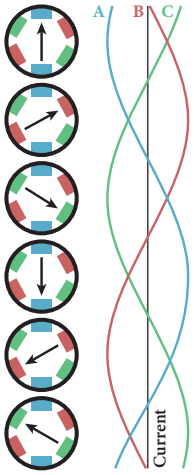
Predictive maintenance is one of the most widely promoted techniques that forms part of this evolution towards smart industry. Visual inspection, which is the oldest and yet the most powerful and widely-used predictive maintenance method, is being replaced by automated methods that use advanced signal processing techniques based on artificial intelligence (Hashemian, 2010). The use of these techniques can assist the experts responsible for maintenance with quick and efficient detection of anomalous behaviours. Moreover, in the most extreme case, they could even replace the expert completely, by autonomously detecting malfunctions and automatically triggering the necessary maintenance task. Therefore, predictive maintenance should improve productivity, product quality, and the overall effectiveness of manufacturing and production plants.

1.4.2 *Induction motors*

Induction motors are a type of electric variable speed drive supplied with alternating current. These motors, in general have many attractive features

over other types of electric motors, such as simplicity, reliability, robustness, low maintenance requirements, wide speed range, high reliability, low torque noise, and low cost (Jain and Kumar, 2018). They can also operate in hostile (i.e., dirty and explosive) environments. There are two types of induction motors: wound-rotor and squirrel-cage. The latter is the most commonly used in industry, due to its competitive prices and power efficiency.

The way a squirrel-cage induction motor works is as follows: a three-phase induction drive has three stator windings that develop a rotating magnetic flux that rotates at synchronous speeds. Depending on the motor pole number and supply frequency, this speed will vary. The rotating flux intersects the rotor windings, inducing an electromagnetic field, which in turn results in circulating current. The rotor currents produce a second magnetic flux that interacts with the stator flux. As a result, torque that accelerates the machine is produced. After rotor acceleration, when an equilibrium speed is reached the induced rotor current is sufficient to produce the torque demanded by the load (Shakweh, 2018).



Three-phase rotating magnetic flux

Sensors and physical magnitudes

As stated before, industrial machinery is starting to be equipped with different sorts of sensors to measure its operation. In induction motors, there are sensors to measure magnitudes such as the voltage, the intensity of the current, and the rotating speed. Accelerometers or acoustic sensors are intended to measure vibrations; and temperature sensors can register the thermal data. Although the list of useful sensors to measure induction motors operation is greater, those mentioned above are the most widely used for predictive maintenance.

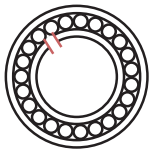
Operating conditions and mechanical faults

When industrial machinery operates as expected, *i.e.*, without any issues, it is generally said to be in a healthy operating condition. In addition to being the most common condition, it is desirable to preserve it for as long as possible. For that purpose, the scientific community has focused its attention on reducing the occurrence of undesired conditions, which can be either electrical or mechanical. This thesis only addresses mechanical faulty conditions, some of which are described below.

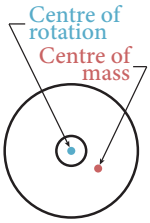
Induction motors have rolling element bearings whose constant motion causes their degradation over time. This can lead to faults that may occur

in the outer race, the inner race, the cage, or a rolling element (Randall and Antoni, 2011). These faults are generally referred to as bearing defects and its early diagnosis is usually through the use of vibration data. Another part of the motor prone to deteriorate are the rotor bars. Usually made of aluminium, copper, or brass, the rotor bars are situated around the surface of the rotor, passing from one end of the rotor to the other (Yu et al., 2011). The movement and load on the motor can lead to the rupture of one or more of these rotor bars. This fault is known as a broken rotor bar, and motor vibrations can be useful for diagnosing it, as well as the motor currents (Combet, 2014). Rotor unbalance is a condition in which the centre of mass of the shaft and its fixed components is not coincident with the centre of rotation. It generates excessive vibration which may damage the mechanical elements of the motor, so vibration due to unbalance must be reduced, in order to extend the life of the machine (Saleem et al., 2012). The last type of fault that will be discussed in this section is misalignment, which occurs when the shaft of the rotor is improperly aligned. This type of fault also causes machine vibration. It can be diagnosed by measuring it and by analysing motor currents (Kumar et al., 2015). A wide variety of alternative faults may occur and can also be diagnosed, nevertheless only the most commonly studied in the literature, and the most relevant to the topic of big data have been described.

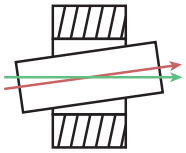
Broken Rotor Bar



Bearing Defect



Rotor Unbalance



Misalignment

Experimental test bench

Data sets generated in a laboratory or a controlled installation, rather than data sets generated from machinery operating in production environments are used in a large part of the experimental studies on predictive maintenance and fault detection. An experimental test bench is carefully prepared with a certain preset conditions such as power supply, and load current. Commonly, the faulty condition is artificially induced by damaging certain parts of the machinery intentionally. For example, bearing or rotor bar defects are induced by drilling holes on the surface of those elements. Mass unbalance is simulated by coupling elements with a certain weight such as bolts. Finally, misalignment can be easily provoked by shifting the belt in the load pulley forward or backwards. This experimental methodology makes it possible to obtain data sets where the faulty conditions are sufficiently well represented. As the data sets were recorded under the same conditions, the results for each one are comparable.

MOTIVATION AND GOALS

2

"The goal of science is to understand the fundamental reality and the goal of technology is to change that reality."

– Kedar Joshi

One of the characteristics of a thesis based on a compendium of publications, is that each paper is the result of a kind of isolated research process. Therefore, in these kinds of theses, there is no one single motivation, but as many motivations as there are publications. Nevertheless, although it is stated more explicitly in some parts of this thesis than in others, all the publications share some common ground: industrial maintenance and fault detection within big data. Hereinafter, in this chapter, the main motivations and goals of the research carried out in this thesis are summarised.

As industry evolves, new technological advances are implemented with the main objective of improving production processes. Basically, if the economic costs are reduced and the profit margin increases, then a production process may be considered to have been improved. Throughout history, there have been several industrial revolutions that have dramatically improved and boosted production processes. The latest industrial revolution is said to be happening right now and is being driven by, among other things, the adoption of the use of sensors to monitor the functioning of industrial machinery. This opens up new opportunities for the application of data mining and machine learning techniques. One of the characteristics of data sets that are generated from sensors is their sheer size (*i.e.*, big data). Because of this, the use of classical machine learning techniques may not be sufficient and therefore, a move towards the implementation of solutions specifically designed for big data is needed. These sorts of solutions are still barely used in industry nowadays and are undoubtedly the way to improve processes such as predictive maintenance and fault detection. The objectives pursued in this thesis to address the problems described above are twofold:

- The study and proposal of machine learning techniques to tackle big-data problems.
- The application of big-data solutions to industrial-maintenance and fault-detection tasks.

The first objective is approached from the point of view of algorithmic parallelisation. In particular, by adapting ensemble algorithms to big-data frameworks and, more specifically, the Apache Spark framework. Thanks to this, the new algorithms will be enabled to work at scale on cloud-based cluster infrastructures. Within this objective the study of certain problems are also considered, such as imbalance. These problems are intrinsic to the characteristics of large data sets, especially those generated in industry. Finally, the second objective covers the early detection of faults that occur in rotating machinery used in industry, specifically in induction motors.

DISCUSSION OF RESULTS

The scientific results of the present thesis are in form of journal papers, original software publications, and other communications such as posters, or conference papers. All the contributions are listed below.

3

"I think combining predictors is one of the two big breakthroughs in prediction."

– Leo Breiman

3.1 JOURNAL PAPERS

1. **Title:** Early and extremely early multi-label fault diagnosis in induction motors.

   Authors: Mario Juez-Gil, Juan José Saucedo-Dorantes, Álvar Arnaiz-González, Carlos López-Nozal, César García-Osorio, David Lowe.

   Journal: ISA Transactions (JCR: Q1, SJR: Q1).

   Year: 2020

   DOI: 10.1016/j.isatra.2020.07.002

2. **Title:** Rotation Forest for Big Data.

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

   Journal: Information Fusion (JCR: Q1, SJR: Q1).

   Year: 2021

   DOI: 10.1016/j.inffus.2021.03.007

3. **Title:** Experimental evaluation of ensemble classifiers for imbalance in Big Data.

   Authors: Mario Juez-Gil, Álvar Arnaiz-González, Juan J. Rodríguez, César García-Osorio.

   Journal: Applied Soft Computing (JCR: Q1, SJR: Q1).

   Year: 2021

   DOI: 10.1016/j.asoc.2021.107447

3.2 ORIGINAL SOFTWARE PUBLICATIONS

Original Software Publications

1. **Title:** Approx-SMOTE: fast SMOTE for Big Data on Apache Spark.

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

   Journal: Under review at Neurocomputing (JCR: Q1, SJR: Q1).

3.3 COMMUNICATIONS

1. **Title:** Multi-label Classification of Induction Motor Faults Using PCA and Decision Trees.

   Authors: Mario Juez-Gil.

   Type of communication: Poster presentation.

   Place: Bilbao Data Science Workshop (BIDAS).

   Year: 2018.

2. **Title:** Doctoral Consortium: Paralelización y adaptación de algoritmos de mantenimiento y detección de fallos a plataformas de cómputo en la nube.

   Authors: Mario Juez-Gil.

   Type of communication: Doctoral Consortium.

   Place: XVIII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA 2018).

   Year: 2018.

3. **Title:** Early multi-fault diagnosis in induction motors.

   Authors: Mario Juez-Gil, Juan José Saucedo-Dorantes, Álvar Arnaiz-González, Carlos López-Nozal, César García-Osorio.

   Type of communication: Extended abstract.

   Place: 15th International Conference on Industrial Engineering and Industrial Management (ICIEIM) - XXV Congreso de Ingeniería de Organización (CIO2021).

   Year: 2021.

3.4 OPEN SOURCE REPOSITORY

The source code of the algorithms developed in the thesis, as well as the Jupyter Notebooks containing experimental evaluations, are publicly available on GitHub: https://github.com/mjuez.

3.5 OTHER JOURNAL PAPERS

This section lists other articles resulting from research in which the doctoral student has actively collaborated. These papers have not been included in the compendium, because the participation of the PhD candidate has not been as relevant as in the other selected articles or because they are the result of research carried out on data sets that do not belong to the big data category.

1. **Title:** A regression-tree multilayer-perceptron hybrid strategy for the prediction of ore crushing-plate lifetimes.

   Authors: Mario Juez-Gil, Ivan Nikolaevich Erdakov, Andrés Bustillo, Danil Yurievich Pimenov.

   Journal: Journal of Advanced Research (JCR: Q1, SJR: Q1).

   Year: 2019

   DOI: 10.1016/j.jare.2019.03.008

2. **Title:** An experimental evaluation of mixup regression forests.

   Authors: Juan J. Rodríguez, Mario Juez-Gil, Álvar Arnaiz-González, Ludmila I. Kuncheva.

   Journal: Expert Systems With Applications (JCR: Q1, SJR: Q1).

   Year: 2020

   DOI: 10.1016/j.eswa.2020.113376

3. **Title:** Rotation Forest for multi-target regression.

   Authors: Juan J. Rodríguez, Mario Juez-Gil, Carlos López-Nozal, Álvar Arnaiz-González.

   Journal: International Journal of Machine Learning and Cybernetics (JCR: Q2, SJR: Q1).

   Year: 2021

   DOI: 10.1007/s13042-021-01329-1

3.6 OTHER COMMUNICATIONS

Other communications

1. **Title:** Aplicaciones de la técnica de "topic model" en repositorios software.

   Authors: Carlos López-Nozal, César García-Osorio, Álvar Arnaiz-González, Mario Juez-Gil.

   Type of communication: Conference paper.

   Place: XVIII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA 2018).

   Year: 2018.

CONCLUSIONS

4

In this thesis, big-data-related classification problems have been approached from three points of view: practical (*i.e.*, application of methods to solve real-world problems), methodological, and experimental. The conclusions derived from these perspectives are discussed below.

4.1 APPLIED MACHINE LEARNING

"The hardest problems of pure and applied science can only be solved by the open collaboration of the world-wide scientific community."

– Kenneth G. Wilson

Firstly, data mining, and machine learning techniques have been specifically applied within industry to solve, for example, predictive maintenance problems in induction motors, thereby contributing to the progress of multidisciplinary science. The most notable result is accurate detection and extremely early diagnosis of various types of faulty operating conditions, by means of combining classical statistical techniques such as PCA with simple yet powerful machine learning methods such as decision trees. An approach that also addresses and solves the added difficulty of those cases in which several faults occur simultaneously; known as the so-called multi-label problem in the machine learning world.

In the literature, the different types of faults studied in this thesis, which are disparate in nature, have mainly been addressed in isolation. For each type of fault, different signal-processing strategies (based on time-domain, frequency-domain, and time-frequency-domain analyses) have been proposed for different types of sensors. For example, current signature analysis has been used to diagnose unbalance, and vibration analysis has been used to diagnose bearing defects. Within these research areas, the contribution of this thesis consists of using large volumes of data recorded by sensors of a different nature (*e.g.*, voltages, currents, and vibrations). Each set of registered measurements are processed with PCA to obtain a reduced alternative representation and then, those representations are used to train a predictive model (*i.e.*, a decision tree). The resulting model can classify as many as four different operating conditions to a high degree of accuracy (over 87%), as well as their combinations, using only the measurements from the first 0.5

seconds of motor operation. Moreover, the model can produce accurate predictions irrespective of motor load or its operating frequency.

4.2 METHODOLOGICAL CONTRIBUTIONS

Methodological contributions

Many methods have been designed and implemented throughout the development of the thesis. The parallel (*i.e.*, MapReduced) adaptation of the Rotation Forest algorithm is undoubtedly the most important one. Its good performance with normal-sized data sets was reported in (Rodriguez et al., 2006) and was expected for large data sets, although it had to be tested and verified, due to the intrinsic peculiarities of big data. Rotation Forest has also shown good scalability, outperforming state-of-the-art alternatives that incorporate PCA such as PCARDE (García-Gil et al., 2018). Certain insights were gained from the experimental study carried out to validate the Rotation Forest adaptation that can be particularly useful when optimising training times without sacrificing the accuracy of the resulting models. The first of these, relating to the size of the bootstrap for calculating PCA, indicates that values as low as 10% (as opposed to the 75% used in (Rodriguez et al., 2006)) have no negative impact on the accuracy of the resulting model. It occurs due to the large size of the data sets where a sample consisting of 10% of the instances will have a reasonable size. The second interesting insight comes from the idea of rotation of random forests (Stiglic et al., 2011), which demonstrated that reusing the same rotation for training several trees with big data clearly improved the training time of the ensemble without losing accuracy.

The other main methodological contribution of the thesis is an approximated SMOTE implementation for creating artificial instances. The SMOTE over-sampling method, originally used k-nearest neighbours (k-NN) for creating artificial instances, the computational complexity of which is a clear drawback when the size of the data sets increases, as happens with big data. The proposal described in this thesis has demonstrated that using hybrid spill trees (Liu et al., 2004) for approximating the k-NN search offers a useful SMOTE solution for big data. It is much faster and more scalable than the existing alternative that uses a distributed exact k-NN approach (Basgall et al., 2018), and it replicates its benefits to overcome the imbalanced classification problem (*i.e.*, classifiers trained on over-sampled data through exact and approximated SMOTE presented equivalent levels of accuracy).

4.3 EXPERIMENTAL RESEARCH

The experimental methodology used with machine learning involves comparing and validating methods by testing their performance on several data sets. Experimental studies carried out in the past used normal-sized data sets, however, with the advent of big data, many of the findings of these studies may not be transferable to large data sets.

Conclusions

In this thesis, it has been found that imbalanced learning is one of the topics that should be carefully revisited. The use of data-level approaches along with decision tree-based ensemble classifiers has led to some unexpected conclusions. For example, oversampling methods, such as SMOTE and ROSE, based on the generation of synthetic examples, were not as beneficial with large data sets as they were with normal-sized data sets. Hence, simpler and faster pre-processing methods such as random over-sampling are preferable. Moreover, the use of weighted impurities such as weighted Gini, rather than any kind of pre-processing, were demonstrably faster and equally effective at addressing imbalance.

Another interesting insight obtained from the experimentation on imbalanced data sets was that the use of different evaluation metrics yielded a more informed view of the results and, consequently, a better evaluation of the performance of a classifier can be assured.

After evaluating the results from the experiments reported in the publications, it was concluded that a large number of benchmark data sets were available for the comparison of algorithms in conventional machine learning studies, while there were far fewer benchmark data sets available reduced (*i.e.*, less than 10) in big data. Moreover, big data experiments take a long time and are computationally expensive, making it difficult to perform experiments on a large number of data sets. As a result, statistical comparisons may not be as informative as we would like and could even lead to weak or erroneous conclusions. Therefore, and as pointed out in (Stapor et al., 2021), special care must be taken over the selection of the most suitable type of statistical tests and over the selection of the data sets, which should be guided by the domain-specific requirements of the particular problem under evaluation (*e.g.*, imbalanced learning).

FUTURE LINES

5

The many and various knowledge areas of science are in continuous development. Sometimes, two areas of *a priori* unrelated knowledge advance to a point where their combination gives rise to revolutionary scientific advances. Known as multidisciplinary science, the present thesis is a good example of convergent lines of research. Here, one part is focused on the study and development of techniques related to data mining, machine learning, and big data, and the other part is focused on the application of these techniques to the resolution of industrial problems such as predictive maintenance. Therefore, future research lines arising from the thesis could be oriented in both directions.

"Moving forward in science is as much unwinding the distorted thinking of the past as it is putting a clearer idea on the table."

– Craig Venter

5.1 ROTATION FOREST

The research concerning Rotation Forest is one of the cornerstones of the thesis, from which numerous lines of future work may emerge.

5.1.1 *Semi-supervised learning*

Semi-supervised learning (van Engelen and Hoos, 2020) is increasingly attracting the interest of the scientific community nowadays. In this type of learning, data sets have both labelled and unlabelled examples. These sorts of data sets are more frequently found in big data, as labelling instances is often a costly process. One of the simplest techniques used in semi-supervised learning is called self-training (Yarowsky, 1995), which consists of incrementally training predictive models as follows: initially a model is trained using only the labelled data, then that model is used to predict the label of unlabelled examples. The example (or examples) of labelling with the highest confidence level is added to the set of labelled examples and the model is re-trained with that set. This process is repeated until there are no unlabelled examples left, or until a pre-set limit of iterations is reached. Co-training (Blum and Mitchell, 1998), tri-training (Zhou and Li, 2005), and democratic co-learning (Zhou and Goldman, 2004) are among other well-known examples of semi-supervised techniques.

Self-training can be used along with Rotation Forest as was done in (Fazakis et al., 2017). However, its adaptation to big data is still pending. Furthermore, the way that Rotation Forest uses PCA can be seen as a generation method of multiple views of the training data set that might enable to develop novel co-training or tri-training-based solutions. Therefore, the use of Rotation Forest in semi-supervised learning problems can be very promising and challenging.

Predictive maintenance

5.1.2 *Imbalanced learning*

There are several studies on Rotation Forest variants for imbalanced and highly imbalanced classification. For example, Hellinger distance decision trees were used in (Su et al., 2015), a fuzzy clustering approach was proposed in (Hosseinzadeh and Eftekhari, 2015), and Rotation Forest was combined with random under-sampling or SMOTE in (Fang et al., 2016). All of these approaches are yet to be adapted and validated for big data, as well as new ones that may arise, for example, from the combination of Random Balance (Díez-Pastor et al., 2015a) and Rotation Forest. To validate whether these "complex" solutions are truly beneficial specifically for big data could also be a compelling line of research. It could all help advance research on imbalance in big data, which, as indicated in the experimental study of this thesis, is still at a very early stage.

5.1.3 *Changes and improvements*

Although it has been shown that the adaptation of Rotation Forest for big data proposed in this thesis has good scalability, the parallel implementation of PCA that it uses could penalise its performance depending on the parameters chosen. Therefore, research in the computation of PCA and its optimisation in new ways and in a distributed manner could lead to new, faster, and more efficient Rotation Forest implementations. Recently in (Gemp et al., 2021), PCA computation was addressed as a competitive game where scalability was one of its most outstanding strengths. An improved Rotation Forest based on that PCA solution may be a very interesting line of future research.

5.2 PREDICTIVE MAINTENANCE

The development of predictive maintenance techniques based on the use of artificial intelligence is attracting a lot of interest in the scientific commu-

nity. A wide variety of strategies can be found in the literature, moving from the classical spectral analysis (Orhan et al., 2006) to complex and expensive techniques based on deep learning (Nguyen and Medjaher, 2019).

Although it at times may appear that everything has already been studied, one of the main concerns of the industry is to optimise costs. Thus, the costs of these solutions based on deep learning, although very powerful, mean they are not always affordable. Therefore, the development of solutions based on simpler and cheaper machine learning techniques, such as ensembles, will always be an interesting line of research. Especially when in many cases their own performance compared to the performance of deep learning algorithms can be statistically indistinguishable.

Future lines

5.2.1 *Rotation forest*

Given that the amount of data recorded by sensors installed in industrial machinery can be very large (*i.e.*, tens of thousands of measurements per second), the use of algorithms specially designed for big data could be very useful for diagnosing abnormal behaviours. In (Santos et al., 2018) a Rotation Forest ensemble classifier turned to be the best solution for diagnosing faults in wind turbines. Therefore, an interesting future line of work could be to apply the big-data adaptation of the Rotation Forest proposed in this thesis to big-data industrial problems.

5.2.2 *Imbalanced learning*

In industry, data sets recorded in real environments (*e.g.*, in production) are usually highly imbalanced, *i.e.*, most of the data corresponds to healthy operating conditions while faulty conditions are dramatically underrepresented. Therefore, training machine learning models on real data could favour models that are biased towards healthy condition, and that ignore the faults. There are some studies that use data-level approaches for tackling imbalance in real applications (Zhang et al., 2018), but these sorts of problems within big data are yet to be approached. In that regard, the insights obtained from the experimental evaluation on imbalance performed in this thesis may be applicable to industry, thus opening up an interesting future line of research.

Part II

•

Publications

EARLY AND EXTREMELY EARLY MULTI-LABEL FAULT DIAGNOSIS IN INDUCTION MOTORS

1

This journal paper presents a new intelligent multi-fault diagnosis method which uses multiple sensor information for predicting the occurrence of single, combined, and simultaneous faulty conditions in an induction motor.

Authors: Mario Juez-Gil, Juan José Saucedo-Dorantes, Álvar Arnaiz-González, Carlos López-Nozal, César García-Osorio, David Lowe.
Type: Journal
Published in: ISA Transactions, 106: 367–381 (JCR: Q1, SJR: Q1)
Year: 2020
Keywords: Multi-fault detection, Early detection, Multi-label classification, Principal component analysis, Load insensitive model, Prediction at low operating frequencies
Reference: Juez-Gil et al. (2020)

ABSTRACT

The detection of faulty machinery and its automated diagnosis is an industrial priority because efficient fault diagnosis implies efficient management of the maintenance times, reduction of energy consumption, reduction in overall costs and, most importantly, the availability of the machinery is ensured. Thus, this paper presents a new intelligent multi-fault diagnosis method based on multiple sensor information for assessing the occurrence of single, combined, and simultaneous faulty conditions in an induction motor. The contribution and novelty of the proposed method include the consideration of different physical magnitudes such as vibrations, stator currents, voltages, and rotational speed as a meaningful source of information of the machine condition. Moreover, for each available physical magnitude, the reduction of the original number of attributes through the Principal Component Analysis leads to retain a reduced number of significant features that allows achieving the final diagnosis outcome by a multi-

label classification tree. The effectiveness of the method was validated by using a complete set of experimental data acquired from a laboratory electromechanical system, where a healthy and seven faulty scenarios were assessed. Also, the interpretation of the results do not require any prior expert knowledge and the robustness of this proposal allows its application in industrial applications, since it may deal with different operating conditions such as different loads and operating frequencies. Finally, the performance was evaluated using multi-label measures, which to the best of our knowledge, is an innovative development in the field condition monitoring and fault identification.

1.1 INTRODUCTION

The operation of most industrial applications is provided by rotating machinery powered by electricity. In that context, the Induction Motor (IM) represents the most used electrical machine, due to its robustness, efficiency, and safety. Despite its effectiveness and reliability, the occurrence of sudden and unexpected faults can affect the IM operation, provoking undesirable interruptions that may lead to crucial stoppages in industrial processes. Thereby, the continuous monitoring for assessing the operating condition to identify faults in rotating machines, that are powered by electricity, is a key safeguard for ensuring the machine reliability and safety in industrial production systems (Liu and Bazzi, 2017; Liu et al., 2018). In that sense, most of the condition monitoring strategies comprise three main tasks: fault detection, fault isolation, and fault identification (Bayar et al., 2015). Specifically, fault detection provides information relating to the existence of a malfunction; fault isolation locates the faulty components; and, fault identification labels the fault type that has been identified (Gao et al., 2015).

Recently, the use of Artificial Intelligence (AI) and classification techniques have been included as a part of the condition monitoring strategies. These techniques usually consist of several steps: data collection (gathering fault-related measurements); feature calculation, extraction and selection (determining fault-related features and measurements); and, training an algorithm (building a classification/regression model with the selected features) (Martin-Diaz et al., 2018; Lei et al., 2016; Saucedo-Dorantes et al., 2017). In that regard, AI techniques such as Neural Networks (NN) and Fuzzy-based inference systems have been developed as classification algorithms. Consequently, automatic Fault Detection and Diagnosis (FDD),

based on AI algorithms using a trained AI model that is capable of identifying the relationship between both input data (information provided by sensors) and output data (machine-assessed condition), can replace the human intervention.

The training process of an AI algorithm usually involves the modeling of several inputs that specifically are the set of features estimated from the available physical magnitudes that characterize the working condition of the machine under observation (Hui et al., 2017). Normally, the output is a single value (a 'label') that can be numeric (for regression problems) or discrete/categorical (for classification problems) but represents a specific condition/operation of the assessed system. If there are only two classes, e.g. fault or no fault, then the problem involves binary classification, and if there are more labels (two or more fault types), it is a multi-class problem. Nevertheless, more than one label can be assigned whenever multiple faults occur simultaneously, which is also possible and a critical issue that must be addressed. This task, known as multi-label classification, has recently been gaining ground (Tsoumakas and Katakis, 2007; Zhang and Zhou, 2014).

Therefore, the main contribution of this paper lies in the proposal of a new intelligent multi-fault diagnosis method based on multiple sensor information for assessing the occurrence of single, combined, and simultaneous faulty conditions in an IM. The condition assessment and fault identification is performed by analyzing the available physical magnitudes, vibrations, stator currents, voltages, and rotational speed, through the Principal Component Analysis a multi-label classification tree. Additionally, the contribution and novelties of the proposed multi-fault detection and identification approach include:

1. Multi-label classification: failure prediction of multiple faults that occur simultaneously.

2. Insensitivity to load and frequency: capable of achieving good performance regardless of whether the motor is running under variable load or no-load conditions. Likewise, training is possible without taking into account the operating frequency.

3. High-performance results in both transient and in steady state regimen: system predictions are generated both in the transient state and in the easier steady mode.

4. The method is able to identify faults without the need of expert knowledge of each failure.

5. Extremely-early detection: the classification performance of the method is remarkable, predicting from signals within only 0.5 of a second.

Related works

The rest of the paper is organized as follows: a brief description of related works is be provided in Section 1.2; in Section 1.3, a general introduction to the relevant Data Analysis procedure is given; then, an explanation of the data set and the process of parameter setting is offered in Section 1.4; the proposed diagnostic methodology is presented in Section 1.5; in Section 1.6, the results of the experimental study for the performance assessment of the method is discussed; and, finally, in Section 1.7, the main conclusions of the paper is summarized, and some lines of future research is discussed.

1.2 RELATED WORKS

The development of strategies for condition monitoring and fault identification in industrial processes is important to reduce unexpected stoppages avoiding economic losses and also for ensuring the continuous machining operations. As stated earlier, undesirable faulty conditions may occur repeatedly during the continuous working cycle of an IM. This issue has normally been addressed as a single-label problem in most condition monitoring strategies that, one by one, identify each single fault mode.

Commonly, the application of condition monitoring strategies for fault identification in IMs has been divided into two widely studied groups: mechanical and electrical problems (Kazzaz and Singh, 2003). In this regard, most of mechanical failures presented in IM are usually associated with problems to the bearing components, axis eccentricities, misalignments, and couplings, among others; in contrast, electrical problems are associated with faults in the stator and rotor windings, such as broken rotor bars, insulation defects, and short circuits (Garcia-Perez et al., 2012; Naderi, 2017). On the other hand, different physical magnitudes, such as voltage, stator currents, temperatures, load torque, and mechanical vibrations are usually monitored in most industrial processes in order to assess its current condition. Indeed, a big deal of the industrial processes usually prefer the application of condition monitoring strategies that are based on the installation of non-invasive sensors. Additionally, the analysis of a unique physical measurement remains as the most preferred approach to condition assessment. Hence, vibration-based and stator current-based analyses have been widely

addressed by using signal processing for condition monitoring (Gangsar and Tiwari, 2017). At present, the classical fault identification in electrical rotating machines with industrial applications involves the analysis of vibration or stator current signals, separately. In fact, prior to the application of condition monitoring strategies in industrial applications, the development of new proposals can also be based on theoretical approaches that consider equivalent mechanical, electrical or magnetic models that aim to simulate different faulty conditions in a specific and real system (Naderi, 2017). Likewise, different data-driven condition monitoring strategies have been proposed to detect faults and their sudden appearance in IMs and, although significant and advantageous results have emerged, most of these proposals have been focused on the analysis of single fault-mode (Gao et al., 2015; Choudhary et al., 2018). Thus, the critical issue of multiple faults and the adaptation of condition monitoring strategies to detect them, in view of their harmful consequences during the regular operation of an IM, therefore suggests that the measurement of different physical magnitudes represent a coherent approach for being considered during condition monitoring applied to electrical machines.

The rest of the section follows the structure of the review by Liu & Bazzi (Liu and Bazzi, 2017). According to those authors, Fault Detection and Diagnosis (FDD) methods can be grouped into four groups: time-domain methods, frequency-domain methods, time-frequency-domain methods, and artificial-intelligence-based methods. Traditionally, signal processing techniques based on time domain, frequency domain, and time-frequency domain approaches are the most common and well-known processing strategies (Rauber et al., 2014). As this paper is not a complete state-of-the-art review, it is recommended that interested readers consult the following papers (Liu and Bazzi, 2017; Liu et al., 2018).

1.2.1 *Time-domain FDD methods*

These methods are used to identify defective machine functions. The use of time-domain methods may be preferred due to one of its advantages is the simplicity in computation burden and implementation, although their low sensitivity, especially in noisy environments, might also be mentioned. Some relevant methods in this category are (Marques Cardoso et al., 1999; Duan and Živanović, 2015; Mahmoud et al., 2016).

1.2.2 *Frequency-domain FDD methods*

Related works

The intrinsic mechanics of all IM mean that their characteristic fault-related patterns are within the frequency domain. Whenever a fault is present, it therefore provoke changes to the spectra, thus, the Fast Fourier Transform (FFT) is among the simplest of mathematical methods of classifying signals from spectrum measurements. Nevertheless, the overlapping of several faults within the same frequency band and the difficulties associated with low loads and frequencies are some of the main drawbacks of the frequency-domain methods. These methods can directly process both raw and preprocessed signals (Betta et al., 2002; Thomas et al., 2003). Other methods subtract the fault-independent components from the faulty signal, to obtain a residual that can be analyzed (Stack et al., 2004).

1.2.3 *Time-frequency-domain FDD methods*

As is known, non-stationary signals are difficult to process and time-frequency domain methods therefore yield results of greater accuracy. Time-frequency methods use a moving time window on which spectral analysis is performed, with the result that the non-stationary components can be treated as constants. These analyses require more complex implementation and computational complexities (log-linear or higher). Some examples of these techniques are the wavelet transform (Yan et al., 2014) and the short-time Fourier-transform (Bouchikhi et al., 2013).

1.2.4 *Artificial Intelligence-based FDD methods*

The consideration of AI techniques in condition monitoring strategies has been towards automatic fault diagnosis. Over the years, different variants of neural networks have been one of the most frequently used techniques: the multilayer perceptron (Chow et al., 1991) and radial-basis functions (Ghate and Dudul, 2011), among others. On the other hand, evolutionary methods (such as genetic algorithms and particle swarm optimization) are widely used to compute the best algorithm parameters. Whereas, Support Vector Machines (SVMs), another popular family of algorithms in this context, have likewise demonstrated good classification performance in hierarchical structures (Keskes and Braham, 2015). Recently, deep learning has been used for FDD (Lei et al., 2016), although its main drawback is the huge amount of data that is needed for training FDD systems. The recent

work of Liu et al. (Liu et al., 2018) is strongly recommended for a detailed review of AI-based methods.

1.2.5 *Unresolved Issues*

Even though the literature on FDD methods is very copious, there are still some unresolved issues that must be addressed, the most critical issues that have to be faced are:

Early and extremely early multi-label fault diagnosis in induction motors

- Simultaneous fault detection (Liu and Bazzi, 2017): most methods analyze a single or, at best, a couple of faults, and are not designed for the simultaneous identification of multiple faults.
- Non-stationary conditions (Liu and Bazzi, 2017): the authors make significant progress examining fault detection under non-stationary conditions.
- Integrated methodology/system (Liu and Bazzi, 2017; Liu et al., 2018): this point can be analyzed from two perspectives. Firstly, there are four main faults in induction motors (broken bar, misalignment, unbalance, and bearing failure); different methods approach different kind of faults, underlining the need for an integrated system that is capable of identifying all four faults (Liu and Bazzi, 2017). Secondly, FDD are usually built as a combination of different parts instead of a whole diagnostic system. The incorporation of both feature extraction and "intelligent" (Machine Learning) FDD in a single system would represent an integrated method (Liu et al., 2018) that would be easier to use in industrial production.

1.2.6 *Our proposal vs. related works*

Although several works focused on the fault detection and identification in IM have been proposed, the important advantages of the diagnosis method presented in this paper are outlined below.

Most condition monitoring approaches that have been proposed are classically based on the analysis of stator current and vibration signals and, the fault diagnosis is usually performed through the estimation of characteristic fault-related frequency patterns. However, although the occurrence of different faulty modes is among the aims of such proposals, the fault identification has been widely addressed as a single-fault identification problem (Martínez-Morales et al., 2018); thus, the advantage of the method pro-

posed in this paper is that it can be applied to assess the occurrence single, combined, and simultaneous faulty conditions.

In fact, the implementation of complex condition monitoring structures including stages relating to calculation, extraction and selection of feature have also been proposed. Those stages are specifically implemented to improve and highlight the characterization of a large set of features estimated through time-domain techniques such as CWT (Continuous Wavelet Transform), HHT (Hilbert Huang Transform) and EMD (Empirical Mode Decomposition) (Konar and Chattopadhyay, 2015; Saucedo-Dorantes et al., 2017; Shen et al., 2013). Although such optimization processes are mostly performed with genetic algorithms, the use of those techniques requires additional knowledge for their proper implementation. Whereas, in the proposed method, the need for expert knowledge on each type of failure is not necessary, as the novel implementation of the PCA and decision trees means that there is no need to configure the condition monitoring approach with specific parameters, depending on the system to be assessed.

Theoretical background

Additionally, the condition monitoring strategies that can detect the occurrence of multiple and combined faults are available in only very few studies. Indeed, such proposals are based on complex time-frequency domain techniques, such as the high-resolution spectral method of multiple signal classification (MUSIC) (Romero-Troncoso et al., 2016), which compromises the computational burden. The strategies are also limited to the identification of multiple faults under the start-up or steady-state regime (Romero-Troncoso et al., 2016). Hence, in this work, the assessment of multiple combined faults is performed during the start-up and steady-state regime under variable load conditions and, in addition, the low computational burden of this tool means that it can be used for real-time condition monitoring.

1.3 THEORETICAL BACKGROUND

If sensor data and records of mechanical operations are not filtered, organized and analyzed, to extract information that may be of use to understand a process and to solve a problem, they are of no inherent utility. The challenge is how to perform the early diagnosis of multiple malfunction conditions in an IM. The techniques used to perform the process of extracting information from raw data, range from the simplest, such as data summary and visualization, to the most sophisticated, such as the application of multivariate statistical methods, Machine Learning and Data Mining. Thus,

in this study, a statistical technique, Principal Component Analysis (Subsection 1.3.1), is combined with a Machine Learning technique (Subsection 1.3.2), Decision Trees, to solve a multiple prediction problem, commonly known as multi-label classification (Subsection 1.3.3).

1.3.1 *Principal Component Analysis*

The Principal Component Analysis (PCA) (Jolliffe, 1986) is commonly used for data exploratory analysis (in data visualization) and it is still one of the most well-known and popular methods for dimensionality reduction (Jolliffe and Cadima, 2016; Roman-Rangel and Marchand-Maillet, 2019). The main objective of the PCA attempts to establish the directions, in the original high-dimensional space, with the highest variance (directions that, assuming Gaussian noise distributions, usually correlate with a more informative content). As a result, the projection of the multivariate measurement data onto the most significant orthogonal principal component directions is the data and the noise reduction step in the process with the lowest loss of overall data variance. The assumption is that the primary information loss is the noise subspace, which is assumed not to be informative for the multi-label classification problem.

More formally, if X is an $n \times m$ matrix with the data (m attributes for n instances), the principal components are obtained by solving an eigenvalue problem:

$$\text{cov}(X)\mathbf{c}_k = \lambda_k \mathbf{c}_k, \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m,$$
$$\text{cov}(X) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^T,$$

where, $\text{cov}(X)$ is the covariance matrix of X, $\mathbf{c}_k$ is a principal component (and an eigenvector), $\mathbf{x}_i$ is a row of X (data instances), and $\overline{\mathbf{x}} = (1/n)\sum_{i=1}^{n}\mathbf{x}_i$ is the average of the $\mathbf{x}_i$. In practice, an explicit calculation of the covariance matrix is not necessary, as other matrix decompositions, such as singular value decomposition (SVD), can be used to obtain the same eigenvectors with greater efficiency, especially if only a few principal components are needed.

In this paper, the value of n is the number of experiments performed in the experimental test bench and the value of m depends on the measurement sampling frequency, e.g. 12 kHz or 3 kHz (see Section 1.4.1). In this paper, 12 matrices were used, one for each of the measurements.

When PCA is used as a dimensionality reduction method, the determination of the number of principal components to use is one of the most important decisions. The wrong choice could reduce the impact of the results. Retaining too few components could lead to a loss of important information (Zwick and Velicer, 1986). Whereas, retaining too many components could produce high dimensional data, which might still embody much of the measurement noise that is detrimental to the performance of subsequent AI models.

There are empirical prescriptions for determining the number of components, some of which were compared in (Zwick and Velicer, 1986). Kaiser's eigenvalue greater than one rule (K1 rule) (Kaiser, 1960) is one of the most popular methods used for the selection of the number of retained principal components (Zwick and Velicer, 1986). The rule states that if the eigenvalue of a component is greater than 1.0, then the component is significant. Catell's scree test (Cattell, 1966) is a graphical method that consists of plotting the eigenvalues in descending order (scree plot) and looking for an "elbow" in the plot, which implies a steep slope from large to small eigenvalues (Ruscio and Roche, 2012).

1.3.2 *Classification trees*

A decision tree is an algorithm in which a set of tests, organized in a hierarchical way, is used to guide the process of class assignment or output value calculation. The process begins at the root node, where the value of one of the attributes of the instances to be classified is compared (or whose output value is needed to be determined). Depending on the result of this comparison the process is directed to one or several branches (typically, two in a binary decision tree), where nodes apply new tests to conditionally branch further in the tree. The process continues until a leaf node (a node without further branches) is reached, at which point a class is assigned to the instance, or there is a function to calculate the output value for that instance.

The process of building a decision tree also starts at the root node. In each node, the training set is divided into subsets, for which it is necessary to determine the best division attribute and the value for the division into subsets. If only one threshold value is used, the division will be into two subsets, but it would be possible to use several ranges of values permitting a node to have more than two branches. The criterion for the selection of attributes and values is the optimization of a merit function. Some of these functions are: the Gini Index (Breiman, 1984), Information Gain, and Gain

Ratio (Quinlan, 2014). Once the attribute and the values of the node have been selected, they are used to determine through which branches each instance proceeds, and the process is repeated for the corresponding arrival nodes. The process terminates when the number of instances is less than a certain value, or when other stop criteria are satisfied. The leaf nodes are assigned the majority class of the instances that have reached that node (or those instances are used to calculate a function that gives the output value when the tree is used for prediction).

Decision trees are popular in Data Mining and Machine Learning for several reasons: they are quick to build, they are interpretable, and they are unstable (that is, small changes in the training set results in very different trees). This last property has made them suitable for the construction of ensembles,[1] both of classifiers (Maudes et al., 2012; Díez-Pastor et al., 2014) and of regressors (Pardo et al., 2013; Arnaiz-González et al., 2016).

In this research, decision trees were used on account of their speed of construction and because decision trees adapted to multi-label problems already exist. Further details on these types of problems are given in the next section.

1.3.3 *Multi-label and multi-output learning*

Supervised Machine Learning is used primarily to solve two types of problems:

- The determination of the relationship between the attributes of an instance and its class, in order to be able to label new instances, that is, so that one class may be assigned to them from a finite number of existing classes; usually single-label classification problems.
- The prediction of the value of a continuous attribute, from those attributes that are already available; known as regression problems.

However, an instance can belong to several classes simultaneously, or, several attributes must be predicted at the same time, which are known as multi-label (classification) and multi-output (regression) problems, respectively. Although the prediction of each of the attributes could be addressed as a problem of single-label or single-output prediction, it has been shown

[1] An ensemble is an algorithm that makes predictions by combining several models, but these models should be diverse, and the instability of the trees makes them ideal in this context.

that simultaneous prediction can be beneficial (Read, 2010), since learning algorithms can exploit the relationships between the attributes to be predicted. So in recent years, research has increasingly focused on adapting existing techniques to these new paradigms, or presenting new techniques to resolve these multi-label problems. Papers have been published on classification (Tsoumakas and Katakis, 2007; Zhang and Zhou, 2014), regression (Spyromitros-Xioufis et al., 2016; Waegeman et al., 2019), and pre-processing techniques (Arnaiz-González et al., 2018b,a; Kordos et al., 2019).

Experimental study

In this paper, a decision tree with multi-label learning was designed to identify motor malfunctions, even for simultaneously occurring faults. The model was also evaluated using multi-label performance measures, which is discussed in Subsection 1.4.2.

1.4 EXPERIMENTAL STUDY

In this section, the method of data acquisition obtained from an experimental test-bench based on a laboratory electromechanical system is described (Subsection 1.4.1). The multi-label evaluation measures is then be introduced (Subsection 1.4.2), and the method for determining the optimal number of principal components is also subsequently described (Subsection 1.4.3).

The proposed condition monitoring method was programmed in Python using the Scikit-learn library (Pedregosa et al., 2011). And, the experimental data was analyzed by using multi-label measures and 10×10-fold cross-validation (cross-validation instead of train-test splits were used, as it has been demonstrated that the cross-validation test better exploits the available data (Kohavi et al., 1995)). For the purpose of achieving load-insensitive models, the training data contained examples of the IM functioning with three types of load conditions: no-load, half-load, and full-load.

All source code is publicly available on Github: https://github.com/mjuez/early-im-fault-diagnosis.

1.4.1 *Data set description*

The experimental test bench used to validate the proposed condition monitoring approach consists of a pulley-belt electromechanical system driven by a 971-W three-phase IM (model WEG 00136APE48T) and coupled to an automotive alternator. The IM had 1 pair of poles, a power supply of

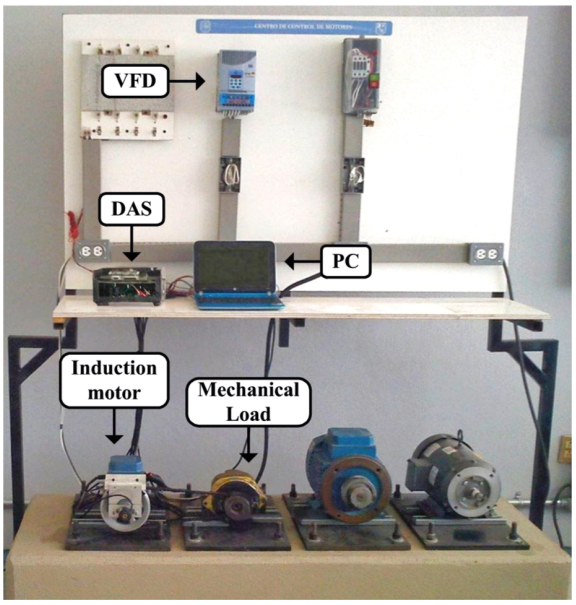


Figure 1.1: Experimental test bench used to validate the proposed condition monitoring method.

220V AC, and its full load current was 3.4 amperes. A variable frequency driver (VFD) (model WEG CFW08) was used to feed and drive the rotational speed of the IM. The automotive alternator was therefore driven by the IM through mechanical coupling based on a pulley-belt system. The alternator was used as the mechanical load and functioned, during the experiments, at three different load capacities: unloaded, half-load and full-load. The experimental test bench is shown in Figure 1.1.

Different physical magnitudes were acquired during the experiments. The effect of mechanical vibrations was recorded by a triaxial accelerometer, model LIS3L02AS4, fixed on top of the IM. Additionally, the voltage and the stator current consumption were continuously monitored and acquired by means of a set of Hall-effect sensors, model L08P050D15, from Tamura corporation. Thus, there were two groups of measurements: 7 related to currents and voltages; voltage A, voltage B, voltage C, current A, current B, current C, current N; and 3 relating to the measurement or vibrations; accelerometer X, accelerometer Y, accelerometer Z. Additionally, there was a last measure, the IM rotational speed that was acquired through a digital encoder. To give an idea of the nature of the signals used for the prediction of failures, Figure 1.2 shows the graphic representation of the

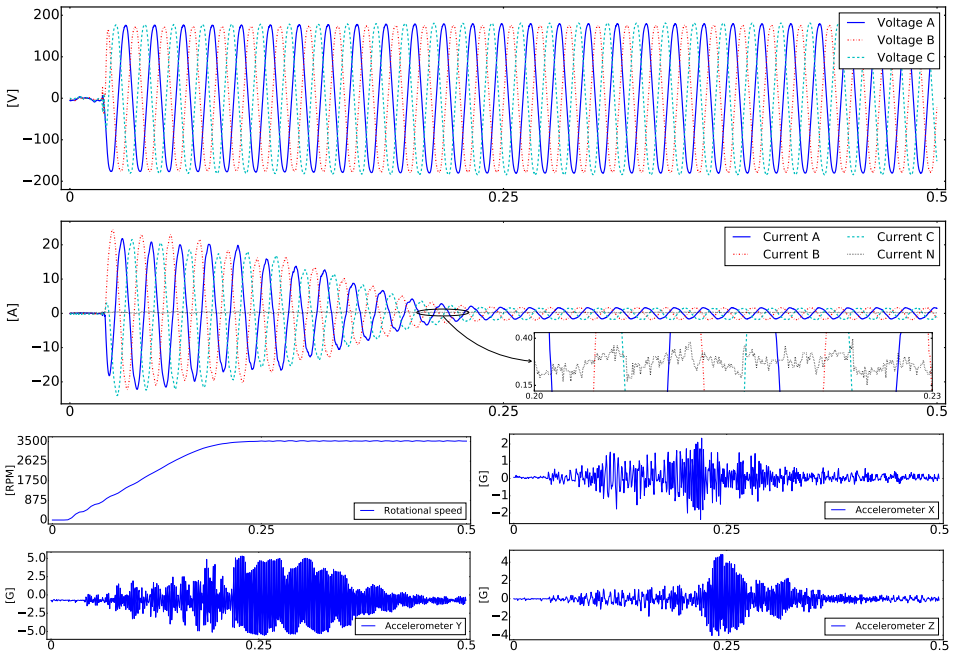
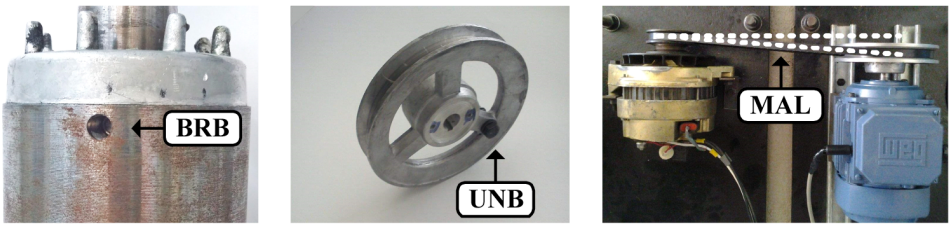


Figure 1.2: First half second of sensor measurements used for predicting faults. On the X-axis, the time in seconds. The Y-axis refers to the sensor value (different scale for each one). The figure shows the measurements of a healthy engine operating at direct supply.

signals (currents, voltages, speed, and accelerometers) during the first half second of an engine in good condition operating at direct supply.

The installed sensors were individually mounted on a board with its corresponding signal conditioning and anti-alias filtering. The data acquisition was performed by means of two 12-bit 4-channel serial-output sampling analog-to-digital converters, model ADS7841, from Texas Instruments, which were used as the Data Acquisition System (DAS); a proprietary low-cost DAS design based on a field programmable gate array technology.

Four different operating conditions were considered for evaluation in this study: healthy (HLT), broken rotor bar (BRB), unbalanced (UNB), misalignment (MAL), as well as their combinations. The conditions under consideration were produced artificially, thus, the BRB condition was produced by drilling a rough-hole of 2 mm diameter, at a depth of 14 mm, into a bar of the rotor, without harming the rotor shaft, in such a way that the hole was drilled through the complete section of the rotor bar. The UNB condition



(a) (b) (c)

Figure 1.3: Set of experimental faults: a) broken rotor bar (BRB); b) unbalance (UNB); and, c) misalignment (MAL).

was reproduced with a bolt in the rotor pulley that provoked a non-uniform load distribution that imbalanced the center of mass of the IM shaft. The MAL condition was caused by shifting the belt in the alternator pulley forward, thereby misaligning the transverse axial rotation of the motor and its load. Figures 1.3-a to Figure 1.3-c depict each of the conditions under consideration.

Moreover, the IM was driven at different operating frequencies during the experiments, so the data set contained the operating frequencies of 3 Hz, 30 Hz, and direct supply.

Each experiment was carried out during 10 seconds, the first 5 seconds corresponded to the IM start-up and were considered as a transient state. The next 5 seconds were considered as steady state (these last 5 seconds were not used in the experiments)[2]. The transient state was considered harder to predict, due to its non-stationary and inherently changing nature (Kim and Parlos, 2002). Nevertheless, as the experimental study addresses an early diagnosis problem, hereinafter only the transient state is considered.

In summary, an almost completely balanced data set was recorded of 2 521 instances: 831 for 3 Hz, 833 for 30 Hz, and 857 for direct supply. As there were eight measures sampled at 12 kHz and four measures sampled at 3 kHz, a single instance consisted of 540 000 attributes ($8 \times 12\,000 \times 5 + 4 \times 3\,000 \times 5$). As a preliminary pre-processing step, all the features were normalized to have values between 0 and 1.

[2] As the IM is powered by an inverter, the IM start-up was controlled, so that it always lasted 5 seconds, regardless of its frequency of use.

1.4.2 *Evaluation metrics for multi-label problems*

Experimental study

In single-label classification, accuracy and confusion matrix-based metrics are commonly used. Nevertheless, the performance of multi-label classification methods are more complex, because of the presence of several labels for each instance, and because the combined performance for each label can be approached in different ways. Hence, different evaluation metrics for multi-label classification and taxonomies have even been proposed to group those measures (Madjarov et al., 2012), although the simplest and most accepted division corresponds to measures based on either predictions or rankings. In the experimental evaluation, the measures in use in the first and the second groups were, accuracy, F_1 and Hamming loss and, in turn, one-error and rank loss, respectively. The definition of all these measurements are given in equations 1.1 to 1.6. The up/down arrow besides the measurement label indicates whether the higher or the lower value is best.

$$\uparrow \text{Accuracy} = \frac{1}{n}\sum_{i=1}^{n}\frac{|\omega_i \cap \hat{\omega}_i|}{|\omega_i \cup \hat{\omega}_i|} \tag{1.1}$$

$$\uparrow \text{Macro F}_1 = \frac{1}{|\Omega|}\sum_{l\in\Omega} F_1(TP_l, FP_l, TN_l, FN_l) \tag{1.2}$$

$$\uparrow \text{Micro F}_1 = F_1\left(\sum_{l\in\Omega} TP_l, \sum_{l\in\Omega} FP_l, \sum_{l\in\Omega} TN_l, \sum_{l\in\Omega} FN_l\right) \tag{1.3}$$

$$\downarrow \mathcal{H}\ \text{Loss} = \frac{1}{n}\sum_{i=1}^{n}\frac{|\omega_i \vartriangle \hat{\omega}_i|}{|\Omega|} \tag{1.4}$$

$$\downarrow \text{One Error} = \frac{1}{n}\sum_{i=1}^{n}\delta\big(\arg\min_{l\in\Omega} r_i(l)\big), \quad \delta(l) = \begin{cases} 1 & \text{if } l \notin \omega_i \\ 0 & \text{otherwise} \end{cases} \tag{1.5}$$

$$\downarrow \mathcal{R}\ \text{Loss} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|\omega_i||\bar{\omega}_i|}\Big|(l_q, l_r) : r_i(l_q) > r_i(l_r), (l_q, l_r) \in \omega_i \times \bar{\omega}_i\Big| \tag{1.6}$$

The notation for the equations is as follows: n is the number of instances; Ω is the set of labels; $\omega_i \subseteq \Omega$ is the actual labelset of instance $\mathbf{x}_i$; $\hat{\omega}_i$ is the predicted labelset of instance $\mathbf{x}_i$; and $\bar{\omega}_i$ is the complementary labelset of ω_i. Then, $r_i(l_j)$ is the predicted rank of class label l_j for instance $\mathbf{x}_i$ and $\triangle$ is the symmetric difference between two labelsets. Finally, *TP*, *TN*, *FP*, and *FN*, are the number of true positives, true negatives, false positives, and false negatives, respectively. All of these, *TP*, *FP*, *TN*, and *FN*, are computed by comparing the predicted and the actual labelsets.

1.4.3 *Determining the number of principal components*

As explained in Subsection 1.3.1, the number of the selected principal components is essential, if accurate models are subsequently to be trained. Two classical approaches were used to determine that number: the K1 rule and the Scree test. Parallel analysis (Horn, 1965) was also used, as it is more reliable (Zwick and Velicer, 1986), although the computational resources it requires can be prohibitive, due to the high dimensionality of the data set.

The models trained with the number of principal components selected with the K1 rule performed slightly better than those that used the scree test results. Also, for all sensors except those monitoring voltages, the number of retained components was lower following the K1 rule, meaning that a higher dimensionality reduction had occurred. Although Kaiser's rule states that only those components with an eigenvalue greater than 1 should be retained, whenever there was no eigenvalue greater than 1, then at least the first principal component was selected in this study, as it would be of no interest to either discard a sensor or completely disregard any measurement.

As described in the Section 1.5, the proposed predictive model is a combination of the results of applying twelve PCA transformations (one for each input sensor) and a multi-label classifier. It means that the model has twelve input parameters, which correspond to the number of principal components that will be retained for each signal. Therefore, in this scenario, the determination of the optimal value for those parameters, can be treated as a hyperparameter optimization problem (Bergstra et al., 2011). When optimizing hyperparameters, a set of parameter combinations is defined, then the model performance is evaluated for each combination. Thus, in this particular case, good model evaluations will indicate an optimal number of principal components to be retained.

Considering the K1 rule and the scree test results, 50 was selected as the maximum number of principal components to retain, and 1 as the minimum. 12 parameters with values between 1 and 50 will produce a set of 2.44×10^{20} combinations of parameters. Any evaluation of so many models is extremely costly in terms of time and computation effort, so the Hyperopt (Bergstra et al., 2015) library was used for the hyperparameter tuning task, which uses only two algorithms to optimize the parameter search space. In this study, the probabilistic method called the Tree of Parzen Estimators (TPE) was used (Bergstra et al., 2011), rather than the only other possible alternative, Random Search. Optimization of the hyperparameters also implies the minimization of an objective function, which in this study, was the inverse of model accuracy (1 − accuracy). The maximum number of evaluations was set to 500, which leads to 150 000 models evaluated in total (500 parameter combinations × 3 frequencies × 10 repetitions of 10-fold cross-validation).

Multi-fault early diagnosis method

Table 1.1 shows the number of principal components selected according to the different methods (K1 rule, Scree test, and Hyperopt) and frequencies. In terms of model accuracy, the number of principal components selected by Hyperopt produced significantly better results than those selected by the Scree and the K1 methods. Furthermore, using the combination obtained for 3 Hz with the other frequencies provided better results than the specific combinations obtained for each frequency. So, according to these results, and the fact that the 3 Hz combination retained fewer principal components (139), i.e., greater dimensionality reduction was achieved, the above combinations were chosen as the optimal settings for all the subsequent predictive models.

1.5 MULTI-FAULT EARLY DIAGNOSIS METHOD

Since typical problems of this nature are usually solved using time or frequency-domain-based methods, one of their main disadvantages is that expert knowledge on each failure type is required. However, this knowledge is not necessary when artificial intelligence-based approaches are used. These techniques, once trained, are capable of extracting the patterns that are most strongly associated with each fault.

One of the main characteristics of sensor data is the dependent nature of the observations that are highly correlated, because the data consist of a set of adjacent points in time. A common approach for dealing with data of this nature is to use statistical methods such as time series analysis (Shumway

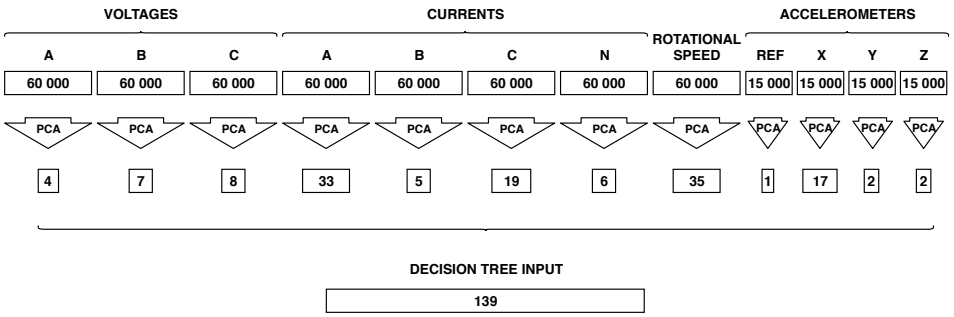
Freq.	Method	Voltages			Currents				Rotat.	Accelerometers				Sum.
		A	B	C	A	B	C	N	Speed	Ref.	X	Y	Z	PCs
3 Hz	K1 rule	26	28	30	14	15	14	1	1	1	1	4	1	136
	Scree test	17	18	18	18	18	17	5	2	11	43	189	105	461
	Hyperopt	4	7	8	33	5	19	6	35	1	17	2	2	139
30 Hz	K1 rule	18	18	18	14	14	14	1	1	1	1	9	1	110
	Scree test	15	18	18	16	17	17	3	1	9	83	171	151	519
	Hyperopt	32	8	4	24	4	22	35	49	3	4	4	22	211
Direct supply	K1 rule	17	19	18	12	15	15	1	1	1	1	31	1	132
	Scree test	16	16	16	18	19	19	1	2	12	68	210	176	573
	Hyperopt	6	25	46	2	10	8	2	10	16	38	33	17	213

Table 1.1: Number of principal components selected by the different methods and frequencies. The last column shows the sum of the number of principal components that each method retained, i.e., the number of decision tree inputs.

and Stoffer, 2017). Most statistical methods require the data to be stationary, which is not the case in this study, because data from the IM start-up (transient data) were used.

As stated earlier, in our data set each instance consists of 540 000 attributes. It is well-known that most Machine Learning algorithms suffer from difficulties when analyzing data sets with a large number of attributes, commonly known as the *curse of dimensionality* (Bellman, 2015). For this reason, PCA was used in this proposal to reduce the number of attributes. As previously explained, PCA is able to reduce the dimension of attributes, by retaining the combination of the most important features in which the maximum variance is retained as possible. Furthermore, when applying PCA to correlated data, one of the properties of the transformed data is that the correlation between attributes is removed (Wang et al., 2016). Although the robustness of decision trees leaves them unaffected by the collinearity of attributes, the elimination of the correlation can be beneficial for other Machine Learning methods.

Briefly, in the early multi-fault diagnosis method proposed in this paper, the information provided by the sensors was firstly reduced by means of PCA. Since each sensor measured a different physical magnitude, PCA was independently applied to each one. Then, all the retained principal components were joined as input to the multi-label classification method (a decision tree was used, although it could have been any other multi-label



Multi-fault early diagnosis method

Figure 1.4: Graphical representation of the dimensionality reduction approach by using PCA. The numbers of input measures correspond to the start-up of the IM (i.e., first 5 seconds of operation).

classifier). The output of the method is the predicted label set, where each label is a possible fault.

The graphical illustration of the process, with the number of principal components retained for each sensor measurement, is represented in Figure 1.4. A dramatic reduction in dimensionality can be observed, where 540 000 input attributes are converted into 139.

Although PCA has been used before for pre-processing single signals (Cadzow et al., 1983; Lowe, 1998; James and Lowe, 2001; Woon and Lowe, 2004), the novelty of the method presented here is the way in which multiple signals are combined after their dimensionality is reduced using PCA. The attributes that are passed to the classifier, a decision tree in our case, are the concatenation of the results obtained from PCA for each of the signals, that include voltages, currents and vibrations (see Figure 1.4).

The other novelty is that the decision tree is designed to solve multi-label problems, so it is capable of predicting potential faults, even when several are happening at the same time. The evaluation of the model is therefore done using well established measures in the field of multi-label classification. All other previous studies have used conventional single-label prediction methods, even when the fault-detection problem could occur simultaneously.

Despite the simplicity of the approach, the method obtained is robust under varying frequencies and load conditions, is able to detect errors swiftly in the first few seconds, even when there are simultaneous failures and the motor is still in the transition state (when other methods cannot be applied).

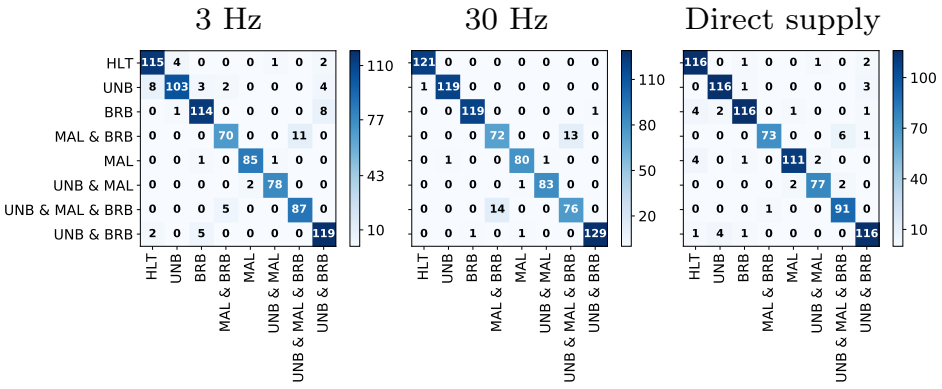


Figure 1.5: Confusion matrices of the three predictive models (one for each frequency). The X-axis corresponds to the predicted fault condition, while the Y-axis represents the actual fault condition.

1.6 RESULTS AND DISCUSSION

In this section a thorough experimental program will be reported for assessing the performance of the proposed method. First, in Subsection 1.6.1, the training of the multi-label classifier trees and their performance tests at each isolated frequency (3 Hz, 30 Hz, and direct supply) will be detailed; second, in Subsection 1.6.2, the test results of the time interval reductions used for training the models will be described; third, in Subsection 1.6.3, the test results of the model using all the frequencies (frequency insensitivity) at the same time will be presented; in Subsection 1.6.4, the test results of another fault type -a Bearing Defect (BD)- will be discussed; and, finally, in Subsection 1.6.5, an analysis of the multi-fault data set with Fast Fourier Transform will be described.

1.6.1 *Full transient state results*

Having determined the optimal number of principal components for the proposed method, multi-label decision trees were trained and tested across each operational range.

In Figure 1.5 are shown the confusion matrices of each multi-label decision tree for each of the three frequencies. The predicted fault condition is represented on the X-axis while the actual fault condition is on the Y-axis.

The model efficacy is indicated by the high numbers present on the matrices' diagonals. It should be noted that some combinations of failures

Time int. (s)	Voltages & currents	Rotational speed	Accelerometers	Total
0.01	840	120	120	1 080
0.02	1 680	240	240	2 160
0.05	4 200	600	600	5 400
0.10	8 400	1 200	1 200	10 800
0.20	16 800	2 400	2 400	21 600
0.50	42 000	6 000	6 000	54 000
0.75	63 000	9 000	9 000	81 000
1.00	84 000	12 000	12 000	108 000
2.00	168 000	24 000	24 000	216 000
5.00	420 000	60 000	60 000	540 000

Table 1.2: Total number of sensors measurements within each time interval.

(such as MAL & BRB) were under-represented and are therefore in a lighter color. According to tables 1.3- 1.5, the frequency-specific percentage accuracies are: 93.6% for 3 Hz, 95.5% for 30 Hz, and 95.8% for direct supply.

Although accurate fault detection can be seen at all frequencies, the accuracy at 3 Hz is particularly noteworthy due to the accepted difficulty of detecting faults at this frequency. Good BRB detection rates are also noteworthy, with a 95% percentage accuracy from Figure 1.5, a type of failure that is difficult to detect (Hassan et al., 2018) using other techniques.

1.6.2 *Reducing the time interval*

The method still captured the faults within the first five seconds of the IM start-up, despite the non-stationarity of the initial transient dynamics (Kim and Parlos, 2002). Therefore we also investigated the response capability in the first 0.5 seconds.

Experiments were executed with different time intervals (from 0.01 seconds to 5). Table 1.2 depicts the number of sensor measurements at each time interval. As might be expected, the increase in the number of features was proportional to the size of the time interval.

A classifier was trained and tested with each time interval. Tables 1.3, 1.4, and 1.5 group these results for each of the frequencies under analysis.

As expected, the best results were obtained with the longest time interval (5 seconds), i.e., with the data of the whole experiment. Average ranks were computed and the Hochberg post-hoc procedure (Hochberg, 1988) was applied, to determine the minimum time interval to use. Figure 1.6

Time int. (s)	Macro F_1 ↑	Micro F_1 ↑	Accuracy ↑	$\mathcal{R}$ Loss ↓	$\mathcal{H}$ Loss ↓	One Error ↓
0.01	0.875	0.874	0.749	0.137	0.121	0.251
0.02	0.895	0.892	0.781	0.123	0.103	0.219
0.05	0.917	0.917	0.835	0.109	0.078	0.165
0.10	0.931	0.931	0.869	0.096	0.065	0.131
0.20	0.949	0.948	0.895	0.075	0.049	0.105
0.50	0.954	0.954	0.912	0.066	0.043	0.089
0.75	0.955	0.955	0.917	0.065	0.043	0.083
1.00	0.954	0.954	0.918	0.063	0.044	0.082
2.00	0.970	0.970	0.946	0.043	0.029	0.054
5.00	0.975	0.974	0.936	0.038	0.025	0.064

Table 1.3: Model results for 3 Hz by the time interval in use.

Time int. (s)	Macro F_1 ↑	Micro F_1 ↑	Accuracy ↑	$\mathcal{R}$ Loss ↓	$\mathcal{H}$ Loss ↓	One Error ↓
0.01	0.843	0.848	0.722	0.200	0.144	0.278
0.02	0.880	0.884	0.785	0.171	0.111	0.216
0.05	0.873	0.878	0.775	0.180	0.116	0.225
0.10	0.914	0.917	0.857	0.116	0.079	0.143
0.20	0.911	0.914	0.851	0.119	0.082	0.149
0.50	0.975	0.975	0.946	0.035	0.024	0.055
0.75	0.976	0.975	0.942	0.033	0.024	0.058
1.00	0.977	0.977	0.944	0.030	0.022	0.056
2.00	0.982	0.980	0.952	0.021	0.019	0.048
5.00	0.984	0.983	0.955	0.026	0.016	0.045

Table 1.4: Model results for 30 Hz by the time interval in use.

Time int. (s)	Macro F_1 ↑	Micro F_1 ↑	Accuracy ↑	$\mathcal{R}$ Loss ↓	$\mathcal{H}$ Loss ↓	One Error ↓
0.01	0.853	0.855	0.698	0.174	0.137	0.302
0.02	0.888	0.890	0.762	0.158	0.103	0.238
0.05	0.909	0.911	0.823	0.111	0.083	0.177
0.10	0.923	0.924	0.836	0.103	0.071	0.164
0.20	0.932	0.932	0.856	0.088	0.063	0.145
0.50	0.955	0.954	0.902	0.062	0.043	0.098
0.75	0.960	0.960	0.920	0.054	0.037	0.080
1.00	0.963	0.964	0.927	0.042	0.034	0.073
2.00	0.964	0.965	0.929	0.046	0.033	0.071
5.00	0.982	0.982	0.958	0.027	0.017	0.042

Table 1.5: Model results for direct supply by the time interval in use.

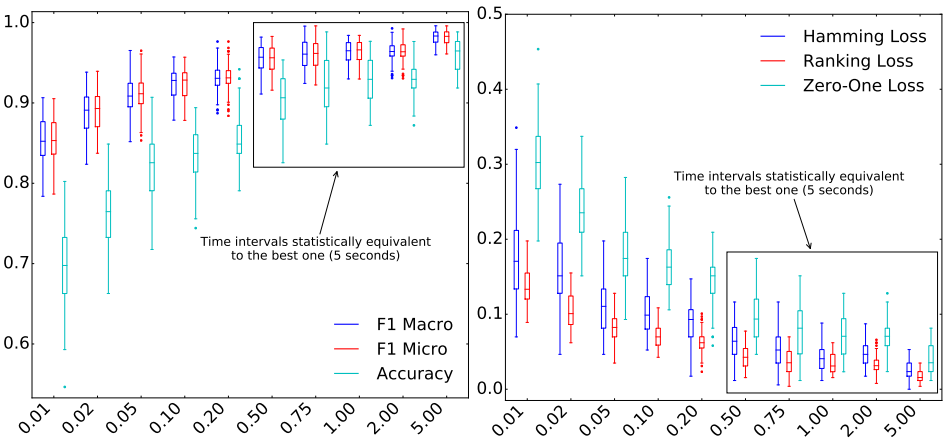


Figure 1.6: Multi-label evaluation metrics of a classifier trained with signals from the first 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 0.75, 1, 2, and 5 seconds. The X-axis depicts the time interval in seconds, while the Y-axis refers to the metric value in range $[0 - 1]$. The plotted results show the direct supply of the frequency. The results statistically equivalent to the use of the signals of the first 5 seconds are highlighted within a square.

provides a graphical representation of the data in Table 1.5. The results highlighted within the square are for the time intervals that were statistically equivalent (using Hochberg procedure) to the best one (i.e., 5 seconds). According to all multi-label metrics, the results of using the signals corresponding to the first 0.5, 0.75, 1, and 2 seconds, were statistically equivalent to the results of using the signals for the first 5 seconds. Only the results for time intervals shorter than 0.5 seconds were significantly worse.

In this regard, in order to quantify the performance reduction from the information constriction of short time intervals, the multi-label approach was investigated on the first 0.5 seconds. Figure 1.7 shows the confusion matrices of the models trained for the three frequencies. The predicted fault condition is represented on the X-axis while the actual fault condition is on the Y-axis.

Despite the constricted transient information within just the first 0.5 seconds, the multi-label decision tree still provides fault detection accuracies over 90%: 91.1% at 3 Hz, 94.5% at 30 Hz and 90.2% for direct supply.

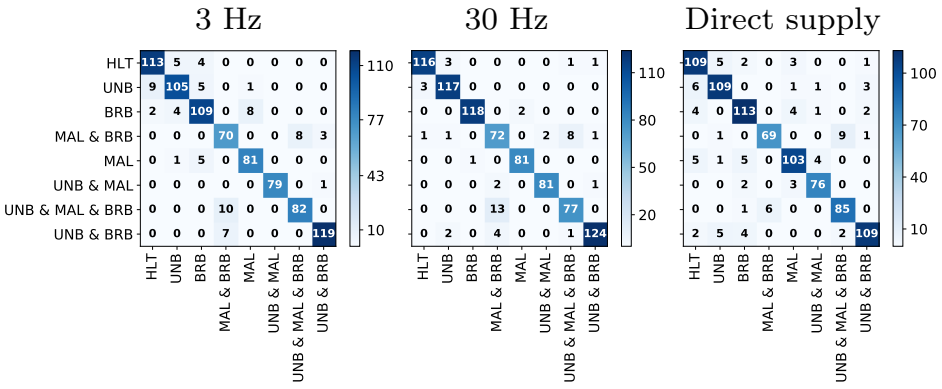


Figure 1.7: Confusion matrices of the three predictive models trained with 0.5 seconds (one for each frequency). The X-axis corresponds to the predicted fault condition, while the Y-axis represents the actual fault condition.

Early and extremely early multi-label fault diagnosis in induction motors

1.6.3 *Insensitivity to IM operating frequency*

A model that is insensitive to the operating frequency of an IM might seem useless, because the frequency is commonly known. However, training a separate model for each operating frequency might not be feasible in some scenarios. Our method can also be trained and deal with data acquired from IMs operating at different frequencies and the resulting model will still be able to classify fault conditions accurately. This is because the transformation of sensor measurements by means of PCA yields features that highlight the fault conditions, so multi-label decision trees can learn to classify them, regardless of the operating frequency.

Multiple operation frequency-insensitive models were trained with all of the data-set examples. Table 1.6 shows the results obtained for each time interval size. As expected, and consistent with Section 1.6.2, the best fault detection accuracy of 91.6% occurs for the longest time interval. Notably, the detection accuracy only dropped to 87.6% for time intervals of 0.5 seconds. Operation frequency-insensitive models therefore also appear appropriate for transient start-up time fault detection.

1.6.4 *Testing another type of fault: bearing defect*

Bearing defects (BDs) are common IM faults widely studied in the literature (Hui et al., 2017; Rauber et al., 2014; Li et al., 2016; Prieto et al., 2012; Cae-

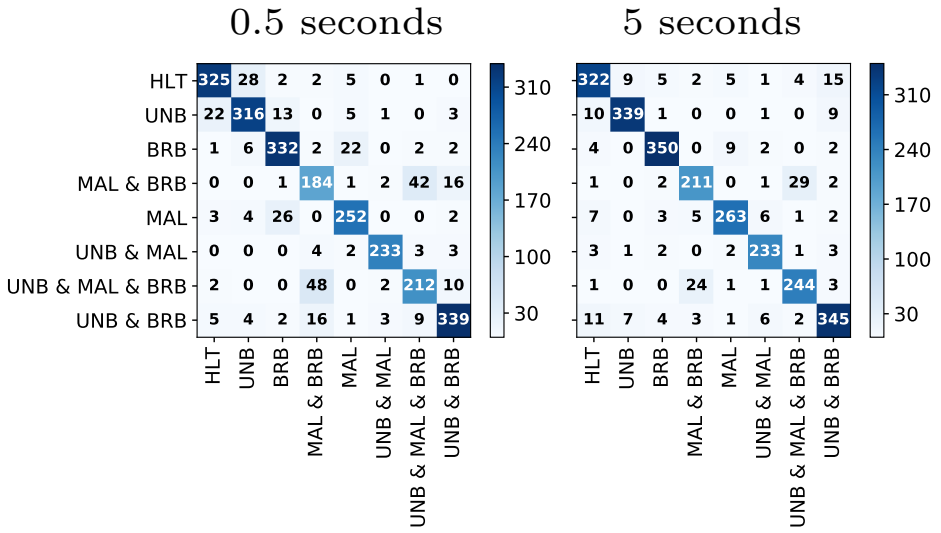


Figure 1.8: Confusion matrices of the general predictive models trained with all examples of all operating frequencies. The matrix on the left corresponds to first 0.5 seconds of IM operation, while the one on the right corresponds to the first 5 seconds (full transient state). The X-axis corresponds to the predicted fault condition, while the Y-axis represents the actual fault condition.

Time int. (s)	Macro F_1 ↑	Micro F_1 ↑	Accuracy ↑	$\mathcal{R}$ Loss ↓	$\mathcal{H}$ Loss ↓	One Error ↓
0.01	0.866	0.868	0.749	0.183	0.123	0.251
0.02	0.875	0.876	0.761	0.174	0.116	0.239
0.05	0.895	0.896	0.800	0.143	0.098	0.200
0.10	0.902	0.903	0.810	0.128	0.091	0.190
0.20	0.911	0.911	0.830	0.122	0.084	0.170
0.50	0.937	0.938	0.876	0.085	0.059	0.124
0.75	0.945	0.946	0.891	0.081	0.051	0.109
1.00	0.932	0.932	0.865	0.088	0.064	0.135
2.00	0.958	0.958	0.908	0.056	0.040	0.092
5.00	0.960	0.959	0.916	0.050	0.039	0.084

Table 1.6: Model results for all frequencies depending on the time interval in use.

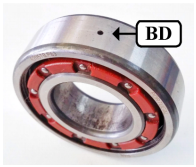


Figure 1.9: Induced Bearing Defect (BD) fault for the experimentation.

Time int. (s)	3 Hz	30 Hz	Direct supply	All frequencies
0.5	95.9%	97.0%	98.7%	96.3%
5	95.1%	100.0%	97.0%	96.0%

Table 1.7: Classification results in terms of accuracy for Bearing Defect (BD) fault condition.

sarendra and Tjahjowidodo, 2017). As a defective bearing is worn down, it will produce a complete mechanical breakdown. This makes almost impossible to experimentally reproduce this fault together with others on a test bench. In other words, although this kind of fault (as any other) can theoretically be predicted in combination with other faults, in practice it is almost impossible to obtain a data set with BDs and other faults occurring at the same time.

In this study, properly functioning examples and BD examples were employed, to prove that a model trained using our approach could also learn to predict these types of faults to high levels of accuracy.

BD examples were artificially provoked by drilling a through-hole on the outer race with a 1.191 mm tungsten drill bit, as shown in Figure 1.9. A total of 724 examples were used for testing the classification of a fault due to a BD. Of these, 363 corresponded to a healthy condition, and the remaining 361 to the BD fault condition. Isolating each frequency, 242 (122 HLT, 120 BD) were obtained at 3 Hz, 241 (121 HLT, 120 BD) were obtained at 30 Hz, and 241 (120 HLT, 121 BD) were obtained at direct supply. The type and number of measurements are the same as previously described in Subsection 1.4.1. The results, in terms of accuracy, are reported in Table 1.7.

A graphical representation of the classification results, using confusion matrices, is shown in Figure 1.10, where the top row refers to the models using 0.5 seconds of data, and the bottom row refers to the models using the full transient state (5 seconds of data).

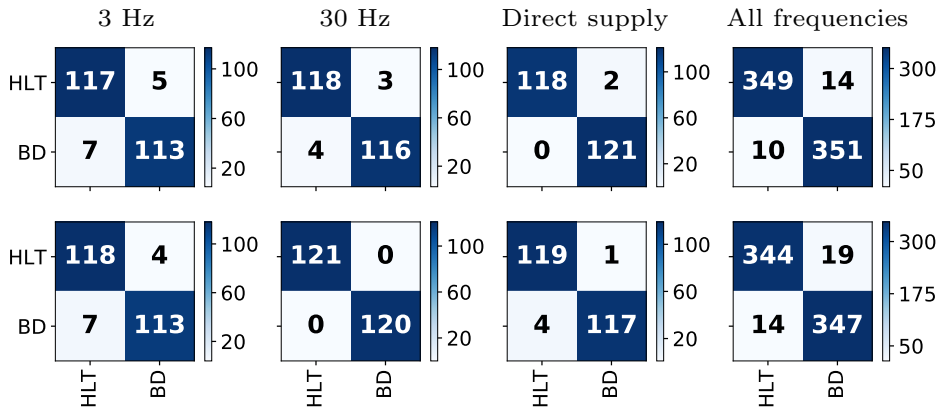


Figure 1.10: Confusion matrices of the four predictive models trained (one for each frequency and one operation frequency-insensitive). The X-axis corresponds to the predicted fault condition, while the Y-axis represents the actual fault condition. The matrices of the top row refers to the first 0.5 seconds of the IM operation, while the bottom row refers to the first 5 seconds (full transient state).

1.6.5 *Weakness of FFT when simultaneous faults occur*

On the other hand, as mentioned in Section 2, several works have been successfully proposed for detecting the occurrence of faults in induction motors. Indeed, in most of these proposals, the identification of faults is performed through classical frequency domain analysis (i.e., Fourier Transform applied to acquired vibration signals or stator current signatures). In this regard, and although the occurrence of faulty conditions in electric machines may be detected by means of calculating specific fault-related features (i.e., fault-related frequency features following classical motor current signature analysis—MCSA), the identification of simultaneous or multiple faulty conditions represent a major issue. Under this statement and considering the types of failures evaluated in this study, the unbalance (UNB) and misalignment (MAL) conditions are special scenarios that may produce similar effects on classical physical magnitudes that are acquired during the continuous monitoring of the rotatory electrical machine, such as vibrations or stator current signals.

Thereby, by analyzing the theoretical effects produced by the UNB and MAL conditions, both scenarios produce effects that are hard to distinguish. In the case of UNB condition due to a centrifugal force that generates a high

vibration amplitude equal to $1 \times$ RPM ($1\times$ rotational speed). On the other hand, the MAL condition causes high radial and/or axial vibrations that usually produces their dominant frequency components at $2 \times$ RPM, but also at $1 \times$ RPM (Tsypkin, 2017). Consequently, both faulty conditions tend to generate similar fault-related frequency components over an estimated stator current spectrum. Indeed, those components have normally been estimated with the characteristic frequency component for the UNB condition as follows:

$$f_{ecc} = f_s \left[1 + -k \left(\frac{1-s}{p} \right) \right] = f_s + -kf_r \tag{1.7}$$

where, f_s is the electrical supply frequency, f_r is the rotational frequency, s is the per-unit slip, p is the number of pole pairs, and $k = 1, 2, 3 \ldots$. If the number of pole pairs is equal to one, these equations are reduced to its simplified form, as presented.

Likewise, the characteristic fault-related frequencies associated with the MAL condition are estimated as follows (Carlos Verucchi and Acosta, 2016):

$$f_{sb} = f_s + -kf_r \tag{1.8}$$

Therefore, when performing the theoretical analysis of the effects that these faults may produce in relation to the corresponding stator current signal, it is concluded that both faulty conditions may appear masked or overlapping each other. This makes it difficult to identify and differentiate failures during condition assessment and, in addition, reduces the reliability of the condition monitoring strategy. In this regard, in order to experimentally demonstrate that the occurrence (isolated or simultaneous) of these faulty conditions, UNB and MAL, produces similar effects and that their fault-related frequency components appear overlapped, the stator current signals of the considered experimental test bench are analyzed.

Accordingly, Figure 1.11 shows the frequency spectra obtained by means of applying the FFT technique to the acquired stator current signature when the supply frequency, to feed the IM, was set at 30 Hz and direct supply (60 Hz), respectively. For both supply frequencies, the stator current spectra is shown around the closest to $f_s + f_r$, respectively.

Moreover, from these obtained spectra, it should be highlighted that the $f_s + f_r$ components of the UNB, MAL and UNB+MAL conditions appear at the same place with slightly different amplitudes. On the other side, despite the slightly differences between the amplitudes of each frequency component, these are not big enough as to identify the machine condition because

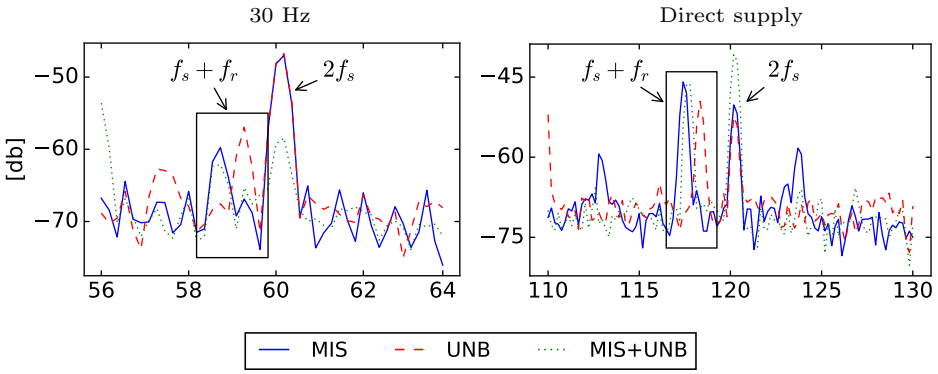


Figure 1.11: Conventional condition monitoring assessment based on the stator current spectrum at the closest region to $f_s + f_r$ when the supply frequency is set to 30 Hz (left) and direct supply (right). The X-axis depicts the frequency in Hz, while the Y-axis refers to the amplitude in db.

the amplitude of these frequency components will change depending on the severity of the individual faults. Hence, the number of false negatives will be higher and the performance of the fault identification will be affected. Summing up, through this experimental validation, it has been shown that classical condition monitoring strategies based on frequency domain analysis by means of the FFT have critical disadvantages when dealing with the identification of multiple fault sources that appear simultaneously, such as the UNB and MAL conditions in rotating machines like IM.

1.7 CONCLUSIONS AND FUTURE LINES

The problem of motor fault diagnosis has normally been addressed by using signal processing based on time domain, frequency domain, and time-frequency domain analyses. Nevertheless, the large data volumes collected by sensors for monitoring the motor operations are a source of information for the study of new multi-fault identification techniques, for faster and more efficient fault identification.

In this work, a new technique based on multiple sensor information for early diagnosis of single, combined, and simultaneous faulty conditions in IMs has been proposed, implemented and validated. In this regard, there exist several aspects that must be highlighted; first, the proposed model is able to perform with high-performance during the analysis of the transient

state because the PCA allows to retain a reduce number of significant features for each available physical measurement, that are vibrations, stator currents, voltages and rotational speed. Additionally, through the application of the PCA, the most representative information related to each evaluated condition is extracted without the need to use expert knowledge about each specific type failure. Secondly, despite the simplicity of the proposed approach, the consideration of PCA and multi label decision trees facilitates the identification and evaluation of multiple simultaneous failure conditions because the decision trees are specially adapted to this type of problems. Finally, the obtained results shown the effectiveness, robustness and capability for the extremely early detection with high-performance results even within the first 0.5 seconds of operation, during transient and steady state regimes. In this sense, it should be highlighted that the early fault diagnosis at low frequencies, such as 3 Hz, is a very challenging task and, the proposed method has demonstrated that achieves good results at different frequencies: 3 Hz, 30 Hz, and direct supply; and even, when variable load conditions are considered.

Therefore, those results are of special interest, because classical signal processing is not useful for light loads, and faults such as a broken rotor bar are very difficult to detect under these conditions (Hassan et al., 2018). Moreover, when the load at which the IM is working is known in advance, even better results could be achieved by training several models, one for each of the load conditions, and by selecting the right model as a function of the motor operating load. Although the best classification ratios were obtained from the models for each specific operating frequency, it has been demonstrated that the proposed method can also be trained using measures from variable operation frequency conditions without too much accuracy shrinkage. It may therefore be concluded that the proposed method is insensitive to the operational frequency.

With the aim of demonstrating that the proposed method is suitable for motors with other types of malfunctions, and additional test to diagnose a bearing-related fault has been conducted. Thus, it has demonstrated the capability of the proposed method to learn to diagnose BD fault with high accuracy rates. It may also be concluded that the method is capable of efficiently solving binary classification problems, although when our PCA and decision trees strategy was used with both multi-label and binary problems, the accuracy of the resultant models was greater for the multi-label problems. Therefore, it is the first time that a model applied to multi-fault diagnosis in induction motors has been proposed and evaluated using the

specific performance metrics for multi-label problems: accuracy, F1, Hamming loss, one-error, and rank loss.

As a future line of research, one promising option could be the use of supervised projections methods as alternatives to PCA, such as Linear Discriminant Analysis (LDA), which makes use of the labels to calculate the directions of data projection, and for which Wang et al. (Wang et al., 2010) have already presented an adaptation to the multi-label case.

Conclusions and future lines

Straightforward fault identification normally works by taking into account whether the fault is present or not. However, it is usually more useful to know when the fault is starting to occur. As a future line of research, our interest will be directed at the detection of fault severity or different magnitudes of malfunctioning.

ACKNOWLEDGMENTS

This work was supported through project TIN2015-67534-P (MINECO/-FEDER, UE) of the *Ministerio de Economía y Competitividad* of the Spanish Government, project BU085P17 (JCyL/FEDER, UE) of the *Junta de Castilla y León* (both projects co-financed through European Union FEDER funds), and by the *Consejería de Educación* of the *Junta de Castilla y León* and the European Social Fund through a pre-doctoral grant (EDU/1100/2017). The authors gratefully acknowledge the support of NVIDIA Corporation and its donation of the TITAN Xp GPUs used in this research.

ROTATION FOREST FOR BIG DATA

This journal paper presents a novel MapReduced adaptation of the well-known Rotation Forest algorithm for Big Data under the Apache Spark framework.

2

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

Type: Journal

Published in: Information Fusion, 74: 39–49 (JCR: Q1, SJR: Q1)

Year: 2021

Keywords: Rotation Forest, Random Forest, Ensemble learning, Machine learning, Big Data, Spark

Reference: Juez-Gil et al. (2021b)

ABSTRACT

The Rotation Forest classifier is a successful ensemble method for a wide variety of data mining applications. However, the way in which Rotation Forest transforms the feature space through PCA, although powerful, penalizes training and prediction times, making it unfeasible for Big Data. In this paper, a MapReduce Rotation Forest and its implementation under the Spark framework are presented. The proposed MapReduce Rotation Forest behaves in the same way as the standard Rotation Forest, training the base classifiers on a rotated space, but using a functional implementation of the rotation that enables its execution in Big Data frameworks. Experimental results are obtained using different cloud-based cluster configurations. Bayesian tests are used to validate the method against two ensembles for Big Data: Random Forest and PCARDE classifiers. Our proposal to compute PCA and to train the trees in parallel provides a scalable solution that achieves a competitive runtime, while retaining a remarkable performance level. In addition, extensive experimentation shows that by setting some parameters of the classifier (i.e., bootstrap sample size, number of trees, and

number of rotations), the execution time is reduced with no significant loss of performance using a small ensemble.

2.1 INTRODUCTION

Introduction

We are living in the era of Big Data where the growing sizes of some data sets have never been seen before (Bello-Orgaz et al., 2016; Chen et al., 2014). The way our world works is drastically changed by the influence of Big Data. Science, engineering, business, finances, sports, and healthcare are some fields where an important role is now played by Big Data processing techniques. Hence, the consolidation of Big Data as a research topic, that has been gaining special attention in academia, and as an essential process both for government and for industry.

From a theoretical point of view, we understand Big Data to mean high volumes of information that are processed at high velocity and in a wide variety of formats, most widely known as the three V's of Big Data (Luengo et al., 2020). Thus, designers of computer systems and data mining algorithms have therefore had to respond to the challenge (Moretti et al., 2008) through innovative high-performance processing solutions, such as computer clusters.

While proprietary cluster environments could be built, in order to arrive at the large-scale and complex computing requirements that are needed for processing these data sets, it is now claimed that the most powerful architecture to address Big Data is cloud computing. Cloud computing provides flexible and scalable resources on-demand that have intensive processing capacities to conduct data analysis tasks (Hashem et al., 2015). The flexibility and versatility of cloud computing produce economic solutions that rival other cluster architectures, which may even become obsolete within a short period of time.

In the past few years, many frameworks, programming models, and algorithms have emerged to deal with large data sets. In most cases, the main focus of these solutions is the exploitation of the parallelization opportunities provided by multi-core processors and cluster architectures. Unfortunately, the use of these solutions makes it necessary to redesign the algorithms to allow their execution in Big Data frameworks. During the last decade two main Big Data frameworks have gained notoriety: Hadoop and Spark; both base their performance in the MapReduce programming model (Dean and Ghemawat, 2008).

Likewise, ensemble methods have demonstrated their remarkable performance over the past few decades (Kuncheva, 2014). Their simplicity and flexibility make them useful in several domains. Moreover, their modularity (i.e., they are based on individual base classifiers or regressors) makes their parallelization feasible. Unfortunately, the number of ensemble algorithms available on Big Data frameworks is still limited. The aim of this paper is to present the MapReduce design and implementation of the well-known Rotation Forest ensemble (Rodriguez et al., 2006; Kuncheva and Rodríguez, 2007) and its evaluation. To assess whether the Rotation Forest ensemble maintains its good performance in Big Data, it was thoroughly compared against the few ensemble algorithms available for Big Data within a Spark cluster by using several large data sets with a number of instances ranging between 400 000 and 11 000 000, and a number of attributes ranging between 11 and 2 000.

Despite the remarkable performance of Rotation Forest, its main drawback is that it is a time-consuming algorithm. A Rotation Forest classifier needs to perform multiple PCA calculations as one of its steps, and rotate both the training and the testing data, making it slower than other ensemble methods. Nonetheless, an efficient design and implementation in Big Data frameworks can make it suitable for the new challenges posed by the need to process large data sets.

In the study presented in this paper, the original Rotation Forest algorithm was adapted to Big Data using the parallelization approach provided by Spark (i.e., using MapReduce). The parallelization was performed at critical points of the code (i.e., those with the largest execution times). Spark's parallel implementation of PCA and Random Forest were used, and the parallelization was also applied to the reordering of rotation matrices and to matrix multiplications, among other steps within the algorithm, making the proposal highly scalable.

The rest of the paper will be organized as follows. Works related to Big Data will be summarized in Section 2.2. In Section 2.3, the background to Random Forest and Rotation Forest ensembles, and MapReduce will be presented. In Section 2.4, the MapReduce implementation of the Rotation Forest algorithm will be explained. The experimentation will be presented in Section 2.5, and the conclusions and future research lines, in Section 2.6.

2.2 RELATED WORKS

Related works

Since Google laid the foundations of MapReduce programming model (Dean and Ghemawat, 2008), several frameworks have emerged for Big Data, such as Hadoop (Ghemawat et al., 2003) and Spark (Zaharia et al., 2012). As it is well known, Spark makes intensive use of memory rather than disk, which means it is faster and more convenient for data processing (Zaharia et al., 2010); an advantage that has made it increasingly popular for parallel computation. This section presents a brief review of algorithms and libraries for both Hadoop and Spark.

Spark MLlib is the Spark's Machine Learning library (Meng et al., 2016). It offers several algorithms for a broad variety of tasks, including classification, regression, and clustering, among others (Assefi et al., 2017). Nevertheless, the number of algorithms is still scarce in comparison with other Machine Learning tools such as Weka (Witten et al., 2011) and Scikit-learn (Buitinck et al., 2013).

Lazy learners, such as k-NN (k-Nearest Neighbors) (Cover and Hart, 1967; Fix and Hodges Jr, 1951), are successful methods that have demonstrated their value in several domains. Unfortunately, their high memory requirements, consequence of storing the entire data set, rather than calculating a model, makes their MapReduce implementation difficult. Despite this, there are recent Spark implementation proposals for both batch learning (Maillo et al., 2017) and online learning (Ramírez-Gallego et al., 2017).

Ensemble methods rely on the fact that a bunch of base learners (regressors or classifiers) are more likely to make proper predictions than the base methods alone (Kuncheva, 2014). One of the benefits of ensembles is that their construction can be easily parallelized. At the moment, one active research line within the area of Big Data focuses on adapting ensemble algorithms to the MapReduce paradigm. Albeit scarce, some implementations have already been proposed in recent years: In 2011, Tyree et al. proposed the pGBRT algorithm (Tyree et al., 2011), a MapReduce version of the Gradient Boosting Regression Trees ensemble. Later, in 2016 Chen et al. published a parallel Random Forest for the Apache Spark framework (Chen et al., 2017). In 2017 a set of ensemble implementations for multi-label learning on Apache Spark was proposed by Gonzalez-Lopez et al. (Gonzalez-Lopez et al., 2017). PCARDE, presented by García-Gil et al. (García-Gil et al., 2018), is another Big Data adaptation of an ensemble algorithm for Apache Spark; it is based on Principal Components Analysis and Random Discretization.

Another popular family of supervised classification models is Bayesian Networks (Friedman et al., 1997). The most expensive task of these methods is the computation of multidimensional contingency tables, which requires the estimation of probability distributions (Arias et al., 2017). A MapReduce version, implemented in Spark, for discrete Bayesian network classifiers was recently presented by Arias et al. (Arias et al., 2017).

Rotation Forest for Big Data

It is not only classifiers and regressors that are used in data mining, but also data preprocessing methods, which constitute a very important stage in the knowledge discovery process (García et al., 2016; Ramírez-Gallego et al., 2018b). This is what explains why many preprocessing techniques have also recently been adapted to Big Data frameworks (Luengo et al., 2020). There are several tasks related to data preprocessing, such as discretizers (Ramírez-Gallego et al., 2018b), noise filtering (García-Gil et al., 2019), feature selection (Ramírez-Gallego et al., 2018), and instance selection (Arnaiz-González et al., 2016; Arnaiz-González et al., 2017; García-Osorio et al., 2010), among others.

2.3 BACKGROUND

This section presents the background of the paper, focusing on Random and Rotation Forest ensemble algorithms and the MapReduce runtime environment.

2.3.1 *Random Forest*

Ensemble learning relies on the idea of generating several models (called base classifiers, in classification problems) instead of only one, looking for a combination of models that outperforms the individual performance of the models that are combined. The key to the success of this process is to achieve ensembles that, without damaging the average performance of the models, make them diverse (in the sense that their predictions are different from each other).

The Random Forest algorithm, proposed by Breiman (Breiman, 2001) in 2001, is still considered a state-of-the-art classifier (Fernández-Delgado et al., 2014). Its simply, yet effective, underlying idea is to produce a bootstrap sample (as bagging (Breiman, 1996) does) and it then randomly selects features at each node of the tree being constructed in order to increase the diversity of the ensemble. That is, instead of searching for the best feature (and threshold value for dividing the data), it only considers a random

subset of all the features. Its simplicity and effectiveness have made it an extremely popular ensemble method.

2.3.2 *Rotation Forest*

Background

The Rotation Forest ensemble algorithm was presented for classification in (Rodriguez et al., 2006). It relies on the inherent instability of the tree construction algorithms when the input space is rotated (Kuncheva, 2014). Data rotation, which is in fact the cornerstone of Rotation Forest, is performed internally, prior to training the base classifiers. Rather than an arbitrary rotation, the rotation is operated using Principal Component Analysis (PCA). Base classifiers (commonly trees) can therefore divide the decision space both parallel to the feature axes and in other directions after the rotation. This feature makes it much more powerful than other traditional ensemble techniques, especially when all the attributes have real values (Bagnall et al., 2018).

Since the standard Rotation Forest was presented, several variants have been proposed: Rotation Forests for regression (Pardo et al., 2013), and Boosting of Rotation Forests (Zhang and Zhang, 2008), among others.

2.3.3 *MapReduce*

The MapReduce runtime environment (Dean and Ghemawat, 2008) has become the most widely used paradigm in Big Data scenarios nowadays (Fernández et al., 2017; Ramírez-Gallego et al., 2018a). Since the release of Hadoop in 2006, the first open source implementation of MapReduce, some drawbacks have been identified. The main shortcoming of Hadoop is its disk usage to ensure cluster fault tolerance, making it slow, especially for iterative processes. Spark was therefore developed in 2010 in an attempt to use memory intensively and to minimize disk access.

Both, Hadoop and Spark use the MapReduce programming model. To take advantage of MapReduce, an algorithm must be redesigned so that it consists of two stages: Map and Reduce. The map phase divides the data into several parts and applies a first processing step to each of them, while the reduction phase is responsible for applying a second processing step in which the data from the first phase is collected and integrated to obtain the final result. A typical example is the task of counting words in a document. The sequential version processes the document word by word while storing the words and their number of occurrences in a dictionary or hash table. On the contrary, in the MapReduce version of the algorithm, after dividing

the document into several parts, the map step transforms every single word into a tuple of `(word, 1)`; then, the reduce step groups all the tuples with the same key (i.e., the word) and adds the values (the "ones"); and, finally, the process is repeated in a final reduction step that calculates the global word count for the entire document. The result is a collection of tuples with the word as the key and its number of occurrences as the value.

Rotation Forest for Big Data

Therefore, the implementation of a sequential algorithm into a parallel (MapReduce) environment requires the redesign of the algorithm itself, dividing its internal processing into mapping and reducing phases. This task is not always trivial and can in fact be a paramount challenge in some cases (García et al., 2016).

2.4 ROTATION FOREST FOR BIG DATA

This section presents the parallel implementation of the Rotation Forest classifier (Rodriguez et al., 2006) for Big Data. Our proposal maintains the main structure of the standard Rotation Forest, but adds some changes to face the problems that Big Data induces. The main difference is the paradigm shift (from sequential to functional) and the use of Random Forest as the base classifier instead of a single decision tree. This last idea was presented in (Stiglic et al., 2011) to show that rotation of Random Forests outperform most widely used ensembles. Here, we have found that this is also a very promising approach to optimize training times when using large data sets. An advantage of using Random Forest is the versatility that it offers: it can be parameterized to behave as a single decision tree, as a Bagging of decision trees, or as Bagging of random trees (i.e., Random Forest) among others.

Training a Rotation Forest is a very time-consuming task, which is its main drawback, at least in the context of Big Data. As noted, this is mainly because the PCA calculation requires more computing resources, which is greater as the number of instances and features increase. In Rotation Forest, data are transformed through a sparse rotation matrix computed by arranging K PCA rotation matrices (each of them calculated for K random feature subsets). Additionally, since Rotation Forest is an ensemble, as many sparse rotation matrices as the size of the ensemble will be computed (i.e., for an ensemble of size L, PCA is computed $L \times K$ times).

Nonetheless, this can be done in an efficient parallel way, as PCA can be solved using matrix algebra (i.e., singular value decomposition or covariance matrix calculations followed by an eigenvalue decomposition). In

the Spark framework, PCA is already implemented using a parallel singular value decomposition (SVD) algorithm. Decision tree-based algorithms, such as Random Forest, can also be parallelized in many ways. Spark provides its own parallel Random Forest implementation, whose optimizations are based on the PLANET project (Panda et al., 2009).

Rotation Forest for Big Data

Both PCA and Random Forest parallel implementations provided by Spark are used, in order to take advantage of Rotation Forest for Big Data processing.

The training stage of Rotation Forest is presented in Algorithm 1. The algorithm rotates the input data **X** and then trains a Random Forest of size T using the rotated data. The process is performed L times, in order to build an ensemble that is composed of L rotation matrices and L Random Forests. Hence, the total number of trees in the ensemble is $L \times T$.

The construction of the rotation matrix $\mathbf{R}^a$ is performed in lines 3 to 12. Initially, the feature set, **F**, of the input data, **X**, is randomly split into a partition, **Q**, of K subsets (K is a parameter of the algorithm). For each feature subset, **S**, of **Q**, a submatrix, **W**, is extracted from **X** that only contains the features in **S** (that is, **W** only contains a subset of columns of **X**). The matrix, **W**, is now further reduced by removing some of its rows, more specifically a random selection of classes is made and all instances not belonging to the selected classes are removed. The result is a new matrix, $\mathbf{W}'$. An additional reduction step generates the matrix, $\mathbf{W}''$, by retaining a bootstrap sample of a percentage, B, of the instances in $\mathbf{W}'$. PCA is applied to this last matrix, $\mathbf{W}''$, to obtain a rotation matrix, **C**.

In the reduction phase, all the resulting PCA rotation matrices, K, are arranged into a block diagonal matrix, **R**, although it cannot yet be used to rotate the original input data, **X**, because the order of its columns does not match the order of the corresponding features in the original data. In consequence, the columns of **R** have to be rearranged to match the original order of features using a permutation matrix, **P**, and a MapReduce implementation of matrix multiplication.

Finally, a Random Forest classifier is trained using the data obtained by rotating the input data, **X**, using the reorganized matrix, $\mathbf{R}^a$ ($\mathbf{X}\,\mathbf{R}^a$).

Figure 2.1 illustrates the process of calculating the rotation matrix $\mathbf{R}^a$ using as an example a data set **X** with 12 instances, 6 features, and 3 classes $\{0, 1, 2\}$; and algorithm parameters, K and B, equal to 3 and to 50%, respectively.

Algorithm 2 shows the steps for the Rotation Forest prediction stage. A novelty with regard to the standard Rotation Forest, is that the prediction

Algorithm 1: Rotation Forest for Big Data (Training stage).

Input: A training set $(\mathbf{X}, \mathbf{Y})$ where $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ defined in a feature set $\mathbf{F}$, $\mathbf{Y} = \{y_1, \ldots, y_n\}$ with labels $y_i \in \Omega = \{\omega_1, \ldots, \omega_c\}$ representing c classes, number of rotations L, number of trees T, number of feature subsets K, bootstrap size B.

Output: Trained ensemble E (tuples: rotation matrix $\mathbf{R}^a$, base classifier D).

1 $E \leftarrow$ **map** $i \in \{1, \ldots, L\}$
2 $\mathbf{Q} \leftarrow$ random partition of $\mathbf{F}$ into K subsets of features
3 $\mathbf{M} \leftarrow$ **map** $\mathbf{S} \in \mathbf{Q}$
4 $\mathbf{W} \leftarrow$ submatrix of $\mathbf{X}$ with the columns corresponding to the features in $\mathbf{S}$
5 $\mathbf{Y}' \leftarrow$ random selection of classes in Ω
6 $\mathbf{W}' \leftarrow$ submatrix of $\mathbf{W}$ with rows corresponding to instances of classes in $\mathbf{Y}'$
7 $\mathbf{W}'' \leftarrow$ bootstrap sample of size B% of the number of instances in $\mathbf{W}'$
8 $\mathbf{C} \leftarrow$ rotation matrix from `parallel-PCA`$(\mathbf{W}'')$
9 **emit** $\langle \mathbf{C} \rangle$
10 $\mathbf{R} \leftarrow$ `reduce` $(\mathbf{M})$ `// block diagonal matrix`
11 $\mathbf{P} \leftarrow$ permutation matrix, matching the order of the features in $\mathbf{F}$
12 $\mathbf{R}^a \leftarrow \mathbf{P}\,\mathbf{R}$ `// rearrangement of` $\mathbf{R}$ `(in parallel)`
13 $D \leftarrow$ `train-parallel-random-forest`$(\mathbf{X}\,\mathbf{R}^a, \mathbf{Y}, T)$
14 **emit** $\langle \mathbf{R}^a, D \rangle$
15 **return** E

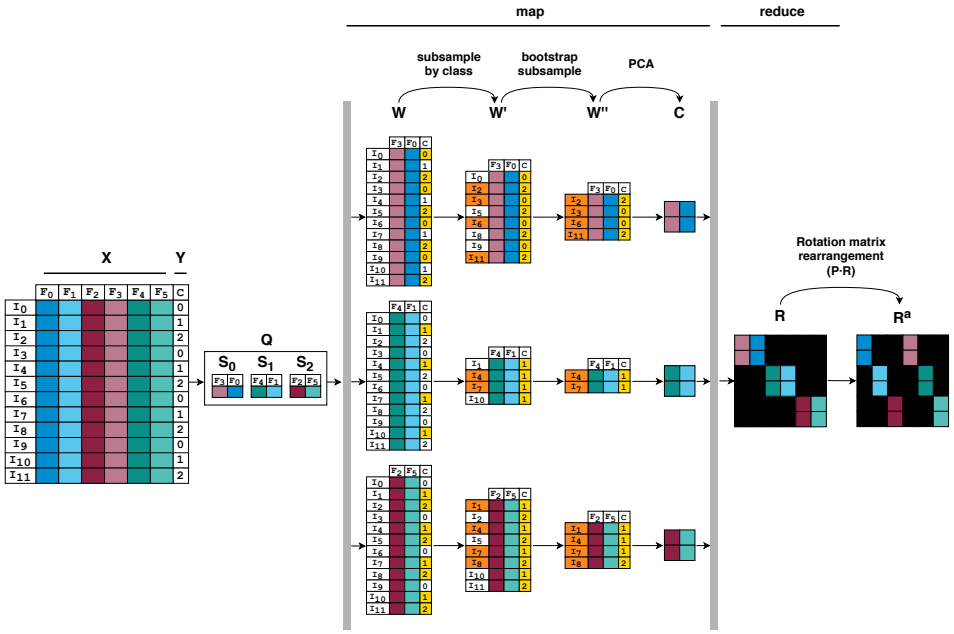


Figure 2.1: MapReduce implementation of the process to generate the rotation matrix ($\mathbf{R}^a$).

is not performed for a single instance at a time but for a set of instances in parallel, which is more convenient in Big Data. An ensemble of size L has L rotation matrices and L trained base classifiers (i.e., L Random Forest). Firstly, each rotation matrix, $\mathbf{R}^a$, and its corresponding base classifier, D, are used in the map phase (see Line 1), where the set of instances, $\mathbf{X}$, is rotated through a parallel matrix multiplication ($\mathbf{X}\,\mathbf{R}^a$), and then used as input to a base classifier D. Its output, $\mathbf{P}$, will consist of the predicted probabilities of each instance in the set for each class (i.e., the dimension of the matrix is $n \times c$, where c is the number of classes and n the number of instances for which the prediction is being made). All predictions (i.e., base classifier probabilities) are arranged in a three-dimensional matrix, $\mathbf{S}$, of size $L \times n \times c$. Then, the probabilities of each base classifier for each class for the instances are added up and averaged in the reduce phase, thus the $\mathbf{T}$ matrix is of size $n \times c$. Finally, the last map (see Line 6) computes the final prediction of the whole Rotation Forest ensemble. The predicted class for each instance (ω_j) will be the class with the highest probability in $\mathbf{u}$, and an array, $\mathbf{y}$, containing the predicted classes of the instances, will be the final output of the ensemble.

Algorithm 2: Rotation Forest for Big Data (Prediction stage).

Input: A set $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ defined in a feature set $\mathbf{F}$, the ensemble E (tuples: rotation matrix $\mathbf{R}^a$ and base classifier D).

Output: Predicted classes $\mathbf{y} = \{y_1, \ldots, y_n\}$ with labels $y_i \in \Omega = \{\omega_1, \ldots, \omega_c\}$ representing c classes

```
S ← map (R^a, D) ∈ E
    X' ← X R^a          // parallel rotation of input set
    P ← D(X')                    // parallel prediction
    emit ⟨P⟩
T ← reduce (S)              // average the probabilities
y ← map u ∈ T
    j ← arg max_{i∈{1,...,c}} u_i
    emit ⟨ω_j⟩
return y
```


2.5 EXPERIMENTAL RESULTS

The aim in this section is to assess whether the MapReduce version of Rotation Forest, presented in this paper, is suitable for Big Data processing. A thorough experimentation was performed taking into account both accuracy and execution time, using several representative data sets. Since the proposed Rotation Forest[1] is a tree-based ensemble classifier, it was compared to the official Spark implementation of Random Forest (Breiman, 2001). It was also compared to the Principal Components Analysis Random Discretization Ensemble (PCARDE) (García-Gil et al., 2018), because it shares the idea of using PCA with Rotation Forest as part of its data transformation step.

2.5.1 *Experimental framework*

Six popular classification data sets for Big Data were used for conducting the experiments[2]. Table 2.1 summarizes the data sets. All features of the

[1] The Rotation Forest classifier implementation for Spark is publicly available at https://github.com/mjuez/rotation-forest-spark.

[2] The poker-hand, covtype, susy, higgs, and hepmass data sets are available at the UCI Machine Learning repository (Dua and Graff, 2017) https://archive.ics.uci.edu/ml/index.php.

Dataset	Instances	Attributes	Classes	Size (GB)
poker-hand	1 025 010	11	10	0.02
covtype	581 012	54	7	0.07
susy	5 000 000	18	2	2.33
higgs	11 000 000	28	2	7.84
hepmass	10 500 000	28	2	7.58
epsilon	400 000	2000	2	11.87

Table 2.1: Experimental data sets.

data sets were normalized. As the features of Rotation Forest need to be numeric, nominal features were binarized using one-hot encoding.

The experimental setup of Rotation Forest was as follows. The number of trees, T, was set to 1, so the ensemble size was the same as the number of rotations, L.[3] Following (Rodriguez et al., 2006), we used three different ensemble sizes, to represent small (10 trees), medium (50 trees), and large ensembles (100 trees). The bootstrap value was set at 25% and 4 was selected as the number of feature subsets K. As stated before, the Rotation Forest classifier proposed in this study uses Random Forest as its base classifier. The Spark Random Forest was parameterized in such a way that it behaves as a decision tree (as the standard Rotation Forest does). It was achieved by setting featureSubsetStrategy to "all" (i.e., all features are used for training). For the rest of the parameters, their default values were used.

Regarding the experimental setup of Random Forest and PCARDE, the default values were used for all the parameters except for the number of trees, which was set to 100 for small, to 200 for medium, and to 500 for large ensembles. In the case of PCARDE, only a single ensemble size with 10 trees was used because this was the setting recommended by the authors (García-Gil et al., 2018) (they reported equivalent results for small, medium, and large ensembles).

Given that the values of the parameters were not changed for each individual data set, the results obtained were not entirely optimal. If the parameters had been individually tuned for each data set, then the results could

The epsilon data set is available at the LIBSVM data repository https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#epsilon

[3] This is what the standard Rotation Forest does, in which there is a different rotation for each tree, later we will experiment with other approach where the same rotation is shared among several trees.

have been improved. It would however have made the experimentation unnecessarily time-consuming. Nevertheless, the same restriction was used with all the methods, so the comparison was fair.

The experiments were carried out using 5-fold cross-validation. Ensemble learning requires randomization to train different and diverse base classifiers. For this reason, with the aim of enabling experimental repeatability, a random seed value was fixed at 46.

Accuracy measure (defined in Equation 2.1) was used for evaluating classification performance.

$$accuracy = \frac{\text{correctly classified instances}}{\text{total number of instances}} \tag{2.1}$$

Bayesian analysis (Benavoli et al., 2017) was used to compare the ensemble classifiers (the library to perform the analysis was *baycomp*[4]). The number of samples for all Bayesian comparisons was set to 50 000. The value for the region of practical equivalence (ROPE) was set to 0.01, which means that two algorithms with a difference in accuracy of less than 1% will be considered equivalent. Bayesian comparison results were represented through ternary plots (e.g., Figure 2.2) where the space is divided into three areas of interest: L (opponent wins), ROPE (both tie), and R (Rotation Forest wins).

The experimentation was performed in cloud-based clusters provided by the Google Cloud Platform. A cluster was composed of one master node and seven computing/worker nodes. All nodes were of the n1-standard-16 type, which had 16 virtual CPUs, and 60 GB of RAM. Hence, the cluster size was 128 vCPUs and 480 GB of RAM. At that point in time, the vCPUs of n1-standard nodes could be of one of the following types: Intel Xeon (Skylake), Intel Xeon E5 (Sandy Bridge), Intel Xeon E5 v2 (Ivy Bridge), Intel Xeon E5 v3 (Haswell), or Intel Xeon E5 v4 (Broadwell E5). We used Google Dataproc software, version 1.4, running on Debian 9, with Apache Hadoop 2.9.2, and Apache Spark 2.4.5. Google Cloud Storage was used as a distributed file system for storing data sets and experimental results.

2.5.2 *Accuracy performance*

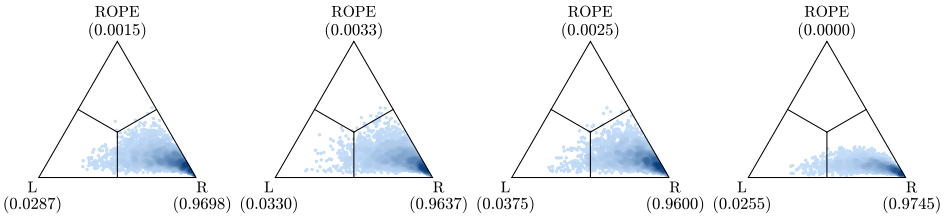
Table 2.2 gathers the accuracy results of the three ensemble methods: Random Forest (RanF), PCARDE, and Rotation Forest (RotF). The best results are highlighted in bold. As can be seen, for all data sets, the best results

[4]The baycomp library is publicly available at https://baycomp.readthedocs.io/en/latest/.

Dataset	Random Forest			PCARDE	Rotation Forest		
	100	200	500	10	10	50	100
poker-hand	51.42 ±0.40	51.32 ±0.30	51.18 ±0.31	50.77 ±1.36	**62.46** ±3.20	58.22 ±1.12	57.26 ±1.16
covtype	67.17 ±0.26	67.22 ±0.29	67.13 ±0.22	67.01 ±0.85	72.08 ±0.38	**72.33** ±0.16	72.28 ±0.14
susy	77.67 ±0.05	77.67 ±0.04	77.66 ±0.01	72.64 ±1.34	78.40 ±0.09	**78.59** ±0.06	78.59 ±0.06
higgs	67.52 ±0.31	67.58 ±0.28	67.67 ±0.11	58.33 ±1.66	68.16 ±0.33	68.70 ±0.10	**68.80** ±0.10
hepmass	82.21 ±0.08	82.15 ±0.12	82.19 ±0.07	81.33 ±0.33	84.06 ±0.32	**84.44** ±0.12	84.42 ±0.11
epsilon	72.56 ±0.31	72.69 ±0.33	73.02 ±0.35	78.40 ±0.13	77.19 ±0.91	80.12 ±0.44	**80.33** ±0.29

Experimental results

Table 2.2: Experimental results in terms of classification accuracy for Random Forest, PCARDE, and Rotation Forest. Best results are highlighted in bold. The value at the right of each accuracy corresponds to the standard deviation.



(a) RanF 100 (b) RanF 200 (c) RanF 500 (d) PCARDE

Figure 2.2: Hierarchical Bayesian test heatmap for Random Forest and PCARDE against Rotation Forest with 10 trees.

were achieved by Rotation Forest. The table also shows that the size of the ensemble for Random Forest has very little influence on the results. On the other hand, this influence appears to be somewhat greater in the case of the Rotation Forest. Although at first glance medium-size Rotation Forests appear to be a better option than small-size Rotation Forests, Section 2.5.4 will show that, from the point of view of Bayesian statistical analysis, Rotation Forest performs equivalently for all sizes.

Figure 2.2 shows the hierarchical Bayesian test in heatmap representations for Rotation Forest with 10 trees against Random Forest (Figure 2.2.a–c) and PCARDE (Figure 2.2.d). The R area in the triangles corresponds to Rotation Forest while the L area corresponds to the opponent (i.e., PCARDE or Random Forest). The high density of points (each point corresponds to one Bayesian simulation) concentrated in the right corners means that Rotation Forest clearly outperformed the opponents (i.e., R probability close to 1).

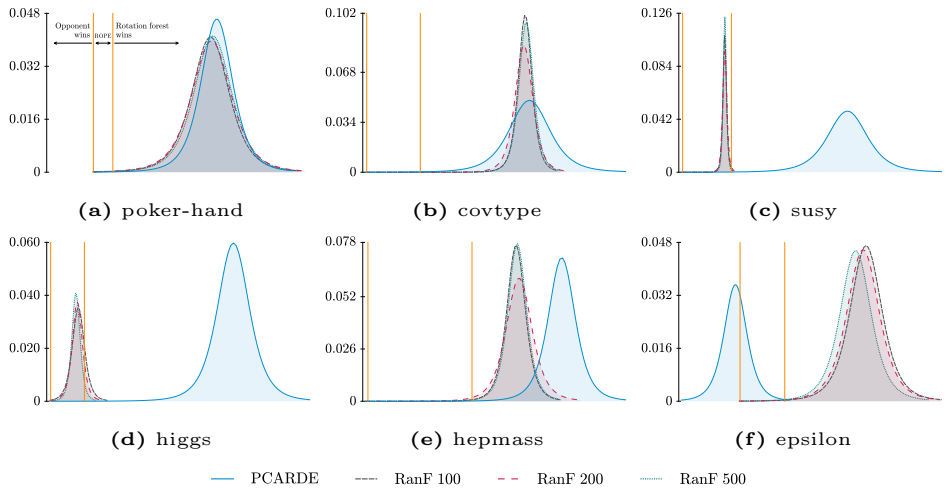


Figure 2.3: Bayesian correlated t test density plots. It compares Random Forest and PCARDE against Rotation Forest with 10 trees.

In the previous comparison, cross-validation accuracy results for all data sets were used to get a general overview of Rotation Forest performance compared to Random Forest and PCARDE. Cross-validation accuracy results for a single data set could also be analyzed through the Bayesian correlated t test (Corani and Benavoli, 2015). Figure 2.3 shows the density plots comparing Rotation Forest with 10 trees against PCARDE and Random Forest for each data set. As in ternary plots, there are three areas of interest: the region to the left of the leftmost line, which corresponds to the statistical primacy of the opponent; the region between the two lines, which corresponds to the ROPE (i.e., statistical equivalence between two classifiers); and the region to the right of the rightmost line, which corresponds to the statistical primacy of Rotation Forest. This test shows that, for five out of six data sets, Rotation Forest performed better than PCARDE. Only for epsilon was PCARDE better (blue density curve). In the comparison with Random Forest (gray, red, and green density curves), Rotation Forest performed better for four data sets, while Rotation and Random Forest performed equivalently for susy and higgs.

Table 2.3 gives another perspective on this information, showing the detailed probabilities for the three areas of interest of the Bayesian correlated t test explained earlier. The region (Left, Right, or ROPE) with the highest probability, is highlighted in bold. Most of the time, the highest

Dataset	Method	Ensemble size	Left prob.	ROPE	Right prob.
poker-hand	RanF	100	0.18 %	0.17 %	**99.65%**
		200	0.22 %	0.20 %	**99.58%**
		500	0.17 %	0.16 %	**99.67%**
	PCARDE	10	0.08 %	0.08 %	**99.84%**
covtype	RanF	100	0.00 %	0.01 %	**99.99%**
		200	0.00 %	0.02 %	**99.98%**
		500	0.00 %	0.01 %	**99.99%**
	PCARDE	10	0.07 %	0.25 %	**99.68%**
susy	RanF	100	0.00 %	**98.94%**	1.06 %
		200	0.00 %	**98.18%**	1.82 %
		500	0.00 %	**99.26%**	0.74 %
	PCARDE	10	0.07 %	0.17 %	**99.76%**
higgs	RanF	100	0.59 %	**79.76%**	19.65 %
		200	0.37 %	**87.13%**	12.50 %
		500	0.21 %	**93.82%**	5.97 %
	PCARDE	10	0.02 %	0.02 %	**99.96%**
hepmass	RanF	100	0.00 %	0.49 %	**99.51%**
		200	0.02 %	1.21 %	**98.77%**
		500	0.00 %	0.44 %	**99.56%**
	PCARDE	10	0.01 %	0.14 %	**99.85%**
epsilon	RanF	100	0.08 %	0.31 %	**99.61%**
		200	0.09 %	0.38 %	**99.53%**
		500	0.10 %	0.46 %	**99.44%**
	PCARDE	10	**64.37%**	34.95 %	0.68 %

Table 2.3: Bayesian correlated t test probabilities for Random Forest and PCARDE against Rotation Forest with 10 trees. The right probability is the probability of Rotation Forest performing better than the opponent. The region (Left, Right, or ROPE) with the highest probability is highlighted in bold.

probability was over 98%, showing that the best classifier far outperformed its contender. In most cases, this clear winner was Rotation Forest.

2.5.3 *Execution time analysis*

The evaluation in the previous section was conducted in terms of accuracy. Although the strengths of Rotation Forest were demonstrated in (Rodriguez et al., 2006) for small and medium-sized data sets, this paper reinforces that conclusion by working with large data sets and using modern Bayesian

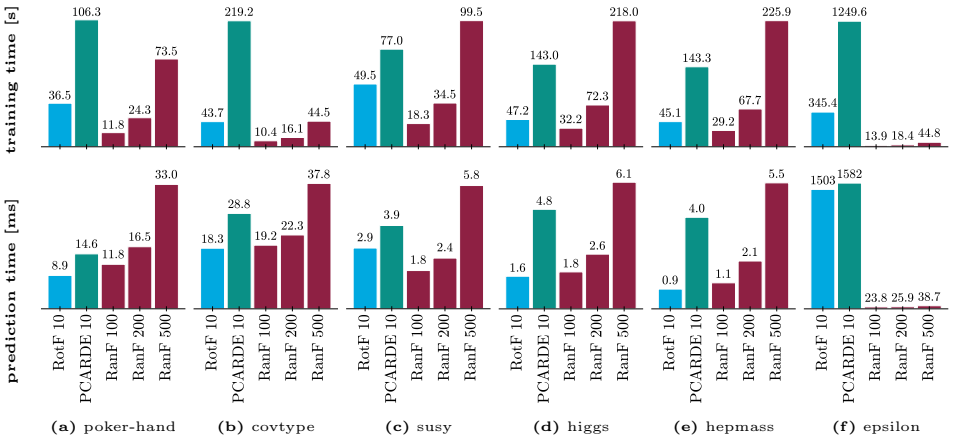


Figure 2.4: Training times (top row) and prediction times (bottom row) for each data set. Blue bar corresponds to Rotation Forest, green bar corresponds to PCARDE, and burgundy bars correspond to Random Forest.

statistical tests. Nevertheless, the main contribution of this research is the adaptation of Rotation Forest to work in Big Data environments where execution time is crucial. In this regard, an evaluation and comparison is presented in terms of both execution time and speedup.

Figure 2.4 shows a comparison of execution times for Rotation Forest with 10 trees (blue bar), PCARDE (green bar), and Random Forest (burgundy bars). The figures on the top row refer to training times (in seconds), while those on the bottom row refer to prediction times (in milliseconds).

Regarding training time, Random Forest with 100 trees was the fastest on six data sets, as expected. If we compare the two methods that use PCA as part of their data transformation step (Rotation Forest and PCARDE, both with 10 trees), the fastest method was clearly Rotation Forest.

Looking close at prediction times, it is somewhat surprising that the Rotation Forest classifier with 10 trees was the fastest on four out of six data sets. However, this is because the size of Rotation Forest was ten times smaller than Random Forest, and PCA was not computed at prediction (i.e., only a matrix multiplication for rotating data was performed). Regarding the comparison of Rotation Forest against PCARDE, as we might expect, prediction was also faster with Rotation Forest.

For the epsilon data set, the difference between Random Forest against the other two is dramatic. The explanation of this may be found in the number of features. While for the other five data sets, the numbers of features

range between 11 and 54, epsilon data set has 2 000. PCA calculation and data rotation require much more time as the number of features (i.e., data set dimensionality) grows, and thus, Rotation Forest and PCARDE execution times are highly penalized. Nevertheless, we consider that the execution time for Rotation Forest with the epsilon data set was acceptable for Big Data scenarios.

Experimental results

Having shown the competitiveness of Rotation Forest execution times, it is of great importance to determine how well it scales (i.e., how runtime decreases as the number of machines increases). In order to evaluate this, a cluster with one master node and a number of worker nodes ranging from 2 to 24 was used. Each node was of the n1-highmem-2 type, which had 2 virtual cores of the same type as the nodes described in Section 2.5.1. The following comparison of execution time and speedup, was performed on the medium-size susy data set.

Figure 2.5 shows the evolution of training time (left) and prediction time (right). As was expected, the execution time decreased as the number of workers increased. The improvement begins to be less remarkable around the fourteenth or the sixteenth worker, moreover, with the addition of the eighteenth worker, the time actually increases. This behavior is expected because every algorithm has a certain execution time that cannot be optimized, our experiments with the susy data set are a good example of that. This is because the data set is not extremely large, thus, when the number of workers increases above 20, the time required for data communication and transfer was greater than the benefit of increasing the computational power (i.e., additional workers).

Speedup (see Equation 2.2) is conventionally used as a metric for measuring algorithm scalability in a cluster.

$$speedup = \frac{\text{sequential execution time}}{\text{parallel execution time}} \tag{2.2}$$

For computing the sequential (not parallel) execution time, a machine with one n1-highmem-2 node, and one Spark partition was used.

Figure 2.6 shows the speedup evolution of training and prediction. As seen in Figure 2.5, the improvement of using more nodes both in training and testing is clear up to 16 nodes. Thereafter, the improvement was minimal, and using more than 20 workers decreased the speedup. It is worth noting that the speedup is higher in prediction than in training, because PCA computation for the K feature subgroups was not fully parallel in training.

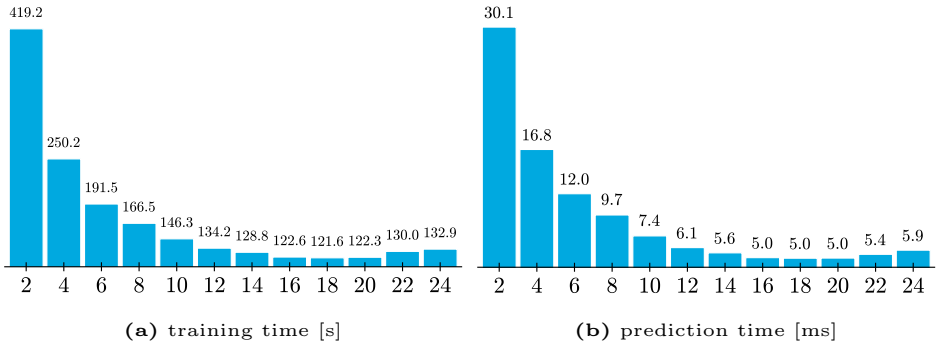


Figure 2.5: Execution time evolution for susy data set. The left bar plot refers to the training time (in seconds) while the right plot shows prediction time (in milliseconds). The x-axis indicates the number of worker nodes.

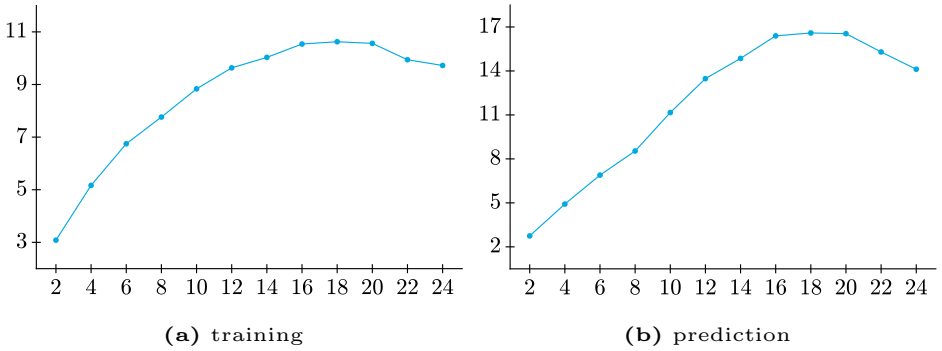
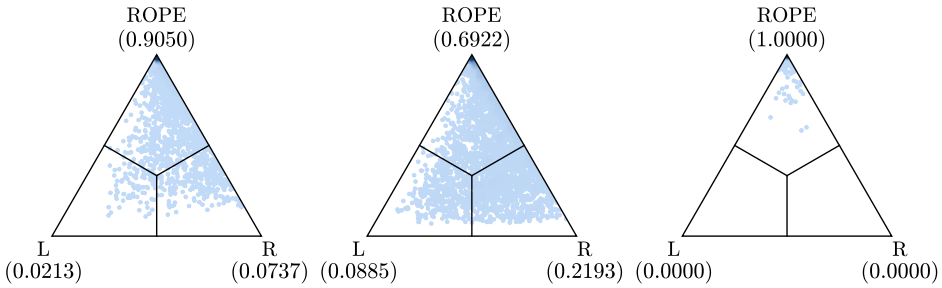


Figure 2.6: Speedup (y-axis) for susy data set. The left plot refers to training and the right plot to prediction. The x-axis indicates the number of worker nodes.

However, in the prediction phase, the input data rotation was a fully parallel MapReduce matrix multiplication.

2.5.4 *Study of ensemble size*

When Rotation Forest classifiers are trained, PCA is the most time-consuming step. Therefore, the main optimization will be to reduce the number of PCAs that have to be calculated. For this purpose, three parameters of the algorithm can be adjusted: L, the number of rotations; T, the number of trees; and K, the number of feature subsets. In this section, we will focus only on the first two, because they determine the ensemble size.



(a) small *vs* medium **(b)** small *vs* large **(c)** medium *vs* large

Figure 2.7: Hierarchical Bayesian test heatmap for comparing the influence of ensemble size in Rotation Forest. Three sizes were compared: (a) 10×1 vs 50×1, (b) 10×1 vs 100×1, and (c) 50×1 vs 100×1.

Up until this point, all the experiments had been performed with $T = 1$, which meant that the ensemble size was equal to L (i.e. $L \times 1$ as in the standard Rotation Forest). Figure 2.7 shows a hierarchical Bayesian test for statistical comparison of the three ensemble sizes that have so far been used. It can be roughly concluded that small (10×1), medium (50×1), and large (100×1) ensembles achieved equivalent levels of accuracy. Specifically, Figure 2.7.a shows the similar performance levels of small and medium ensembles in 90.5% of cases; in 2.7.b, small and large ensembles are shown to perform similarly in 69.2% of cases; and, no statistical difference between medium and large ensembles is shown in 2.7.c. The use of additional trees in the ensemble offered no significant advantage, and therefore, a small Rotation Forest with 10 trees was both sufficiently acceptable (taking into account accuracy) and computationally cheaper.

Commonly, the calculation of a data rotation matrix requires much more time than the training of a single tree. Thus, a strategy that could be followed to accelerate the training of Rotation Forest of a certain size is to decrease the number of rotations, but at the same time, in order not to reduce the number of base classifiers in the ensemble, more than one tree will be trained with the same rotation. This approach, already proposed in (Stiglic et al., 2011), acquires special relevance in the context of Big Data, since it would allow the reduction of the computing workload. Of course, as long as this simplification of the process does not significantly affect the final performance. This is precisely what will be evaluated in this section. Specifically, a small Rotation Forest ensemble of 10 trees could be built by rotating the data ten times and training one tree with each rotation ($L \times T = 10 \times 1$),

or by rotating the data five times and training two trees with each rotation ($L \times T = 5 \times 2$). Training the 5×2 ensemble will take about half the time of training the 10×1 ensemble. With this in mind, different experiments were launched varying both the number of rotations and the ensemble size.

Table 2.4 shows the results of the hierarchical Bayesian tests comparing different configurations of number of rotations (L) and number of trees per rotation (T). In all cases, for a given ensemble size, the left region corresponds to the method with the highest number of rotations and the right region to the fastest alternative, with a reduced number of rotations. The results of using the proposed strategy to accelerate the training phase of Rotation Forest are quite interesting. For small Rotation Forest (size 10), in 94.43% of cases, 10×1 and 5×2, were equivalent. For medium Rotation Forest (size 50), in 88.82% of cases, 50×1 and 10×5, performed in a similar way. In 54.5% of cases, 50×1 resulted better than 5×10, which means that for this specific scenario, reducing the number of rotations tenfold would be counterproductive for accuracy (it appears that here the additional diversity provided by the 50 rotations is relevant to obtain good results). Ending with the last of the medium Rotation Forests comparisons (10×5 vs. 5×10), in 89.18% of occasions they turned out to be equivalent. Regarding the large Rotation Forest (size 100), in 98.35% of cases, 100×1 and 10×10, were equivalent, as well as 100×1 and 5×20, which showed to be equivalent in 83.65% of cases. Finally, equivalent performance is also shown between 10×10 and 5×20, which occurred in 94.15% of cases.

We trained small, medium, and large Rotation Forest classifiers with $L = 5$ (5×2, 5×10, and 5×20, respectively), and with $L = 10$ (10×1, 10×5, and 10×10, respectively), in what follows, the value of L is fixed (to 5 or to 10) and the influence of the parameter T is analyzed. Figure 2.8 shows the hierarchical Bayesian test heatmap for $L = 5$ and $L = 10$ on the top and bottom row, respectively. For both values of L, all the tests shown in Figure 2.8.a–f depict statistical equivalence with independence of the value of T.

2.5.5 *Influence of bootstrap in Big Data*

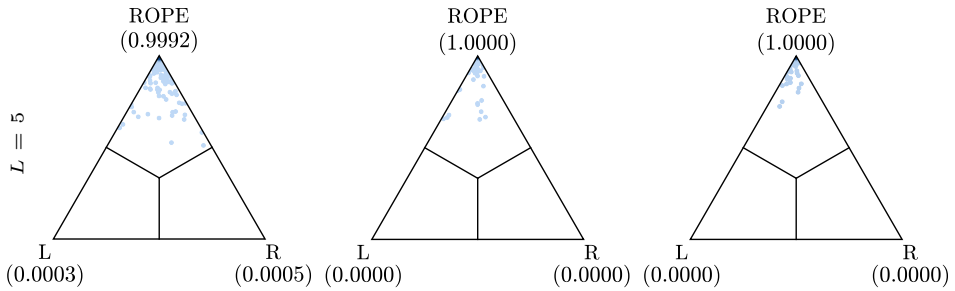
Bootstrapping is used in Rotation Forest as a part of the PCA calculation step, to avoid repeatedly obtaining the same principal components. The reason for doing so is to attempt to maximize the diversity of the ensemble. In (Rodriguez et al., 2006), a bootstrap size of 75% was recommended. Nevertheless, this value could be lower, because the number of instances in

Ensemble size	Left			Right			ROPE	Heatmap
	L	T	Prob.	L	T	Prob.		
small (10)	10	1	0.0505	5	2	0.0052	**0.9443**	
	50	1	0.0873	10	5	0.0245	**0.8882**	
medium (50)	50	1	**0.5450**	5	10	0.1695	0.2855	
	10	5	0.0757	5	10	0.0325	**0.8918**	
	100	1	0.0100	10	10	0.0065	**0.9835**	
large (100)	100	1	0.1340	5	20	0.0295	**0.8365**	
	10	10	0.0490	5	20	0.0095	**0.9415**	

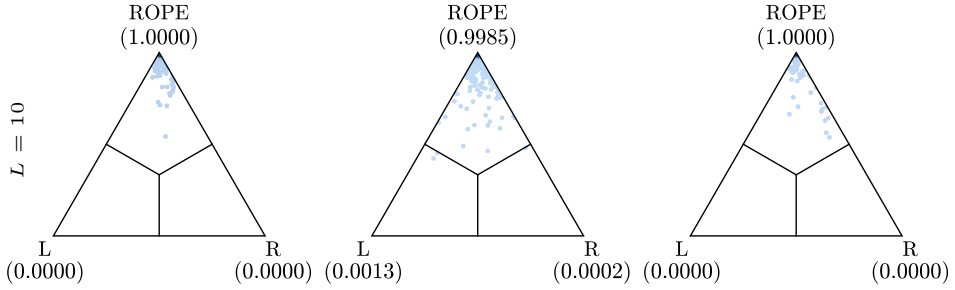
Table 2.4: Hierarchical Bayesian test results comparing the influence of different variants of ensemble size in Rotation Forest. The variants are obtained by setting a combination of L and T parameters for constructing small, medium, and large ensembles. The region (Left, Right, or ROPE) that gather the highest probability is highlighted in bold.

Big Data is very high. Hence, a bootstrap sample containing a low percentage of instances should be sufficiently representative of the entire classification problem. The representativeness of the bootstrap sample is therefore not as important as the fact that it can generate different principal components, which is why lower bootstrap values could be beneficial or, at least, not harmful. Another thing to bear in mind is that in our case, the smaller the bootstrap sample, the faster the computation of the PCA. So, fixing a parameter that is as low as possible will shorten the training time.

Small Rotation Forest with $L = 10$ and $T = 1$ were trained with different bootstrap sizes – 10%, 25%, and 50% – to assess whether the bootstrap size might have an effect on the accuracy rates. Figure 2.9 shows the hierarchical Bayesian test heatmaps, so that the results for the three bootstrap size cases may be compared. The Bayesian test shows that the bootstrap size did not have a significant impact on the accuracy of Rotation Forest, because the ROPE values in Figures 2.9.a, 2.9.b, and 2.9.c were close to 100%. Hence, for Big Data environments, faster training times could be achieved by using small bootstrap sample sizes, e.g., 10%.

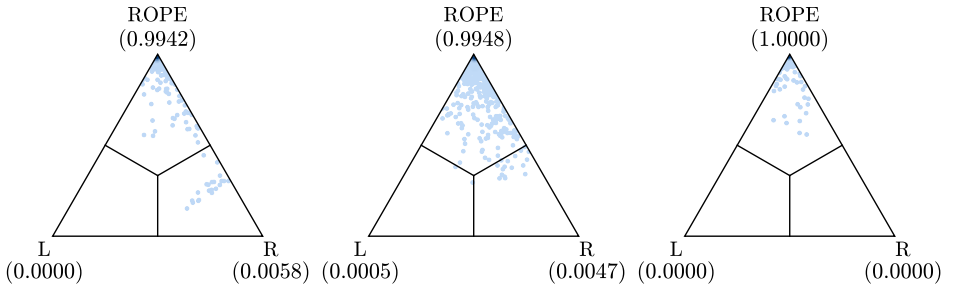


(a) $T = 2$ *vs* $T = 10$ **(b)** $T = 2$ *vs* $T = 20$ **(c)** $T = 10$ *vs* $T = 20$



(d) $T = 1$ *vs* $T = 5$ **(e)** $T = 1$ *vs* $T = 10$ **(f)** $T = 5$ *vs* $T = 10$

Figure 2.8: Hierarchical Bayesian test heatmap for comparing the influence of increasing the number of trees for each rotation: on the top row, 5 rotations with comparisons of 2, 10, and 20 trees. On the bottom row, 10 rotations with comparisons of 1, 5, and 10 trees.



(a) 10% *vs* 25% **(b)** 10% *vs* 50% **(c)** 25% *vs* 50%

Figure 2.9: Hierarchical Bayesian test heatmap for comparing the influence of bootstrap size in Rotation Forest. Three sizes were compared: 10%, 25%, and 50%.

2.6 CONCLUSIONS AND FUTURE WORK

Conclusions and future work

This paper has presented a MapReduce design of a variant of Rotation Forest and its implementation in Spark (the implementation is publicly available[5]). A thorough experimental campaign with Big Data sets has been performed, to assess the viability of the Rotation Forest algorithm for Big Data processing. The proposal presented in this paper has demonstrated its high performance and fast execution time, as well as its good scalability performance in cloud-based clusters. All of it with six large data sets having a number of instances ranging between 400 000 and 11 000 000, and a number of attributes ranging between 11 and 2 000.

Modern Bayesian tests (hierarchical and correlated t test) were used for evaluating the statistical differences between the different algorithms that were tested. Our experiments have consolidated the results of (Rodriguez et al., 2006), in which the Rotation Forest algorithm was reported to be a better option than Random Forest, and the same conclusion has now been corroborated with massive data sets. Furthermore, the superiority of Rotation Forest in relation to PCARDE, a very recent ensemble algorithm for Big Data, has been demonstrated.

An experimental exploration of some algorithm parameters and their optimal values has been performed. Through that study, the approach for training Rotation Forest with Random Forest rather than of a single decision tree as the base classifier has proved that accurate models with fewer data rotations, and therefore models that train and predict faster, are indeed feasible. The analysis also reported that small ensembles, consisting of 10 trees, are accurate enough for the Big Data sets used in the study.

Additionally an evaluation of the influence of the bootstrap sample size with large data sets has been conducted. The conclusion of that evaluation was that sampling 10% of the data provided classification models with an equivalent performance to those that sampled 25% or 50% of the data. The use of low percentages therefore means simpler PCA calculations and faster training of the Rotation Forest.

Rotation Forest code has been carefully developed following the Spark ML API guidelines, aiming towards its incorporation in the API within the near future. The number of ensemble algorithms available for Big Data is still scarce, specially for tasks such as online learning and unbalanced data sets, among others. Ensembles have also been used for data stream analy-

[5]The implementation for Spark is publicly available at https://github.com/mjuez/rotation-forest-spark.

sis (Krawczyk et al., 2017), the adaptation of Rotation Forest for streaming Big Data frameworks is therefore a challenging task that might mean that it could be applied to several real world problems.

In this paper, the presentation of Rotation Forest only covered classification problems and its adaptation to Big Data regression problems is still a future research line. Likewise, the adaptation of Rotation Forest for unbalanced learning (Díez-Pastor et al., 2015b) in Big Data scenarios, represents an area of potential interest for the scientific community.

ACKNOWLEDGMENTS

This work was supported through project TIN2015-67534-P (MINECO/-FEDER, UE) of the *Ministerio de Economía y Competitividad* of the Spanish Government, projects BU085P17 and BU055P20 (JCyL/FEDER, UE) of the *Junta de Castilla y León* (both projects co-financed through European Union FEDER funds), and by the *Consejería de Educación* of the *Junta de Castilla y León* and the European Social Fund through a pre-doctoral grant (EDU/1100/2017). The project leading to these results has received also funding from "la Caixa" Foundation, under agreement LCF/PR/PR18/-51130007. This material is based upon work supported by Google Cloud.

EXPERIMENTAL EVALUATION OF ENSEMBLE CLASSIFIERS FOR IMBALANCE EN BIG DATA

3

This journal paper presents an experimental evaluation on how the imbalanced problem affects ensemble classifiers in big data, and what impact it has to apply data-level approaches to overcome it.

Authors: Mario Juez-Gil, Álvar Arnaiz-González, Juan J. Rodríguez, César García-Osorio.
Type: Journal
Published in: Applied Soft Computing, 108 (JCR: Q1, SJR: Q1)
Year: 2021
Keywords: Unbalance, Imbalance, Ensemble, Resampling, Big Data, Spark
Reference: Juez-Gil et al. (2021a)

ABSTRACT

Datasets are growing in size and complexity at a pace never seen before, forming ever larger datasets known as Big Data. A common problem for classification, especially in Big Data, is that the numerous examples of the different classes might not be balanced. Some decades ago, imbalanced classification was therefore introduced, to correct the tendency of classifiers that show bias in favor of the majority class and that ignore the minority one. To date, although the number of imbalanced classification methods have increased, they continue to focus on normal-sized datasets and not on the new reality of Big Data. In this paper, in-depth experimentation with ensemble classifiers is conducted in the context of imbalanced Big Data classification, using two popular ensemble families (Bagging and Boosting) and different resampling methods. All the experimentation was launched in Spark clusters, comparing ensemble performance and execution times with statistical test results, including the newest ones based on the Bayesian approach. One very interesting conclusion from the study was that simpler methods applied to unbalanced datasets in the context of Big Data provided

better results than complex methods. The additional complexity of some of the sophisticated methods, which appear necessary to process and to reduce imbalance in normal-sized datasets were not effective for imbalanced Big Data.

Introduction

3.1 INTRODUCTION

In computing science, a new term has emerged to describe the sheer size of datasets, which are rapidly expanding throughout the world: Big Data. The main characteristics of Big Data were initially established by Laney (Laney, 2001) around the 3 Vs: Volume, Velocity and Variety. Additional Vs have since been added such as Value (Gantz and Reinsel, 2011) and Veracity (Jain, 2016)[1]. Big Data can essentially be defined at the intersection between these concepts: Volume (large datasets), Velocity (data-processing speeds must respond to data-generation speeds), Variety (different forms of data), Veracity (must be resilient under uncertain or imprecise data) and Value (usefulness) (Hariri et al., 2019). Real-world classification problems are not usually balanced; *i.e.*, the number of instances that belong to each class is unevenly distributed (Chawla et al., 2004). These classification problems, known as imbalanced or unbalanced learning, have received a lot of attention over recent years (He and Garcia, 2009; Díez-Pastor et al., 2015b; Fernández et al., 2017). The applications of imbalanced learning are broad and diverse, because balanced datasets in real-life are unfortunately rare to find. Moreover, the class in which we are usually interested is the underrepresented one (Krawczyk, 2016). Some applications of imbalanced learning include activity recognition (Gao et al., 2016), behavior analysis (Azaria et al., 2014), industrial systems monitoring (Diez-Pastor et al., 2021; Ramentol et al., 2016), video mining (Gao et al., 2014) and cancer malignancy grading (Krawczyk et al., 2016), among others. A general classifier applied with no strategy to process imbalanced datasets will tend to ignore the minority class and will therefore almost inevitably misclassify it. Imbalanced classification is frequent in standard classification, but it is crucial within Big Data environments (Leevy et al., 2018; Fernández et al., 2017; Luengo et al., 2020). Several alternative approaches for processing imbalanced data

[1] Despite the fact that some authors have identified other Vs within the Big Data equation, the five Vs presented in this paper are by far the most representative.

have been proposed and can be grouped into four categories (Galar et al., 2012)[2]:

- Algorithm-level: the algorithms are internally modified to process the imbalance, *i.e.*, the algorithms are biased towards focusing on minority class instances.
- Data-level: instead of changing the algorithms, the idea is to resample or to rebalance the class distribution within the input dataset.
- Cost-sensitive: the algorithms incorporate different misclassification costs for the different classes within the dataset.
- Classifier ensembles: the capability of ensembles to improve on the results of difficult classification tasks has been demonstrated. A common ensemble construction method for imbalanced learning is at the data-level, by training each base classifier with a pre-processed dataset. Another approach is the use of heterogeneous ensembles (Ghaderi Zefrehi and Altınçay, 2020).

The decision to use one or another approach is highly problem dependent, nevertheless algorithm-level approaches and cost-sensitive approaches are, according to (Galar et al., 2012), more data dependent, while the other two methods are more versatile. Data-level is currently the most popular approach to process imbalanced data (Haixiang et al., 2017). The present paper is broadly focused on data-level and classifier ensemble approaches and specifically on how the resampling techniques help ensemble learning to address imbalanced classification in Big Data problems.

The topic of imbalanced classification in Big Data is still at an early stage (Fernández et al., 2017, 2018a). Two gaps have been identified: the few ensemble methods designed for Big Data problems (González et al., 2020) and perhaps even fewer for processing imbalance within Big Data (Luengo et al., 2020). The aim of this work is therefore to conduct an experimental evaluation of some of the most popular ensembles in the context of imbalanced Big Data classification: Bagging and Boosting using various resampling techniques. The Apache Spark framework, a widely used Big Data platform, will be used for the experimentation. Since proposing new implementations for Big Data is beyond the scope of this study, the experiments

[2] As pointed out in (Díez-Pastor et al., 2015b), these categories are not mutually exclusive.

will only be run on currently available (or easily implementable) ensemble and resampling methods in Apache Spark.

The rest of the paper will be structured as follows. In section 3.2, research into ensembles applied to the imbalance problem will be explained. In section 3.3, the state-of-the-art of data pre-processing techniques for Big Data will be described. In section 3.4, details of the experimentation will be reported, followed by an explanation of the results that will be given in section 3.5. Finally, the main conclusions and future research lines will be discussed in section 3.6.

3.2 ENSEMBLE LEARNING FOR IMBALANCED PROBLEMS

As is well-known, ensemble learning is based on the construction and the subsequent combination of several classifiers to obtain a new classifier that can outperform the base classifiers (Galar et al., 2012). Desirable properties of the classifiers that make up ensembles are accuracy, instability and diversity, among others. Instability is a useful property of the base classifiers of an ensemble (*i.e.*, small variations within an input dataset can generate significant changes within the classifier). Diversity is essential, because if all the base classifiers were to predict the same class, then there might be little point in building an ensemble. There are multiple methods to force diversity, many of which are based on either Bagging (Breiman, 1996) (which resamples the dataset) or Boosting (Schapire, 1990) (which assigns a weight to the instances depending on the difficulty of their classification). However, there are some other methods that involve alternative techniques (Kuncheva and Rodriguez, 2007; Maudes et al., 2009a,b, 2012; Pardo et al., 2011).

Pre-processing techniques that balance the class proportions can be applied in a straightforward manner within an ensemble and will usually either reduce the size of the majority class, increase the size of the minority class, or even achieve both at the same time (Díez-Pastor et al., 2015b). Although there are several techniques, we will only summarize those that are generally used to deal with imbalance in ensemble learning:

- Random UnderSampling (RUS): consists of randomly removing examples of the majority class. The number of examples removed reduces the imbalance ratio, and it can balance the dataset, or even unbalance it in the opposite direction.

- Random OverSampling (ROS): consists of randomly replicating examples of the minority class. As with the previous case, the number of examples generated reduces the imbalance ratio.
- Synthetic Minority Oversampling TEchnique (SMOTE): generates synthetic instances of the minority class by interpolation of two original minority instances (Chawla et al., 2002). It applies k-nearest neighbors to ensure that the instances selected for the interpolation are close to each other.
- Random OverSampling Examples (ROSE): generates a new synthetic dataset of a certain size and a certain imbalance ratio (Menardi and Torelli, 2014). A typical strategy is to create a balanced dataset of the same size as the input dataset. ROSE generates new artificial examples according to a smoothed bootstrap approach.
- Random Balance (RB) (Díez-Pastor et al., 2015a): as the balancing technique, either the generation or the elimination of instances (more suitable for a dataset), and the optimal imbalance ratio will *a priori* be unknown, a generative method (SMOTE) and a reduction of instances method (RUS) are randomly combined in each classifier by the RB ensemble method. The imbalance ratio of the datasets that are generated is also random, all of which yields additional diversity from which the ensemble may also benefit.

These pre-processing techniques can either be performed once at the beginning (before training the ensemble classifier), or before training each single base classifier of the ensemble. For example, the use of undersampling in each base classifier of a bagging ensemble, is known as under-bagging, in the same way that over-bagging consists in using oversampling within bagging (Galar et al., 2012).

During the last two decades, different combinations of ensembles and pre-processing techniques have been proposed to improve classification performance (Galar et al., 2012), both for Bagging and Boosting (Tanha et al., 2020) families, as well as hybrid solutions (Liu et al., 2009).

3.3 IMBALANCED DATA PRE-PROCESSING FOR BIG DATA

Despite the intense research over past decades in imbalanced learning for normal-sized datasets, the imbalance problem in relation to computational

scalability has yet to be thoroughly explored (Jeon and Lim, 2020). The first comparison of several resampling methods in both Hadoop and Spark was presented in (Fernández et al., 2017). For a complete review of these methods, we would recommend the following papers (Leevy et al., 2018; Luengo et al., 2020).

Imbalanced data pre-processing for Big Data

Simple sampling techniques, such as Random OverSampling (ROS) and Random UnderSampling (RUS) were among the first attempts to deal with imbalance in Big Data classification (del Río et al., 2014). The effect of RUS on Big Data with simulated class imbalance datasets was likewise studied in (Hasanin and Khoshgoftaar, 2018).

SMOTE is another popular algorithm to deal with imbalanced datasets, the straightforward implementation and sound performance of which has increased its popularity and led to a proliferation of SMOTE variants, that number over 85 (Fernández et al., 2018b) today. The Big Data version of SMOTE (SMOTE-BD) was recently presented in (Basgall et al., 2018). A Big Data implementation of a SMOTE algorithm based on the Neighborhood Rough Set Model (Hu and Li, 2013) (NRSBoundary-SMOTE) was presented in (Hu et al., 2014a).

The ROSEFW-RF (Triguero et al., 2015) algorithm (Random OverSampling and Evolutionary Feature Weighting for Random Forest) was the winning algorithm in the ECBDL'14 Big Data competition. This algorithm balances the classes using ROS and identifies the most relevant features through an evolutionary feature-selection process, before building a Random Forest classifier. The algorithm demonstrated its strengths winning the competition.

Despite the fact that most studies are based on ensemble and simple sampling techniques, more sophisticated methods have been presented, such as evolutionary undersampling (Triguero et al., 2015), among others.

The lack of methods applicable to imbalanced learning in Big Data frameworks poses difficulties when extracting knowledge from large datasets with unevenly distributed classes. The scarcity of methods becomes even more noticeable when it is compared with the large number of methods for normal-sized datasets (Luengo et al., 2020). Furthermore, some of these methods were implemented only for Hadoop before Spark became more popular, and thus, their comparison is even more challenging. To the best of our knowledge, no comprehensive experimentation has been conducted to date, to compare resampling-based ensemble methods with Big Data imbalance classification, and we consider that there is a need for that kind of experimental evaluation. Knowing whether resampling tech-

niques can and to what extent they can benefit imbalanced Big Data classification is essential to accomplish meaningful and successful future research.

3.4 EXPERIMENTAL SET-UP

The experimental set-up was organized into two groups, in order to determine the effects of the balancing/resampling strategies on the performance of the ensembles:

- Dataset resampling and then training: the dataset was balanced once at the beginning (following one of the strategies) and an ensemble classifier was trained using the resampled dataset.
- Resampling within the ensemble: the ensemble performs a resampling strategy before the training of each base classifier (*i.e.*, there are as many resamples as there are base classifiers).

3.4.1 *Methods*

Five popular sampling techniques were tested in the experimentation: RUS, ROS, SMOTE, ROSE, and RB. All the algorithms were implemented in Scala and executed within the Apache Spark framework. The implementation of RUS and ROS was straightforward using the Spark API. As mentioned above, SMOTE-BD (Basgall et al., 2018) is an implementation of the SMOTE algorithm for Spark, however, we used our own implementation of SMOTE[3] based on Fang's approximated KNN,[4] that uses a hybrid spill-tree approach (Liu et al., 2004), to achieve high accuracy and search efficiency. A parallel version of ROSE was implemented under the Spark framework following the existing R implementation (Lunardon et al., 2014). Finally, the RB implementation is a variant of the original algorithm specifically adapted to Big Data[5]. Table 3.1 lists the sampling methods that were tested.

The performance of the sampling techniques was compared using three different ensemble classifiers available for Spark: Bagging (BAG), Random Forest (RANF), and Gradient Boosting Trees (GBT), thus covering the Bagging and Boosting families of ensembles.

[3] approx-smote on GitHub: https://github.com/mjuez/approx-smote

[4] saurfang spark-knn on GitHub: https://github.com/saurfang/spark-knn.

[5] To speed up its execution with Big Data, instead of the SMOTE and RUS methods in the original algorithm, the new one used a combination of ROS and RUS.

Abbr.	Method	Details and parameters
RUS	Random Undersampling	Undersampling without replacement until 50% of the data belongs to minority class.
ROS	Random Oversampling	Oversampling until 50% of the data belongs to minority class.
SMOTE	SMOTE (approximated)	$K = 5$, Oversampling until 50% of the data belongs to minority class.
ROSE	ROSE	$shrinkFactor = 1$, Generate n synthetic examples with 50% probability of belonging to any class. (n is the size of the original dataset)
RB	Random Balance	A random imbalance ratio is applied for each base classifier.

Experimental set-up

Table 3.1: Sampling techniques used in the experimental study and their configuration.

Abbr.	Method	Details and parameters
BAG	Bagging	$numTrees = 100,\ maxDepth = 5,$ $subsamplingRate = 1.0,\ featureSubsetStrategy = all$
RANF	Random Forest	$numTrees = 100,\ maxDepth = 5,$ $subsamplingRate = 1.0,\ featureSubsetStrategy = auto$
GBT	Gradient Boosting Trees	$maxIter = 100,\ maxDepth = 5$

Table 3.2: Ensemble classifiers used in the experimental study and their configuration.

In the first group of experiments (resampling before training), the ensembles were trained using the balanced datasets obtained after applying the sampling techniques: RUS, ROS, SMOTE, and ROSE. In the second group of experiments (resampling within the ensemble), specific implementations of Bagging and Gradient Boosting Trees were developed for processing the imbalance problem. In these implementations, a new sampling was performed for each base classifier in the ensemble, for which the following methods were used: RUS, ROS and RB.

So that the comparison of the ensemble algorithms was on equal terms, all the ensembles were composed of 100 trees, each with a depth of 5. The details of the algorithms and their parameters are listed in Table 3.2.

The methods without resampling and therefore with no established method of processing imbalance were also included in the experiments, as a baseline performance reference, these methods are referred to in this paper as the Gini variant. Traditionally, classification and regression trees (CART) have used the Gini index as a split criterion. Nevertheless, when datasets are imbalanced, the Gini index is biased in favor of the majority class (Liu et al., 2018). For this reason, the weighted Gini (WGini) index could be beneficial

for the application of trees to imbalanced learning, thus it was also included in this experimental evaluation[6].

To sum up, different ensemble classifiers were evaluated: BAG, RANF, and GBT. For each ensemble, two impurity indexes were used as a baseline (Gini and WGini) and some popular resampling techniques were tested: ROS, RUS, SMOTE, ROSE, and RB. The use of weighted Gini with the resampling techniques was not explored because, as we resampled until both classes were totally balanced, the result would be equivalent to the use of the default Gini index.

3.4.2 *Experimental framework*

Six popular classification datasets for Big Data were used in the experiments[7]. Two standard techniques were used to generate the imbalanced datasets (Basgall et al., 2018). Several imbalanced datasets were generated from the multi-class datasets (*i.e.*, covtype and kddcup), by selecting the majority class and one of the minority classes. Since the binary datasets (*i.e.*, susy, higgs, and hepmass) are actually almost evenly balanced, the imbalance was forced by subsampling the class with fewer instances using two different imbalance ratios: 4 and 16. The ECBDL'14 is already unbalanced, so it was used as is without any transformation. Only two subsamples (approximately 1 million and 10 million instances) with the original imbalance ratio were processed, because of the huge size of this dataset and budgetary limitations on the computing time that may be requested on Google Cloud clusters. Table 3.3 summarizes the resulting 16 datasets. All the datasets were stored in libsvm (Chang and Lin, 2011) format. As the implementation of some algorithms, such as SMOTE or ROSE, cannot process nominal features, all nominal features were binarized using one-hot encoding.

The experiments were performed using 5 repetitions of a 2-fold cross-validation. A random seed value was set at 46 to enable experimental repeatability.

For evaluating and comparing statistical differences in performance, average ranks and statistical tests were used. Average ranks were computed by

[6] Due to the fact that the weighted Gini index is not available on the Spark trees, the instances were weighted according to the imbalance ratio of their class, as proposed by Chen *et al.* (Chen et al., 2004).

[7] The covtype, susy, higgs, hepmass, and kddcup datasets are available from the UCI Machine Learning repository (Dua and Graff, 2017) https://archive.ics.uci.edu/ml/index.php. The ECBDL14 dataset is available from the Evolutionary Computation for Big Data and Big Learning Workshop competition website http://cruncher.ico2s.org/bdcomp/.

Dataset	# instances	# attributes	# maj	# min	IR	Size (GB)
COVTYPE 1 vs 4	229 207	54	211 840	17 367	12.20	0.02
COVTYPE 1 vs 3	232 350	54	211 840	20 510	10.33	0.02
COVTYPE 1 vs 2	247 594	54	211 840	35 754	5.92	0.02
COVTYPE 0 vs 4	300 668	54	283 301	17 367	16.31	0.03
COVTYPE 0 vs 3	303 811	54	283 301	20 510	13.81	0.03
COVTYPE 0 vs 2	319 055	54	283 301	35 754	7.92	0.03
ECBDL'14 1M	999 716	893	978 128	21 588	45.31	15.70
SUSY IR16	2 881 796	18	2 712 173	169 623	15.99	1.04
SUSY IR4	3 389 320	18	2 712 173	677 147	4.00	1.23
KDDCUP dos vs r2l	3 884 496	119	3 883 370	1126	3448.82	0.50
KDDCUP dos vs normal	4 856 151	119	3 883 370	972 781	3.99	0.62
HEPMASS IR16	5 578 586	28	5 250 124	328 462	15.98	3.20
HIGGS IR16	6 194 093	28	5 829 123	364 970	15.97	3.26
HEPMASS IR4	6 561 364	28	5 250 124	1 311 240	4.00	3.77
HIGGS IR4	7 284 166	28	5 829 123	1 455 043	4.00	3.94
ECBDL'14 10M	9 998 491	893	9 783 328	215 163	45.47	45.80

Table 3.3: 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 (Chang and Lin, 2011) format.

assigning, for each dataset, one to the best classifier, two to the second, and so on. When there was a tie between several classifiers, their average rankings were assigned to each one. The final value assigned to each method was the mean of its rankings across all datasets. The Friedman test, followed by the Hochberg $1\times N$ (one vs. all) and the Nemenyi $N\times N$ (all vs. all) *post hoc* procedures (Demšar, 2006) were used to evaluate any statistical difference in the results.

Bayesian analysis (Benavoli et al., 2017) (*baycomp*[8] library was used) was conducted using Bayesian hierarchical sign tests, which separately account for the results of all folds and repetitions, for comparative purposes. The number of samples for all Bayesian comparisons was set at 50 000. A common graphical representation of this type of analysis is a ternary plot (Juez-Gil, 2020) where the region of practical equivalence (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 0.05, which means that two algorithms with a difference in performance of less than 0.05 will be considered equivalent.

[8]The *baycomp* library is publicly available at https://baycomp.readthedocs.io/en/latest/.

The experiments were executed in cloud-based clusters provided by the Google Cloud Platform. The cloud cluster was composed of a total of twenty-eight nodes (one master node and twenty-seven worker nodes). All nodes were of the n1-highmem-8 type, which had 8 virtual CPUs, and 52 GB of RAM. Hence, the cluster size was 224 vCPUs and occupied 1 456 GB of RAM. At that point in time, the vCPUs of n1-highmem nodes could be of the four following types: Intel Xeon (Skylake), Intel Xeon E5 (Sandy Bridge), Intel Xeon E5 v2 (Ivy Bridge), Intel Xeon E5 v3 (Haswell), or Intel Xeon E5 v4 (Broadwell E5). The Apache Spark environment was provided by Google Dataproc software, version 1.4, running on Debian 9, with Apache Hadoop 2.9.2, and Apache Spark 2.4.5[9]. Google Cloud Storage, as a distributed file system, was used for storing the datasets and the experimental results.

3.4.3 *Performance metrics*

Standard accuracy metrics tend to focus on all target classes equally, which is a problem for the minority ones (Brzezinski et al., 2020). For this reason, the use of a single metric in imbalance classification is not recommended, as it is commonly preferred to use several dedicated metrics to contrast interpretations.

In this study, the four distinct metrics under consideration were: F_1 Score (F_1-score), Matthews Correlation Coefficient (MCC), Geometric Mean (G-mean), and Area Under the Curve (AUC). All these metrics were defined using different values given in the confusion matrix. If we consider a problem with two classes: a class of interest, the "positive" class ($+$), and another usually much larger class, the "negative" ($-$) one; then the mistakes (False Positives and False Negatives) and successes (True Positives and True Negatives) of a classifier can be sorted within a confusion matrix, as follows:

		Predicted labels		
		$+$	$-$	total
Actual labels	$+$	TP	FN	P
	$-$	FP	TN	N
	total	$\hat{P}$	$\hat{N}$	

[9]The experiments of the Weighted Gini (WGini) was launched within a Apache Spark 3.0.1 cluster, since the instance weighting on the trees training were not supported in previous versions.

The F_1-score is the harmonic mean of *precision* $\left(\frac{TP}{TP+FP}\right)$ and *recall* $\left(\frac{TP}{TP+FN}\right)$:

$$\begin{aligned} \text{F}_1\text{-score} &= \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} \\ &= 2 \times \frac{\text{recall} \times \text{precision}}{\text{recall} + \text{precision}} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \end{aligned} \tag{3.1}$$

Experimental set-up

The MCC is a metric strongly advised for imbalance classification (Bekkar et al., 2013):

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{\hat{P} \times P \times \hat{N} \times N}} \tag{3.2}$$

The G-mean is the geometric mean of *sensitivity* and *specificity*:

$$\text{G-mean} = \sqrt{\text{sensitivity} \times \text{specificity}} = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{TN + FP}} \tag{3.3}$$

The domains of all the metrics, except MCC, were between 0 and 1, where 1 was the best value. Furthermore, MCC is a metric that expresses the correlation between the actual and the predicted class, with values ranging between -1 (total disagreement) and $+1$ (perfect agreement).

The definition of the AUC is slightly less straightforward. When the result of a binary classifier is the probability that the instance belongs to the class of interest, the final assignment of the class will ultimately depend on a threshold value from which we consider that the probability is sufficient to assign the class of interest. For each threshold value, we would have different results of the classifier, and therefore different *TP* and *FP* rate values (*TPR* and *FPR*). If we consider the entire range of values for the threshold, from zero to one, we can represent the results obtained for each threshold as points with the coordinates (TPR, FPR). The curve obtained by considering all these points is called the Receiver Operating Characteristic curve, or ROC curve. If we now consider the area under the curve (AUC), its value serves as an indicator of classifier performance for all possible thresholds. A good classifier will have a value close to 1 and a random (non-informative) classifier will have a value close to 0.5.

For the sake of simplicity, only the MCC and the AUC metrics are reported in the following sections. The reason is because, for this experimental evaluation, we have found that the four metrics can be paired in such a

way that both metrics of each pair lead to almost the same interpretations. The two groups of metrics are F_1-score and MCC, and G-mean and AUC. Nonetheless the results of the four metrics may be found in the 3.6.

3.5 RESULTS AND DISCUSSION

As previously stated, the experimentation was divided into two groups: resampling before training (see subsection 3.5.1) and resampling within the ensemble (see subsection 3.5.2). A comparison of both approaches may be found in subsection 3.5.3; a comparison of their execution times is presented in subsection 3.5.4; and, finally, all the results are discussed in subsection 3.5.5.

3.5.1 *Resampling before training ensemble*

This subsection shows the performance results of the ensemble methods trained after obtaining a dataset by resampling the original dataset. These experiments correspond with the data-level approach.

Table 3.4.a details the results of the experiments for MCC, and Table 3.4.b for AUC. The best results of the datasets from each ensemble classifier group (*i.e.*, BAG, RANF, and GBT) are shown within gray boxes, and the best overall results are highlighted within black boxes. The blueness intensity of each cell is used for highlighting the results, the higher the metric, the darker the blue. A clear superiority of GBT over BAG and RANF ensemble classifiers was shown. The impact of the RUS, ROS, and SMOTE pre-processing methods gave quite similar performances at a glance. Moreover the classifier trained with the original dataset using the weighted Gini index, could also be included in that similar-performing group of methods (*i.e.*, row-wise blueness intensity is almost the same for the four variants: WGini, RUS, ROS, and SMOTE). ROSE by contrast, generally achieved worse performance. Focusing on each metric separately, the results for MCC tended to reveal that it is better not to pre-process at all and to train the classifier using the Gini index; on the other hand, AUC highlighted a better performance when pre-processing techniques were used or, at least, when the weighted Gini index was used. Nevertheless, it is worth noting that the results of the classifiers trained without taking the imbalance into account (*i.e.*, Gini variants) were, for some datasets, totally unacceptable, such as for example ECBDL or HIGGS, while the other alternatives offered good overall results.

(a) MCC

Dataset	BAG						RANF						GBT					
	Gini	WGini	RUS	ROS	SMOTE	ROSE	Gini	WGini	RUS	ROS	SMOTE	ROSE	Gini	WGini	RUS	ROS	SMOTE	ROSE
COVTYPE 1vs4	0.9522	0.9438	0.9422	0.9314	0.9392	0.8976	0.9206	0.9227	0.9229	0.9235	0.9244	0.5602	0.9906	0.9776	0.9686	0.9801	0.9791	0.8963
COVTYPE 1vs3	0.6753	0.5789	0.5696	0.5655	0.5592	0.0043	0.2925	0.5263	0.5186	0.5173	0.5210	0.0000	0.8548	0.7897	0.7675	0.7917	0.7947	0.0043
COVTYPE 1vs2	0.9801	0.9756	0.9734	0.9745	0.9738	0.9447	0.9759	0.9663	0.9680	0.9689	0.9665	0.6207	0.9975	0.9938	0.9930	0.9959	0.9958	0.9445
COVTYPE 0vs4	0.7408	0.6848	0.6741	0.6739	0.6803	0.2409	0.6734	0.6577	0.6580	0.6572	0.6640	0.2405	0.9051	0.8036	0.7822	0.8095	0.8108	0.2409
COVTYPE 0vs3	0.8993	0.7829	0.7870	0.7762	0.7763	0.5864	0.8184	0.7471	0.7514	0.7504	0.7413	0.0835	0.9751	0.9463	0.9218	0.9493	0.9487	0.5868
COVTYPE 0vs2	0.8852	0.8033	0.7995	0.8008	0.8008	0.3267	0.7828	0.7987	0.7921	0.7905	0.7961	0.3325	0.9317	0.8858	0.8816	0.8878	0.8921	0.3267
ECBDL'14 1M	0.0000	0.1275	0.1269	0.1226	0.1203	0.0883	0.0000	0.1353	0.1340	0.1344	0.1282	0.1180	0.0602	0.1555	0.1405	0.1560	0.1169	0.1166
SUSY IR16	0.4385	0.3061	0.3068	0.3002	0.2917	0.3334	0.4284	0.3091	0.3123	0.3117	0.3017	0.3381	0.4970	0.3537	0.3519	0.3541	0.3445	0.4524
SUSY IR4	0.5345	0.4685	0.4695	0.4665	0.4604	0.4830	0.5174	0.4752	0.4793	0.4775	0.4705	0.4884	0.5796	0.5295	0.5281	0.5287	0.5221	0.5569
KDDCUP dos vs r2l	0.9757	0.9596	0.4275	0.8650	0.7890	0.3895	0.9508	0.8245	0.5325	0.8818	0.8316	0.5442	0.9955	0.9885	0.4128	0.9938	0.9956	0.3936
KDDCUP dos vs nor.	0.9987	0.9991	0.9989	0.9993	0.9993	0.9952	0.9980	0.9979	0.9978	0.9978	0.9977	0.9946	0.9999	0.9999	0.9998	0.9999	0.9999	0.9954
HEPMASS IR16	0.6565	0.4319	0.4190	0.4353	0.4562	0.3364	0.4255	0.3428	0.3431	0.3433	0.3529	0.3244	0.6706	0.4742	0.4714	0.4737	0.5406	0.3121
HIGGS IR16	0.1158	0.1597	0.1591	0.1569	0.1497	0.1168	0.0000	0.1750	0.1761	0.1751	0.1746	0.1203	0.1944	0.2306	0.2289	0.2301	0.2070	0.1691
HEPMASS IR4	0.6920	0.6153	0.6141	0.6158	0.5988	0.5253	0.6353	0.5360	0.5369	0.5363	0.5426	0.5159	0.7188	0.6636	0.6628	0.6630	0.6892	0.4917
HIGGS IR4	0.2004	0.2663	0.2654	0.2644	0.2541	0.1923	0.0843	0.2883	0.2900	0.2895	0.2956	0.1976	0.3440	0.3743	0.3723	0.3735	0.3496	0.2606
ECBDL'14 10M	0.0000	0.1252	0.1246	0.1250	0.1256	0.0922	0.0000	0.1348	0.1346	0.1351	0.1275	0.1213	0.0717	0.1620	0.1583	0.1617	0.1374	0.1210

(b) AUC

Dataset	BAG						RANF						GBT					
	Gini	WGini	RUS	ROS	SMOTE	ROSE	Gini	WGini	RUS	ROS	SMOTE	ROSE	Gini	WGini	RUS	ROS	SMOTE	ROSE
COVTYPE 1vs4	0.9694	0.9933	0.9931	0.9924	0.9929	0.9848	0.9313	0.9892	0.9901	0.9900	0.9901	0.9286	0.9941	0.9973	0.9971	0.9980	0.9977	0.9847
COVTYPE 1vs3	0.7939	0.8902	0.8898	0.8865	0.8876	0.5001	0.5474	0.8830	0.8816	0.8807	0.8821	0.5000	0.9055	0.9629	0.9605	0.9630	0.9628	0.5001
COVTYPE 1vs2	0.9930	0.9958	0.9955	0.9954	0.9953	0.9877	0.9822	0.9916	0.9919	0.9926	0.9924	0.9064	0.9988	0.9989	0.9989	0.9992	0.9992	0.9877
COVTYPE 0vs4	0.8366	0.9595	0.9588	0.9577	0.9577	0.7580	0.7802	0.9473	0.9484	0.9472	0.9475	0.7576	0.9396	0.9806	0.9786	0.9810	0.9803	0.7580
COVTYPE 0vs3	0.9221	0.9750	0.9749	0.9747	0.9744	0.9386	0.8545	0.9627	0.9621	0.9640	0.9639	0.5471	0.9828	0.9939	0.9924	0.9941	0.9941	0.9386
COVTYPE 0vs2	0.9339	0.9630	0.9628	0.9630	0.9628	0.7580	0.8479	0.9580	0.9582	0.9569	0.9580	0.7630	0.9614	0.9825	0.9822	0.9830	0.9833	0.7580
ECBDL'14 1M	0.5000	0.7043	0.7053	0.6986	0.6641	0.6515	0.5000	0.7103	0.7085	0.7091	0.6751	0.6870	0.5030	0.7270	0.7210	0.7285	0.5638	0.6952
SUSY IR16	0.6264	0.7701	0.7702	0.7680	0.7649	0.7466	0.6125	0.7716	0.7722	0.7718	0.7719	0.7553	0.6641	0.7934	0.7928	0.7934	0.7892	0.7278
SUSY IR4	0.7200	0.7689	0.7693	0.7685	0.7661	0.7495	0.6909	0.7712	0.7720	0.7717	0.7718	0.7572	0.7433	0.7941	0.7937	0.7939	0.7918	0.7357
KDDCUP dos vs r2l	0.9761	0.9984	0.9993	0.9984	0.9989	0.9986	0.9524	0.9997	0.9996	0.9997	0.9999	0.9996	0.9953	0.9973	0.9992	0.9979	0.9975	0.9992
KDDCUP dos vs nor.	0.9997	0.9998	0.9998	0.9998	0.9998	0.9987	0.9996	0.9996	0.9996	0.9996	0.9995	0.9974	1.0000	1.0000	1.0000	1.0000	1.0000	0.9987
HEPMASS IR16	0.7550	0.8343	0.8348	0.8343	0.8336	0.8183	0.5976	0.8214	0.8220	0.8221	0.8223	0.8144	0.7645	0.8626	0.8619	0.8623	0.8506	0.8126
HIGGS IR16	0.5102	0.6669	0.6659	0.6644	0.6526	0.5961	0.5000	0.6774	0.6783	0.6776	0.6714	0.5986	0.5313	0.7265	0.7254	0.7261	0.6989	0.6061
HEPMASS IR4	0.8006	0.8342	0.8331	0.8337	0.8327	0.8171	0.7444	0.8212	0.8218	0.8216	0.8222	0.8139	0.8237	0.8620	0.8620	0.8619	0.8564	0.8053
HIGGS IR4	0.5381	0.6663	0.6659	0.6651	0.6565	0.5987	0.5047	0.6771	0.6779	0.6776	0.6795	0.6017	0.6119	0.7265	0.7256	0.7261	0.7099	0.6202
ECBDL'14 10M	0.5000	0.7000	0.7002	0.6985	0.6709	0.6587	0.5000	0.7108	0.7108	0.7106	0.6834	0.6891	0.5040	0.7462	0.7432	0.7458	0.5896	0.6994

Table 3.4: Experimental results performing resampling before the ensemble training for MCC (a) and AUC (b). The best results for each classifier appear within gray boxes, while the best results overall are within black boxes. GBT showed the best performance for both metrics. For MCC, Gini variants, specially the GBT one, showed the best performance for the majority of the datasets. WGini, RUS, ROS, and SMOTE were all preferable options for AUC, depending on the ensemble in use.

Algorithm	Avg. rank
ROS+GBT	3.0000
GBT (WGini)	3.4375
GBT (Gini)	3.4375
SMOTE+GBT	4.0000
RUS+GBT	5.5625
BAG (Gini)	7.9375
BAG (WGini)	9.1875
ROS+BAG	10.5000
RUS+BAG	10.7500
SMOTE+BAG	11.0625
ROS+RANF	11.0625
RUS+RANF	11.2500
SMOTE+RANF	11.5625
RANF (WGini)	11.5625
RANF (Gini)	11.7500
ROSE+GBT	14.0312
ROSE+RANF	15.3125
ROSE+BAG	15.5938

(a) MCC

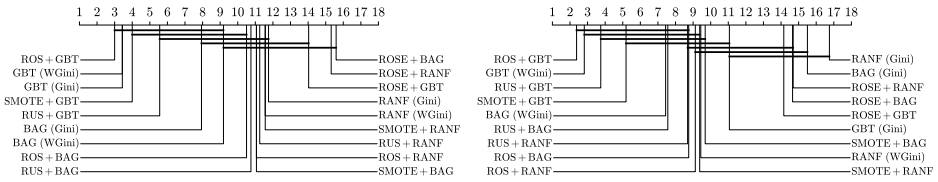
Algorithm	Avg. rank
ROS+GBT	2.3750
GBT (WGini)	2.8125
RUS+GBT	3.7500
SMOTE+GBT	5.1875
BAG (WGini)	7.4375
RUS+BAG	7.5625
RUS+RANF	8.6875
ROS+BAG	8.7500
ROS+RANF	9.1250
SMOTE+RANF	9.3750
RANF (WGini)	9.4375
SMOTE+BAG	9.6875
GBT (Gini)	11.0625
ROSE+GBT	14.1562
ROSE+BAG	14.6562
ROSE+RANF	14.6875
BAG (Gini)	15.5000
RANF (Gini)	16.7500

(b) AUC

Table 3.5: Average ranks of the ensemble classifiers performing resampling before training, according to both the MCC (a) and the AUC (b) metrics. The methods below the dashed line showed significant differences with the best one according to the Hochberg procedure at a confidence level of 95%. ROS+GBT was the best method, followed by GBT (WGini).

Table 3.5 shows the average ranks computed for each metric, the dashed line represents the limit below which the methods differ statistically from the best one at a significance level of 95%. The best method was ROS+GBT for both metrics, followed by GBT (WGini). The AUC metric revealed that some resampling, or at least the use of weighted Gini, contributed to overcoming the imbalance, because regular ensembles (*i.e.*, Gini variants) were ranked extremely low. The methods that showed the worst results were, in general, RANF (Gini) and all the variants involving ROSE.

The results of the Friedman–Nemenyi test can be seen in Figure 3.1. It shows all groups of methods that performed equivalently, providing therefore, additional information to the average rankings discussed above. Also, the number of groups (*i.e.*, from five to seven) could be considered high. Figure 3.1.a shows that SMOTE+GBT, ROS+GBT, and GBT (WGini) performed better than over half of the evaluated methods, which include all RANF variants and three variants of BAG. Regarding the bad performance of the ROSE variants, the test showed that those variants were only worse



(a) MCC (b) AUC

Results and discussion

Figure 3.1: A Friedman–Nemenyi test comparison of the different ensemble classifiers performing resampling before training, according to MCC (a) and AUC (b) metrics. The methods connected with a thick horizontal line show no significant differences between each other at a level of $\alpha = 0.05$. For many methods there is a close statistical equivalence.

than the GBT-based methods. Figure 3.1.b shows that only ROS+GBT and GBT (WGini) performed better than more than half of the tested methods. In contrast to the MCC results, the RANF variants were not so badly ranked by the AUC. Finally, it could be seen that the Gini variants performed worse than their respective WGini variants.

Table 3.6 shows the one-to-one comparisons using the Bayesian hierarchical sign test for each metric and ensemble classifier variant. For the sake of simplicity, instead of showing the ternary heatmap plots, for each comparison the three values (Left, Right, and ROPE) were reported. Left corresponds to the first element under comparison, and Right to the second (*e.g.*, for Gini *vs* WGini comparison, Left corresponds to Gini, and Right to WGini). The best results for each comparison (*i.e.*, a metric and a classifier) appear within black boxes.

For the first set of comparisons, Gini *vs* others, it is remarkable how much it depended on the chosen metric whether the ensemble (using Gini without any resampling) was considered better or not. For example, in BAG and GBT the use of Gini was preferred for MCC (by a narrow margin), whereas the opposite was true for AUC (by a much larger margin). RANF performed in a more consistent way, for all metrics usually the use of resampling or WGini showed itself to be better than the Gini variant. For the second set of comparisons, WGini *vs* ROS, RUS, and SMOTE, Right and Left values were very close, thus no clear winner could be named. Likewise, the comparisons between ROS and RUS were not conclusive and were highly dependent on the ensemble method used and the metric under consideration. The bad performance of ROSE as a resampling method was clearly

Comparison	Metric	BAG			RANF			GBT		
		Left	ROPE	Right	Left	ROPE	Right	Left	ROPE	Right
Gini *vs* WGini	MCC	0.6105	0.0000	0.3895	0.4058	0.0000	0.5942	0.6665	0.0000	0.3335
	AUC	0.1180	0.0000	0.8820	0.0165	0.0000	0.9835	0.0648	0.0002	0.9350
Gini *vs* RUS	MCC	0.7355	0.0000	0.2645	0.4263	0.0000	0.5737	0.7737	0.0000	0.2263
	AUC	0.0695	0.0000	0.9305	0.0030	0.0000	0.9970	0.0845	0.0002	0.9153
Gini *vs* ROS	MCC	0.6868	0.0000	0.3132	0.3832	0.0000	0.6168	0.6155	0.0000	0.3845
	AUC	0.0628	0.0002	0.9370	0.0105	0.0000	0.9895	0.0695	0.0000	0.9305
Gini *vs* SMOTE	MCC	0.7072	0.0000	0.2928	0.4433	0.0000	0.5567	0.6073	0.0000	0.3927
	AUC	0.0985	0.0000	0.9015	0.0255	0.0000	0.9745	0.2112	0.0003	0.7885
Gini *vs* ROSE	MCC	0.9145	0.0000	0.0855	0.9455	0.0000	0.0545	0.9768	0.0000	0.0232
	AUC	0.4535	0.0000	0.5465	0.2827	0.0000	0.7173	0.6402	0.0000	0.3598
WGini *vs* RUS	MCC	0.6723	0.0005	0.3272	0.5642	0.0000	0.4358	0.6230	0.0000	0.3770
	AUC	0.5273	0.0235	0.4492	0.5252	0.0003	0.4745	0.4925	0.0037	0.5038
WGini *vs* ROS	MCC	0.5465	0.0000	0.4535	0.5072	0.0000	0.4928	0.4635	0.0000	0.5365
	AUC	0.5122	0.0000	0.4878	0.4990	0.0005	0.5005	0.4783	0.0195	0.5022
WGini *vs* SMOTE	MCC	0.5302	0.0000	0.4698	0.5002	0.0000	0.4998	0.5042	0.0000	0.4958
	AUC	0.5325	0.0008	0.4667	0.5360	0.0013	0.4627	0.6653	0.0000	0.3347
WGini *vs* ROSE	MCC	0.9657	0.0000	0.0343	0.9585	0.0000	0.0415	0.9550	0.0000	0.0450
	AUC	0.9417	0.0003	0.0580	0.9490	0.0000	0.0510	0.9480	0.0000	0.0520
RUS *vs* ROS	MCC	0.3820	0.0012	0.6168	0.4263	0.0002	0.5735	0.3465	0.0000	0.6535
	AUC	0.5155	0.0092	0.4753	0.5172	0.0248	0.4580	0.4810	0.0210	0.4980
RUS *vs* SMOTE	MCC	0.4173	0.0000	0.5827	0.3957	0.0035	0.6008	0.3200	0.0003	0.6797
	AUC	0.5317	0.0003	0.4680	0.5410	0.0043	0.4547	0.7295	0.0113	0.2592
RUS *vs* ROSE	MCC	0.8760	0.0000	0.1240	0.9698	0.0000	0.0302	0.9515	0.0000	0.0485
	AUC	0.8872	0.0000	0.1128	0.9660	0.0000	0.0340	0.9680	0.0000	0.0320
ROS *vs* SMOTE	MCC	0.4567	0.0000	0.5433	0.6167	0.1293	0.2540	0.4675	0.0000	0.5325
	AUC	0.4940	0.0307	0.4753	0.0813	0.8862	0.0325	0.6912	0.0828	0.2260
ROS *vs* ROSE	MCC	0.9278	0.0000	0.0722	0.9865	0.0000	0.0135	0.9862	0.0000	0.0138
	AUC	0.9172	0.0000	0.0828	0.9828	0.0005	0.0167	0.9888	0.0000	0.0112
SMOTE *vs* ROSE	MCC	0.8992	0.0000	0.1008	0.9597	0.0000	0.0403	0.9908	0.0000	0.0092
	AUC	0.8665	0.0000	0.1335	0.9475	0.0008	0.0517	0.9413	0.0022	0.0565

Table 3.6: One to-one comparison using the Bayesian hierarchical sign test for each metric and classifier variant. The best results for each comparison (metric and classifier) appear within black boxes. Generally, it was unclear whether one variant was better than other. It depended on the specific dataset, metric, and ensemble method. ROSE was the only variant that showed itself to be clearly worse than the rest.

worse than other resampling techniques (RUS, SMOTE, and ROS) and all non-resampling techniques (Gini and WGini). It must be noted that the ROPE region was almost zero for most of the comparisons, which means that the methods hardly performed equally well, but that one method would be better than the other depending on the specific dataset.

Results and discussion

3.5.2 *Resampling within the ensemble*

This subsection collects the results of the ensemble methods that perform the resampling within the ensemble; each base classifier in the ensemble will be trained on a dataset obtained after performing a specific resampling for that classifier. Table 3.7.a details the results of the experiments for MCC, and Table 3.7.b for AUC. As in the previous section, the boxes and the blueness intensity of the cells, are used to highlight the results and provide an enriched representation. As might be expected, for this strategy, GBT was also shown to be better than BAG. Focusing on the results involving resampling (*i.e.*, excluding Gini and WGini variants), for GBT, ROS could be more beneficial, while RUS apparently performed better for BAG. Nevertheless, looking closely at the tables, row-wise blueness intensity is almost the same for RUS, ROS, and RB, which suggests that all the resampling methods performed in a similar way. As stated before, depending on the chosen metric, pre-processing techniques could be discouraged because the Gini variants of the ensembles, apparently performed better. This happens specifically when looking at the MCC metric, but its unacceptable performance with some datasets such as ECBDL and HIGGS should not be forgotten.

In Table 3.8, the average ranks computed for each of the performance metrics are shown. GBT+ROS showed the best results for both metrics, being statistically better than any BAG variant. Looking at AUC, the worst methods were the Gini variants of GBT and BAG. The best methods were GBT variants when resampling was used (ROS, RB and RUS), with no differences between all three. On the other hand, for MCC the worst method was BAG+RB and BAG using the weighted Gini index, leaving it apparently quite clear that resampling techniques performed better than the non-resampling (using Gini or weighted Gini) alternatives.

The results of the Friedman–Nemenyi test are shown in Figure 3.2. As regards the MCC, GBT+ROS and GBT (Gini) performed better than any BAG variant, on the contrary, BAG+RB and BAG (WGini) performed worse than any GBT variant. Thus, it is not clear whether resampling was beneficial or not. Another interesting insight for this metric, is that all GBT

(a) MCC

Dataset	BAG					GBT				
	Gini	WGini	RUS	ROS	RB	Gini	WGini	RUS	ROS	RB
COVTYPE 1vs4	0.9522	0.9438	0.9457	0.9467	0.9445	0.9906	0.9776	0.9690	0.9789	0.9703
COVTYPE 1vs3	0.6753	0.5789	0.5783	0.5786	0.5791	0.8548	0.7897	0.7860	0.8016	0.7801
COVTYPE 1vs2	0.9801	0.9756	0.9753	0.9785	0.9769	0.9975	0.9938	0.9932	0.9952	0.9933
COVTYPE ovs4	0.7408	0.6848	0.6853	0.6863	0.6839	0.9051	0.8036	0.7934	0.8157	0.7941
COVTYPE ovs3	0.8993	0.7829	0.8042	0.7845	0.7958	0.9751	0.9463	0.9385	0.9571	0.9391
COVTYPE ovs2	0.8852	0.8033	0.8039	0.8059	0.8179	0.9317	0.8858	0.8852	0.8956	0.8892
ECBDL'14 1M	0.0000	0.1275	0.1279	0.1276	0.1249	0.0602	0.1555	0.1466	0.1613	0.1469
SUSY IR16	0.4385	0.3061	0.3070	0.3073	0.3100	0.4970	0.3537	0.3513	0.3538	0.3514
SUSY IR4	0.5345	0.4685	0.4698	0.4686	0.4504	0.5796	0.5295	0.5312	0.5314	0.5314
KDDCUP dos vs r2l	0.9757	0.9596	0.4164	0.8699	0.9141	0.9955	0.9885	0.4266	0.9982	0.4388
KDDCUP dos vs nor.	0.9987	0.9991	0.9988	0.9993	0.9992	0.9999	0.9999	0.9998	0.9998	0.9998
HEPMASS IR16	0.6565	0.4319	0.4282	0.4311	0.3847	0.6706	0.4742	0.4751	0.4762	0.4743
HIGGS IR16	0.1158	0.1597	0.1608	0.1604	0.1545	0.1944	0.2306	0.2344	0.2352	0.2344
HEPMASS IR4	0.6920	0.6153	0.6149	0.6160	0.5771	0.7188	0.6636	0.6685	0.6684	0.6686
HIGGS IR4	0.2004	0.2663	0.2683	0.2676	0.2543	0.3440	0.3743	0.3785	0.3793	0.3785
ECBDL'14 10M	0.0000	0.1252	0.1253	0.1253	0.1219	0.0717	0.1620	0.1619	0.1650	0.1619

(b) AUC

Dataset	BAG					GBT				
	Gini	WGini	RUS	ROS	RB	Gini	WGini	RUS	ROS	RB
COVTYPE 1vs4	0.9694	0.9933	0.9935	0.9937	0.9944	0.9941	0.9973	0.9971	0.9979	0.9972
COVTYPE 1vs3	0.7939	0.8902	0.8924	0.8900	0.8973	0.9055	0.9629	0.9652	0.9666	0.9637
COVTYPE 1vs2	0.9930	0.9958	0.9957	0.9963	0.9962	0.9988	0.9989	0.9989	0.9992	0.9989
COVTYPE ovs4	0.8366	0.9595	0.9606	0.9599	0.9611	0.9396	0.9806	0.9806	0.9825	0.9807
COVTYPE ovs3	0.9221	0.9750	0.9772	0.9765	0.9754	0.9828	0.9939	0.9941	0.9953	0.9942
COVTYPE ovs2	0.9339	0.9630	0.9633	0.9630	0.9646	0.9614	0.9825	0.9827	0.9839	0.9831
ECBDL'14 1M	0.5000	0.7043	0.7066	0.7049	0.7030	0.5030	0.7270	0.7288	0.7338	0.7293
SUSY IR16	0.6264	0.7701	0.7710	0.7702	0.7695	0.6641	0.7934	0.7949	0.7953	0.7949
SUSY IR4	0.7200	0.7689	0.7697	0.7695	0.7653	0.7433	0.7941	0.7951	0.7952	0.7951
KDDCUP dos vs r2l	0.9761	0.9984	0.9993	0.9976	0.9979	0.9953	0.9973	0.9989	0.9994	0.9990
KDDCUP dos vs nor.	0.9997	0.9998	0.9997	0.9998	0.9998	1.0000	1.0000	1.0000	1.0000	1.0000
HEPMASS IR16	0.7550	0.8343	0.8357	0.8355	0.8316	0.7645	0.8626	0.8643	0.8645	0.8642
HIGGS IR16	0.5102	0.6669	0.6683	0.6675	0.6577	0.5313	0.7265	0.7299	0.7304	0.7300
HEPMASS IR4	0.8006	0.8342	0.8344	0.8344	0.8318	0.8237	0.8620	0.8639	0.8640	0.8641
HIGGS IR4	0.5381	0.6663	0.6677	0.6672	0.6474	0.6119	0.7265	0.7286	0.7290	0.7287
ECBDL'14 10M	0.5000	0.7000	0.7006	0.7004	0.6990	0.5040	0.7462	0.7476	0.7499	0.7476

Table 3.7: Experimental results performing resampling for each ensemble base classifier for MCC (a) and AUC (b). The best results for each classifier and the best overall results appear within gray and black boxes, respectively. GBT showed the best performance for both metrics. For MCC, Gini variants showed the best performance for the majority of the datasets. With regard to the AUC, RUS and ROS were preferable for BAG and GBT, respectively.

Algorithm	Avg. rank
GBT+ROS	2.1875
GBT (Gini)	2.5625
GBT (WGini)	3.6250
GBT+RB	3.9375
GBT+RUS	4.6875
BAG (Gini)	6.1250
BAG+ROS	7.3125
BAG+RUS	7.9375
BAG (WGini)	8.3125
BAG+RB	8.3125

(a) MCC

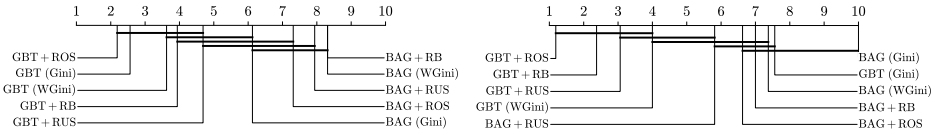
Algorithm	Avg. rank
GBT+ROS	1.1875
GBT+RB	2.3750
GBT+RUS	3.0625
GBT (WGini)	4.0000
BAG+RUS	5.8125
BAG+ROS	6.6250
BAG+RB	7.0000
BAG (WGini)	7.3750
GBT (Gini)	7.5625
BAG (Gini)	10.0000

(b) AUC

Table 3.8: Average ranks of the ensemble classifiers performing resampling for each base classifier, according to the MCC (a) and the AUC (b) metrics. The methods below the dashed line were statistically different from the best one according to the Hochberg procedure at a confidence level of 95%. GBT+ROS was the best method, being statistically better than any BAG variant.

variants were considered equivalent and the same could be said for all BAG variants. Regarding the AUC metric, the use of resampling improved the performance. GBT+ROS and GBT+RB were better than any BAG variant and the Gini variant of GBT. Moreover, GBT (Gini) and BAG (Gini) performed worse than any other GBT variant. For both metrics, five groups of equivalent-performing classifiers were shown, also, the size of the groups ranged from three to five methods so, the statistical differences between most of the methods were pretty small.

Finally, regarding the one-to-one comparisons, Table 3.9 shows the Bayesian hierarchical sign test for each pair of methods belonging to this part of the study. It should be noted that the Gini *vs* WGini comparison was not reported here, because already appears in Table 3.6. In view of the comparisons between Gini *vs* resampling (RUS, ROS, and RB), a clear discrepancy is shown between the MCC and the AUC metrics for both ensemble families (BAG and GBT). In view of MCC, the Gini variants performed better than the resampling-based variants by a narrow margin. On the contrary, the resampling-based variants clearly performed better for the AUC metric (*i.e.*, Right region values were close to 1). Using weighted Gini, for BAG ensembles could be sufficient as it has shown itself to be marginally better than applying any kind of resampling. Nevertheless, it was unclear whether resampling might be beneficial for GBT ensembles, as the differ-



(a) MCC (b) AUC

Figure 3.2: A Friedman–Nemenyi test comparison of the different ensemble classifiers performing resampling for each base classifier, according to MCC (a) and AUC (b) metrics. The methods connected with a thick horizontal line showed no significant differences between each other (at a level of $\alpha = 0.05$). It was not clear whether resampling was beneficial or not for the MCC. The use of resampling improved AUC performance.

ences were minimal and opted for one option or the other depending on the metric in use. Finally, regarding the comparisons between resampling methods (RUS, ROS, and RB), no clear winner was found either for the BAG or for the GBT ensembles. The differences that could led to the choice of one method or the other were very small, and thus, the decision will depend on the specific dataset and metric.

3.5.3 *Comparing the two strategies*

Having separately tested the performance of the two strategies, a comparison of all the previous methods will be reported in this subsection.

In Table 3.10, the average ranks of all the algorithms of the study are shown. GBT+ROS was the best method, followed by most of the other alternatives of GBT which were considered statistically equivalent according to the Hochberg procedure at a confidence interval of 95% (for MCC, RUS+GBT was statistically worse; while for AUC, SMOTE+GBT was statistically worse). The worst results were for ROSE (for all the ensembles) and RANF (without resampling) that were ranked lowest in the tables.

Figure 3.3 shows the performance of the methods by using average ranks with one-to-one comparisons: one metric on each axis. The marker aspect represents: the resampling strategy (shape), the ensemble method (color), and the two resampling strategies (unfilled means resampling before training, and filled means resampling within the ensemble). Two clusters could be found, one contained most of the GBT variants on the best positions (left lower corner), while the other contained RANF and BAG variants, which were located farther to the right. Differences between RANF and BAG were

Comparison	Metric	BAG			GBT		
		Left	ROPE	Right	Left	ROPE	Right
Gini *vs* RUS	MCC	0.6950	0.0000	0.3050	0.7278	0.0000	0.2722
	AUC	0.1430	0.0000	0.8570	0.0575	0.0000	0.9425
Gini *vs* ROS	MCC	0.6218	0.0000	0.3782	0.5615	0.0000	0.4385
	AUC	0.0725	0.0002	0.9273	0.0755	0.0000	0.9245
Gini *vs* RB	MCC	0.6160	0.0000	0.3840	0.7155	0.0000	0.2845
	AUC	0.0833	0.0000	0.9167	0.1235	0.0005	0.8760
WGini *vs* RUS	MCC	0.6557	0.0000	0.3443	0.5727	0.0000	0.4273
	AUC	0.5027	0.0015	0.4958	0.4460	0.0027	0.5513
WGini *vs* ROS	MCC	0.4765	0.0000	0.5235	0.4840	0.0000	0.5160
	AUC	0.4998	0.0232	0.4770	0.4830	0.0350	0.4820
WGini *vs* RB	MCC	0.5242	0.0000	0.4758	0.5865	0.0000	0.4135
	AUC	0.5458	0.0025	0.4517	0.4835	0.0095	0.5070
RUS *vs* ROS	MCC	0.3788	0.0000	0.6212	0.3357	0.0000	0.6643
	AUC	0.5135	0.0002	0.4863	0.4677	0.0225	0.5098
RUS *vs* RB	MCC	0.4283	0.0000	0.5717	0.5125	0.0000	0.4875
	AUC	0.5222	0.0000	0.4778	0.5092	0.0005	0.4903
ROS *vs* RB	MCC	0.5057	0.0000	0.4943	0.6233	0.0000	0.3767
	AUC	0.5157	0.0000	0.4843	0.4907	0.0005	0.5088

Table 3.9: Bayesian hierarchical sign tests comparing different resampling methods (RUS, ROS, and RB) applied within the ensemble (BAG and GBT), according to the MCC and AUC metrics. Resampling methods were also compared to base ensembles using Gini and Weighted Gini impurities. Depending on the specific metric, resampling-based ensembles would be recommended or not. Which resampling method is better, is unclear.

Algorithm	Avg. rank
GBT+ROS	2.9375
ROS+GBT	4.5625
GBT (Gini)	4.6250
GBT (WGini)	5.0625
SMOTE+GBT	5.6250
GBT+RB	5.8750
GBT+RUS	6.7500
RUS+GBT	8.6250
BAG (Gini)	11.0000
BAG+ROS	12.8125
BAG (WGini)	13.9375
BAG+RUS	14.0000
BAG+RB	14.6250
ROS+BAG	15.5625
ROS+RANF	15.6875
RUS+RANF	15.9375
RUS+BAG	16.0000
SMOTE+BAG	16.2500
RANF (WGini)	16.3125
RANF (Gini)	16.3750
SMOTE+RANF	16.4375
ROSE+GBT	19.0312
ROSE+RANF	20.7500
ROSE+BAG	21.2188

(a) MCC

Algorithm	Avg. rank
GBT+ROS	1.8125
GBT+RB	3.3750
GBT+RUS	4.1875
ROS+GBT	4.7500
GBT (WGini)	5.6875
RUS+GBT	6.6875
SMOTE+GBT	8.1250
BAG+RUS	10.3750
BAG+ROS	11.5625
BAG (WGini)	12.5625
RUS+BAG	12.6875
BAG+RB	12.8750
RUS+RANF	13.1875
ROS+RANF	13.6250
RANF (WGini)	13.9375
ROS+BAG	14.1875
SMOTE+RANF	14.2500
SMOTE+BAG	15.1875
GBT (Gini)	16.0000
ROSE+GBT	19.8438
ROSE+RANF	20.3125
ROSE+BAG	20.5312
BAG (Gini)	21.5000
RANF (Gini)	22.7500

(b) AUC

Table 3.10: Average ranks of all ensemble classifiers evaluated in this study, according to the MCC and AUC metrics. The statistical differences between the methods below the dashed line and the best method were significant, according to the Hochberg procedure at a confidence level of 95%. GBT-based ensembles showed themselves to be statistically better than BAG and RANF ensembles.

not so clear, but in general the BAG variants tended to perform better than the RANF ones. The differences between the metrics were remarkable for some methods, especially for the Gini variants (represented with circular markers), which were positioned far away from the diagonal line. This finding offers an interesting insight, insofar as the use of resampling may be not beneficial for the performance of some classifiers according to some metrics. For this reason, the use of several metrics is advisable and may even be crucial when drawing proper conclusions on imbalance within Big Data environments.

The rankings suggested to us that resampling before training had a fairly similar performance to resampling for each base classifier within the ensem-

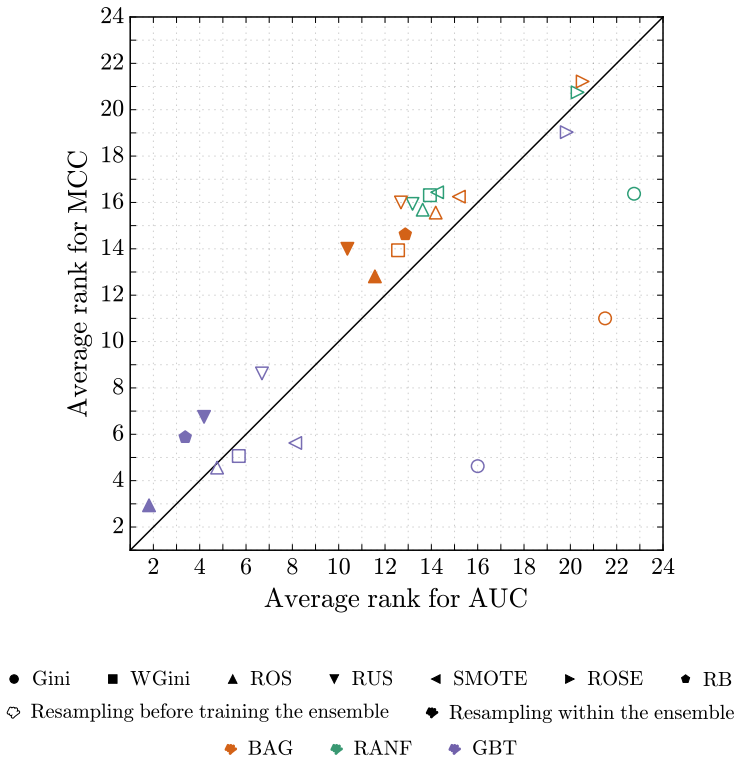
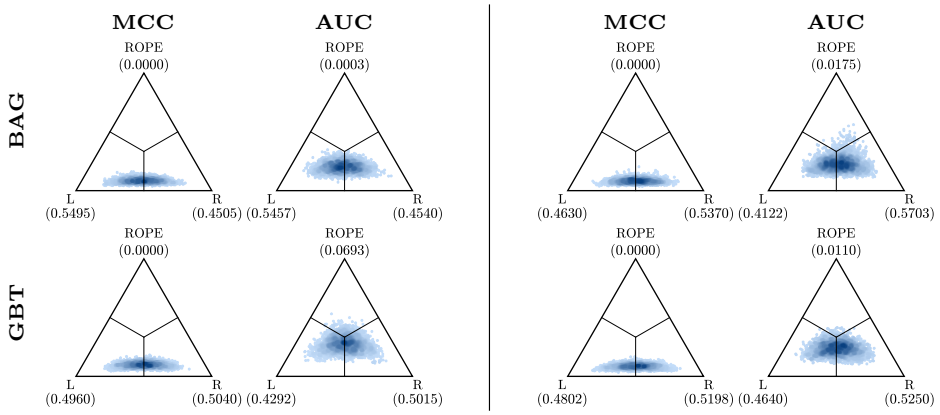


Figure 3.3: Average ranks for all ensemble methods according to MCC and AUC metrics. The shape of the marker represents the resampling strategy, the color of the marker represents the ensemble method, and the fill of the marker represents the two resampling strategies: before training (unfilled), and within the ensemble (filled). In general, GBT variants showed better performance than RANF and BAG variants.

ble. This intuitive evaluation may be contrasted in Figure 3.4, which represents several Bayesian hierarchical sign tests compared to the performance of resampling before training the ensemble (L) with the performance of resampling within the ensemble (R). The application of RUS (Figure 3.4.a) before training, for BAG ensembles, showed a slightly better performance. On the contrary, for GBT ensembles, applying RUS at each iteration of the ensemble marginally outperformed its application once before training. Finally, ROS (Figure 3.4.b) showed that it was moderately better when used for each classifier within the ensemble than when used to obtain a balanced dataset with which to train all the classifiers in the ensemble. Irrespective



(a) RUS **(b)** ROS

Figure 3.4: Bayesian hierarchical sign test heatmaps showing the influence of RUS and ROS resampling before training BAG (top row) and GBT (bottom row) ensembles (L), compared with the influence of RUS and ROS resampling on each iteration of the ensemble (R). Each column represents one metric. There was no clear winner, as the each approach performed better than the others as a function of the specific dataset.

of the detailed analysis presented above, the overall idea, as the clouds of points were almost centered and situated outside the ROPE region, was that no clear winner could be named. Neither could a similar performance between strategies be noted. Therefore, depending on the specific dataset one approach will be better than the other and vice-versa.

3.5.4 *Execution time analysis*

In this study, two popular families of ensembles were evaluated: Bagging and Boosting. Their intrinsic differences make their execution times significantly different. As is well known, execution times are crucial in Big Data environments. With this in mind, the training and prediction times of the different ensemble algorithms and resampling techniques for the ECBDL14'10M dataset (see Table 3.3) are presented in this section. Figure 3.5 shows a bar plot with the training and prediction times for all ensemble methods that were tested. On the left hand-side of the plot, orange bars depict prediction times in microseconds (μs). On the right-hand side,

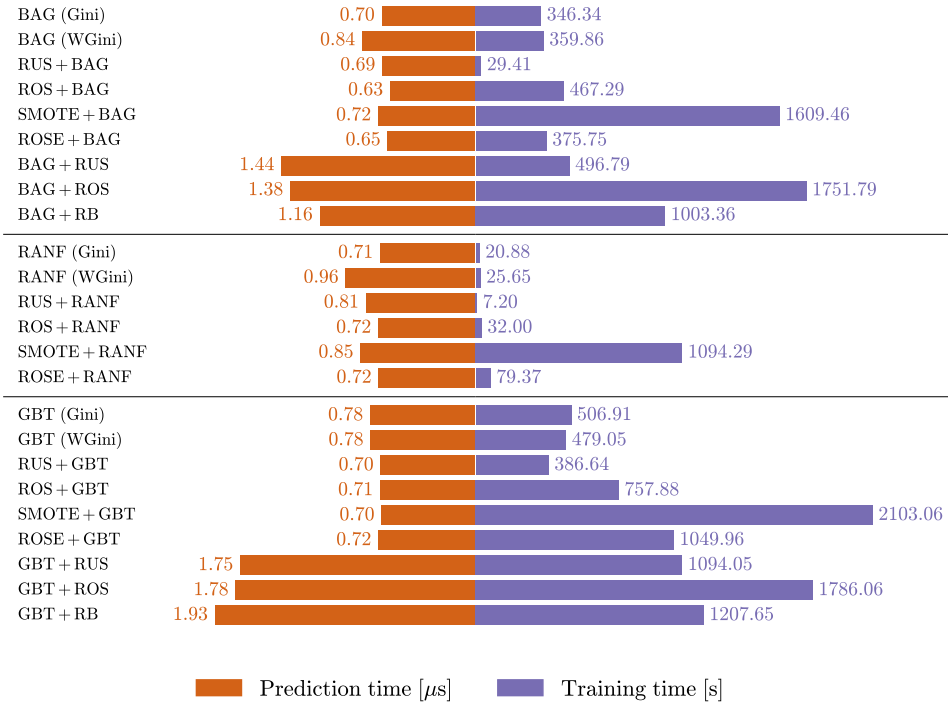


Figure 3.5: Comparison of execution times for ECBDL14'10M dataset. Orange bars (on the left) depict the prediction time in microseconds (μs). Purple bars (on the right) depict the training time in seconds (s). RANF showed itself to be the fastest ensemble method, and RUS, the fastest resampling strategy. GBT ensembles and SMOTE resampling strategies, were the two slowest choices. Also, resampling before training the ensemble was faster than resampling within the ensemble.

purple bars depict training times in seconds (s). BAG, RANF, and GBT ensembles (from top to bottom) were grouped with black horizontal lines.

In training, the fastest method was Random Forest trained with a dataset subsampled by RUS, followed by Random Forest without resampling (Gini and WGini variants). This result was predictable, because RUS decreases the number of instances in the dataset (speeding up the training process), and Random Forest builds decision trees using feature subsets. The approaches that apply resampling within the ensemble (BAG+RUS, BAG+ROS, BAG+RB; and GBT+RUS, GBT+ROS, GBT+RB) were computationally more expensive than those that apply it before training, which was therefore reflected by their execution times. The slowest method was

SMOTE+GBT. It is well known that SMOTE is a time-consuming method, because of the kNN computation, and its combination with boosting makes it a less recommendable alternative for Big Data in terms of training time.

In prediction, the differences between the tested methods were much smaller. This result is because, once all the trees of an ensemble are built, voting and prediction is almost the same, regardless of the ensemble family. The resampling before training methods were the fastest, followed by the methods that involved training each base classifier with a different resample.

3.5.5 *Discussion*

The experimental results revealed that GBT outperformed Bagging and Random Forest for imbalanced Big Data classification, which is in line with what happens with normal-sized datasets, where Boosting has traditionally outperformed Bagging. More specifically, the combination of ROS within the GBT training (*i.e.*, GBT+ROS) was the best method for all the measures. The main drawback of GBT is the much lengthier time taken for training than Bagging approaches (*e.g.*, RUS+GBT was around 13 times slower than RUS+BAG). Also, GBT scalability in Big Data was limited, because Boosting performs an iterative process, that cannot be fully parallelized.

Overall, resampling for each base classifier in the ensemble had a better performance than resampling only once before training. Nevertheless, the differences were not as clear as might be expected, in view of the Hochberg procedure that showed their statistical equivalence, and the Bayesian tests that showed quite balanced distributions for both alternatives, revealing that one strategy could outperform the other on around 50% of occasions.

SMOTE is a powerful but computationally expensive method, which appears not to be as good in Big Data as it is in normal-sized datasets. Something similar happens with ROSE that, although it has proven its validity with normal-sized datasets, clearly showed a worse performance with datasets of greater size. In other words, methods that generate synthetic examples were unable to outperform ROS, which is simpler and less computationally intensive. Our intuitive understanding of this unexpected behavior is that, although the proportion of instances in minority classes remains unfavorable in Big Data, it seems that the increase in these instances in large datasets is enough to make introducing new synthetic instances no beneficial.

It is worth noting that algorithm performance differs greatly depending on the metric that is used. Special attention should be paid to studies and

experimentation, because the use of one metric or another might lead to completely different conclusions. For this reason, the use of several imbalance metrics is highly recommended within Big Data environments, just as it is for normal-sized datasets.

Conclusions and future work

With regard to the MCC, ensemble methods without balancing the datasets (*i.e.*, using the Gini index) were quite competitive, specially BAG, which was ranked better than its combination with any resampling technique. Nevertheless, it has to be noted that for some datasets, such as ECBDL14 and HIGGS, an ensemble trained without any balancing technique is the alternative that clearly performed worst of all. Bearing this in mind, and knowing that Gini variants achieved very poor results for AUC metric, we can affirm that balancing techniques can actually contribute functional methods to mitigate the problem of imbalance within Big Data. Whenever no resampling techniques are used, we highly encourage the use of impurity indexes, at the least, that take into account the imbalance, such as the weighted Gini index. The use of weighted Gini has demonstrated itself in this study to be a reasonably good solution for imbalanced Big Data: it is fast and straightforward to apply by instance weighting.

3.6 CONCLUSIONS AND FUTURE WORK

Although there are numerous studies on the classification of imbalanced data, these studies are practically non-existent in the context of imbalanced Big Data classification. This paper has shed light on the impact of using data-level approaches within Big Data ensemble classification. Those approaches mainly involve the use of pre-processing techniques such as RUS, ROS, SMOTE, or ROSE for transforming an imbalanced dataset into a balanced one. Whether rebalancing is better performed once only before training the ensemble, or as many times as there are base classifiers contained in the ensemble, was also evaluated. All the experiments were performed on Bagging-based and Boosting-based ensembles, highlighting how resampling techniques specifically affect both ensemble families, in terms of performance. The conclusions of the study, although some might appear ambiguous, can be very useful to help guide future research work into imbalance in Big Data classification.

Within the experimental framework, Boosting ensembles, although requiring more computational power, clearly outperformed Bagging-based alternatives. The use on the trees construction of an impurity index that takes into account the imbalance, such as weighted Gini, offered roughly

equivalent results to the use of resampling techniques. Surprisingly, the training of the ensembles on the original datasets without any change (using the standard Gini index), offered quite good results overall (for MCC and F_1-score metrics). However, this procedure is not advisable, because the results were dreadful for some datasets (clearly visible when AUC or G-mean were used), but still an accurate indicator of a lack of robust solutions for improving the performance for all the metrics. Regarding the resampling methods, ROS generally achieved better results, but with only a minimal advantage, closely followed by RUS and SMOTE with no statistically significant differences. In contrast, ROSE was clearly the worst alternative. Our conclusion is therefore that complex methods that involve the generation of synthetic instances are not as effective for Big Data as they are for normal-sized datasets. Although they could, depending on the dataset, be the best option, in general we discourage their use in favor of simpler and faster methods such as ROS.

Ensembles specifically designed to overcome the imbalance problem (*i.e.*, resampling before training each base classifier), achieved better performance than resampling a dataset once and then training a conventional ensemble with it. Nevertheless, the differences between the two strategies were very small, suggesting that whether one strategy is actually better than the other will strongly depend on the dataset to which it is applied. Therefore, whenever execution times are considered critical, as it is often the case with Big Data, the general recommendation would be to use the faster strategies based on a single initial resampling or the use of impurity indexes that take into account imbalance (*e.g.*, weighted Gini).

An interesting insight is the importance of using different evaluation metrics when dealing with imbalance problems. This is preferable, because each metric uses the values within the confusion matrix in a specific way and therefore has its own strengths and weaknesses. Hence, using more than one metric yields a more informed view of the results and a better evaluation of the performance of a single classifier. The conclusions that can be drawn by using MCC and F_1-score are very different than using AUC or G-mean, thus at least one metric of each group should be used for future Big Data imbalance studies (in addition, it is enough to use one from each group, since the conclusions that can be obtained are similar for the two metrics within each group).

Despite the advances relating to the classification of Big Data in recent years, research into imbalanced Big Data is still scarce and more studies and surveys are needed to unify the publications that periodically emerge.

Conclusions and future work

More precisely, the presence of the pathologies that are traditionally associated with imbalanced datasets, such as overlapping, noisy examples, small disjuncts, and borderline instances (Díez-Pastor et al., 2015a), have yet to be studied in Big Data.

The resampling in this study perfectly balanced the datasets with 50% of the examples belonging to each class (with the exception of RB, for which the imbalanced ratios were random for each base classifier). Another interesting future line of research would be the evaluation of how different resampling ratios affect different classifiers.

In view of the results given by the data-level approaches for imbalanced Big Data learning, we place the focus on the exploration of algorithm-level approaches, which should assist more fruitful advances for these kinds of problems. An interesting research line could be the evaluation of novel forest-based approaches that have recently emerged, such as Random Forest quantile classifier (O'Brien and Ishwaran, 2019) and the adaptation of Oblique Random Forest (Katuwal et al., 2020), for imbalanced learning, specially for large datasets. The design and implementation of new classifiers for Big Data frameworks, such as Apache Spark, is a promising research line nowadays, as is the adaptation and revalidation of any popular proposal for normal-sized datasets.

ACKNOWLEDGMENTS

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 (JCyL/-FEDER, UE) co-financed through European Union FEDER funds, and by the *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.

APPENDIX A. FULL RESULTS

This appendix gathers the results of the experimentation performed in the paper for all the measures that were taken.

Dataset	BAG						RANF						GBT					
	Gini	WGini	RUS	ROS	SMOTE	ROSE	Gini	WGini	RUS	ROS	SMOTE	ROSE	Gini	WGini	RUS	ROS	SMOTE	ROSE
COVTYPE 1vs4	0.9557	0.9470	0.9455	0.9350	0.9426	0.9025	0.9238	0.9268	0.9269	0.9276	0.9284	0.5359	0.9913	0.9791	0.9706	0.9815	0.9806	0.9012
COVTYPE 1vs3	0.6913	0.5899	0.5787	0.5757	0.5673	0.1623	0.1732	0.5282	0.5191	0.5180	0.5218	0.1622	0.8654	0.7974	0.7740	0.7994	0.8029	0.1623
COVTYPE 1vs2	0.9830	0.9790	0.9771	0.9780	0.9774	0.9522	0.9792	0.9710	0.9725	0.9733	0.9711	0.6432	0.9979	0.9947	0.9940	0.9964	0.9964	0.9520
COVTYPE ovs4	0.7503	0.6715	0.6589	0.6592	0.6670	0.2021	0.6756	0.6449	0.6447	0.6443	0.6524	0.2018	0.9100	0.8012	0.7777	0.8075	0.8093	0.2021
COVTYPE ovs3	0.9034	0.7815	0.7862	0.7741	0.7744	0.5623	0.8187	0.7458	0.7510	0.7491	0.7386	0.1378	0.9767	0.9490	0.9249	0.9519	0.9513	0.5628
COVTYPE ovs2	0.8975	0.8160	0.8121	0.8135	0.8135	0.3428	0.7964	0.8129	0.8059	0.8048	0.8102	0.3474	0.9392	0.8948	0.8908	0.8967	0.9010	0.3428
ECBDL'14 1M	0.0000	0.0912	0.0898	0.0880	0.1028	0.0645	0.0000	0.0980	0.0973	0.0974	0.1069	0.0887	0.0120	0.1142	0.0988	0.1140	0.1372	0.0814
SUSY IR16	0.3893	0.3064	0.3073	0.2999	0.2906	0.3563	0.3596	0.3093	0.3133	0.3126	0.2990	0.3573	0.4696	0.3533	0.3513	0.3539	0.3440	0.4848
SUSY IR4	0.5877	0.5833	0.5841	0.5817	0.5770	0.5902	0.5422	0.5885	0.5916	0.5903	0.5844	0.5963	0.6297	0.6298	0.6287	0.6292	0.6240	0.6133
KDDCUP dos vs r2l	0.9753	0.9596	0.3093	0.8565	0.7676	0.2643	0.9500	0.8095	0.4421	0.8746	0.8178	0.4570	0.9951	0.9889	0.2914	0.9938	0.9956	0.2686
KDDCUP dos vs nor.	0.9990	0.9993	0.9991	0.9994	0.9994	0.9962	0.9984	0.9983	0.9982	0.9982	0.9982	0.9957	1.0000	0.9999	0.9999	0.9999	0.9999	0.9964
HEPMASS IR16	0.6480	0.4279	0.4108	0.4323	0.4592	0.3095	0.3258	0.3161	0.3161	0.3164	0.3301	0.2949	0.6640	0.4644	0.4611	0.4639	0.5522	0.2765
HIGGS IR16	0.0406	0.1816	0.1820	0.1795	0.1820	0.1770	0.0000	0.1967	0.1976	0.1968	0.2022	0.1797	0.1180	0.2352	0.2336	0.2348	0.2252	0.2245
HEPMASS IR4	0.7300	0.6957	0.6948	0.6962	0.6822	0.6154	0.6497	0.6249	0.6255	0.6250	0.6319	0.6064	0.7599	0.7326	0.7319	0.7321	0.7532	0.5795
HIGGS IR4	0.1514	0.4388	0.4373	0.4377	0.4326	0.3591	0.0189	0.4547	0.4558	0.4555	0.4598	0.3639	0.3691	0.5128	0.5113	0.5122	0.4967	0.3914
ECBDL'14 10M	0.0000	0.0907	0.0898	0.0913	0.1058	0.0662	0.0000	0.0970	0.0968	0.0976	0.1019	0.0916	0.0160	0.1123	0.1093	0.1122	0.1558	0.0853

Table 3.11: Experimental results (F_1-score) performing resampling before ensemble training. The best results for each classifier appear within gray boxes, while the best overall results appear within black boxes. Gini variants, specially the GBT one, showed the best performance for the majority of the datasets.

Dataset	BAG						RANF						GBT					
	Gini	WGini	RUS	ROS	SMOTE	ROSE	Gini	WGini	RUS	ROS	SMOTE	ROSE	Gini	WGini	RUS	ROS	SMOTE	ROSE
COVTYPE 1vs4	0.9522	0.9438	0.9422	0.9314	0.9392	0.8976	0.9206	0.9227	0.9229	0.9235	0.9244	0.5602	0.9906	0.9776	0.9686	0.9801	0.9791	0.8963
COVTYPE 1vs3	0.6753	0.5789	0.5696	0.5655	0.5592	0.0043	0.2925	0.5263	0.5186	0.5173	0.5210	0.0000	0.8548	0.7897	0.7675	0.7917	0.7947	0.0043
COVTYPE 1vs2	0.9801	0.9756	0.9734	0.9745	0.9738	0.9447	0.9759	0.9663	0.9680	0.9689	0.9665	0.6207	0.9975	0.9938	0.9930	0.9959	0.9958	0.9445
COVTYPE ovs4	0.7408	0.6848	0.6741	0.6739	0.6803	0.2409	0.6734	0.6577	0.6580	0.6572	0.6640	0.2405	0.9051	0.8036	0.7822	0.8095	0.8108	0.2409
COVTYPE ovs3	0.8993	0.7829	0.7870	0.7762	0.7763	0.5864	0.8184	0.7471	0.7514	0.7504	0.7413	0.0835	0.9751	0.9463	0.9218	0.9493	0.9487	0.5868
COVTYPE ovs2	0.8852	0.8033	0.7995	0.8008	0.8008	0.3267	0.7828	0.7987	0.7921	0.7905	0.7961	0.3325	0.9317	0.8858	0.8816	0.8878	0.8921	0.3267
ECBDL'14 1M	0.0000	0.1275	0.1269	0.1226	0.1203	0.0883	0.0000	0.1353	0.1340	0.1344	0.1282	0.1180	0.0602	0.1555	0.1405	0.1560	0.1169	0.1166
SUSY IR16	0.4385	0.3061	0.3068	0.3002	0.2917	0.3334	0.4284	0.3091	0.3123	0.3117	0.3017	0.3381	0.4970	0.3537	0.3519	0.3541	0.3445	0.4524
SUSY IR4	0.5345	0.4685	0.4695	0.4665	0.4604	0.4830	0.5174	0.4752	0.4793	0.4775	0.4705	0.4884	0.5796	0.5295	0.5281	0.5287	0.5221	0.5569
KDDCUP dos vs r2l	0.9757	0.9596	0.4275	0.8650	0.7890	0.3895	0.9508	0.8245	0.5325	0.8818	0.8316	0.5442	0.9955	0.9885	0.4128	0.9938	0.9956	0.3936
KDDCUP dos vs nor.	0.9987	0.9991	0.9989	0.9993	0.9993	0.9952	0.9980	0.9979	0.9978	0.9978	0.9977	0.9946	0.9999	0.9999	0.9998	0.9999	0.9999	0.9954
HEPMASS IR16	0.6565	0.4319	0.4190	0.4353	0.4562	0.3364	0.4255	0.3428	0.3431	0.3433	0.3529	0.3244	0.6706	0.4742	0.4714	0.4737	0.5406	0.3121
HIGGS IR16	0.1158	0.1597	0.1591	0.1569	0.1497	0.1168	0.0000	0.1750	0.1761	0.1751	0.1746	0.1203	0.1944	0.2306	0.2289	0.2301	0.2070	0.1691
HEPMASS IR4	0.6920	0.6153	0.6141	0.6158	0.5988	0.5253	0.6353	0.5360	0.5369	0.5363	0.5426	0.5159	0.7188	0.6636	0.6628	0.6630	0.6892	0.4917
HIGGS IR4	0.2004	0.2663	0.2654	0.2644	0.2541	0.1923	0.0843	0.2883	0.2900	0.2895	0.2956	0.1976	0.3440	0.3743	0.3723	0.3735	0.3496	0.2606
ECBDL'14 10M	0.0000	0.1252	0.1246	0.1250	0.1256	0.0922	0.0000	0.1348	0.1346	0.1351	0.1275	0.1213	0.0717	0.1620	0.1583	0.1617	0.1374	0.1210

Table 3.12: Experimental results (MCC) performing resampling before the ensemble training. The best results for each classifier appear within gray boxes, while the best overall results appear within black boxes. Gini variants, specially the GBT one, showed the best performance for the majority of the datasets.

Dataset	BAG						RANF						GBT					
	Gini	WGini	RUS	ROS	SMOTE	ROSE	Gini	WGini	RUS	ROS	SMOTE	ROSE	Gini	WGini	RUS	ROS	SMOTE	ROSE
COVTYPE 1vs4	0.9690	0.9933	0.9931	0.9924	0.9929	0.9848	0.9288	0.9892	0.9901	0.9900	0.9901	0.9260	0.9941	0.9973	0.9971	0.9980	0.9977	0.9847
COVTYPE 1vs3	0.7702	0.8902	0.8898	0.8865	0.8875	0.0142	0.3081	0.8824	0.8807	0.8798	0.8813	0.0000	0.9013	0.9629	0.9604	0.9630	0.9628	0.0142
COVTYPE 1vs2	0.9930	0.9958	0.9955	0.9954	0.9953	0.9877	0.9821	0.9916	0.9919	0.9926	0.9924	0.9015	0.9988	0.9989	0.9989	0.9992	0.9992	0.9877
COVTYPE 0vs4	0.8221	0.9594	0.9586	0.9575	0.9575	0.7184	0.7505	0.9472	0.9483	0.9471	0.9475	0.7178	0.9379	0.9805	0.9785	0.9810	0.9802	0.7184
COVTYPE 0vs3	0.9190	0.9749	0.9748	0.9746	0.9743	0.9374	0.8423	0.9627	0.9621	0.9639	0.9638	0.3068	0.9826	0.9939	0.9924	0.9941	0.9941	0.9374
COVTYPE 0vs2	0.9322	0.9629	0.9626	0.9628	0.9627	0.7184	0.8356	0.9580	0.9581	0.9568	0.9579	0.7252	0.9608	0.9825	0.9821	0.9829	0.9833	0.7184
ECBDL'14 1M	0.0000	0.7042	0.7048	0.6983	0.6476	0.6257	0.0000	0.7101	0.7083	0.7090	0.6612	0.6868	0.0778	0.7254	0.7209	0.7272	0.3837	0.6914
SUSY IR16	0.5056	0.7690	0.7691	0.7672	0.7644	0.7331	0.4761	0.7705	0.7708	0.7705	0.7715	0.7446	0.5760	0.7915	0.7911	0.7915	0.7876	0.6873
SUSY IR4	0.6773	0.7681	0.7684	0.7678	0.7654	0.7378	0.6275	0.7701	0.7706	0.7705	0.7713	0.7483	0.7086	0.7923	0.7920	0.7922	0.7903	0.7000
KDDCUP dos vs r2l	0.9758	0.9984	0.9993	0.9984	0.9989	0.9986	0.9512	0.9997	0.9996	0.9997	0.9999	0.9996	0.9953	0.9973	0.9992	0.9979	0.9975	0.9992
KDDCUP dos vs nor.	0.9997	0.9998	0.9998	0.9998	0.9998	0.9987	0.9996	0.9996	0.9996	0.9996	0.9995	0.9974	1.0000	1.0000	1.0000	1.0000	1.0000	0.9987
HEPMASS IR16	0.7157	0.8329	0.8341	0.8327	0.8308	0.8165	0.4422	0.8200	0.8205	0.8206	0.8218	0.8111	0.7288	0.8621	0.8615	0.8618	0.8462	0.8052
HIGGS IR16	0.1444	0.6642	0.6638	0.6613	0.6526	0.5493	0.0000	0.6774	0.6783	0.6776	0.6702	0.5525	0.2543	0.7264	0.7253	0.7260	0.6979	0.5210
HEPMASS IR4	0.7804	0.8328	0.8316	0.8322	0.8321	0.8149	0.7020	0.8197	0.8202	0.8200	0.8213	0.8106	0.8094	0.8616	0.8616	0.8615	0.8538	0.7955
HIGGS IR4	0.2898	0.6622	0.6591	0.6610	0.6565	0.5541	0.0977	0.6770	0.6779	0.6775	0.6794	0.5587	0.4948	0.7265	0.7255	0.7260	0.7097	0.5606
ECBDL'14 10M	0.0000	0.7000	0.7001	0.6985	0.6554	0.6373	0.0000	0.7107	0.7108	0.7105	0.6761	0.6883	0.0899	0.7462	0.7431	0.7458	0.4532	0.6980

Table 3.13: Experimental results (G-mean) performing resampling before ensemble training. The best results for each classifier appear within gray boxes, while the best overall results appear within black boxes. WGini, RUS, ROS or SMOTE were preferable depending on the ensemble in use.

Dataset	BAG						RANF						GBT					
	Gini	WGini	RUS	ROS	SMOTE	ROSE	Gini	WGini	RUS	ROS	SMOTE	ROSE	Gini	WGini	RUS	ROS	SMOTE	ROSE
COVTYPE 1vs4	0.9694	0.9933	0.9931	0.9924	0.9929	0.9848	0.9313	0.9892	0.9901	0.9900	0.9901	0.9286	0.9941	0.9973	0.9971	0.9980	0.9977	0.9847
COVTYPE 1vs3	0.7939	0.8902	0.8898	0.8865	0.8876	0.5001	0.5474	0.8830	0.8816	0.8807	0.8821	0.5000	0.9055	0.9629	0.9605	0.9630	0.9628	0.5001
COVTYPE 1vs2	0.9930	0.9958	0.9955	0.9954	0.9953	0.9877	0.9822	0.9916	0.9919	0.9926	0.9924	0.9064	0.9988	0.9989	0.9989	0.9992	0.9992	0.9877
COVTYPE 0vs4	0.8366	0.9595	0.9588	0.9577	0.9577	0.7580	0.7802	0.9473	0.9484	0.9472	0.9475	0.7576	0.9396	0.9806	0.9786	0.9810	0.9803	0.7580
COVTYPE 0vs3	0.9221	0.9750	0.9749	0.9747	0.9744	0.9386	0.8545	0.9627	0.9621	0.9640	0.9639	0.5471	0.9828	0.9939	0.9924	0.9941	0.9941	0.9386
COVTYPE 0vs2	0.9339	0.9630	0.9628	0.9630	0.9628	0.7580	0.8479	0.9580	0.9582	0.9569	0.9580	0.7630	0.9614	0.9825	0.9822	0.9830	0.9833	0.7580
ECBDL'14 1M	0.5000	0.7043	0.7053	0.6986	0.6641	0.6515	0.5000	0.7103	0.7085	0.7091	0.6751	0.6870	0.5030	0.7270	0.7210	0.7285	0.5638	0.6952
SUSY IR16	0.6264	0.7701	0.7702	0.7680	0.7649	0.7466	0.6125	0.7716	0.7722	0.7718	0.7719	0.7553	0.6641	0.7934	0.7928	0.7934	0.7892	0.7278
SUSY IR4	0.7200	0.7689	0.7693	0.7685	0.7661	0.7495	0.6909	0.7712	0.7720	0.7717	0.7718	0.7572	0.7433	0.7941	0.7937	0.7939	0.7918	0.7357
KDDCUP dos vs r2l	0.9761	0.9984	0.9993	0.9984	0.9989	0.9986	0.9524	0.9997	0.9996	0.9997	0.9999	0.9996	0.9953	0.9973	0.9992	0.9979	0.9975	0.9992
KDDCUP dos vs nor.	0.9997	0.9998	0.9998	0.9998	0.9998	0.9987	0.9996	0.9996	0.9996	0.9996	0.9995	0.9974	1.0000	1.0000	1.0000	1.0000	1.0000	0.9987
HEPMASS IR16	0.7550	0.8343	0.8348	0.8343	0.8336	0.8183	0.5976	0.8214	0.8220	0.8221	0.8223	0.8144	0.7645	0.8626	0.8619	0.8623	0.8506	0.8126
HIGGS IR16	0.5102	0.6669	0.6659	0.6644	0.6526	0.5961	0.5000	0.6774	0.6783	0.6776	0.6714	0.5986	0.5313	0.7265	0.7254	0.7261	0.6989	0.6061
HEPMASS IR4	0.8006	0.8342	0.8331	0.8337	0.8327	0.8171	0.7444	0.8212	0.8218	0.8216	0.8222	0.8139	0.8237	0.8620	0.8620	0.8619	0.8564	0.8053
HIGGS IR4	0.5381	0.6663	0.6659	0.6651	0.6565	0.5987	0.5047	0.6771	0.6779	0.6776	0.6795	0.6017	0.6119	0.7265	0.7256	0.7261	0.7099	0.6202
ECBDL'14 10M	0.5000	0.7000	0.7002	0.6985	0.6709	0.6587	0.5000	0.7108	0.7108	0.7106	0.6834	0.6891	0.5040	0.7462	0.7432	0.7458	0.5896	0.6994

Table 3.14: Experimental results (AUC) performing resampling before ensemble training. The best results for each classifier appear within gray boxes, while the best overall results appear within black boxes. WGini, RUS, ROS or SMOTE were preferable depending on the ensemble in use.

Algorithm	Avg. rank
SMOTE+GBT	3.0000
ROS+GBT	3.1875
GBT (WGini)	3.4375
GBT (Gini)	4.8125
RUS+GBT	5.7500
BAG (Gini)	8.7500
BAG (WGini)	9.3750
SMOTE+BAG	10.0000
ROS+BAG	10.5000
RUS+BAG	10.8125
SMOTE+RANF	10.9375
ROS+RANF	11.1875
RUS+RANF	11.5000
RANF (WGini)	11.5625
RANF (Gini)	12.5000
ROSE+GBT	13.9062
ROSE+RANF	14.5625
ROSE+BAG	15.2188

(a) F_1-score

Algorithm	Avg. rank
ROS+GBT	3.0000
GBT (WGini)	3.4375
GBT (Gini)	3.4375
SMOTE+GBT	4.0000
RUS+GBT	5.5625
BAG (Gini)	7.9375
BAG (WGini)	9.1875
ROS+BAG	10.5000
RUS+BAG	10.7500
SMOTE+BAG	11.0625
ROS+RANF	11.0625
RUS+RANF	11.2500
SMOTE+RANF	11.5625
RANF (WGini)	11.5625
RANF (Gini)	11.7500
ROSE+GBT	14.0312
ROSE+RANF	15.3125
ROSE+BAG	15.5938

(b) MCC

Algorithm	Avg. rank
ROS+GBT	2.4375
GBT (WGini)	2.7500
RUS+GBT	3.7500
SMOTE+GBT	5.1875
BAG (WGini)	7.3750
RUS+BAG	7.6875
RUS+RANF	8.7500
ROS+BAG	8.7500
ROS+RANF	9.0625
SMOTE+RANF	9.1875
RANF (WGini)	9.3750
SMOTE+BAG	9.5625
GBT (Gini)	11.5625
ROSE+GBT	14.3438
ROSE+BAG	14.4688
ROSE+RANF	14.5000
BAG (Gini)	15.5000
RANF (Gini)	16.7500

(c) G-mean

Algorithm	Avg. rank
ROS+GBT	2.3750
GBT (WGini)	2.8125
RUS+GBT	3.7500
SMOTE+GBT	5.1875
BAG (WGini)	7.4375
RUS+BAG	7.5625
RUS+RANF	8.6875
ROS+BAG	8.7500
ROS+RANF	9.1250
SMOTE+RANF	9.3750
RANF (WGini)	9.4375
SMOTE+BAG	9.6875
GBT (Gini)	11.0625
ROSE+GBT	14.1562
ROSE+BAG	14.6562
ROSE+RANF	14.6875
BAG (Gini)	15.5000
RANF (Gini)	16.7500

(d) AUC

Table 3.15: Average ranks of the ensemble classifiers performing resampling before training, according to the F_1-score (a), the MCC (b), the G-mean (c), and the AUC (d) metrics. The methods below the dashed line showed significant differences alongside the best one according to the Hochberg procedure at a confidence level of 95%. Except for F_1-score where SMOTE+GBT was top ranked, ROS+GBT was the best method followed by GBT (WGini)

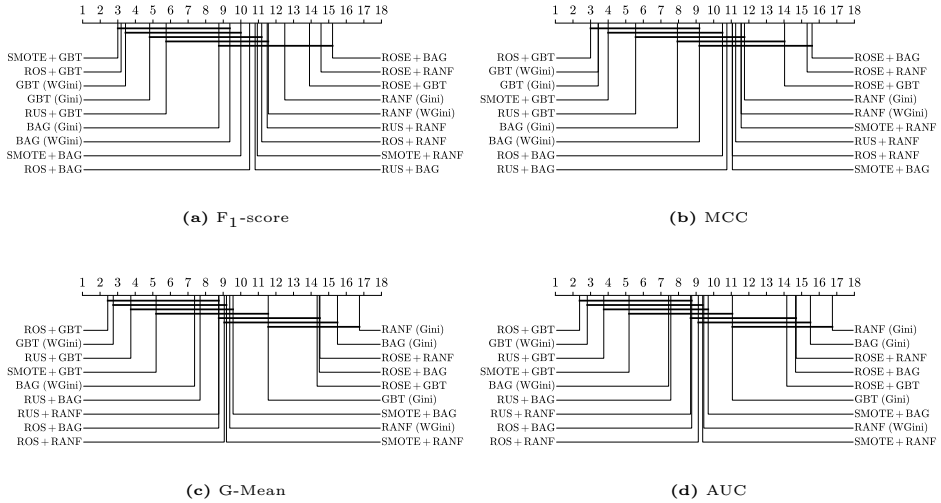


Figure 3.6: A Friedman-Nemenyi test comparison of the different ensemble classifiers performing resampling before training, according to the F_1-score (a), the MCC (b), the G-mean (c), and the AUC (d) metrics. Those methods connected with a thick horizontal line have no significant differences between them (at a level of $\alpha = 0.05$). Close statistical equivalence between many methods may be noted.

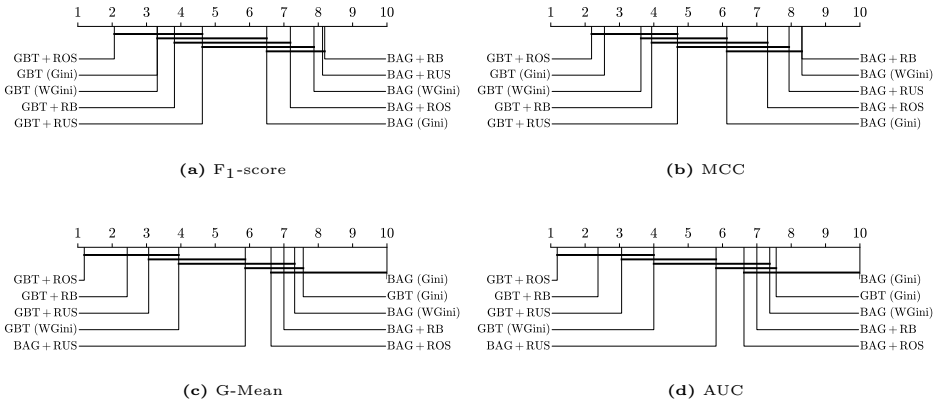
Comparison	Metric	BAG			RANF			GBT		
		Left	ROPE	Right	Left	ROPE	Right	Left	ROPE	Right
Gini *vs* WGini	F_1-score	0.5720	0.0000	0.4280	0.2967	0.0000	0.7033	0.5797	0.0000	0.4203
	MCC	0.6105	0.0000	0.3895	0.4058	0.0000	0.5942	0.6665	0.0000	0.3335
	G-mean	0.0395	0.0000	0.9605	0.0060	0.0000	0.9940	0.0343	0.0000	0.9657
	AUC	0.1180	0.0000	0.8820	0.0165	0.0000	0.9835	0.0648	0.0002	0.9350
Gini *vs* RUS	F_1-score	0.6693	0.0000	0.3307	0.3230	0.0000	0.6770	0.7370	0.0000	0.2630
	MCC	0.7355	0.0000	0.2645	0.4263	0.0000	0.5737	0.7737	0.0000	0.2263
	G-mean	0.0212	0.0000	0.9788	0.0008	0.0000	0.9992	0.0483	0.0000	0.9517
	AUC	0.0695	0.0000	0.9305	0.0030	0.0000	0.9970	0.0845	0.0002	0.9153
Gini *vs* ROS	F_1-score	0.6243	0.0000	0.3757	0.2565	0.0000	0.7435	0.5440	0.0000	0.4560
	MCC	0.6868	0.0000	0.3132	0.3832	0.0000	0.6168	0.6155	0.0000	0.3845
	G-mean	0.0312	0.0000	0.9688	0.0025	0.0000	0.9975	0.0350	0.0000	0.9650
	AUC	0.0628	0.0002	0.9370	0.0105	0.0000	0.9895	0.0695	0.0000	0.9305
Gini *vs* SMOTE	F_1-score	0.6280	0.0000	0.3720	0.3330	0.0000	0.6670	0.5127	0.0000	0.4873
	MCC	0.7072	0.0000	0.2928	0.4433	0.0000	0.5567	0.6073	0.0000	0.3927
	G-mean	0.0425	0.0000	0.9575	0.0052	0.0000	0.9948	0.1005	0.0000	0.8995
	AUC	0.0985	0.0000	0.9015	0.0255	0.0000	0.9745	0.2112	0.0003	0.7885
Gini *vs* ROSE	F_1-score	0.8823	0.0000	0.1177	0.8965	0.0000	0.1035	0.9695	0.0000	0.0305
	MCC	0.9145	0.0000	0.0855	0.9455	0.0000	0.0545	0.9768	0.0000	0.0232
	G-mean	0.2498	0.0000	0.7502	0.1178	0.0000	0.8822	0.4673	0.0000	0.5327
	AUC	0.4535	0.0000	0.5465	0.2827	0.0000	0.7173	0.6402	0.0000	0.3598
WGini *vs* RUS	F_1-score	0.6877	0.0015	0.3108	0.5877	0.0000	0.4123	0.6365	0.0000	0.3635
	MCC	0.6723	0.0005	0.3272	0.5642	0.0000	0.4358	0.6230	0.0000	0.3770
	G-mean	0.5315	0.0220	0.4465	0.5108	0.0002	0.4890	0.4940	0.0022	0.5038
	AUC	0.5273	0.0235	0.4492	0.5252	0.0003	0.4745	0.4925	0.0037	0.5038
WGini *vs* ROS	F_1-score	0.5473	0.0000	0.4527	0.5025	0.0000	0.4975	0.4663	0.0000	0.5337
	MCC	0.5465	0.0000	0.4535	0.5072	0.0000	0.4928	0.4635	0.0000	0.5365
	G-mean	0.5127	0.0000	0.4873	0.5137	0.0000	0.4863	0.4775	0.0213	0.5012
	AUC	0.5122	0.0000	0.4878	0.4990	0.0005	0.5005	0.4783	0.0195	0.5022
WGini *vs* SMOTE	F_1-score	0.5137	0.0000	0.4863	0.4908	0.0000	0.5092	0.4740	0.0000	0.5260
	MCC	0.5302	0.0000	0.4698	0.5002	0.0000	0.4998	0.5042	0.0000	0.4958
	G-mean	0.5457	0.0003	0.4540	0.5288	0.0012	0.4700	0.7298	0.0000	0.2702
	AUC	0.5325	0.0008	0.4667	0.5360	0.0013	0.4627	0.6653	0.0000	0.3347
WGini *vs* ROSE	F_1-score	0.9575	0.0000	0.0425	0.9497	0.0000	0.0503	0.9490	0.0000	0.0510
	MCC	0.9657	0.0000	0.0343	0.9585	0.0000	0.0415	0.9550	0.0000	0.0450
	G-mean	0.9527	0.0000	0.0473	0.9527	0.0003	0.0470	0.9595	0.0000	0.0405
	AUC	0.9417	0.0003	0.0580	0.9490	0.0000	0.0510	0.9480	0.0000	0.0520
RUS *vs* ROS	F_1-score	0.3382	0.0008	0.6610	0.3708	0.0002	0.6290	0.2965	0.0000	0.7035
	MCC	0.3820	0.0012	0.6168	0.4263	0.0002	0.5735	0.3465	0.0000	0.6535
	G-mean	0.5222	0.0055	0.4723	0.5208	0.0187	0.4605	0.4885	0.0145	0.4970
	AUC	0.5155	0.0092	0.4753	0.5172	0.0248	0.4580	0.4810	0.0210	0.4980
RUS *vs* SMOTE	F_1-score	0.3850	0.0000	0.6150	0.3468	0.0002	0.6530	0.2550	0.0000	0.7450
	MCC	0.4173	0.0000	0.5827	0.3957	0.0035	0.6008	0.3200	0.0003	0.6797
	G-mean	0.5510	0.0000	0.4490	0.5735	0.0037	0.4228	0.8080	0.0020	0.1900
	AUC	0.5317	0.0003	0.4680	0.5410	0.0043	0.4547	0.7295	0.0113	0.2592
RUS *vs* ROSE	F_1-score	0.8625	0.0000	0.1375	0.9848	0.0000	0.0152	0.9535	0.0000	0.0465
	MCC	0.8760	0.0000	0.1240	0.9698	0.0000	0.0302	0.9515	0.0000	0.0485
	G-mean	0.9120	0.0000	0.0880	0.9712	0.0005	0.0283	0.9692	0.0000	0.0308
	AUC	0.8872	0.0000	0.1128	0.9660	0.0000	0.0340	0.9680	0.0000	0.0320
ROS *vs* SMOTE	F_1-score	0.4360	0.0000	0.5640	0.5358	0.0005	0.4637	0.3458	0.0067	0.6475
	MCC	0.4567	0.0000	0.5433	0.6167	0.1293	0.2540	0.4675	0.0000	0.5325
	G-mean	0.5250	0.0268	0.4482	0.1048	0.8502	0.0450	0.8200	0.0092	0.1708
	AUC	0.4940	0.0307	0.4753	0.0813	0.8862	0.0325	0.6912	0.0828	0.2260
ROS *vs* ROSE	F_1-score	0.9320	0.0000	0.0680	0.9825	0.0000	0.0175	0.9818	0.0000	0.0182
	MCC	0.9278	0.0000	0.0722	0.9865	0.0000	0.0135	0.9862	0.0000	0.0138
	G-mean	0.9490	0.0000	0.0510	0.9732	0.0003	0.0265	0.9812	0.0000	0.0188
	AUC	0.9172	0.0000	0.0828	0.9828	0.0005	0.0167	0.9888	0.0000	0.0112
SMOTE *vs* ROSE	F_1-score	0.8945	0.0000	0.1055	0.9718	0.0000	0.0282	0.9912	0.0000	0.0088
	MCC	0.8992	0.0000	0.1008	0.9597	0.0000	0.0403	0.9908	0.0000	0.0092
	G-mean	0.8990	0.0000	0.1010	0.9313	0.0012	0.0675	0.9035	0.0002	0.0963
	AUC	0.8665	0.0000	0.1335	0.9475	0.0008	0.0517	0.9413	0.0022	0.0565

Table 3.16: One-to-one comparison using Bayesian hierarchical sign test for each metric and classifier variant. The best results for each comparison (metric and classifier) appear within black boxes. Generally, it was unclear whether one variant was better than other. It depended on the specific dataset, metric, and ensemble method. ROSE was the only variant that showed itself to be clearly worse than the rest.

Full results

Dataset	BAG					GBT				
	Gini	WGini	RUS	ROS	RB	Gini	WGini	RUS	ROS	RB
COVTYPE 1vs4	0.9557	0.9470	0.9489	0.9499	0.9477	0.9913	0.9791	0.9710	0.9804	0.9722
COVTYPE 1vs3	0.6913	0.5899	0.5878	0.5897	0.5857	0.8654	0.7974	0.7925	0.8090	0.7866
COVTYPE 1vs2	0.9830	0.9790	0.9787	0.9815	0.9801	0.9979	0.9947	0.9942	0.9959	0.9942
COVTYPE ovs4	0.7503	0.6715	0.6715	0.6731	0.6696	0.9100	0.8012	0.7897	0.8140	0.7905
COVTYPE ovs3	0.9034	0.7815	0.8047	0.7828	0.7959	0.9767	0.9490	0.9413	0.9594	0.9419
COVTYPE ovs2	0.8975	0.8160	0.8166	0.8188	0.8308	0.9392	0.8948	0.8943	0.9043	0.8980
ECBDL'14 1M	0.0000	0.0912	0.0905	0.0911	0.0885	0.0120	0.1142	0.1023	0.1178	0.1024
SUSY IR16	0.3893	0.3064	0.3069	0.3079	0.3120	0.4696	0.3533	0.3491	0.3521	0.3492
SUSY IR4	0.5877	0.5833	0.5842	0.5832	0.5684	0.6297	0.6298	0.6311	0.6312	0.6312
KDDCUP dos vs r2l	0.9753	0.9596	0.2959	0.8623	0.9102	0.9951	0.9889	0.3085	0.9983	0.3234
KDDCUP dos vs nor.	0.9990	0.9993	0.9990	0.9994	0.9994	1.0000	0.9999	0.9999	0.9999	0.9999
HEPMASS IR16	0.6480	0.4279	0.4223	0.4262	0.3672	0.6640	0.4644	0.4644	0.4658	0.4635
HIGGS IR16	0.0406	0.1816	0.1817	0.1821	0.1844	0.1180	0.2352	0.2376	0.2384	0.2377
HEPMASS IR4	0.7300	0.6957	0.6954	0.6963	0.6625	0.7599	0.7326	0.7364	0.7364	0.7365
HIGGS IR4	0.1514	0.4388	0.4394	0.4391	0.4277	0.3691	0.5128	0.5159	0.5165	0.5159
ECBDL'14 10M	0.0000	0.0907	0.0905	0.0906	0.0868	0.0160	0.1123	0.1115	0.1141	0.1115

Table 3.17: Experimental results (F_1-score) performing resampling for each ensemble base classifier. The best results for each classifier appear within gray boxes, while the best overall results appear within black boxes. Gini variants showed the best performance for the majority of the datasets.



(a) F_1-score **(b)** MCC

(c) G-Mean **(d)** AUC

Figure 3.7: A Friedman-Nemenyi test comparison of the different ensemble classifiers performing resampling for each base classifier, according to the F_1-score (a), the MCC (b), the G-mean (c), and the AUC (d) metrics. The methods connected with a thick horizontal line showed no significant differences between each other (at a level of $\alpha = 0.05$). In view of (a) and (b), it is not clear whether resampling was beneficial or not. For (c) and (d), the use of resampling improved the performance.

Dataset	BAG					GBT				
	Gini	WGini	RUS	ROS	RB	Gini	WGini	RUS	ROS	RB
COVTYPE 1vs4	0.9522	0.9438	0.9457	0.9467	0.9445	0.9906	0.9776	0.9690	0.9789	0.9703
COVTYPE 1vs3	0.6753	0.5789	0.5783	0.5786	0.5791	0.8548	0.7897	0.7860	0.8016	0.7801
COVTYPE 1vs2	0.9801	0.9756	0.9753	0.9785	0.9769	0.9975	0.9938	0.9932	0.9952	0.9933
COVTYPE ovs4	0.7408	0.6848	0.6853	0.6863	0.6839	0.9051	0.8036	0.7934	0.8157	0.7941
COVTYPE ovs3	0.8993	0.7829	0.8042	0.7845	0.7958	0.9751	0.9463	0.9385	0.9571	0.9391
COVTYPE ovs2	0.8852	0.8033	0.8039	0.8059	0.8179	0.9317	0.8858	0.8852	0.8956	0.8892
ECBDL'14 1M	0.0000	0.1275	0.1279	0.1276	0.1249	0.0602	0.1555	0.1466	0.1613	0.1469
SUSY IR16	0.4385	0.3061	0.3070	0.3073	0.3100	0.4970	0.3537	0.3513	0.3538	0.3514
SUSY IR4	0.5345	0.4685	0.4698	0.4686	0.4504	0.5796	0.5295	0.5312	0.5314	0.5314
KDDCUP dos vs r2l	0.9757	0.9596	0.4164	0.8699	0.9141	0.9955	0.9885	0.4266	0.9982	0.4388
KDDCUP dos vs nor.	0.9987	0.9991	0.9988	0.9993	0.9992	0.9999	0.9999	0.9998	0.9998	0.9998
HEPMASS IR16	0.6565	0.4319	0.4282	0.4311	0.3847	0.6706	0.4742	0.4751	0.4762	0.4743
HIGGS IR16	0.1158	0.1597	0.1608	0.1604	0.1545	0.1944	0.2306	0.2344	0.2352	0.2344
HEPMASS IR4	0.6920	0.6153	0.6149	0.6160	0.5771	0.7188	0.6636	0.6685	0.6684	0.6686
HIGGS IR4	0.2004	0.2663	0.2683	0.2676	0.2543	0.3440	0.3743	0.3785	0.3793	0.3785
ECBDL'14 10M	0.0000	0.1252	0.1253	0.1253	0.1219	0.0717	0.1620	0.1619	0.1650	0.1619

Table 3.18: Experimental results (MCC) performing resampling for each ensemble base classifier. The best results for each classifier appear within gray boxes, while the best overall results appear within black boxes. Gini variants showed the best performance for the majority of the datasets.

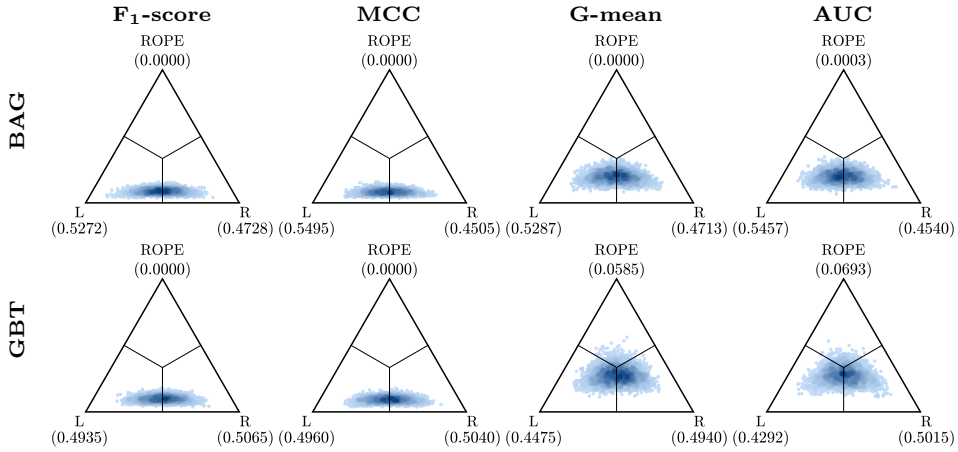


Figure 3.8: Bayesian hierarchical sign test heatmaps showing the influence of RUS before training BAG (top row) and GBT (bottom row) ensembles (L), compared with the influence of RUS on each iteration of the ensemble (R). Each column represents one metric (F_1-score, MCC, G-mean, and AUC). There was no clear winner, depending on the specific dataset and metric, one approach was better than the other and vice-versa.

Full results

Dataset	BAG					GBT				
	Gini	WGini	RUS	ROS	RB	Gini	WGini	RUS	ROS	RB
COVTYPE 1vs4	0.9690	0.9933	0.9935	0.9937	0.9944	0.9941	0.9973	0.9971	0.9979	0.9972
COVTYPE 1vs3	0.7702	0.8902	0.8924	0.8900	0.8972	0.9013	0.9629	0.9651	0.9666	0.9636
COVTYPE 1vs2	0.9930	0.9958	0.9957	0.9963	0.9962	0.9988	0.9989	0.9989	0.9992	0.9989
COVTYPE ovs4	0.8221	0.9594	0.9605	0.9598	0.9609	0.9379	0.9805	0.9805	0.9824	0.9807
COVTYPE ovs3	0.9190	0.9749	0.9771	0.9764	0.9753	0.9826	0.9939	0.9941	0.9953	0.9941
COVTYPE ovs2	0.9322	0.9629	0.9632	0.9629	0.9645	0.9608	0.9825	0.9827	0.9838	0.9830
ECBDL'14 1M	0.0000	0.7042	0.7061	0.7047	0.7023	0.0778	0.7254	0.7287	0.7323	0.7293
SUSY IR16	0.5056	0.7690	0.7700	0.7691	0.7680	0.5760	0.7915	0.7935	0.7938	0.7936
SUSY IR4	0.6773	0.7681	0.7689	0.7688	0.7652	0.7086	0.7923	0.7934	0.7935	0.7934
KDDCUP dos vs r2l	0.9758	0.9984	0.9993	0.9976	0.9979	0.9953	0.9973	0.9989	0.9994	0.9990
KDDCUP dos vs nor.	0.9997	0.9998	0.9997	0.9998	0.9998	1.0000	1.0000	1.0000	1.0000	1.0000
HEPMASS IR16	0.7157	0.8329	0.8346	0.8343	0.8316	0.7288	0.8621	0.8639	0.8641	0.8638
HIGGS IR16	0.1444	0.6642	0.6651	0.6649	0.6577	0.2543	0.7264	0.7298	0.7303	0.7299
HEPMASS IR4	0.7804	0.8328	0.8332	0.8330	0.8318	0.8094	0.8616	0.8635	0.8636	0.8637
HIGGS IR4	0.2898	0.6622	0.6619	0.6617	0.6397	0.4948	0.7265	0.7286	0.7289	0.7286
ECBDL'14 10M	0.0000	0.7000	0.7005	0.7004	0.6983	0.0899	0.7462	0.7475	0.7499	0.7475

Table 3.19: Experimental results (G-mean) performing resampling for each ensemble base classifier. The best results for each classifier appear within gray boxes, while the best overall results appear within black boxes. RUS and ROS were preferable for BAG and GBT respectively.

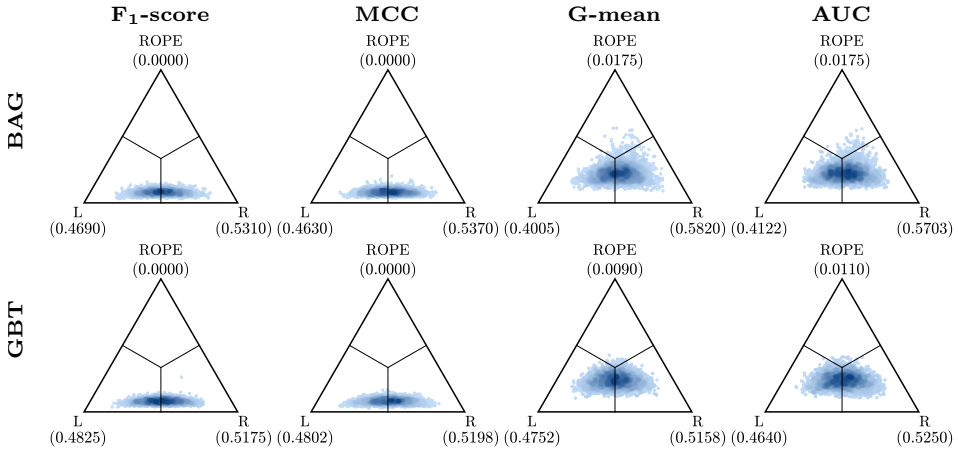


Figure 3.9: Bayesian hierarchical sign test heatmaps showing the influence of ROS before training, BAG (top row) and GBT (bottom row) ensembles (L) compared with the influence of ROS on each iteration of the ensemble (R). Each column represent one metric: (F_1-score, MCC, G-mean, and AUC). There was no clear winner, one approach was better than the other and *vice-versa* dependent on the specific dataset and metric.

Dataset	BAG					GBT				
	Gini	WGini	RUS	ROS	RB	Gini	WGini	RUS	ROS	RB
COVTYPE 1vs4	0.9694	0.9933	0.9935	0.9937	0.9944	0.9941	0.9973	0.9971	0.9979	0.9972
COVTYPE 1vs3	0.7939	0.8902	0.8924	0.8900	0.8973	0.9055	0.9629	0.9652	0.9666	0.9637
COVTYPE 1vs2	0.9930	0.9958	0.9957	0.9963	0.9962	0.9988	0.9989	0.9989	0.9992	0.9989
COVTYPE ovs4	0.8366	0.9595	0.9606	0.9599	0.9611	0.9396	0.9806	0.9806	0.9825	0.9807
COVTYPE ovs3	0.9221	0.9750	0.9772	0.9765	0.9754	0.9828	0.9939	0.9941	0.9953	0.9942
COVTYPE ovs2	0.9339	0.9630	0.9633	0.9630	0.9646	0.9614	0.9825	0.9827	0.9839	0.9831
ECBDL'14 1M	0.5000	0.7043	0.7066	0.7049	0.7030	0.5030	0.7270	0.7288	0.7338	0.7293
SUSY IR16	0.6264	0.7701	0.7710	0.7702	0.7695	0.6641	0.7934	0.7949	0.7953	0.7949
SUSY IR4	0.7200	0.7689	0.7697	0.7695	0.7653	0.7433	0.7941	0.7951	0.7952	0.7951
KDDCUP dos vs r2l	0.9761	0.9984	0.9993	0.9976	0.9979	0.9953	0.9973	0.9989	0.9994	0.9990
KDDCUP dos vs nor.	0.9997	0.9998	0.9997	0.9998	0.9998	1.0000	1.0000	1.0000	1.0000	1.0000
HEPMASS IR16	0.7550	0.8343	0.8357	0.8355	0.8316	0.7645	0.8626	0.8643	0.8645	0.8642
HIGGS IR16	0.5102	0.6669	0.6683	0.6675	0.6577	0.5313	0.7265	0.7299	0.7304	0.7300
HEPMASS IR4	0.8006	0.8342	0.8344	0.8344	0.8318	0.8237	0.8620	0.8639	0.8640	0.8641
HIGGS IR4	0.5381	0.6663	0.6677	0.6672	0.6474	0.6119	0.7265	0.7286	0.7290	0.7287
ECBDL'14 10M	0.5000	0.7000	0.7006	0.7004	0.6990	0.5040	0.7462	0.7476	0.7499	0.7476

Table 3.20: Experimental results (AUC) performing resampling for each ensemble base classifier. The best results for each classifier appear within gray boxes, while the best overall results appear within black boxes. RUS and ROS were preferable for BAG and GBT respectively.

Algorithm	Avg. rank
GBT+ROS	2.0625
GBT (Gini)	3.3125
GBT (WGini)	3.3125
GBT+RB	3.8125
GBT+RUS	4.6250
BAG (Gini)	6.5000
BAG+ROS	7.1875
BAG (WGini)	7.8750
BAG+RUS	8.1250
BAG+RB	8.1875

(a) F_1-score

Algorithm	Avg. rank
GBT+ROS	2.1875
GBT (Gini)	2.5625
GBT (WGini)	3.6250
GBT+RB	3.9375
GBT+RUS	4.6875
BAG (Gini)	6.1250
BAG+ROS	7.3125
BAG+RUS	7.9375
BAG (WGini)	8.3125
BAG+RB	8.3125

(b) MCC

Algorithm	Avg. rank
GBT+ROS	1.1875
GBT+RB	2.4375
GBT+RUS	3.0625
GBT (WGini)	3.9375
BAG+RUS	5.8750
BAG+ROS	6.6250
BAG+RB	7.0000
BAG (WGini)	7.3125
GBT (Gini)	7.5625
BAG (Gini)	10.0000

(c) G-mean

Algorithm	Avg. rank
GBT+ROS	1.1875
GBT+RB	2.3750
GBT+RUS	3.0625
GBT (WGini)	4.0000
BAG+RUS	5.8125
BAG+ROS	6.6250
BAG+RB	7.0000
BAG (WGini)	7.3750
GBT (Gini)	7.5625
BAG (Gini)	10.0000

(d) AUC

Table 3.21: Average ranks of the ensemble classifiers performing resampling for each base classifier, according to the F_1-score (a), the MCC (b), the G-mean (c), and the AUC (d) metrics. The methods below the dashed line are statistically different than the best one according to the Hochberg procedure at a confidence level of 95%. GBT+ROS was the best method and was statistically better than any BAG variant.

Full results

Comparison	Metric	BAG			GBT		
		Left	ROPE	Right	Left	ROPE	Right
Gini *vs* RUS	F_1-score	0.6402	0.0000	0.3598	0.6935	0.0000	0.3065
	MCC	0.6950	0.0000	0.3050	0.7278	0.0000	0.2722
	G-mean	0.0610	0.0000	0.9390	0.0337	0.0000	0.9663
	AUC	0.1430	0.0000	0.8570	0.0575	0.0000	0.9425
Gini *vs* ROS	F_1-score	0.5565	0.0000	0.4435	0.5010	0.0000	0.4990
	MCC	0.6218	0.0000	0.3782	0.5615	0.0000	0.4385
	G-mean	0.0333	0.0000	0.9667	0.0440	0.0000	0.9560
	AUC	0.0725	0.0002	0.9273	0.0755	0.0000	0.9245
Gini *vs* RB	F_1-score	0.5533	0.0000	0.4467	0.6753	0.0000	0.3247
	MCC	0.6160	0.0000	0.3840	0.7155	0.0000	0.2845
	G-mean	0.0350	0.0000	0.9650	0.0693	0.0007	0.9300
	AUC	0.0833	0.0000	0.9167	0.1235	0.0005	0.8760
WGini *vs* RUS	F_1-score	0.6628	0.0000	0.3372	0.6148	0.0000	0.3852
	MCC	0.6557	0.0000	0.3443	0.5727	0.0000	0.4273
	G-mean	0.5100	0.0005	0.4895	0.4475	0.0018	0.5507
	AUC	0.5027	0.0015	0.4958	0.4460	0.0027	0.5513
WGini *vs* ROS	F_1-score	0.4597	0.0000	0.5403	0.4882	0.0000	0.5118
	MCC	0.4765	0.0000	0.5235	0.4840	0.0000	0.5160
	G-mean	0.4965	0.0185	0.4850	0.4848	0.0332	0.4820
	AUC	0.4998	0.0232	0.4770	0.4830	0.0350	0.4820
WGini *vs* RB	F_1-score	0.5295	0.0000	0.4705	0.6148	0.0000	0.3852
	MCC	0.5242	0.0000	0.4758	0.5865	0.0000	0.4135
	G-mean	0.5390	0.0020	0.4590	0.4843	0.0102	0.5055
	AUC	0.5458	0.0025	0.4517	0.4835	0.0095	0.5070
RUS *vs* ROS	F_1-score	0.3535	0.0000	0.6465	0.3183	0.0000	0.6817
	MCC	0.3788	0.0000	0.6212	0.3357	0.0000	0.6643
	G-mean	0.5050	0.0007	0.4943	0.4660	0.0185	0.5155
	AUC	0.5135	0.0002	0.4863	0.4677	0.0225	0.5098
RUS *vs* RB	F_1-score	0.3905	0.0000	0.6095	0.5270	0.0000	0.4730
	MCC	0.4283	0.0000	0.5717	0.5125	0.0000	0.4875
	G-mean	0.5232	0.0000	0.4768	0.5160	0.0005	0.4835
	AUC	0.5222	0.0000	0.4778	0.5092	0.0005	0.4903
ROS *vs* RB	F_1-score	0.4868	0.0000	0.5132	0.6457	0.0000	0.3543
	MCC	0.5057	0.0000	0.4943	0.6233	0.0000	0.3767
	G-mean	0.5290	0.0002	0.4708	0.4990	0.0002	0.5008
	AUC	0.5157	0.0000	0.4843	0.4907	0.0005	0.5088

Table 3.22: Bayesian hierarchical sign tests comparing different resampling methods (RUS, ROS, and RB) applied within the ensemble (BAG and GBT), according to the following measures: F_1-score, MCC, G-mean and AUC. Resampling methods were also compared to base ensembles using Gini and weighted Gini impurities. Depending on the specific metric, resampling-based ensembles may or may not be recommended. Which resampling method is better, is unclear.

Algorithm	Avg. rank
GBT+ROS	3.0000
SMOTE+GBT	4.0625
ROS+GBT	4.5625
GBT (WGini)	4.8750
GBT+RB	6.1875
GBT (Gini)	6.5625
GBT+RUS	7.0625
RUS+GBT	8.8125
BAG (Gini)	12.1250
BAG+ROS	12.8750
BAG (WGini)	13.8125
BAG+RB	14.5000
BAG+RUS	14.5625
SMOTE+BAG	14.7500
ROS+BAG	15.5000
SMOTE+RANF	15.6875
ROS+RANF	15.8125
RUS+BAG	16.0000
RUS+RANF	16.1875
RANF (WGini)	16.3125
RANF (Gini)	17.3750
ROSE+GBT	19.1562
ROSE+RANF	19.5625
ROSE+BAG	20.6562

(a) F_1-score

Algorithm	Avg. rank
GBT+ROS	2.9375
ROS+GBT	4.5625
GBT (Gini)	4.6250
GBT (WGini)	5.0625
SMOTE+GBT	5.6250
GBT+RB	5.8750
GBT+RUS	6.7500
RUS+GBT	8.6250
BAG (Gini)	11.0000
BAG+ROS	12.8125
BAG (WGini)	13.9375
BAG+RUS	14.0000
BAG+RB	14.6250
ROS+BAG	15.5625
ROS+RANF	15.6875
RUS+RANF	15.9375
RUS+BAG	16.0000
SMOTE+BAG	16.2500
RANF (WGini)	16.3125
RANF (Gini)	16.3750
SMOTE+RANF	16.4375
ROSE+GBT	19.0312
ROSE+RANF	20.7500
ROSE+BAG	21.2188

(b) MCC

Algorithm	Avg. rank
GBT+ROS	1.8125
GBT+RB	3.4375
GBT+RUS	4.1875
ROS+GBT	4.8125
GBT (WGini)	5.5625
RUS+GBT	6.6875
SMOTE+GBT	8.1250
BAG+RUS	10.4375
BAG+ROS	11.6250
BAG (WGini)	12.4375
BAG+RB	12.7500
RUS+BAG	12.8125
RUS+RANF	13.2500
ROS+RANF	13.5625
RANF (WGini)	13.8750
SMOTE+RANF	14.0625
ROS+BAG	14.1250
SMOTE+BAG	15.1250
GBT (Gini)	16.5000
ROSE+GBT	20.0938
ROSE+RANF	20.1250
ROSE+BAG	20.3438
BAG (Gini)	21.5000
RANF (Gini)	22.7500

(c) G-mean

Algorithm	Avg. rank
GBT+ROS	1.8125
GBT+RB	3.3750
GBT+RUS	4.1875
ROS+GBT	4.7500
GBT (WGini)	5.6875
RUS+GBT	6.6875
SMOTE+GBT	8.1250
BAG+RUS	10.3750
BAG+ROS	11.5625
BAG (WGini)	12.5625
RUS+BAG	12.6875
BAG+RB	12.8750
RUS+RANF	13.1875
ROS+RANF	13.6250
RANF (WGini)	13.9375
ROS+BAG	14.1875
SMOTE+RANF	14.2500
SMOTE+BAG	15.1875
GBT (Gini)	16.0000
ROSE+GBT	19.8438
ROSE+RANF	20.3125
ROSE+BAG	20.5312
BAG (Gini)	21.5000
RANF (Gini)	22.7500

(d) AUC

Table 3.23: Average ranks of all ensemble classifiers evaluated in this study, according to the following metrics: F_1-score, MCC, G-mean, and AUC. The statistical differences between the methods below the dashed line and the best method were significant according to the Hochberg procedure at a confidence level of 95%. GBT-based ensembles showed statistically better performances than BAG and RANF ensembles.

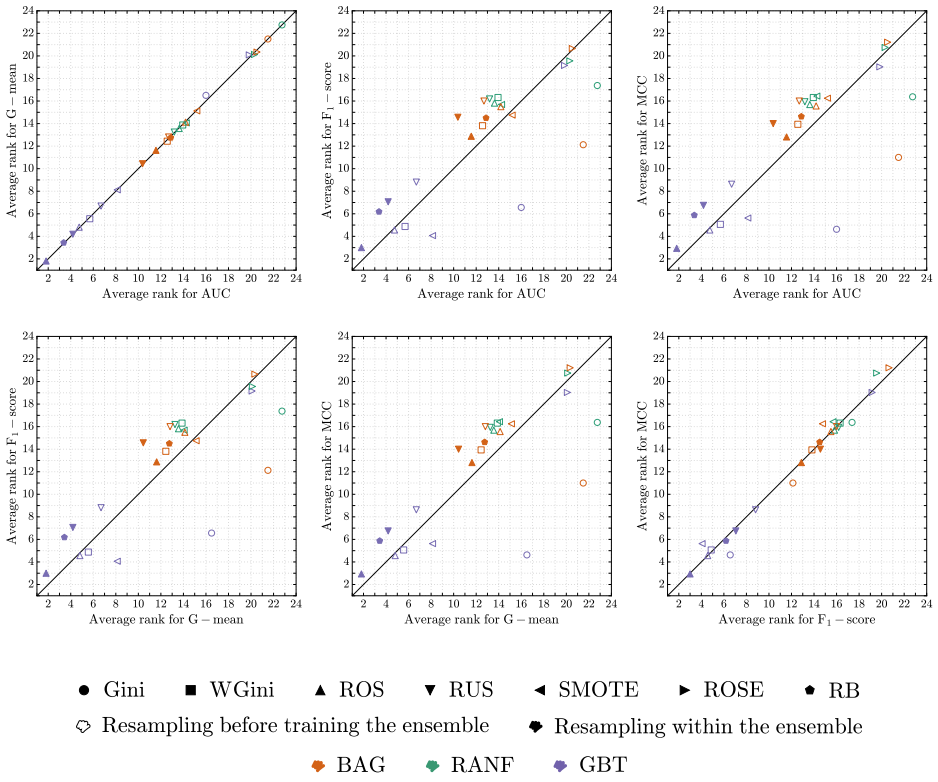


Figure 3.10: Average ranks for all ensemble methods according to every pair of metrics (G-mean and AUC, F_1-score and AUC, MCC and AUC, F_1-score and G-mean, MCC and G-mean, and MCC and F_1-score).The shape of the marker represents the resampling strategy, the color of the marker represents the ensemble method, and the fill of the marker represents the two resampling strategies: before training (unfilled), and within the ensemble (filled). GBT variants showed better performance than RANF and BAG variants in general. Great similarities between G-mean and AUC metrics, and also between MCC and F_1-score could be seen in top-left corner and bottom-right corner plots (*i.e.*, most of the markers were situated on the diagonal).

APPROX-SMOTE: FAST SMOTE FOR BIG DATA ON APACHE SPARK

4

This open source publication presents a fast SMOTE implementation for big data by means of using an approximated k-NN approach.

Authors: Mario Juez-Gil, Álvar Arnaiz-González, Juan J. Rodríguez, César García-Osorio.
Type: Open Source Publication
Journal: Neurocomputing (JCR: Q1, SJR: Q1)
Status: Under review
Keywords: SMOTE, Imbalance, Spark, Big data, Data mining

ABSTRACT

Big Data is the term used to describe those datasets made up of large volumes of data that are usually generated very rapidly, and probably in a wide variety of forms. These characteristics make Big Data a very challenging topic because traditional data mining methods are unable to process it, at the very least, in acceptable times. Class imbalance is a common problem present in Big Data classification. This means that one class is underrepresented compared to the others, which increases the risks of obtaining biased classifiers. To help avoid this problem, Approx-SMOTE has been designed. Approx-SMOTE is a parallel implementation of the SMOTE algorithm for Apache Spark framework, which uses an approximated k-Nearest Neighbor approach specifically intended for Big Data.

4.1 INTRODUCTION

In the era of Big Data, frameworks like Apache Spark (Zaharia et al., 2016) are growing and gaining more attention. One of the strengths of this kind of open source projects is that they allow contributions from third party developers. Through Spark MLlib (Meng et al., 2016), this framework provides a machine learning library that contains a wide variety of algorithms

for different tasks. Classification, regression, clustering, data extraction, data transformation, or data selection are some of them. Although powerful, its functionality is still limited compared to other machine learning frameworks like Scikit-Learn (Pedregosa et al., 2011), 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) (Chawla et al., 2002), 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 (Liu et al., 2004) for achieving accurate and efficient distributed nearest neighbor search.

Problems and Background

4.2 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 (Chawla et al., 2004; He and Garcia, 2009; Díez-Pastor et al., 2015b), it is still in an early research stage within Big Data scenarios (Sleeman IV and Krawczyk, 2021).

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 (Chawla et al., 2002), which generates new synthetic examples from on a small number of close examples. To find those close examples, k-NN is used. The main downside of k-NN is that it is computationally intense, with a complexity quadratic in the number of instances, $\mathcal{O}(n^2)$. Thus, when the number of instances of the dataset is huge, k-NN computation becomes infeasible (Neeb and Kurrus, 2016). Fortunately, there are a few approaches to efficiently approximate the nearest neighbor search by taking advantage of parallel and distributed computing (Liu et al., 2004; Liu et al., 2007). This approaches

are used by spark-knn package[1] implemented by Forest Fang (saurfang on GitHub[2]).

There exists an adaptation of SMOTE for Big Data, SMOTE-BD (Basgall et al., 2018) implemented and published in the SparkPackages repository[3]. Unfortunately, it uses an iterative implementation of exact k-NN (Maillo et al., 2017), which limits its applicability to solve real Big Data classification problems, as it requires a lot of computing power. The solution we are presenting, Approx-SMOTE, greatly reduces the required computing power by using an approximate nearest neighbor search approach. It should be noted that using an approximation to the nearest neighbors does not seem to negatively affect the final results.

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

4.3 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.

4.3.1 *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.

4.3.2 *Software Functionalities*

The Approx-SMOTE functionality consists in synthesizing new examples belonging to the minority class from an imbalanced binary classification

[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

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 number is a percentage of the number of minority class instances, and its default value is 100 (*i.e.*, the number of minority items is doubled). As k-NN is used for synthesizing new examples, all the parameters of Saurfang's approximated k-NN can be adjusted (`k`, `maxDistance`, `balanceThreshold`, `topTreeSize`, `topTreeLeafSize`, `subTreeLeafSize`, `bufferSizeSampleSizes`, and `bufferSize`). The parameters meaning and functioning is explained in detail in (Liu et al., 2007). The information about all the parameters default values is available at Saurfang's `spark-knn` GitHub repository.

4.4 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 4.1.

4.4.1 *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

Dataset	# instances	# attributes	# maj	# min	IR	Size (GB)
SUSY IR4	3 389 320	18	2 712 173	677 147	4.00	1.23
SUSY IR16	2 881 796	18	2 712 173	169 623	15.99	1.04
HIGGS IR4	7 284 166	28	5 829 123	1 455 043	4.00	3.94
HIGGS IR16	6 194 093	28	5 829 123	364 970	15.97	3.26
HEPMASS IR4	6 561 364	28	5 250 124	1 311 240	4.00	3.77
HEPMASS IR16	5 578 586	28	5 250 124	328 462	15.98	3.20

Table 4.1: 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 (Chang and Lin, 2011) format.

was executed on the biggest cluster, using 2-fold stratified cross-validation repeated 5 times[8].

For ensuring experiments to be repeatable, 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 (Basgall et al., 2018). All other parameters were kept as default.

For evaluating and comparing statistical differences in performance, a Bayesian analysis was conducted using Bayesian hierarchical sign tests (Benavoli et al., 2017) (*baycomp*[9] library was used). The number of samples for all Bayesian comparisons was set as 50 000. The graphical representation for this type of analysis, is a ternary plot (Juez-Gil, 2020) 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 (Sun et al., 2009).

[8]This cross-validation strategy is commonly used for imbalanced learning scenarios (Díez-Pastor et al., 2015a)

[9]The *baycomp* library is publicly available at https://baycomp.readthedocs.io/en/latest/.

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	$\boxed{0.772 \pm 0.001}$	0.542 ± 0.003	0.583 ± 0.003	$\boxed{0.589 \pm 0.003}$
SUSY IR16	0.622 ± 0.009	0.771 ± 0.001	$\boxed{0.772 \pm 0.001}$	$\boxed{0.380 \pm 0.019}$	0.293 ± 0.003	0.300 ± 0.004
HIGGS IR4	0.507 ± 0.002	0.671 ± 0.002	$\boxed{0.678 \pm 0.002}$	0.028 ± 0.008	0.452 ± 0.002	$\boxed{0.459 \pm 0.002}$
HIGGS IR16	0.500 ± 0.000	0.660 ± 0.001	$\boxed{0.672 \pm 0.002}$	0.000 ± 0.000	0.197 ± 0.001	$\boxed{0.202 \pm 0.001}$
HEPMASS IR4	0.748 ± 0.004	$\boxed{0.821 \pm 0.001}$	$\boxed{0.821 \pm 0.001}$	$\boxed{0.655 \pm 0.007}$	0.633 ± 0.003	0.630 ± 0.002
HEPMASS IR16	0.591 ± 0.014	$\boxed{0.822 \pm 0.001}$	$\boxed{0.822 \pm 0.001}$	0.306 ± 0.040	$\boxed{0.338 \pm 0.004}$	0.329 ± 0.005

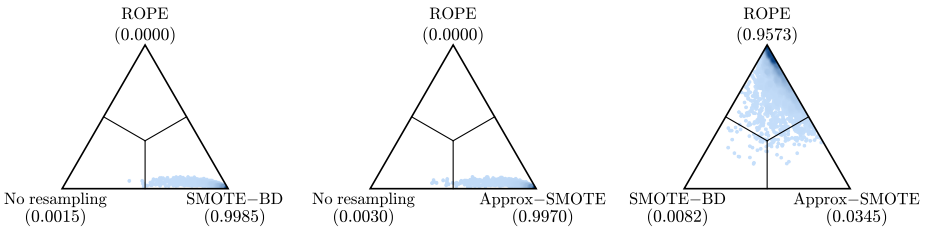
Table 4.2: Classification performance on Random Forest with 100 trees. The best results appear within black boxes. The higher the blueness intensity, the better the performance. The value at the right of the $\pm$ sign, refers to the standard deviation between cross-validation folds.

4.4.2 *Results and discussion*

The comparative results about classification performance is shown in Table 4.2. 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. Bayesian statistical tests are shown in Figure 4.1. The equivalence between SMOTE-BD and Approx-SMOTE is 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.

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 4.3 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 baseline, revealed good scalability for Approx-SMOTE, and scalability issues for SMOTE-BD. Figure 4.2 shows a graphical representation of the execution times and speedup comparisons where SMOTE-BD approach is depicted in purple, and Approx-SMOTE, in orange.

(a) AUC



(b) F_1-score

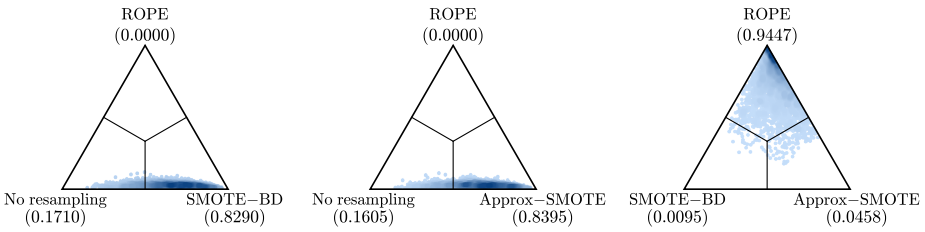


Figure 4.1: Bayesian hierarchical sign tests comparing No resampling vs. SMOTE-BD (left column), No resampling vs. Approx-SMOTE (central column), and SMOTE-BD vs. Approx-SMOTE (right column). According to AUC (a) and F_1-score (b) metrics.

Approach	2w	4w	6w	8w	10w
SMOTE-BD	1321.39	2218.60	2103.68	1587.29	2172.05
Approx-SMOTE	175.70	123.70	113.89	91.30	77.15

Table 4.3: 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.

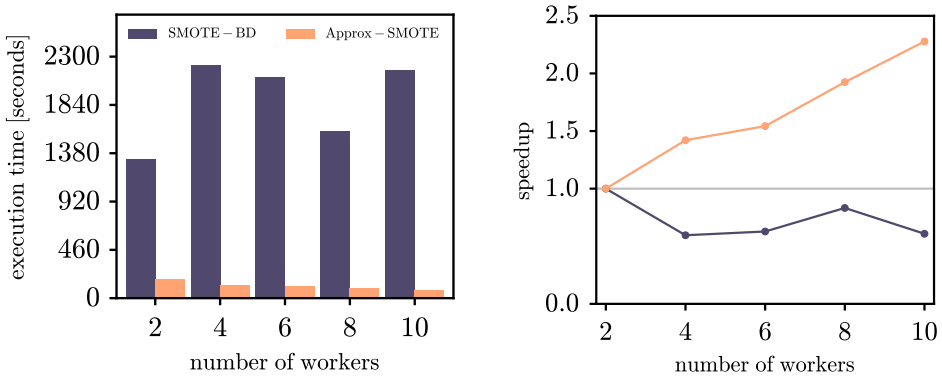


Figure 4.2: Execution time (left) and speedup (right) of SMOTE-BD (purple) vs. Approx-SMOTE (orange) on balancing SUSY IR16 dataset task. Different cluster configurations (2, 4, 6, 8, and 10 workers) were tested.

4.5 ILLUSTRATIVE EXAMPLES

As Approx-SMOTE follows Spark-ML design guidelines, its use is similar to any other Transformer in the Spark API. Code 4.1 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.

```
1  import org.apache.spark.ml.instance.ASMOTE
2
3  // reading dataset
4  val ds = session.read
5                  .format("libsvm")
6                  .option("inferSchema", "true")
7                  .load("binary.libsvm")
8
9  // using Approx-SMOTE
10 val asmote = new ASMOTE().setK(5)
11                         .setPercOver(100)
12                         .setSeed(46)
13 val resampledDF = asmote.transform(ds)
```

Listing 4.1: A basic example, written in Scala, showing how to use Approx-SMOTE for oversampling a dataset.

4.6 CONCLUSIONS

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 several times faster than SMOTE-BD. Moreover, the new proposal scales very well, going from an improvement of $7\times$ in a small cluster of 2 worker nodes, to almost $30\times$ when the number of workers is increased to 10. 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.

As there exist other algorithms for fast approximate nearest neighbor search, by means of using hashing for example (Gionis et al., 1999; Tchaye-Kondi et al., 2021), 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.

ACKNOWLEDGMENTS

The project leading to these results has received funding from "la Caixa" Foundation, under agreement LCF/PR/PR18/51130007. This work was supported through project BU055P20 (JCyL/FEDER, UE) of the *Junta de Castilla y León* (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.

BIBLIOGRAPHY

Arias, J., Gamez, J. A., and Puerta, J. M. (2017). Learning distributed discrete bayesian network classifiers under MapReduce with Apache Spark. *Knowledge-Based Systems*, 117:16 – 26. Volume, Variety and Velocity in Data Science.

Arnaiz-González, Á., Díez-Pastor, J. F., García-Osorio, C., and Rodríguez, J. J. (2016). Random feature weights for regression trees. *Progress in Artificial Intelligence*, 5(2):91–103.

Arnaiz-González, Á., Díez-Pastor, J.-F., Rodríguez, J. J., and García-Osorio, C. (2016). Instance selection of linear complexity for big data. *Knowledge-Based Systems*, 107:83 – 95.

Arnaiz-González, Á., Díez-Pastor, J.-F., Rodríguez, J. J., and García-Osorio, C. (2018a). Local sets for multi-label instance selection. *Applied Soft Computing*, 68:651 – 666.

Arnaiz-González, Á., Díez-Pastor, J.-F., Rodríguez, J. J., and García-Osorio, C. (2018b). Study of data transformation techniques for adapting single-label prototype selection algorithms to multi-label learning. *Expert Systems with Applications*, 109:114–130.

Arnaiz-González, Á., González-Rogel, A., Díez-Pastor, J.-F., and López-Nozal, C. (2017). MR-DIS: democratic instance selection for big data by MapReduce. *Progress in Artificial Intelligence*, 6(3):211–219.

Assefi, M., Behravesh, E., Liu, G., and Tafti, A. P. (2017). Big data machine learning using Apache Spark MLlib. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 3492–3498.

Azaria, A., Richardson, A., Kraus, S., and Subrahmanian, V. S. (2014). Behavioral analysis of insider threat: A survey and bootstrapped prediction in imbalanced data. *IEEE Transactions on Computational Social Systems*, 1(2):135–155.

Bagnall, A., Flynn, M., Large, J., Line, J., Bostrom, A., and Cawley, G. (2018). Is Rotation Forest the best classifier for problems with continuous features?

Basgall, M. J., Hasperué, W., Naiouf, M., Fernández, A., and Herrera, F. (2018). SMOTE-BD: An exact and scalable oversampling method for imbalanced classification in big data. *Journal of Computer Science and Technology*, 18(03):203–209.

Bayar, N., Darmoul, S., Hajri-Gabouj, S., and Pierreval, H. (2015). Fault detection, diagnosis and recovery using artificial immune systems: A review. *Engineering Applications of Artificial Intelligence*, 46:43 – 57.

Bekkar, M., Djemaa, H. K., and Alitouche, T. A. (2013). Evaluation measures for models assessment over imbalanced data sets. *Journal of Information Engineering and Applications*, 3(10).

Bellman, R. E. (2015). *Adaptive control processes: a guided tour*, volume 2045. Princeton university press.

Bello-Orgaz, G., Jung, J. J., and Camacho, D. (2016). Social big data: Recent achievements and new challenges. *Information Fusion*, 28:45 – 59.

Benavoli, A., Corani, G., Demšar, J., and Zaffalon, M. (2017). Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. *Journal of Machine Learning Research*, 18(77):1–36.

Benavoli, A., Corani, G., Mangili, F., and Zaffalon, M. (2015). A Bayesian nonparametric procedure for comparing algorithms. In *International Conference on Machine Learning*, pages 1264–1272. PMLR.

Benavoli, A., Corani, G., Mangili, F., Zaffalon, M., and Ruggeri, F. (2014). A Bayesian Wilcoxon signed-rank test based on the Dirichlet process. In *International conference on machine learning*, pages 1026–1034. PMLR.

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., and Cox, D. D. (2015). Hyperopt: a Python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554.

Betta, G., Liguori, C., Paolillo, A., and Pietrosanto, A. (2002). A DSP-based FFT-analyzer for the fault diagnosis of rotating machine based on vibration analysis. *IEEE Transactions on Instrumentation and Measurement*, 51(6):1316–1322.

Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100.

Bouchikhi, E. H. E., Choqueuse, V., and Benbouzid, M. E. H. (2013). Current frequency spectral subtraction and its contribution to induction machines' bearings condition monitoring. *IEEE Transactions on Energy Conversion*, 28(1):135–144.

Breiman, L. (1984). *Classification and Regression Trees*. Routledge.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Brzezinski, D., Stefanowski, J., Susmaga, R., and Szczech, I. (2020). On the dynamics of classification measures for imbalanced and streaming data. *IEEE Transactions on Neural Networks and Learning Systems*, 31(8):2868–2878.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.

Cadzow, J. A., Baseghi, B., and Hsu, T. (1983). Singular-value decomposition approach to time series modelling. In *IEE Proceedings F (Communications, Radar and Signal Processing)*, volume 130, pages 202–210. IET.

Caesarendra, W. and Tjahjowidodo, T. (2017). A review of feature extraction methods in vibration-based condition monitoring and its application for degradation trend estimation of low-speed slew bearing. *Machines*, 5(4):464–481.

Callaway, E. (2020). 'It will change everything': DeepMind's AI makes gigantic leap in solving protein structures. *Nature*.

Carlos Verucchi, José Bossio, G. B. and Acosta, G. (2016). A misalignment detection in induction motors with flexiblecoupling by means of estimated torque analysis and mcsa. *Mechanical Systems and Signal Processing*, 80:570 – 581.

Caruana, R. and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 161–168, New York, NY, USA. Association for Computing Machinery.

Casquilho, J. P. (2020). On the weighted Gini–Simpson index: estimating feasible weights using the optimal point and discussing a link with possibility theory. *Soft Computing*, 24(22):17187–17194.

Cattell, R. B. (1966). The scree test for the number of factors. *Multivariate Behavioral Research*, 1(2):245–276.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357.

Chawla, N. V., Japkowicz, N., and Kotcz, A. (2004). Editorial: Special issue on learning from imbalanced data sets. *SIGKDD Explorations Newsletter*, 6(1):1–6.

Chen, C., Liaw, A., and Breiman, L. (2004). Using random forest to learn imbalanced data. *University of California, Berkeley*, 110(1-12):12.

Chen, J., Li, K., Tang, Z., Bilal, K., Yu, S., Weng, C., and Li, K. (2017). A parallel random forest algorithm for big data in a Spark cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 28(4):919–933.

Chen, M., Mao, S., and Liu, Y. (2014). Big data: A survey. *Mobile networks and applications*, 19(2):171–209.

Choudhary, A., Goyal, D., Shimi, S. L., and Akula, A. (2018). Condition monitoring and fault diagnosis of induction motors: A review. *Archives of Computational Methods in Engineering*.

Chow, M.-Y., Yee, S., and Mangum, P. (1991). A neural network approach to real-time condition monitoring of induction motors. *IEEE Transactions on Industrial Electronics*, 38(6):448–453.

Clare, A. and King, R. D. (2001). Knowledge discovery in multi-label phenotype data. In *European conference on principles of data mining and knowledge discovery*, pages 42–53. Springer.

Combet, F. (2014). Gear fault diagnosis by motor current analysis: application to industrial cases. In Velex, P., editor, *International Gear Conference 2014: 26th–28th August 2014, Lyon*, pages 969–975. Chandos Publishing, Oxford.

Corani, G. and Benavoli, A. (2015). A Bayesian approach for comparing cross-validated algorithms on multiple data sets. *Machine Learning*, 100(2-3):285–304.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

Cukier, K. and Mayer-Schoenberger, V. (2013). The rise of big data: How it's changing the way we think about the world. *Foreign Aff.*, 92:28.

Dagum, L. and Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55.

De Gooijer, J. G. and Hyndman, R. J. (2006). 25 years of time series forecasting. *International journal of forecasting*, 22(3):443–473.

Dean, J. and Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of ACM*, 51(1):107–113.

del Río, S., López, V., Benítez, J. M., and Herrera, F. (2014). On the use of MapReduce for imbalanced big data using Random Forest. *Information Sciences*, 285:112 – 137.

Demšar, J. (2008). On the appropriateness of statistical tests in machine learning. In *Workshop on Evaluation Methods for Machine Learning in conjunction with ICML*, page 65.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30.

Díez-Pastor, J.-F., García-Osorio, C., and Rodríguez, J. J. (2014). Tree ensemble construction using a GRASP-based heuristic and annealed randomness. *Information Fusion*, 20:189–202.

Diez-Pastor, J. F., Gil Del Val, A., Veiga, F., and Bustillo, A. (2021). High-accuracy classification of thread quality in tapping processes with ensembles of classifiers for imbalanced learning. *Measurement*, 168:108328.

Díez-Pastor, J. F., Rodríguez, J. J., García-Osorio, C., and Kuncheva, L. I. (2015a). Random balance: Ensembles of variable priors classifiers for imbalanced data. *Knowledge-Based Systems*, 85:96 – 111.

Díez-Pastor, J. F., Rodríguez, J. J., García-Osorio, C. I., and Kuncheva, L. I. (2015b). Diversity techniques improve the performance of the best imbalance learning ensembles. *Information Sciences*, 325:98 – 117.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Duan, F. and Živanović, R. (2015). Condition monitoring of an induction motor stator windings via global optimization based on the hyperbolic cross points. *IEEE Transactions on Industrial Electronics*, 62(3):1826–1834.

Fang, X., Zheng, X., Tan, Y., and Zhang, H. (2016). Highly imbalanced classification using improved rotation forests. *International Journal of Wireless and Mobile Computing*, 10(1):35–41.

Fazakis, N., Karlos, S., Kotsiantis, S., and Sgarbas, K. (2017). Self-trained rotation forest for semi-supervised learning. *Journal of Intelligent & Fuzzy Systems*, 32(1):711–722.

Fernández, A., del Río, S., Chawla, N. V., and Herrera, F. (2017). An insight into imbalanced big data classification: outcomes and challenges. *Complex & Intelligent Systems*, 3(2):105–120.

Fernández, A., del Río, S., López, V., Bawakid, A., del Jesus, M. J., Benítez, J. M., and Herrera, F. (2014). Big Data with Cloud Computing: an insight on the computing environment, MapReduce, and programming frameworks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(5):380–409.

Fernández, A., García, S., Galar, M., Prati, R. C., Krawczyk, B., and Herrera, F. (2018a). *Learning from imbalanced data sets*, volume 11. Springer.

Fernández, A., Garcia, S., Herrera, F., and Chawla, N. V. (2018b). SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *Journal of artificial intelligence research*, 61:863–905.

Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *The journal of machine learning research*, 15(1):3133–3181.

Fix, E. and Hodges Jr, J. L. (1951). Discriminatory analysis-nonparametric discrimination: consistency properties. Technical report, California Univ Berkeley.

Friedman, N., Geiger, D., and Goldszmidt, M. (1997). Bayesian network classifiers. *Machine learning*, 29(2-3):131–163.

Fu, T.-c. (2011). A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181.

Galar, M., Fernandez, A., Barrenechea, E., Bustince, H., and Herrera, F. (2012). A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484.

Gangsar, P. and Tiwari, R. (2017). Comparative investigation of vibration and current monitoring for prediction of mechanical and electrical faults in induction motor based on multiclass-support vector machine algorithms. *Mechanical Systems and Signal Processing*, 94:464–481.

Gantz, J. and Reinsel, D. (2011). Extracting value from chaos. *IDC iview*, 1142(2011):1–12.

Gao, X., Chen, Z., Tang, S., Zhang, Y., and Li, J. (2016). Adaptive weighted imbalance learning with application to abnormal activity recognition. *Neurocomputing*, 173:1927 – 1935.

Gao, Z., Cecati, C., and Ding, S. X. (2015). A survey of fault diagnosis and fault-tolerant techniques – Part I: Fault diagnosis with model-based and signal-based approaches. *IEEE Transactions on Industrial Electronics*, 62(6):3757–3767.

Gao, Z., Zhang, L.-f., Chen, M.-y., Hauptmann, A., Zhang, H., and Cai, A.-N. (2014). Enhanced and hierarchical structure algorithm for data imbalance problem in semantic extraction under massive video dataset. *Multimedia tools and applications*, 68(3):641–657.

García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J. M., and Herrera, F. (2016). Big data preprocessing: methods and prospects. *Big Data Analytics*, 1(1):9.

García-Gil, D., Luengo, J., García, S., and Herrera, F. (2019). Enabling smart data: Noise filtering in big data classification. *Information Sciences*, 479:135 – 152.

García-Gil, D., Ramírez-Gallego, S., García, S., and Herrera, F. (2018). Principal components analysis random discretization ensemble for big data. *Knowledge-Based Systems*, 150:166 – 174.

García-Osorio, C., de Haro-García, A., and García-Pedrajas, N. (2010). Democratic instance selection: A linear complexity instance selection algorithm based on classifier ensemble concepts. *Artificial Intelligence*, 174(5-6):410–441.

Garcia-Perez, A., Romero-Troncoso, R. J., Cabal-Yepez, E., Osornio-Rios, R. A., and Lucio-Martinez, J. A. (2012). Application of high-resolution spectral analysis for identifying faults in induction motors by means of sound. *Journal of Vibration and Control*, 18(11):1585–1594.

Ge, Z., Song, Z., Ding, S. X., and Huang, B. (2017). Data mining and analytics in the process industry: The role of machine learning. *IEEE Access*, 5:20590–20616.

Gemp, I., McWilliams, B., Vernade, C., and Graepel, T. (2021). EigenGame: PCA as a Nash Equilibrium. In *International Conference on Learning Representations*.

Ghaderi Zefrehi, H. and Altınçay, H. (2020). Imbalance learning using heterogeneous ensembles. *Expert Systems with Applications*, 142:113005.

Ghate, V. N. and Dudul, S. V. (2011). Cascade neural-network-based fault classifier for three-phase induction motor. *IEEE Transactions on Industrial Electronics*, 58(5):1555–1563.

Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The Google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, page 29–43, New York, NY, USA. Association for Computing Machinery.

Gionis, A., Indyk, P., Motwani, R., et al. (1999). Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529.

González, S., García, S., Del Ser, J., Rokach, L., and Herrera, F. (2020). A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities. *Information Fusion*, 64:205 – 237.

Gonzalez-Lopez, J., Cano, A., and Ventura, S. (2017). Large-scale multi-label ensemble learning on Spark. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 893–900.

Haixiang, G., Yijing, L., Shang, J., Mingyun, G., Yuanyue, H., and Bing, G. (2017). Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications*, 73:220 – 239.

Hariri, R. H., Fredericks, E. M., and Bowers, K. M. (2019). Uncertainty in big data analytics: survey, opportunities, and challenges. *Journal of Big Data*, 6(1):44.

Hasanin, T. and Khoshgoftaar, T. (2018). The effects of random undersampling with simulated class imbalance for big data. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 70–79.

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., and Ullah Khan, S. (2015). The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, 47:98–115.

Hashemian, H. M. (2010). State-of-the-art predictive maintenance techniques. *IEEE Transactions on Instrumentation and measurement*, 60(1):226–236.

Hassan, O. E., Amer, M., Abdelsalam, A. K., and Williams, B. W. (2018). Induction motor broken rotor bar fault detection techniques based on fault signature analysis–a review. *IET Electric Power Applications*, 12(7):895–907.

He, H. and Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284.

Hochberg, Y. (1988). A sharper Bonferroni procedure for multiple tests of significance. *Biometrika*, 75(4):800–802.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Horn, J. L. (1965). A rationale and test for the number of factors in factor analysis. *Psychometrika*, 30(2):179–185.

Hosseinzadeh, M. and Eftekhari, M. (2015). Improving rotation forest performance for imbalanced data classification through fuzzy clustering. In *2015 The International Symposium on Artificial Intelligence and Signal Processing (AISP)*, pages 35–40. IEEE.

Hu, F. and Li, H. (2013). A novel boundary oversampling algorithm based on neighborhood rough set model: NRSBoundary-SMOTE. *Mathematical Problems in Engineering*, 2013.

Hu, F., Li, H., Lou, H., and Dai, J. (2014a). A parallel oversampling algorithm based on NRSBoundary-SMOTE. *Journal of Information & Computational Science*, 11(13):4655–4665.

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

Hui, K. H., Lim, M. H., Leong, M. S., and Al-Obaidi, S. M. (2017). Dempster-shafer evidence theory for multi-bearing faults diagnosis. *Engineering Applications of Artificial Intelligence*, 57:160 – 170.

Iman, R. L. and Davenport, J. M. (1980). Approximations of the critical region of the fbietkan statistic. *Communications in Statistics-Theory and Methods*, 9(6):571–595.

Jain, A. (2016). The 5 V's of big data. IBM.

Jain, S. and Kumar, L. (2018). 31 - Fundamentals of power electronics controlled electric propulsion. In Rashid, M. H., editor, *Power Electronics Handbook (Fourth Edition)*, pages 1023–1065. Butterworth-Heinemann, fourth edition.

James, C. and Lowe, D. (2001). Single channel analysis of electromagnetic brain signals through ICA in a dynamical systems framework. In *2001 Conference Proceedings of the 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 2, pages 1974–1977. IEEE.

Jeon, Y. and Lim, D. (2020). PSU: Particle stacking undersampling method for highly imbalanced big data. *IEEE Access*, 8:131920–131927.

Jia, Z., Zaharia, M., and Aiken, A. (2018). Beyond data and model parallelism for deep neural networks. *arXiv preprint arXiv:1807.05358*.

Jolliffe, I. T. (1986). *Principal Component Analysis*. Springer Series in Statistics. Springer New York, New York, NY.

Jolliffe, I. T. and Cadima, J. (2016). Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202.

Juez-Gil, M. (2020). mjuez/baycomp_plotting.

Juez-Gil, M., Arnaiz-González, Á., Rodríguez, J. J., and García-Osorio, C. (2021a). Experimental evaluation of ensemble classifiers for imbalance in Big Data. *Applied Soft Computing*, page 107447.

Juez-Gil, M., Arnaiz-González, Á., Rodríguez, J. J., López-Nozal, C., and García-Osorio, C. (2021b). Rotation Forest for Big Data. *Information Fusion*, 74:39–49.

Juez-Gil, M., Saucedo-Dorantes, J. J., Arnaiz-González, Á., López-Nozal, C., García-Osorio, C., and Lowe, D. (2020). Early and extremely early multi-label fault diagnosis in induction motors. *ISA Transactions*, 106:367–381.

Kaiser, H. F. (1960). The application of electronic computers to factor analysis. *Educational and Psychological Measurement*, 20(1):141–151.

Katuwal, R., Suganthan, P., and Zhang, L. (2020). Heterogeneous oblique random forest. *Pattern Recognition*, 99:107078.

Kazzaz, S. A. S. A. and Singh, G. (2003). Experimental investigations on induction machine condition monitoring and fault diagnosis using digital signal processing techniques. *Electric Power Systems Research*, 65(3):197 – 221.

Keskes, H. and Braham, A. (2015). Recursive undecimated wavelet packet transform and dag svm for induction motor diagnosis. *IEEE Transactions on Industrial Informatics*, 11(5):1059–1066.

Kim, K. and Parlos, A. G. (2002). Model-based fault diagnosis of induction motors using non-stationary signal segmentation. *Mechanical Systems and Signal Processing*, 16(2-3):223–253.

Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 14, pages 1137–1145. Montreal, Canada.

Konar, P. and Chattopadhyay, P. (2015). Multi-class fault diagnosis of induction motor using hilbert and wavelet transform. *Applied Soft Computing*, 30:341 – 352.

Kordos, M., Arnaiz-González, Á., and García-Osorio, C. (2019). Evolutionary prototype selection for multi-output regression. *Neurocomputing*.

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

Krawczyk, B., Galar, M., Łukasz Jeleń, and Herrera, F. (2016). Evolutionary undersampling boosting for imbalanced classification of breast cancer malignancy. *Applied Soft Computing*, 38:714 – 726.

Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J., and Woźniak, M. (2017). Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132–156.

Kumar, C., Krishnan, G., and Sarangi, S. (2015). Experimental investigation on misalignment fault detection in induction motors using current and vibration signature analysis. In *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, pages 61–66. IEEE.

Kuncheva, L. I. (2014). *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons.

Kuncheva, L. I. and Rodriguez, J. J. (2007). Classifier ensembles with a random linear oracle. *IEEE Transactions on Knowledge and Data Engineering*, 19(4):500–508.

Kuncheva, L. I. and Rodríguez, J. J. (2007). An experimental study on rotation forest ensembles. In Haindl, M., Kittler, J., and Roli, F., editors,

Multiple Classifier Systems, pages 459–468, Berlin, Heidelberg. Springer Berlin Heidelberg.

Laney, D. (2001). 3-d data management: controlling data volume, velocity and variety. Technical report, META Group Research Note.

Lasi, H., Fettke, P., Kemper, H.-G., Feld, T., and Hoffmann, M. (2014). Industry 4.0. *Business & information systems engineering*, 6(4):239–242.

Leevy, J. L., Khoshgoftaar, T. M., Bauder, R. A., and Seliya, N. (2018). A survey on addressing high-class imbalance in big data. *Journal of Big Data*, 5(1):42.

Lei, Y., Jia, F., Lin, J., Xing, S., and Ding, S. X. (2016). An intelligent fault diagnosis method using unsupervised feature learning towards mechanical big data. *IEEE Transactions on Industrial Electronics*, 63(5):3137–3147.

Li, C., de Oliveira, J. V., Cerrada, M., Pacheco, F., Cabrera, D., Sanchez, V., and Zurita, G. (2016). Observer-biased bearing condition monitoring: From fault detection to multi-fault classification. *Engineering Applications of Artificial Intelligence*, 50:287 – 301.

Liu, H., Zhou, M., Lu, X. S., and Yao, C. (2018). Weighted gini index feature selection method for imbalanced data. In *2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC)*, pages 1–6.

Liu, R., Yang, B., Zio, E., and Chen, X. (2018). Artificial intelligence for fault diagnosis of rotating machinery: A review. *Mechanical Systems and Signal Processing*, 108:33 – 47.

Liu, T., Moore, A. W., Gray, A., and Yang, K. (2004). An investigation of practical approximate nearest neighbor algorithms. In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, NIPS'04, page 825–832, Cambridge, MA, USA. MIT Press.

Liu, T., Rosenberg, C., and Rowley, H. A. (2007). Clustering billions of images with large scale nearest neighbor search. In *2007 IEEE Workshop on Applications of Computer Vision (WACV '07)*, pages 28–28.

Liu, X., Wu, J., and Zhou, Z. (2009). Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):539–550.

Liu, Y. and Bazzi, A. M. (2017). A review and comparison of fault detection and diagnosis methods for squirrel-cage induction motors: State of the art. *ISA Transactions*, 70:400 – 409.

Lowe, D. (1998). Feature space embeddings for extracting structure from single channel wake EEG using RBF networks. In *Neural Networks for Signal Processing VIII. Proceedings of the 1998 IEEE Signal Processing Society Workshop (Cat. No. 98TH8378)*, pages 428–437. IEEE.

Lowe, D. (2015). Radial basis function networks-revisited. *Mathematics Today*, 51(3):124–126.

Luengo, J., García-Gil, D., Ramírez-Gallego, S., García, S., and Herrera, F. (2020). *Big Data Preprocessing: Enabling Smart Data*. Springer Nature.

Lunardon, N., Menardi, G., and Torelli, N. (2014). ROSE: A package for binary imbalanced learning. *The R journal*, 6(1).

Madjarov, G., Kocev, D., Gjorgjevikj, D., and Džeroski, S. (2012). An extensive experimental comparison of methods for multi-label learning. *Pattern recognition*, 45(9):3084–3104.

Mahmoud, H., Abou-Elyazied Abdallh, A., Bianchi, N., El-Hakim, S. M., Shaltout, A., and Dupré, L. (2016). An inverse approach for interturn fault detection in asynchronous machines using magnetic pendulous oscillation technique. *IEEE Transactions on Industry Applications*, 52(1):226–233.

Maillo, J., Ramírez, S., Triguero, I., and Herrera, F. (2017). kNN-IS: An iterative Spark-based design of the k-Nearest Neighbors classifier for big data. *Knowledge-Based Systems*, 117:3 – 15. Volume, Variety and Velocity in Data Science.

Marques Cardoso, A. J., Cruz, S. M. A., and Fonseca, D. S. B. (1999). Inter-turn stator winding fault diagnosis in three-phase induction motors, by Park's vector approach. *IEEE Transactions on Energy Conversion*, 14(3):595–598.

Martin-Diaz, I., Morinigo-Sotelo, D., Duque-Perez, O., Osornio-Rios, R. A., and Romero-Troncoso, R. J. (2018). Hybrid algorithmic approach oriented to incipient rotor fault diagnosis on induction motors. *ISA Transactions*, 80:427 – 438.

Martínez-Morales, J. D., Palacios-Hernández, E. R., and Campos-Delgado, D. (2018). Multiple-fault diagnosis in induction motors through support vector machine classification at variable operating conditions. *Electrical Engineering*, 100(1):59–73.

Maudes, J., Rodríguez, J. J., and García-Osorio, C. (2009a). Disturbing neighbors diversity for decision forests. In *Applications of supervised and unsupervised ensemble methods*, pages 113–133. Springer.

Maudes, J., Rodríguez, J. J., and García-Osorio, C. (2009b). Disturbing neighbors ensembles for linear SVM. In *International Workshop on Multiple Classifier Systems*, pages 191–200. Springer.

Maudes, J., Rodríguez, J. J., García-Osorio, C., and García-Pedrajas, N. (2012). Random feature weights for decision tree ensemble construction. *Information Fusion*, 13(1):20–30.

McCormick, P. and Ahrens, J. (2005). 27 - Large-scale data visualization and rendering: A problem-driven approach. In Hansen, C. D. and Johnson, C. R., editors, *Visualization Handbook*, pages 533–549. Butterworth-Heinemann, Burlington.

Menardi, G. and Torelli, N. (2014). Training and assessing classification rules with imbalanced data. *Data Mining and Knowledge Discovery*, 28(1):92–122.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., and et al. (2016). MLlib: Machine learning in Apache Spark. *The Journal of Machine Learning Research*, 17(1):1235–1241.

Mobley, R. K. (2002). *An introduction to predictive maintenance*. Elsevier.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). *Foundations of machine learning*. MIT press, second edition.

Moretti, C., Steinhaeuser, K., Thain, D., and Chawla, N. V. (2008). Scaling up classifiers to cloud computers. In *2008 Eighth IEEE International Conference on Data Mining*, pages 472–481.

Naderi, P. (2017). Modified magnetic-equivalent-circuit approach for various faults studying in saturable double-cage-induction machines. *IET Electric Power Applications*, 11:1224–1234(10).

Neeb, H. and Kurrus, C. (2016). Distributed k-nearest neighbors.

Nguyen, K. T. and Medjaher, K. (2019). A new dynamic predictive maintenance framework using deep learning for failure prognostics. *Reliability Engineering & System Safety*, 188:251–262.

O'Brien, R. and Ishwaran, H. (2019). A random forests quantile classifier for class imbalanced data. *Pattern Recognition*, 90:232 – 249.

Orhan, S., Aktürk, N., and Celik, V. (2006). Vibration monitoring for defect diagnosis of rolling element bearings as a predictive maintenance tool: Comprehensive case studies. *Ndt & E International*, 39(4):293–298.

Panda, B., Herbach, J. S., Basu, S., and Bayardo, R. J. (2009). PLANET: Massively parallel learning of tree ensembles with MapReduce. In *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB-2009)*.

Pardo, C., Diez-Pastor, J. F., García-Osorio, C., and Rodríguez, J. J. (2013). Rotation forests for regression. *Applied Mathematics and Computation*, 219(19):9914–9924.

Pardo, C., Rodríguez, J. J., Díez-Pastor, J. F., and García-Osorio, C. (2011). Random oracles for regression ensembles. In *Ensembles in Machine Learning Applications*, pages 181–199. Springer.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Prieto, M. D., Cirrincione, G., Espinosa, A. G., Ortega, J. A., and Henao, H. (2012). Bearing fault detection by a novel condition-monitoring scheme based on statistical-time features and neural networks. *IEEE Transactions on Industrial Electronics*, 60(8):3398–3407.

Qi, Q. and Tao, F. (2018). Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison. *IEEE Access*, 6:3585–3593.

Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

Ramentol, E., Gondres, I., Lajes, S., Bello, R., Caballero, Y., Cornelis, C., and Herrera, F. (2016). Fuzzy-rough imbalanced learning for the diagnosis of High Voltage Circuit Breaker maintenance: The SMOTE-FRST-2T algorithm. *Engineering Applications of Artificial Intelligence*, 48:134 – 139.

Ramírez-Gallego, S., Fernández, A., García, S., Chen, M., and Herrera, F. (2018a). Big data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce. *Information Fusion*, 42:51 – 61.

Ramírez-Gallego, S., García, S., Benítez, J., and Herrera, F. (2018b). A distributed evolutionary multivariate discretizer for big data processing on Apache Spark. *Swarm and Evolutionary Computation*, 38:240 – 250.

Ramírez-Gallego, S., García, S., Xiong, N., and Herrera, F. (2018). BELIEF: A distance-based redundancy-proof feature selection method for Big Data.

Ramírez-Gallego, S., Krawczyk, B., García, S., Woźniak, M., Benítez, J. M., and Herrera, F. (2017). Nearest neighbor classification for high-speed big data streams using Spark. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(10):2727–2739.

Randall, R. B. and Antoni, J. (2011). Rolling element bearing diagnostics—a tutorial. *Mechanical systems and signal processing*, 25(2):485–520.

Rauber, T. W., de Assis Boldt, F., and Varejão, F. M. (2014). Heterogeneous feature models and feature selection applied to bearing fault diagnosis. *IEEE Transactions on Industrial Electronics*, 62(1):637–646.

Read, J. (2010). *Scalable multi-label classification*. PhD thesis, University of Waikato.

Rodriguez, J. J., Kuncheva, L. I., and Alonso, C. J. (2006). Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1619–1630.

Rokach, L. (2009). A survey of clustering algorithms. In *Data mining and knowledge discovery handbook*, pages 269–298. Springer.

Roman-Rangel, E. and Marchand-Maillet, S. (2019). Inductive t-SNE via deep learning to visualize multi-label images. *Engineering Applications of Artificial Intelligence*, 81:336 – 345.

Romero-Troncoso, R., Garcia-Perez, A., Morinigo-Sotelo, D., Duque-Perez, O., Osornio-Rios, R., and Ibarra-Manzano, M. (2016). Rotor unbalance and broken rotor bar detection in inverter-fed induction motors at start-up and steady-state regimes by high-resolution spectral analysis. *Electric Power Systems Research*, 133:142 – 148.

Ruscio, J. and Roche, B. (2012). Determining the number of factors to retain in an exploratory factor analysis using comparison data of known factorial structure. *Psychological Assessment*, 24(2):282–292.

Saleem, M. A., Diwakar, G., and Satyanarayana, M. (2012). Detection of unbalance in rotating machines using shaft deflection measurement during its operation. *IOSR J. Mech. Civ. Eng*, 3(3):08–20.

Santos, P., Maudes, J., and Bustillo, A. (2018). Identifying maximum imbalance in datasets for fault diagnosis of gearboxes. *Journal of Intelligent Manufacturing*, 29(2):333–351.

Saucedo-Dorantes, J. J., Delgado-Priet, M., Osornio-Rios, R. A., and de Jesus Romero-Troncoso, R. (2017). Multifault diagnosis method applied to an electric machine based on high-dimensional feature reduction. *IEEE Transactions on Industry Applications*, 53:3086 – 3097.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5(2):197–227.

Schapire, R. E. (1999). A brief introduction to boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, page 1401–1406, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Seeger, M. (2002). Learning with labeled and unlabeled data. Technical report, Institute for Adaptive and Neural Computation, University of Edinburgh.

Shakweh, Y. (2018). Drive types and specifications. In *Power Electronics Handbook*, pages 913–944. Elsevier.

Shen, C., Wang, D., Kong, F., and Tse, P. W. (2013). Fault diagnosis of rotating machinery based on the statistical parameters of wavelet packet paving and a generic support vector regressive classifier. *Measurement*, 46:1551 – 1564.

Shumway, R. H. and Stoffer, D. S. (2017). *Time Series Analysis and Its Applications*. Springer Texts in Statistics. Springer International Publishing, Cham.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The Hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. IEEE.

Singer, G., Anuar, R., and Ben-Gal, I. (2020). A weighted information-gain measure for ordinal classification trees. *Expert Systems with Applications*, 152:113375.

Sleeman IV, W. C. and Krawczyk, B. (2021). Multi-class imbalanced big data classification on Spark. *Knowledge-Based Systems*, 212:106598.

Spyromitros-Xioufis, E., Tsoumakas, G., Groves, W., and Vlahavas, I. (2016). Multi-target regression via input space expansion: treating targets as inputs. *Machine Learning*, 104(1):55–98.

Stack, J. R., Habetler, T. G., and Harley, R. G. (2004). Bearing fault detection via autoregressive stator current modeling. *IEEE Transactions on Industry Applications*, 40(3):740–747.

Stapor, K., Ksieniewicz, P., García, S., and Woźniak, M. (2021). How to design the fair experimental classifier evaluation. *Applied Soft Computing*, 104:107219.

Stiglic, G., Rodriguez, J. J., and Kokol, P. (2011). *Rotation of random forests for genomic and proteomic classification problems*, pages 211–221. Springer New York, New York, NY.

Su, C., Ju, S., Liu, Y., and Yu, Z. (2015). Improving random forest and rotation forest for highly imbalanced datasets. *Intelligent Data Analysis*, 19(6):1409–1432.

Sun, Y., Wong, A. K., and Kamel, M. S. (2009). Classification of imbalanced data: A review. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(4):687–719.

Tanha, J., Abdi, Y., Samadi, N., Razzaghi, N., and Asadpour, M. (2020). Boosting methods for multi-class imbalanced data classification: an experimental review. *Journal of Big Data*, 7(1):1–47.

Tchaye-Kondi, J., Zhai, Y., and Zhu, L. (2021). A new hashing based nearest neighbors selection technique for big datasets.

Thomas, V., Vasudevan, K., and Kumar, V. (2003). Online cage rotor fault detection using air-gap torque spectra. *IEEE Transactions on Energy Conversion*, 18(2):265–270.

Triguero, I., del Río, S., López, V., Bacardit, J., Benítez, J. M., and Herrera, F. (2015). ROSEFW-RF: The winner algorithm for the ECBDL'14 big data competition: An extremely imbalanced big data bioinformatics problem. *Knowledge-Based Systems*, 87:69 – 79. Computational Intelligence Applications for Data Science.

Triguero, I., Galar, M., Vluymans, S., Cornelis, C., Bustince, H., Herrera, F., and Saeys, Y. (2015). Evolutionary undersampling for imbalanced big data classification. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 715–722.

Tsoumakas, G. and Katakis, I. (2007). Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3):1–10,12–13.

Tsoumakas, G., Katakis, I., and Vlahavas, I. (2009). Mining multi-label data. In *Data mining and knowledge discovery handbook*, pages 667–685. Springer.

Tsypkin, M. (2017). The origin of the electromagnetic vibration of induction motors operating in modern industry: Practical experience, analysis and diagnostics. *IEEE Transactions on Industry Applications*, 53:1669 – 1676.

Tyree, S., Weinberger, K. Q., Agrawal, K., and Paykin, J. (2011). Parallel boosted regression trees for web search ranking. In *Proceedings of the 20th International Conference on World Wide Web*, page 387–396, New York, NY, USA. Association for Computing Machinery.

van Engelen, J. E. and Hoos, H. H. (2020). A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440.

Waegeman, W., Dembczyński, K., and Hüllermeier, E. (2019). Multi-target prediction: a unifying view on problems and methods. *Data Mining and Knowledge Discovery*, 33(2):293–324.

Wang, C. J., Ng, C. Y., and Brook, R. H. (2020). Response to COVID-19 in Taiwan: Big Data Analytics, New Technology, and Proactive Testing. *JAMA*, 323(14):1341–1342.

Wang, H., Ding, C., and Huang, H. (2010). Multi-label linear discriminant analysis. In Daniilidis, K., Maragos, P., and Paragios, N., editors, *Computer Vision – ECCV 2010*, pages 126–139. Springer Berlin Heidelberg, Berlin, Heidelberg.

Wang, T., Qi, J., Xu, H., Wang, Y., Liu, L., and Gao, D. (2016). Fault diagnosis method based on FFT-RPCA-SVM for Cascaded-Multilevel Inverter. *ISA Transactions*, 60:156–163.

Witten, I. H., Frank, E., and Hall, M. A., editors (2011). *Data Mining: Practical Machine Learning Tools and Techniques (Third Edition)*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, Boston, third edition edition.

Woon, W. L. and Lowe, D. (2004). Can we learn anything from single-channel unaveraged MEG data? *Neural Computing & Applications*, 13(4):360–368.

Yan, J., Meng, Y., Lu, L., and Li, L. (2017). Industrial big data in an industry 4.0 environment: Challenges, schemes, and applications for predictive maintenance. *IEEE Access*, 5:23484–23491.

Yan, R., Gao, R. X., and Chen, X. (2014). Wavelets for fault diagnosis of rotary machines: A review with applications. *Signal Processing*, 96:1 – 15. Time-frequency methods for condition based maintenance and modal analysis.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*, pages 189–196.

Yu, J., Zhang, T., and Qian, J. (2011). 9 - How to improve the energy-efficiency of electrical motor products. In Yu, J., Zhang, T., and Qian, J., editors, *Electrical Motor Products*, pages 139–172. Woodhead Publishing.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient distributed

datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, San Jose, CA. USENIX.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., Stoica, I., et al. (2010). Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., et al. (2016). Apache Spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65.

Zhang, C.-X. and Zhang, J.-S. (2008). RotBoost: A technique for combining Rotation Forest and AdaBoost. *Pattern Recognition Letters*, 29(10):1524 – 1536.

Zhang, M.-L. and Zhou, Z.-H. (2014). A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837.

Zhang, R., Isola, P., and Efros, A. A. (2016). Colorful image colorization. In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *Computer Vision – ECCV 2016*, pages 649–666, Cham. Springer International Publishing.

Zhang, Y., Li, X., Gao, L., Wang, L., and Wen, L. (2018). Imbalanced data fault diagnosis of rotating machinery using synthetic oversampling and feature learning. *Journal of manufacturing systems*, 48:34–50.

Zhou, Y. and Goldman, S. (2004). Democratic co-learning. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 594–602. IEEE.

Zhou, Z.-H. and Li, M. (2005). Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on knowledge and Data Engineering*, 17(11):1529–1541.

Zwick, W. R. and Velicer, W. F. (1986). Comparison of five rules for determining the number of components to retain. *Psychological Bulletin*, 99(3):432–442.