2021

Learning from Multi-Class Imbalanced Big Data with Apache Spark

William C. Sleeman IV
Virginia Commonwealth University

LEARNING FROM MULTI-CLASS IMBALANCED BIG DATA WITH APACHE SPARK

A submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

WILLIAM C. SLEEMAN IV

B.S., University of Virginia, USA - August 2000 to May 2004

M.S., Virginia Commonwealth University, USA - August 2004 to May 2007

Director: Bartosz Krawczyk, Ph.D.,

Assistant Professor, Department of Computer Science

Virginia Commonwewalth University

Richmond, Virginia

November, 2021

Acknowledgements

First, I would like to thank Dr. Bartosz Krawczyk for his excellent guidance as my PhD advisor. During this program, I have learned the exact skills I was hoping to and the entire process has gone very smoothly, especially as a part-time student. The work we have done on imbalanced data has been very fulfilling and has given a great foundation for my future research interests.

I would also like to thank my committee members for generously volunteering their time and feedback: Drs. Alberto Cano, Roberto Corizzo, Kostadin Damevski and Michał Woźniak.

I am appreciative of the excellent teachers I had in Computer Science while working towards this degree, and who made learning these new topics more enjoyable. The additional research guidance from Dr. Alberto Cano, who helped me figure out how to actually run Apache Spark jobs, and Dr. Preetam Ghosh, who has been instrumental in my research work in the field of Radiation Oncology, have enhanced my education in machine learning and parallel computing.

I am also grateful for the support I received from Dr. Jatinder Palta and the Department of Radiation Oncology at Virginia Commonwealth University while I completed this degree and the opportunity to apply my newly learned machine learning skills to problems related to cancer care.

Most importantly, I would like to thank my family for their support and understanding while I split my time between work and school over the last few years. Without the help and encouragement of my parents, Cliff and Julia, and my mother-in-law May Ligon Huff, it is unlikely I would have ever attempted going back to school. I would also like to give a special thanks to my son William who had to put up with my extra work the most and my attempts to make him learn Python.

Lastly, I would like to dedicate this work to my late wife Elizabeth F. Sleeman who had inspired me to continue my education.

TABLE OF CONTENTS

List of Algorithms

LIST OF TABLES

LIST OF FIGURES

Abstract

LEARNING FROM MULTI-CLASS IMBALANCED BIG DATA WITH APACHE SPARK

By William C. Sleeman IV

A submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2021.

Director: Bartosz Krawczyk, Ph.D.,

Assistant Professor, Department of Computer Science

With data becoming a new form of currency, its analysis has become a top priority in both academia and industry. The vast amounts of data collected in every domain has led to advancements in high-performance computing and machine learning. However, these large, real-world datasets come with additional complications such as noise and class overlap. Problems are magnified when multi-class data is presented, especially since many of the popular algorithms were originally designed for binary data. Another challenge arises when the number of examples are not evenly distributed across all classes in a dataset. This often causes classifiers to favor the majority class over the minority classes, leading to undesirable results as learning from the rare cases may be the primary goal. Since many of the classic machine learning algorithms were not designed for multi-class imbalanced data or parallel computing, their effectiveness is hindered.

Apache Spark is a distributed computing framework based on the MapReduce model and has been a popular tool for big data analysis. A collection of oversampling

methods, originally designed for small binary class datasets, have been implemented using Apache Spark for the first time. Experimentation showed that not all of these methods translate well to a distributed computing environment, indicating the need for a new generation of algorithms targeting modern high-performance computing.

Previous works also showed that some data examples may be harder to learn from than others. An extensive study was performed to learn how instance level difficulty affects the learning from large datasets. Experimentation showed that the best classifier performance came when considering instance level difficulty during the class balancing process. Similar results were shown with a novel version of UnderBagging which also considered the difficulty of individual examples when constructing bags.

Approximate tree based algorithms are a common way to improve execution time for tasks like nearest neighbors. However, no such methods are currently availability in the standard Apache Spark libraries so the k-Dimensional (KD) tree was implemented for the first time. While often not as accurate as a third-party hybrid-spill tree, the KD-tree still provided similar results and was up to three times faster, showing a good accuracy/speed trade-off. Additional experiments were performed to provide benchmarks on scalability and the effect leaf size on performance.

In the field of Radiation Oncology, physicians must delineate anatomical structures on Computed Tomography (CT) images to aid in the therapeutic delivery of radiation. However, the structure names chosen by the physicians are frequently not standardized making it impracticable for multi-facility studies. An Apache Spark based solution was created to automate the process of standardizing these structure name labels. This method showed high accuracy with some structure types and also highlighted others that are still difficulty to classify. Including both image and dose related information improved results and class balancing was also investigated.

CHAPTER 1

INTRODUCTION

Data in all forms has become a valuable commodity and is being collected at an ever increasing rate. Although the cost of computation and storage has dropped, the complexity of analyzing this data has increased. This new paradigm, now going by the term Big Data, introduces new challenges and opportunities. According to IBM, Big Data is "data sets whose size or type is beyond the ability of traditional relational databases to capture, manage and process the data with low latency" [1]. Big Data is also described with the 5 V's: Volume, Velocity, Variety, Variability and Value [2].

With the understanding of the value of data, Machine Learning (ML) and Artificial Intelligence (AI) methods have permeated almost all business sectors. Fields like banking [3], retail sales [3], advertising [4], and real estate [5] have been utilizing these methods to gain competitive advantages. ML tools and platforms, such as IBM Watson, are now used in healthcare to help discover new drugs and improve patient outcomes [6, 7].

Much of the initial machine learning researched was focused on binary problems; those consisting of only two distinct classes of data. In reality, datasets may have more than two classes that need to be analyzed which makes the task of analyzing the data more difficult. If only two classes are present, the random chance of picking an example is fifty percent but this drops to twenty five percent if four classes are provided. Building the best possible models from this kind of data will require additional insight on which classes might be most difficult to learn from and what methods could be applied to further improve results.

Data imbalance also makes learning from data more difficult. In many real world datasets, the most interesting or critical events to discover make up a small portion of all observations. Many machine learning algorithms tend to favor the majority class over the minority class as it tries to maximize its objection function and this may lead to undesirable results. Ideally, the trained model would provide similar predictive performance on all presented classes.

A major challenge facing analysis of these large datasets is how to process the data in a timely manner. The execution time of many of the popular machine learning algorithms increases in a non-linear fashion as the dataset size grows. In addition, the datasets themselves may be too large to fit on a standard computer which ultimately requires parallel or distributed computing. Message Passing Interface (MPI) is a popular framework for parallelizing algorithms which provides low level access to underlying hardware its running on. However, this comes at the cost of a high software development resources and less portability. The Apache Hadoop system was later created to solve some of these limitations. It uses the MapReduce [8] model originally created by Google and provides a framework for running distributed applications on any supported hardware and operating system. One major limitation to Hadoop is that each stage in the MapReduce process is written to disk which may add significant time penalty. Apache Spark [9] was then created to improve upon Hadoop. While it is also based on the MapReduce model, Apache Spark tries to keep all intermediate stages in main memory and also optimizes the data transformations using a directed acyclic graph (DAG).

This dissertation begins in Chapter 2 with a review of existing challenges with imbalanced data, SMOTE, ensemble learning, minority types and the MapReduce framework. Chapter 3 poses five research questions which are addressed in Chapters 4 – 8. In Chapter 9, future research directions are discussed and Chapter 10 summarizes

the main contributions of this work.

CHAPTER 2

BACKGROUND

Modern systems increasingly generate massive amounts of data, driving the desire and necessity to have algorithms that can learn from this data. The data mining community has developed, and continues to develop, many algorithms that are capable of learning from big data, but the ever increasing volume of data still presents challenges. Datasets may easily be larger than is possible to store on a single machine, or data may arrive continuously as an infinite stream, requiring rapid processing. For this much data, distributed and parallel processing is essential which has led to an increased focus in such platforms.

The improved computer hardware and cheaper storage does not in itself make it any easier to learn from the every growing and diverse data. These real world datasets include noisy data as well as class imbalanced, leading to examples that are difficult to learn from. The following sections provide an overview of approaches that have been applied to these kind of data problems.

2.1 Learning from Imbalanced Data

Imbalanced data occurs when examples are non-uniformly distributed across the available classes in a dataset which can lead to reduced performance of learning algorithms [10]. Traditional machine learning methods tend to be favor the larger (majority) class at the expense of the underrepresented, smaller class (minority). However, in many real world cases the most critical information to learn or discover is present in the minority class. The few anomalous transactions in fraud detection are

the most important to locate just as finding occurrences of rare but serious medical conditions in clinical decision systems. The problem is further complicated when more than two classes is present in a dataset, leading to cases where there can be multiple majority classes, multiple minority classes or a combination of both. Even within these classes, there can be additional inter-class relationships which represents new majority/minority decision boundaries [11].

While imbalance in datasets can make it harder for learning algorithms, challenges are often not from the imbalance itself but class overlapping and noise [12]. If the classes in an imbalanced dataset are well separated, it is still relatively easy to find the correct decision boundaries. However, if there is class overlap it because much more difficult to determine where one class end and the other begins. This problem is made worse with class imbalanced as the majority class is likely to be disproportionately preferred in the overlapping section leading to lower performance for the minority class.

In addition to simple class overlapping, small disjoint clusters and changes of class representation between the training and testing datasets may also exist [13]. Class noise can also severely impact performance and this is more severe in the minority classes [14]. The problem of imbalanced data becomes more difficult when more than two classes are present in the dataset. In these multi-class problems, the same issues of class overlap, disjoint clusters and noise are now magnified with more potential classes to choose from.

Although traditional machine learning has been the basis for most imbalanced learning up to this point [15], more such research is now being directed towards deep learning problems using various approaches. In [16], an evolutionary cost-sensitive deep belief network (ECS-DBN) was designed. This method learns misclassification costs during training to combat class imbalance issues. A study on imbalanced learn-

ing with convolutional neural networks (CNN) [17] said that these networks can be negatively affected by class imbalance as with traditional machine learning. It was also reported that oversampling works well in general and undersampling can be used in cases of extreme class imbalance. Unlike some previous algorithms, CNNs tend to be less susceptible to overfitting when using oversampling.

The strategies for dealing with data imbalance can be divided into two categories: data-level and algorithm-level. Data-level methods include algorithms that perform data preprocessing with the aim of reducing the imbalance ratio, either by decreasing the number of majority observations (undersampling) or increasing the number of minority observations (oversampling). After applying such preprocessing, the transformed data can be later classified using traditional learning algorithms. Algorithm-level methods alter the traditional learning algorithms to eliminate the shortcomings they display when applied to imbalanced data problems. These methods do not modify the training data directly unlike data-level approaches.

2.1.1 Binary Imbalanced Problems

One of the most prevalent data-level approaches is the Synthetic Minority Oversampling Technique (SMOTE) [18] algorithm. It is a guided oversampling technique, in which synthetic minority observations are created by interpolation of the existing instances. Today it is considered a cornerstone for the majority of the following oversampling methods [19]. However, due to the underlying assumption of minority class homogeneity, SMOTE can inappropriately alter the class distribution when factors such as disjoint data distributions, noise, and outliers are present. Numerous modifications of the original SMOTE algorithm have been proposed in the literature. The most notable include Borderline SMOTE [20], which focuses on the process of synthetic observation generation around the instances close to the decision border;

Safe-level SMOTE [21] and LN-SMOTE [22], which aim to reduce the risk of introducing synthetic observations inside regions of the majority class; and ADASYN [23], that prioritizes the difficult instances.

The second category of methods for dealing with data imbalance consists of algorithm-level solutions. These techniques alter the traditional learning algorithms to eliminate the shortcomings they display when applied to imbalanced data problems. Notable examples of algorithm-level solutions include: kernel functions [24], splitting criteria in decision trees [25], and modifications of the underlying loss function to make it cost-sensitive [26]. However, contrary to the data-level approaches, algorithm-level solutions necessitate a choice of a specific classifier. Still, in many cases, they are reported to lead to a better performance than sampling approaches [27].

2.1.2 Multi-class Imbalanced Problems

While performing binary classification, one can easily define the majority and the minority class, as well as quantify the degree of imbalance between the classes. This relationship becomes more convoluted when transferring to the multi-class setting. One of the earlier proposals for the taxonomy of multi-class problems used either: the concept of multi-minority, a single majority class accompanied by multiple minority classes; or multi-majority, a single minority class accompanied by multiple majority classes [28]. However in practice, the relationship between classes tends to be more complicated, and a single class can act as a majority towards some, a minority towards others, and have a similar number of observations to the rest of the classes. Such situations are not well-encompassed by the current taxonomies. Since categorizations such as the one proposed by Napierała and Stefanowski [29] played an essential role in the development of specialized strategies for dealing with data imbalance in the binary setting, the lack of a comparable alternative for the multi-class setting can be

seen as a limiting factor for the further research.

The difficulties associated with the imbalanced data classification are also further pronounced in the multi-class setting, where each additional class increases the complexity of the classification problem. This includes overlapping data distributions, where multiple classes can simultaneously overlap a particular region, and the presence of noise and outliers, where on one hand a single outlier can affect class boundaries of several classes at once, and on the other can cease to be an outlier where some of the classes are excluded. Finally, any data-level instance generation or removal must be done by a careful analysis of how action on a single class influences different types of instances in the remaining classes. This leads to a conclusion that algorithms designed explicitly to handle the issues associated with multi-class imbalance are required to adequately address the problem.

The existing methods for handling multi-class imbalance can be divided into two categories, binarization solutions, which decompose a multi-class problem into either $M(M-1)/2$ (one-vs-one, OVO), or M (one-vs-all, OVA) binary sub-problems [30] where each sub-problem can then be handled individually using a selected binary algorithm. An obvious benefit of this approach is the possibility of utilization existing algorithms [31], however binarization solutions have several significant drawbacks.

Most importantly, they suffer from the loss of information from class relationships. In essence, we either completely exclude the remaining classes in a single step of OVO decomposition or discard the inner-class relations by merging classes into a single majority in OVA decomposition. Furthermore, especially in the case of OVO, the computational cost of decomposition can quickly grow with the number of classes and observations, making the approach ill-suited for dealing with big data problems. Among the binarization solutions, recent literature suggests the efficacy of using ensemble methods with OVO decomposition [32], augmenting with cost-sensitive learn-

ing [33], or applying dedicated classifier combination methods [27].

The second category of methods consists of ad-hoc solutions: techniques that treat the multi-class problem natively, proposing dedicated solutions for exploiting the complex relationships between the individual classes. Ad-hoc solutions require either a significant modification to the existing algorithms, or exploring an entirely novel approach to overcome data imbalance, both on the data and algorithm level. However, they tend to significantly outperform binarization solutions, offering a promising direction for further research. The most popular data-level approaches include extensions of the SMOTE algorithm for the multi-class setting [11, 34], strategies using feature selection [35, 36], and alternative methods for instance generation by using Mahalanobis distance [37, 38]. Algorithm-level solutions include decision tree adaptations [39], cost-sensitive matrix learning [40], and ensemble solutions utilizing Bagging [36, 41] and Boosting [28].

The previously mentioned techniques were developed for two-class big imbalanced data. To the best our knowledge, there exists no algorithms dedicated specifically for large-scale multi-class imbalanced data, neither for GPUs nor CPU clusters [27, 42], apart from our preliminary work on Spark-based ensembles [43].

2.2 Other Class Balancing Methods

The strengths and weaknesses of the original SMOTE algorithm has continued to inspire more recent class balancing solutions. Majority Weighted Minority Oversampling TEchnique (MWMOTE) [44] uses the proximity of majority examples to detect minority examples that are harder to learn in order to guide the synthetic example generation. Another method, Radial-Based Oversampling (RBO) [45], uses a Gaussian radial basis functions to detect regions that should be used for oversampling based on imbalance distributions. This provides an advantage over standard

SMOTE as it takes in account the neighborhood of the examples used for sampling. In a similar fashion, Sampling WIth the Majority (SWIM) [46] uses majority class information for minority class oversampling and generates new minority examples with a similar probability density of existing examples. Clustering has also been used to aid SMOTE based oversampling. KSMOTE [47] uses the k-Means algorithm to split the dataset into two clusters to use SMOTE on subsets of the data. The k-Means SMOTE method [48] performs SMOTE oversampling on clusters that have a high ratio of minority examples.

The increasing popularity of deep learning has lead to a new class of class balancing algorithms using a variant of generative adversarial networks (GAN) [49]. One such method, BAGAN [50], uses an autoencoder to initialize the GAN based on the probability distribution of the training images in the latent space for each class. Both GAMO [51] and BCGAN [52] were designed to generate synthetic examples near the boarder between different classes. GAMO generates examples within a convex hull keeping the new synthetic minority near the class boundary. BCGAN instead identifies borderline examples in the training set to make a new minority class. By using majority, minority and borderline minority classes, the discriminator can explicitly focus on difficult examples while not ignoring those further from the decision boundaries. MFC-GAN [53] maps real classes to fake ones to adjust for poorly generated or discriminated examples with large mismatches between the real and fake classes incurring a higher loss.

GAN based oversampling methods have also been applied to more specific subproblems in imbalanced data. While many of these GAN based methods were designed to deal with high dimensional data, such as images, there has been some work in imbalanced tabular data using Conditional Wasserstein GANs [54]. The Doping with Infrequent Normal Generator (DOPING) [55] also uses an adversarial autoen-

coder (AAE) to train on unlabeled examples for unsupervised anomaly detection. Synthetic examples can then be generated from the resulting latent space by using vectors that a far from the mean latent vector norms. Generative Adversarial Networks and Markov Random Fields (GANSO) [56] was designed for dealing with very small training sets by incorporating information about the structure of the training data. Experiments showed GANSO outperformed SMOTE and performed well even if only five labeled examples were present during training. This method is a good candidate for semi-supervised learning and can help alleviate the time required for manually labeling data.

Although GAN based class balancing methods have shown promising results, they have several limitations with model complexity. The DeepSMOTE [57] method combats some of these issues by not needing a discriminator model and being designed to avoid mode collapse. This method is based on an encode/decoder, SMOTE style oversampling and uses a dedicated loss function. DeepSMOTE was also able to create more realistic looking images when compared to the previous methods of BAGAN and GAMO. In addition, DeepSMOTE was shown to outperform other methods even with very high levels of class imbalance.

2.3 Ensemble Learning

The previous methods may be further improved by employing ensemble methods. These ensembles combine multiple simple models to incorporate their individual strengths to make a single predictor which is more powerful. Bagging [58] is a popular method for ensemble learning which creates a group of classifiers based on bags of data sampled from the total dataset. The majority class can be undersampled with the UnderBagging approach and the minority class can be oversampled with OverBagging. These bagging concepts were extended with SMOTEBagging [59] which uses

SMOTE to class balance bags for training. Boosting [60] works by training a weak learning which it strengthened by adjusting weights on training data based on how hard they were to predict. A popular implementation of this concept is AdaBoost [61] which dynamically adjusts data samples weights based on classification errors in order to build an ensemble of weak learners.

The Random Forest classifier [62] has been one of the most popular ensemble learning algorithms to date. It creates a group of decision trees using bagging and random features selection and combines the resulting trees with a majority vote. This approach was found to be as accurate as AdaBoost but faster. It was also robust to outliers and noise while being easy to run in parallel. This work has lead to other tree based ensembles such as Gradient Boost Trees [63][64][65] and Extremely Randomized Trees [66]. The idea of tree ensembles can be extended by including specific trees to improve the overall ensemble performance. In a recent work, [67] used out-of-bag examples to determine the best trees to add in the final model.

In addition to traditional machine learning, ensemble methods are now being applied to neural network based classifiers. Using a CNN architecture, the Hydra ensemble starts with a coarsely optimized model which is used for generating additional networks [68]. This base model is then used to generate new fine tuned models for the ensemble using different data augmentation methods and class weights. The multi-scale multi-CNN (MSM-CNN) [69] was also created to tackle the problem of skin legion classification. Similar to Hydra, an ensemble of three CNN models was created using different image sizing and scaling factors. For robot arm placement, the random cropping ensemble neural network (RCE-NN) [70] used an ensemble of cropped regions of the target image area. They found this approach worked better than considering the entire image at once because weighting the sub-images can help to mitigate corrupted local information.

2.4 Minority Types

An extension to the idea of noisy data and class overlap is to look at the difficulty of learning from individual examples for data cleaning, clustering or classification. Traditionally, examples were considered as safe or unsafe but can be further broken down into four categories: safe, borderline, rare, outlier [71]. For each example in the dataset, the proportion of k nearest neighbors with the same class label is determined. From that ratio, a difficulty type is assigned:

- Safe: $0.8 \leq ratio \leq 1.0$
- Borderline: $0.4 \leq ratio < 0.8$
- Rare: $0.2 \leq ratio < 0.4$
- Outlier: $ratio < 0.2$

It was shown that error rates increase from the safe to outlier minority types, suggesting that some examples will be harder to learn from than others. In addition, minority classes tend to have a higher proportion of the more unsafe types which may be one of the reasons minority classes often perform worse than the majority class. Most datasets contain some combination of these difficulty types in the minority classes but with different ratios [29]. Some minority classes could also be primarily outlier or rare examples so treating them as noise may not be appropriate. Results from [29] showed that global imbalance ratios were not as influential as the minority type for classification. Adjusting the k value for the neighborhood can change the proportion of minority type ratios for a given dataset.

For each dataset, there may be a combination of minority types that should be used for class balancing to improve predictive accuracy [11]. Using SMOTE on

data filtered by some combination of minority types improved results for the C5.0 algorithm compared to using SMOTE on the entire dataset for a number of datasets [72]. Again, the overall imbalance ratio was shown not to be as significant as the properties of the data itself.

While most classifiers ignore the difficulty to learn from individual examples, the boosting with instance difficulty invariance (BIDI) classifier was proposed to address this issue [73]. Using of an ensemble of classifiers, the difficulty of each example is determined by the rate it was correctly predicted. Using a [0, 1] scale, examples are given a difficulty score where 0 is the easiest and 1 is the hardest. At each round of training, more difficult examples are introduced while still ensuring the easy examples are correctly classified. This method showed good results with 18 UCI datasets [74], but the proposed algorithm did not address multi-class or very larger datasets which provides new opportunities with future research.

2.5 MapReduce

As the rate of data creation has continued to increase, having methods to process it effectively has become more critical. Google addressed this issue with their MapReduce model [8] which has led to the development of the distributed computing frameworks Hadoop and Spark. In this section, we discuss MapReduce, the Spark architecture, distributed data management and available machine learning tools.

MapReduce is a distributed computing model for processing data represented as key/value pairs with three main stages: map, shuffle, and reduce. The map phase takes the input data and splits it between multiple worker processes which may reside on different processors, cluster nodes, or distributed computers. Processing is done in parallel on each worker and the intermediate results are sent to the reduce phase which completes the task. Intermediate results may need to be shuffled between phases if a

certain data grouping is required. Multiple map/reduce tasks can be chained together to form more complicated transformations.

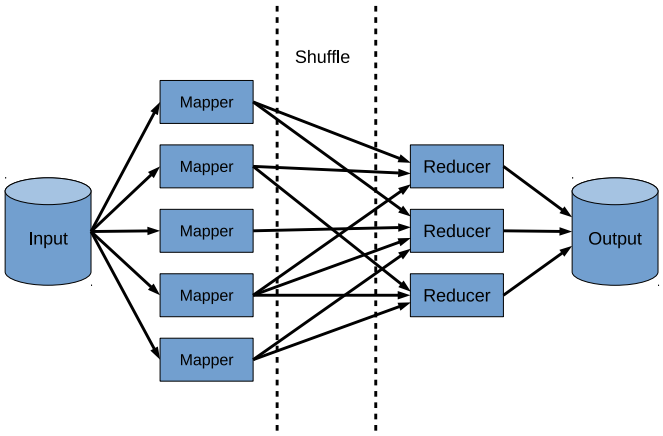


Figure 1: The three phases of the MapReduce model: map, shuffle, reduce.

2.5.1 Apache Spark Architecture

Open source implementations of the MapReduce model includes Hadoop [75] and Spark [76] that are designed to work with the Hadoop Distributed File System (HDFS) [77]. HDFS provides redundancy, data streaming support, and the ability to handle datasets in the order of terabytes. While Hadoop and Spark follows the MapReduce model and can use HDFS, they differ on how data is managed between stages. Between each independent map/reduce step, Hadoop stores the results to HDFS and potentially introducing a significant time penalty.

Spark instead stores the results between map/reduce steps in main memory when possible. Unlike Hadoop, Spark can look ahead and generate a directed acyclic graph (DAG) based on the chained map/reduce steps. In this context, individual computational tasks are ordered by dependencies so that redundant operations can be removed and each operation is performed only when its result is needed for another calculation.

From the driver node, a Spark Context object is created which connects to a cluster manager such as Mesos or YARN. The Spark Context then requests resources from the cluster manager and acquires executors on the cluster nodes. The executor processes manage the task computations on one or more CPU threads. In Figure 2, dashed lines shows the Spark job set up with the driver node process acquiring resources from the cluster manager. The driver node and executor processes pass data around as tasks are completed, shown in solid lines.

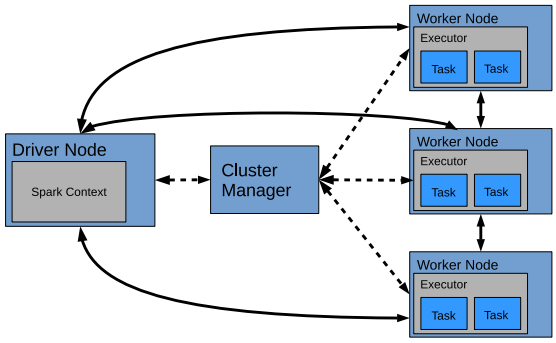


Figure 2: The Spark cluster architecture for resource allocation and data transfer

Spark uses the Resilient Distributed Dataset (RDD) [78] data structure to abstract data for parallel computations. The RDD stores a dataset as partitions that may be split over one or more cluster nodes. This abstraction allows the user to perform the same operations regardless how the data is distributed. Functions like map, filter, and reduce can be run directly on RDDs and SQL-like table operations are available through the further abstracted data structures of DataFrames and Datasets.

2.5.2 Machine Learning with Apache Spark

The Apache Spark platform comes with its Machine Learning library (MLlib) [79] that includes a number of tools for data transformation, classification and clustering with the support for third party algorithms. The resent versions of Apache Spark

support the DataFrame and DataSets data structures which allow for the use of SQL-like operations while also utilizing the optimized DAG. However, one potential downside of this platform is the high main memory usage. Attempting to keep all operations in memory can greatly reduce run time but can stress the physical limits of the host system and may require fine tuning of the algorithm in use to work optimally. Apache Spark can also be used in a truly distributed system such as Amazon Web Services (AWS) Elastic MapReduce (EMR) which greatly increases the upper limit of addressing big data problems.

Although the Apache Spark MLlib mostly includes traditional machine learning algorithms, it does support the multilayer perceptron classifier (MLP). In [80], a cascade learning approach was applied by using linear regression to include additional information before training a MLP network. More advanced deep learning methods like Long Short-Term Memory (LSTM) were also implemented within the Spark framework [81]. Using the Deeplearning4j Java library, a LSTM network was created that could be called directly from a Spark pipeline. Because Spark uses the Java Virtual Machine (JVM) under the hood, integrating Java based code is straightforward.

Another work [82] illustrated a potential performance bottleneck with some machine learning algorithms implemented in Apache Spark. The stochastic gradient descent (SGD) algorithm is used by several other methods including linear regression and support vector machines (SVM) and can cause excessive communication between the driver node and the executors. To avoid this communication inefficiency, the SGD method was improved with model averaging where gradients at each executor are processed locally and the final results are averaged.

2.5.3 Specialized Hardware Support

In recent years, a rise of distributed computing cluster architectures has been observed, especially ones using the MapReduce methodology [83] such as Apache Hadoop and Spark. They offer high elasticity, reliability, and scalability in an user-friendly manner. However, this comes at the cost of increased monetary price of hardware and lack of explicit controlling of scheduling that is available in GPUs. While CPU clusters seem like a highly attractive solution to imbalanced data, one must be aware of their potential limitations. A study on effects of different resampling techniques on Random Forest showed that SMOTE achieves a highly unsatisfactory performance in the MapReduce environment [84]. This is caused by the lack of control over data partitioning and in turn creating chunks of data with corrupted spatial relationships. When SMOTE is used in each map with such an unreliable neighborhood, it tends to create artificial instances in incorrect regions, leading to increasing overlapping among classes and shifting true class distributions. Random oversampling and undersampling performs significantly better on CPU clusters. Success stories of MapReduce usage for imbalanced big data include efficient and scalable feature selection [85], rule induction [86], and data resampling [87]. Even though Apache Spark was designed for distributed CPU based computing, it can still benefit from other specialized hardware like GPUs and Field Programmable Gate Arrays (FPGA).

Graphic Processing Units (GPUs) offer a powerful high-performance computing environment at a fraction of the cost of a traditional cluster [88]. While they are characterized by an excellent data-level parallelism and unbeatable degree of control over each aspect of data scheduling and partitioning, they require highly specialized skills to write efficient and optimized code. Additionally, at the current moment they are unable to handle terabyte-level datasets, due to limitations on the embedded

memory.

In the context of imbalanced data, the main success of GPUs lies in a highly efficient implementation of the popular SMOTE technique. Due to a full control over data partitioning, one can maintain the meaningful creation of artificial instances in given neighborhoods - a feat that is unreachable in most of CPU clusters. GPUs have been successfully applied to skew-insensitive variants of nearest-neighbor classifiers [89], inducting classification rules using genetic programming [90], as well as efficient large-scale discretization of imbalanced data [91].

Previous works with joining Spark and GPUs have shown performance improvements with real time data streams [92], k-Means, linear regression, clustering and single value decomposition [93]. One paper [94] proposed a Spark modification to support the use of GPUs by invoking CUDA kernels for computationally intensive tasks. Several caching methods were investigated to reduce network overhead, and experiments using both local and remote GPUs showed significant speed improvements. The ability of Spark to consume huge amounts of data was also leveraged to partition and map satellite image data for scheduling GPU processing on worker nodes [95]. The HetSpark [96] framework was developed for extending the Spark architecture to include heterogeneous executors, including both GPU and FPGA hardware. Some limited support for GPUs has been included in Apache Spark starting with version 3.0.

Field Programmable Gate Arrays (FPGA) are integrated circuits that can be programmed to implement algorithms at the hardware level, giving excellent performance on specialized tasks. These FPGAs can then be connected to CPU based computational units providing additional hardware acceleration. As demonstrated with k-Means [97] and 2-D Fast Fourier Transforms [98], FPGAs can be successfully integrated in Spark clusters.

While these hardware acceleration methods have been shown to improve execution time, they are not officially supported by Apache Spark and so, using non-CPU based hardware remains a non-trivial task.

CHAPTER 3

PROPOSED RESEARCH QUESTIONS

In the field of machine learning, there are many difficult and important problems that still remain as challenging tasks. In this dissertation, I primarily focus on problems in the areas of class imbalance, instance level difficulty, ensemble learning and MapReduce based algorithms. Here, I propose the following research questions (RQ):

RQ1: How do oversampling algorithms behave when used within the MapReduce framework? Oversampling has been a popular method to address class imbalance and so many such algorithms have been written over the years. However, many of the well known algorithms, including SMOTE, were originally designed for small, binary class problems. Today, with datasets getting progressively larger and multi-class problems showing up more frequently, new learning approaches are required. In Chapter 4, I introduce the first MapReduce based oversampling methods written in Scala for Apache Spark. In addition, extensive experimentation was performed to evaluate how these algorithms work on Spark as well as an in-depth discussion on what algorithmic features make such methods successful in this distributed computing framework.

RQ2: What role does instance level difficulty play in the context of big data classification? The majority of classification problems have treated the importance of all training examples the same. However, resent research has suggested that individual examples should be treated differently depending on how hard they are to correctly label. In Chapter 5, I provide the results from extensive experimentation on classifier performance when considering different kinds of instance level difficulty

on five large multi-class datasets.

RQ3: Do traditional bagging methods benefit from instance level difficulty information? Bagging is an ensemble method of training a group of classifier with different subsets of the entire training data. Like other machine learning methods, traditional bagging does not consider how hard it is to classify individual examples when creating the bags of data. In Chapter 6, I compare results between multiple bagging techniques, including the proposed approach of using instance level difficulty, using several base classifiers on Apache Spark.

RQ4: Can KD-trees be effectively implemented using the MapReduce framework? Tree based algorithms are powerful tools for both supervised and unsupervised learning. While there have been many implementations of these algorithms, there are very few for Apache Spark. The K-Dimensional (KD) Tree algorithm is one of the most straightforward methods but had not been implemented on Spark. Chapter 7 provides an implementation of the KD-tree on Spark, its performance compared to a popular hybrid spill tree implementation and experimentation on the use of clustering with KD-trees.

RQ5: Can class imbalanced radiotherapy structure set data be accurately classified? The field of Radiation Oncology, which treats cancer with therapeutic radiation, is now putting more focus on how AI and machine learning can improve clinical practice. Data standardization is required before large multi-institutional datasets can be created, which is required for any big data research as each facility may only treat up to 1,000 patients a year. One of required steps in radiotherapy is the delineation of anatomical structures which helps to define how radiation should be delivered. However, the names chosen by physicians are often not standardized making it difficult to calculate Quality Measures based on the dose delivered to specific parts of the body. In Chapter 8, I present a method to automate

the process of standardizing these names and experimental results with some new class balancing methods.

CHAPTER 4

IMBALANCED BIG DATA OVERSAMPLING

4.1 Introduction

Learning from imbalanced data poses significant challenge to most of machine learning algorithms. Standard classifiers were designed to learn from roughly balanced class distributions. Therefore, if one of the classes becomes a predominant one (i.e., majority class), the decision boundary becomes biased towards it. This is rooted in loss functions that assumes uniform costs among all instances. Thus, the more frequent and easier majority class will dominate the training procedure, leading to a poor performance on minority class. This learning difficulty can be further augmented by various instance-level characteristics, such as overlapping distributions, borderline examples, small disjuncts, high dimensionality [99] or noisy class labels. When dealing with more than two classes, relationships among them become much more complex, leading to even more challenging learning scenarios. Despite over two decades of progress in this field, imbalanced data problem is as vital as ever. This can be contributed to its prevalence in emerging real-world applications, as well as novel problems accompanying class imbalance, such as streaming nature of data, concept drift, or complex data representations. The emergence of the deep learning paradigm has only boosted the need for effective ways of handling skewed distributions, while introducing novel challenges, such as long-tailed recognition, or the need for skew-insensitive generative models.

Class imbalance, while challenging on its own, is often accompanied by big data volume. Having skewed and massive datasets calls for dedicated high-performing ar-

chitectures, such as Apache Spark. While highly efficient, these computing paradigms require specific ways of implementing algorithms on them that will allow to harness the power of their distributed processing. Oversampling methods are abundant for standard imbalanced data sets, but lack effective implementations for big data problems.

Research goal. To propose the first, self-contained and holistic work on oversampling for learning from imbalanced big data that encapsulates all the crucial aspects of this domain: (i) taxonomy and details of modern oversampling methods applicable to massive datasets; (ii) software package encompassing of the most efficient oversampling algorithms implemented for MapReduce and Apache Spark environments; (iii) a thorough and detailed experimental study on both binary and multi-class imbalanced big data; and (iv) guidelines and future directions for developing novel oversampling algorithms for imbalanced big data.

Motivation. While there is a plethora of oversampling algorithms proposed for single-CPU environments, most of them cannot be directly applied to high-performance computing environments, such as Spark. Distributed architecture of such systems poses unique challenges for the design of oversampling algorithms and require specific solutions tuned to work in the MapReduce environment. There is a need to identify which specific components of oversampling can be easily transferred to distributed computing setups and how to do this in an efficient manner. Until this point, there were no dedicated software packages that offered any oversampling algorithms more advanced than SMOTE for MapReduce systems. Finally, there is a need to propose an unified taxonomy for imbalanced big data oversampling, as well as outline the directions for future research and challenges awaiting researchers in this important domain.

Summary. We propose a complete and holistic work on the oversampling of big imbalanced data, merging both the theoretical, survey-style input with practical, software and empirically oriented contributions [100]. We propose a thorough taxonomy of existing oversampling methods that is designed to identify the most suitable methods that can work in big data context. Then, we discuss in details the adaptation of 14 diverse oversampling algorithms to high-performance computing environments, highlighting their useful properties and implementation difficulties. This is followed up with a complete software package that can be used by any researcher. We extend this by a thorough experimental analysis of those oversampling algorithms on 26 binary and multi-class imbalanced big data benchmarks. Finally, we offer design guidelines for future researchers on how to create efficient oversampling methods for distributed environments, as well as discuss open challenges and future directions for learning from imbalanced big data.

Main contributions. This works addresses the discussed challenges in learning from imbalanced big data with the following research contributions:

- **Taxonomy of oversampling algorithms.** We propose the first detailed taxonomy of oversampling algorithms that analyzes the main groups of methods based on their instance generation mechanisms and how they utilize inner- and intra-class information.

- **Adaptation of popular oversampling algorithms to MapReduce and Spark environments.** We show a detailed way of adapting 14 popular and effective oversampling algorithms to Spark environment in a way that leverages their usefulness and captures the unique demands of high-performance computing environments.

- **Software package for big data oversampling.** We propose the first com-

plete software package written in the Scala language that offers optimized and efficient implementations of 14 oversampling methods for imbalanced big data. This will increase the reproducibility of future works in this domain and serve as a baseline for researchers who will develop their own novel algorithms.

- **In-depth experimental study.** We present a thorough experimental study on 26 imbalanced big data benchmarks that capture both binary and multi-class problems. We investigate performance of oversampling techniques based on 4 popular performance metrics, their relationships with different types of data and base classifiers, as well as accuracy/computational complexity trade-offs.

- **Design guidelines for imbalanced big data oversampling.** Based on our experiences with implementing various oversampling methods for high-performance distributed environments, we propose a set of guidelines for researches on what should be taken into account when developing novel oversampling methods and what mechanisms works favorably within the MapReduce framework.

- **Open challenges and future directions for imbalanced big data oversampling.** We outline perspectives and challenges that should be addressed by the research community while designing novel oversampling algorithms for imbalanced big data.

4.2 Oversampling Algorithms for Imbalanced Big Data

4.2.1 Taxonomy of Oversampling Algorithms for Big Data

Blind vs guided oversampling is the first-level taxonomy of oversampling algorithms that is based on using any data-level information to create artificial instances for minority classes:

- **Blind oversampling.** This approach assumes random multiplication of existing instances by either copying them in order to increase the minority class size, or by adding noise / jitter (so-called smearing) to create new artificial instances that are not the exact copies of existing ones.

- **Guided oversampling.** This approach uses various class-level and instance-level properties to generate new artificial instances in a way that increases the size of the minority class. It also decreases the difficulty of learning from this class by making the decision boundary better aligned towards a balanced recognition of all classes.

Guided oversampling in details is the second-level taxonomy of oversampling algorithms that is based on what and how data-level information is utilized to create artificial instances for minority classes:

- **Using global or local information:**

  - **Basic local information.** Guided oversampling aims at targeted injection of artificial instances into the minority class in a way that will lower the bias, while preserving class characteristics. SMOTE achieved this by incorporating basic information about the class structure by finding k nearest neighbors for each minority instance and injecting artificial samples along hyperplanes between them. This idea was later extended into taking into account the composition of the neighborhood [101] and ratio between minority and majority instances within them [20, 21].

  - **Advanced local information.** With the success of using local information came the observation of non-uniform minority class properties over the feature space. This showed that not only each neighborhood instance

may have different importance while using it for oversampling, but also that the size of neighborhood and the resampling ratio should be adapted to each locality [102]. More advanced local information can be extracted in a form of discrete or continuous taxonomy of minority instance types [11], or by analyzing the strength of each class presence in a given area [103].

– **Basic global information.** Alternative approach postulated that one of the biggest learning difficulties embedded in minority classes may be their potential for being of a multi-modal nature [104]. Ignoring this fact would lead to oversampling that increases overlapping between minority and majority classes, thus effectively further enhancing the bias instead of alleviating it. Restricting oversampling to individual modalities (e.g., by clustering or density analysis) will therefore not affect decision boundaries in a negative way. Another type of popular global information includes noise [105, 102] and outlier analysis [106] in order to exclude potentially harmful or incorrect instances from being used as a resamling seed.

– **Advanced global information.** While modalities and data perturbations are important factors, one can gain even deeper insight into minority class structure. This is especially important in the case of multi-class problems, where each individual minority class may have different properties. Therefore, global information can be used to guide the oversampling for each class independently, by taking into account such factors as class compactness, presence of substructures, or substructure size and complexity.

– **Combining local and global information.** Local and global information can be effectively combined, to detect minority class properties and

modalities, and then analyze instance-level difficulty within each modality in order to offer locally adaptive resampling that follows global class characteristics [103].

- **Using nearest neighbor distances:**
  - **Euclidean-based neighborhood.** Most of guided oversampling methods rely on a neighborhood definition that allows for injecting the artificial minority instances. This is usually connected with the k-nearest neighborhood that was first utilized by SMOTE and then adapted by other oversampling algorithms. Eucliudean distance is commonly used for vector comparison due to its ease of computation and well-understood properties when dealing with uniform feature types [107].
  - **Non-Euclidean neighborhood.** Euclidean distance is subject to various limitations, such as its ineffectiveness in handling mixed feature types or a higher number of dimensions [108]. Therefore, non-Euclidean alternatives have been used in oversampling, such as Mahalanobis distance (due to its effective properties for modeling more complex manifolds or class boundaries), Hellinger distance [109], other L_k norms with $k > 1$ and manifolds [110, 111], kernel density functions [102], or metric learning [112, 113].
- **Inter-class vs intra-class relationships:**
  - **Inter-class relationships.** This family of methods focuses on extracting information only from the minority class itself. This can either include local or global information, aiming at enriching the minority class in such a way that will empower its presence in the decision space while alleviating its potential difficulties [114], such as noise or presence of atypical observations

[72].

  - **Intra-class relationships.** Using only inter-class properties can lead to improper performance of oversampling methods, as improving the class itself may harm the remaining classes. To avoid this, modern oversampling methods take into account not only the target minority class itself, but also the remaining classes from the dataset [103]. This helps to define class margins or areas with high probability/potential of belonging to a certain class [102], allowing for the injection of artificial instances so that the overlapping between classes [115] will not increase, offering a more balanced performance on all classes instead of replacing one bias with another.

- **Oversampling ratio:**

  - **Fixed oversampling ratio.** The majority of oversampling methods take as a user input how many artificial instances for the minority class should be created (i.e., oversampling ratio). Most studies show that oversampling should aim at balancing minority and majority classes [116]. However, in the case of extreme class imbalance the combination of minority oversampling with majority class undersampling leads to better results [117]. Other resampling methods learn the oversampling ratio on their own, fixing it for each dataset independently [118]. However, they still use this fixed ratio over the entire feature space.

  - **Adaptive oversampling ratio.** As minority classes may have multimodal structure, it may be beneficial to adapt the oversampling ratio to each modality. Some of them may be easier to analyze, while the remaining ones require to be more empowered in order to reduce bias imposed on

Algorithm 1 Transform

Input: DataFrame of instances that needs to be class balanced
Parameters: DF: DataFrame of instances
Output: Oversampled DataFrame

function: transform(DF)
$classDFs \leftarrow DF$.groupBy($classLabel$)
$majorityClassLabel, majorityClassCount \leftarrow$
  $max(classExamples$.map$(x \rightarrow x.count))$
$examplesToAdd \leftarrow classExamples$.map$(x \rightarrow$
  $majorityClassCount - x$.count$())$
$oversampledClasses \leftarrow DFs$.map$(x \rightarrow$
  ClassBalance*$(x, examplesToAdd(x.label)))$
return $union(oversampledClassses)$

them by a classifier [102, 119]. Furthermore, in multi-class settings, the oversampling ratio could be adapted dynamically not only for within class structures, but also for each class independently [103]. Finally, adaptive oversampling ratios are necessary under non-stationary data properties, where the resampling algorithm must be able to constantly adapt how many artificial instances are generated at a given moment [120].

4.2.2 Details of Oversampling Algorithms

In this section, a short description of each sampling method is presented with high level pseudo code. These algorithms were implemented as described in their original publications as closely as possible, although some changes were required to work efficiently within the MapReduce programming model. All of these algorithms were originally designed for binary class problems so multi-class supported was added by performing oversampling on all minority classes using a one-vs-all pattern. Algorithm 1 shows the process of oversampling the minority classes, where the **ClassBalance*** method is a placeholder for one of the presented sampling methods.

ADASYN [23] begins with kNN applied to the entire dataset and then the nearest neighbors for each minority example is returned. An imbalance ratio of nearest neigh-

Algorithm 2 ADASYN

Input: DataFrame of the full dataset
Parameters: DF: DataFrame of all examples in the dataset, *minorityLabel*: label of the minority class, *totalExamplesToAdd*: number of synthetic examples to create, k: neighbor count for kNN
Output: Synthetic examples

function: ADASYN($DF, minorityLabel, totalExamplesToAdd, k$)
$minorityDF \leftarrow DF$.filter($label = minorityLabel$)
$knn \leftarrow$ KNN.fit(DF)
$nearestNeighbors \leftarrow knn$.transform($minorityDF$)
$nearestNeighbors[majorityRatio] \leftarrow nearestNeighbors$
.map($x \rightarrow x.neighbors.tail$.filter($label \neq minorityLabel$).count())
$majorityRatioSum \leftarrow nearestNeighbors[majorityRatio]$.sum()
$nearestNeighbors[examplesToAdd] \leftarrow nearestNeighbors$
.map($x \rightarrow (x[majorityRatio]$ / $majorityRatioSum)$
$* totalExamplesToAdd$)
return $nearestNeighbors$.map($x \rightarrow$ createSyntheticExamples(x))

function: createSyntheticExamples($example$)
return (0 to $example[examplesToAdd]$).map($x \rightarrow$createExample(x, k))

function: createExample($example, k$)
$example \leftarrow example.neighbors[0]$
$randomNeighbor \leftarrow example.neighbors.tail[randomInt(0, k)]$
return Array($example, randomNeighbor$).transpose
.map($x \rightarrow x[0]$ + randomDouble() $* (x[1] - x[0])$)

bors is calculated for each minority example and then all of these ratios are summed. The number of synthetic examples to be created from each minority example is proportional to the level of its class imbalance. If a minority example has no minority neighbors, simple example replication is performed, otherwise synthetic examples are created between a target example and one of its minority class neighbors. ADASYN was one of the earliest algorithms that attempted to utilize the local instance difficulty factors in order to introduce artificial instances in a more guided manner that would reduce the complexity of a classifier's decision boundary.

ANS [121] starts with a 1-NN of the minority class and then counts the number of majority examples within these radii. Minority examples that are found to have many majority examples nearby are then removed. From the remaining minority ex-

amples, the maximum distance between nearest neighbors is determined and used for the radius limit for distance based nearest neighbor calculation. From these nearest neighbor results, the number of majority examples close to each minority class example is counted. Each majority neighbor count is divided by the total sum to produce a probability score that is then used for generating SMOTE style synthetic examples. ANS is an extension of SMOTE-based oversampling that adds a data cleaning step in order to reduce the number of irrelevant or noisy instances that will be used as SMOTE input.

Algorithm 3 ANS

Input: DataFrame of the full dataset
Parameters: DF: DataFrame of all examples in the dataset, *minorityLabel*: label of the minority class, *totalExamplesToAdd*: number of synthetic examples to create, k: neighbor count for kNN, *CMaxRatio*: ratio value for outcast detection
Output: Synthetic examples

function: ANS($DF, minorityLabel, totalExamplesToAdd, k, CMaxRatio$)
$minorityDF \leftarrow DF$.filter($x \rightarrow x.label = minorityLabel$)
$majorityDF \leftarrow DF$.filter($x \rightarrow x.label \neq minorityLabel$)
$CMax \leftarrow DF$.count() $*$ $cMaxRatio$
$minorityKnn \leftarrow$KNN.fit($minorityDF, returnDistances = True$)
$minorityNeighbors \leftarrow minorityKNN$.transform($minorityDF$)
$minorityNeighbors[closestMinorityDistance] \leftarrow minorityNeighbors$
.map($x \rightarrow x.neighbors$.filter($y \rightarrow y.label = minorityLabel$)[0].$distance$)
$outBorderKnn \leftarrow$KNN.fit($majorityDF$)
$outBorderNeighbors \leftarrow outBorderKnn$.transform($minorityNeighbors$)
$outBorderExamples \leftarrow outBorderNeighbors$
.map($x \leftarrow x.neighbors$.filter($closestMinorityDistance < x.radius$))
$outBorderExamples[outBorder] \leftarrow outBorderExamples$
.map($x \rightarrow x.neighbor$.count())
$outBorderCounts \leftarrow outBorderExamples.select(outBorder)$

$previousOutcasts \leftarrow -1$
for $c = 1$ to $CMax$ **do**
  $numOfOutcasts \leftarrow outBorderCounts$.filter($x \rightarrow x \geq c$).sum()
  if $|numOfOutcasts - previousOutcasts| = 0$ **then**
    $C \leftarrow c$
    if $outBorderExamples$.filter($outborder < C$).count() > 0 **then**
      break
    end if
  end if
  $previousOutcasts \leftarrow numOfOutcasts$
end for

$Pused \leftarrow outBorderExamples$.filter($outBorder < C$)
$maxClosestMinorityExample \leftarrow Pused$.select($closestMinorityDistance$).max()
$PusedKnn \leftarrow$KNN.fit($Pused$,
  $searchRadius = maxClosestMinorityExample$)
$PusedDistances \leftarrow PusedKnn$.transform($Pused$)
$Pused[neighborCount] \leftarrow PusedDistance$
.map($x \rightarrow x.neighbors$.count())
$neighborCountSum \leftarrow Pused[neighborCount]$.sum()
$Pused[examplesToAdd] \leftarrow Pused$.map($x \rightarrow$
  ($x[neighborCount]$ / $neighborCountSum$) $*$ $totalExamplesToAdd$
return $Pused$.map($x \rightarrow$generateExamples($x, x[examplesToAdd, k]$))

function: generateExamples($example, samplingRatio, k$)
return (0 to $samplingRatio$)
.map(_ $\rightarrow$ generateSingleExample($example, k$))

function: generateSingleExample($example, k$)
$example \leftarrow example.neighbors$[0] 35
$randomNeighbor \leftarrow example.neighbors.tail$[randomInt($0, k$)]
return Array($example, randomNeighbor$)
.map($x \rightarrow x[0]$ + randomDouble() $*$ ($x[1] - x[0]$))

Borderline SMOTE [20] performs oversampling from minority class examples that have multiple majority class neighbors. kNN is performed and the minority class data is filtered to only include examples that share *m/2* to *m-1* majority neighbors (labeled as *danger*), where m is the number of nearest neighbors. kNN is performed again, but only on the minority class and the nearest neighbors for the *danger* examples are returned. Each *danger* example is used to create synthetic examples as in standard SMOTE. Borderline SMOTE utilizes the concept of introducing artificial instances in the uncertainty area between classes, thus aggressively countering the bias towards majority class.

Algorithm 4 Borderline SMOTE

Input: DataFrame of the full dataset
Parameters: DF: DataFrame of all examples in the dataset, *minorityLabel*: label of the minority class, *totalExamplesToAdd*: number of synthetic examples to create, k: neighbor count for kNN
Output: Synthetic examples

function: BorderlineSMOTE($DF, minorityLabel, totalExamplesToAdd, k$)
$knnModel \leftarrow$ KNN.fit(DF, k)
$nearestNeigbors \leftarrow knnModel$.transform($minorityDF$)
$dangerDF \leftarrow nearestNeighbors$.filter(isDanger($nearestNeighbors[neighbors.labels], k$))

$samplingRate \leftarrow totalExamplesToAdd$ / $dangerDF$.count()
$dangeKnnModel \leftarrow$ KNN.fit($minorityDF, k$)
$dangerNeigbors \leftarrow dangerKnnModel$.transform($dangerDF$)
return (0 to $examplesToAdd$)
.map($x \rightarrow$ generateExample($dangerNeigbors[x], samplingRate$))

function: isDanger($labels, k$)
$label \leftarrow labels[0]$
$majorityNeighbors \leftarrow labels.tail$
.map($x \rightarrow if\ x = label : 0; else : 1$).sum()
if $k/2 \leq majorityNeighbors < k$ **then**
return $True$
else
return $False$
end if

function: generateExamples($dangerExample, samplingRate$)
$example \leftarrow dangerExample.neighbors[0]$
$randomNeighbors \leftarrow$ (0 to $samplingRate$)
.map($x \rightarrow example.neighbors.tail$[randomInt($0, k$)]
return $randomNeighbors$.map($x \rightarrow$ generateSingleExample($example, x$))

function: generateSingleExample($example, neighbor$)
return Array($example, neighbor$).transpose
.map ($x \rightarrow x[0]$ + randomDouble() * ($x[1]$ - $x[0]$)))

CCR [117] has two phases for class balancing: cleaning the minority class regions from majority class examples and oversampling in the cleaned regions. First, an energy budget is set and a radius is expanded iteratively until the budget has been spent. Majority class examples in that region are pushed out the distance of the radius. Synthetic examples are created in the cleaned minority class regions proportionally to favor original minority class examples within small radii, thus creating more examples in the most difficult regions. CCR combines data cleaning with oversampling, but instead of removing instances it relocates them. That allows for creating safe regions for oversampling without discarding useful information about majority class.

Algorithm 5 CCR

Input: DataFrame of the full dataset
Parameters: *DF*: DataFrame of all examples in the dataset, *minorityLabel*: label of the minority class, *totalExamplesToAdd*: number of synthetic examples to create, *engeryBudget*: amount of energy to expend when finding the minority class radii
Output: Synthetic examples

function: CCR($DF, minorityLabel, totalExamplesToAdd, energyBudget$)
$minorityDF \leftarrow DF.filter(x \rightarrow x.label = minorityLabel)$
$majorityDF \leftarrow DF.filter(x \rightarrow x.label \neq minorityLabel)$
$minorityDF[radius] \leftarrow minorityDF$.map($x \rightarrow$findRadius$(x, energyBudget)$)
$movedMajorityExamples \leftarrow minorityDF$
  .map($x \rightarrow$moveMajorityExamples$(x, majorityDF)$)
$uniqueMajorityExamples \leftarrow movedMajorityExamples$.groupBy($minorityDF.index$)
  .map($x \rightarrow x$[randomInt(0, $x.count()$)])
$minorityDF[inverseRadius] \leftarrow 1$ / $minorityDF[radius]$
$inverseRadiusSum \leftarrow minorityDF[inverseRadius]$.sum()
return $minorityDF$.map($x \rightarrow$generateExamples$(x, inverseRadiusSum, totalExamplesToAdd)$)

fu nction: generateExamples($example, inverseRadiusSum, totalExamplesToAdd$)

$gi \leftarrow (example[inverseRadius]$ / $inverseRadiusSum)$
  $* totalExamplesToAdd$
$syntheticExamples \leftarrow$ (0 to gi).map(_ $\rightarrow$ (0 to $example$.transpose
  .map($x \rightarrow$ randomChoice(-1, 1) $*$ randomDouble()
  $* example[radius]) + x)$
return $syntheticExamples$

function: moveMajorityExamples($example, minorityLabel$)
$majorityNeighbors \leftarrow example.neighbors.tail$
  .filter($label \neq minorityLabel$)
$majorityNeighbors[withinRadius] \leftarrow majorityNeighbors$
  .map($x \rightarrow$withinRadius$(example, x, example[radius])$)
$majorityNeighbors \leftarrow majorityNeighbors$.filter($withinRadius = 1$)
$majorityNeighbors[distance] \leftarrow majorityNeighbors$
  .map($x \rightarrow$Array$(examples, x)$.transpose.map($x \rightarrow$abs($x[0] - x[1]$))
return $majorityNeighbors$.map($x \rightarrow$ shiftMajorityExample(x)))

function: shiftMajorityExample($majorityExample, minorityExample$)
$distance \leftarrow majorityExample[distance]$
$radius \leftarrow minorityExample[radius]$
return Array($majorityExample, minorityExample$).transpose.map($x \rightarrow x[0] +$ $((radius - distance)$ / $distance) * (x[1] - x[0])$)

function: findRadius($example, energyBudget$)
$energy \leftarrow energyBudget$
$radius \leftarrow 0$
while $energy > 0$ **do**
  $movedMajorityExamples \leftarrow minorityWithRadius$
    .map($x \rightarrow$moveMajorityExamples$(x, majorityDF)$)
  $uniqueMajorityExamples \leftarrow movedMajorityExamples$.groupBy($index$)
    .map($x \rightarrow$randomChoice(x))
  $deltaR \leftarrow energy$ / NoP($example, radius$)
  if NoP($example, radius + deltaR$) >NoP($example, radius$) **then**
    $deltaR \leftarrow$ distance to nearest majority example outside of radius
  end if
  $radius \leftarrow radius + deltaR$
  $energy \leftarrow engery - deltaR *$ NoP($example, radius$)
end while
return $radius$

function: NoP($example, radius, minorityLabel$)
$movedMajorityExamples \leftarrow minorityWithRadius$
  .map($x \rightarrow$moveMajorityExamples$(x, majorityDF)$)
$uniqueMajorityExamples \leftarrow movedMajorityExamples$.groupby($index$)
  .map($x \rightarrow$randomChoice(x))
$majorityNeighbors \leftarrow examples.neighbors.tail$
  .filter($label \neq minorityLabel$)
return $majorityNeighbors$.map($x \rightarrow$withinRadius(x)).sum() + 1

function: withinRadius($example, neighbor, radius$)
$distance \leftarrow$ Array($example, neighbor$).transpose.map($x \rightarrow (x[0] - x[1]) * (x[0] -$ $x[1])$).sum()
if $distance \leq radius$ **then**
  return 1
else
  return 0
end if
return

Cluster SMOTE [122] is a straightforward method that utilizes clustering when generating synthetic examples. The minority class is clustered using k-Means and kNN is performed on each cluster. For each synthetic example to be created, a random cluster is selected and then a random example within that cluster is selected. A new synthetic example is then created on a random point between the selected examples and one of its nearest neighbors. Cluster SMOTE was one of the first solutions addressing a major SMOTE drawback – poor performance on multi-modal data, where artificial instances should not be introduced between modalities.

Algorithm 6 Cluster SMOTE

Input: DataFrame of the full dataset
Parameters: DF: DataFrame of all examples in the dataset, *minorityLabel*: label of the minority class, *totalExamplesToAdd*: number of synthetic examples to create, k: number of nearest neighbors to consider, *clusterK*: number of clusters to create
Output: Synthetic examples

function: ClusterSMOTE($DF, minorityLabel, totalExamplesToAdd, k, clusterK$)
$minorityDF \leftarrow DF$.filter($label = minorityLabel$)
$kMeans \leftarrow$ KMeans.fit($minorityDF$)
$minorityDF[clusterId] \leftarrow kMeans$.transform($minorityDF$)
$clusters \leftarrow minorityDF$.groupBy($clusterId$)
$clusterKNNs \leftarrow clusters$.map($x \rightarrow$KNN.fit($x$))
$clusterNearestNeighbors \leftarrow$ (0 to $clusterK$)
  .map($x \rightarrow clusterKNNs[x]$.transform($clusters[x]$))
$randomClusterIds \leftarrow$ (0 to $totalExamplesToAdd$)
  .map($x \rightarrow$randomInt(0, $clusterK$))
return $randomClusterIds$.map($x \leftarrow$generateExample($clusterNearestNeighbors[x]$))

function: generateExample($cluster$)
$example \leftarrow cluster[randomInt(0, cluster.\text{count}())]$
$randomNeighbor \leftarrow example.neighbors.tail[\text{randomInt}(0, k)]$
return Array($example, randomNeighbor$)
  .map($x \rightarrow x[0] + \text{randomDouble}() * (x[1] - x[0])$)

Gaussian SMOTE [123] works very similar to standard SMOTE but uses a Gaussian instead of a uniform distribution when picking a point between two minority class examples. Gaussian SMOTE aims at enriching the representation of the minority class by introducing more spread within the artificial instances. This helps to avoid

the patterns introduced with the original SMOTE algorithm where synthetic examples were only placed on the hyperplanes between existing examples.

Algorithm 7 Gaussian SMOTE

Input: DataFrame of the full dataset
Parameters: DF: DataFrame of all examples in the dataset, *minorityLabel*: label of the minority class, *totalExamplesToAdd*: number of synthetic examples to create, k: number of nearest neighbors to consider, *sigma*: standard deviation for Gaussian sampling
Output: Synthetic examples

function: GaussianSMOTE($DF, minorityLabel, totalExamplesToAdd, k, sigma$)
$minorityDF \leftarrow DF$.filter($label = minorityLabel$)
$knnModel \leftarrow$ KNN.fit($minorityDF, k$)
$nearestNeigbors \leftarrow knnModel$.transform($minorityDF$)
$randomIndexes \leftarrow$ (0 to $nearestNeigbors$.count())
.map($x \rightarrow$ randomInt(0, $totalExamplesToAdd$))
return $randomIndexes$.map($x \rightarrow$ createSyntheticExample($nearestNeigbors[x], k$))

function: createSyntheticExample($example, k$)
$example \leftarrow nearestNeighbors[0]$
$randomNeighbor \leftarrow example.neighbors.tail$(randomInt(0, k))
$gap \leftarrow$ nextGaussian(0, $sigma$)
return Array($example, randomNeighbor$).map($x \rightarrow x[0]$
$+ gap * (x[1] - x[0])$)

***k*-Means SMOTE** [124] starts by performing k-Means and keeps clusters that have more minority than majority examples. For each cluster, the density of the minority examples is calculated to create a sampling weight. Proportional to the weight of each cluster, SMOTE is used to create synthetic examples. This is a direct extension of Cluster SMOTE, where additional information regarding the minority class distribution in each cluster is used to control the SMOTE oversampling.

Algorithm 8 k-Means SMOTE

Input: DataFrame of the full dataset
Parameters: DF: DataFrame of all examples in the dataset, *minorityLabel*: label of the minority class, *totalExamplesToAdd*: number of synthetic examples to create, k: number of nearest neighbors to consider, *clusterK*: number of clusters to create
Output: Synthetic examples

function: kMeansSMOTE($DF, minorityLabel, totalExamplesToAdd, k, clusterK$)

$minorityDF \leftarrow DF$.filter($x \leftarrow x.label = minorityLabel$)
$majorityDF \leftarrow DF$.filter($x \leftarrow x.label \neq minorityLabel$)
$kMeans \leftarrow$ KMeans.fit(DF)
$DF[clusterId] \leftarrow kMeans$.transform($DF$)
$clusters \leftarrow predictions$.groupBy($DF[clusterId]$))
$clusters[imbalancedRatio] \leftarrow clusters$.map($x \rightarrow$getImbalancedRatio($x$))
$filteredClusters \leftarrow clusters$.filter($imbalancedRatio < irt$)
$filteredClusters[averageDistance] \leftarrow filteredClusters$
  .map($x \rightarrow$getAverageDistance(x))
$filteredClusters[densityFactor] \leftarrow filteredClusters$
  .map($x \rightarrow x$.count() / pow($x[averageDistance], de$))
$filteredClusters[sparsityFactor] \leftarrow filteredClusters$
  .map($x \rightarrow 1/x[densityFactor]$)
$sparsitySum \leftarrow filteredClusters$.map($x \rightarrow x[sparsityFactor]$).sum()
$filteredClusters[samplingWeight] \leftarrow filteredClusters$
  .map($x \rightarrow x[sparsityFactor]$ / $sparsitySum$)
$clusterKNNs \leftarrow filteredClusters$.map($x \rightarrow$KNN.fit($x$))
$clusterNearestNeighbors \leftarrow$ (0 to $clusterK$)
  .map($x \rightarrow clusterKNNs[x]$.transform($filteredClusters[x]$))
return $clusterNearestNeighbors$.map($x \rightarrow$generateExamples(x))

function: getImbalancedRatio($clusterDF$)
$minorityCount \leftarrow clusterDF$
  .filter($x \leftarrow x.label = minorityLabel$).count()
$majorityCount \leftarrow clusterDF$
  .filter($x \leftarrow x.label \neq minorityLabel$).count()
return $(majorityCount + 1)$ / $(minorityCount + 1)$

function: getAverageDistance($clusterDF$)
return $clusterDF$.map($x \leftarrow clusterDF$.map($y \leftarrow$getDistance(x, y)).sum()).sum()
  / ($clusterDF$.count() * $clusterDF$.count())

function: getDistance(x, y)
return Array(x, y).transpose.map($x \rightarrow (x[0] - x[1]) * (x[0] - x[1])$).sum()

function: generateExamples($cluster, totalExamplesToAdd$)
$examplesToAdd \leftarrow cluster[samplingWeight] * totalExamplesToAdd$
return (0 to $examplesToAdd$).map($x \rightarrow$ generateSingleExample(x))

function: generateSingleExample($neighbors, k$)
$example \leftarrow neighbors.neighbors[0]$
$randomNeighbor \leftarrow example.neighbors.tail$[randomInt($0, k$)]
return Array($example, randomNeighbor$)
  .map($x \rightarrow x[0]$ + randomDouble() $* (x[1] - x[0])$)

MWMOTE [125] has three main phases: identify the most difficult minority class examples to learn from, assign a weight to each example based on its importance, and create synthetic examples. In the first phase, 1-NN is performed on the dataset and minority class examples with at least one other minority neighbor is kept. MWMOTE follows the idea that difficult instances are the ones that pose highest challenge to a classifier and are most likely to introduce errors during training. Therefore, MWMOTE focuses on decreasing the difficulty of such instances by creating a uniform minority class regions around them.

Algorithm 9 MWMOTE

Input: DataFrame of the full dataset
Parameters: DF: DataFrame of all examples in the dataset, *minorityLabel*: label of the minority class, *totalExamplesToAdd*: number of synthetic examples to create, *k1, k2, k3*: number of nearest neighbors to consider,
Output: Synthetic examples

function: MWMOTE($DF, minorityLabel, totalExamplesToAdd, k1, k2, k3$)
$Smin \leftarrow DF$.filter($label = minorityLabel$)
$Smaj \leftarrow DF$.filter($label \neq minorityLabel$)
$knn1 \leftarrow KNN$.fit($DF, k1$)
$SminNN \leftarrow knn1$.transform($Smin$)
$Sminf \leftarrow SminNN$.filter(hasMinorityNeighbors() > 0)
$SmajNN \leftarrow$KNN.fit($Smaj, k2$)
$Nmaj \leftarrow SmajNN$.transform($Sminf$)
$Sbmaj \leftarrow Nmaj$.map($x \rightarrow x.neighbors$).distinct()
$SminNN \leftarrow$KNN.fit($Smin, k3$)
$Nmin \leftarrow SminNN$.transform($Sbmaj$)
$Simin \leftarrow Nmin$.map($x \rightarrow x.neighbors$).distinct()
$Iw \leftarrow Sbmaj$.map($y \rightarrow y$.map($x \rightarrow$Cf(y, x) $*$ Df(y, x))
$Sw \leftarrow Simin$.map($x \rightarrow Sbmaj$.map($y \rightarrow$Iw(y, x)).sum())
$Sp \leftarrow Sw$.map($x \rightarrow x$ / Sw.sum())
$kMeans \leftarrow$ KMeans.fit($Smin$)
$Smin[clusterId] \leftarrow kMeans$.transform($Smin$)
return (0 to $examplesToAdd$).map(_ $\rightarrow$ generateExample($Smin$))

function: generateExample($Smin$)
$example \leftarrow chooseByProbability(Smin)$
$clusterExamples \leftarrow Smin.filter(clusterId = example.clusterId)$
$neighbor \leftarrow clusterExamples[randomInt(0, clusterExamples.count()))]$
return Array($example, neighbors$)
.transpose.map($x \rightarrow x +$ randomDouble() $*$ ($y - x$))

function: hasMinorityNeighbors($example$)
$label \leftarrow example.neighbors[0].label$
return $example.neighbors$
.map($x \rightarrow$if $x.label = label$: 1; else: 0).sum()

function: Df($y, x, Simin$)
$cf \leftarrow$ Cf(y, x)
$denominator \leftarrow Simin$.map($q \rightarrow$ Cf(y, q)).sum()
return $distance$ / $denominator$

function: Cf(y, x)
$featureLength \leftarrow y$.length()
$distance \leftarrow$ Array(y, x).transpose.map($x \rightarrow (x[0] - x[1]) * (x[0] - x[1])$)
/ $featureLength$
if $1/distance \leq CfThreshold$ **then**
$cutoff \leftarrow 1$ / $difference$
else
$cutoff \leftarrow CfThreshold$
end if
return ($cutoff$ / $CfThrehold$) $* CMAX$

NRAS [126] works be removing noisy minority examples before creating SMOTE style synthetic examples. Using linear regression, the probability that each example in the dataset belongs to the minority class is determined. All minority class examples that have a membership probability less a threshold parameter value are removed from the dataset and SMOTE style oversampling is performed using the remaining minority class examples. NRAS was designed to handle highly noisy datasets that may be subject to various types of noise (both affecting class labels and features). A potential drawback may lie in forcefully trying to find noise even when it is not present in the training data.

Algorithm 10 NRAS

Input: DataFrame of the full dataset
Parameters: DF: DataFrame of all examples in the dataset, *minorityLabel*: label of the minority class, *totalExamplesToAdd*: number of synthetic examples to create, k: neighbor count for kNN, *threshold*: minimum number of minority neighbors needed to keep an example
Output: Synthetic examples

function: NRAS($DF, minorityLabel, totalExamplesToAdd, k, threshold$)
$minorityDF \leftarrow DF$.filter($x \rightarrow x.label = minorityLabel$)
$minorityDF \leftarrow DF$.filter($x \rightarrow x.label \neq minorityLabel$)
$DF[propensityScore] \leftarrow$ LinearRegression(DF)
$knn \leftarrow$KNN.fit(DF, k)
$nearestNeighbors \leftarrow knn$.transform($minorityDF$)
$samplingRatio \leftarrow nearestNeighbors$.count() / $totalExamplesToAdd$
$keepMinority \leftarrow nearestNeighbors$.filter($neighbors.tail$
.map($x \rightarrow if\ x.label = minorityLabel$: 1; $else$: 0).sum() $\geq threshold$))
return $keepMinority$.map($x \rightarrow$generateExamples($x, samplingRatio, k$))

function: generateExamples($example, samplingRatio, k$)
return (0 to $samplingRatio$)
.map(_ $\rightarrow$ generateSingleExample($example, k$))

function: generateSingleExample($example, k$)
$example \leftarrow example.neighbors$[0]
$randomNeighbor \leftarrow example.neighbors.tail$[randomInt($0, k$)]
return Array($example, randomNeighbor$)
.map($x \rightarrow x[0]$ + randomDouble() $*$ ($x[1] - x[0]$))

Random Oversampling duplicates existing examples instead of creating synthetic

ones. For each minority class, random examples are duplicated until the desired level of class balance is achieved.

Algorithm 11 Random Oversample

Input: DataFrame of the full dataset STATE **Parameters:** DF: DataFrame of all examples in the dataset $minorityLabel$: label of the minority class, $examplesToAdd$: number of synthetic examples to create
Output: Synthetic examples

function: RandomOversample($DF, minorityLabel, examplesToAdd$)
$minorityDF \leftarrow DF$.filter($x \rightarrow x.label = minorityLabel$)
$samplingRate \leftarrow examplesToAdd$ / $minorityDF$.count()
return $minorityDF$.sample($withReplacement = True, samplingRate$)

RBO [102] begins by picking a random minority example for each synthetic example to be created. Next, a randomized translation vector is created and applied to a copy of the picked example. The *phi* function is run on both examples and if the result corresponding to the translated example is smaller, the translation vector is applied to the original example. This process is repeated until the maximum number of iterations have completed or a random stopping probability condition has been met. Finally, the translated point is added as a new synthetic example. RBO offers a powerful alternative to SMOTE-based oversampling that is free of nearest neighbor search. Additionally, RBO automatically identifies difficult regions within each minority class and adjusts its local oversampling ratio accordingly.

Safe Level SMOTE [21] performs kNN and the nearest neighbors for each minority example is returned. For each of these examples (p), a random neighbor (n) is selected and the number of neighboring minority examples are counted. A safe level ratio is created by dividing the p minority nearest neighbor count by the n minority nearest neighbor count. Based on this ratio, different explicit rules are applied to generate new synthetic examples. Safe Level SMOTE aims at avoiding the enhancement of noisy or difficult instances. However, very often those difficult instances are the core

Algorithm 12 RBO

Input: DataFrame of the full dataset
Parameters: DF: DataFrame of all examples in the dataset, *minorityLabel*: label of the minority class, *examplesToAdd*: number of synthetic examples to create, *gamma*: spread of radial basis function, *stepSize*: step size for point translation, *iterations*: number of iterations per synthetic sample, *probability*: probability of stopping early
Output: Synthetic examples

function: RBO($DF, minorityLabel, examplesToAdd, gamma, stepSize, iterations, probability$)
$minorityDF \leftarrow DF$.filter($x \rightarrow x.label = minorityLabel$)
$majorityDF \leftarrow DF$.filter($x \rightarrow x.label \neq minorityLabel$)
return (0 to $examplesToAdd$)
  .map(_ →createdExample($minorityDF, majorityDF, gamma$,
    $stepSize, iterations, probability$))

function: createExample($majorityDF, minorityDF, gamma$,
  $stepSize, iterations, probability$)
$point \leftarrow minortyDF$.getRandomExample()
$pointPhi \leftarrow calculatePhi(point, majorityDF, minorityDF, gamma)$
$randomStopIndex \leftarrow$getRandomStopIndex($iterations, probability$)
for $i = 0$ to $randomStopIndex$ **do**
  $randomDirection \leftarrow features.indicies$.map($x \rightarrow$randomChoice($-1, 1$))
  $randomVector \leftarrow features.indicies$.map($x \rightarrow$randomDouble())
  $translated \leftarrow point + randomDirection * randomVector * stepSize$
  if $|calculatePhi(translated, majorityDF, minorityDF, gamma)| <$
    $|calculatePhi(point, majorityDF, minorityDF, gamma)|$ **then**
    $point \leftarrow translated$
  end if
end for
return *point*

function: getRandomStopIndex($iterations, probability$)
if $probability = 1.0$ **then**
  return *iterations*
else
  return $iterations * probability$ + randomGaussian() $*$ $iterations *$
    $probability$)
end if

function: calculatePhi($x, K, k, gamma$)
$majorityValue \leftarrow K$.map($Ki \rightarrow$
  exp(-power(pointDifference(Ki, x)/$gamma$, 2))).collect().sum()
$minorityValue \leftarrow k$.map($ki \rightarrow$
  exp(-power(pointDifference(ki, x)/$gamma$, 2))).collect().sum()
return $majorityValue - minorityValue$

function: pointDifference(x, y)
return (x, y).transpose.map($x \rightarrow ||x[0] - x[1]||$).sum()

concepts of the minority class and therefore avoiding their oversampling may lead to a drop of performance on test sets.

SMOTE [127] performs kNN on the minority class and then randomly chooses existing examples as the basis for new synthetic examples. Each synthetic example is created on a random point between the previously chosen example and one of its random nearest neighbors.

SMOTE-D [128] is designed to be a deterministic version of the standard SMOTE algorithm. kNN is performed on the minority examples and the standard deviation between their nearest neighbors is calculated. Each minority class examples is oversampled proportional to its standard deviation divided by the sum of all standard deviations. For each of these minority examples, synthetic examples are evenly created between itself and each nearest neighbor such that neighbors further away are used to generate the most examples.

4.3 Software Package

The proposed library (https://github.com/fsleeman/spark-class-balancing) was developed to both compare the original oversampling methods and provide the first Spark implementation written in Scala. These oversampling algorithms include: ADASYN, ANS, Borderline SMOTE, CCR, Cluster SMOTE, Gaussian SMOTE, k-Means SMOTE, MWMOTE, NRAS, Random Oversampling, RBO, Safe Level SMOTE, SMOTE, SMOTE-D. Since each of these algorithms were designed for binary class problems, this Spark implementation has added multi-class support. Each minority class is processed separately but for algorithms that take in account the majority class, a one-vs-all approach was used. In this case, all examples not part of the minority class was treated as if they were part of a single majority class. The sampling methods were developed in the style of the Spark Machine Learning Library

Algorithm 13 Safe Level SMOTE

Input: DataFrame of the full dataset
Parameters: DF: DataFrame of all examples in the dataset, *minorityLabel*: label of the minority class, *examplesToAdd*: number of synthetic examples to create, k: neighbor count for kNN
Output: Synthetic examples

function: SafeLevelSMOTE($DF, minorityLabel, examplesToAdd, k$)
$minorityDF \leftarrow DF$.filter($x \rightarrow x.label = minorityLabel$)
$knn \leftarrow$ KNN.fit(DF, k)
$minorityNearestNeighbors \leftarrow knn$.transform($minorityDF$)
$randomIndexes \leftarrow$ (0 to $examplesToAdd$)
  .map($x \rightarrow$ randomInt($0, minorityNearestNeighbors$.count()))
return $randomIndexes$.map($x \rightarrow$generateExample($minorityNearestNeighbors[x]$))

function: generateExample($nearestNeighbors$)
$example \leftarrow nearestNeighbors.head$
$randomNeighbor \leftarrow nearestNeighbors.tail$(randomInt(0, k))
$safeLevel_{\mathrm{p}} \leftarrow example.neighbors$
  .filter($label = minorityLabel$).count()
$safeLevel_{\mathrm{n}} \leftarrow randomNeighbor.neighbors$
  .filter($x \rightarrow label = minorityLabel$).count()
$safeLevelRatio \leftarrow 0$
if $safeLevel_{\mathrm{n}} \neq 0$ **then**
  $safeLevelRatio = safeLevel_{\mathrm{p}}$ / $safeLevel_{\mathrm{n}}$
else
  $safeLevelRatio \leftarrow \infty$
end if
if $safeLevelRatio = \infty$ and $saveLevel_{\mathrm{p}} = 0$ **then**
  return $\varnothing$
else
  if $safeLevelRatio = \infty$ and $safeLevel_{\mathrm{p}} \neq 0$ **then**
    $gap \leftarrow 0$
  else if $safeLevelRatio = 1$ **then**
    $gap \leftarrow$randomDouble$(0, 1)$
  else if $safeLevelRatio > 1$ **then**
    $gap \leftarrow$randomDouble$(0, 1$ / $safeLevelRatio)$
  else if $safeLeveRatio < 1$ **then**
    $gap \leftarrow$randomDouble$(1 - safeLevelRatio, 1))$
  end if
end if
return $features$.map($x \rightarrow x[0] + gap * (x[1] - x[0])$)

Algorithm 14 SMOTE

Input: DataFrame of the full dataset
Parameters: DF: DataFrame of all examples in the dataset, *minorityLabel*: label of the minority class, *examplesToAdd*: number of synthetic examples to create, k: neighbor count for kNN
Output: Synthetic examples

function: SMOTE($DF, minorityDF, examplesToAdd, k$)
$knnModel \leftarrow$ KNN.fit($minorityDF, k$)
$nearestNeigbors \leftarrow knnModel$.transform($minorityDF$)
$randomIndexes \leftarrow$ (0 to $examplesToAdd$).map($x \rightarrow$ randomInt$(0, k)$)
return $randomIndexes$.map($x \rightarrow$ createSmoteStyleExample($nearestNeighbors[x], k$))

function: createSmoteStyleExample($nearestNeighbors, k$)
$example \leftarrow nearestNeighbors.head$
$randomNeighbor \leftarrow nearestNeighbors.tail$(randomInt(0, k))
$gap \leftarrow$ randomDouble()
$features \leftarrow$Array($example.features, randomNeighbor.features$)
return $features$.map($x \rightarrow x[0] + gap * (x[1] - x[0])$)

(MLlib) methods using the *fit* and *transform* pattern.

Each of these oversampling methods were originally presented as serial algorithms which required some modifications when implemented for Spark. However, we have kept these implementation as close to the original as possible and have not made major improvements on performance or accuracy. In Section 4.5, we discuss some ways the running time of these algorithms could be improved and a discussion on the types of components that are most appropriate for the Spark platform.

4.3.1 Dependencies

The spark-class-balancing library was developed using Spark 3.0.1 with Scala 2.12.1 and currently has following dependencies: spark-core, spark-sql, spark-mllib, breeze-natives, breeze, spark-knn (from the fsleeman fork). The original spark-knn implementation was based on Spark 2 and so the forked version was updated to be compatible with Spark 3.

Algorithm 15 SMOTE-D

Input: DataFrame of the full dataset
Parameters: DF: DataFrame of all examples in the dataset, *minorityLabel*: label of the minority class, *examplesToAdd*: number of synthetic examples to create, k: neighbor count for kNN
Output: Synthetic examples

function: getSmoteSamples($DF, minorityLabel, examplesToAdd, k$)
$knn \leftarrow$ KNN.fit($minorityDF, k$)
$nearestNeighbors \leftarrow$ knn.transform($minorityDF$)
$nearestNeighbors[std] \leftarrow minorityNeighbors$
  .map($x \rightarrow$ getDistanceSTD(x))
$nearestNeighbors[stdWeights] \leftarrow neighborDistanceSTDs$
  .map($x \rightarrow x[std]/neighborNeighbors[std].sum$)
$nearestNeighbors[examplesToAdd] \leftarrow$
  $nearestNeighbors[stdWeights] * examplesToAdd$
return $nearestNeighbors$.map($x \rightarrow$ sampleCurrentExample(x))

function: sampleCurrentExample($example$)
$neighbors \leftarrow example.neighbors$
$neighbors[distances] \leftarrow neighbors$
  .map($x \rightarrow$distance($example, x$))
$neighbors[examplesToAdd] \leftarrow neighbors[distances]$
  .map($x \rightarrow (x * example[examplesToAdd])$ / $distances.sum$)
return $neighbors.map(x \rightarrow$createExamples($example, x$))

function: createExamples($example, neighbor$)
$examplesToAdd \leftarrow neighbor[examplesToAdd]$
$syntheticExamples \leftarrow$ (0 to $neighbor[examplesToAdd]$)
  .map($x \rightarrow example + (neighbor - examples)$
  $* (x + 1)/neighbor[examplesToAdd]$)
return $syntheticExamples$

Figure 3: Shown with black borders, synthetic examples from each sampling method are combined with the original data. The upper left plot shows the original data without oversampling.

4.3.2 spark-knn

Many of these sampling algorithms, such as SMOTE variants, include a k-Nearest Neighbors search. Instead of creating that method from scratch, an efficient implementation from spark-knn (https://github.com/saurfang/spark-knn) [129] was used. However, the spark-knn implementation did not support nearest neighbor search by distance radius which was required by ANS and CCR. To address this limitation, spark-knn was forked (https://github.com/fsleeman/spark-knn) and the distance search feature was added.

4.3.3 Using the spark-class-balancing Library

The spark-class-balancing library is designed to build a fat jar using the *sbt* tool, with the following commands:

```
sbt compile
sbt assembly
```

Since the spark-knn (fsleeman fork) library is a dependency of spark-class-balancing, it must also be built as far jar using a similar method and the *build.sbt* file must be updated to include that generated file. The resulting spark-class-balancing jar file can then be used for running a Spark job, both in the local or cluster mode.

4.3.4 Invoking Sampling Methods

Invoking one of the oversampling methods is straightforward as shown below in this example using standard SMOTE.

```
val method = new SMOTE
val model = method.fit(trainData).setK(5)
val sampledData = model.transform(trainData)
```

Using the *fit/transform* pattern, a new SMOTE estimator is instantiated and then a model is created with the *fit* function. The resulting model is set with prediction parameters, in this case setting the k value to 5, and finally the *trainData* DataFrame is transformed to produce oversampled data. The same process is done for every other oversampling methods, although available parameters may differ.

4.3.5 Limitations and Future Extensions

These algorithms were implemented in the spirit of their original design to better assess how they translate to the Spark platform in terms of performance and accuracy. While some of these implementations have limitations in efficiency, they can now be updated based on the algorithm design guidelines presented in Section 4.5. While this initial version of spark-class-balancing included 14 oversampling methods, it can be extended to include other types of sampling methods including undersampling, hybrid methods and ensembles.

The kNN algorithm from the spark-knn package can run into errors depending on the settings used and properties of the data. The spark-knn implementation uses a hybrid spill tree which switches between spill and metric trees based on data metrics at each node split. If data is not well balanced between the two child notes, the tree generation may fail. However, this can be avoided by forcing the use of metric trees at all nodes. Other issues can occur with small datasets or few examples for a given class, but in that case spark-class-balancing includes a fallback to use the brute force kNN mode.

In the future, this library can be built as a Spark package with a better integration with the custom fork of the spark-knn package. The spark-class-balancing library can also be updated to the newest versions of Spark as they come out. These Spark updates can include new built-in machine learning tools or general features which can

be utilized in future sampling methods.

4.3.6 Adding New Algorithms

A Scala class named *samplingTemplate* has been added as blank slate for developing new algorithms. After this class is copied to a new file, the placeholder names can be replaced and the core code can be added, using the other algorithms as example code.

4.4 Experimental Study

We have designed the experimental study to answer the following research questions:

RQ1: Which of the proposed big data oversampling approaches adapted to Spark architecture offers the best predictive performance?

RQ2: What is the computational cost of the proposed big data oversampling approaches?

RQ3: Which of the proposed big data oversampling approaches offers the best trade-off between predictive accuracy and computational complexity?

RQ4: How does the examined algorithms handle binary and multi-class problems, as well as different types of features?

RQ5: How well does the examined oversampling methods scale with the size of the data and which algorithms become prohibitively expensive to be used efficiently?

4.4.1 Setup

Hardware. All experiments were performed using 32 threads on a large shared machine with Intel Xeon E7-8894 cores.

Datasets. We have chosen twenty six datasets that include two-classes with binary features, two-classes with continuous features and multi-class with continuous features. These datasets provide a wide range of features and class imbalance ratios which gives some insight on how different oversampling-classifier combinations perform on these types of data. Initial tests showed that using the full datasets was not feasible with the time required for each test, so we decided to select a sub-section of 100,000 examples from each dataset which is still significantly more than used in the original algorithm publications. In addition to making the running time achievable, it also allows for the direct time comparisons between each dataset. Table 1 shows high level properties of these datasets.

Classifiers. We have used three popular classifiers that are available in the Spark Machine Learning Library (MLlib) - Random Forest, Support Vector Machines (SVM) and Naive Bayes. Since each classifier uses a different base algorithm, our experiments give some insight on how sampling methods may affect classifiers from different families.

Parameters of classifier. Table 2 shows the hyperparameters used for each classifier. These values were chosen by a manual search that provided good accuracy compared to running time on several datasets.

Oversampling methods. We have implemented the fourteen oversampling detailed in Section 4.2 and were used with all combinations of classifiers and datasets. However, the sampling algorithms ANS, MWMOTE and RBO were significantly slower and full experiments were not feasible. Section 4.5 provides a discussion on properties of these algorithms which prevented scalability.

Parameters of oversampling methods. Table 3 shows the list of hyperparameters used for each oversampling algorithm. Since the following experiments included combination of datasets, classifier and oversampling methods, an exhaustive hyperpa-

Table 1: High level properties of the twenty six datasets included in the following experiments.

Dataset Descriptions				
Dataset	**Class Count**	**Feature Count**	**Feature Type**	**Max Imbalanced Ratio**
Bitcoin	2	8	Continuous	69.42
Cover Type	7	54	Continuous	103.08
Fuzzing	2	115	Continuous	4.19
HIGGS: Imbalanced Ratio 16:1	2	28	Continuous	16.00
HIGGS: Imbalanced Ratio 4:1	2	28	Continuous	4.00
HIGGS: Imbalanced Ratio 8:1	2	28	Continuous	8.00
IoT	11	115	Continuous	4.82
MIRAI	2	116	Continuous	5.28
Poker: 0 vs 2	2	85	Discrete	10.53
Poker: 0 vs 3	2	85	Discrete	23.73
Poker: 0 vs 4	2	85	Discrete	129.04
Poker: 0 vs 5	2	85	Discrete	251.53
Poker: 0 vs 6	2	85	Discrete	352.36
Poker: 1 vs 2	2	85	Discrete	8.87
Poker: 1 vs 3	2	85	Discrete	20.00
Poker: 1 vs 4	2	85	Discrete	108.77
Poker: 1 vs 5	2	85	Discrete	211.77
Poker: 1 vs 6	2	85	Discrete	296.62
SEER	10	11	Continuous	5.69
SSL Renegotiation	2	115	Continuous	22.83
SUSY: Imbalanced Ratio 16:1	2	18	Continuous	16.00
SUSY: Imbalanced Ratio 4:1	2	18	Continuous	4.00
SUSY: Imbalanced Ratio 8:1	2	18	Continuous	8.00
SYN DOS	2	115	Continuous	392.70
Traffic	7	26	Continuous	7.60
Video Injection	2	115	Continuous	23.12

Table 2: Classifier hyperparameters used in the sampling experiments.

Algorithm	**Hyperparameters**
Support Vector Machine	*number of iterations* = 100 *regularization parameter (C)* = 10.0
Random Forest	*number of trees* = 100 *maximum depth* = 20
Naive Bayes	None

rameter search was not practical and so we have chosen a single set of hyperparameters for each oversampling method. These values were chosen from the original papers and a limited manual search was performed if no values were presented. As the value of 5

is often used for k, including the original SMOTE algorithm, we have chosen to use that value as well. The hyperparameters for RBO were set in such a way that only a single iteration per synthetic example to be created which deviates from the experiments in the original paper. These values were chosen in an attempt to compensate for the slow running time which is further discussed in Section 4.5.

Table 3: Hyperparameters used for each sampling method.

Algorithm	**Hyperparameters**
ADASYN	$k = 5$
ANS	$k = 5$ *C max ratio* $= 0.25$ *distance neighbor limit* $= 100$
Borderline SMOTE	$k = 5$
CCR	$k = 5$ *energy* $= 1.0$ *distance neighbor limit* $= 100$
Cluster SMOTE	$k = 5$ *cluster* $k = 5$
Gaussian SMOTE	$k = 5$ *sigma* $= 0.5$
k-Means SMOTE	$k = 5$ *cluster* $k = 5$ *imbalancedThreshold* $= 10.0$
MWMOTE	*k1* $= 5$ *k2* $= 5$ *k3* $= 5$ *cluster* $k = 10$ *C max* $= 3.0$ $C_f(th) = 50.0$
NRAS	$k = 5$ *threshold* $= 3$
Random Oversampling	None
RBO	*gamma* $= 1.0$ *iterations* $= 1$ *step size* $= 0.01$ *stopping probability* $= 1.0$
Safe Level SMOTE	$k = 5$ *sampling correction rate* $= 0.05$
SMOTE	$k = 5$
SMOTE-D	$k = 5$

$$AvAcc = \sum_{i=1}^{C} \frac{tp_i + tn_i}{tp_i + tn_i + fp_i + fn_i}$$

$$MAvG = \sqrt[C]{\prod_{i=0}^{C} recall_i}$$

$$AvF_\beta = \frac{1}{C} \sum_{i=1}^{C} \frac{(1+\beta^2) \cdot precision_i \cdot recall_i}{\beta^2 \cdot precision_i + recall_i}$$

$$CBA = \sum_{i=1}^{C} \frac{mat_{i,i}}{max(\sum_{j=1}^{C} mat_{i,j}, \sum_{j=1}^{C} mat_{j,i})}$$

Figure 4: Formulas for the metrics used to evaluate the classifier and sampling combinations.

Adaptation to multi-class problems. Each minority class was oversampled to the size of the majority class individually, resulting in a fully balanced dataset.

Training and testing. We have used a stratified 5-fold cross validation in order to maintain the original class distribution among the folds.

Evaluation metrics. In order to properly evaluate the performance of classifiers on imbalanced domains, skew-insensitive metrics are required. As multi-class imbalanced problems were not popular until recently among researchers, there is a lack of uniform approach to which measures should be considered as a standard. In order to gain an in-depth insight into the performance of the analyzed classifiers, Figure 4 shows four popular metrics for multi-class imbalanced data used to evaluate the presented oversampling algorithms.

Experiments. The experiments performed include all of the combinations of datasets, classifier and sampling method with 5-fold cross validation. The ANS, MWMOTE and RBO algorithms were not included in the full experiments because of the excessive

Table 4: Classifier hyperparameters used in the sampling experiments.

Algorithm	**Hyperparameters**
Support Vector Machine	*number of iterations* = 100 *regularization parameter (C)* = 10.0
Random Forest	*number of trees* = 100 *maximum depth* = 20
Naive Bayes	None

run time and this limitation is further discussed in Section 4.5.

Table 5: Comparison of 11 oversampling methods using four different metrics with Random Forest as the base classifier.

		ADASYN	Borderline SMOTE	CCR	Cluster SMOTE	Gaussian SMOTE	k-Means SMOTE	NRAS	Random Oversample	Safe Level SMOTE	SMOTE	SMOTE-D
Bitcoin	AvAcc	98.08	98.56	**98.60**	96.02	95.32	97.88	98.59	96.35	97.42	96.32	96.74
	MAvG	48.69	68.14	**75.80**	40.51	38.52	43.86	71.02	42.07	38.28	41.16	40.29
	AvFb	58.03	55.22	53.53	65.26	65.14	57.60	53.09	**65.64**	57.75	64.81	62.32
	CBA	57.28	53.89	52.36	56.66	55.47	57.15	52.00	**57.49**	56.78	57.11	57.04
Cover Type	AvAcc	95.00	**95.21**	93.54	94.86	94.34	94.69	94.87	94.78	94.40	94.74	94.76
	MAvG	67.87	**81.76**	35.15	67.62	61.40	61.76	67.84	66.39	61.40	65.78	65.94
	AvFb	**81.39**	77.21	41.31	81.03	79.06	70.40	78.17	80.69	77.78	80.69	80.94
	CBA	68.84	**73.28**	36.19	68.82	63.29	64.10	68.74	67.85	63.21	67.33	67.49
Fuzzing	AvAcc	99.99	99.98	99.87	99.98	99.99	99.99	99.87	99.99	99.94	**100.00**	99.99
	MAvG	99.97	99.96	99.68	99.95	99.99	99.99	99.67	99.99	99.83	**99.99**	99.99
	AvFb	99.99	99.97	99.86	99.98	**99.99**	99.98	99.87	99.98	99.93	99.99	99.99
	CBA	99.96	99.95	99.61	99.94	99.98	99.97	99.59	99.97	99.79	**99.99**	99.98
HIGGS: 4:1	AvAcc	77.53	74.38	80.58	77.64	79.68	77.50	78.63	**82.21**	72.02	76.35	74.83
	MAvG	62.68	60.14	**76.93**	62.86	65.70	60.82	63.54	70.90	57.42	61.61	59.88
	AvFb	67.81	67.66	50.79	67.87	**68.88**	63.63	66.65	65.95	65.30	67.85	66.80
	CBA	64.21	59.16	43.13	64.41	**68.02**	61.35	66.37	61.39	56.30	62.19	60.03
HIGGS: 8:1	AvAcc	84.91	86.69	88.99	82.64	85.05	83.29	87.81	**89.06**	76.46	80.97	79.02
	MAvG	56.03	59.89	**81.16**	53.63	57.73	48.99	60.32	68.65	46.00	52.01	49.81
	AvFb	64.19	64.31	49.70	65.04	**66.39**	59.00	58.33	61.18	60.64	64.90	63.72
	CBA	62.40	**63.47**	45.27	59.12	63.05	57.55	54.92	57.29	51.51	56.98	54.61
HIGGS: 16:1	AvAcc	91.56	94.01	**94.13**	86.93	89.21	88.73	93.92	93.82	85.42	85.08	83.48
	MAvG	49.53	**66.11**	33.56	43.38	47.89	38.45	47.54	63.71	36.99	41.54	39.66
	AvFb	59.66	55.34	49.51	61.91	**63.60**	56.39	50.43	57.48	57.25	61.46	60.35
	CBA	**59.17**	52.58	47.17	54.76	58.15	54.28	48.06	54.84	51.64	52.83	51.16
IoT	AvAcc	99.68	99.78	99.77	**99.85**	99.74	99.83	99.71	99.64	99.78	99.67	99.69
	MAvG	98.86	99.14	99.01	**99.42**	98.98	99.33	98.93	98.63	99.13	98.78	98.93
	AvFb	98.66	99.15	98.95	**99.40**	98.96	99.29	98.82	98.57	99.15	98.68	98.74
	CBA	97.61	98.37	98.11	**98.86**	98.03	98.69	97.84	97.34	98.36	97.57	97.74
MIRAI	AvAcc	**99.91**	99.83	99.62	99.87	99.87	99.81	99.84	99.90	99.87	99.86	99.85
	MAvG	99.72	99.60	98.90	99.74	99.68	99.67	99.60	**99.79**	99.60	99.71	99.70
	AvFb	**99.89**	99.75	99.55	99.76	99.81	99.61	99.77	99.82	99.84	99.76	99.72
	CBA	99.67	99.55	98.72	99.70	99.64	99.55	99.55	**99.78**	99.53	99.70	99.64
Poker: 0 vs 2	AvAcc	97.41	92.60	91.50	97.15	91.32	94.33	95.62	**99.87**	99.37	97.56	96.92
	MAvG	98.61	96.18	95.65	96.38	0.00	84.69	97.67	**99.93**	99.66	98.69	98.36
	AvFb	87.00	58.01	50.38	86.33	49.07	72.98	76.89	**99.37**	97.00	87.77	84.24
	CBA	83.71	53.59	46.78	83.45	45.66	69.29	72.45	**99.16**	96.05	84.64	80.60
Poker: 0 vs 3	AvAcc	96.35	96.10	96.01	96.76	95.96	95.71	97.04	**99.27**	98.96	96.06	96.14
	MAvG	98.15	98.03	78.39	93.61	0.00	60.11	98.35	**99.62**	99.46	98.01	98.05
	AvFb	55.59	51.72	50.37	61.76	49.58	51.85	65.38	**92.37**	89.04	51.16	52.42
	CBA	53.08	49.76	48.63	58.71	47.98	50.06	61.98	**90.53**	86.61	49.30	50.37
Poker: 0 vs 4	AvAcc	99.23	99.23	99.23	99.03	**99.23**	99.22	96.65	95.86	98.90	99.23	99.23
	MAvG	0.00	0.00	0.00	**63.96**	0.00	5.14	9.64	32.87	41.41	34.01	0.00
	AvFb	49.92	49.92	49.92	56.21	49.92	50.00	52.91	**64.21**	51.10	50.16	49.92
	CBA	49.62	49.62	49.62	**54.08**	49.62	49.68	49.55	53.49	50.34	49.81	49.62
Poker: 0 vs 5	AvAcc	99.60	99.60	99.87	99.60	99.60	99.57	99.76	**99.96**	99.93	99.56	99.61
	MAvG	39.92	0.00	99.94	90.94	0.00	42.75	99.54	**99.98**	99.97	23.00	39.92
	AvFb	50.27	49.96	86.18	57.08	49.96	51.92	71.04	**96.34**	92.64	50.24	50.28
	CBA	50.06	49.80	84.14	55.84	49.80	51.45	69.45	**95.56**	91.26	50.05	50.06
Poker: 0 vs 6	AvAcc	99.72	99.73	99.72	99.72	99.72	99.72	**99.91**	99.78	99.74	99.72	99.72
	MAvG	19.97	99.87	0.00	39.95	0.00	39.95	79.98	**99.89**	98.13	39.94	19.97
	AvFb	50.19	53.44	49.97	51.06	49.97	50.85	**85.05**	63.55	55.33	50.41	50.19
	CBA	50.03	52.69	49.86	50.74	49.86	50.57	**83.97**	61.39	54.28	50.21	50.04
Poker: 1 vs 2	AvAcc	89.88	89.87	89.87	89.89	89.87	87.60	89.86	**91.00**	89.92	89.87	89.88
	MAvG	71.57	0.00	0.00	80.00	0.00	29.94	55.25	87.05	**92.92**	56.88	89.25
	AvFb	48.95	48.90	48.90	49.46	48.90	50.48	48.93	**57.27**	49.25	48.92	48.94
	CBA	44.98	44.94	44.94	45.44	44.94	47.47	44.97	**52.52**	45.24	44.95	44.97
Poker: 1 vs 3	AvAcc	95.26	95.24	95.24	95.20	95.24	94.84	95.26	**96.10**	95.66	95.24	95.24
	MAvG	78.08	19.52	0.00	83.61	0.00	32.38	94.99	96.14	**97.80**	19.52	19.52
	AvFb	49.74	49.52	49.51	50.22	49.51	50.02	49.81	**60.76**	55.00	49.52	49.53
	CBA	47.82	47.63	47.62	48.25	47.62	48.23	47.87	**57.44**	52.29	47.63	47.64
Poker: 1 vs 4	AvAcc	99.09	99.09	99.09	98.77	**99.09**	99.05	99.09	96.24	99.06	99.08	99.09
	MAvG	0.00	0.00	0.00	52.58	0.00	19.46	0.00	38.34	**54.83**	28.81	28.81
	AvFb	49.91	49.91	49.91	58.61	49.91	50.15	49.91	**67.69**	51.30	50.11	50.04
	CBA	49.54	49.54	49.54	**57.32**	49.54	49.77	49.54	55.72	50.71	49.71	49.66
Poker: 1 vs 5	AvAcc	99.53	99.53	**99.94**	99.59	99.53	99.53	99.56	99.94	99.53	99.53	99.53
	MAvG	0.00	0.00	**99.97**	96.13	0.00	0.00	59.87	99.97	19.95	0.00	0.00
	AvFb	49.95	49.95	**94.78**	56.68	49.95	49.95	53.85	94.42	50.09	49.95	49.95
	CBA	49.77	49.77	**93.69**	55.75	49.77	49.76	53.08	93.27	49.87	49.77	49.77
Poker: 1 vs 6	AvAcc	99.66	99.66	99.66	99.66	99.66	99.65	**99.67**	99.66	99.66	99.66	99.66
	MAvG	0.00	0.00	0.00	0.00	0.00	0.00	**74.02**	39.93	0.00	0.00	0.00
	AvFb	49.97	49.97	49.97	49.97	49.97	49.96	**51.07**	50.52	49.97	49.97	49.97
	CBA	49.83	49.83	49.83	49.83	49.83	49.83	**50.73**	50.28	49.83	49.83	49.83
SEER	AvAcc	93.71	93.76	92.18	93.77	93.73	93.88	**94.30**	93.94	93.94	93.74	93.99
	MAvG	55.73	56.50	0.00	56.43	55.98	57.58	**60.83**	57.12	57.58	55.90	57.30
	AvFb	61.30	60.55	37.32	61.09	61.66	58.27	61.35	**62.29**	62.08	61.48	60.59
	CBA	57.09	57.51	30.72	56.62	56.82	53.47	57.51	58.13	**58.65**	56.94	55.64
SSL Renegotiation	AvAcc	100.00	100.00	99.99	100.00	100.00	99.99	99.98	100.00	99.95	100.00	**100.00**
	MAvG	99.96	99.96	99.94	99.98	99.96	99.97	99.73	**99.98**	99.43	99.98	99.96
	AvFb	99.98	99.98	99.98	99.98	99.98	99.97	99.93	99.98	99.86	99.98	**99.99**
	CBA	99.95	99.95	99.93	99.96	99.95	99.94	99.72	99.96	99.41	99.96	**99.96**
SUSY: 4:1	AvAcc	85.12	84.24	84.87	84.31	86.24	84.39	86.75	**87.01**	83.45	83.96	83.25
	MAvG	75.22	73.68	**85.96**	73.80	77.53	74.08	79.03	80.01	72.46	73.24	72.17
	AvFb	78.09	78.10	64.40	77.78	**78.19**	76.24	77.68	77.26	77.56	77.70	77.44
	CBA	75.63	73.34	56.72	73.74	**77.87**	75.11	76.01	74.80	71.80	72.93	71.42
SUSY: 8:1	AvAcc	90.34	90.92	90.91	87.52	89.81	87.92	91.50	**91.70**	86.81	86.92	86.20
	MAvG	73.09	75.11	**89.38**	66.00	71.41	66.56	77.95	79.46	64.67	64.85	63.68
	AvFb	75.74	76.15	61.37	75.62	**76.41**	74.41	75.13	74.27	75.27	75.26	75.13
	CBA	75.41	**75.63**	56.11	68.00	73.57	69.07	73.21	71.65	66.61	66.82	65.46
SUSY: 16:1	AvAcc	94.25	94.95	94.95	90.41	92.36	91.61	**95.08**	94.80	91.03	89.70	89.08
	MAvG	70.36	75.86	**91.77**	56.65	62.25	59.12	77.46	74.87	57.96	55.13	53.93
	AvFb	72.99	72.13	59.10	72.62	**74.21**	71.38	71.26	70.79	72.68	72.22	71.83
	CBA	**72.60**	70.29	55.60	62.60	67.16	65.07	68.99	68.83	63.79	61.30	60.26
SYN DOS	AvAcc	100.00	99.99	99.99	99.99	99.98	99.99	99.99	100.00	99.01	100.00	**100.00**
	MAvG	99.61	99.42	99.21	99.42	95.81	98.48	99.60	99.61	45.08	99.81	**100.00**
	AvFb	99.76	99.25	98.41	99.41	98.95	99.22	99.12	99.61	77.41	99.49	**99.84**
	CBA	99.42	98.64	98.03	98.84	95.91	98.50	98.82	99.41	59.68	99.22	**99.80**
Traffic	AvAcc	84.87	84.85	80.06	84.70	84.75	84.36	84.68	**85.10**	84.82	84.81	84.79
	MAvG	45.07	45.94	14.89	43.07	42.83	42.20	**46.62**	45.28	44.11	43.34	43.27
	AvFb	43.55	43.19	18.03	44.26	44.36	41.08	41.39	**45.64**	43.37	44.34	44.34
	CBA	40.20	39.58	12.86	40.96	40.75	37.81	37.77	**42.12**	40.20	41.33	41.42
Video Injection	AvAcc	99.99	99.99	**100.00**	99.99	99.98	99.99	99.99	99.99	99.97	99.99	99.99
	MAvG	99.93	99.93	**99.97**	99.88	99.81	99.93	99.95	99.91	99.63	99.88	99.89
	AvFb	99.96	99.94	99.97	99.95	99.95	99.93	**99.97**	99.96	99.90	99.94	99.95
	CBA	99.90	99.91	**99.96**	99.87	99.80	99.87	99.92	99.90	99.61	99.86	99.87

Table 6: Comparison of 11 oversampling methods using four different metrics with SVM as the base classifier.

		ADASYN	Borderline SMOTE	CCR	Cluster SMOTE	Gaussian SMOTE	k-Means SMOTE	NRAS	Random Oversample	Safe Level SMOTE	SMOTE	SMOTE-D
Bitcoin	AvAcc	20.85	98.58	98.58	9.79	**98.58**	98.58	80.16	59.72	98.58	41.34	98.58
	MAvG	2.38	0.00	0.00	12.33	0.00	0.00	13.07	2.38	0.00	**13.59**	0.00
	AvFb	12.66	49.86	49.86	8.62	**49.86**	49.86	45.23	31.26	49.86	27.37	49.86
	CBA	10.43	49.29	49.29	5.03	**49.29**	49.29	41.36	29.86	49.29	21.34	49.29
Cover Type	AvAcc	81.40	**89.35**	85.36	83.95	83.27	83.98	83.27	82.87	83.62	83.11	83.04
	MAvG	22.76	4.75	0.00	25.58	**26.43**	18.26	24.56	25.07	25.93	24.26	24.65
	AvFb	33.03	32.25	11.80	38.85	**40.52**	31.73	38.72	38.19	39.80	37.88	37.45
	CBA	20.25	25.53	6.97	25.70	**26.54**	22.95	24.68	24.13	25.49	24.64	24.22
Fuzzing	AvAcc	60.55	**80.99**	80.72	80.94	71.80	80.90	58.78	59.19	62.19	62.79	43.98
	MAvG	57.43	**83.86**	0.00	78.45	63.66	74.49	56.46	56.64	58.07	58.42	50.62
	AvFb	63.77	48.79	47.72	50.48	**73.59**	48.82	62.24	62.60	65.19	65.73	49.38
	CBA	42.15	41.33	40.36	43.21	**52.87**	41.41	40.46	40.82	43.49	44.07	28.11
HIGGS: 4:1	AvAcc	20.00	80.00	80.00	79.77	80.00	79.15	**80.00**	44.00	80.00	80.00	80.00
	MAvG	0.00	0.00	0.00	17.43	0.00	20.56	**30.61**	0.00	0.00	0.00	0.00
	AvFb	27.78	47.62	47.62	47.83	47.62	**48.27**	47.64	35.71	47.62	47.62	47.62
	CBA	10.00	40.00	40.00	40.32	40.00	**41.13**	40.02	22.00	40.00	40.00	40.00
HIGGS: 8:1	AvAcc	11.11	88.89	88.89	88.77	**88.89**	86.66	88.40	73.33	88.89	88.89	88.89
	MAvG	0.00	0.00	0.00	25.31	0.00	23.47	**54.21**	0.00	0.00	0.00	0.00
	AvFb	19.23	48.78	48.78	48.96	48.78	**50.08**	49.90	42.87	48.78	48.78	48.78
	CBA	5.56	44.44	44.44	44.65	44.44	**46.76**	45.71	36.67	44.44	44.44	44.44
HIGGS: 16:1	AvAcc	76.47	94.12	94.12	94.09	**94.12**	90.62	71.04	41.18	94.12	94.12	94.12
	MAvG	0.00	0.00	0.00	24.37	0.00	30.08	**32.74**	0.00	0.00	0.00	0.00
	AvFb	41.89	49.38	49.38	49.44	49.38	**49.85**	44.05	26.90	49.38	49.38	49.38
	CBA	38.24	47.06	47.06	**47.12**	47.06	47.12	39.52	20.59	47.06	47.06	47.06
IoT	AvAcc	89.93	92.91	88.49	93.60	90.26	92.76	90.39	91.39	91.71	91.06	**93.69**
	MAvG	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**30.78**
	AvFb	39.99	54.87	27.24	62.68	45.55	53.46	47.19	52.24	55.30	51.55	**62.71**
	CBA	29.25	45.74	19.24	52.27	35.75	42.45	37.28	41.92	45.35	41.65	**52.70**
MIRAI	AvAcc	89.79	84.60	84.29	89.72	**90.89**	87.21	89.73	89.72	89.73	89.73	89.72
	MAvG	78.05	75.74	72.98	77.95	**81.52**	77.87	77.97	77.95	77.96	77.96	77.94
	AvFb	**89.33**	51.88	49.83	89.27	84.26	69.99	89.28	89.27	89.27	89.27	89.26
	CBA	74.38	45.56	43.60	74.26	**81.74**	60.80	74.29	74.27	74.28	74.28	74.26
Poker: 0 vs 2	AvAcc	91.32	91.32	91.32	56.19	**91.32**	91.32	54.33	74.79	91.32	51.51	91.32
	MAvG	0.00	0.00	0.00	27.90	0.00	0.00	27.96	0.00	0.00	**28.14**	0.00
	AvFb	49.07	49.07	49.07	42.73	**49.07**	49.07	42.06	42.47	49.07	40.97	49.07
	CBA	45.66	45.66	45.66	33.05	**45.66**	45.66	31.93	37.40	45.66	30.25	45.66
Poker: 0 vs 3	AvAcc	40.81	95.96	95.96	54.22	**95.96**	95.96	57.90	59.19	95.96	52.03	95.96
	MAvG	0.00	0.00	0.00	19.45	0.00	0.00	**19.76**	0.00	0.00	19.09	0.00
	AvFb	25.06	49.58	49.58	37.15	**49.58**	49.58	39.08	33.23	49.58	35.88	49.58
	CBA	20.40	47.98	47.98	29.29	**47.98**	47.98	31.33	29.60	47.98	28.07	47.98
Poker: 0 vs 4	AvAcc	20.46	99.23	99.23	67.24	**99.23**	99.23	82.66	59.85	99.23	66.67	99.23
	MAvG	0.00	0.00	0.00	**12.06**	0.00	0.00	9.63	0.00	0.00	11.89	0.00
	AvFb	11.48	49.92	49.92	39.31	**49.92**	49.92	44.62	30.70	49.92	38.96	49.92
	CBA	10.23	49.62	49.62	34.37	**49.62**	49.62	42.06	29.92	49.62	34.06	49.62
Poker: 0 vs 5	AvAcc	59.92	99.60	79.76	62.00	**99.60**	99.60	66.87	40.08	99.60	59.04	99.60
	MAvG	0.00	0.00	0.00	6.32	0.00	0.00	**7.12**	0.00	0.00	6.97	0.00
	AvFb	30.36	49.96	40.16	34.53	**49.96**	49.96	37.07	20.57	49.96	33.33	49.96
	CBA	29.96	49.80	39.88	31.25	**49.80**	49.80	33.74	20.04	49.80	29.78	49.80
Poker: 0 vs 6	AvAcc	40.06	99.72	99.72	63.10	**99.72**	99.72	64.57	59.94	99.72	59.75	99.72
	MAvG	0.00	0.00	0.00	5.38	0.00	0.00	**5.86**	0.00	0.00	5.17	0.00
	AvFb	20.41	49.97	49.97	34.78	**49.97**	49.97	35.17	30.26	49.97	33.16	49.97
	CBA	20.03	49.86	49.86	31.73	**49.86**	49.86	32.51	29.97	49.86	30.04	49.86
Poker: 1 vs 2	AvAcc	26.08	89.87	89.87	52.24	**89.87**	89.87	56.10	73.92	89.87	51.54	89.87
	MAvG	0.00	0.00	0.00	30.05	0.00	0.00	30.10	0.00	0.00	**30.31**	0.00
	AvFb	24.20	48.90	48.90	42.30	**48.90**	48.90	43.94	42.72	48.90	42.21	48.90
	CBA	13.04	44.94	44.94	31.46	**44.94**	44.94	33.88	36.96	44.94	31.05	44.94
Poker: 1 vs 3	AvAcc	22.86	95.24	95.24	54.68	**95.24**	95.24	60.66	40.95	95.24	52.50	95.24
	MAvG	0.00	0.00	0.00	21.50	0.00	0.00	**21.79**	0.00	0.00	21.28	0.00
	AvFb	17.90	49.51	49.51	38.60	**49.51**	49.51	41.50	25.80	49.51	37.44	49.51
	CBA	11.43	47.62	47.62	29.99	**47.62**	47.62	33.33	20.48	47.62	28.76	47.62
Poker: 1 vs 4	AvAcc	40.18	99.09	99.09	60.92	**99.09**	99.09	92.53	40.18	99.09	60.21	99.09
	MAvG	0.00	0.00	0.00	**12.11**	0.00	0.00	11.40	0.00	0.00	12.01	0.00
	AvFb	21.28	49.91	49.91	36.38	**49.91**	49.91	48.97	21.28	49.91	35.99	49.91
	CBA	20.09	49.54	49.54	31.19	**49.54**	49.54	47.29	20.09	49.54	30.81	49.54
Poker: 1 vs 5	AvAcc	59.91	99.53	99.53	57.71	**99.53**	99.53	59.03	59.91	99.53	57.32	99.53
	MAvG	0.00	0.00	0.00	6.06	0.00	0.00	**7.21**	0.00	0.00	6.62	0.00
	AvFb	30.43	49.95	49.95	32.43	**49.95**	49.95	33.41	30.43	49.95	32.41	49.95
	CBA	29.95	49.77	49.77	29.10	**49.77**	49.77	29.81	29.95	49.77	28.92	49.77
Poker: 1 vs 6	AvAcc	59.93	99.66	99.66	59.86	**99.66**	99.66	55.57	40.07	99.66	57.40	99.66
	MAvG	0.00	0.00	0.00	5.41	0.00	0.00	5.31	0.00	0.00	**5.45**	0.00
	AvFb	30.31	49.97	49.97	33.28	**49.97**	49.97	31.20	20.48	49.97	32.12	49.97
	CBA	29.97	49.83	49.83	30.12	**49.83**	49.83	27.96	20.03	49.83	28.88	49.83
SEER	AvAcc	89.00	**90.01**	89.21	88.92	89.01	89.26	89.20	88.24	89.28	89.24	89.37
	MAvG	32.82	33.75	0.00	31.09	34.42	26.12	**37.15**	31.57	35.23	33.45	32.20
	AvFb	38.57	39.91	18.53	37.63	40.01	32.48	**42.33**	37.45	41.07	39.90	37.87
	CBA	31.26	**35.54**	11.39	31.56	32.36	27.11	33.47	29.98	33.96	32.92	31.47
SSL Renegotiation	AvAcc	90.08	95.80	**95.80**	83.51	90.12	89.35	90.10	90.08	90.11	90.10	83.51
	MAvG	54.04	0.00	0.00	45.04	**54.11**	16.13	54.08	54.06	54.10	54.08	45.04
	AvFb	78.83	49.57	49.57	70.87	78.87	48.79	78.85	78.84	**78.87**	78.85	70.87
	CBA	59.53	47.90	47.90	51.54	**59.60**	47.92	59.57	59.54	59.59	59.56	51.54
SUSY: 4:1	AvAcc	**81.12**	80.27	80.00	80.32	80.07	80.06	80.63	80.97	80.19	80.36	80.31
	MAvG	78.28	**79.26**	0.00	78.27	78.82	69.25	79.22	77.91	79.06	78.09	78.46
	AvFb	**53.06**	48.76	47.62	49.03	47.90	48.18	50.32	52.70	48.44	49.22	48.97
	CBA	**45.79**	41.06	40.00	41.32	40.26	40.55	42.54	45.55	40.76	41.50	41.26
SUSY: 8:1	AvAcc	89.11	88.89	88.89	89.04	88.90	88.97	**89.17**	73.99	88.96	89.05	89.04
	MAvG	**77.98**	37.71	0.00	76.89	70.77	69.15	76.78	68.78	76.83	76.29	77.42
	AvFb	51.08	48.79	48.78	50.40	48.95	50.05	**51.89**	46.24	49.51	50.60	50.35
	CBA	46.53	44.45	44.44	45.91	44.60	45.60	**47.28**	39.78	45.10	46.09	45.86
SUSY: 16:1	AvAcc	**94.15**	94.12	94.12	94.11	94.11	94.09	94.09	88.84	94.11	94.11	94.11
	MAvG	**70.10**	0.00	0.00	66.97	57.65	59.90	67.21	57.18	66.20	67.79	68.03
	AvFb	52.64	49.38	49.38	51.05	49.51	51.08	52.91	**56.24**	49.96	51.28	51.10
	CBA	49.96	47.06	47.06	48.53	47.17	48.58	**50.23**	49.73	47.57	48.74	48.57
SYN DOS	AvAcc	79.35	99.75	**99.75**	77.33	79.35	99.71	99.62	72.19	70.44	71.46	99.62
	MAvG	9.92	0.00	0.00	17.54	9.82	**52.10**	47.77	9.56	9.23	9.39	47.66
	AvFb	43.73	49.97	49.97	44.79	43.68	59.08	**60.67**	40.39	39.46	39.99	60.50
	CBA	40.17	49.87	49.87	41.51	40.16	58.16	**60.27**	36.52	35.61	36.13	60.08
Traffic	AvAcc	78.10	78.99	78.78	77.74	**79.13**	77.68	78.13	78.29	78.91	78.37	78.77
	MAvG	20.40	**20.79**	7.67	18.35	20.68	17.89	19.85	19.91	20.40	19.46	20.37
	AvFb	20.84	**21.90**	14.03	19.71	21.27	19.31	21.38	20.90	21.83	21.03	20.95
	CBA	16.67	18.33	9.05	16.12	16.65	16.11	**18.36**	17.13	18.22	17.61	16.48
Video Injection	AvAcc	31.77	95.85	**95.85**	54.22	54.69	75.50	54.41	53.11	54.30	42.63	71.12
	MAvG	18.45	0.00	0.00	**34.18**	20.81	19.04	20.68	20.47	20.65	25.60	24.62
	AvFb	27.18	49.57	49.57	40.02	39.57	47.83	39.31	38.45	39.23	35.79	**50.40**
	CBA	17.27	47.93	**47.93**	29.70	29.14	40.00	28.97	28.23	28.91	23.42	39.11

Table 7: Comparison of 11 oversampling methods using four different metrics with Naive Bayes as the base classifier.

		ADASYN	Borderline SMOTE	CCR	Cluster SMOTE	Gaussian SMOTE	k-Means SMOTE	NRAS	Random Oversample	Safe Level SMOTE	SMOTE	SMOTE-D
Bitcoin	AvAcc	50.83	98.58	98.58	55.62	**98.58**	70.90	61.26	50.74	63.35	49.58	68.48
	MAvG	13.60	0.00	0.00	12.09	0.00	11.44	12.95	13.49	**14.19**	13.37	11.29
	AvFb	32.24	49.86	49.86	33.39	**49.86**	40.61	36.88	32.14	38.60	31.49	39.46
	CBA	26.24	49.29	49.29	28.61	**49.29**	36.43	31.58	26.19	32.77	25.58	35.18
Cover Type	AvAcc	83.99	**89.57**	89.14	85.78	86.14	83.40	86.48	86.15	86.27	86.09	85.64
	MAvG	28.59	**37.62**	0.00	31.59	31.43	18.51	34.95	31.42	32.11	31.36	30.49
	AvFb	42.35	43.71	29.06	44.82	46.63	30.62	**49.94**	46.59	47.46	46.54	45.52
	CBA	28.38	**38.64**	23.56	31.56	32.90	24.37	36.93	32.87	33.67	32.79	31.38
Fuzzing	AvAcc	66.17	**81.14**	80.99	77.95	75.43	80.60	71.06	70.63	72.22	70.80	73.75
	MAvG	60.21	66.44	**89.83**	52.12	64.68	61.54	62.89	62.67	63.50	62.76	64.04
	AvFb	68.57	57.26	48.65	57.04	**74.78**	48.67	72.67	72.36	73.51	72.48	74.21
	CBA	47.35	50.86	41.19	49.07	**57.75**	41.38	52.21	51.75	53.52	51.93	55.47
HIGGS: 4:1	AvAcc	47.40	54.18	80.00	52.78	**80.00**	57.95	53.64	53.73	60.76	53.93	54.93
	MAvG	42.67	43.22	0.00	41.54	0.00	41.75	42.48	43.35	43.25	**43.44**	43.16
	AvFb	47.06	50.19	47.62	48.30	47.62	49.77	49.40	50.14	**51.82**	50.29	50.39
	CBA	32.62	38.50	40.00	37.34	40.00	41.81	38.06	38.10	**44.34**	38.27	39.17
HIGGS: 8:1	AvAcc	51.73	88.89	88.89	55.34	**88.89**	57.64	55.79	56.14	67.08	54.98	56.45
	MAvG	33.88	0.00	0.00	32.42	0.00	32.21	33.60	**34.36**	33.78	34.17	33.89
	AvFb	44.70	48.78	48.78	45.03	48.78	45.55	46.07	46.90	**49.80**	46.30	46.65
	CBA	31.89	44.44	44.44	34.10	**44.44**	35.57	34.56	34.85	42.01	34.06	35.00
HIGGS: 16:1	AvAcc	57.67	94.12	94.12	57.24	**94.12**	61.44	60.88	53.16	71.95	54.55	55.13
	MAvG	25.20	0.00	0.00	24.21	0.00	23.78	23.93	25.57	**26.18**	25.64	25.54
	AvFb	42.30	49.38	49.38	41.37	**49.38**	42.81	42.57	40.57	48.36	41.27	41.46
	CBA	32.49	47.06	47.06	32.12	**47.06**	34.47	34.17	29.95	40.85	30.75	31.08
IoT	AvAcc	92.30	92.34	**92.83**	90.91	92.25	90.59	91.35	91.88	91.82	91.88	92.41
	MAvG	0.00	0.00	0.00	0.00	**0.00**	0.00	0.00	0.00	0.00	0.00	0.00
	AvFb	54.18	54.62	54.14	49.80	55.87	41.82	54.87	56.86	56.67	**56.88**	56.32
	CBA	42.18	44.31	42.08	35.41	43.48	29.54	40.91	43.27	42.93	43.26	**46.25**
MIRAI	AvAcc	89.88	89.89	89.89	89.87	89.89	**90.38**	89.88	89.88	89.88	89.88	89.86
	MAvG	78.19	78.20	78.20	78.18	78.20	**79.77**	78.19	78.18	78.19	78.19	78.16
	AvFb	89.43	**89.44**	89.44	89.42	89.44	87.16	89.42	89.42	89.42	89.42	89.41
	CBA	74.55	74.57	74.57	74.54	74.57	**77.83**	74.55	74.54	74.55	74.55	74.52
Poker: 0 vs 2	AvAcc	56.38	91.32	91.32	57.05	**91.32**	53.57	55.49	51.67	57.26	51.81	52.25
	MAvG	**28.58**	0.00	0.00	28.02	0.00	28.27	27.93	28.18	28.40	28.15	28.02
	AvFb	43.38	49.07	49.07	43.17	**49.07**	41.36	42.54	41.07	43.61	41.11	41.21
	CBA	33.25	45.66	45.66	33.59	**45.66**	31.53	32.64	30.35	33.77	30.43	30.68
Poker: 0 vs 3	AvAcc	53.34	95.96	95.96	55.17	**95.96**	64.65	59.76	51.78	56.91	52.53	53.83
	MAvG	19.36	0.00	0.00	19.41	0.00	19.13	**19.84**	19.01	19.41	19.09	19.24
	AvFb	36.68	49.58	49.58	37.57	**49.58**	41.47	39.98	35.71	38.40	36.12	36.84
	CBA	28.81	47.98	47.98	29.81	**47.98**	34.92	32.35	27.93	30.75	28.34	29.06
Poker: 0 vs 4	AvAcc	73.05	99.23	99.23	70.75	**99.23**	80.08	84.94	69.03	66.73	70.16	69.53
	MAvG	12.28	0.00	0.00	**12.64**	0.00	9.55	8.49	12.39	10.56	12.46	12.28
	AvFb	42.04	49.92	49.92	41.21	**49.92**	44.26	45.21	40.30	38.35	40.85	40.47
	CBA	37.36	49.62	49.62	36.21	**49.62**	40.71	43.13	35.31	34.00	35.90	35.56
Poker: 0 vs 5	AvAcc	61.16	99.60	**99.60**	64.46	99.60	71.16	71.43	59.90	62.31	61.08	63.18
	MAvG	6.80	0.00	0.00	6.50	**11.52**	5.10	7.45	7.01	7.38	6.97	6.98
	AvFb	34.28	49.96	49.96	35.71	**50.11**	38.40	39.25	33.75	35.01	34.30	35.29
	CBA	30.85	49.80	49.80	32.49	**49.93**	35.79	36.06	30.21	31.45	30.81	31.87
Poker: 0 vs 6	AvAcc	78.69	99.72	99.72	65.91	**99.72**	75.05	66.05	59.91	70.06	61.88	63.69
	MAvG	**5.79**	0.00	0.00	5.40	0.00	5.43	4.25	5.25	5.14	5.19	5.39
	AvFb	41.94	49.97	49.97	36.09	**49.97**	40.24	35.74	33.26	37.94	34.17	35.07
	CBA	39.59	49.86	49.86	33.14	**49.86**	37.74	33.16	30.12	35.22	31.11	32.03
Poker: 1 vs 2	AvAcc	51.13	89.87	89.87	53.05	**89.87**	61.23	57.24	52.16	60.17	51.78	52.36
	MAvG	30.24	0.00	0.00	29.97	0.00	30.20	30.04	30.18	**30.55**	30.34	30.39
	AvFb	41.96	48.90	48.90	42.59	**48.90**	45.72	44.34	42.37	45.79	42.33	42.62
	CBA	30.78	44.94	44.94	31.95	**44.94**	37.12	34.58	31.42	36.50	31.20	31.57
Poker: 1 vs 3	AvAcc	53.38	95.24	95.24	55.58	**95.24**	59.98	62.62	52.58	59.94	53.00	53.45
	MAvG	21.40	0.00	0.00	21.50	0.00	21.36	**21.87**	21.40	21.60	21.34	21.28
	AvFb	37.94	49.51	49.51	39.01	**49.51**	40.67	42.39	37.56	41.07	37.72	37.88
	CBA	29.26	47.62	47.62	30.48	**47.62**	32.90	34.43	28.82	32.91	29.04	29.28
Poker: 1 vs 4	AvAcc	68.29	99.09	99.09	63.78	**99.09**	69.98	98.25	61.60	64.87	62.69	63.53
	MAvG	**12.48**	0.00	0.00	12.43	0.00	11.94	7.61	12.19	12.08	12.30	12.41
	AvFb	39.99	49.91	49.91	37.89	49.91	40.43	**50.22**	36.75	38.22	37.31	37.77
	CBA	34.99	49.54	49.54	32.67	49.54	35.81	**49.63**	31.54	33.21	32.11	32.54
Poker: 1 vs 5	AvAcc	57.04	99.53	99.53	59.41	**99.53**	61.85	60.94	57.00	58.39	58.71	59.58
	MAvG	6.42	0.00	0.00	6.15	0.00	**7.54**	7.21	6.65	6.57	6.62	6.61
	AvFb	32.22	49.95	49.95	33.25	**49.95**	34.86	34.32	32.27	32.91	33.08	33.49
	CBA	28.77	49.77	49.77	29.96	**49.77**	31.25	30.77	28.76	29.46	29.62	30.06
Poker: 1 vs 6	AvAcc	81.85	99.66	99.66	61.75	**99.66**	72.01	56.74	57.65	76.20	59.10	60.42
	MAvG	**5.97**	0.00	0.00	5.36	0.00	5.70	5.41	5.43	5.51	5.46	5.41
	AvFb	43.37	49.97	49.97	34.16	**49.97**	38.95	31.79	32.23	40.78	32.93	33.54
	CBA	41.21	49.83	49.83	31.07	**49.83**	36.24	28.55	29.01	38.34	29.74	30.40
SEER	AvAcc	88.63	**90.90**	89.47	90.05	89.96	88.46	89.26	89.54	90.75	89.50	90.17
	MAvG	34.56	0.00	0.00	34.84	38.99	21.18	37.49	38.21	**41.09**	38.12	36.85
	AvFb	34.90	31.56	19.48	35.71	38.88	26.51	37.96	38.29	**39.56**	38.17	36.21
	CBA	29.26	25.75	12.58	31.24	33.37	22.33	32.75	32.26	**33.73**	32.16	31.29
SSL Renegotiation	AvAcc	83.45	83.46	83.46	83.49	83.46	**92.47**	83.46	83.46	83.46	83.46	83.47
	MAvG	44.61	44.41	44.41	**44.85**	44.66	24.03	44.67	44.67	44.67	44.67	44.75
	AvFb	70.30	70.02	70.01	**70.61**	70.35	50.08	70.37	70.37	70.37	70.37	70.47
	CBA	51.38	51.33	51.33	**51.48**	51.40	49.31	51.40	51.40	51.40	51.40	51.43
SUSY: 4:1	AvAcc	73.41	79.15	80.00	73.84	**82.86**	70.92	77.25	74.38	77.62	73.79	74.29
	MAvG	59.83	65.02	17.89	60.28	**75.34**	53.61	62.61	60.55	63.15	60.10	60.44
	AvFb	67.89	**69.00**	47.62	68.30	62.66	61.03	68.29	68.30	68.71	68.07	68.22
	CBA	57.69	**66.89**	40.00	58.21	56.00	55.28	63.59	59.01	64.11	58.19	58.90
SUSY: 8:1	AvAcc	77.05	88.94	88.89	74.30	**89.31**	73.79	81.28	74.43	81.39	74.16	74.91
	MAvG	49.78	**79.68**	0.00	48.32	70.47	44.69	52.66	48.44	52.93	48.24	48.64
	AvFb	64.60	49.18	48.78	63.69	60.77	59.98	65.41	63.80	**65.65**	63.62	63.94
	CBA	52.83	44.79	44.44	50.15	56.62	49.05	57.46	50.27	**57.64**	50.01	50.72
SUSY: 16:1	AvAcc	78.56	94.12	**94.12**	74.92	93.82	76.22	84.98	75.52	84.74	74.71	75.37
	MAvG	38.78	19.40	0.00	37.01	**64.66**	33.59	42.98	37.49	42.81	37.03	37.34
	AvFb	60.31	49.40	49.38	58.31	59.44	55.38	**62.99**	58.86	62.95	58.31	58.70
	CBA	47.63	47.08	47.06	44.85	**56.87**	44.94	53.22	45.35	52.99	44.72	45.21
SYN DOS	AvAcc	70.49	79.39	79.39	83.63	70.51	95.65	**99.70**	70.46	70.46	70.46	99.63
	MAvG	8.93	10.93	10.97	18.58	9.22	58.71	**60.21**	9.22	9.23	9.22	48.78
	AvFb	39.36	44.23	44.25	48.08	39.49	58.11	**61.29**	39.46	39.47	39.46	60.58
	CBA	35.62	40.27	40.27	44.75	35.64	56.23	**60.32**	35.62	35.62	35.62	60.08
Traffic	AvAcc	76.86	78.77	78.56	76.43	**79.10**	77.04	77.84	76.91	78.99	76.79	77.04
	MAvG	20.10	**22.06**	0.00	17.66	20.85	16.31	19.61	20.72	20.71	20.58	20.25
	AvFb	18.06	**21.64**	9.14	15.77	21.14	16.10	20.07	18.66	19.54	18.46	16.89
	CBA	14.52	17.67	3.77	13.32	**18.49**	13.21	17.32	14.77	15.54	14.51	12.98
Video Injection	AvAcc	43.72	95.85	**95.85**	75.78	84.64	86.31	95.00	82.59	88.51	82.71	94.60
	MAvG	16.25	0.00	0.00	37.80	41.77	48.68	**66.13**	40.04	47.86	40.17	64.62
	AvFb	30.01	49.57	49.57	56.03	66.34	55.54	**81.39**	64.77	71.57	64.94	79.86
	CBA	23.40	47.93	47.93	45.52	51.39	50.72	**70.06**	49.53	56.11	49.65	69.07

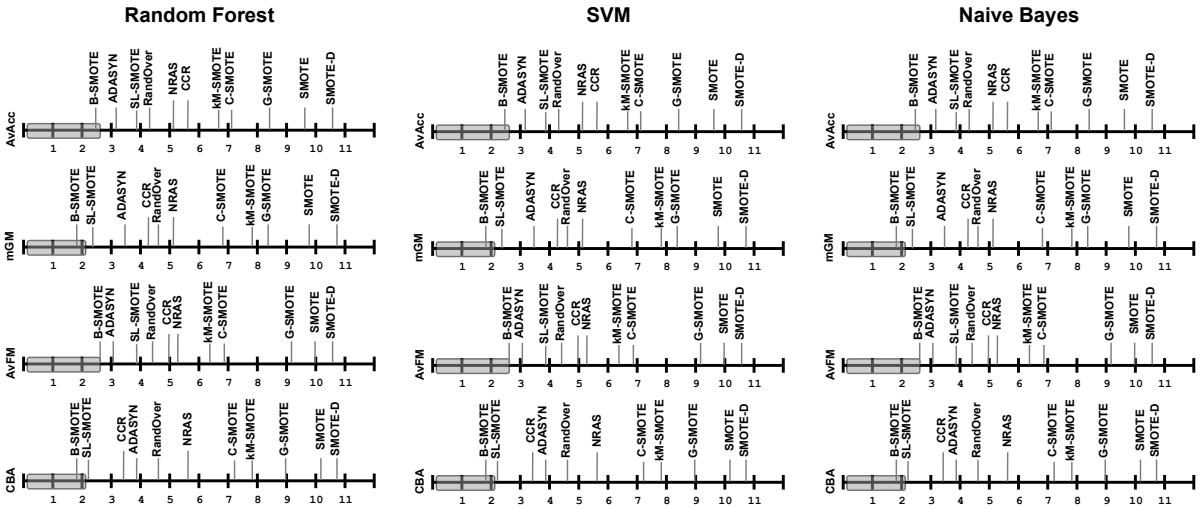


Figure 5: The Bonferroni–Dunn test for comparison among examined oversampling methods with respect to evaluation metric and used classifier.

4.4.2 Experiment 1: Oversampling Algorithms for Binary Big Imbalanced Data

Overview of learning difficulties. Binary big imbalanced data has well-defined class roles, hence the learning difficulty comes from the imbalanced ratio combined with the volume of each class and potential presence of instance-level difficulties, such as borderline instances, subconcepts, multiple modalities, or class overlap. Furthermore, due to the distributed nature of the Spark system, an additional challenge may arise when the minority class is significantly smaller (i.e., extreme imbalance ratio) than the majority class. In such a case, the minority class will be further limited in size by being distributed among multiple nodes, leading to potential problem of learning from a small sample size. Therefore, oversampling methods are a natural choice for tackling such problems, by overcoming limitations in minority class size in each Spark node.

Informed vs standard oversampling. As discussed in previous sections, all the

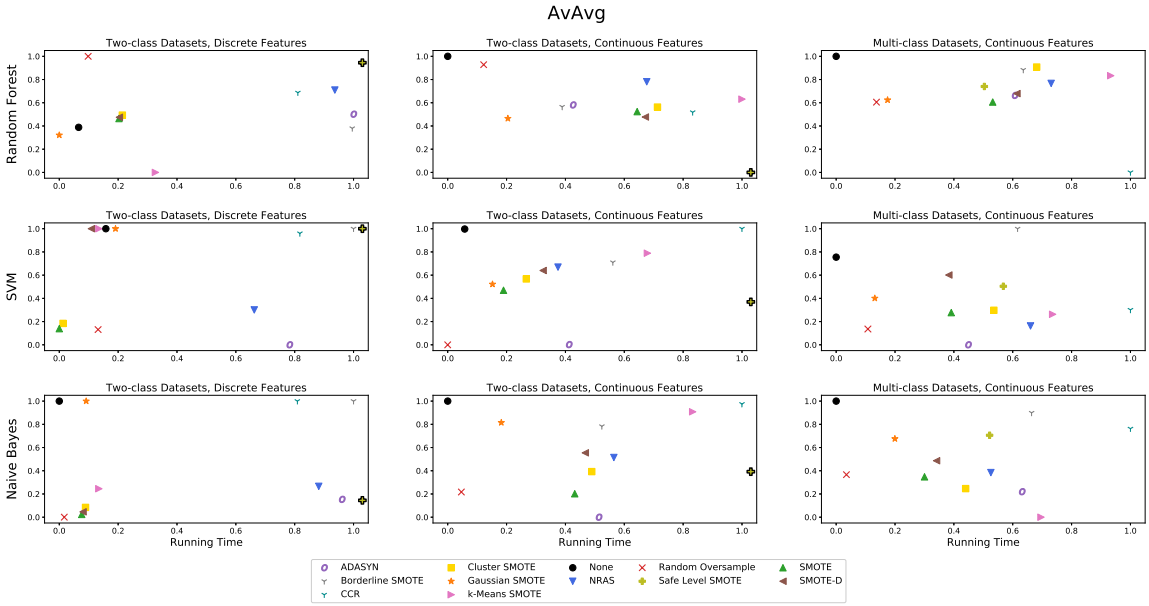


Figure 6: The relative run times and AvAvg scores for the different dataset groups, classifiers and sampling methods.

proposed oversampling methods for Spark can be analyzed from the point of how much information regarding class/instance distribution they use. Random Oversampling is the extreme example of using no such information, SMOTE uses only the neighborhood information in a basic way, while methods like CCR or RBO rely on an extensive analysis of instance-level properties. While intuition suggests that informed oversampling will always outperform standard approaches, our results show that this is not always the case. Random oversampling is capable of outperforming multiple informed variants, such as all SMOTE modifications based on clustering or Gaussian spread data. This shows that not all of information about instances is equally important when utilized on the Spark architecture. As Spark uses its own data partitioning algorithm when creating data subsets for each node, the spatial relationship among instances is affected. Thus, oversampling methods based on in-

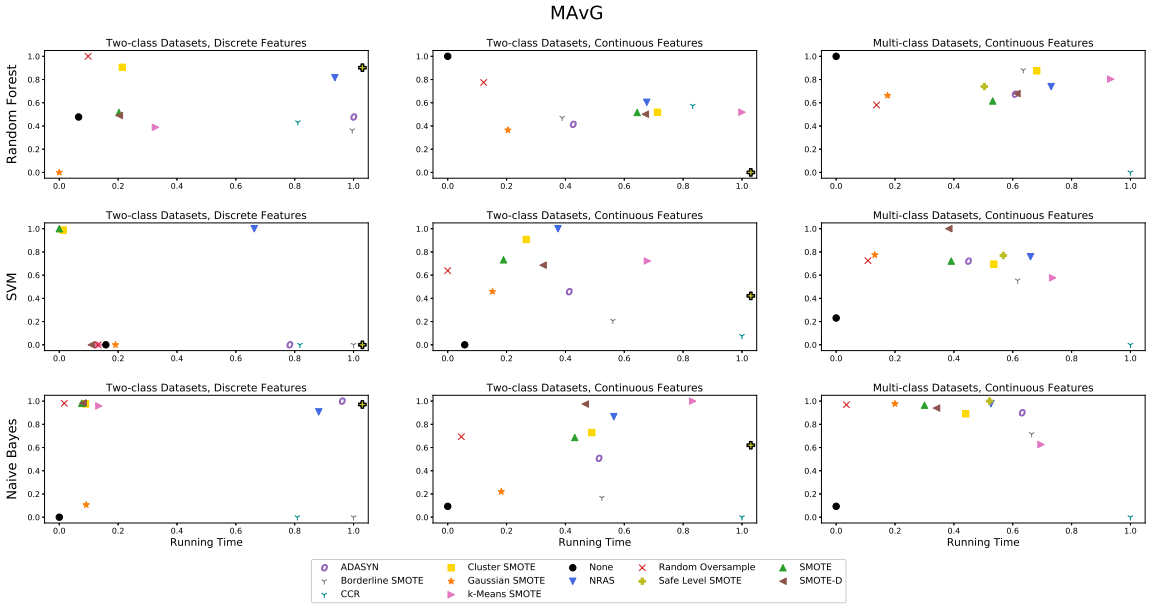


Figure 7: The relative run times and MAvG scores for the different dataset groups, classifiers and sampling methods.

troducing instances within each cluster / Gaussian mode cannot work efficiently and random oversampling can outperform them. However, oversampling methods that use more advanced information regarding neighborhood of each instance display the best possible performance. This proves that local information is much more useful to big data oversampling than modality-based information for each class.

Comparison among informed oversampling methods. The four best performing informed oversampling methods, regardless of analyzed metric, are Borderline SMOTE, Safe-level SMOTE, ADASYN, and CCR. All those methods use information about local data difficulty factors, such as class overlapping or homogeneity of neighborhood, to introduce new artificial instances. Differences in performance of these algorithms on big binary datasets are small and depend on specific characteristics of the considered data set. Borderline SMOTE performs best in cases when

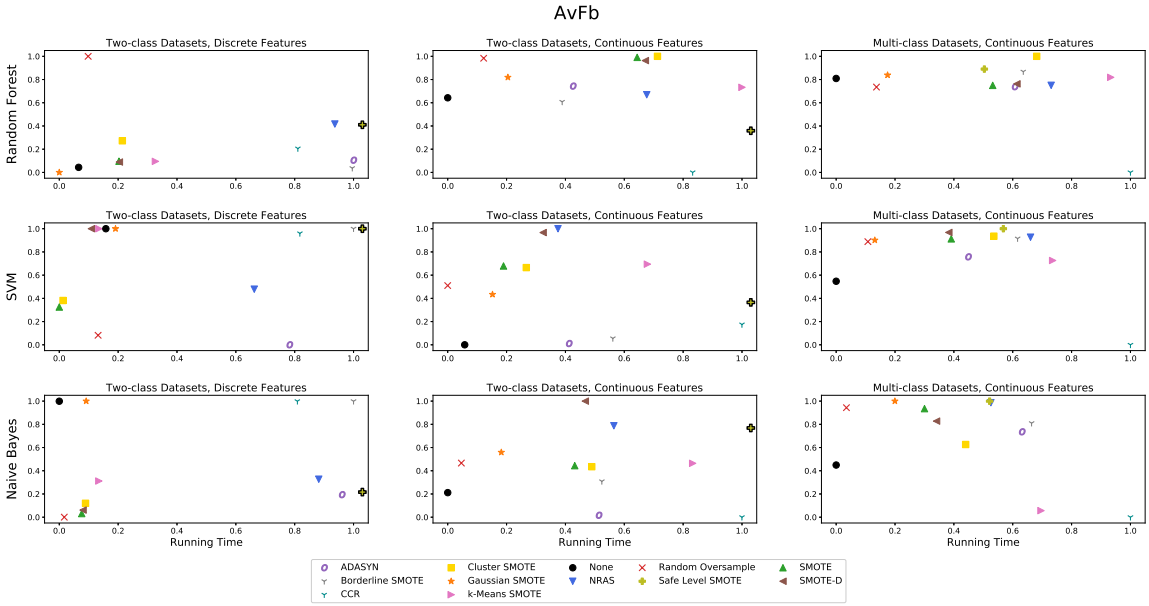


Figure 8: The relative run times and AvFb scores for the different dataset groups, classifiers and sampling methods.

there is a high class overlapping and minority class needs to be reinforced around the uncertainty region (i.e., decision boundary). Safe-level SMOTE performs best for datasets where the disproportion between classes is the main source of learning difficulty. Safe-level SMOTE reinforces homogeneous regions of minority class, leading to balancing in the instance quantity without affecting uncertainty regions (as new instances are introduced in highly certain regions of minority class). ADASYN and CCR offer excellent mechanisms for dealing with rare instances and outliers, hence performing best on difficult datasets with high number of small subconcepts or atypical observations. CCR further augments its performance by cleaning the overlapping regions from majority class observations, leading to excellent performance on small disjuncts.

Impact of base classifier selection. Interestingly, all three base classifiers (Ran-

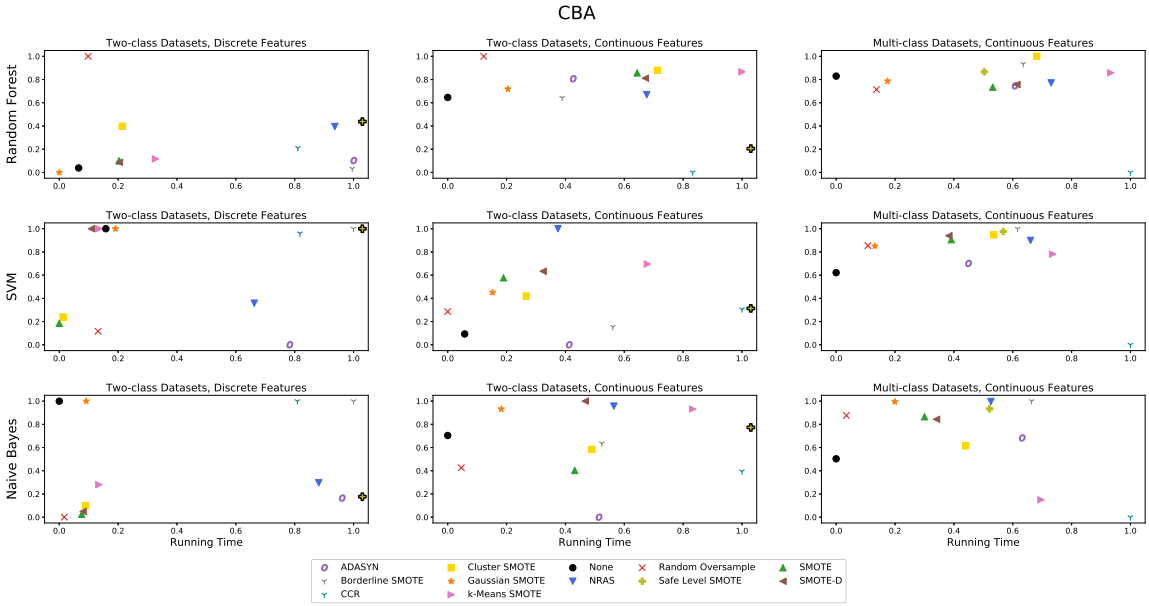


Figure 9: The relative run times and CBA scores for the different dataset groups, classifiers and sampling methods.

dom Forest, SVM, and Naïve Bayes) offer similar correlations on which oversampling methods work the best. Random Forest can utilize random oversampling more efficiently, as it can be combined with its ensemble structure to offer improved diversity. Single-model classifiers (SVM and Naïve Bayes) benefit from using guided oversampling much more significantly, as this leads to better separation margins or class probability estimations. In summary, four guided oversampling methods highlighted in the previous point (Borderline-SMOTE, Safe-level SMOTE, ADASYN, and CCR) offer excellent and robust performance regardless of with which classifier they are being paired.

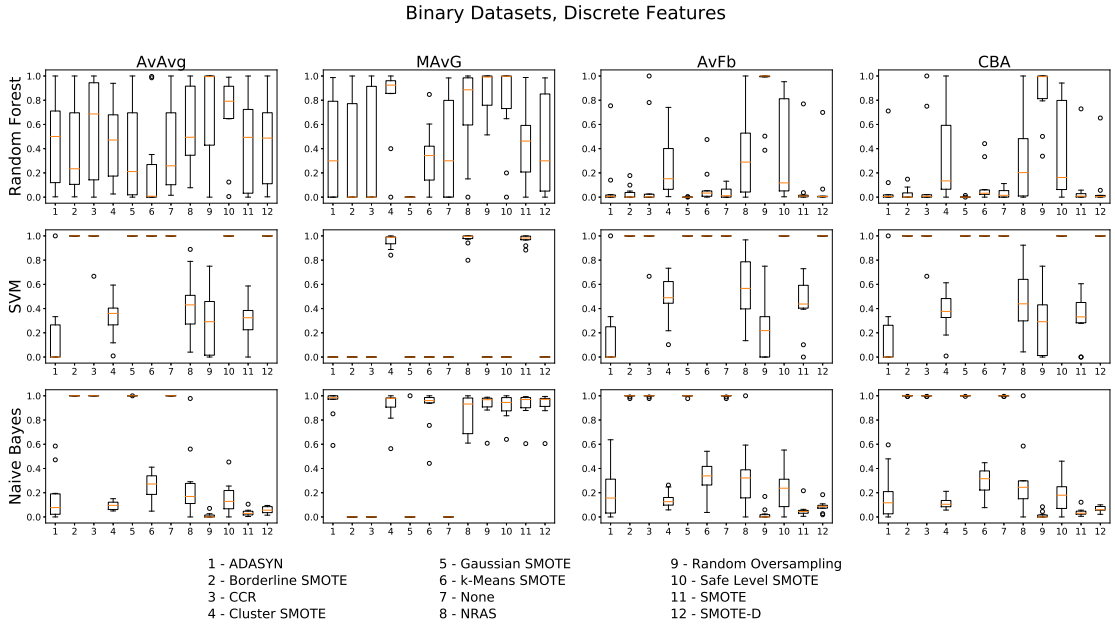


Figure 10: The distribution of metric scores for each sampling method and classifier using the Binary Datasets with Discrete Features.

4.4.3 Experiment 2: Oversampling Algorithms for Multi-class Big Imbalanced Data

Differences between binary and multi-class problems. Multi-class imbalanced problems pose a variety of unique challenges to oversampling algorithms. Our software package offers universal implementations of the discussed oversampling methods that can work with binary and multi-class massive datasets. While there are oversampling solutions dedicated specifically to multi-class problems, they usually are characterized by a prohibitive computational cost when dealing with big data (e.g., MC-RBO, extension of RBO). All proposed algorithms have been adapted with one-vs-all approach, allowing them to leverage global and local data properties, without intra-class relationships becoming a choking point. We can observe similar trends with binary data, where informed oversampling approaches significantly outperformed other so-

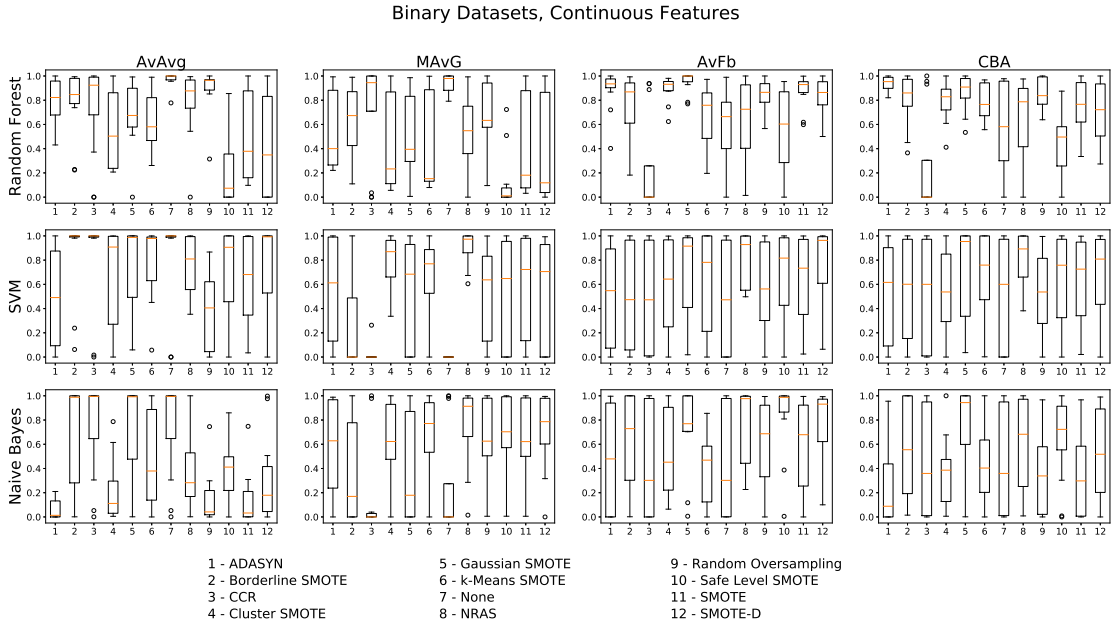


Figure 11: The distribution of metric scores for each sampling method and classifier using the Binary Datasets with Continuous Features.

lutions, although the differences are even more pronounced. This shows that the importance of guided oversampling becomes even more crucial when dealing with multiple classes.

Role of informative oversampling. While most of the examined oversampling algorithms behave similarly to their binary counterparts, we can see improvement in the performance of Cluster SMOTE. This is the only global informative oversampling that returns predictive accuracy like its local informative counterparts. We can explain this by the fact that in the case of multiple classes we are more likely to deal with more complex problems, where modalities in each class are more pronounced. Additionally, by using the one-vs-all approach, we create a super-class of majority instances that consists of multiple joint classes. This scenario allows Cluster SMOTE to create pairwise oversampling areas, detecting multiple potential decision boundaries.

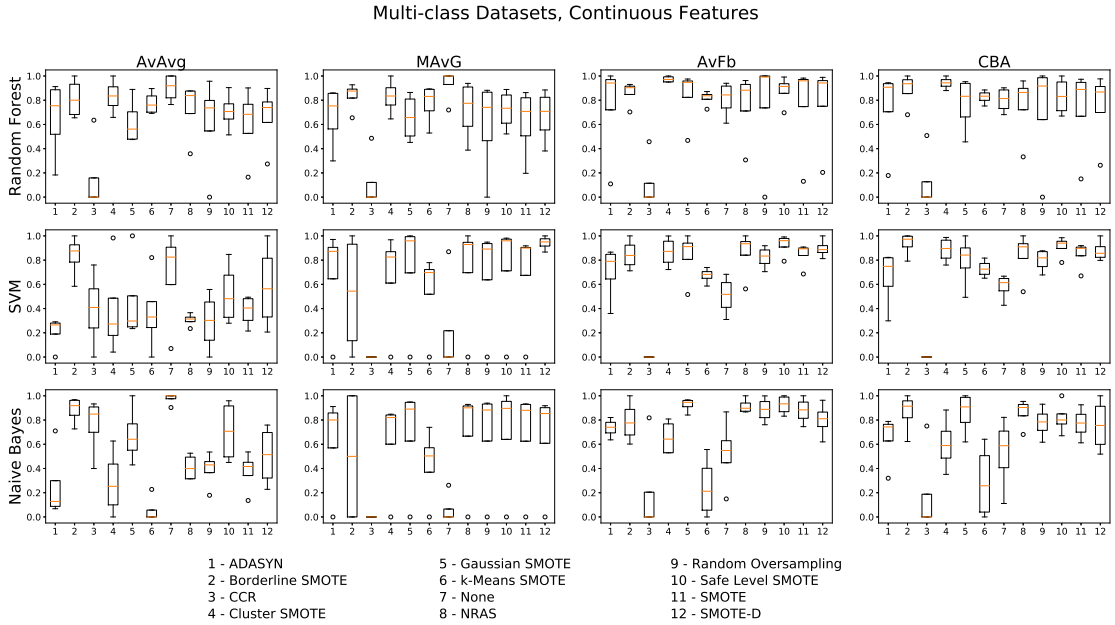


Figure 12: The distribution of metric scores for each sampling method and classifier using the Mutli-class Datasets with Continuous Features.

While its performance is significantly better than in the binary case, local methods such as Borderline SMOTE, ADASYN, and CCR are still the best performing ones. This shows that the importance of local instance-level information is as important in multi-class scenarios as it is in binary ones.

Behavior of base classifiers in multi-class problems. While in binary scenarios, all three classifiers returned similar synergies with oversampling methods and we can see that Random Forest and SVM behave differently from Naive Bayes. This is because Random Forest is an explicit ensemble and SVM is an implicit ensemble for multi-class problems (as we train them in one-vs-all mode). Therefore, those compound classifiers are capable of better generalization over multiple skewed distributions, leveraging the additional information provided by oversampling algorithms.

4.4.4 Experiment 3: Investigating Oversampling Trade-off Between Accuracy and Time Complexity

When selecting an oversampling algorithm for a given big data problem, it is important to realize the existence of a trade-off between accuracy and run time. We believe that the end user should be able to chose any of the oversampling algorithms present in our software package based on their needs and available resources. To this end, we present the visualizations of the relationships between the analyzed oversampling method accuracy (according to four metrics) and computational time in Figures 6 - 9.

As shown in these results, the combination of data properties and the target classifier may by the deciding factor when choosing a class balancing algorithm. For example, Random Oversampling produced some of the best trade-offs for the Two-Class datasets with the Random Forest classifier but did not perform as well for the Multi-class datasets. In addition, it often performed quite poorly with the SVM and Naive Bayes classifiers on all three dataset groups. Algorithms such as Borderline SMOTE, Cluster SMOTE, Gaussian SMOTE and NRAS showed to have some of the best accuracy and time trade-offs for certain dataset and classifier combinations but no algorithm was the clear overall winner.

4.4.5 Experiment 4: Scalability of Oversampling Algorithms

In addition to the twelve methods presented so far, three other sampling algorithms were also implemented: ANS, MWMOTE, RBO. As these methods have shown competitive results in their original publications and received interested from other researchers, they were added to our slate of oversampling methods. However, during the implementation and initial testing of these methods, it became apparent

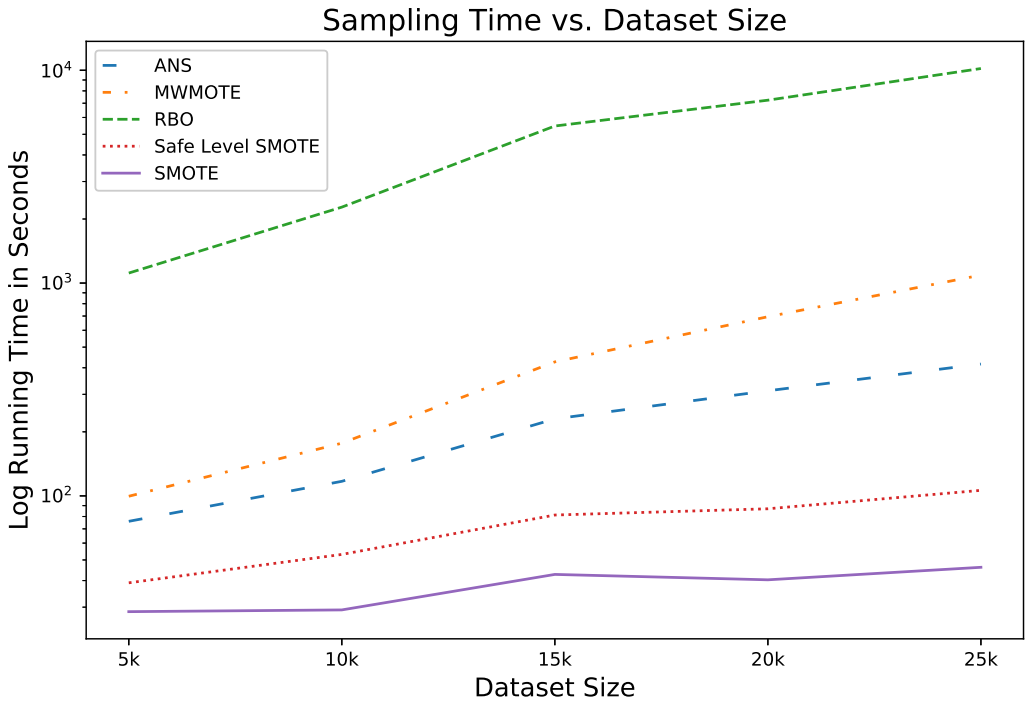


Figure 13: Sampling times for increasing dataset sizes with five sampling methods on the Covertype dataset.

that their lack of scalability made them unsuitable for our main experiments.

To illustrate this issue, Figure 13 shows the running times for several small subsets of the Covertype dataset with ANS, MWMOTE, RBO and two reference methods of SMOTE and Safe Level SMOTE. Like our main experiments, these were performed with 5-fold cross validation so in each run only 80% of the data was used for oversampling. The three slower method took significantly longer than the reference methods and required a log scale for the time axis in order to reasonably present these results.

Using linear regression, the sampling times were estimated for using a dataset of 100,000 examples as shown in Table 8. Compared to standard SMOTE, the others methods were significantly slower with the following approximate time increases: ANS (15x), MWMOTE (40x), RBO (400x). While ANS and MWMOTE may be feasible for 100,000 example datasets, it was not practicable for this projects as we used 5-fold cross validation and 26 different datasets, resulting in 130 individual experiments per sampling method/classifier combination.

Projected Sampling Times				
SMOTE	**Safe Level SMOTE**	**ANS**	**MWMOTE**	**RBO**
116	359	1,718	4,744	44,540

Table 8: Projected sampling times in seconds for oversampling the Covtype dataset with 100k examples.

4.5 Guidelines for Designing Oversampling Algorithms for Imbalanced Big Data

4.5.1 Insight into Commonly Used Algorithmic Components

Each of the implemented oversampling methods were based on one or more commonly used algorithmic components. In this section, we provide guidelines on the appropriateness of these components in distributed algorithms designed for processing large datasets. Table 9 shows a list of the components present in each oversampling method, sorted by approximate run time (fastest to slowest).

k-NN. Although the k-NN algorithm is included in many machine learning libraries, it is not currently part of the official Spark MLlib package. For this reason, we have used the efficient distributed implementation from spark-knn which is based on a hybrid spill tree. Since k-NN has an approximate asymptotic complexly of $\mathcal{O}(n \log n)$, it is practical to be used for large datasets as shown in our results. However, including multiple k-NN models can noticeably increase run time and should only be done when truly necessary. In some cases, it may be possible to avoid multiple k-NNs by using one model with a larger k value and filter the results for use in different sub-tasks.

k-Means. The k-Means clustering algorithm is part of Spark MLlib and is optimized for distributed tasks making it straightforward to use. While k-Means has a higher theoretical asymptotic complexity then k-NN, our results confirm that the Spark MLlib implementation can run effectively in practice. When using k-Means as part of an oversampling method, the optimal results and run time may be dependent on the number of clusters and total iterations. Discovering these exact values may require an expensive hyperparameter search.

Dependent Loop. A dependent loop occurs when each iteration requires results

from a previous iteration, preventing trivial parallelism. While this approach can work well in serial algorithms, it becomes problematic with distributed computing and is incompatible with the MapReduce design. Ideally, MapReduce decomposes tasks so that all independent work can be done in parallel. By adding a dependency in a loop, MapReduce cannot break down these tasks further and so each loop must be run in a serial manner. In cases where there are more CPUs than loops, a number of these CPUs will go idle. If these loops are being run per training example, as it is the case for a number of the presented algorithms, there is no guarantee that each loop will take the same amount of time, potentially leading to significant bottlenecks. These dependent loops should be avoided whenever possible as they limit scalability and make run time less predictable.

Probability. Oversampling methods that use probability select examples based on rank ordering. Like dependent loops, this method does not translate well from serial to distributed environments. Probabilistic sampling requires the calculation of a probability/likelihood score, sorting the values and indexing the resulting list for example selection. With MapReduce, data is distributed across multiple computation nodes and the probability based methods require collecting all of the results back to the driver to ensure correct results which requires an expensive serial step.

Linear Regression. Like k-Means, Spark MLlib includes linear regression so its use in oversampling methods is trivial. Although it is not prohibitively slow, including linear regression does add to the execution time and is one of the reasons NRAS is slower than similar algorithms.

Neighbors by Radius. Many of the neighborhood calculations done with oversampling algorithms only consider a fixed number of neighbors as done with a k-NN search. However, several of the presented algorithms instead need to find neighbors

Algorithmic Components Present in Oversampling Methods							
Sampling Method	**kNN Count**	**k-Means**	**Dependent Loop**	**Probability**	**Linear Regression**	**Neighbors by Radius**	**$\geq\mathcal{O}(n^2)$**
ADASYN	1	×	×	×	×	×	×
SMOTE	1	×	×	×	×	×	×
Gaussian SMOTE	1	×	×	×	×	×	×
SMOTE-D	1	×	×	✓	×	×	×
NRAS	1	×	×	×	✓	×	×
Cluster SMOTE	1	✓	×	×	×	×	×
k-Means SMOTE	1	✓	×	×	×	×	✓
CCR	1	×	✓	×	×	✓	×
Safe Level SMOTE	2	×	×	×	×	×	×
Borderline SMOTE	2	×	×	×	×	×	×
ANS	3	×	✓	×	×	✓	×
MWMOTE	3	✓	×	✓	×	×	✓
RBO	0	×	✓	×	×	×	✓

Table 9: A listing of high level algorithmic components present in each oversampling method. The list is sorted by approximate running time, fastest to slowest.

within a radius. One downside of using a radius based search is that there is no limit of how many neighbors may be returned. This can lead to potential bottlenecks with computational time and memory use as each computational node may be processing different amounts of data depending on the density of these radius based neighborhoods. This approach should be avoided if possible but could be somewhat mitigated by setting a hard limit on the number of neighbors to return.

$\mathcal{O}(n^2)$ Complexity. Using any sub-components that have an asymptotic complexity of $\mathcal{O}(n^2)$ or worse should be avoided at all costs. While such methods may work for small datasets, the lack of scalability makes them unsuitable for big data problems.

4.5.2 Design Choices and Future Directions for Big Data Oversampling Algorithms.

Local over global data characteristics. Informative oversampling methods can be divided into two groups: (i) ones utilizing global data characteristics; and (ii) ones utilizing local data characteristics. Global oversampling algorithms are usually based on finding modalities in data, such as subconcepts or underlying clusters. Lo-

cal oversampling algorithms rely on analysis of the neighborhood of each minority instance and using this information to guide the artificial instance generation and placement. From our experimental study, we can see that global methods (such as Cluster SMOTE) returned highly unsatisfactory performance, while local methods were constantly among the best performing ones. We contribute this fact to the way Spark architecture creates data subsets for each node, leading to decreased importance of global and spatial data characteristics, while the local ones still hold. Therefore, we recommend to focus on local instance-level difficulties when designing novel oversampling algorithms for imbalanced big data.

Trade-off on instance-level characteristics analysis. While local instance-level characteristics can be highly beneficial to the design of oversampling methods, there is a plethora of information that can be considered here. The more detailed information on instance neighborhoods, region homogeneity, or class overlap that is taken into consideration, leading to a higher potential for effective generation of artificial instances. However, extraction of such information over big data comes at the expense of a significant computational cost. This can be seen with RBO that uses advanced analysis of binary and multi-class potential generated by instances and classes to optimize the oversampling process. While RBO returns excellent performance, its complexity leads to a lack of scalability to big data, making it an unfeasible algorithm for Spark. Therefore, we recommend utilizing low-complexity analysis of local data properties. Even simple small neighborhood analysis or sketching of local distributions will lead to better predictive accuracy, while still being scalable to massive datasets.

Potential in hybrid solutions. The experimental study presented in this paper, as well as our previous works and other findings from the literature, point to the superiority of oversampling to undersampling when handling imbalanced big data on

the Spark architecture. These findings point to the fact that undersampling tends to remove important instances (especially for multi-class problems) and that the consolidation of node-based undersampled data does not lead to good generalization capabilities. However, one of the best performing methods considered in this study (CCR) employs a form of undersampling / data cleaning by translating majority observations. This allows us to conclude that while undersampling on its own may deliver underwhelming results, it potentially should be considered as a component for enhancing oversampling methods. Several works on small-scale datasets point out to the usefulness of oversampling for cleaning uncertainty regions and this approach can be effectively translated to big data scenarios. We recommend exploring the direction of hybrid oversampling that leverages undersampling / data cleaning steps to improve the empowerment of the minority classes. At the same time, we recommend using undersampling that considers local data characteristics, as it returns much more favorable results for the Spark architecture than random approach.

Challenging hyperparameter tuning. Another potential bottleneck for oversampling methods lies in the difficulty of hyperparameter tuning for massive datasets. While using either grid search or one of dedicated parameter tuning methods is possible on Spark, it relates to significant computational resource consumption. This can be prohibitive, especially with on-demand computing using commercially available Spark hardware. Let us again use RBO as an example – apart from its high computational complexity, it has a significant number of hyperparameters to be tuned. This is another potential choke point for deploying such an algorithm in real-world applications. Other oversampling methods presented in this study use much fewer parameters, making their tuning a more feasible task. We recommend designing oversampling algorithms for imbalanced big data with the smallest possible number of parameters (ideally non-parametric) or enhancing them with self-tuning solutions that

can automatically adapt to the provided data without needing tedious user-guided tuning.

4.6 Conclusions

In this work, we have presented a holistic view on imbalanced big data oversampling. We have shown that the problem of learning from binary and multi-class skewed problems is highly pervasive in the big data domain, posing unique learning difficulties for machine learning systems. Existing oversampling methods cannot be directly utilized for high-performance architectures due to their specific computing requirements. We have shown how to adapt 14 popular oversampling algorithms to Spark architecture, making them suitable for high-performance distributed computing. Even with these adaptations, we have observed that not all oversampling methods translate well to big data problems and the complexity of some of them becomes prohibitive and does not scale well with data set size. We identified crucial components of modern oversampling algorithms and shown their impact on the classification performance and time complexity. This allowed us to identify specific oversampling components that translate well to high-performance distributed architectures and that have desired properties from the predictive performance viewpoint. We have also created guidelines and future directions for designing novel oversampling algorithms for imbalanced big data that can be adopted by the research community. To facilitate future research and reproducibility in this domain, we have prepared an efficient library for Spark with implementations of the discussed oversampling methods.

CHAPTER 5

MINORITY TYPE CONSIDERATIONS

Despite more than two decades of progress, learning from imbalanced data is still considered as one of the contemporary challenges in machine learning. This has been further complicated by the advent of the big data era, where popular algorithms dedicated to alleviating the class skew impact are no longer feasible due to the volume of datasets. Additionally, most of existing algorithms focus on binary imbalanced problems, where majority and minority classes are well-defined. Multi-class imbalanced data poses further challenges as the relationship between classes is much more complex and simple decomposition into a number of binary problems leads to a significant loss of information. In this paper, we propose the first compound framework for dealing with multi-class big data problems, addressing at the same time the existence of multiple classes and high volumes of data. We propose to analyze the instance-level difficulties in each class, leading to understanding what causes learning difficulties. We embed this information in popular sampling algorithms which allows for informative balancing of multiple classes. We propose an efficient implementation of the discussed algorithm on Apache Spark, including a novel version of SMOTE that overcomes spatial limitations in distributed environments of its predecessor. Extensive experimental study shows that using instance-level information significantly improves learning from multi-class imbalanced big data [130].

5.1 Oversampling for Multi-class Imbalanced Data on Spark

A popular approach to address class imbalanced is to either over or undersample the dataset. Examples can be added or removed to make the size of minority classes closer to the majority class. The simplest method is to undersample the majority class as shown in Algorithm 17. Two popular methods of oversampling are Synthetic Minority Over-sampling Technique (SMOTE) [127] and simple oversampling as shown in Algorithms 16 and 18. Simple oversampling is performed by duplicating randomly selected instances and adding them to the current dataset. In Spark, this is achieved by using the *sample* method on a DataFrame, with or without replacement. Sampling with replacement may be necessary to balance the classes if a minority class needs to be more than doubled in size. The sampled instances are then joined to the existing DataFrame to form the balanced dataset.

SMOTE is a more advance method of oversampling that creates new instances from existing ones rather than simply adding duplicates. To create a new instance, five random existing instances are chosen and their features are averaged creating a new synthetic instance. Algorithm 16 shows a method in Spark to create a new SMOTE instance.

Algorithm 16 Generate SMOTE instance

Ensure: Instances in DF belong to the same class
  procedure: SMOTE($DF, ClassLabel$)
  $fvs = Array[5]$
  for $i = 1$ to 5 **do**
    $index \leftarrow Random.nextInt(DF.count)$
    $instance \leftarrow DF[index]$
    $fvs[i] \leftarrow instance.filter(featureVector)$
  end for
  $transposed \leftarrow fvs^{\mathrm{T}}$
  $averages \leftarrow transposed.map(rowSum/5)$
  $smoteFeatureVector \leftarrow averages^{\mathrm{T}}$
  $smoteInstance \leftarrow \{smoteFeatureVector, ClassLabel\}$
  EMIT $smoteInstance$

5.2 Instance-level Difficulty in Multi-class Imbalanced Big Data

In this section, we discuss the importance of analyzing instance-level properties of the minority class, the taxonomy behind evaluating the difficulty of a given instance, and how to extend this taxonomy to multi-class problems. We also give details on our proposal to directly incorporate the given taxonomy of instance-level difficulty into previously discussed sampling methods for imbalanced big data.

5.2.1 Local Data Characteristics

While the disproportion in the number of instances among classes (imbalance ratio) is often considered as the main challenge to be faced, it is actually not the sole factor behind learning difficulties. Recent works showed that if one would have well-separated classes, then even extreme imbalance ratio would not lead to a skewed classifier [27]. This leads to a conclusion that other factors must be in play. The new trend in imbalanced data lies in analyzing the instance-level difficulties [131] (or instance hardness [132]) as a way of understanding what may cause classifiers to fail [42]. Specific instances may pose a much higher challenge for a classifier and thus hinder significantly the training procedure. Popular taxonomy divides instances that may be present in a class into four distinct types [29]:

- **Safe.** These instances lie far away from the potential decision boundary and usually form one or more dense clusters. They pose low difficulty to most of classifiers and thus often can be omitted during preprocessing.

- **Borderline.** These instances lie close to the decision boundary and may overlap to some degree with another class. They pose a significant difficulty, as incorrect sampling may lead to either increase in overlapping or removing minority instances from important part of the decision space.

- **Rare.** These instances form small disjuncts, often overlapping with another class. They pose high difficulty, as one must deal with both class overlap and small sample size, usually not sufficient to properly represent minority class in this part of the decision space.

- **Outliers.** These instances are singular objects located far from the main distribution of the minority class. They may be either rare and atypical cases, or noisy instances. Determining their nature requires a background knowledge about the classification problem. In case of noisy instances, they should be filtered out. In case of rare instances, they should be preserved and enhanced in order to allow a classifier to capture this specific part of the minority class distribution.

Example visualization of this taxonomy over a real-world dataset is presented in Figure 14.

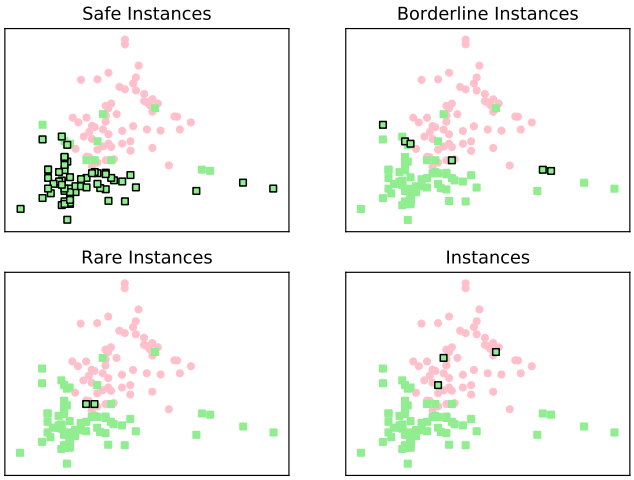


Figure 14: Instances of the first two labels in the UCI Wine dataset [74], using features *Magnesium* and *Color intensity*. Minority type specific instances have black borders.

This taxonomy can be extended to a multi-class imbalanced scenario [11]. Using

instance-level information can further improve the sampling approaches dedicated to multi-class imbalanced data, especially when dealing with multiple classes overlapping or multi-modal classes. Example visualization of multi-class extension of this taxonomy is presented in Figure 15.

This taxonomy has not been used for imbalanced big data so far. We propose to incorporate this approach into various data-level algorithms dedicated to Spark architecture and multi-class problems. Using CPU clusters will allow us to analyze the instance-level difficulty of very large datasets in significantly reduced amount of time. Due to the nature of imbalanced big data, we envision that this additional information will be highly useful to deal with class bias when dealing with millions of instances.

5.2.2 Incorporating Local Data Characteristics into Oversampling

Correcting for class imbalance is often done by under- or oversampling but the standard approaches do not take in account the quality of the instances to be sampled. Here we propose a Spark implementation for class balancing using instance minority types.

To calculate the minority class status of each existing instance we must first perform All k-Nearest Neighbors (AkNN) with k set to five. The AkNN algorithm performs k-NN on all instances while excluding the query points themselves from the returned neighbors. Each instance is given a minority type label based on the number of neighboring instances that belong to its own class. The resulting minority type labels for each instance is stored as a DataFrame which can also be saved to disk so it can be reused for future use. The AkNN calculations were generated using the *spark-knn* Spark package [129].

Data preprocessing starts with the dataset filtered using a user specified combi-

nation of minority types. There are fifteen valid combinations of the *safe*, *borderline*, *rare*, and *outlier* types. For example, if the *safe* and *rare* types are chosen only instances sharing those properties will be used and so the *borderline* and *outlier* instances will be dropped. Choosing all types would not remove any instances leaving the original dataset. Figure 15 shows an example of the UCI Wine dataset [74] filtered using minority types and the resulting SMOTE oversampling.

After the filtering, each class is then oversampled to match the number of instances of the majority class. The new oversampled instances are then unioned with the filtered dataset which results in

$$number\ of\ classes\ *majority\ class\ count$$

total instances in the preprocessed dataset. This procedure is detailed in Algorithm 18.

5.3 Leveraging SMOTE for MapReduce Environments

Clustering can also be introduced to further tune instance selection for oversampling. Even if minority types are used in oversampling, the indiscriminately sampled instances could fall almost anywhere in the total feature space. This raises an issue when using a method like SMOTE as a combination of disparate instances may result in a synthetic instance that do not represent any original instances in the dataset.

First, all instances in a given class are clustered using k-Means. Then for each new instance to be added, one of the clusters is randomly selected and SMOTE is performed on that subset of the dataset. Unlike standard SMOTE, the instances created are only based on localized regions of the dataset. This procedure is detailed in Algorithm 19.

This approach also addresses the challenge of using SMOTE on distributed data.

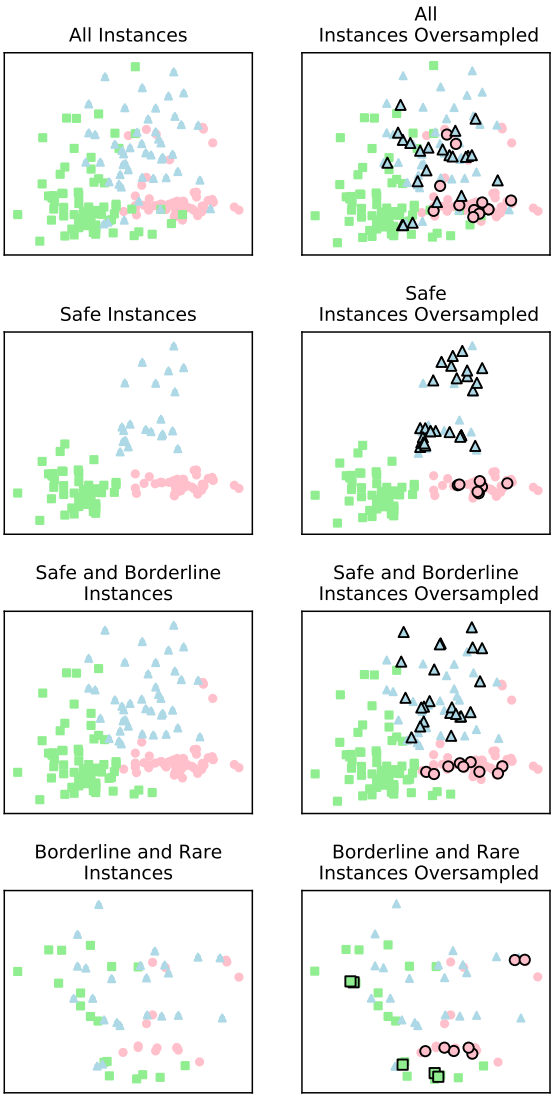


Figure 15: UCI Wine dataset [74], using features *Alcohol* and *Malic acid*, filtered by minority class types.

As Spark partitions data over its processing nodes, there is no guarantee that each partition preserves any spatial relationships. These resulting partitions may result in new synthetic examples that are not representative of the global population and perform much worse than non-distributed SMOTE. In addition, the quality of the synthetic examples may very greatly between different runs and thus making the

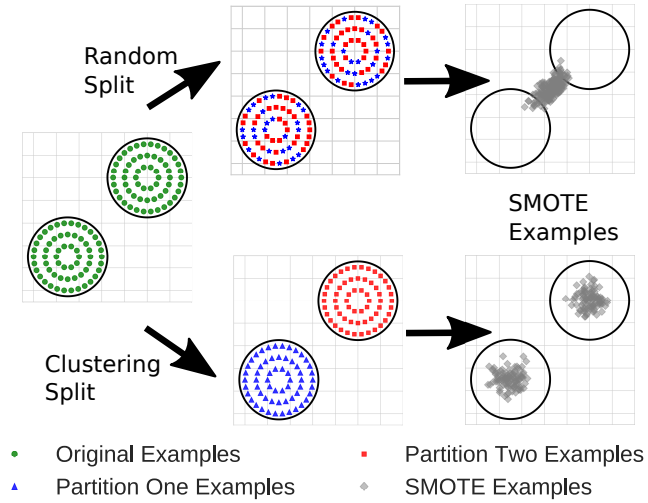


Figure 16: The impact of using clustering-based partitioning for SMOTE to maintain spatial coherency among instances in each class for each node in Spark.

Algorithm 17 Perform majority class undersampling

procedure: MajorityClassUndersample($DF, NumberOfClasses, TargetCount$)
$sampledDataFrames \leftarrow DataFrame[1 \text{ to } NumberOfClasses]$
for $c = 1$ to $NumberOfClasses$ **do**
  $DF_c \leftarrow DF.filter(x \rightarrow x.class == c)$
  if $DF_c.count > TargetCount$ **then**
    $sampledDataFrames[c] \leftarrow DF_c.sample(TargetCount)$
  else
    $sampledDataFrames[c] \leftarrow DF_c$
  end if
end for
EMIT $sampledDataFrames.union()$

final results dependent on how the data was distributed. Clustering with k-Means forces the data to be partitioned spatially resulting in SMOTE using only localized examples. The cluster variability is also lower when using k-Means compared to random partitioning.

Algorithm 18 Perform minority class oversampling

procedure: MinorityTypeResample($DF, MinorityTypes$, $NumberOfClasses, MajorityClassCount$)
$DF_{\text{mt}} \leftarrow DF.filter(x.minorityType \in MinorityTypes)$
for $c = 1$ to $NumberOfClasses$ **do**
  $DF_{\text{class}} \leftarrow DF_{\text{mt}}.filter(x \rightarrow x.class == c)$
  $instancesToAdd \leftarrow (1\ to\ MajorityClassCount\ -\ DF_{\text{class}}.count)$
  $DF_{\text{smote}} \leftarrow instancesToAdd.map(x \rightarrow \text{SMOTE}(DF_{\text{class}}, c))$
  $DF_{\text{mt}} \leftarrow DF_{\text{mt}}.join(DF_{\text{smote}})$
end for
EMIT DF_{mt}

Algorithm 19 Perform minority class type SMOTE oversampling within clusters

procedure: MinorityTypeSMOTEClusters(DF, k, $MinorityTypes, NumberOfClasses$, $MajorityClassCount$)
$DF_{\text{mt}} \leftarrow DF.filter(x.instanceType \in MinorityTypes)$
for $c = 1$ to $NumberOfClasses$ **do**
  $DF_{\text{class}} \leftarrow DF_{\text{mt}}.filter(x \rightarrow x.class == c)$
  $classClusters < clusterID, DF_{\text{clusterID}} > \leftarrow kMeans(DF_{\text{class}}, k)$
  $instancesToAdd \leftarrow (1\ to\ MajorityClassCount\ -\ DF_{\text{class}}.count)$
  $DF_{\text{smote}} \leftarrow instancesToAdd.map(x \rightarrow \text{SMOTE}(classClusters[randomID], c))$
  $DF_{\text{mt}} \leftarrow DF_{\text{mt}}.join(DF_{\text{smote}})$
end for
EMIT DF_{mt}

5.4 Experimental Study

This experimental study was carefully designed in order to answer the following research questions:

- **RQ1**: What is the most effective sampling solution for multi-class imbalanced big data when used in the Spark environment?
- **RQ2**: Does modifying sampling methods to take into account the instance-level difficulty types leads to a significant improvement in achieved predictive power?
- **RQ3**: Does the proposed clustering-based partitioning of data in each Spark node improve the performance of the SMOTE algorithm?

In the following section, we present the details of the used benchmarks, experimental set-up, as well as show obtained results and their discussion.

5.4.1 Dataset Benchmarks

In order to evaluate our proposed framework for handling multi-class imbalanced big data with Spark, we have selected five diverse real-world benchmark datasets. They are characterized by a large volume (500k - 3M instances), varied size of feature space (5 - 115 features), and high number of classes (7-55). Details of used datasets, together with their composition with respect to four instance difficulty levels, are depicted in Table 10.

5.4.2 Set-up

Classifiers. We have used two popular classifiers that are provided in Spark environment - Random Forest and Naive Bayes. This allows us to evaluate behavior of examined sampling methods with both single and ensemble classifiers, giving us insight on their areas of usefulness for such families of models.

Parameters of methods. Random Forest used 100 trees with maximum depth = 20.

Adaptation to multi-class problems. This is the first effort to propose algorithms for multi-class imbalanced big data on Spark. Both of the considered classifiers (Random Forest and Naive Bayes) are capable of handling multi-class problems. Additionally, we propose our multi-class adaptations of SMOTE, under- and oversampling, where each class is treated individually and sampled to a given size.

Training and testing. We have used a stratified 10-fold cross validation in order to maintain the original class distribution among the folds.

Evaluation metrics. In order to properly evaluate the performance of classifiers on

Table 10: Details about multi-class imbalanced big datasets used in experimental

Dataset	Instances	Features	Classes	Class distribution		Safe[%]	Borderline[%]	Rare[%]	Outliers[%]
covtype	581,012	54	7	Class 1	211840	84.97	10.43	3.03	1.57
				Class 2	283301	87.62	9.33	2.08	0.97
				Class 3	35754	80.81	13.96	3.41	1.82
				Class 4	2747	56.86	25.48	9.79	7.86
				Class 5	9493	60.32	25.18	8.57	5.93
				Class 6	17367	65.80	23.40	6.85	3.94
				Class 7	20510	87.25	8.37	2.85	1.54
traffic	1,378,663	26	7	Class 1	163473	22.35	32.59	22.96	22.10
				Class 2	214173	25.47	31.73	21.93	20.88
				Class 3	275622	34.96	32.68	17.81	14.55
				Class 4	343133	39.47	33.74	16.05	10.74
				Class 5	157710	24.75	32.21	21.87	21.17
				Class 6	179367	27.08	35.69	20.68	16.55
				Class 7	45185	22.85	19.79	19.33	38.03
seer	2,532,629	11	10	Class 1	396140	50.53	26.83	9.90	12.75
				Class 2	567055	79.01	14.12	4.07	2.80
				Class 3	241330	17.91	31.69	23.15	27.25
				Class 4	528648	60.71	28.71	6.11	4.48
				Class 5	135436	37.15	18.31	16.56	27.98
				Class 6	137908	58.65	29.12	6.16	6.06
				Class 7	100481	1.25	15.92	28.22	54.60
				Class 8	110411	26.39	20.59	23.27	29.75
				Class 9	166217	46.04	30.77	11.81	11.38
				Class 10	149003	33.03	38.15	16.50	12.33
sensors	2,219,803	5	55	Class 1	43039	53.47	28.28	11.15	7.10
				Class 2	46915	71.04	19.21	5.95	3.80
				Class 3	46631	55.52	29.91	9.43	5.14
				Class 4	43790	65.86	23.22	6.97	3.95
				Class 6	35654	62.37	21.71	9.07	6.85
				Class 7	55302	64.51	21.62	8.70	5.17
				Class 8	15810	32.68	37.25	17.45	12.61
				Class 9	45210	63.11	23.23	8.25	5.42
				Class 10	47120	54.23	27.79	10.76	7.22
				Class 11	41832	51.37	28.90	11.78	7.95
				...	...	...	...	...	...
				Class 47	56848	59.47	22.17	11.62	6.75
				Class 48	58210	64.88	20.22	8.96	5.94
				Class 49	34800	55.86	25.49	10.57	8.08
				Class 50	15737	35.75	35.21	16.56	12.48
				Class 51	42285	50.17	27.39	12.91	9.52
				Class 52	34058	52.81	26.84	12.43	7.92
				Class 53	25582	27.89	36.90	19.12	16.10
				Class 54	28748	33.06	36.84	17.07	13.03
				Class 55	2850	72.39	12.35	6.49	8.77
				Class 56	2372	81.58	11.68	4.30	2.45
				Class 58	4497	89.53	7.69	1.71	1.07
iot	3,000,000	115	11	Class 1	215643	99.90	0.06	0.02	0.02
				Class 2	275258	99.25	0.56	0.14	0.05
				Class 3	230494	99.96	0.02	0.01	0.00
				Class 4	313359	99.97	0.01	0.01	0.01
				Class 5	527165	99.69	0.24	0.05	0.02
				Class 6	224499	99.84	0.11	0.04	0.02
				Class 7	219925	99.96	0.03	0.01	0.00
				Class 8	111940	99.93	0.06	0.01	0.00
				Class 9	109278	99.95	0.03	0.01	0.01
				Class 10	367388	99.99	0.01	0.00	0.00
				Class 11	405051	99.98	0.02	0.00	0.00

imbalanced domains, skew-insensitive metrics are required. As multi-class imbalanced problems were not popular until recently among researchers, there is a lack of uniform

approach to which measures should be considered as a standard. In order to gain an in-depth insight into the performance of analyzed classifiers we have used the following nine popular metrics for multi-class imbalanced data:

$$AvAcc = \frac{1}{C}\sum_{i=1}^{C}\frac{tp_i + tn_i}{tp_i + tn_i + fp_i + fn_i}$$

$$Rec_M = \frac{1}{C}\sum_{i=1}^{C} recall_i$$

$$Prec_M = \frac{1}{C}\sum_{i=1}^{C} precision_i$$

$$Rec_U = \sum_{i=1}^{C} tp_i \Big/ \sum_{i=1}^{C} t_i$$

$$Prec_U = \sum_{i=1}^{C} tp_i \Big/ \sum_{i=1}^{C} p_i$$

$$F_{\beta M} = \frac{(1+\beta)^2 \cdot Prec_M \cdot Rec_M}{\beta^2 \cdot Prec_M + Rec_M}$$

$$F_{\beta \mu} = \frac{(1+\beta)^2 \cdot Prec_\mu \cdot Rec_\mu}{\beta^2 \cdot Prec_\mu + Rec_\mu}$$

$$AvF_\beta = \frac{1}{C}\sum_{i=1}^{C}\frac{(1+\beta^2) \cdot precision_i \cdot recall_i}{\beta^2 \cdot precision_i + recall_i}$$

$$CBA = \frac{1}{C}\sum_{i=1}^{C}\frac{mat_{i,i}}{max(\sum_{j=1}^{C} mat_{i,j}, \sum_{j=1}^{C} mat_{j,i})}$$

Statistical analysis of results. We have used a 10-fold cross validation F-test with confidence threshold $\alpha = 0.05$ to check if the obtained differences between algorithms are statistically significant.

Used Spark architecture. All experiments were performed in Spark environment with 32 computing nodes provided by Amazon Web Services.

5.4.3 Results and Discussion

What multi-class sampling method is most effective one for Spark (RQ1)? Tables 11 and 12 present obtained results for three evaluated multi-class sampling methods for Random Forest and Naive Bayes classifiers respectively. For both of these classifiers we can see similar trends - random oversampling performs better than random undersampling and SMOTE. This confirms the observations for binary imbalanced data on Spark reported in [84], where authors discussed the superiority of random oversampling. It is very interesting to see that SMOTE returns inferior performance, which is not typical for smaller-scale imbalanced datasets [27]. Therefore, there must be some additional factors specific only to the Spark environment that negatively impacts SMOTE. We will discuss this further when answering RQ3.

Does incorporating information about instance-level difficulties improve sampling (RQ2)? Tables 11 and 12 present results for enhanced versions of examined sampling methods (denoted with "+") that incorporate the information about instance types in each class. Furthermore, Figure **??** depicts the number of times when the enhanced sampling method was statistically significantly better than its basic counterpart according to 10 fold CV F-test. We can see that for vast majority of cases including the instance-level information leads to a significantly better sampling of multi-class imbalanced big data, regardless of the sampling method employed. This is proof that imbalance ratio is not the sole reason behind learning difficulties in multi-class imbalanced big data. When dealing with such large-scale and difficult problems additional information regarding properties of each class leads to a significantly better alleviation of skewed distributions.

Figures 18 and 19 depict relationships between sampling various combinations of instance types in each class and performance of the proposed informed sampling

to standard sampling and no sampling. Here we can clearly see that practically for every dataset there exist at least one combination of instance types that leads to improvements over uniformly sampling all instances. Random Forest benefits more from using our informed sampling than Naive Bayes which can be explained because the informed sampling can be an efficient tool for creating more diverse ensembles. This can then lead to creating more accurate base trees within Random Forest without sacrificing diversity or robustness of the ensemble.

Does clustering-based partitioning in Spark nodes improve SMOTE performance (RQ3)? As mentioned in RQ1, SMOTE returned highly unsatisfactory performance in the Spark environment. This is proof to our claims that SMOTE is not designed for MapReduce environments, as they do not maintain the spatial coherence of instances in each node. Tables 11 and 12 present results of our improved version of SMOTE that uses clustering-based partitioning of each class within each computing node. This way we ensure that new instances are created within spatially coherent subconcepts of each class and adapt to new distributions of classes in each Spark node. Our experiments conform that this approach leads to significant improvements to SMOTE algorithm, allowing it to outperform both random under- and oversampling. Furthermore, when combined with instance-level difficulty information, this variation of SMOTE is single best sampling method for multi-class imbalanced big data on Spark.

Table 11: Random Forest classification results using nine multi-class metrics.

Dataset	None	Undersample	Undersample+	Oversample	Oversample+	SMOTE	SMOTE+	SMOTE Clusters	SMOTE Clusters+
Random Forest - AvAcc									
covtype	62.95	65.11	68.99	68.67	69.51	66.99	**70.12**	66.86	68.25
traffic	80.34	80.00	80.72	80.05	80.60	80.35	80.71	80.50	**81.06**
seer	92.46	91.09	92.55	91.08	92.33	92.41	92.49	92.52	**92.83**
sensors	94.90	94.92	94.94	94.93	**94.98**	94.93	94.97	94.92	94.97
iot	98.52	99.24	99.31	98.38	99.21	98.01	98.34	**99.42**	99.41
Random Forest - RecM									
covtype	7.53	17.55	21.88	16.63	**21.93**	14.36	19.67	14.92	19.48
traffic	31.66	31.20	32.33	31.58	**33.09**	26.96	28.57	29.24	30.20
seer	57.52	53.41	56.67	53.40	**59.32**	54.59	56.83	55.31	58.41
sensors	1.03	1.81	2.62	1.55	**2.80**	1.48	2.71	1.79	2.45
iot	94.83	94.56	95.40	91.98	94.79	91.48	92.83	95.73	**96.36**
Random Forest - Recu									
covtype	0.00	0.00	0.00	0.00	0.00	8.73	**10.61**	8.27	9.23
traffic	21.92	30.47	**32.44**	30.56	32.12	25.55	27.76	28.68	30.41
seer	42.66	51.91	52.87	51.85	52.80	45.43	47.58	52.87	**53.73**
sensors	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
iot	88.79	95.17	95.97	90.27	95.00	87.79	89.82	**96.11**	96.09
Random Forest - PrecM									
covtype	24.47	10.08	16.20	9.44	14.81	23.34	**27.72**	19.15	22.77
traffic	31.18	30.01	32.54	30.18	32.10	31.24	32.49	31.74	**33.72**
seer	62.32	55.44	62.76	55.39	61.67	62.03	62.45	62.62	**64.14**
sensors	1.79	2.62	2.57	2.74	2.84	2.33	2.56	2.62	**3.11**
iot	91.84	95.83	96.20	91.09	95.65	89.08	90.87	**96.82**	96.77
Random Forest - Precu									
covtype	**18.09**	6.47	12.50	6.39	12.33	15.68	17.67	11.60	15.28
traffic	31.18	30.01	32.54	30.18	32.10	31.24	32.49	31.74	**33.72**
seer	62.32	55.44	62.76	55.39	61.67	62.03	62.45	62.62	**64.14**
sensors	1.79	2.62	2.55	2.75	2.76	2.31	2.57	2.63	**3.12**
iot	91.84	95.83	96.20	91.09	95.65	89.08	90.87	**96.82**	96.77
Random Forest - FbM									
covtype	2.25	3.62	6.23	4.11	5.75	9.46	**11.27**	9.08	10.00
traffic	23.35	30.61	**32.41**	30.76	32.17	25.82	27.91	28.70	30.09
seer	44.99	52.20	53.29	52.15	53.04	47.00	48.53	53.15	**54.24**
sensors	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
iot	89.94	95.05	95.86	90.60	94.93	88.51	89.93	96.03	**96.07**
Random Forest - Fbu									
covtype	18.90	6.95	12.83	6.75	12.72	16.77	**18.94**	12.59	16.33
traffic	31.18	30.01	32.54	30.18	32.10	31.24	32.49	31.74	**33.72**
seer	62.32	55.44	62.76	55.39	61.67	62.03	62.45	62.62	**64.14**
sensors	1.79	2.62	2.56	2.75	2.78	2.32	2.57	2.63	**3.12**
iot	91.84	95.83	96.20	91.09	95.65	89.08	90.87	**96.82**	96.77
Random Forest - AvFb									
covtype	5.38	6.18	9.37	6.87	9.73	8.78	**10.79**	8.67	9.41
traffic	19.76	28.87	**31.28**	29.00	30.76	24.95	26.91	28.02	29.44
seer	41.96	49.35	52.30	49.28	50.74	45.12	47.05	51.42	**52.58**
sensors	1.13	1.65	1.74	1.65	1.69	1.55	**1.90**	1.63	1.80
iot	89.13	94.82	95.65	89.71	94.81	87.38	89.58	95.95	**95.98**
Random Forest - CBA									
covtype	3.33	5.56	7.60	6.24	7.93	7.23	**9.55**	7.76	8.42
traffic	13.86	22.86	26.63	22.74	**26.69**	20.52	23.18	23.43	26.16
seer	34.39	41.37	**47.60**	41.31	45.92	38.54	41.67	44.38	45.67
sensors	0.78	1.09	1.09	1.00	0.99	1.00	**1.41**	0.82	1.09
iot	84.31	91.12	91.90	84.02	91.81	80.64	86.65	93.54	**94.19**

Table 12: Naive Bayes classification results using nine multi-class metrics.

Dataset	None	Undersample	Undersample+	Oversample	Oversample+	SMOTE	SMOTE+	SMOTE Clusters	SMOTE Clusters+
Naive Bayes - AvAcc									
covtype	**69.08**	64.77	65.15	64.92	65.45	64.94	65.45	65.11	65.29
traffic	**78.97**	75.24	75.48	75.32	75.60	75.39	75.67	77.54	78.00
seer	**91.04**	89.16	89.57	89.22	89.57	89.23	89.58	89.78	89.84
sensors	94.88	94.88	94.90	94.89	94.92	94.88	94.91	94.91	**95.16**
iot	**92.93**	91.95	92.03	92.06	92.11	92.09	92.09	92.33	92.34
Naive Bayes - RecM									
covtype	**3.16**	1.52	2.19	1.52	2.27	1.52	2.46	1.79	2.23
traffic	**27.23**	23.55	23.07	23.03	23.44	23.04	23.33	22.97	23.45
seer	39.65	45.17	45.52	45.33	45.59	45.36	45.56	45.63	**46.01**
sensors	0.04	0.58	0.91	1.33	1.54	1.39	2.51	2.69	**3.71**
iot	61.44	60.13	60.15	60.20	60.21	60.21	60.20	60.49	**61.73**
Naive Bayes - Recu									
covtype	1.65	2.02	**5.92**	2.07	4.74	2.09	4.73	2.60	3.06
traffic	14.79	19.36	19.58	19.32	19.46	19.36	19.43	21.57	**22.03**
seer	32.37	39.52	39.58	39.59	39.78	39.62	39.74	40.17	**40.41**
sensors	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
iot	58.47	62.38	62.61	62.68	**62.82**	62.77	62.77	60.72	60.76
Naive Bayes - PrecM									
covtype	1.05	1.67	3.21	1.76	2.53	1.77	2.81	2.27	**3.35**
traffic	**26.40**	13.34	14.17	13.61	14.59	13.87	14.86	21.39	23.01
seer	**55.22**	45.80	47.84	46.11	47.86	46.16	47.92	48.88	49.20
sensors	1.46	1.46	2.02	1.84	2.05	1.55	1.74	2.20	**2.39**
iot	**61.11**	55.73	56.19	56.31	56.60	56.50	56.50	57.83	57.86
Naive Bayes - Precu									
covtype	0.22	0.92	2.78	0.95	2.13	0.95	2.40	1.33	**2.88**
traffic	**26.40**	13.34	14.17	13.61	14.59	13.87	14.86	21.39	23.01
seer	**55.22**	45.80	47.84	46.11	47.86	46.16	47.92	48.88	49.20
sensors	1.46	1.47	2.02	1.84	2.05	1.56	1.74	2.21	**2.39**
iot	**61.11**	55.73	56.19	56.31	56.60	56.50	56.50	57.83	57.86
Naive Bayes - FbM									
covtype	1.76	1.82	**4.22**	1.85	3.41	1.88	3.53	2.31	2.60
traffic	16.27	20.07	20.09	19.96	19.95	20.00	19.91	21.83	**22.10**
seer	33.61	40.53	40.56	40.62	40.82	40.65	40.78	41.16	**41.37**
sensors	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
iot	59.04	61.92	62.10	62.17	**62.27**	62.24	62.24	59.72	59.79
Naive Bayes - Fbu									
covtype	0.26	1.01	2.85	1.04	2.20	1.04	2.48	1.45	**2.93**
traffic	**26.40**	13.34	14.17	13.61	14.59	13.87	14.86	21.39	23.01
seer	**55.22**	45.80	47.84	46.11	47.86	46.16	47.92	48.88	49.20
sensors	1.46	1.47	2.02	1.84	2.05	1.55	1.74	2.21	**2.39**
iot	**61.11**	55.73	56.19	56.31	56.60	56.50	56.50	57.83	57.86
Naive Bayes - AvFb									
covtype	1.58	1.12	**2.57**	1.15	2.02	1.16	2.29	1.43	1.97
traffic	10.47	12.32	13.24	12.44	13.17	12.62	13.23	19.30	**20.51**
seer	30.81	37.83	38.52	37.93	38.55	37.98	38.52	38.75	**39.22**
sensors	0.17	0.68	0.89	0.89	1.16	0.82	1.07	1.09	**1.36**
iot	54.40	57.32	57.66	57.73	**57.94**	57.86	57.87	56.27	56.32
Naive Bayes - CBA									
covtype	**1.41**	0.56	1.08	0.57	1.03	0.58	1.17	0.71	1.07
traffic	5.05	9.87	10.67	10.02	10.40	10.23	10.46	15.69	**17.31**
seer	23.44	31.85	33.80	32.02	33.72	32.06	33.65	33.50	**34.35**
sensors	0.04	0.35	0.44	0.49	0.61	0.52	0.65	0.66	**0.82**
iot	42.25	43.60	44.04	44.09	44.37	44.26	44.28	45.03	**45.08**

,,

’’’’

Figure 18: Relationship between various combinations of class instance types and the performance of sampling algorithms working on these combinations. Random Forest used as a base classifiers and results presented with the respect to CBA [%] metric.

Figure 19: Relationship between various combinations of class instance types and the performance of sampling algorithms working on these combinations. Naive Bayes used as a base classifiers and results presented with the respect to CBA [%] metric.

5.5 Conclusion

Multi-class imbalance presents significant challenges extending beyond problems encountered in two-class skewed problems. When further combined with a massive volume of data, we encounter a highly demanding area that has not yet been properly addressed by the research community. In this paper we have proposed the first comprehensive framework for learning from multi-class imbalanced big data on Spark. Not only we proposed efficient implementations of multi-class sampling methods (SMOTE, under- and oversampling) in Scala language, but further augmented them with two important extension: (i) informative sampling; and (ii) novel partitioning for SMOTE in Spark nodes. The former included a thorough analysis of

instance-level difficulties and using a taxonomy of four difficulty types to label instances in each class. This was used to perform a selective sampling that focused on chosen types of instances, enhancing their presence in balanced datasets. The latter proposed a clustering-based data partitioning in each Spark node that alleviated the problem of a lack of spatial coherence among instances from each class due to random data splitting among nodes. This way we were able to leverage SMOTE in Spark, reducing its chances to create erroneous artificial instances.

Extensive experimental study showed that using information about instance-level difficulties leads to a significant improvement for all considered sampling methods, as they can focus on enhancing the presence of most relevant instances for each class. The proposed clustering-based improvement to SMOTE also led to significant gains in predictive power, making it more suitable for distributed environments.

In our future works, we plan to investigate meta-learning approaches for automatic selection of which combinations of instance types should be chosen for sampling for each class.

CHAPTER 6

BAGGING USING INSTANCE-LEVEL DIFFICULTY FOR MULTI-CLASS IMBALANCED BIG DATA CLASSIFICATION ON SPARK

Most machine learning methods work under the assumption that classes have a roughly balanced number of instances. However, in many real-life problems we may have some types of instances appearing predominantly more frequently than the others which causes a bias towards the majority class during classifier training. This becomes even more challenging when dealing with multiple classes, where relationships between them are not easily defined. Learning from multi-class imbalanced data has not been widely considered in the context of big data mining, despite the fact that this is a learning difficulty frequently appearing in this domain. In this paper, we address this challenge by proposing a comprehensive ensemble-based framework. We propose to analyze each class to extract instance-level characteristics describing their difficulty levels. We embed this information into the existing UnderBagging framework. Our ensemble samples instances with probabilities proportional to their difficulty levels. This allows us to focus the learning process on the most difficult instances, better capturing the properties of multi-class imbalanced problems. We implemented our framework on Apache Spark to allow for high-performance computing over big data sets. This experimental study shows that taking into account the instance-level difficulty leads to training of significantly more accurate ensembles.

6.1 Bagging with Instance-level Difficulty

In this section, we will present the details of our proposed ensemble framework for multi-class imbalanced big data classification. In the following sections, we will describe details on how to combine ensemble learning with information about instance-level difficulty to form our proposed algorithm UnderBagging+, as well as how to implement our framework on Apache Spark for high-performance computing.

6.1.1 Combining UnderBagging with Instance-level Difficulty

Bagging-based algorithms are very popular in imbalanced domains. Many studies show that they are especially efficient when combined with undersampling in each bag, for both two-class [133] and multi-class [36] problems. However, all existing algorithms based on the idea of UnderBagging assume that all instances in each class are equally important. We propose to include information about instance-level characteristics into the undersampling process conducted in each bag. So in order to do so, we need a way of calculating the difficulty of each instance. This is usually done by analyzing the instance neighborhood and assigning a given level based on its uniformness [134]. The more contaminated the neighborhood with instances from other classes, the higher difficulty level is assigned to analyzed instance. One of four labels (safe / borderline / rare / outlier) is selected and this information is further used by sampling or the learning algorithm.

Calculating instance-level difficulty. Previous works used a discrete approach by either selecting or removing all instances with a given difficulty label from the training set. This required a time-consuming manual tuning of all possible combinations of instance types. In this work, we propose to use a probabilistic approach, where the difficulty of each instance will affect the chances of it being selected for a given bag. In

order to do so, let us define the function for calculating the difficulty of each instance based on the number of neighbors from the same class:

$$ID(x,k) = \frac{\sum_{i=1}^{k} x_i \in \text{kNN}(x) \wedge \text{label}(x_i) \neq \text{label}(x)}{k}, \tag{6.1}$$

where k is the number of neighbors used for the difficulty analysis and kNN stands for k nearest neighbors. This function takes values in $[0,1]$ and assigns higher values to instances that are in a more contaminated neighborhood (i.e., higher number of neighbors originating from another classes).

Modification of instance selection probability for UnderBagging. Having established a way to calculate the instance difficulty, we must utilize it in a form of a selection likelihood during the undersampling in each bag. As we deal with a multi-class problem, we must calculate the likelihood of each m-th class independently, in order to conduct our proposed instance difficulty-based undersampling on each of them. The higher the difficulty of a given instance, the more valuable it is to a classifier, as it represents a region where bias towards the majority class may occur. Therefore, we want to enhance the importance of difficult instances during the training phase. We achieve this by having a higher likelihood of difficult instances to be selected in each bag:

$$f_m(x) = \frac{1}{n_m} + ID(x,k), \tag{6.2}$$

where n_m is the number of instances in m-th class. Using this approach, we increase the likeliness of preserving difficult instances in each bag. At the same time, we give chance for safe instances to be selected, achieving a mix of different types of instances. This should lead to the creation of a more diverse set of classifiers as compared to previous works that suggested to discard the entire set of a given instance type.

The likelihood is then transformed into a normalized probability function for selecting a given instance in m-th class:

$$p_m(x) = \frac{f_m(x)}{\sum_{i=1}^{n_m} f_m(x)}. \tag{6.3}$$

This normalization ensures that the proposed probability function p_m is a proper probabilistic distribution. This is necessary for using it during Bagging instance selection when choosing with replacement.

Other characteristics of UnderBagging+. The proposed model differs from existing Bagging-based solutions for imbalanced data by modifying its sampling probability according to instance difficulty level and by working directly on multi-class problems. The size of each bag is defined as a user-based parameter, as there exists two approaches to this issue: (i) selecting the bag size equal to the size of the class with lowest number of instances; and (ii) selecting the bag size smaller than the size of the class with lowest number of instances. As for the prediction phase, UnderBagging+ uses a standard majority voting scheme.

6.1.2 Implementation on Apache Spark

UnderBagging+ was developed to use in-depth information about instances in each class. In order to make it applicable to massive imbalanced datasets, we propose an efficient implementation in the Apache Spark environment.

Undersampling on Spark. The proposed modified undersampling that takes into account the instance-level difficulty is the backbone of our system. It will be executed by each worker on each node multiple times to form base classifiers for the UnderBagging+ ensemble. Additionally, it must work on each class independently and take as an input parameter the size of each class after undersampling procedure.

Algorithm 20 Perform target class undersampling

procedure: TargetClassUndersample($DF, NumOfClasses, TargetCount$)
$sampledDataFrames \leftarrow DataFrame[0 \text{ to } c]$
for $c = 1$ to $NumberOfClasses$ **do**
  $DF_c \leftarrow DF.filter(x \rightarrow x.class == c)$
  if $DF_c.count() > TargetCount$ **then**
    $sampledDataFrames_c \leftarrow DF_c.sample(TargetCount)$
  else
    $sampledDataFrames_c \leftarrow DF_c$
  end if
end for
return $sampledDataFrames.union()$

We have written an efficient undersampling code in Scala that takes advantage of the Spark architecture, in order to offer scalability to massive datasets. Additionally, as UnderBagging+ uses sampling with replacement, we join the selected instances into a DataFrame structure to form a balanced dataset. Details of the proposed undersampling implementation are given in a pseudocode form in Algorithm 1.

UnderBagging+ on Spark. A separate UnderBagging+ ensemble is constructed in each node of the Spark cluster and then intermediate results from each ensemble are combined during reduce phase to give a final model. Each node will already obtain a subset of data from a given map. Therefore, the calculation of instance difficulties according to Eq. 6.1 is done in each node independently, in order to capture local data characteristics. This leads to the creation of locally specialized classifiers in each node and benefits the overall diversity of the proposed UnderBagging+ architecture. As the instance difficulty calculation requires an analysis of the neighborhood of each instance, we use an efficient k-NN implementation from *spark-knn* package.

6.2 Experimental Study

In order to evaluate the effectiveness of UnderBagging+, we have designed an in-depth experimental study that aims at answering the following research questions:

- **RQ1:** Does UnderBagging+ display better performance than existing methods for multi-class imbalanced big data available in Spark?
- **RQ2:** Does modification of sampling used in UnderBagging+ lead to improved results over standard Bagging?
- **RQ3:** How does the improved sampling method influence the size and diversity of UnderBagging+?

In the following section, we will present the experimental set-up, obtained results, as well as discuss them.

6.2.1 Data Benchmarks

Learning from imbalanced big data suffers from the lack of proper benchmark repositories. While there are a couple of popularly used large-scale two-class imbalanced datasets [135], multi-class imbalanced big datasets are still not standardized or widely available. For the purpose of this study we have selected and prepared five real-world datasets that have the following important properties: (i) are originally multi-class; (ii) have a large volume; and (iii) are characterized by a significant imbalance among classes and various data-level difficulties. Their details are given in Table 10. Please note that we have calculated the ratios of different types of instances in each class using all data. However, during actual experiments the instance-level difficulty is calculated only using the training data and independently in each computational node of the Spark cluster. Therefore, we report global properties of each benchmark, but runtime local characteristics may differ.

6.2.2 Set-up

In this section we give details regarding the experimental study set-up and parameters.

Reference methods. As there is practically no work done on multi-class imbalanced data classification with Spark (see [27]), we adapted existing methods dedicated to two-class problems. We compare UnderBagging+ with the standard version of UnderBagging, as well as Random Forest with both under- and oversampling, adapted from [84]. Finally, we use standard CART tree with both under- and oversampling as a counterpart to ensemble methods.

Adaptation to multi-class problems. As mentioned above, there are no available methods for learning from multi-class imbalanced data on Spark. However, both Random Forest and CART are capable of handling multi-class problems. We extended them with our multi-class adaptations of under- and oversampling, where each class is treated individually and sampled to a given size.

Parameters. We perform parameter selection for each classifier using internal 3-fold cross-validation on training data. Each ensemble uses CART as a base classifier. Each ensemble size is chosen from $\{20, 40, \cdots, 200\}$. Oversampling ratios are in $\{0.5n_{max}, n_{max}, 1.5n_{max}, 2n_{max}\}$, where n_{max} is the size of the biggest class in the analyzed dataset. Undersampling ratios are in $\{1.5n_{min}, 1.5n_{min}, n_{min}, 0.5n_{min}, 0.25n_{min}\}$, where n_{min} is the size of the smallest class.

Training and testing. We have used a stratified 10-fold cross validation in order to maintain the original class distribution among the folds.

Evaluation metrics. In order to properly evaluate the performance of classifiers on imbalanced domains, nine skew-insensitive metrics were used which can be found in Section 5.4.2.

Additionally, we use an Entropy measure for diversity analysis among ensemble members.

Statistical analysis. We analyze the significance of obtained results using Bayesian sign-rank test [136] with its visualization capabilities for pairwise comparison of classifiers.

High-performance computing environment. All experiments were performed in Spark environment with 32 computing nodes provided by Amazon Web Services.

6.2.3 Results and Discussion

Results of our computational experiments with respect to nine selected skew-insensitive performance metrics are given in Table 13, while the outcomes of Bayesian sign-rank test of statistical significance are depicted in Figure 20. We will discuss the obtained results from the perspective of the three research questions stated at the beginning of this section.

Comparison with reference classifiers. Let us firstly compare UnderBagging+ with the single CART classifier. Unsurprisingly, UnderBagging+ always achieves significantly better results, as further verified by Bayesian statistical tests. Regardless of the used type of sampling, CART always offered significantly lower predictive power. This verifies a natural assumption that multi-class big imbalanced data are too complex to be effectively modelled by a single classifier, even when augmented with data-level balancing. Therefore, ensembles are shown to be the most promising direction for future research on big imbalanced data. Random Forest offers much more interesting observations. When combined with sampling for each base tree, Random Forest is considered as the best performing ensemble for imbalanced big data. Standard UnderBagging performs significantly below both versions of Random Forest. However, the proposed UnderBagging+ is capable of outperforming both versions of

Table 13: Results for UnderBagging+ and reference methods according to nine performance metrics on five multi-class big imbalanced datasets.

Dataset	UnBag+	UnBag	RandF$_{un}$	RandF$_{ov}$	CART$_{un}$	CART$_{ov}$
			AvAcc			
covtype	**71.18**	61.45	65.11	68.67	52.10	51.04
traffic	**84.33**	78.39	80.00	80.05	71.03	70.13
seer	**92.18**	90.03	91.09	91.08	82.78	81.99
sensors	**95.72**	94.01	94.92	94.93	86.44	86.52
iot	**99.66**	98.38	99.24	98.38	95.36	95.12
			RecM			
covtype	**33.12**	14.87	17.55	16.63	10.34	10.07
traffic	**36.43**	30.11	31.20	31.58	26.56	26.01
seer	**55.88**	53.28	53.41	53.40	50.02	49.84
sensors	1.75	1.17	**1.81**	1.55	0.87	0.82
iot	**96.88**	91.48	94.56	91.98	88.27	88.44
			RecU			
covtype	**5.46**	3.18	0.00	0.00	0.00	0.00
traffic	**35.18**	28.42	30.47	30.56	21.67	22.15
seer	51.82	49.75	**51.91**	51.85	43.19	44.01
sensors	**6.78**	4.11	0.00	0.00	0.00	0.00
iot	**96.82**	93.88	95.17	90.27	87.56	82.99
			PrecM			
covtype	**17.99**	12.74	10.08	9.44	7.52	7.33
traffic	**36.19**	29.87	30.01	30.18	25.04	25.17
seer	**60.07**	53.84	55.44	55.39	51.99	50.79
sensors	**12.99**	8.99	2.62	2.74	1.17	1.43
iot	**96.77**	94.02	95.83	91.09	88.32	87.61
			PrecU			
covtype	**12.97**	8.05	6.47	6.39	4.99	5.28
traffic	**37.92**	34.09	30.01	30.18	26.62	27.01
seer	55.10	51.92	**55.44**	55.39	47.81	48.04
sensors	**13.01**	7.28	2.62	2.55	0.91	0.93
iot	**95.83**	92.48	**95.83**	91.09	87.71	81.98
			FbM			
covtype	**18.37**	11.94	3.62	4.11	2.07	2.11
traffic	**35.61**	29.99	30.61	30.76	27.82	28.03
seer	**54.82**	51.04	52.50	52.15	47.86	46.94
sensors	**6.07**	3.98	0.00	0.00	0.00	0.00
iot	**96.11**	94.86	95.05	90.60	87.19	83.48
			FbU			
covtype	**24.81**	19.63	6.95	6.75	2.98	2.88
traffic	**36.17**	32.10	30.01	30.18	23.98	24.83
seer	**57.98**	54.19	55.44	55.39	49.82	49.75
sensors	**4.82**	2.97	2.62	2.75	0.99	1.01
iot	**97.82**	93.99	95.83	91.09	88.49	88.72
			AvFb			
covtype	**20.76**	17.84	6.18	6.87	3.98	4.04
traffic	**34.17**	30.82	28.87	29.00	24.27	23.99
seer	**51.88**	50.37	49.35	49.28	44.88	44.64
sensors	**5.15**	2.99	1.65	1.65	0.63	0.64
iot	**95.69**	92.88	94.82	89.71	84.56	84.60
			CBA			
covtype	**21.28**	15.92	5.56	6.24	3.82	3.46
traffic	**30.03**	23.88	22.86	22.74	17.23	15.62
seer	**50.99**	43.72	41.37	41.31	30.18	29.08
sensors	**12.84**	6.98	1.09	1.00	0.17	0.19
iot	**93.56**	90.36	91.12	84.02	78.23	69.62

Random Forest in a statistically significant way. Additionally, as seen in Figure 21, it is capable of doing so while maintaining a much smaller pool of classifiers. This shows the importance of incorporating instance-level information into the ensemble training procedure and how much can be gained by careful understanding of the underlying data properties in multi-class imbalanced big data.

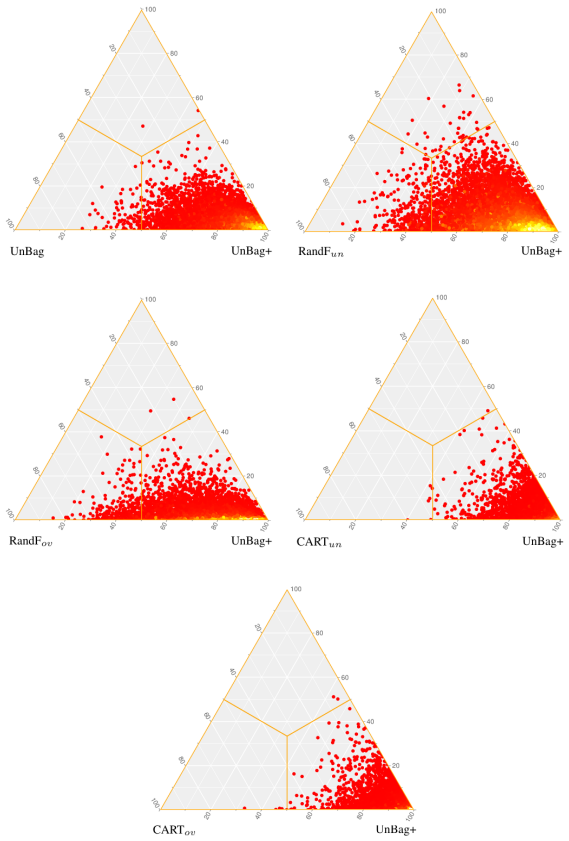


Figure 20: Visualizations of Bayesian sign-rank tests for pairwise statistical comparison between UnderBagging+ and reference methods

Role of improved undersampling with instance-level difficulty. The major difference between standard UnderBagging and the proposed UnderBagging+ lies in utilized sampling function. UnderBagging assumes a uniform probability of sampling every single instance, while UnderBagging+ correlates the probability of instance se-

lection with its difficulty level. Therefore, any differences in results between these two methods originate from this sampling part. UnderBagging+ is statistically significantly better than its counterpart, sometimes achieving gains of up to 10% on some metrics (which considering the size of datasets is a non-trivial gain). Therefore, we can see that forming bags that preserve difficult instances leads to more locally specialized classifiers and thus to improved ensemble predictive power. Additionally, as the instance difficulty calculation is done independently in each computing node of Spark, we are able to overcome the potential unfavourable distribution of instances and adapt to local data partitioning.

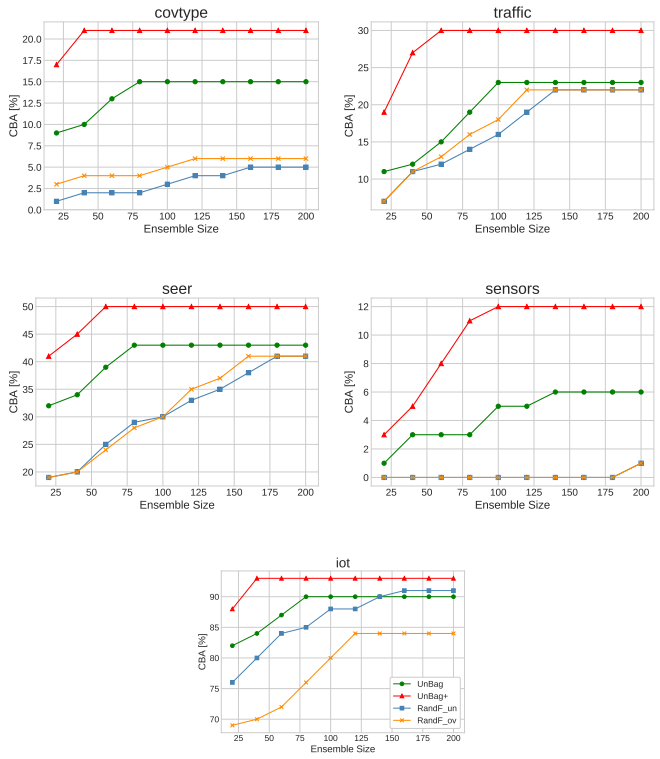


Figure 21: Relationship between ensemble size and CBA performance metric

Analysis of ensemble size and diversity of UnderBagging+. When analyzing the performance of ensemble classifiers, it is important to verify how the size

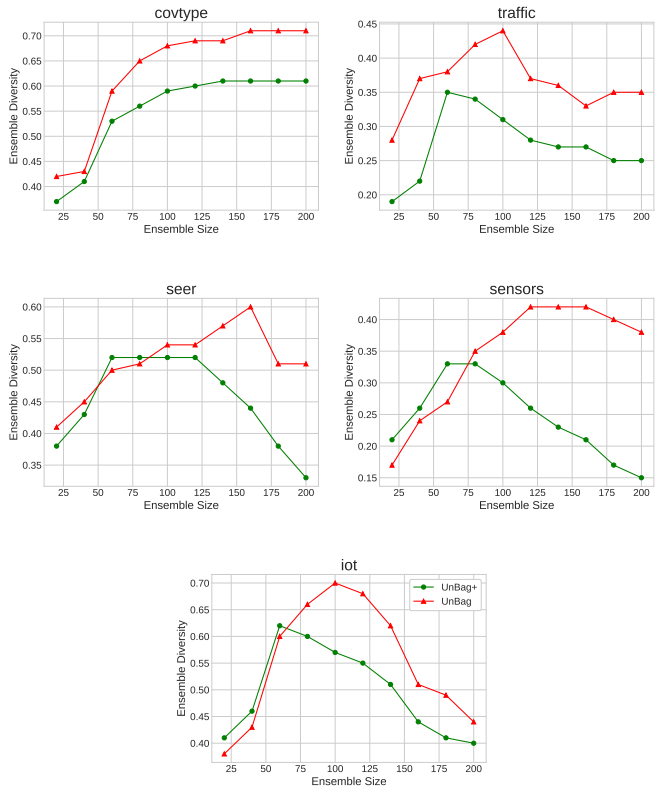


Figure 22: Analysis of ensemble diversity of UnderBagging+ with the respect to ensemble size

of the ensemble influences its predictive power and diversity. In the context of big data analytics it is desirable to create smaller ensembles that can perform similarly to their larger counterparts. Smaller ensembles are faster to compute and require less space in memory. Figure 21 shows that UnderBagging+ achieves its best performance with much smaller ensemble size than Random Forest, while displaying better performance. This can be explained by creating a more meaningful and less random bootstrap samples for base classifiers, thus being able to cover the decision space sufficiently with lower number of base models. Interestingly, diversity analysis shown in Figure 22 points to the fact that base classifiers in UnderBagging+ are less randomized. We can see a small drop in diversity compared to UnderBagging, which

can be explained by the fact that UnderBagging+ forces its base classifiers to focus on difficult instances. This reduced diversity can actually be seen as a positive factor, as it leads to more stable classifiers. Both of these factors further augment the high usability of UnderBagging+ for learning from multi-class imbalanced big data.

6.3 Conclusions and Future Works

In this paper we have addressed the contemporary issue of learning from big data under multi-class imbalance. We have discussed various challenges related with this domain and pointed to a lack of dedicated solutions for such problems. To bridge this gap, we have proposed UnderBagging+ with efficient implementation on the Apache Spark architecture that is able to learn from large-scale skewed datasets with multiple classes.

UnderBagging+ uses the idea of incorporating the instance-level difficulty into the instance selection process. For each training instance, we analyze its neighborhood. Based on the number of nearest neighbors not belonging to the same class for as given instance, we assign it a difficulty level. Then UnderBagging+ transforms this into a probability metric and uses it to guide the sampling process for creating training bags. This way difficult instances from each class are more likely to be selected, leading to focusing base classifiers on these challenging cases. Experimental study demonstrated that this approach allows for UnderBagging+ to significantly outperform reference methods, while maintaining a smaller ensemble size.

Our future works will concentrate on further improvements in the sampling procedures and adding new steps dedicated to alleviating other difficulties of multi-class imbalanced big data, such as class overlapping and extreme class imbalance.

CHAPTER 7

IMPROVED KD-TREE BASED IMBALANCED BIG DATA CLASSIFICATION AND OVERSAMPLING FOR MAPREDUCE PLATFORMS

In the era of big data, it is necessary to provide novel and efficient platforms for training machine learning models over large volumes of data. The MapReduce approach and its Apache Spark implementation are among the most popular methods that provide high-performance computing for classification algorithms. However, they require dedicated implementations that will take advantage of such architectures. Additionally, many real-world big data problems are plagued by class imbalance, posing challenges to the classifier training step. Existing solutions for alleviating skewed distributions do not work well in the MapReduce environment. In this paper, we propose a novel KD-tree based classifier, together with a variation of the SMOTE algorithm dedicated to the Spark platform. Our algorithms offer excellent predictive power and can work simultaneously with binary and multi-class imbalanced data. Exhaustive experiments conducted using the Amazon Web Service platform showcase the high efficiency and flexibility of our proposed algorithms.

7.1 Proposed Algorithms

In this section, we present two different KD-tree based implementations for the Apache Spark framework: a classifier and a novel method for addressing class imbalance.

7.1.1 KD-Tree Classifier

While the KD-tree classifier has been implemented in several other languages and frameworks [137, 138], we are unaware of any publicly available implementations in Scala for Apache Spark. Our KD-tree implementation is based on the existing Apache Spark hybrid-spill tree implementation [139] and since it fits the same machine learning pipelineing model the two methods are interchangeable. The top level metric-tree partitioning presented in the hybrd-spill tree was also used for the KD-tree to allow for a direct comparison.

The KD-tree classifier works as an approximate nearest neighbor algorithm by creating a KD-tree and performing DFS to find a leaf node. Examples present in a leaf node are tested to find the k-nearest neighbors. Like the spill-tree, a buffer region was used to help with the query examples near the median splits. A static buffer size of 25% of the node data size was chosen empirically, as it seemed to have a good balance between speed and performance while not introducing more complexity into this simple algorithm. Future work will be needed to determine if that value is universally appropriate or if there is another computationally inexpensive method for discovering an optimal value for a given dataset. Although training this KD-tree requires $\mathcal{O}(n \log^2 n)$ time compared to $\mathcal{O}(n \log n)$ for the hybrid-spill tree, its partitioning approach is independent to the number of dimensions. Like the hybrid-spill tree, querying for the nearest neighbor is also $\mathcal{O}(m \log n)$.

Algorithm 21 shows the pseudo code for the Apache Spark implementation for the KD-tree training phase. As with the hybrid-tree method, the query is performed by iterating through the trained tree. However, instead of a pivot point, the branch selection is based on the median value of the axis specified for that level in the tree. Once a leaf node is reached, all of the present examples are examined to find the

k-nearest neighbors using the brute force approach; majority voting is performed to pick the predicted class. To improve the run time of this algorithm, backtracking was not performed, but could be easily added if required.

Algorithm 21 Generate KD-Tree

KD-TREE Data Structure:
pivot: vector
median: Double
axis: Int
radius: Double
leftChild: Tree
rightChild: Tree

procedure: BUILD-KD-TREE($data : vector, leafSize : Int, axis : Int$)

if $data.size == 0$ **then**
return Leaf(Empty)
else if $data.size <= leafSize$ **then**
return Leaf($data$)
else
$sorted \leftarrow sortByAxis(data)$
$medIdx \leftarrow sorted.size/2$
$medExample \leftarrow sorted[medIdx]$
$radius \leftarrow max(data.map(x \rightarrow dist(x, data[medIdx]))$
$left \leftarrow data[0, medIdx * 1.25]$
$right \leftarrow data[medIdx - medIdx * 0.25, data.size]$
$axis \leftarrow axis + 1$
return KD-TREE($medExample, medIdx, axis, radius,$
end if

7.1.2 KD-Tree Based SMOTE

Challenges can arise when attempting to classify datasets that are class imbalanced. Classifiers tend to favor majority classes when presented with imbalanced data which can negatively impact performance, especially with the minority classes [140]. One solution is to balance the classes by oversampling the minority classes with methods such as random oversampling or SMOTE.

The SMOTE algorithm creates new examples for the minority classes to alleviate the class imbalance in the dataset. For each class to be oversampled, five examples

Algorithm 22 Generate SMOTE Example DataFrame

Ensure: Examples in DataFrame belong to the same class
  procedure: SMOTE($DF, ClassLabel$)
  $fvs = Array[5]$
  for $i = 1$ to 5 **do**
    $index \leftarrow Random.nextInt(DF.count)$
    $example \leftarrow DF[index]$
    $fvs[i] \leftarrow example.filter(featureVector)$
  end for
  $transposed \leftarrow fvs^{\mathrm{T}}$
  $averages \leftarrow transposed.map(rowSum/5)$
  $smoteFeatureVector \leftarrow averages^{\mathrm{T}}$
  $smoteExample \leftarrow \{smoteFeatureVector, ClassLabel\}$
  return $smoteExample$

are randomly selected and the average of their feature values are used to create a new example. These synthetic examples are added to the dataset until the given minority class has the desired size. While these examples were not part of the original dataset, the idea is that they will occupy underrepresented regions of the true class space which will then improve classifier accuracy. Algorithm 22 shows our Scala based Apache Spark implementation of SMOTE.

However, this approach may result in synthetic examples that do not well represent the true feature space for the given class. Figure 23 shows an example dataset with the classes represented by green triangles and blue circles. If the blue class needs to be oversampled with SMOTE, the traditional approach will sample from all possible examples in that class to create new synthetic examples. Shown in red crosses, we can see that the examples created by SMOTE often fall within the feature space of the wrong class and likely will negatively affect classification accuracy.

Instead of having SMOTE sample from all possible class examples, we present a method that only uses examples that are close together. This can be achieved by clustering the training dataset using the leaves of a KD-tree. In our implementation, one KD-tree is created per class and each new SMOTE example is generated with

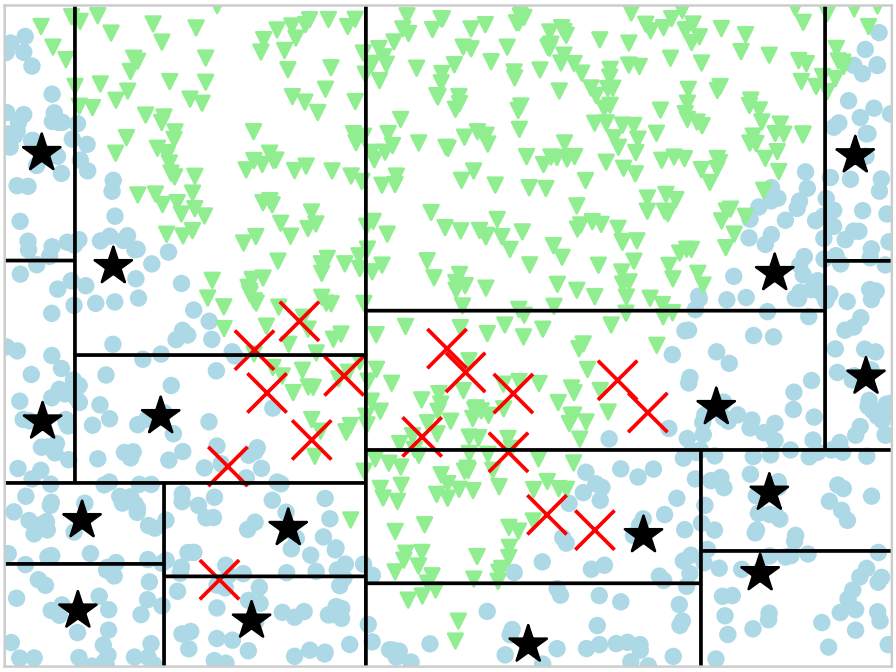


Figure 23: Example dataset showing over sampling the blue circle class with SMOTE (red crosses) and with SMOTE based on KD-trees (black stars). The KD-tree partitioning of the blue circle class is overlaid with black lines.

five random examples from a single KD-tree leaf.

Algorithm 23's KD-CLUSTER-FIT method starts by filtering the dataset by class and calculates the size of the majority class. In our experiments, the minority classes were oversampled to match the size of the majority class. For each class, the CREATE-TREE method first determines which features to use for future tree splitting. Using these selected features, the BUILD-TREE method is started but returns an array of the resulting leaves instead of a complete KD-tree. Each of these leaves represent a cluster of approximate nearest neighbors.

Features that have less than three unique values will not be used. If a splitting feature contains only one unique value, there is no way to tell where to place the

median splitting point and it would be unlikely that the partitioning would provide any information gain. While a median point could be easily found if two unique values exist for a splitting feature, the resulting branches could be highly imbalanced, which may negatively affect speed and accuracy.

After the features are selected, the specific feature to be used for each split can be adjusted. The SMOTE based class balancing did not work well in initial tests when features were used in the same order as presented in the training data. However, results improved when the feature with the smallest standard deviation at the current node was used. Future work will be required to determine what considerations are needed when addressing binary features and choosing the optimal feature at each split.

Once the training data has been clustered using the KD-tree leaves, oversampling can be performed with Algorithm 24. For each class, a corresponding leaf cluster is randomly chosen and five examples in that collection are used to generate a new SMOTE example. This process is continued until each class has as many examples as the majority class.

7.2 Experimental Study

This experimental study was designed to answer the following research questions (RQs):

- **RQ1:** Does the proposed KD-tree implementation outperform the existing state-of-the-art Hybrid-Spill Tree in both predictive power and computational efficiency?

- **RQ2:** Is the proposed SMOTE variant for the MapReduce platform capable of efficiently combating class imbalance, while avoiding the MapReduce pitfall of

Algorithm 23 Fitting KD-Tree model using SMOTE

```
procedure: KD-CLUSTER-FIT(DF)
labels ← DF.select("label").distinct()
classDFs ← labels.map(x → DF.filter"label" == x))
maxClassCount ← max(classDFs.map(x → x.size)))
return labels.map(classDF → CREATE-TREE(classDF))

procedure: CREATE-TREE(label, DF)
for index from 0 until DF.numberOfFeatures do
  if DF[index].distinct() > 2 then
      axisToUse.append(index)
  end if
end for
return BUILD-TREE(DF, splitMethod, leafSize,axisToUse, axisToUse[0])

procedure: BUILD-TREE(data, splitMethod, leafSize,
  axisToUse, currentAxis)
if data.size == 0 then
  return EMPTY
else if data.size <= leafSize then
  return data
else
  nextAxis ← min(axisToUse.map(axis → STD(data[axis]), splitMethod))
  sortedData ← data.sortedBy(nextAxis)
  leftData ← sortedData[0, sortedData.median]
  rightData ← sortedData[sortedData.median, sortedData.size]
  leftResults ← BUILD-TREE(leftData, splitMethod, leafSize,
    axisToUse, nextAxis)
  rightResults ← BUILD-TREE(rightData, splitMethod, leafSize,
    axisToUse, nextAxis)
  return leftResult + rightResults
end if
```

local data partitioning?

- **RQ3:** Is our method flexible enough to work with both binary and multi-class imbalanced problems, without a need for any changes in its structure?

7.2.1 Performance Metrics

The average accuracy (AvAcc) and class balance accuracy (CBA) metrics were used for the model accuracy metrics because most of the presented datasets are multi-class and imbalanced. As shown in Formula 7.1, the basic accuracy only considers

Algorithm 24 Oversampling with SMOTE

```
procedure: SMOTE-OVERSAMPLE(DF)
labels ← DF.select("label").distinct()
classDFs ← labels.map(x → DF.filter(label == x))
maxClassCount ← max(classDFs.map(x → x.count))
if usingStandardSMOTE then
   sampledClassDFs ← classDFs.map(classDF →
      STANDARD-SMOTE-OVERSAMPLE(classDF, maxClassCount))
else if usingKDTrees then
   classClusterArrays ← KD-CLUSTER-FIT(DF)
end if

procedure: KD-CLUSTER-SMOTE-OVERSAMPLE(classDF,
   targetSampleCount, trees)
samplesToAdd ← targetSampleCount - classDF.count()
sampledData ← trees.map(label →
   (0 to samplesToAdd).map(x →SMOTE(x.randomCluster, label))
return union(sampledData)

procedure: STANDARD-SMOTE-OVERSAMPLE(classDF,
   targetSampleCount)
if classDF.count() < targetSampleCount then
   samplesToAdd ← targetSampleCount - classDF.count()
   newSamples ← (0 to samplesToAdd)
      .map(x ← SMOTE (classDF, classDF.label))
   return union(classDF, newSamples)
else
   return classDF
end if
```

the global accuracy and may be biased towards to the majority classes. Both AvAcc and CBA take in account performance per class and introduce a higher penalty for predicting the minority class examples incorrectly.

$$
\begin{aligned}
accuracy &= \frac{tp + tn}{tp + tn + fp + fn} \\
AvAcc &= \sum_{i=1}^{C} \frac{tp_i + tn_i}{tp_i + tn_i + fp_i + fn_i} \\
CBA &= \sum_{i=1}^{C} \frac{mat_{i,i}}{max(\sum_{j=1}^{C} mat_{i,j}, \sum_{j=1}^{C} mat_{j,i})}
\end{aligned}
\tag{7.1}
$$

Table 14: Number of features, classes and maximum class balance ratio for each test dataset.

Dataset	Instances	Features	Classes	Imb. Ratio
Cover Type	581,012	54	7	104.4
Traffic Violations	1,378,663	26	7	10.5
SEER	2,532,629	11	10	5.7
Intel Sensors	2,219,803	5	58	31.2
IoT	3,000,000	115	11	4.9
SUSY	5,000,000	18	2	1.2
HIGGS	11,000,000	28	2	1.1

7.2.2 Datasets

To evaluate these algorithms, we have chosen seven datasets [141, 142, 143, 144, 145, 146] with varying properties. As shown in Table 14, there is a wide variation in the number of classes and features for each dataset allowing us to see how the proposed methods behave when presented with these combinations. Also shown is the imbalance ratio of the largest class size over the smallest.

7.2.3 Experiments on AWS

To perform our experiments, we have chosen the Amazon Web Service (AWS) platform with Elastic MapReduce (EMR). The EMR platform allows for easy deployment of cluster based applications, such as Hadoop and Spark, in a distributed environment. All experiments were performed using 1, 2, 4, and 6 c5.2xlarge instances for the computational work with each c5.2xlarge virtual instance providing 8 vCores, or threads, and 16 GB of RAM. For both classification and KD-tree based SMOTE, we have allocated 10 GB of memory to each executor and used one executor per

instance.

Our first experiments compare the run time performance and classifier accuracy of the proposed KD-tree and the existing hybrid-spill tree classifiers. For each dataset, both algorithms are run with a combination of leaf sizes (10, 100, 500, 1000, 2500) and number of computational threads (8, 16, 32, 48). The second round of experiments compares the standard SMOTE algorithm against the proposed KD-tree based SMOTE modified algorithm. These experiments were run on all datasets with same number of threads as with the classifier experiments.

7.3 Results

This section presents the results of the KD-tree classifier and SMOTE experiments.

7.3.1 Classifiers

For the classifier experiments, we compared the running time and model accuracy between the existing Hybrid-Spill tree against our proposed KD-tree implementation. Two significant parameters for both algorithms are the leaf size of the tree and the k value for the number of examples to use for class prediction. Figures 24, 25 and 26 show running times for various leaf sizes against the number of threads used and the value k=5 was used for all experiments.

In almost all presented cases, the KD-tree was faster than the Hybrid-Spill tree for both training and prediction phases. Thread level scaling was also shown to be dependent on the leaf size for both algorithms. Figure 24 shows that adding more threads consistently improves running times but this effect is diminished as the leaf size increases. Using a leaf size of 500 still shows some scaling but the improvements do not continue beyond 16 or 32 threads and performance starts to degrade when the

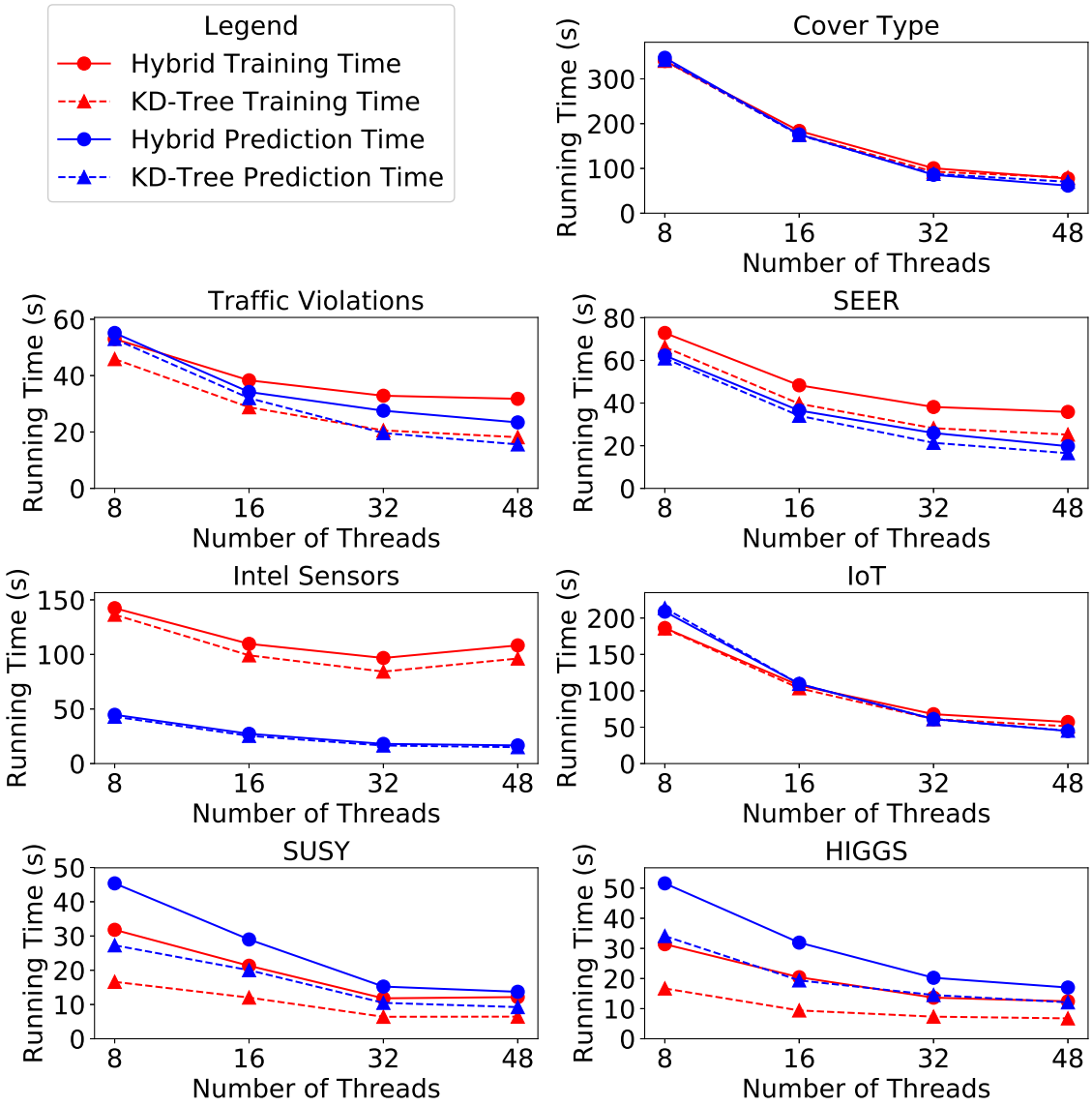


Figure 24: Classification running times for leaf size 10

leaf size is 2500.

Increasing the leaf size improves training times for both algorithms, but the KD-tree can be almost two times faster than the Hybrid-Spill tree for small leaf sizes. However, this performance difference decreases as the leaf size increases and at leaf size 2500 the times are almost identical. The running time between the two methods

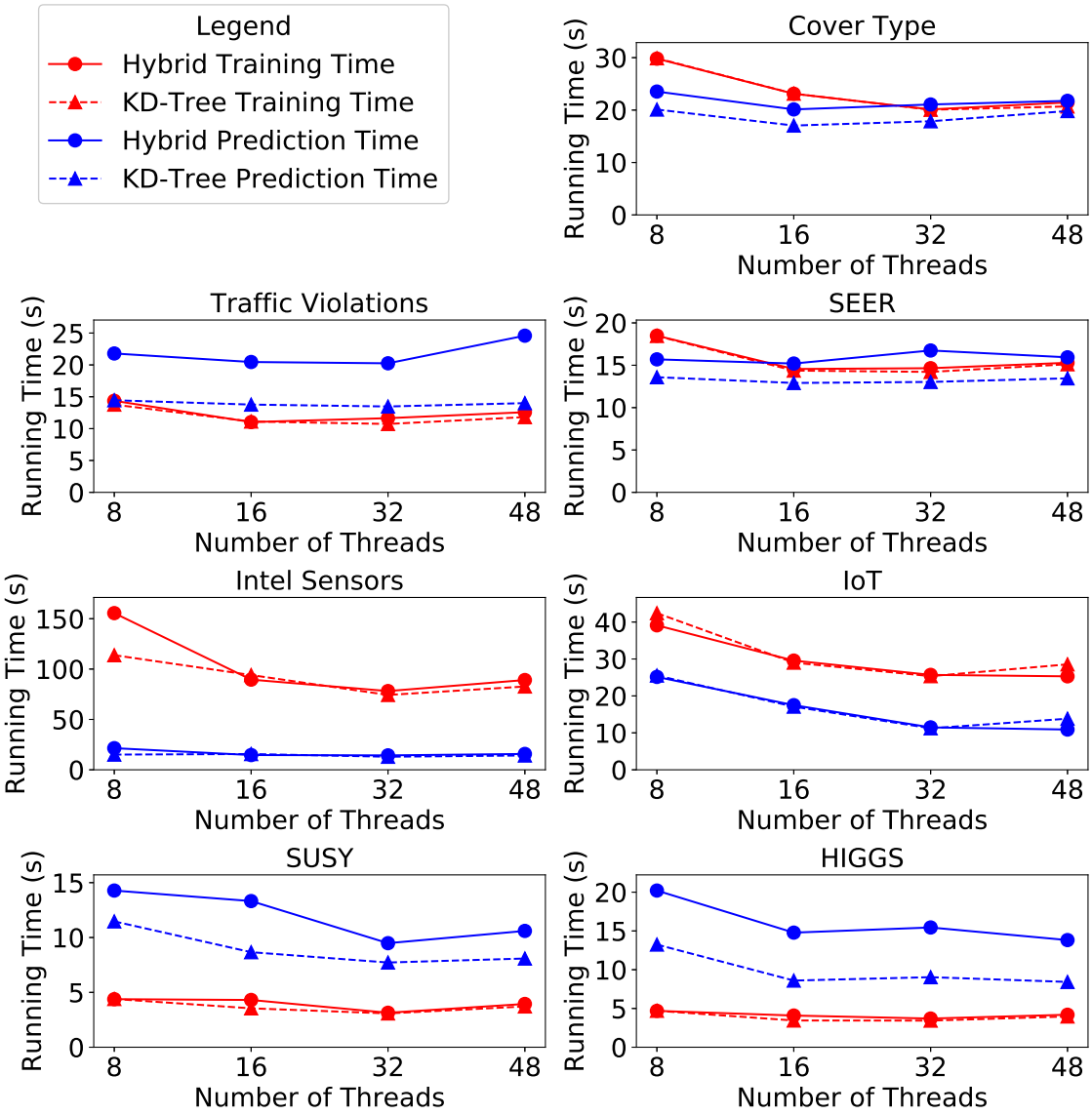


Figure 25: Classification running times for leaf size 500

is much more noticeable for making predictions. As the leaf size increases, the gap between the KD-tree and Hybrid-Spill tree performance widens with many cases where the KD-tree is two to three times faster (**RQ1** answered). This is apparent in Figure 27, where the KD-tree performance is mostly flat as leaf size increases but the Hybrid-Spill tree starts slowing down.

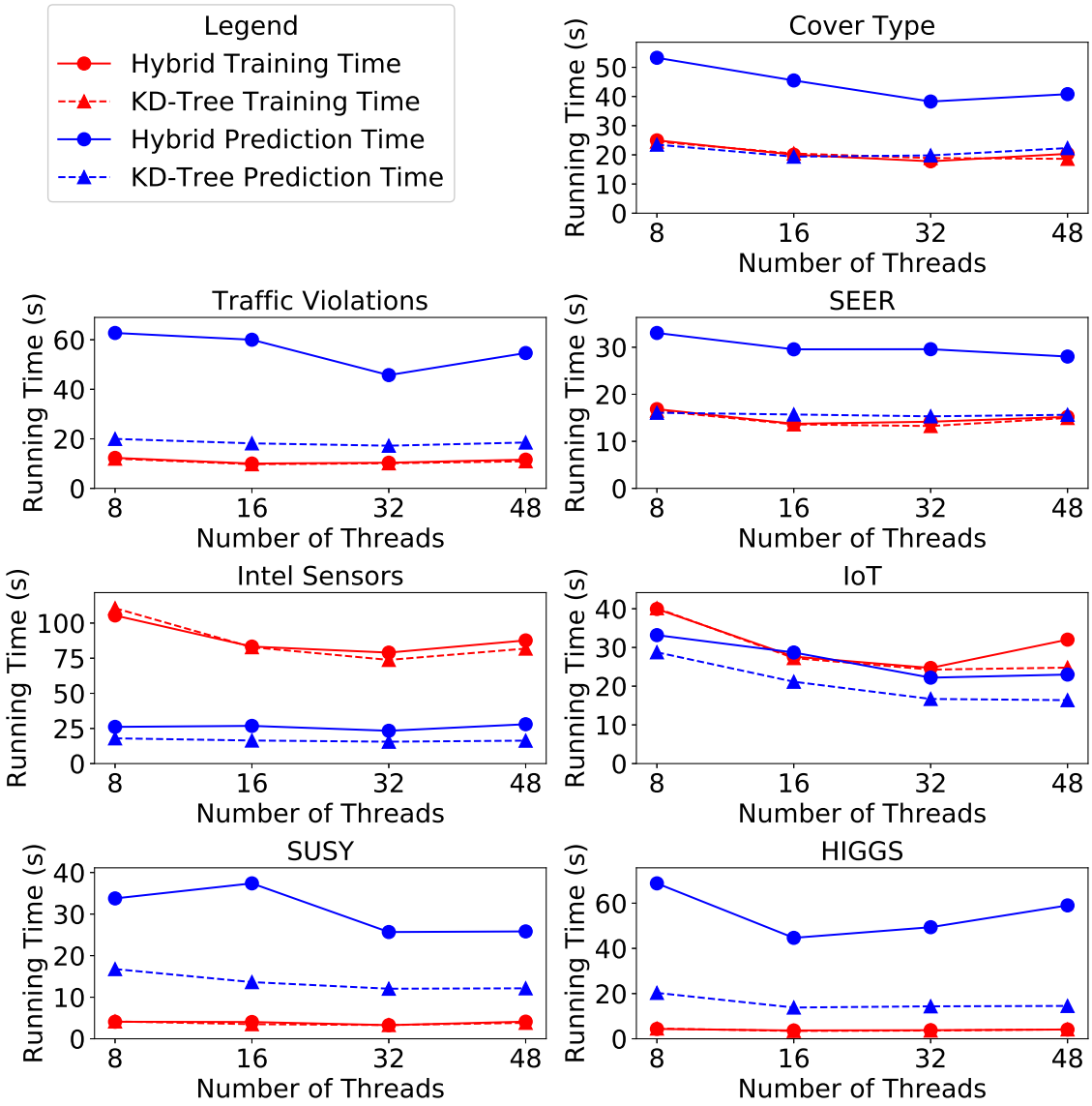


Figure 26: Classification running times for leaf size 2500

The Hybrid tree tends to be more accurate than the KD-tree for both metrics, but the AvAcc differences are minimal. However, the KD-tree performs best for two datasets including Covtype, where its CBA result is two times better than that with the Hybrid-Spill tree (**RQ1** answered).

The Hybrid-Spill tree CBA results were not largely affected by increasing leaf

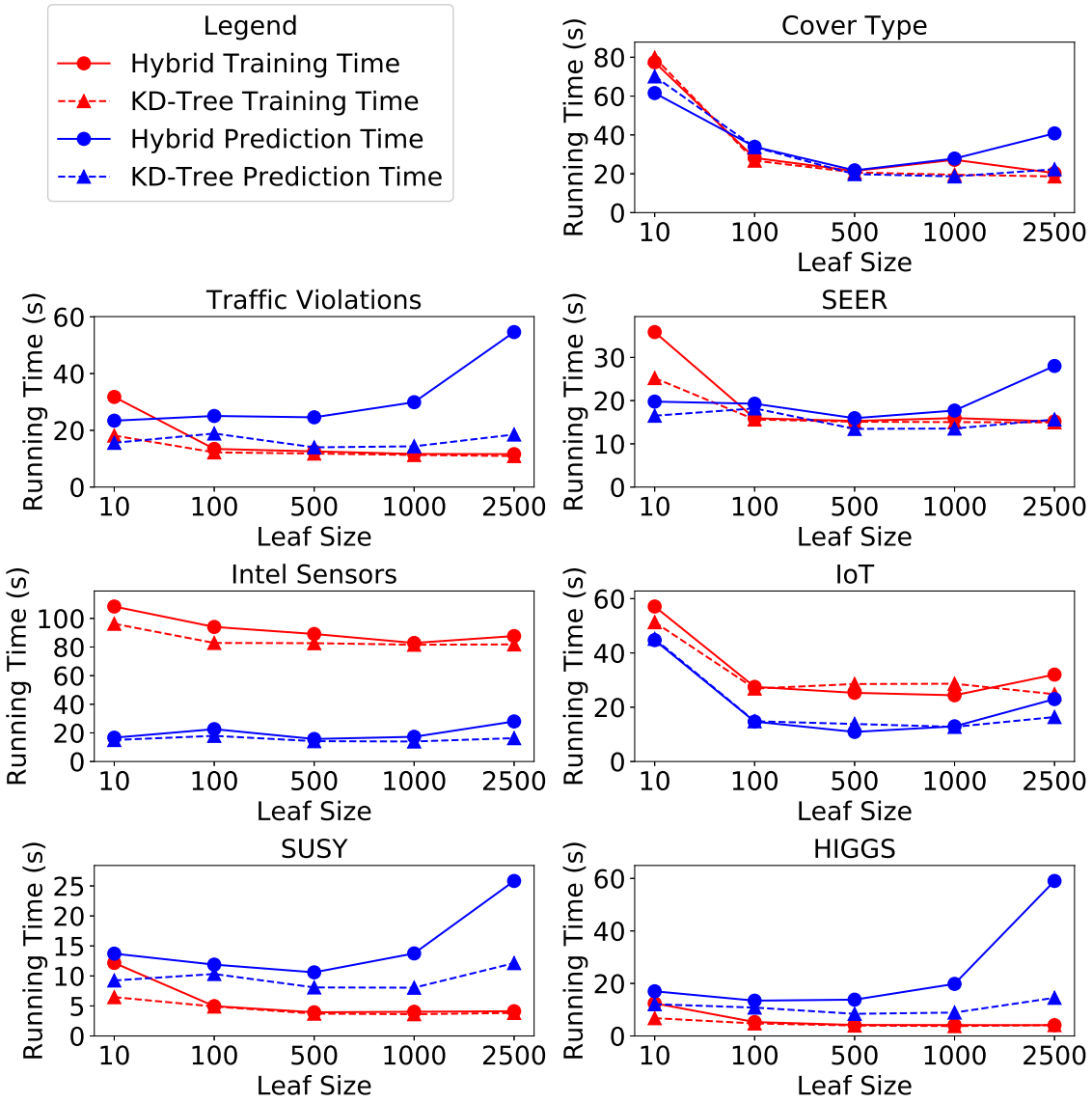


Figure 27: Classification running times with 48 threads for each leaf size

sizes, but this generally had a negative impact on the KD-tree classifier, although this is opposite for the Covtype dataset. Additionally, the AvAcc and CBA metrics were not significantly influenced by the thread count for either the KD-tree or the Hybrid-Spill tree.

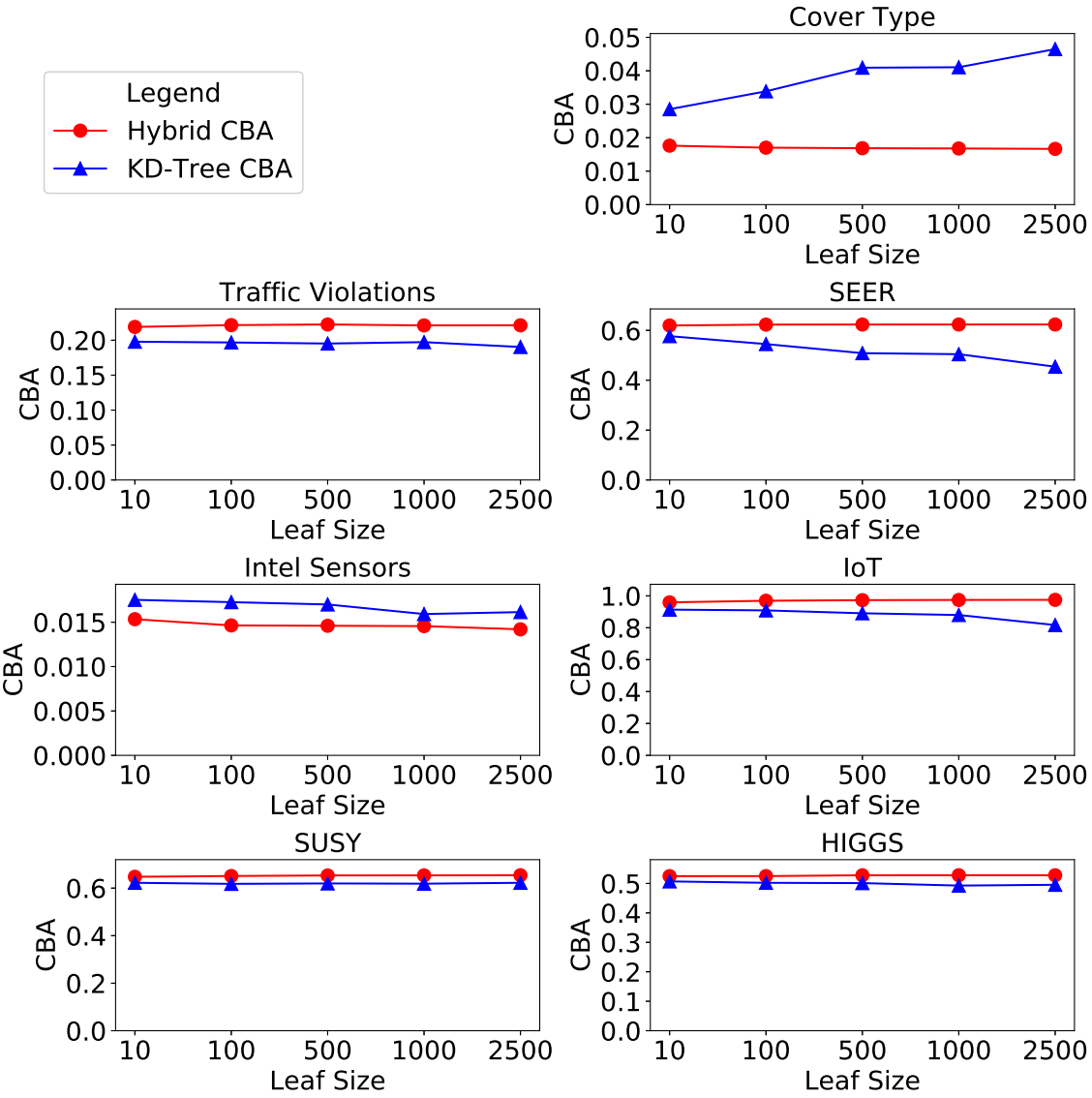


Figure 28: CBA results for each leaf size using 48 threads

7.3.2 SMOTE

To test the impact of KD-trees on SMOTE, the Spark ML Random Forest classifier was used with 5-fold cross validation and the KD-tree leaf size was set to 64 as it seemed to work well across all datasets. We observed that the class prediction accuracy largely varies depending on the individual dataset. Figure 29 shows the

Table 15: AvAcc accuracy ratio between SMOTE and SMOTE with KD-trees

Dataset	**8 Threads**	**16 Threads**	**32 Threads**	**48 Threads**	**Max Difference**	**Absolute Difference**
Cover Type	64.35	64.36	64.34	64.36	3.11	0.02
Traffic Violations	81.08	81.20	81.08	81.16	15.05	0.12
SEER	93.32	93.37	93.33	93.38	5.57	0.05
Intel Sensors	94.90	94.91	94.90	94.90	0.42	0.01
IoT	99.61	99.54	99.52	99.39	22.14	0.22
SUSY	77.60	77.56	77.66	77.59	13.67	0.11
HIGGS	66.87	67.16	67.00	67.16	44.26	0.30

accuracy ratios between standard SMOTE and SMOTE with KD-Tree example selection. Using standard SMOTE as a baseline, we can see that the tree method is best in five of the seven cases. The new method does worse on the SUSY dataset, but this only has a relative difference of 3%. The best result was with the Sensors dataset, where the KD-tree method had a 12% percent improvement on CBA accuracy. The average improvement across all seven datasets was 0.14% for AvAcc and 3.4% for CBA. While the KD-tree based method had little influence on AvAcc compared to standard SMOTE, it had a much larger effect on the CBA results. This may be because the more discerning KD-tree based oversampling method mostly benefits small and difficult to classify classes. Another observation is that the KD-tree clustering method shows the largest improvements on datasets with high class imbalance which may be the main strength of this approach (**RQ2** answered).

Tables 15 and 16 show the AvAcc and CBA values for the number of threads used. As shown in Table 15, there was very little difference for the AvAcc metric relative to the number of threads used with all maximum ranges under 0.5%. However, there was a larger range of CBA values for all datasets with the largest being over 7% for the Intel Sensors dataset. This dataset has a high number of classes, 58, which may be a contributing factor to this result. The changes in the minority class accuracy

Table 16: CBA accuracy ratio between SMOTE and SMOTE with KD-trees

Dataset	**8 Threads**	**16 Threads**	**32 Threads**	**48 Threads**	**Max Difference**	**Absolute Difference**
Cover Type	4.67	4.69	4.78	4.61	3.77	0.17
Traffic Violations	21.72	22.13	21.65	21.64	2.28	0.49
SEER	44.80	45.20	45.06	44.77	1.16	0.52
Intel Sensors	0.94	0.92	0.88	0.95	7.47	0.07
IoT	96.61	96.25	95.99	95.10	1.58	1.50
SUSY	73.46	73.48	73.48	73.38	13.90	0.10
HIGGS	64.44	65.28	64.99	65.85	2.19	1.42

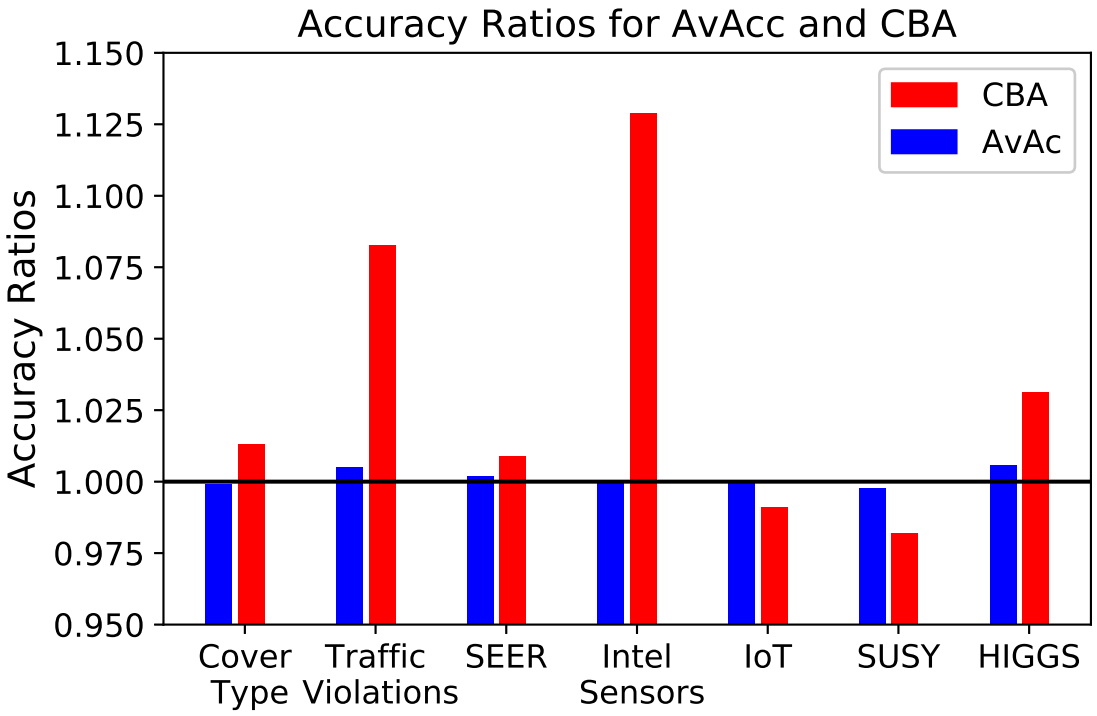


Figure 29: Accuracy results between standard SMOTE and SMOTE with KD-trees

can have a significant effect on the CBA metric such that any perturbations in the class balancing may be more pronounced (**RQ3** answered).

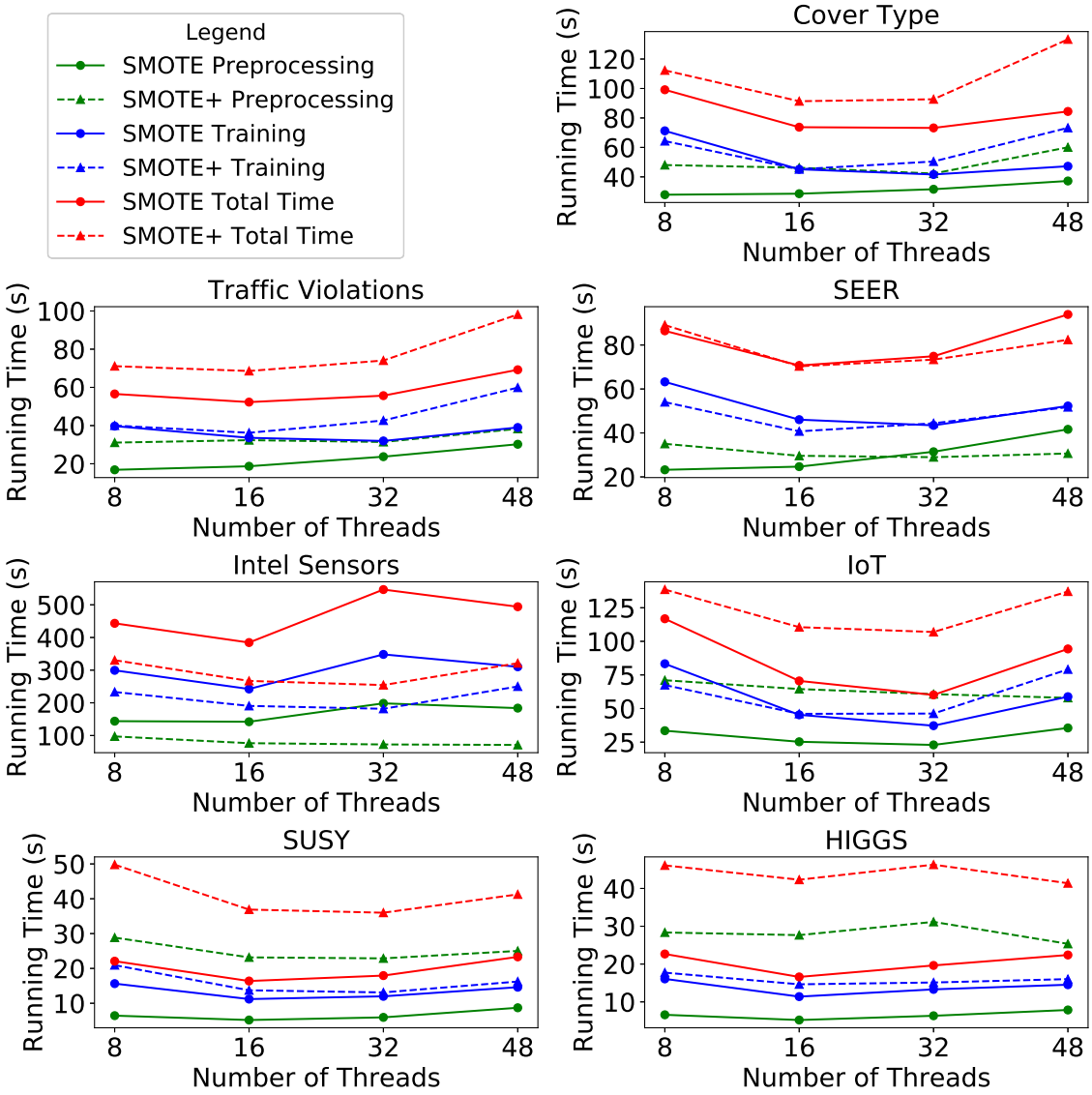


Figure 30: Running times for SMOTE and SMOTE with KD-trees

7.4 Conclusions and Future Works

We have presented a new KD-tree classifier and a novel class balancing method, both implemented in Scala for the Apache Spark framework. Our comparisons between KD and hybrid-spill tree algorithms on the AWS distributed computing platform has given some insight on the strengths and weaknesses of these two methods.

In this paper, we also demonstrated that the quality of oversampling with SMOTE can be improved using the leaves of trained KD-trees. These implementations are available on Github at https://github.com/fsleeman/spark-knn.

While the hybrid-spill classifier often provided the highest accuracy, it was slower than the KD-tree implementation. For two datasets, the KD-tree had the best classification accuracy showing that in some cases the KD-tree is both faster and more accurate. Our experiments showed that the leaf size played a significant role in running time of both methods, but was more significant for the hybrid-spill tree as the leaf sizes increased. Large leaves mean that the resulting tree will not be as tall, so more work will be required to perform nearest neighbor on the leaf examples. The hybrid-spill tree is at risk of getting penalized multiple times when backtracking on metric-trees, something that the KD-tree avoids. The number of threads did not have a significant effect on classification accuracy, but run time performance increased until approximately 16 to 32 threads and this trend was exhibited for both algorithms.

For class balancing, our KD-tree clustering method showed classifier improvements for five out of seven datasets. The cluster based oversampling method showed the biggest difference when applied to the most class imbalanced datasets, highlighting a potential application for this method. Standard SMOTE oversampling ran faster except for the *sensors* dataset which has many classes, illustrating another potential use for the KD-tree clustering method. Like with the classification experiments, run time performance increased until 16 to 32 threads. The datasets used in these experiments were larger, but the size of the generated KD-trees were proportional to the number of examples in each class, significantly smaller than the original datasets.

CHAPTER 8

A MACHINE LEARNING METHOD FOR RELABELING ARBITRARY DICOM STRUCTURE SETS TO TG-263 DEFINED LABELS

In recent years, the field of Radiation Oncology has shown an increased interest in applying Artificial Intelligence (AI) and Machine Learning (ML) methods to its domain specific problems. The inherent digital nature of radiotherapy provides an unique opportunity for investigating new methods to improve outcomes and overall patient care quality. Advances in algorithms and decreasing costs of computer hardware have now opened the door for addressing larger problems in oncology but fully utilizing these large data sets requires both data aggregation and standardization [147, 148].

An important step in the Radiation Oncology treatment planning is to delineate anatomical volumes, referred to as structures, which includes organs-at-risk (OAR) and target volumes. These structures are differentiated with a descriptive label given by the physician such as Bladder, Rectum, or Heart. Other medical domains also utilize delineated anatomical structures such as radiology [149, 150], cardiology [151, 152] and forensic science [153].

The labels given to these anatomical structures are completely up to the physician and naming conventions may differ between individuals, treatment facilities or could be influenced by the treatment planning software used. Physician specified structure set labels for three different prostate patients, shown below, illustrates the variability of labels even for the same disease site.

Patient 1: 1-PTV 45, BODY, PTV NODES, CTV, BOWEL, Prostate, Nodes,

PENILE_BULB, markers, SEM_VES, FEMUR_RT, 2-PTV-75.6, FEMUR.LT, BLADDER, RECTUM

Patient 2: gtv, nodes + .5, Trigone, sem ves, PTV1-45, Pelvic LN, CTV, BODY, Penile Bulb, markers, Prostate, SV + MARG, Femur R, Bowel, A2-PTV2-34.2, Femur L, bladder, rectum, Sigmoid

Patient 3: External, Prostate, CTV_4500, PTV_4500, bone, gtv, Fiducials, CTV_7920, rectosigmoid, SemVes_Prox, PenileBulb, SeminalVesicle, SEMVES MAG PTV1, PROX MARG PTV2, Femur_R, PTV_7920, Femur_L, Bladder, Rectum, Sigmoid

Although these three patients were treated for the same disease, the anatomical regions delineated and the given labels are not consistent. For example, the "Right Femur" anatomical region was given labels FEMUR.LT, Femur L, and Femur_L. Other inconsistencies in labels are shown for structures such the Recum, PTV and Seminal Vesicles.

The lack of a standard taxonomy and data dictionary in oncology can negatively affect both patient safety [154] and the ability to perform data analysis [155]. Standardizing the labels given to anatomical structures will enhance the safety and quality efforts within and between clinics for streamlining the clinical practice, data pooling for outcome research, registries and clinical trials. In order to define a standard for structure labels [156], the American Association of Physicists in Medicine (AAPM) Task Group 263 (TG-263) [157] proposed a comprehensive naming schema. Since this report has been published recently, its recommendations have not yet been universally adopted. Furthermore, global adherence to the TG-263 report in itself will not address historical data that have used non-standard labels. Manually relabeling these data sets at an individual clinic may not be feasible as this process is very time consuming and requires expert knowledge.

To aid in the standardization of structure labels, a few recent projects have proposed automated or semi-automated methods. Stature [158] is a system that allows human experts to identify structure label synonyms which can be used to help standardize the previously chosen labels. Although this method was shown to decrease the required time to manually edit structure set labels, it was designed to work with a single institution and the relabeling process may not work as well when presented with external data. Using the Stature system on data from other institutions would likely require additional expert input as there is no guarantee the same synonyms exist. Two other projects [159] and [160] uses neural networks to automatically relabel OAR structures with high accuracy for head and neck patients. Both of those works used selected (curated) data of OAR structures rather than all delineated structures present in the original clinical treatment plans.

Here we provide definitions of the type of data used in this work:

VA Data: This data set contains patient data from over 1,200 Veteran Affairs (VA) lung and prostate cases and was used for training and validation.

VCU Data: This data set contains patient data from 50 lung and 50 prostate patients from Virginia Commonwealth University (VCU) and was used for testing the models trained on VA data.

Curated Structures: The delineated OAR and target structures of interest which have been manually annotated after data collection to correctly associate physician specific labels to TG-263 defined labels. We have assigned the following structures for each disease site as curated: Lung: Esophagus, Heart, PTV, SpinalCord, Brachial_Plexus Prostate: Rectum, Bladder, Femur_L, Femur_R, PTV, Bowel_Large, Bowel_Small

Non-Annotated Structures: All structures not given one of the curated labels by the manual annotation process are identified as *non-annotated*. These structures

can include expansion margins of the target/tumor region, sub-regions of an OAR (i.e. lobes of the lung), anatomical structures not commonly delineated for the disease site in question and non-anatomical regions delineated to help guide the treatment planning system (TPS) to meet a dose delivery optimization goal. For the purpose of classification, all of these structures are given the non-annotated label.

Curated Data Set: A subset of the original clinical data which only includes structures that were given the previously defined curated labels. This means only one structure of the same anatomical region can exist per patient. Curated data sets were created by filtering the original VA and VCU data sets.

Non-Curated Data Set: This includes all structures present in the original clinical Radiation Oncology treatment plans from the VA and VCU. The non-curated data represents what is likely to be found at an arbitrary Radiation Oncology treatment facility.

An important factor not previously investigated is how to standardize structure labels in the the non-curated data found in clinical practices. Relabeling structures can be relatively straightforward if the potential options are limited and represent anatomically distinct objects. Since the non-annotated structures can represent a near infinite number of permutations, relabeling all structures in a clinical data set becomes a harder problem.

We propose an approach based on volumetric bitmap representations of the structures as well as the corresponding bony anatomy to build predictive models for relabeling clinically specified structure set labels to the TG-263 proposed labels [161]. Experiments were performed with five machine learning algorithms on the Apache Spark platform Machine Learning library.

Our proposed pipeline for structure set relabeling revealed the following observations:

- The best F_1 scores for lung was 98.77 and 95.06 for prostate; anatomical properties of prostate data may make it harder to classify.
- Classifying real world clinical (non-curated) data sets, which includes arbitrary structures, remains a challenging problem but F_1 scores above 90 are still achievable.
- Including bony anatomy data and treating structures as patient dependent improved classifier accuracy for both lung and prostate data.
- Using k-Means based undersampling on non-annotated structures can improve classifier predictive performance and run time.
- A significant percentage of structure labels unique to the test data was not present in the training data and highlights the limitations of simple text mappings.

8.1 Methods and Materials

8.1.1 Creation of Structure Sets

Once a patient has been diagnosed with cancer and radiotherapy is prescribed as part of the treatment, a patient model needs to be created to determine the radiation dose to the target volume, OARs and the coverage volume. Imaging of the patient is a necessary step and is often performed with Computed Tomography (CT) that provides tissue density information rendering the patient's anatomy. A physician will delineate the target/tumor region, OARs and any other structures deemed necessary for the current case. This delineation is usually done within the TPS software which will then allow for the creation of the dose delivery treatment plan.

Figure 31 shows the axial, coronal and sagital cut sections of a prostate cancer CT, overlaid with several delineated structures. The imaging and structure set infor-

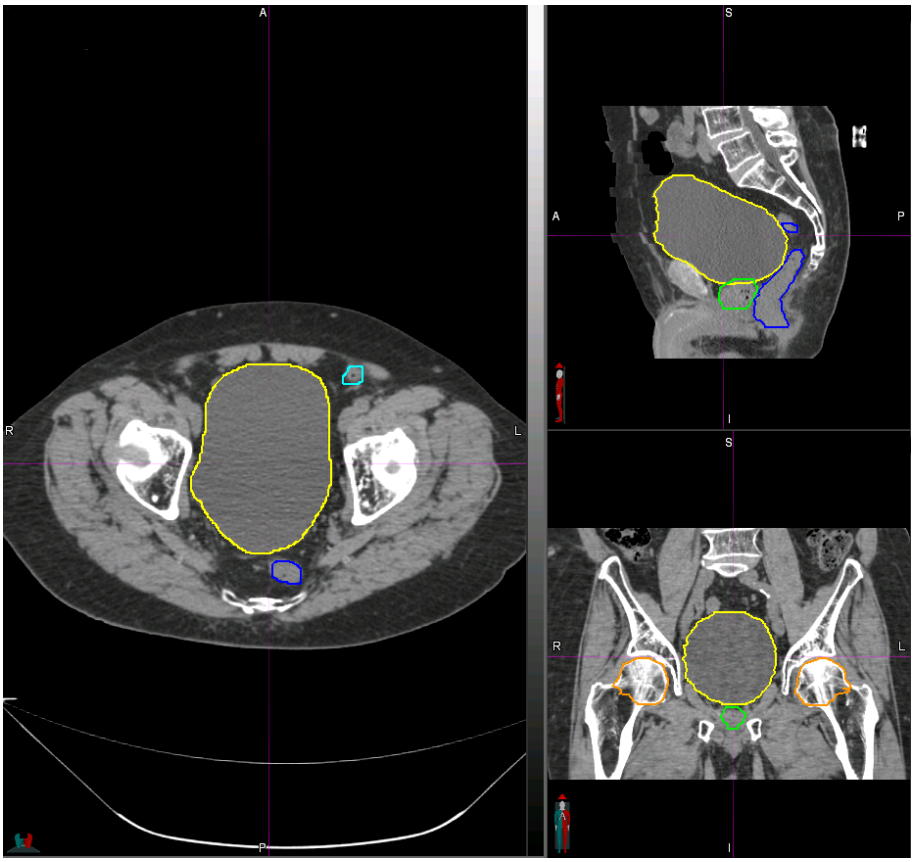


Figure 31: Planning CT from a prostate cancer patient with the following delineated structures: Bladder (yellow), Rectum (blue), Left and Right Femurs (orange), Small Bowel (aqua), PTV (green).

mation is in the Digital Imaging and Communications in Medicine (DICOM) format which is the industry standard for the storage and transmission of medical imaging data. This data is traditionally stored as slices on the axial axis but can be rendered on any axis. A physician will delineate any necessary structures using the delineation tool-sets in the TPS software; often by adding individual points or by using a free-hand drawing tool to create a closed polygon. For a given structure, this process is performed on each imaging slice until the delineation is complete.

For the same patient, Figure 32 shows the planning target volume (PTV) (green)

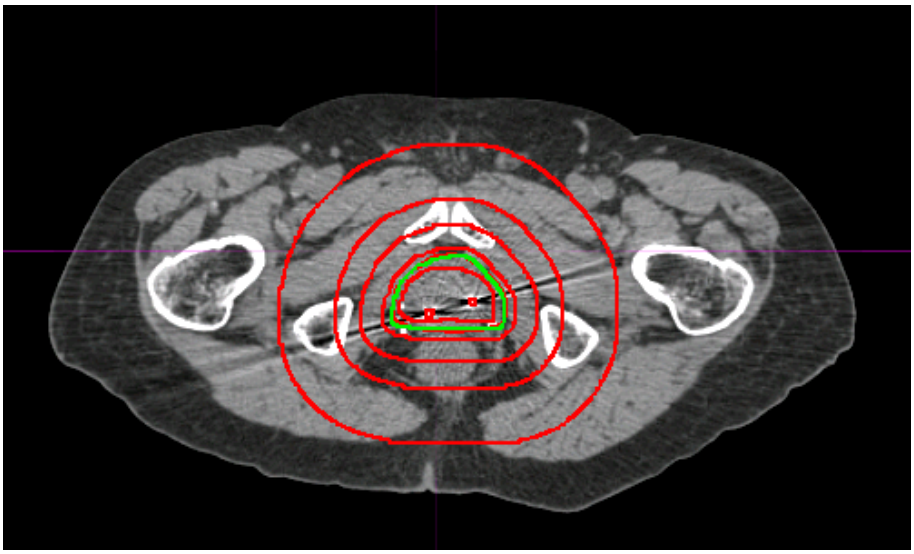


Figure 32: The PTV (green) as drawn by the physician and multiple other treatment planning related structures (red) delineated on a CT image. These planning structures include rings, implanted seeds, and several interpretations of the target volume by adding uniform or non-uniform expansions.

and multiple planning related structures (red). The PTV represents the region that will be receiving the prescribed radiation dose. It is also common to have other structures that are very similar to the PTV as presented here and may include a clinical target volume (CTV), gross tumor volume (GTV) or expansions of the PTV. Also presented in this figure are rings, used for helping to guide the TPS dose optimization process, and implanted marker seeds.

8.1.2 Datasets

The training and validation data set included a total of 709 lung and 752 prostate patients from the 40 VA hospitals which perform radiotherapy in-house and this data was originally collected as part of the Radiation Oncology Quality Surveillance Program (VA-ROQS) [162]. The de-identified data for each patient included the DICOM

Table 17: Number of individual structures and unique labels in the VA and VCU lung data sets.

Standard Label	**VA Lung Data Set**		**VCU Lung Data Set**	
	Structure Count	**Unique Labels**	**Structure Count**	**Unique Labels**
Non-Annotated	10,292	3,639	775	317
Esophagus	613	26	47	3
Heart	670	20	45	2
PTV	680	286	36	4
SpinalCord	681	37	48	6
Brachial_Plexus	108	44	4	4
Total	**13,044**	**4,052**	**955**	**336**

structure set files representing anatomical structures of interest and the corresponding treatment planning DICOM CT image. From the VCU Department of Radiation Oncology's TPS, 50 prostate and 50 lung patients were extracted to build an external test data set. Like the VA data, each of these patients were manually annotated to provide a correct TG-263 labeling for the listed structure types.

Tables 17 and 18 show the distribution of structures in the data sets and the number of distinct labels used for each structure type. The average number of structures for each patient in this data set is approximately 18 for lung and 20 for prostate which results in a high proportion of non-annotated structures. There is also a significant number of unique lables per structure type which illustrates why a comprehensive text mapping system has not yet been proposed. For all data sets, there is a high level of class imbalance especially between the non-annotated structures and the individual curated structures. The VCU prostate data did not include any Bowel_Large structures because they were not delineated in the patients included; a number of VA facilities also did not delineate bowel structures resulting in the lower numbers.

Table 18: Number of individual structures and unique labels in the VA and VA prostate data sets.

Standard Label	**VA Prostate Data Set**		**VCU Prostate Data Set**	
	Structure Count	**Unique Labels**	**Structure Count**	**Unique Labels**
Non-Annotated	11,038	2,799	980	434
Rectum	719	14	50	3
Bladder	609	10	50	3
Femur_L	694	59	29	14
Femur_R	700	62	29	13
PTV	714	236	38	16
Bowel_Large	341	34	0	0
Bowel_Small	250	40	49	10
Total	**15,065**	**3,254**	**1,225**	**493**

8.1.3 Data Preparation

The following process, shown in Figure 33, takes the DICOM structure set and imaging data and converts it into features vectors to be used as input for the classification algorithms. Figure 33(a) shows an original DICOM planning image and its associated structure set. The bladder structure delineation is shown in yellow.

Since the planning images available in our data set did not have consistent voxel count, voxel resolution or origins, a standard grid was needed so that all structure sets could be stored in a consistent manner. The standard grid chosen for this purpose was 96 x 96 x 48 voxels and with a voxel resolution of 2mm x 2mm x 3mm. These parameters were chosen by manually inspecting a number of bitmap examples from each structure type to verify that the bounding box was large enough to cover the structures of interest. It should be noted that this manual step is needed only once considering all the structures of interest and a large enough bounding box will lead to accurate results albeit needing slightly more execution time than the minimal bounding box, which is a computationally hard problem since some of the structure sizes

considered here demonstrate high variability. Future work is required to determine if such a one-size-fits-all based solution is sufficient, especially considering large structures like the entire lungs. In addition, each original planning image and structure set was shifted such that the geometric center of the given structure was aligned to the geometric center of this standardized grid.

The workflow for creating feature vectors from the imaging and structure set data is demonstrated using one prostate patient as shown in Figure 33(a). For each individual structure in the data set, an empty three-dimensional bitmap object was created with the standard grid dimensions as previously defined. Each polygon point in the DICOM structure set is mapped to its corresponding voxel in the new bitmap with a value of 1 as shown in Figure 33(b). Then for each transverse slice of the bitmap, the sequential points were connected with new line segments which results in one or more closed polygons per slice as shown in Figure 33(c). A flood fill algorithm [163] was then run on each closed polygon to set all interior values to 1 resulting in a solid bitmap structure shown in Figure 33(d). Voxels belonging to the structure in question would then have the value of 1 and all other voxels would remain as 0. The procedure used for generating these bitmaps was derived from the Research Computing Framework package [164].

In addition to the structure set data, imaging data was also used to add some spatial context to the location of each structure in the human anatomy. A density threshold was applied to each planning image such that voxels with Hounsfield units (HU) above 1,300 were set to 1 and all others set to 0, leaving only the bony anatomy. While bone density starts around 1,050 HU [165], we have chosen a slightly higher value to focus on the gross skeletal structure and reduce noise from borderline tissue. The resulting bony image was then interpolated to the same standardized grid used by the structures so that both data types were properly aligned. Figure 33(e) shows

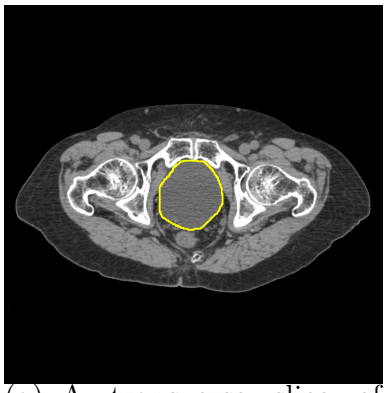
just the bony anatomy and Figure 33(f) shows the bony anatomy and structure set data combined.

To create features vectors, the 96 x 96 x 48 bitmap object was stretched out into a 442,368 x 1 vector by simply creating an array of each voxel value with increasing x, y, z axis indices. From this bitmap creation process, two data sets per disease site were created: *Without Bones* and *With Bones*.

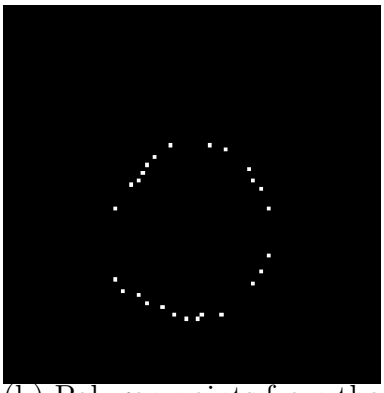
- *Without Bones*: The feature vectors were created with only structure set data as shown in Figure 33(d). The total length of the feature vector is 442,368.
- *With Bones*: The feature were created by appending the *No Bones* feature vectors with the bony anatomy data as shown in Figure 33(f). The total length of the feature vector is 884,736.

Very long feature vectors make the model training phase slow and susceptible to the *Curse of Dimensionality* [166]. One popular solution to this problem is to perform feature reduction by either removing features that are not strongly influencing predictions or condensing multiple features in such a way that still preserves importance. We have chosen to use truncated singular value decomposition (SVD) as it uses much smaller matrix multiplications when compared to methods like principal component analysis (PCA) or standard SVD [167]. This approach can approximate the input $m \times n$ matrix as $[m \times k] \times [k \times n]$ where k is the numerical rank [168]. When testing both methods, the truncated SVD ran faster and required less memory while still producing an explained variance within 0.1% of the result from PCA.

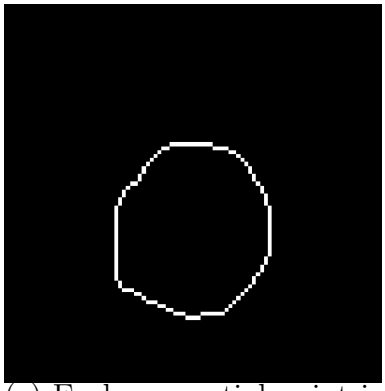
Figure 34 shows the explained variance of the disease sites for the *Without Bones* and *With Bones* data sets. All cases show a similar pattern and the cumulative variance curves start to flatten out around 100 features. For that reason, we have chosen 100 as the number of SVD features to use in our experiments as increasing



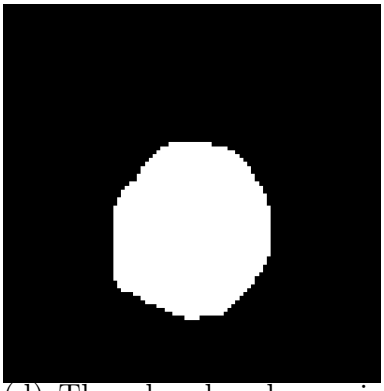
(a) A transverse slice of the original planning image with the bladder structure shown in yellow.



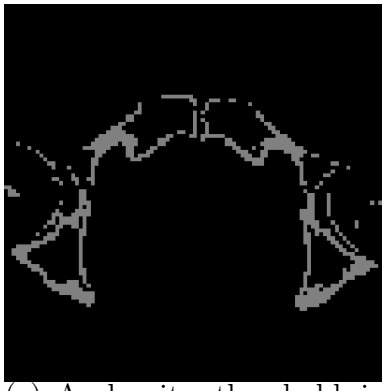
(b) Polygon points from the DICOM bladder structure set delineation. These individual points are interpolated on to the standardized bitmap volume.



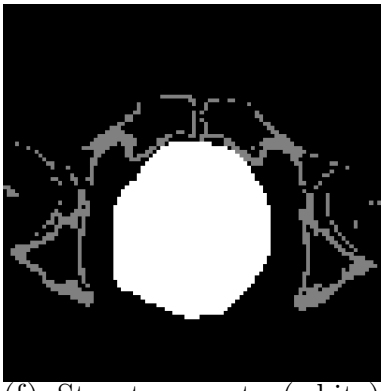
(c) Each sequential point is connected to form a close polygon.



(d) The closed polygon is flood filled to create a solid structure.



(e) A density threshold is applied to the planning image such that only voxels that belonged to bony anatomy remain.



(f) Structure set (white) and bony anatomy (grey) data shown together with the same frame of reference.

Figure 33: Workflow for creating structure set and bony anatomy bitmaps for feature vector creation.

the explained variance by more than a few percent would require at least doubling the total number of features. Initial tests using up to 1,000 SVD features did not improve classifier accuracies (data not shown).

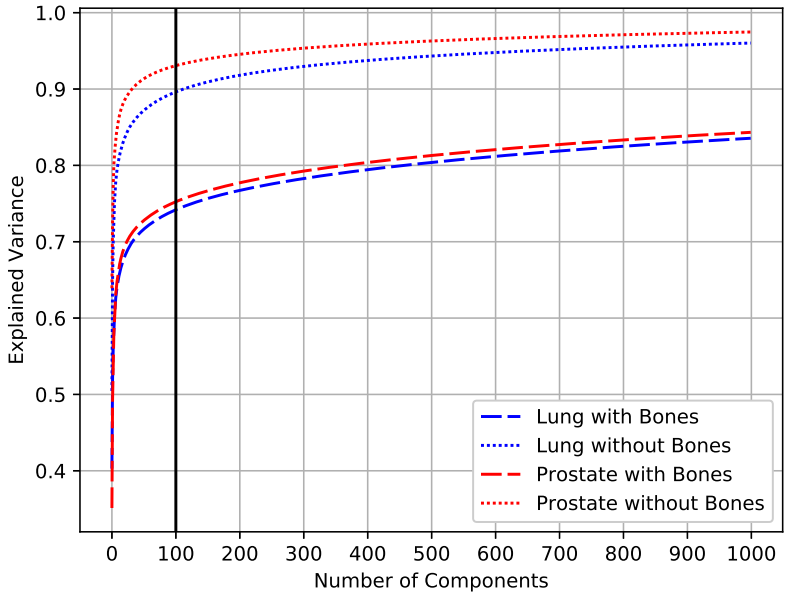


Figure 34: Cumulative explained variance from the number of features created by the SVD process. We have chosen the top 100 features in all models.

The anonymized patient identifier, physician specified label, and the TG-263 standardized label for each structure were added as features, not for model training, but for patient filtering and assessing the model accuracy.

8.1.4 Proposed Experiments

To train, validate and test our models, we have used five different classifier algorithms from the Apache Spark machine learning library [79]: Naive Bayes (NB)[169], Random Forest (RF)[62], Gradient-Boost Trees (GBT)[63], Multilayer Perceptron (MLP)[170], Support Vector Machine (SVM)[171]. Table 19 shows the hyperparam-

Table 19: Classifier hyperparameters used in the following experiments.

Algorithm	**Hyperparameters**
Support Vector Machine	*maxIter* = 25 *regParam (C)* = 0.001
Random Forest	*numTrees* = 50 *maxDepth* = 20
Gradient-Boost Trees	*maxIter (trees)* = 75 *maxDepth* = 5
Multilayer Perceptron	*maxIter* = 1000 *blockSize* = 64 one hidden layer with 70 neurons
Naive Bayes	None

eters used in these experiments which were discovered by a grid search.

In these experiments, we also investigate the contribution of treating structures patient dependent or independent. If structures are treated patient independent, the class with the highest predicted probability is chosen as done traditionally with classifiers. However, if structures are treated as patient dependent, their predicted class label is influenced by the predictions of other structures belonging to the same patient. In that case, the validation or test structures are grouped by the initial predicted label and then sorted by the raw probability values provided by the classifier. For each predicted label, the structure with the highest probability is given that label and all other structures are labeled as non-annotated. Structures predicted as non-annotated are also given that label. Our underlying assumption is that there can only be one type of structure per curated label for a given patient.

Table 20 shows an example of classified structures for a single patient. Each structure that truly belongs to one of the curated structures types is shown in bold.

Table 20: Example classification treating structures as patient dependent. Structures are grouped by predicted label and the one with the highest probability is relabeled. The highlighted structures are the ones truly being to one of the curated structure types.

Predicted Label	**Raw Probability**	**Original Label**
Not-Annotated	50	MarkedISO
	50	FinalISO
	50	BoostISO
	49	Seeds
	38	SVs
	38	1cmproxSV
	38	1700
	32	penile bulb
	28	PostRectum
	23	4900 (InitialIMRT)
	22	extless2cm
	22	External
	22	2cmstrip
	18	PTV nodes
	16	Nodes
	15	**PTV45Gy**
	14	RingPTVBoost
	12	RingPTV45

Predicted Label	**Raw Probability**	**Original Label**
Rectum	13	Prostate
	11	CTVBoost
Bladder	21	**Bladder**
Femur_L	21	**Lt Femur**
Femur_R	25	**Rt Femur**
	13	**rectum**
	12	ptv5mm
	9	PTV prostate + SV initial
PTV	14	Target
	12	PTVBoost
	11	BoostPTVMod
SmallBowel	22	**SmallBowel**
	17	Pelvis
	16	ptv4cm

The Bladder, LT Femur, Rt Femur and SmallBowel were given the correct label but the PTV (shown as PTV45Gy) and Rectum were mislabeled. Only one structure was predicted as being the Bladder and Femur_L so the assignment is automatic. The raw probability values in this example were the number of trees in the RF algorithm that chose the predicted class.

Classifiers can be biased towards the majority class, especially if the level of imbalance is high, and one strategy is to balance the classes before the training phase. To address this issue, the non-annotated structures were randomly under-sampled to the size of the largest curated structure class. All curated structure classes were then

randomly over-sampled to the size of the largest curated class so that all classes were balanced. The same oversampling was performed for both curated and non-curated data sets.

In summary, the following combinations of experiments were run:

- Algorithm: RF, GBT, MLP, SVM, NB
- Disease site: lung, prostate
- Data set type: curated, clinical
- Bones: without bones, with bones
- Patient Dependence: independent, dependent

Using the best data set combinations from those experiments, we then investigated the potential benefit of using k-Means clustering to aid undersampling. Each experiment used the VA data with 5-fold cross-validation [172, 173] and a majority vote ensemble of the resulting five models were used on the VCU test data set.

8.2 Results

The results are broken into four sections: curated data results, non-curated data results, investigating k-Means for undersampling, and run time analysis.

8.2.1 Curated Data Results

Table 21 shows the F_1 scores for each algorithm and data set using only curated structures. Except for Naive Bayes, all classifiers on the VA training data produced scores above 90 and best overall results for lung and prostate were 98.77 and 95.06 respectively. Including the bony anatomy data increased scores in all cases, most

significantly for Naive Bayes. Although treating structures as patient dependent did improve results in the majority of cases, the benefits were minimal.

Previous works published on this topic showed very high accuracies on head neck patient data sets [159, 160]. Our best result for lung, 98.77, is comparable to results from the HN_MAIA data set which reported an F_1 score of 98.90 [160]. The prostate scores were not as high but this may highlight a challenge with classifying prostate structures. Abdominal structures such as the rectum, bladder and bowels can have variable sizes and shapes as they are filled and emptied. In addition, different regions of the the digestive tract may be associated with the same structure type. For example, one physician may delineate the entire anatomical rectum while another will only delineate the region closest to the prostate as it is most needed for treatment planning. Even with these challenges, our model was still able to achieve an F_1 score of 95.06. For both disease sites, most of the best results came from the RF algorithm followed by MLP.

These models were then tested on the external VCU data sets using majority voting with the models created from the five folds of training. Table 21 shows results were not as high as with training but the best results still achieved scores above 95 for lung and 91 for prostate. Unlike during training, MLP outperformed RF in the majority of the tests.

8.2.2 Non-Curated Data Results

As expected, classifier accuracies dropped for all five algorithms when trained on the non-curated data sets as shown in Table 22. Using the bony anatomy information improved results but treating structures as patient dependence provided a more significant boost. The RF algorithm outperformed all other methods in both lung and prostate data sets and Naive Bayes again performed the worst.

Table 21: Classifier results for the Curated VA and VCU data sets.

Curated Data F_1 Scores										
Data Parameters	**VA**					**VCU**				
Lung	**GBT**	**NB**	**SVM**	**RF**	**MLP**	**GBT**	**NB**	**SVM**	**RF**	**MLP**
Without Bones, Patient Independent	93.10	76.71	89.65	93.10	**93.96**	84.33	15.96	79.13	83.27	**90.52**
Without Bones, Patient Dependent	93.02	77.59	89.94	93.68	**94.39**	82.69	17.05	81.27	83.26	**91.02**
With Bones, Patient Independent	97.75	94.91	97.42	**98.73**	98.48	93.96	79.14	92.23	95.24	**95.57**
With Bones, Patient Dependent	97.81	94.92	97.45	**98.77**	98.70	94.49	78.44	92.69	**95.67**	95.10
Prostate	**GBT**	**NB**	**SVM**	**RF**	**MLP**	**GBT**	**NB**	**SVM**	**RF**	**MLP**
Without Bones, Patient Independent	91.82	88.31	91.76	**92.94**	92.17	84.98	34.13	81.55	85.89	**87.50**
Without Bones, Patient Dependent	91.53	87.15	91.58	92.63	**92.74**	**87.44**	32.98	81.45	85.96	86.65
With Bones, Patient Independent	93.41	89.95	**95.02**	94.87	94.22	85.58	65.32	88.95	89.13	**91.06**
With Bones, Patient Dependent	93.44	89.86	95.02	**95.06**	94.68	86.14	65.16	87.55	88.13	**90.22**

The results from the VCU non-curated test data scores were similar between the training and testing for lung but were higher for prostate. Like with training, the RF algorithm consistently outperformed all other algorithms on both test data sets.

8.2.3 Clustering-based Undersampling with *k*-Means

In our previous experiments for both lung and prostate data, the best results came from using bony anatomy with patient dependence with the RF classifier. These experiments used random undersampling which is a popular method for addressing class imbalance but may not lead to the best possible classification results if the sampled data is not representative of the global population [27]. While random undersampling tend to work well when combined with instance selection and ensemble architecture [43], on its own may remove highly useful observations from the training set. The structures belonging to the non-annotated class actually represent multiple sub-concepts so clustering may improve the undersampling process [42]. These sub-concepts represent specific anatomical regions present in some of the patients. For example, the motorized table on which a patient lays on during treatment is often

Table 22: Classifier results for the Non-Curated VA and VCU data sets.

Non-Curated Data F_1 Scores										
Data Parameters	**VA**					**VCU**				
Lung	**GBT**	**NB**	**SVM**	**RF**	**MLP**	**GBT**	**NB**	**SVM**	**RF**	**MLP**
Without Bones, Patient Independent	79.01	51.42	53.88	**80.09**	77.79	79.83	23.27	40.54	**82.50**	80.21
Without Bones, Patient Dependent	88.43	76.13	80.32	**89.11**	88.29	88.39	73.62	75.19	**88.70**	86.52
With Bones, Patient Independent	80.26	54.13	62.47	**82.02**	79.45	81.26	37.97	58.65	**82.99**	80.29
With Bones, Patient Dependent	89.81	78.05	87.00	**90.89**	89.78	90.23	80.53	81.07	**90.74**	88.69
Prostate	**GBT**	**NB**	**SVM**	**RF**	**MLP**	**GBT**	**NB**	**SVM**	**RF**	**MLP**
Without Bones, Patient Independent	76.54	44.89	68.69	**77.21**	74.13	**79.84**	45.83	71.54	79.78	77.69
Without Bones, Patient Dependent	85.03	78.14	83.81	**86.48**	83.41	88.63	72.46	86.34	88.73	**90.10**
With Bones, Patient Independent	76.53	63.11	74.43	**78.47**	76.61	81.06	53.91	78.05	**81.22**	80.78
With Bones, Patient Dependent	86.14	78.82	86.90	**87.38**	85.48	89.43	81.86	89.18	**91.22**	90.95

delineated. Using clustering, structures belonging to the treatment table sub-concept could be grouped so that those structures would be undersampled proportional to its rate of occurrence.

To evaluate this approach, the k-Means algorithm was used with cluster counts of 10, 25, and 50 on the non-annotated structures. Each resulting cluster was undersampled proportional to its size so that the total number of remaining structures equaled the size of the largest curated class. Table 23 shows the F_1 scores of different non-annotated structure undersampling methods using the previous best combination of the RF algorithm using bony anatomy and treating structures as patient dependent.

Table 23 shows that F_1 scores with clustering is similar to random undersampling for both lung and prostate data. While the undersampling methods with prostate data provided higher scores than not undersampling, it was not the case for lung. To get a more complete picture of the effect of undersampling, Tables 24 and 25 show results for each structure type. Not undersampling the lung data provided the best result for the non-annotated class but at the cost of poor performance for a number of other structures, especially the PTV which had a score of only 1.16. All of structures

Table 23: F_1 scores for the best previous RF models with different undersampling methods.

Undersampling Method	**Lung**	**Prostate**
No Undersampling	**91.97**	86.55
Random Undersampling	90.89	87.38
10 Clusters	90.80	87.45
25 Clusters	90.86	87.31
50 Clusters	90.95	**87.61**

Table 24: Structure specific results for VA lung data using RF with random undersampling, k-Means cluster undersampling and no undersampling.

Structure	**None**	**Random**	***k*=10**	***k*=25**	***k*=50**
Non-Annotated	**95.10**	94.12	94.16	94.19	94.26
Esophagus	89.60	92.95	92.60	93.22	**93.78**
Heart	93.36	95.56	95.34	**95.94**	94.82
PTV	1.16	33.21	**37.25**	36.38	36.80
SpinalCord	94.54	**96.23**	96.16	95.59	95.73
Brachial_Plexus	53.72	**61.04**	51.75	53.79	56.09

from the prostate data set had the highest scores with some form of undersampling. The Rectum, Bladder and PTV structures performed significantly worse compared to any of the included undersampling methods. Overall, balancing the classes improved the F_1 scores for the most difficulty classes, although there was not a significant difference between random undersamping and k-Means clustering with varying sizes of k.

Table 25: Structure specific results for VA prostate data using RF with random undersampling, k-Means cluster undersampling and no undersampling.

Structure	**None**	**Random**	***k*=10**	***k*=25**	***k*=50**
Non-Annotated	91.42	91.03	91.53	91.48	**91.67**
Rectum	70.13	**81.70**	80.36	80.50	80.34
Bladder	60.55	**79.66**	75.74	76.93	75.02
Femur_L	94.13	93.85	96.01	**96.17**	95.95
Femur_R	94.21	92.76	96.41	**96.48**	96.34
PTV	18.11	53.98	53.39	51.42	**54.97**
Bowel_Large	59.51	54.97	**64.89**	63.03	64.60
Bowel_Small	44.84	45.39	54.99	53.50	**56.45**

8.2.4 Run Time Analysis

Table 26 shows the average run time per training fold using random undersampling, no undersampling and k-Means based undersampling with cluster counts of 10, 25, and 50 performed on the non-annotated structures. The GBT and MLP algorithms were significantly slower than the other three algorithms and NB was by far the fastest, although it consistently produced the worst F_1 scores. Not undersampling was faster than random undersampling, but as previously mentioned it comes with a cost of poor performance for certain structure types. Random undersampling was the slowest sampling method for all algorithms and was significantly worse for all except for MLP. Clustering provided the best run times for the RF and MLP algorithms which also had the highest classification accuracies shown in Tables 21 and 22.

8.3 Including Radiomic and Dosiomic Features

To extend the initial work on structure set standardization, the potential benefit of using radiomic [174] and dosiomic [175] information was also investigated. While the term *-omics* was originally used with molecular biology, it is now being applied

Table 26: Run time for the VA prostate data with random undersampling, k-Means cluster undersampling and no undersampling.

Algorithm	**None**	**Random**	k**=10**	k**=25**	k**=50**
GBT	9,113	81,277	14,498	15,066	14,602
NB	99	222	115	124	126
SVM	995	6,610	1,771	1,553	1,538
RF	452	929	308	289	297
MLP	7,630	10,077	10,538	8,759	6,905

to other high-dimensional data used in the field of medicine [176]. In this context, radiomics and dosiomics refers to the use of extracted features from imaging and dosimetric data.

8.3.1 Manual Feature Extraction

With the matching image and dose files, 27 high-level radiomic and dosiomic features were manually chosen [177], such as median, skew, kurtosis, signal-to-noise ratio, and uniformity [178]. Using the region that overlapped with the existing structure set bitmaps, these *-omic* features were extracted and concatenated to the existing 100 features from the singular value decomposition data which resulted in a new input vector.

Like before, the Random Forest classifier with 5-fold cross-validation was used with the Apache Spark platform. Because the dataset had a high level of class imbalance, the majority class (non-curated structures) was randomly undersampled and the minority classes (curated structured) were randomly oversampled to the size of the largest minority class.

In these experiments, it was shown that including soft tissue and dose related radiomic features can be used to improve the predictive power of a structure name

Comparing Structure Set Models		
	Structures Features Only	With Radiomic and Dosiomic Features
Non-Curated	91.08	**93.09**
Rectum	77.61	**88.22**
Bladder	74.87	**80.46**
Femur_L	95.97	**96.82**
Femur_R	**95.41**	95.21
PTV	54.04	**62.63**
Bowel_Large	66.33	**75.43**
Bowel_Small	51.56	**53.14**

Table 27: F_1 scores for the base model using only structure set data and the model using structure set, image, and dose information.

relabeling system. Most structures saw at least several percents of improvements of labeling accuracy but some, like the PTV Recum, and Bowel_Large, saw gains close to 10%. However, the overall performance for the PTV and the Bowels remained low.

8.3.2 Deep Learning Feature Extraction

While the manual feature extraction method showed improved results, it may be difficult always pick the most important features when learning from a specific dataset. Instead, a deep learning architecture can automate the feature engineering process and can potentially discover complicated features that would be difficult for a human practioner discern.

In these experiments [179], we used 9,750 structures from 550 prostate patients

to automatically label the same structures as the previous experiments. A bounding box was centered on each structure in the dataset and a bitmap was extracted. The same bounding box was used to extract the imaging and dose data. Four CNN models were tested: structure only, structure plus image, structure plus dose, structure plus image plus dose.

Using Keras, a network of three CNN layers, two dense layers and a softmax classifier was created. The CNN layers have 32, 64 to 128 filters, all using max 3D pooling, dropout of 0.2 and batch normalization. The first CNN layer used a kernel size of 7x7, while the next two used 3x3. The dense layers also use 0.2 dropout and use ReLU for an activation function, with the final class predictions performed by softmax.

8.3.2.1 Dynamic Undersampling for Deep Learning

While random undersampling is a common method for class balancing, it may not be optimal because the importance of the selected examples are not considered. This may result in examples that do not accurately represent the underlying distribution of the given class. The examples most useful for discovering decision boundaries may be very specific, as demonstrated in level difficulty problems [11], and the most useful examples may be excluded when sampling at random.

In order to address this issue, a dynamic undersampling method was employed. By overloading the standard Keras data loader with a custom method, the undersampling of the majority class was performed at the start at each epoch. While random undersampling permanently removes examples at the beginning of the training process, this dynamic approach provides a different subset of the majority class at each epoch. This means all of the majority examples will be seen at some point during training. The shuffling of data and the presentation of more diverse training exam-

ples can lead to more stable models and reduce overfitting. While this method was used for a single dataset and without directed experimentation, initial results shows improved performance over random undersampling and would benefit from future investigation.

8.3.2.2 Results

Table 28 shows that using all three data types usually gave the best results, with each of the other models providing the best result for a single structure. When compared to the previous experiments in Table 27, the results are mixed as some the CNN model performed better on some structures and worse on others. However, the CNN model performed better on the Bowel_Large and Bowel_Small structures which were among the most difficult to classify. In Table 29, the similar aggregate results between the VA training and VCU test data provided similar results show that the CNN models were not overfitting.

8.4 Conclusions

8.4.1 Future Work

While this work shows promise for automatic structure relabeling, there still may be room for improvement. Other classifiers may also provide better results individually or as part of a larger ensemble. Clustering was also shown to potentially improve classifier and run time performance but additional investigation is needed to maximize its benefits.

Including aspects of natural language processing (NLP) may add value as many of the structures can be logically relabeled to TG-263 standardized labels if they have permutations such as different capitalization, spacing or special character separators.

Comparing Model Performance for Each Structure				
	Structure	With Image	With Dose	All Three
Non-Curated	86.62	80.90	87.08	**88.22**
Rectum	81.66	70.34	78.74	**83.81**
Bladder	77.55	73.68	**79.21**	78.62
Femur_L	96.07	94.63	95.04	**97.11**
Femur_R	93.77	93.33	94.44	**96.22**
PTV	53.29	48.41	58.61	**60.72**
Bowel_Large	72.56	**77.47**	69.64	76.36
Bowel_Small	**57.50**	52.87	51.16	54.34

Table 28: F_1 scores comparing the results for individual structure type using four different models.

Test Set Results		
Dataset and Model	Weighed F_1 Scores VA Validation Set	Weighed F_1 Scores VCU External Test Set
Structures Only	84.17	82.92
With Image	79.07	80.05
With Dose	84.52	83.13
All Three	**86.08**	**85.70**

Table 29: A comparision of results between the VA validation and the VCU external test set which shows similar weighted F_1 scores.

However, NLP in itself may not be sufficient to correctly label structures in all cases. As shown in Tables 17 and 18, we have observed that some structures have several hundred different label permutations. There were also 220 lung and 378 prostate structure labels present in the VCU data sets that were not found in the larger, multi-facility VA data illustrating that different clinical data sets may have a high level of structure label variability.

Another major challenge is to correctly label target related structures. We have observed that these labels are often appended with text related to margins, dose values, physician initials or plan names. For a given patient, there can be structures such as GTV, ITV, and PTV as well as multiple permutations of each. When using the PTV structure for Quality Measure analysis, the correct related structure must be distinguished from other structures that appear similar geometrically and by their entered labels. If the geometry based classifier and label matching is not enough to relabel those target structures, using soft tissue, dose or plan information may help. Having more data for training often improves results but this kind of radiation oncology data can be difficult to obtain and the manual annotation process is very time intensive.

8.4.2 Summary

In this chapter, we showed that DICOM structure sets can be relabeled to selected TG-263 specified labels with high accuracy when considering curated structures only. While the accuracies are not as high when considering all arbitrary clinical structures, our approach still shows success for most structure types and these results may be further improved with the proposed future solutions. This also illustrates the difficulties of classifying data sets with non-annotated structures as it introduces the complexities of noise and class imbalance. The average number of structures per

patient from the VA and VCU is approximately 20, so it is likely that there will be a high number of these non-annotated structures in other clinical data sets.

We achieved similar results compared to previous works [159] and [160] (that used some variants of MLP that was implemented here) for curated data sets. In addition, we also investigated the challenges and opportunities when presented with large, clinical data sets. Even with including non-annotated structures, we were still able to achieve F_1 scores above 90 for both lung and prostate. In almost all cases, using bony anatomy and treating structures as patient dependent improved classifier results. With these parameters, the RF algorithm outperformed the other algorithms for all non-curated training experiments and six out of eight testing experiments.

Using the best data combinations, the effects of undersampling the majority class of non-annotated structures was further investigated. While not undersampling can provide competitive overall F_1 scores, it tends to come at the cost of poor performance on one or more of the other structure classes. The popular method of random undersampling resulted in significantly slower run times for all five algorithms compared to not undersampling or clustered based undersampling. While the k-Means clustering F_1 scores were comparable to the standard random understampling, it resulted in significantly faster run times for all five algorithms tested.

The main methodological contribution is the proposed pipeline for structure classification. Our pipeline incorporates image-based features (using volumetric bitmaps computed on a bounding box with feature reduction using SVD), standard machine learning classifiers and undersampling techniques (e.g., random and K-means clustering) to alleviate data bias. Prior work has only considered a subset of these components using cleaned data sets but we have investigated how our entire workflow could be used on real clinical datasets. Additionally, our pipeline allows for influencing the label given to a structure based on other structures for the same patient which has

not been previously investigated to the best of our knowledge.

The ultimate goal of this work is to allow for the development of an automated software system that will take arbitrary DICOM structure sets and accurately relabel them to TG-263 standardized labels. This would allow for efficient preprocessing of large radiation oncology data sets for various types of analysis but we also illustrate that this is a difficult problem. Correctly identifying the PTV within all possible tumor related structures proved to be a major challenge that will require future work.

If it turns out to be impractical to fully automate the relabeling process, a triage step may provide the best possible results. If a predictive model has high confidence that a given structure should be assigned a label, that label can be automatically applied. If not, those structures can be flagged for expert human review which hopefully would be a small portion of the total data. While using the standardized labels as defined by TG-263 is the best option going forward, we have shown some promise in automating the structure set relabeling in existing clinical data sets.

CHAPTER 9

FUTURE DIRECTIONS

The work presented in this dissertation has included novel experimentation, insights and solutions for the domain of imbalanced learning with Apache Spark. However, this only scratches the surface of what may be possible and so there are many opportunities for continued research.

9.1 Sampling with Apache Spark

The sampling methods mentioned in Chapter 2 were not designed specifically for big data problems or parallel execution. While many of these algorithms performed well on the smaller training datasets, it was not guaranteed they would perform the same when presented with larger datasets. The results shown in Chapter 4 highlighted a number of issues with using class balancing algorithms designed for serial execution. Although a number of algorithms were successfully implemented for Apache Spark, they could be further improved if redesigned with the MapReduce framework as part of the inherent design.

At this time, developing instance level difficulty based algorithms for Apache Spark is not trivial. Although the spark-knn package is available, it is not part of the official MLlib software and neither is any other unsupervised tree algorithm, including any implementation of kNN. Including these kinds of methods in the standard libraries will greatly increase the accessibility of developing algorithms dependent on tree structures. There are also no class balancing algorithms in MLlib which also hinders related research.

There is also limited work on how to best employ sampling based algorithm on distributed systems. One benefit of Apache Spark is that it does the data distribution under the hood making development and deployment easier. However, this can lead to data being distributed in a way harmful for the target algorithm. Future work will be needed to determine what rules should be followed when distributing data between cluster nodes and new algorithms should be designed to explicitly address this potential issue.

9.2 Minority Type Based Ensembles

As shown in [43], there can be a benefit using ensembles that take in account the difficulty of certain examples. Further expanding on this idea, such as ensembles from multiple values of k, may lead to better results. While considering instance level difficulty was shown to improve classification results, there are currently no methods to determine what combination of instance types should be included at training. Embedding such a feature in a class balancing algorithm would allow for an end-to-end solution. There is also a need to investigate if instance level difficulties should be treated independently for each class rather than the entire training dataset at once.

9.3 Multimodal Learning

While the majority of machine learning research has focused on data representing a single view, real world data can include information from multiple perspectives, also referred to as multimodal. These datasets can contain any number of combinations or multiple instances of tabular, image, text and signal data. For example, a dataset may include a photo with a caption, video with an audio track, or multiple types of satellite imagery for the same land feature.

Learning from this kind of data is more complicated because the inter- and intra-

relationships that must be understood. Although there have been a number of recent surveys on multimodal learning [180] [181] [182], there is almost no mention of issues related to class imbalance or instance level difficulty. However, there have been domain specific works that addressed multimodal class imbalance with random undersampling [183], random oversampling [184], cost-sensitive learning [185], weighted backpropagation, categorical cross entropy loss [186] and manual data pre-balancing [187]. The lack of more advanced class balancing techniques like SMOTE provides the opportunity for future work in this untapped area.

9.4 Cluster, Tree and Graph Based Sampling

Chapters 7 and 8 showed the benefits of both tree and cluster based methods on predictive power and run time. Since only two tree algorithms have been implemented for Apache Spark, hybrid spill and KD-trees, it would be beneficial if more were implemented. Chapter 4 also reported that k-Means on Apache Spark was a run time appropriate method and therefore could be considered for big data applications. Clustering could also be a valuable component in new versions of these classic algorithms as it could contribute in more explicit data partitioning. While not directly investigated in this dissertation, graph based methods may also be useful with big data imbalance problems, such as multimodal learning.

9.5 Class Imbalance with Deep Learning

Deep learning has become increasingly popular with the affordability of GPU hardware and the plug-and-play ability of modern software architectures. However, work on imbalanced data in this area has not caught up with the decades of research focusing on traditional machine learning and shallow neural networks. Many topics like instance level difficulty, the ordering of examples during training, class imbalance

aware networks and custom loss functions have not been fully explored. While AEs, AAEs and GANs have all shown potential in addressing class imbalance issues, it is not yet clear if the best solutions are algorithm based, data based or a combination of both.

CHAPTER 10

SUMMARY

This work has primarily focused on the MapReduce framework with Apache Spark, ensembles, class imbalance and instance level difficulty. The contributions of this dissertation includes novel Apache Spark implementation of class imbalance solutions, unique observations on how existing algorithms behave when used with the MapReduce model and additional evidence that instance level difficulty can significantly affect machine learning performance. In conclusion, the original proposed research questions from Chapter 3 are answered here:

RQ1: How do oversampling algorithms behave when used within the MapReduce framework? As shown in Chapter 4, the results illustrate a significant variation between classification accuracy and run time performance with the presented oversampling methods. Although some algorithms tended to work better than others, there was some dependence on the underlying classifier and the type of data used. These results provide some insight on what combination of the oversampling method and classifier might work well for a given problem. This work also provided a unique perspective on what kind of algorithmic components are likely to be appropriate for MapReduce based platforms. Any sub-component requiring $\mathcal{O}(n^2)$ or worse become intractable very quickly but kNN and k-Means are feasible if not overused. It was also observed while implementing these algorithms on Apache Spark that the simplicity of algorithms developed for single CPU machines should not be taken for granted. While they all can technically be written for Spark, features like adjusting examples from multiple class at once, rank ordering and probabilistic sampling can be inefficient to

run and are a nightmare to write. In addition, this was the first implementation of oversampling methods for Apache Spark written in Scala.

RQ2: What role does instance level difficulty play in the context of big data classification? The results in Chapter 5 showed that instance level difficulty can have a significant impact on classifier performance on large datasets. While Naive Bayes performed best without any oversampling in a significant number of experiments, it also gave poor overall scores on the nine multi-class metrics used. However, Random Forest performed much better across the board and did even better when considering instance level difficulty. This was true for random undersampling, random oversampling and SMOTE. A version of SMOTE with clustering was also used which provided many of the best overall results. In these experiments, the fifteen combinations of safe, borderline, rare and outliers were tested. In almost every case, at least one of these combinations provided a better result than the traditional approach of considering all examples when sampling. Future work is required to detect *a priori* which instance level difficulties should be used at training, but this work showed that an optimal combination likely exists.

RQ3: Do traditional bagging methods benefit from instance level difficulty information? The UnderBagging+ method, which favors more difficult examples when creating bags, outperformed all other tested methods on the large test datasets as shown in Chapter 6. It was also significantly better than standard UnderBagging, further showing the importance of choosing the right examples during training. Another observation is that UnderBagging+ provided its maximal result with fewer ensemble bags compared to the other tested methods. By focusing on the more difficult examples, UnderBagging+ also created less randomized ensembles which should lead to more model stability. In this work, UnderBagging+ was shown to have state-of-the-art performance and was the first such implementation on Apache

Spark.

RQ4: Can KD-trees be effectively implemented using the MapReduce framework? Chapter 7 showed that while the hybrid spill tree method often provided the most accurate results, the KD-tree implementation was close behind and in some cases ran three time faster. For some of the datasets, KD-tree outperformed the hybrid method showing that it may be the superior tree algorithm to use in some cases. Novel to this work was the extensive experimentation on performance based on leaf size and the number of threads when running on an AWS cluster. From these tests, it was shown that the chosen leaf size can significantly affect run time performance. Making the leaves larger decreases the tree depth but also increases the brute force cost when searching all examples in a target leaf. This means the best performance will come when these forces are balanced. The KD-tree was also used with a custom version of SMOTE with clustering which gave better results than the base SMOTE in most cases and was significantly better on some of the datasets.

RQ5: Can class imbalanced radiotherapy structure set data be accurately classified? In Chapter 8, it was shown that individual structures types can be classified with accuracies above 95 percent. Unique to prior works, these experiments focused the hardest problem of real clinical data which included a large amount of unwanted examples. The major challenge presented was how to deal with the large majority class which dwarfed multiple minority classes. After the initial experiments were performed with structure set data only, manually extracted image and dose features were also incorporated in the models which significantly improved performance for several of the minority classes. The same data was then tested with a CNN based model that provided similar results without the needed to perform manual feature extraction. While the first two experiments used random under and oversampling for class balancing, the CNN model tried a new approach of randomly undersampling

the majority class at each epoch instead of once at the beginning of training. While limited experiments were performed, these sampling results looked promising and suggest further investigation.

REFERENCES

[1] *Big Data Analytics*. Apr. 2020. URL: https://www.ibm.com/analytics/hadoop/big-data-analytics.

[2] *The 5 Vs of big data*. Apr. 2020. URL: https://www.ibm.com/blogs/watson-health/the-5-vs-%%20f-big-data/.

[3] Amir M Hormozi and Stacy Giles. "Data mining: A competitive weapon for banking and retail industries". In: *Information systems management* 21.2 (2004), pp. 62–71.

[4] Louis Columbus. "10 Ways Machine Learning Is Revolutionizing Marketing". In: (2018). URL: https://www.forbes.com/sites/louiscolumbus/2018/02/25/10-ways-machine-learning-is-revolutionizing-marketing/#2c29c8755bb6.

[5] Byeonghwa Park and Jae Kwon Bae. "Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data". In: *Expert Systems with Applications* 42.6 (2015), pp. 2928–2934.

[6] Christian Castaneda et al. "Clinical decision support systems for improving diagnostic accuracy and achieving precision medicine". In: *Journal of clinical bioinformatics* 5.1 (2015), p. 4.

[7] Nic Fleming. "How artificial intelligence is changing drug discovery". In: *Nature* 557.7706 (2018), S55–S55.

[8] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[9] Matei Zaharia et al. “Apache spark: a unified engine for big data processing”. In: *Communications of the ACM* 59.11 (2016), pp. 56–65.

[10] Haibo He and Edwardo A Garcia. “Learning from imbalanced data”. In: *IEEE Transactions on knowledge and data engineering* 21.9 (2009), pp. 1263–1284.

[11] José A. Sáez, Bartosz Krawczyk, and Michal Woźniak. “Analyzing the oversampling of different classes and types of examples in multi-class imbalanced datasets”. In: *Pattern Recognition* 57 (2016), pp. 164–178.

[12] Ronaldo C Prati, Gustavo EAPA Batista, and Maria Carolina Monard. “Class imbalances versus class overlapping: an analysis of a learning system behavior”. In: *Mexican international conference on artificial intelligence*. Springer. 2004, pp. 312–321.

[13] Alberto Fernández, Salvador Garcı**********a, and Francisco Herrera. “Addressing the classification with imbalanced data: open problems and new challenges on class distribution”. In: *International Conference on Hybrid Artificial Intelligence Systems*. Springer. 2011, pp. 1–10.

[14] Jason Van Hulse and Taghi Khoshgoftaar. “Knowledge discovery from imbalanced and noisy data”. In: *Data & Knowledge Engineering* 68.12 (2009), pp. 1513–1542.

[15] Justin M Johnson and Taghi M Khoshgoftaar. “Survey on deep learning with class imbalance”. In: *Journal of Big Data* 6.1 (2019), pp. 1–54.

[16] Chong Zhang et al. “A cost-sensitive deep belief network for imbalanced classification”. In: *IEEE transactions on neural networks and learning systems* 30.1 (2018), pp. 109–122.

[17] Mateusz Buda, Atsuto Maki, and Maciej A Mazurowski. "A systematic study of the class imbalance problem in convolutional neural networks". In: *Neural Networks* 106 (2018), pp. 249–259.

[18] Nitesh V Chawla et al. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[19] György Kovács. "smote-variants: a Python Implementation of 85 Minority Oversampling Techniques". In: *Neurocomputing* 366 (2019). (IF-2019=4.07), pp. 352–354. DOI: 10.1016/j.neucom.2019.06.100.

[20] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. "Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning". In: *International conference on intelligent computing*. Springer. 2005, pp. 878–887.

[21] Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. "Safe-Level-SMOTE: Safe-Level-Synthetic Minority Over-Sampling TEchnique for Handling the Class Imbalanced Problem". In: *Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*. PAKDD '09. Bangkok, Thailand: Springer-Verlag, 2009, pp. 475–482. ISBN: 978-3-642-01306-5. DOI: 10.1007/978-3-642-01307-2_43. URL: http://dx.doi.org/10.1007/978-3-642-01307-2_43.

[22] Tomasz Maciejewski and Jerzy Stefanowski. "Local neighbourhood extension of SMOTE for mining imbalanced data". In: *2011 IEEE symposium on computational intelligence and data mining (CIDM)*. IEEE. 2011, pp. 104–111.

[23] Haibo He et al. "ADASYN: Adaptive synthetic sampling approach for imbalanced learning". In: *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. IEEE. 2008, pp. 1322–1328.

[24] Bhagat Singh Raghuwanshi and Sanyam Shukla. "Classifying imbalanced data using BalanceCascade-based kernelized extreme learning machine". In: *Pattern Anal. Appl.* 23.3 (2020), pp. 1157–1182.

[25] Fenglian Li et al. "Cost-sensitive and hybrid-attribute measure multi-decision tree over imbalanced data sets". In: *Inf. Sci.* 422 (2018), pp. 242–256.

[26] Salman H. Khan et al. "Cost-Sensitive Learning of Deep Feature Representations From Imbalanced Data". In: *IEEE Trans. Neural Networks Learn. Syst.* 29.8 (2018), pp. 3573–3587.

[27] Alberto Fernández et al. *Learning from Imbalanced Data Sets*. Springer, 2018.

[28] Shuo Wang and Xin Yao. "Multiclass Imbalance Problems: Analysis and Potential Solutions". In: *IEEE Trans. Syst. Man Cybern. Part B* 42.4 (2012), pp. 1119–1130.

[29] Krystyna Napierala and Jerzy Stefanowski. "Types of minority class examples and their influence on learning classifiers from imbalanced data". In: *J. Intell. Inf. Syst.* 46.3 (2016), pp. 563–597.

[30] Alberto Fernández et al. "Analysing the classification of imbalanced datasets with multiple classes: Binarization techniques and ad-hoc approaches". In: *Knowl. Based Syst.* 42 (2013), pp. 97–110.

[31] Xuesong Zhang et al. "Transfer Boosting With Synthetic Instances for Class Imbalanced Object Recognition". In: *IEEE Trans. Cybern.* 48.1 (2018), pp. 357–370.

[32] Zhongliang Zhang et al. "Empowering one-vs-one decomposition with ensemble learning for multi-class imbalanced data". In: *Knowl. Based Syst.* 106 (2016), pp. 251–263.

[33] Bartosz Krawczyk. "Cost-sensitive one-vs-one ensemble for multi-class imbalanced data". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2016, pp. 2447–2452.

[34] Tuanfei Zhu, Yaping Lin, and Yonghe Liu. "Synthetic minority oversampling technique for multiclass imbalance problems". In: *Pattern Recognition* 72 (2017), pp. 327–340.

[35] Alberto Fernández et al. "A Pareto-based Ensemble with Feature and Instance Selection for Learning from Multi-Class Imbalanced Datasets". In: *Int. J. Neural Syst.* 27.6 (2017), 1750028:1–1750028:21.

[36] Mateusz Lango and Jerzy Stefanowski. "Multi-class and feature selection extensions of Roughly Balanced Bagging for imbalanced data". In: *J. Intell. Inf. Syst.* 50.1 (2018), pp. 97–127.

[37] Lida Abdi and Sattar Hashemi. "To Combat Multi-Class Imbalanced Problems by Means of Over-Sampling Techniques". In: *IEEE Trans. Knowl. Data Eng.* 28.1 (2016), pp. 238–251.

[38] X. Yang et al. "AMDO: an Over-Sampling Technique for Multi-Class Imbalanced Problems". In: *IEEE Transactions on Knowledge and Data Engineering* (2017), pp. 1–1. DOI: 10.1109/TKDE.2017.2761347.

[39] T. Ryan Hoens et al. "Building Decision Trees for the Multi-class Imbalance Problem". In: *Advances in Knowledge Discovery and Data Mining - 16th Pacific-Asia Conference, PAKDD 2012, Kuala Lumpur, Malaysia, May 29-June 1, 2012, Proceedings, Part I*. Vol. 7301. Lecture Notes in Computer Science. Springer, 2012, pp. 122–134.

[40] Simon Bernard et al. “The Multiclass ROC Front method for cost-sensitive classification”. In: *Pattern Recognition* 52 (2016), pp. 46–60.

[41] Guillem Collell, Drazen Prelec, and Kaustubh R. Patil. “A simple plug-in bagging ensemble based on threshold-moving for classifying binary and multiclass imbalanced data”. In: *Neurocomputing* 275 (2018), pp. 330–340.

[42] Bartosz Krawczyk. “Learning from imbalanced data: open challenges and future directions”. In: *Progress in Artificial Intelligence* 5.4 (2016), pp. 221–232.

[43] William C. Sleeman IV and Bartosz Krawczyk. “Bagging Using Instance-Level Difficulty for Multi-Class Imbalanced Big Data Classification on Spark”. In: *2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, December 9-12, 2019*. IEEE, 2019, pp. 2484–2493.

[44] Sukarna Barua et al. “MWMOTE–majority weighted minority oversampling technique for imbalanced data set learning”. In: *IEEE Transactions on knowledge and data engineering* 26.2 (2012), pp. 405–425.

[45] Michał Koziarski, Bartosz Krawczyk, and Michał Woźniak. “Radial-based approach to imbalanced data oversampling”. In: *International Conference on Hybrid Artificial Intelligence Systems*. Springer. 2017, pp. 318–327.

[46] Shiven Sharma et al. “Synthetic oversampling with the majority class: A new perspective on handling extreme imbalance”. In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2018, pp. 447–456.

[47] Wanthanee Prachuabsupakij and Nuanwan Soonthornphisaj. “Clustering and combined sampling approaches for multi-class imbalanced data classification”.

In: *Advances in information technology and industry applications*. Springer, 2012, pp. 717–724.

[48] Georgios Douzas, Fernando Bacao, and Felix Last. "Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE". In: *Information Sciences* 465 (2018), pp. 1–20.

[49] Ian J Goodfellow et al. "Generative adversarial networks". In: *arXiv preprint arXiv:1406.2661* (2014).

[50] Giovanni Mariani et al. "BAGAN: Data Augmentation with Balancing GAN". In: *CoRR* abs/1803.09655 (2018). arXiv: 1803.09655. URL: http://arxiv.org/abs/1803.09655.

[51] Sankha Subhra Mullick, Shounak Datta, and Swagatam Das. "Generative adversarial minority oversampling". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1695–1704.

[52] Minjae Son et al. "BCGAN: A CGAN-based over-sampling model using the boundary class for data balancing". In: *The Journal of Supercomputing* (2021), pp. 1–25.

[53] Adamu Ali-Gombe and Eyad Elyan. "MFC-GAN: class-imbalanced dataset classification using multiple fake class generative adversarial network". In: *Neurocomputing* 361 (2019), pp. 212–221.

[54] Justin Engelmann and Stefan Lessmann. "Conditional Wasserstein GAN-based oversampling of tabular data for imbalanced learning". In: *Expert Systems with Applications* 174 (2021), p. 114582.

[55] Swee Kiat Lim et al. "Doping: Generative data augmentation for unsupervised anomaly detection with gan". In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2018, pp. 1122–1127.

[56] Addisson Salazar, Luis Vergara, and Gonzalo Safont. "Generative Adversarial Networks and Markov Random Fields for oversampling very small training sets". In: *Expert Systems with Applications* 163 (2021), p. 113819.

[57] Damien Dablain, Bartosz Krawczyk, and Nitesh V Chawla. "DeepSMOTE: Fusing Deep Learning and SMOTE for Imbalanced Data". In: *arXiv preprint arXiv:2105.02340* (2021).

[58] Leo Breiman. "Bagging predictors". In: *Machine learning* 24.2 (1996), pp. 123–140.

[59] Shuo Wang and Xin Yao. "Diversity analysis on imbalanced data sets by using ensemble models". In: *2009 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE. 2009, pp. 324–331.

[60] Robert E Schapire. "The strength of weak learnability". In: *Machine learning* 5.2 (1990), pp. 197–227.

[61] Yoav Freund and Robert E Schapire. "A desicion-theoretic generalization of on-line learning and an application to boosting". In: *European conference on computational learning theory*. Springer. 1995, pp. 23–37.

[62] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[63] Jerome H Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* (2001), pp. 1189–1232.

[64] Jerry Ye et al. “Stochastic gradient boosted distributed decision trees”. In: *Proceedings of the 18th ACM conference on Information and knowledge management*. 2009, pp. 2061–2064.

[65] Guolin Ke et al. “Lightgbm: A highly efficient gradient boosting decision tree”. In: *Advances in neural information processing systems*. 2017, pp. 3146–3154.

[66] Pierre Geurts, Damien Ernst, and Louis Wehenkel. “Extremely randomized trees”. In: *Machine learning* 63.1 (2006), pp. 3–42.

[67] Zardad Khan et al. “Ensemble of optimal trees, random forest and random projection ensemble classification”. In: *Advances in Data Analysis and Classification* 14.1 (2020), pp. 97–116.

[68] Rodrigo Minetto, Maurı**********cio Pamplona Segundo, and Sudeep Sarkar. “Hydra: An ensemble of convolutional neural networks for geospatial land classification”. In: *IEEE Transactions on Geoscience and Remote Sensing* 57.9 (2019), pp. 6530–6541.

[69] Amirreza Mahbod et al. “Transfer learning using a multi-scale and multi-network ensemble for skin lesion classification”. In: *Computer methods and programs in biomedicine* 193 (2020), p. 105475.

[70] Bo Cheng et al. “Random cropping ensemble neural network for image classification in a robotic arm grasping system”. In: *IEEE Transactions on Instrumentation and Measurement* 69.9 (2020), pp. 6795–6806.

[71] Krystyna Napierala and Jerzy Stefanowski. “Identification of different types of minority class examples in imbalanced data”. In: *International Conference on Hybrid Artificial Intelligence Systems*. Springer. 2012, pp. 139–150.

[72] Przemysław Skryjomski and Bartosz Krawczyk. "Influence of minority class instance types on SMOTE imbalanced data oversampling". In: *first international workshop on learning with imbalanced domains: theory and applications*. 2017, pp. 7–21.

[73] Shuang Yu et al. "BIDI: A classification algorithm with instance difficulty invariance". In: *Expert Systems with Applications* 165 (2021), p. 113920.

[74] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: `http://archive.ics.uci.edu/ml`.

[75] Apache Software Foundation. *Apache Hadoop*. URL: `http://hadoop.apache.org/` (visited on 03/31/2019).

[76] Apache Software Foundation. *Apache Hadoop*. URL: `https://spark.apache.org/docs/latest/cluster-overview.html` (visited on 03/31/2019).

[77] Apache Software Foundation. *HDFS Architecture*. 2010. URL: `http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html` (visited on 03/31/2010).

[78] Apache Software Foundation. *RDD Programming Guide*. URL: `https://spark.apache.org/docs/latest/rdd-programming-guide.html` (visited on 03/31/2019).

[79] Xiangrui Meng et al. "Mllib: Machine learning in apache spark". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1235–1241.

[80] Anand Gupta et al. "A big data analysis framework using apache spark and deep learning". In: *2017 IEEE international conference on data mining workshops (ICDMW)*. IEEE. 2017, pp. 9–16.

[81] Muhammad Ashfaq Khan, Md Karim, Yangwoo Kim, et al. “A two-stage big data analytics framework with real world applications using spark machine learning and long Short-term memory network”. In: *Symmetry* 10.10 (2018), p. 485.

[82] Yunyan Guo et al. “Model averaging in distributed machine learning: a case study with Apache Spark”. In: *The VLDB Journal* (2021), pp. 1–20.

[83] Sergio Ramı**********rez-Gallego et al. “Big Data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce”. In: *Information Fusion* 42 (2018), pp. 51–61.

[84] Sara del Rı**********o et al. “On the use of MapReduce for imbalanced big data using Random Forest”. In: *Inf. Sci.* 285 (2014), pp. 112–137.

[85] Sergio Ramı**********rez-Gallego et al. “An Information Theory-Based Feature Selection Framework for Big Data Under Apache Spark”. In: *IEEE Trans. Systems, Man, and Cybernetics: Systems* 48.9 (2018), pp. 1441–1453.

[86] Alberto Fernández, Eva Almansa, and Francisco Herrera. “Chi-Spark-RS: An Spark-built evolutionary fuzzy rule selection algorithm in imbalanced classification for big data problems”. In: *2017 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2017, Naples, Italy, July 9-12, 2017*. 2017, pp. 1–6.

[87] Isaac Triguero et al. “Evolutionary undersampling for extremely imbalanced big data classification under apache spark”. In: *IEEE Congress on Evolutionary Computation, CEC 2016, Vancouver, BC, Canada, July 24-29, 2016*. 2016, pp. 640–647.

[88] Alberto Cano. "A survey on graphic processing unit computing for large-scale data mining". In: *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* 8.1 (2018).

[89] Alberto Cano, Amelia Zafra, and Sebastián Ventura. "Weighted Data Gravitation Classification for Standard and Imbalanced Data". In: *IEEE Trans. Cybernetics* 43.6 (2013), pp. 1672–1687.

[90] Alberto Cano et al. "A classification module for genetic programming algorithms in JCLEC". In: *J. Mach. Learn. Res.* 16 (2015), pp. 491–494.

[91] Alberto Cano et al. "ur-CAIM: improved CAIM discretization for unbalanced and balanced data". In: *Soft Comput.* 20.1 (2016), pp. 173–188.

[92] M Mazhar Rathore et al. "Real-time big data stream processing using GPU with spark over hadoop ecosystem". In: *International Journal of Parallel Programming* 46.3 (2018), pp. 630–646.

[93] KR Jayaram et al. "Adaptively accelerating map-reduce/spark with gpus: A case study". In: *2019 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE. 2019, pp. 105–114.

[94] Yasuhiro Ohno, Shin Morishima, and Hiroki Matsutani. "Accelerating spark RDD operations with local and remote GPU devices". In: *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE. 2016, pp. 791–799.

[95] Dalton Lunga et al. "Apache spark accelerated deep learning inference for large scale satellite image analytics". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 13 (2020), pp. 271–283.

[96] Klodjan Klodi Hidri, Angelos Bilas, and Christos Kozanitis. "HetSpark: A Framework that Provides Heterogeneous Executors to Apache Spark". In: *Procedia Computer Science* 136 (2018), pp. 118–127.

[97] Ehsan Ghasemi and Paul Chow. "Accelerating apache spark with fpgas". In: *Concurrency and Computation: Practice and Experience* 31.2 (2019), e4222.

[98] Junjie Hou et al. "A case study of accelerating apache spark with FPGA". In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE. 2018, pp. 855–860.

[99] Lukasz Korycki and Bartosz Krawczyk. "Low-Dimensional Representation Learning from Imbalanced Data Streams". In: *Advances in Knowledge Discovery and Data Mining - 25th Pacific-Asia Conference, PAKDD 2021, Virtual Event, May 11-14, 2021, Proceedings, Part I*. Vol. 12712. Lecture Notes in Computer Science. Springer, 2021, pp. 629–641.

[100] William C. Sleeman IV and Bartosz Krawczyk. *Imbalanced Big Data Oversampling: Taxonomy, Algorithms, Software, Guidelines and Future Directions*. 2021. arXiv: `2107.11508 [cs.LG]`.

[101] Payel Sadhukhan and Sarbani Palit. "Reverse-nearest neighborhood based oversampling for imbalanced, multi-label datasets". In: *Pattern Recognit. Lett.* 125 (2019), pp. 813–820.

[102] Michal Koziarski, Bartosz Krawczyk, and Michal Wozniak. "Radial-Based oversampling for noisy imbalanced data classification". In: *Neurocomputing* 343 (2019), pp. 19–33.

[103] Bartosz Krawczyk, Michal Koziarski, and Michal Wozniak. "Radial-Based Oversampling for Multiclass Imbalanced Data Classification". In: *IEEE Transactions on Neural Networks and Learning Systems* 31.8 (2020), pp. 2818–2831.

[104] Shiven Sharma, Anil Somayaji, and Nathalie Japkowicz. "Learning over subconcepts: Strategies for 1-class classification". In: *Comput. Intell.* 34.2 (2018), pp. 440–467.

[105] José A Sáez et al. "SMOTE–IPF: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering". In: *Information Sciences* 291 (2015), pp. 184–203.

[106] Michal Koziarski, Michal Wozniak, and Bartosz Krawczyk. "Combined Cleaning and Resampling algorithm for multi-class imbalanced data with label noise". In: *Knowledge-Based Systems.* 204 (2020), p. 106223.

[107] Ahmad S. Tarawneh et al. "SMOTEFUNA: Synthetic Minority Over-Sampling Technique Based on Furthest Neighbour Algorithm". In: *IEEE Access* 8 (2020), pp. 59069–59082.

[108] Gede Angga Pradipta et al. "Radius-SMOTE: A New Oversampling Technique of Minority Samples Based on Radius Distance for Learning From Imbalanced Data". In: *IEEE Access* 9 (2021), pp. 74763–74777.

[109] Joanna Grzyb, Jakub Klikowski, and Michal Wozniak. "Hellinger Distance Weighted Ensemble for imbalanced data stream classification". In: *J. Comput. Sci.* 51 (2021), p. 101314.

[110] Colin Bellinger, Christopher Drummond, and Nathalie Japkowicz. "Manifold-based synthetic oversampling with manifold conformance estimation". In: *Mach. Learn.* 107.3 (2018), pp. 605–637.

[111] Lingkai Yang, Yi-Nan Guo, and Jian Cheng. "Manifold Distance-Based Over-Sampling Technique for Class Imbalance Learning". In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 10071–10072.

[112] Lin Feng et al. "Learning a Distance Metric by Balancing KL-Divergence for Imbalanced Datasets". In: *IEEE Trans. Syst. Man Cybern. Syst.* 49.12 (2019), pp. 2384–2395.

[113] Fanxia Zeng et al. "Re-KISSME: A robust resampling scheme for distance metric learning in the presence of label noise". In: *Neurocomputing* 330 (2019), pp. 138–150.

[114] Yi-Hsun Liu, Chien-Liang Liu, and Shin-Mu Tseng. "Deep Discriminative Features Learning and Sampling for Imbalanced Data Problem". In: *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*. IEEE Computer Society, 2018, pp. 1146–1151.

[115] Pattaramon Vuttipittayamongkol and Eyad Elyan. "Neighbourhood-based undersampling approach for handling imbalanced and overlapped data". In: *Inf. Sci.* 509 (2020), pp. 47–70.

[116] Sara del Rı**********o, José Manuel Benı**********
and Francisco Herrera. "Analysis of Data Preprocessing Increasing the Oversampling Ratio for Extremely Imbalanced Big Data Classification". In: *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 2*. IEEE, 2015, pp. 180–185.

[117] Michal Koziarski and Michal Wozniak. "CCR: A combined cleaning and resampling algorithm for imbalanced data classification". In: *International Journal of Applied Mathematics and Computer Science* 27 (Dec. 2017), pp. 727–736.

[118] Junnan Li et al. "A novel oversampling technique for class-imbalanced learning based on SMOTE and natural neighbors". In: *Inf. Sci.* 565 (2021), pp. 438–455.

[119] Baiyun Chen et al. "RSMOTE: A self-adaptive robust SMOTE for imbalanced problems with label noise". In: *Inf. Sci.* 553 (2021), pp. 397–428.

[120] Lukasz Korycki and Bartosz Krawczyk. "Online Oversampling for Sparsely Labeled Imbalanced and Non-Stationary Data Streams". In: *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*. IEEE, 2020, pp. 1–8.

[121] W Siriseriwan and Krung Sinapiromsaran. "Adaptive neighbor synthetic minority oversampling technique under 1NN outcast handling". In: 39 (Sept. 2017), pp. 565–576.

[122] D. A. Cieslak, N. V. Chawla, and A. Striegel. "Combating imbalance in network intrusion datasets". In: *2006 IEEE International Conference on Granular Computing*. May 2006, pp. 732–737. DOI: 10.1109/GRC.2006.1635905.

[123] Hansoo Lee, Jonggeun Kim, and Sungshin Kim. "Gaussian-Based SMOTE Algorithm for Solving Skewed Class Distributions". In: *International Journal of Fuzzy Logic and Intelligent Systems* 17.4 (2017), pp. 229–234.

[124] Georgios Douzas, Fernando Bacao, and Felix Last. "Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE". In: *Information Sciences* 465 (2018), pp. 1–20.

[125] S. Barua et al. "MWMOTE–Majority Weighted Minority Oversampling Technique for Imbalanced Data Set Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 26.2 (Feb. 2014), pp. 405–425. ISSN: 1041-4347. DOI: 10.1109/TKDE.2012.232.

[126] William A. Rivera. "Noise Reduction A Priori Synthetic Over-Sampling for class imbalanced data sets". In: *Information Sciences* 408 (2017), pp. 146–161. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2017.04.046. URL: http://www.sciencedirect.com/science/article/pii/S0020025517307089.

[127] Nitesh V Chawla et al. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[128] Fredy Rodríguez Torres, Jesús A. Carrasco-Ochoa, and José Fco. Martínez-Trinidad. "SMOTE-D a Deterministic Version of SMOTE". In: *Pattern Recognition*. Ed. by José Francisco Martínez-Trinidad et al. Cham: Springer International Publishing, 2016, pp. 177–188. ISBN: 978-3-319-39393-3.

[129] Forest Fang. *spark-knn*. https://github.com/saurfang/spark-knn. 2015.

[130] William C Sleeman IV and Bartosz Krawczyk. "Multi-class imbalanced big data classification on Spark". In: *Knowledge-Based Systems* (2020), p. 106598.

[131] Michael R. Smith, Tony R. Martinez, and Christophe G. Giraud-Carrier. "An instance level analysis of data complexity". In: *Mach. Learn.* 95.2 (2014), pp. 225–256.

[132] Ricardo B. C. Prudêncio. "Cost Sensitive Evaluation of Instance Hardness in Machine Learning". In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part II*. Vol. 11907. Lecture Notes in Computer Science. Springer, 2019, pp. 86–102.

[133] Jerzy Blaszczynski and Jerzy Stefanowski. "Neighbourhood sampling in bagging for imbalanced data". In: *Neurocomputing* 150 (2015), pp. 529–542.

[134] Felipe N. Walmsley et al. "An Ensemble Generation Method Based on Instance Hardness". In: *2018 International Joint Conference on Neural Networks, IJCNN 2018, Rio de Janeiro, Brazil, July 8-13, 2018*. 2018, pp. 1–8.

[135] Isaac Triguero et al. "ROSEFW-RF: The winner algorithm for the ECBDL'14 big data competition: An extremely imbalanced big data bioinformatics problem". In: *Knowl.-Based Syst.* 87 (2015), pp. 69–79.

[136] Alessio Benavoli et al. "Time for a Change: a Tutorial for Comparing Multiple Classifiers Through Bayesian Analysis". In: *J. Mach. Learn. Res.* 18 (2017), 77:1–77:36.

[137] F. Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[138] Sergey Bochkanov and Vladimir Bystritsky. "ALGLIB-a cross-platform numerical analysis and data processing library". In: *ALGLIB Project. Novgorod, Russia* (2011).

[139] Forest Fang. *spark-knn*. `https://github.com/saurfang/spark-knn`. Accessed: 12-14-2018. 2018.

[140] Alberto Fernández et al. "Foundations on imbalanced classification". In: *Learning from Imbalanced Data Sets*. Springer, 2018, pp. 19–46.

[141] Remote Sensing and GIS Program, Colorado State University. *Covertype data set*. Retrieved from: https://archive.ics.uci.edu/ml/datasets/Covertype. 1998.

[142] Montgomery County of Maryland. *Traffic violations*. Retrieved from: https://catalog.data.gov/violations-56dda. 2018.

[143] Surveillance, Epidemiology, and End Results (SEER) Program (www.seer.cancer.gov) Research Data (1975-2016), National Cancer Institute, DCCPS, Surveillance Research Program, released April 2019, based on the November 2018 submission.

[144] Intel Berkeley Research Lab. *Intel lab data*. Retrieved from: http://db.csail.mit.edu/labdata/la 2004.

[145] Yair Meidan et al. *N-baiot-network-based detection of iot botnet attacks using deep autoencoders*. 2018.

[146] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. "Searching for exotic particles in high-energy physics with deep learning". In: *Nature Communications* 5 (2014), p. 4308.

[147] Todd R. McNutt, Kevin L. Moore, and Harry Quon. *Big data needs and challenges for big data in radiation oncology*. 2016. DOI: 10.1016/j.ijrobp.2015.11.032.

[148] Reid F. Thompson et al. *Artificial intelligence in radiation oncology: A specialty-wide disruptive transformation?* Dec. 2018. DOI: 10.1016/j.radonc.2018.05.030.

[149] Hieu Trung Huynh et al. “Fully automated MR liver volumetry using watershed segmentation coupled with active contouring”. In: *International journal of computer assisted radiology and surgery* 12.2 (2017), pp. 235–243.

[150] Ardene Harris et al. “Splenic volume measurements on computed tomography utilizing automatically contouring software and its relationship with age, gender, and anthropometric parameters”. In: *European journal of radiology* 75.1 (2010), e97–e101.

[151] Cristian Lorenz and Jens von Berg. “A comprehensive shape model of the heart”. In: *Medical image analysis* 10.4 (2006), pp. 657–670.

[152] Florentino Luciano Caetano Dos Santos et al. “Fusion of edge enhancing algorithms for atherosclerotic carotid wall contour detection in computed tomography angiography”. In: *Computing in Cardiology 2014*. IEEE. 2014, pp. 925–928.

[153] Kavin R Jain and NC Chauhan. *Dental Image Analysis for Disease Diagnosis*. Springer, 2019. Chap. 2, pp. 21–38.

[154] Jean L. Wright et al. “Standardizing Normal Tissue Contouring for Radiation Therapy Treatment Planning: An ASTRO Consensus Paper”. In: *Practical Radiation Oncology* 9.2 (Mar. 2019), pp. 65–72. ISSN: 18798500. DOI: 10.1016/j.prro.2018.12.003.

[155] Stanley H Benedict et al. “Overview of the American Society for Radiation Oncology–National Institutes of Health–American Association of Physicists in Medicine Workshop 2015: Exploring opportunities for radiation oncology in the era of big data”. In: *International Journal of Radiation Oncology* Biology* Physics* 95.3 (2016), pp. 873–879.

[156] Lakshmi Santanam et al. "Standardizing naming conventions in radiation oncology". In: *International Journal of Radiation Oncology Biology Physics* 83.4 (July 2012), pp. 1344–1349. ISSN: 03603016. DOI: 10.1016/j.ijrobp.2011.09.054.

[157] Charles S Mayo et al. "American Association of Physicists in Medicine Task Group 263: standardizing nomenclatures in radiation oncology". In: *International Journal of Radiation Oncology* Biology* Physics* 100.4 (2018), pp. 1057–1066.

[158] Thilo Schuler et al. "Big Data Readiness in Radiation Oncology: An Efficient Approach for Relabeling Radiation Therapy Structures With Their TG-263 Standard Name in Real-World Data Sets". In: *Advances in radiation oncology* 4.1 (2019), pp. 191–200.

[159] D Rhee et al. "TG263-Net: A Deep Learning Model for Organs-At-Risk Nomenclature Standardization". In: *Medical Physics*. Vol. 46. 6. Wiley 111 River St, Hoboken 07030-5774, NJ USA. 2019, E263–E263.

[160] Qiming Yang et al. "A Novel Deep Learning Framework for Standardizing the Label of OARs in CT". In: *Workshop on Artificial Intelligence in Radiation Therapy*. Springer. 2019, pp. 52–60.

[161] William C Sleeman IV et al. "A Machine Learning method for relabeling arbitrary DICOM structure sets to TG-263 defined labels". In: *Journal of Biomedical Informatics* 109 (2020), p. 103527.

[162] Michael Hagan et al. "VA-Radiation Oncology Quality Surveillance Program". In: *International Journal of Radiation Oncology* Biology* Physics* (2020).

[163] Henry Lieberman. "How to color in a coloring book". In: *ACM SIGGRAPH Computer Graphics*. Vol. 12. 3. ACM. 1978, pp. 111–116.

[164] Mirek Fatyga, Baoshe Zhang, and William C Sleeman. "Designing and implementing a computing framework for image-guided radiation therapy research". In: *Computing in Science & Engineering* 14.4 (2011), pp. 57–68.

[165] Uwe Schneider, Eros Pedroni, and Antony Lomax. "The calibration of CT Hounsfield units for radiotherapy treatment planning". In: *Physics in Medicine & Biology* 41.1 (1996), p. 111.

[166] R. Bellman, Rand Corporation, and Karreman Mathematics Research Collection. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957. ISBN: 9780691079516.

[167] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[168] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. "Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions". In: *SIAM Review* 53.2 (2011), pp. 217–288.

[169] Charles Elkan. "Boosting and naive Bayesian learning". In: *Proceedings of the International Conference on Knowledge Discovery and Data Mining*. 1997.

[170] Geoffrey E Hinton. "Connectionist learning procedures". In: *Machine learning*. Elsevier, 1990, pp. 555–610.

[171] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[172] Chih-Wei Hsu and Chih-Jen Lin. "A comparison of methods for multiclass support vector machines". In: *IEEE transactions on Neural Networks* 13.2 (2002), pp. 415–425.

[173] Kwok Tai Chui and Miltiadis D Lytras. "A novel MOGA-SVM multinomial classification for organ inflammation detection". In: *Applied Sciences* 9.11 (2019), p. 2284.

[174] Philippe Lambin et al. "Radiomics: extracting more information from medical images using advanced feature analysis". In: *European journal of cancer* 48.4 (2012), pp. 441–446.

[175] Bin Liang et al. "Dosiomics: extracting 3D spatial features from dose distribution to predict incidence of radiation pneumonitis". In: *Frontiers in oncology* 9 (2019), p. 269.

[176] Robert J Gillies, Paul E Kinahan, and Hedvig Hricak. "Radiomics: images are more than pictures, they are data". In: *Radiology* 278.2 (2016), pp. 563–577.

[177] W Sleeman et al. "Relabeling Non-Standard to Standard Structure Names Using Geometric and Radiomic Information". In: *Medical Physics*. Vol. 47. 6. Wiley 111 River St, Hoboken 07030-5774, NJ USA. 2020, E438–E438.

[178] Alex Zwanenburg et al. "The image biomarker standardization initiative: standardized quantitative radiomics for high-throughput image-based phenotyping". In: *Radiology* 295.2 (2020), pp. 328–338.

[179] W Sleeman et al. "Using CNNs to Extract Standard Structure Names While Learning Radiomic Features". In: *Medical Physics*. Vol. 48. 6. Wiley 111 River St, Hoboken 07030-5774, NJ USA. 2021.

[180] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. "Multimodal machine learning: A survey and taxonomy". In: *IEEE transactions on pattern analysis and machine intelligence* 41.2 (2018), pp. 423–443.

[181] Jing Zhao et al. "Multi-view learning overview: Recent progress and new challenges". In: *Information Fusion* 38 (2017), pp. 43–54.

[182] Yingming Li, Ming Yang, and Zhongfei Zhang. "A survey of multi-view representation learning". In: *IEEE transactions on knowledge and data engineering* 31.10 (2018), pp. 1863–1883.

[183] Jingqi Song et al. "Multiview multimodal network for breast cancer diagnosis in contrast-enhanced spectral mammography images". In: *International Journal of Computer Assisted Radiology and Surgery* 16.6 (2021), pp. 979–988.

[184] Matthias Guggenmos et al. "A multimodal neuroimaging classifier for alcohol dependence". In: *Scientific reports* 10.1 (2020), pp. 1–12.

[185] Giuseppe Aceto et al. "MIMETIC: Mobile encrypted traffic classification using multimodal deep learning". In: *Computer Networks* 165 (2019), p. 106944.

[186] Charles A Ellis et al. "Explainable Sleep Stage Classification with Multimodal Electrophysiology Time-series". In: *bioRxiv* (2021).

[187] Aili Shen et al. "A joint model for multimodal document quality assessment". In: *2019 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. IEEE. 2019, pp. 107–110.