SN Computer Science (2023) 4:462
https://doi.org/10.1007/s42979-023-01880-4

ORIGINAL RESEARCH

# Exploring Maximum Tree Depth and Random Undersampling in Ensemble Trees to Optimize the Classification of Imbalanced Big Data

John T. Hancock III[1] · Taghi M. Khoshgoftaar[1]

Received: 22 December 2022 / Accepted: 8 May 2023

**Abstract**
We present findings from experiments in Medicare fraud detection, that are the result of research on two new, publicly available datasets. In this research, we employ popular, open-source Machine Learning algorithms to identify fraudulent healthcare providers in Medicare insurance claims data. As far as we know, we are the first to publish a study that includes datasets compiled from the latest Medicare Part B and Medicare Part D data. The datasets became available in 2021, and are the largest such datasets that we know of. We report details on two important findings. The first finding is that increased maximum tree depth is associated with the best performance in terms of area under the receiver-operating characteristic curve (AUC) for both datasets. The second finding, which is an important counterbalance to the first finding, is that one may utilize random undersampling (RUS) to reduce the size of the training data and simultaneously achieve similar or better AUC scores.To the best of our knowledge, our study is novel in reporting the importance of maximum tree depth for classifying imbalanced Big Data. Moreover, this work is unique in demonstrating that one may employ RUS to mitigate the increased resource consumption of higher maximum tree depth.

**Keywords** Bagging · Boosting · Class imbalance · Big data · Random undersampling

## Introduction

The exploration of new data is fertile ground for the development of contributions to an application domain. New Medicare insurance claims data became publicly available in 2021. These data, which we use in our study, come from two related sources. The first source is *Medicare Physician & Other Practitioners—by Provider and Service* (Part B) [1]. The second source is *Medicare Part D Prescribers—by Provider and Drug* (Part D) [2] insurance claims data. The datasets we compile from the Part B and Part D data are highly imbalanced Big Data. Our Part B dataset has approximately 68 million training instances with a minority-to-majority class ratio of approximately 0.0019, and our Part D dataset has approximately 173 million instances, with a minority-to-majority class ratio of approximately 0.0039. With all the facts available to us, we claim that this is the first study to contain findings that cover the latest Part B and Part D data in a single study on Medicare fraud detection as a supervised Machine Learning task.

Medicare is the United States' public health insurance program. It provides health insurance for millions of Americans aged 65 and over, as well as those with certain disabilities. Medicare insurance fraud detection is a worthwhile pursuit, because facts indicate that a large amount of money could be recovered. Once recovered, it could be spent on providing more extensive healthcare to Medicare beneficiaries. The Centers for Medicare and Medicaid Services (CMS) are the United States government departments responsible for Medicare. In 2019, the CMS provided an estimate that it made approximately $100 billion in improper payments [3]. In the same year, the United States Department of Justice published a report stating it recovered approximately

This article is part of the topical collection "Recent Trends on AI for HealthCare" guest edited by Lydia Bouzar-Benlabiod.

✉ John T. Hancock III
jhancoc4@fau.edu

Taghi M. Khoshgoftaar
khoshgof@fau.edu

1 Department of Electrical Engineering and Computer Science, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431, USA

Published online: 22 June 2023

$3 billion prosecuting insurance fraud [4]. The CMS uses the term improper payments to cover payments made due to fraud and other errors. However, it is reasonable to assume that, due to the amount of estimated improper payments, more fraud could be detected. Therefore, Medicare fraud detection is a fitting application domain for the field of Machine Learning, since it is a suitable tool for processing the Big Data repository of Medicare Insurance claims information.

The contributions we make in this research on the subject of classifying highly imbalanced Big Data for Medicare fraud detection concern three key concepts: maximum tree depth, Random Undersampling (RUS), and Area Under the Receiver-Operating Characteristic Curve (AUC) [5]. These contributions are an expansion of work we published previously [6]. Maximum tree depth refers to the longest allowable path from root to leaf node in a decision tree. It is an adjustable parameter in the decision-tree-based ensemble classifiers we use. The classifiers we use are: CatBoost [7], XGBoost, [8], Random Forest [9], and Extremely Randomized Trees (ET) [10]. RUS is a technique to improve classification results when working with data that has a low minority-to-majority class ratio. Such data are known as imbalanced data. To perform RUS, one chooses a minority-to-majority class ratio, and then randomly discards instances of the majority class until the remaining data have the desired class ratio. AUC is a metric for measuring the performance of a classifier. It is calculated by varying the classification decision threshold from zero to one in small increments and plotting the false-positive rate versus the true-positive rate that the classifier yields for each value of the decision threshold. Once the points are plotted, a curve is formed, and AUC is the area under this curve. A perfect classifier yields an AUC score of 1.0, and a classifier that assigns instances to classes randomly yields an AUC score of approximately 0.5. These definitions of RUS, maximum tree depth, and AUC are key to understanding the contributions in this work.

This body of research makes two contributions to the field of research in classifying imbalanced Big Data. The first contribution of our research is to show that maximum tree depth in decision tree-based ensemble classifiers is a highly effective parameter to optimize AUC scores for Medicare fraud detection. We show that optimizing this parameter yields AUC scores over 0.98 in some cases. We vary maximum tree depth in our experiments over a wide range of values to show its substantial effect on experimental outcomes. Moreover, for each experiment, we use one, and only one, value for maximum tree depth.

The second contribution we make is to show that one may apply RUS to the Part B and Part D data and build Machine Learning models that yield AUC scores that are similar to, or better than AUC scores of models built with the full datasets. This is an important finding, because it allows for faster execution of model training. Since the Part B and Part D data are highly imbalanced, when we apply RUS to induce larger class ratios in the model training data, we reduce the size of the training data. Training the popular, open-source Machine Learning algorithms we employ is faster with smaller training data. This finding is a boon to researchers working in the field, since it enables one to conduct more experiments with the Part B and Part D data. The sections that follow this introduction are: Related Work, Algorithms, Methodology, Results, Statistical Analysis, and Conclusions.

## Related Work

In this section, we provide background on research leading up to our study, and make the case for the novelty of our work. Ensemble decision tree-based Machine Learning techniques, the effect of maximum tree depth, RUS, and Big Data are the essential subjects of our study. Therefore, related works pertain to these concepts. Many of the studies were not conducted with datasets on the scale of the datasets we use. Moreover, in our search for related work, we found similar studies, but none that show the impact of maximum tree depth and RUS on AUC scores to the extent that we do, none use CatBoost for encoding categorical features, and none contain results of experiments performed with the latest Part B and Part D data.

In their 2018 study on data sampling and imbalanced big data, Bauder et al. compare multiple sampling techniques for Medicare fraud detection [11]. They compile a dataset by combining the Part B, Part D, and one additional Medicare dataset known as the "Medicare Durable Medical Equipment, Devices & Supplies—by Referring Provider and Service" (DMEPOS) dataset [12]. Because their study was published in 2018, a smaller amount of Medicare claims data were available than what we experiment with here. Therefore, their combined dataset has fewer than 1 million instances. They employ versions of Random Forest, Logistic Regression [13], and Gradient Boosted Trees [14] for the Apache Spark environment [15]. In their findings, Bauder et al. report experimental outcomes for six data sampling techniques: RUS, Random Oversampling (ROS), Synthetic Minority Oversampling Technique (SMOTE), two adaptations of borderline SMOTE [16], and Adaptive Synthetic Sampling Approach for Imbalanced Learning (ADASYN). Of the six techniques, Bauder et al. report that applying RUS to the training data fed to their classifiers yields the best performance. For these reasons, we use RUS as well. On the other hand, Bauder et al. do not show the impact of maximum tree depth on AUC scores for Medicare fraud

Detection. In addition, they do not employ CatBoost encoding as we do to handle categorical features.

"An ensemble random forest algorithm for insurance big data analysis" by Lin et al. is a related study [17], because it involves Random Forest, sampling techniques, and classifying imbalanced data. In their study, the authors propose a variation on the Random Forest algorithm for predicting the likelihood that a consumer will purchase life insurance. They compare the performance of models built from training data with, and without SMOTE applied to address class imbalance. However, in their study, details on the class ratios induced by SMOTE are not apparent. Their study appears to be more concerned with the impact of their sampling technique on running time. We find that RUS has a negligible impact on running time, and therefore focus on its impact on classification results. We did not find details on the features of Lin et al.'s dataset, such as whether it has categorical features. One useful feature of our study is that we document the use of CatBoost encoding [7], a technique for handling categorical features that is practical for large datasets. The dataset Lin et al. work with has approximately 500,000 instances and 16 attributes. Our study involves two much larger datasets. Moreover, we do not aim to propose a new variation on Random Forest. We use a publicly available, open-source version of Random Forest. For these reasons, our study is set apart from the one done by Lin et al.

A second study which involves Random Forest and Big Data is by Del Río et al. [18]. The dataset they use has approximately 6 million instances and 41 attributes. Therefore, it is on a smaller scale than what we work with here. Their study also does not concern maximum tree depth. Rather than experiment with maximum tree depth, Del Río et al. use one maximum tree depth setting for all experiments. Therefore, it is not a factor which can be analyzed for effect as we do here. While Del Río et al. document that RUS is applied to their data, it is only applied to induce a 1:1 class ratio. Here, we document the application of RUS to induce five class ratios and show its effect on experimental outcomes. Therefore, the key differences between our study and Del Río et al.'s are our treatment of maximum tree depth, and RUS level as experimental factors.

Herrera et al. [19] published a related work that explores the impact of maximum tree depth with a classifier in common with one that we use. This is another study where Random Forest is employed to classify the so-called Big Data. However, the dataset Herrera et al. conduct their experiments with contains 581,012 training instances, which is much smaller than the datasets we work with. Furthermore, we note that Herrera et al. use only a single dataset, whereas we present results covering two datasets. The focus of Herrera et al.'s study is a novel implementation of Random Forest for a high-performance computing environment. Hence, their interest in maximum tree depth is its impact on the running time of their implementation. Our focus on maximum tree depth is its impact on AUC scores for classifying Medicare insurance claims data.

In their 2017 study, Genuer et al. evaluate multiple Random Forest variants' performance to classify a dataset with approximately 120 million instances [20]. They compare the performance of five variants of Random Forest classifiers in terms of prediction error. Subsampling is an important term in their study, but it is not a technique for addressing class imbalance. Genuer et al. present subsampling as a technique that is a part of building Random Forest models. We cover RUS, which is a sampling technique for addressing class imbalance. Moreover, Genuer et al.'s study concerns subsampling in conjunction with variations on Random Forest, whereas we present results from experiments combining RUS with Random Forest and other classifiers as well.

Fauzan and Murfi perform experiments with XGBoost and an insurance company's customer data to forecast whether the customer will file an insurance claim in [21]. The customer data comprise a dataset of approximately 1.5 million instances, with 57 attributes. As part of hyperparameter tuning, Fauzan and Murfi vary maximum tree depth between four and five. We take a more in-depth look at maximum tree depth, and we look at a broader range of maximum tree depths. In our experiments, maximum tree depth takes the values from 6, 16, 24, 32, and 48. Moreover, our experiments involve other learners, Random Forest, CatBoost, and ET in addition to XGBoost. Similar to Genuer et al., Fauzan and Murfi document experiments with subsampling, but we do not find that they experiment with RUS as a technique for addressing class imbalance in Big Data.

In their research on the use of XGBoost to predict loan default, Li et al. employ hyperparameter tuning to optimize results [22]. While they document that they use a grid search method to do hyperparameter tuning, they do not cover the impact of maximum tree depth on classification results. Furthermore, Li et al. use a dataset that has approximately 130,000 instances and 143 features. Therefore, their study does not probe the impact of maximum tree depth on the classification of Big Data in the manner that our study does. One similarity between our study and Li et al. is the use of RUS. Their dataset is imbalanced, with a 1:22 minority-to-majority class ratio, and they write that they undersample their data to induce a 1:4 class ratio. We experiment with a wider range of class ratios and provide analysis of the effect of RUS on the classification of two larger datasets.

In a study with data on a scale more characteristic of Big Data, Wang et al. document the performance of a classifier

**Table 1** LEIE exclusion codes and rules

| Rule number | Description |
|---|---|
| 1128(a)(1) | Conviction of program-related crimes |
| 1128(a)(2) | Conviction relating to patient abuse or neglect |
| 1128(a)(3) | Felony conviction relating to health care fraud |
| 1128(b)(4) | License revocation or suspension |
| 1128(b)(7) | Fraud, kickbacks and other prohibited activities |
| 1128(c)(3)(g)(i) | Conviction of two mandatory exclusion offenses 10 years |
| 1128(c)(3)(g)(ii) | Conviction of three mandatory exclusion offenses indefinite |

built from a combination of logistic regression and XGBoost [23]. The dataset they use has approximately 50 million instances. The aim of their study is to show the performance of their proposed classifier in predicting the users' activity in e-commerce websites. While Wang et al.'s research involves XGBoost and Big Data, it does not explore the effectiveness of maximum tree depth or RUS in the classification task. Another difference between the studies is that, for easy repeatability, we use popular, open-source classifiers without modification. These facts differentiate our studies.

In 2019, Johnson and Khoshgoftaar published a study which involves data sampling and Medicare fraud detection with deep learning classifiers [24]. However, their study involves Medicare insurance fraud data which does not include the additional data that became publicly available in 2021 that we use here. Furthermore, Johnson and Khoshgoftaar use a different technique for handling the categorical attributes of the Medicare insurance claims data. They aggregate the data by healthcare provider, and replace some categorical data with descriptive statistics of the numeric data in the records that they aggregate over. Since the dataset they work with is aggregated, it is smaller than the datasets we work with here. In their study, Johnson and Khoshgoftaar experiment with a dataset that has approximately 6 million instances. Another key difference between the two studies is that we use decision tree-based classifiers, whereas Johnson and Khoshgoftaar use deep learning classifiers.

In our review of related work, we find opportunities to make contributions. There are many studies that discuss different components of our study, but none that reveal the synergy they provide when taken together. We find studies that claim to involve Big Data, but the data used are not on the scale of the data we use. We find studies that mention maximum tree depth, but do not systematically investigate the impact of maximum tree depth on the classification of imbalanced Big Data. To the best of our knowledge, this study is novel, because it covers experiments performed on two new, highly imbalanced Big Data datasets, with a unique methodology to show the impact of RUS and maximum tree depth on AUC scores.

## Datasets

We use data provided by the CMS. CMS collected the Part B and Part D data from 2013 to 2019, and published it in 2021. To the best of our knowledge, these datasets cover the longest period of time used in any study on Machine Learning for Medicare fraud detection. Moreover, the attributes of the data are different from those used in the previous studies. The Part B and Part D data are not immediately suitable for supervised Machine Learning. For example, the instances of the dataset must be labeled.

For both of the Part B and Part D datasets, we use the same logic to label them. The data for the labels are published by another department of the United States Government, the Office of the Inspector General (OIG). The publication is known as the List of Excluded Individuals and Entities (LEIE) [25]. The LEIE is updated monthly. The individuals and entities appearing in the LEIE are excluded from government backed healthcare programs, including Medicare. In the LEIE, the OIG provides categories of exclusions along with identifying information. For the application domain of Medicare fraud detection, we select the records of the LEIE that contain the same exclusion categories as documented by Herland et al. [26]. Table 1 is a copy of the table from Herland et al. that specifies which exclusion rule codes we use to identify the records of fraudulent providers.

The identifying information of healthcare providers in the LEIE includes a National Provider Identifier (NPI). Records of the Part B and Part D data also contain an NPI. Therefore, we can match NPIs in the Part B and Part D data with the LEIE data, and label instances of the Part B and Part D data as fraudulent when there is a match, and the year value of the Part B or Part D record falls within the period of time when fraudulent claims may have been submitted. We define that period of time as follows: fraudulent claims may have been submitted any time before the start of the exclusion period until the end of the exclusion period. However, the year is the most specific time-related information we have in the Part B and Part D data, whereas the year and month is the most specific time-related information we have in the LEIE

**Table 2** Features of the Part B dataset

| Feature name | Description |
|---|---|
| Rndrng_Prvdr_Crdntls | The referring provider's credentials [example: MD]; categorical, 23,672 distinct values |
| Rndrng_Prvdr_Gndr | The referring provider's gender; categorical, three distinct values |
| Rndrng_Prvdr_Ent_Cd | Type of entity [individual or organization] reported in NPPES; categorical, 2 distinct values |
| Rndrng_Prvdr_Type | Derived from the Medicare provider/supplier specialty code reported on all of the NPI's Part B non-institutional claims (DMEPOS and non-DMEPOS); categorical, 204 distinct values |
| Rndrng_Prvdr_Mdcr_Prtcptg_Ind | Identifies whether the provider participates in Medicare and/or accepts assignment of Medicare allowed amounts; categorical two distinct values |
| HCPCS_Cd | HCPCS code used to identify the specific medical service furnished by the provider; categorical 7738 distinct values |
| HCPCS_Desc | Description of the HCPCS code for the specific medical service furnished by the provider; categorical, 8252 distinct values |
| HCPCS_Drug_Ind | Identifies whether the HCPCS code for the specific service furnished by the provider is a HCPCS listed on the Medicare Part B Drug Average Sales Price (ASP) File; categorical, two distinct values |
| Place_Of_Srvc | Identifies whether the place of service submitted on the claims is a facility (value of "F") or non-facility (value of "O"); categorical two distinct values |
| Tot_Benes | Number of distinct Medicare beneficiaries receiving the service for each Rndrng_NPI, HCPCS_Cd, and Place_Of_Srvc |
| Tot_Srvcs | Number of services provided; note that the metrics used to count the number provided can vary from service to service |
| Tot_Bene_Day_Srvcs | Number of distinct Medicare beneficiary/per day services |
| Avg_Sbmtd_Chrg | Average of the charges that the provider submitted for the service |
| Avg_Mdcr_Alowd_Amt | Average of the Medicare allowed amount for the service |
| Avg_Mdcr_Pymt_Amt | Average amount that Medicare paid after deductible and coinsurance amounts have been deducted for the line item service |
| Avg_Mdcr_Stdzd_Amt | Average amount that Medicare paid after beneficiary deductible and coinsurance amounts have been deducted for the line item service and after standardization of the Medicare payment has been applied |

data. Therefore, we round the end of the exclusion period to the nearest year listed in the LEIE. For example, if the LEIE data contain a record stating that a provider's exclusion period ends in March 2017, then we label the provider's records as fraudulent if their year value is less than 2017. However, if the exclusion period ends in July 2017, we label the provider's data as fraudulent if the year value is less than 2018. If the NPI of a provider in the Part B or Part D data does not exist in the LEIE data, or the year of the record is after the end of the period when fraudulent claims were possible, then we label the instance as not fraudulent.

Once a provider's exclusion period is over, the provider is removed from the LEIE. Therefore, one must obtain the previous editions of the LEIE to completely label the Part B or Part D data. These previous editions are available in the Internet Archive.[1]

Although we use the same labeling process for both datasets, they have characteristics that make them unique. Records of the Part B dataset contain information on the treatments and procedures that healthcare providers administer to their patients for 1 year. One important acronym for understanding which treatment or procedure is specified in the record is HCPCS, which stands for Healthcare Common Procedure Coding System. Every record has an HCPCS code, which represents a specific treatment or procedure. We use the HCPCS code as a categorical feature. All the fields in the Part B data that we use as independent variables are defined in Table 2. We have copied the definitions of the features from the Part B Data Dictionary [27]. In addition, we augment the definitions in the data dictionary with the number of distinct values of the categorical features. CMS provides the NPI, name, and address of the provider in the Part B data. We discard this information, since these fields are unique identifiers and may hinder a Machine Learning model's ability to generalize. When the labeling process is complete, our Part B dataset contains 67,856,547 records, with a minority-to-majority class ratio of approximately 0.0019.

The Part D data are similar to the Part B data. However, instead of information on treatments and procedures, the Part D data have information on medications that healthcare providers prescribe for their patients. There is one record in the Part D data for every combination of provider, year, and medication. Medications are identified by brand name and generic name. We use these as categorical features. Similar to the Part B data, the Part D data also contain NPIs,

[1] http://archive.org/web.

**Table 3** Features of the Part D dataset

| Feature name | Description |
|---|---|
| Prscrbr_Type | Derived from the Medicare provider/supplier specialty code; categorical, 249 distinct values |
| Prscrbr_Type_Src | Source of the Medicare provider/supplier specialty code; categorical, 2 distinct values |
| Brnd_Name | Brand name (trademarked name) of the drug filled; categorical, 3907 distinct values |
| Gnrc_Name | A term referring to the chemical ingredient of a drug rather than the trademarked brand name under which the drug is sold; categorical, 2272 distinct values |
| Tot_Clms | The number of Medicare Part D claims |
| Tot_30day_Fills | The aggregate number of Medicare Part D standardized 30-day fills |
| Tot_Day_Suply | The aggregate number of day's supply for which this drug was dispensed |
| Tot_Drug_Cst | The aggregate drug cost paid for all associated claims |
| Tot_Benes | The total number of unique Medicare Part D beneficiaries with at least one claim for the drug |

names, and addresses of providers, which we discard to aid our models' generalization. We list the fields in the Part D data that we use as features in Table 3. The definitions of the features are from the CMS Part D Data Dictionary [28]. Once we have completed labeling the Part D data, we have a dataset with 173,677,665 records. The minority-to-majority class ratio of the dataset is approximately 0.0039.

## Algorithms

Our study concerns tree-based algorithms only. Moreover, the focus of our study is on the impact of maximum tree depth and RUS on the classifier's ability to execute Medicare fraud detection. We employ publicly available open-source classifiers. This enables easily reproducible results, since the software we use in our experiments is readily available. As we stated in the introduction, the classifiers we use in this study are: CatBoost, XGBoost, ET, and Random Forest. All of these classifiers are implementations of ensemble techniques that leverage collections of decision trees.

Of the four classifiers, Random Forest is the simplest. It leverages a technique known as Bagging. Breiman coined the term Bagging in a study published in 1996 [29]. One may apply Bagging to both classification and regression tasks. Since our study involves classification, we focus on how bagging may be applied for classification. Bagging relies on a sampling technique known as bootstrap sampling. A bootstrap sample is a sample taken with replacement. To apply Bagging, one trains an ensemble of the same classifiers on different bootstrap samples of the training data. After the classifiers are trained, one may use them for classification. Bagging specifies that we treat the classifiers' output as votes for the classification outcome. The class assigned to an instance is the class that the majority of the members of the ensemble assign the instance to. Some informal reasoning about the probability of a correct result explains the appeal of Bagging. Suppose the probability of a correct classification of one of the learners is greater than one half. Then, as the size of the ensemble of learners increases, the probability that more than half of the learners will agree on the correct class of an instance will increase in proportion with the size of the ensemble.

In 2001, Breiman published the seminal work on Random Forest, a study where the Bagging technique is applied to ensembles of decision trees [9]. Moreover, Random Forest has an important feature that is separate from the Bagging technique, and applies to the decision trees that constitute a Random Forest ensemble. The feature has to do with the decision tree split-finding process that is executed when the decision trees are trained on the bootstrap samples. A split in a decision tree is the value used in a node of a decision tree to decide which edge to traverse out of the node. Split finding is the process of finding the best value to compare a feature of an instance to, to select the correct edge to follow in the decision tree. Random Forest's innovation to split finding is in the initial selection of the feature to use for comparison. Random Forest randomly samples from a subset of the features in a dataset as a means to select the feature for comparison. Recently, a Random Forest implementation for Graphics Processing Units (GPUs) became publicly available.[2] We utilize this implementation of Random Forest, since it has a much lower training time relative to other Random Forest implementations.

ET is a classifier that is closely related to Random Forest. Geurts et al. published the seminal work on ET in 2006 [10]. ET functions similarly to Random Forest, with one key difference: splits in decision trees are selected randomly. In the other three classifiers we use, splits in decision trees are selected systematically. The motivation for random selection of splits is that it can be faster than an optimization technique for calculating the best value for a split. Our results show that this technique of randomly selecting values to use for

[2] https://docs.rapids.ai/api/cuml/stable/api.html#random-forest.

splits may yield results that are similar to, or better than the other classifiers that use more sophisticated split-finding techniques.

In contrast to the bagging technique that ET and Random Forest apply, CatBoost and XGBoost are gradient boosted machines (GBMs). For an additional study comparing CatBoost and XGBoost's performance in the classification of imbalanced Big Data; please see [30]. Friedman introduced the concept of GBMs in 2001 [31]. Like Bagging-based classifiers, GBMs are ensembles of learners. However, the ensembles are formed differently in the two techniques. GBM ensembles are formed iteratively. We start with a single learner, which yields a set of estimates $\hat{\mathbf{y}}$ of the target $\mathbf{y}$. Therefore, we may define a vector of residual values $\mathbf{e} = L(\mathbf{y}, \hat{\mathbf{y}})$, where $L$ is a loss function. The next step in forming the ensemble is then to add a learner that uses the training data to estimate the vector $\mathbf{e}$. We then use a combination of the output of the first two learners to find a new vector of estimates, which is closer to the vector of target values, that yields a new vector of residuals, with which we can apply the same process to add a third learner to the ensemble, and so on until we detect overfitting or reach a pre-determined number of iterations. This iterative process to form an ensemble of learners characterizes GBMs.

Chronologically, XGBoost was introduced before CatBoost. Chen and Guestrin published the initial work on XGBoost in 2016 [8]. In their study, Chen and Guestrin describe XGBoost as having enhancements to GBMs that make it an attractive candidate for Machine Learning applications. They explain that the enhancements in XGBoost provide scalability. In the context of their paper, scalability means that XGBoost operates efficiently on large datasets. They go on to explain that XGBoost achieves scalability through five techniques: sparsity awareness, approximate tree learning, effective cache access, data compression, and sharding. Sparsity awareness refers to a capability built into XGBoost that enables XGBoost to detect when data have mostly a constant value interspersed with anomalies, and to more quickly determine values for splits in decision trees when this condition in the data is detected. Approximate Tree Learning is Chen and Guestrin's name for the implementation of their weighted quantile sketch algorithm, which is another technique XGBoost employs for efficiently determining decision tree splits. Cache-awareness refers to a data storage and retrieval strategy, built in to XGBoost, that optimizes CPU cache utilization. Data compression is another strategy for efficiently using system resources that XGBoost employs to achieve scalability. Compressed data can be read from a disk and decompressed in less time than the same data in its uncompressed form can be read from disk. XGBoost leverages this fact to handle processing datasets more efficiently. The final enhancement XGBoost adds to GBMs is data sharding. During model fitting, XGBoost generates intermediate data, and stores the data in blocks, which may then be distributed in separate storage media, so that input/output operations on the data are executed in parallel. We can attest to Chen and Guestrin's claims about the scalability of XGBoost, since we are able to use it to efficiently classify the Part B and Part D Big Data.

CatBoost is the fourth classifier we use in our study. Like XGBoost, the authors of the seminal work on CatBoost, Prokhorenkova et al., introduce CatBoost as a GBM implementation with enhancements. For an extensive survey of CatBoost's applications, please see [32]. However, the enhancements they design for CatBoost are focused on preventing overfitting in two ways. The first way CatBoost guards against overfitting is with the Ordered Boosting technique. Ordered Boosting is a method for selecting samples for training and evaluating trees to prevent overfitting. In Ordered Boosting, a set of candidate trees are trained on distinct samples of the training data. CatBoost then selects the best candidate based on its ability to estimate the independent variable from a sample of the training data that is not used to train any of the candidate trees. This is how Ordered Boosting protects against overfitting. The second precaution against overfitting that CatBoost offers is Ordered Target Statistics. Ordered Target statistics is a technique for encoding categorical features that is conceptually similar to Ordered Boosting, due to its emphasis on sample selection. In the Ordered Target Statistics encoding technique, the encoded value of a categorical feature is computed from the target (dependent variable) value that it co-occurs with. However, it is not permitted to use the target value from the same instance the categorical feature appears with. This prevents the encoded value from directly depending on the target value of the instance. Prokhorenkova et al. define this type of dependence as "target leakage". Target leakage is similar to the overfitting that can occur when a model memorizes output values associated with unique identifiers, and therefore fails to generalize.

## Methodology

We use the Machine Learning algorithms discussed in the previous section to perform the experiments on the Part B and Part D data we described above. All four of the classifiers have implementations as libraries for the Python programming language [33]. All the libraries are available for download on the Internet, free of cost.

Machine Learning algorithms are stochastic in nature. Put another way, they tend to produce different results unless initial conditions for learning are kept exactly the same. Robust performance under the variance of initial conditions indicates a task that is suitable for Machine Learning. Therefore, we execute ten iterations of fivefold cross-validation with a recorded sequence of different seeds for the random number generators used in our experiments to ensure such variance in the initial conditions.

Every round of fivefold cross-validation yields one AUC score. Therefore, we collect a total of 50 AUC scores over the ten iterations of fivefold cross-validation. This provides an ample number of samples for calculating summary statistics of the results as well as performing statistical tests and analyses.

Just as we have libraries that provide CatBoost, XGBoost, Random Forest, and ET, we have a library, scikit-learn [34], that facilitates fivefold cross-validation and evaluation of the AUC scores. The use of these libraries also facilitates the repeatability of our results.

Researchers have many options for encoding categorical features. For a domain-specific technique for encoding Medicare Big Data, see [35]. In this study, we use CatBoost encoding. Readers interested in a comprehensive investigation of encoding techniques should explore [36].

Encoding of categorical features and RUS are steps in our experiments. The order in which they are performed in conjunction with fivefold cross-validation is important. In the program for running our experiments, we first shuffle and split the data into training and testing subsets. We then fit a CatBoost encoder on the training data, to encode the categorical features. The CatBoost encoder is part of the category-encoders library. After fitting the CatBoost encoder to the training data, we then encode the categorical features in the training and the test datasets. Then, for experiments where RUS is required, we use the RandomUnderSampler object from the imblearn library to induce a specified class ratio. After encoding, and applying RUS to the training data, we fit CatBoost, XGBoost, ET, or Random Forest to it. Then, we evaluate the fitted classifier's AUC score on the test data.

One important note for reproducibility is that we use GPU implementations of CatBoost, XGBoost, and Random Forest. In preliminary experiments, we found these versions of the classifiers to process the Part B and Part D data much more quickly than their equivalent CPU implementations.

We conduct experiments over a number of combinations of classifier, RUS level, and maximum tree depth. Different classifiers support different levels of maximum tree depth, so it is not possible to test all combinations. The maximum tree depth of 6 is the default value of maximum tree depth for CatBoost and XGBoost [37, 38]. Initially, we selected maximum tree depths values of 6, 16, 24, 32, and 48 to get a reasonable spread of maximum tree depth values between the lowest default value of maximum tree depth of 6, and the largest maximum tree depth we estimated would be practical to use with our system, 48. Since we are working with large datasets, in the interest of time, we opted not to run experiments on maximum tree depths for classifiers if a pattern of improving AUC scores was already established. Hence, for Random Forest, we executed experiments with maximum tree depths of 16, 24, 32, and 48. CatBoost only supports maximum tree depth values up to 16, so 6 and 16 are the only maximum tree depths we use with CatBoost. XGBoost supports all the levels of maximum tree depth we wish to use. However, we encountered runtime errors due to memory limitations for maximum tree depth values beyond 24. Therefore, our experimental results include maximum tree depth values of 6, 16, and 24, for XGBoost. ET has unlimited maximum tree depth by default. Therefore, it was only necessary to conduct experiments with ET with maximum tree depths of 16, 24, and the unlimited, default, maximum tree depth settings to establish the pattern that ET's AUC scores improve in proportion with maximum tree depth. Table 4 contains all the non-default values of hyperparameters that we use in our experiments.

Any of the classifiers are compatible with all the levels of RUS used in our experiments. We use RUS to induce class ratios of 1:1, 1:3, 1:9, 1:27, and 1:81. Therefore, we include combinations of classifier, maximum tree depth, and the five levels of RUS. The levels of RUS used are the same we used in [39]. In addition to the experiments where RUS is applied, we also conduct experiments where RUS is not applied to determine whether results are better without RUS.

The system we use to run our experiments is a high-performance computing environment. The environment consists of 16 nodes. Each node is equipped with a 16 core Intel Xeon CPU, 256 GB RAM, and an Nvidia V100 GPU.

## Results

Table 5 contains the results of the initial experiments that we conducted with the Part B and Part D data. Our key observation in the results is that the classifiers with higher maximum tree depth yield better performance. In our initial experiments, we used default maximum tree depths for all classifiers. CatBoost and XGBoost both have a default maximum tree depth of 6. We see that the AUC scores that CatBoost and XGBoost yield are the lowest. ET's default setting for maximum tree depth is unlimited, and we see that it yields the highest AUC scores. Furthermore, the Random Forest implementation we use has a default maximum tree depth of 16, and the AUC scores it yields are in between CatBoost's, XGBoost's, and ET's. These initial results establish a clear pattern that the AUC score is in proportion to the maximum tree depth. We sought to confirm the pattern with further experiments.

**Table 4** Changed hyperparameter settings

| Classifier/hyperparameter | Setting | Description |
|---|---|---|
| CatBoost | | |
| task_type | GPU | Use GPU(s) for execution |
| Devices | 0 | GPU device ID |
| max_ctr_complexity | 1 | Maximum number of features to combine |
| max_depth | 6, 16 | Maximum tree depth |
| n_estimators | 100 | Number of trees in ensemble |
| XGBoost | | |
| tree_method | gpu_hist | Use GPU(s) for execution |
| gpu_id | 0 | GPU device ID |
| n_estimators | 100 | Number of trees in ensemble |
| max_depth | 6, 16, 24 | Maximum tree depth |
| Random Forest | | |
| max_depth | 16, 24, 32, or 48 | Maximum tree depth |
| n_streams | 8 | Number of simultaneous GPU streams active during the fit phase |
| ET | | |
| max_depth | 16, 24, no limit (default) | Maximum tree depth |

**Table 5** Mean AUC scores (ten iterations of fivefold cross-validation)

| Dataset | CB | ET | RF-GPU-16 | XGB |
|---|---|---|---|---|
| B | 0.8656 | 0.9812 | 0.9350 | 0.9023 |
| | (0.0013) | (0.0008) | (0.0010) | (0.0014) |
| D | 0.7323 | 0.9675 | 0.8082 | 0.7563 |
| | (0.0009) | (0.0004) | (0.0009) | (0.0007) |

Standard deviations are below mean values in parentheses

Due to space limitations, we abbreviate classifier names in the results and statistical analysis as follows: CatBoost is abbreviated as CB, Random Forest is abbreviated as RF-GPU, XGBoost is abbreviated as XGB, and Extremely Randomized Trees is abbreviated as ET. When a number follows the abbreviation, it indicates the maximum tree depth. For example, XGB-16 means XGBoost with a maximum tree depth of 16.

Maximum tree depth is a challenging parameter to optimize, since increasing it by one allows the learner to potentially double the number of nodes in a decision tree, and therefore, consume twice as much memory. We ran into the same limitations on maximum tree depth with both the Part B and Part D datasets. CatBoost builds balanced decision trees. A balanced decision tree is a decision tree with the additional requirement that the maximum number of nodes are populated on every level before the next level may be populated. Therefore, increasing the maximum tree depth by one for CatBoost guarantees a doubling of memory consumed by the decision trees. CatBoost has strict limits on maximum tree depth values, and will halt execution when it detects that the user has specified a maximum tree depth larger than 16. Hence, we provide results for CatBoost with its default maximum tree depth of 6 and 16 in Table 6.

We were able to run experiments with XGBoost with maximum tree depth as high as 24. We encountered runtime errors when attempting to use XGBoost with larger values of maximum tree depth.

By default, ET has no limit on maximum tree depth. Unlike CatBoost, ET does not build balanced decision trees, so the unlimited maximum tree depth does not guarantee a huge memory consumption. Since we confirm the trend that ET's performance in terms of AUC increases as maximum tree depth increases from 16 to 24, and we have AUC scores ET yields with its default maximum tree depth, there is sufficient evidence in Table 6 that ET conforms to our thesis that increasing maximum tree depth is a highly effective method for improving classifiers' performance in terms of AUC for Medicare fraud detection.

Due to the exponential relationship between maximum tree depth and memory consumption, we found it necessary to seek a mitigating technique. RUS is a natural candidate, since it reduces the size of the training data. Therefore, we took the best-performing models documented in Table 6, and conducted further experiments with RUS. The outcomes of those experiments are documented in Table 7. The mean AUC scores in Table 7 offer further confirmation that higher maximum tree depth, regardless of classifier, correlates with higher AUC scores. CatBoost, with the smallest maximum tree depth of 16, yields the lowest AUC scores, whereas other learners with greater maximum tree depth yield mean AUC scores that are much higher. The second important

**Table 6** Performance of classifiers with varying levels of maximum tree depth Mean in terms of AUC Scores (ten iterations of fivefold cross-validation)

| Dataset | Classifier | No limit | 6 | 16 | 24 | 32 | 48 |
|---|---|---|---|---|---|---|---|
| B | RF-GPU | | | 0.9350 (0.0010) | 0.9870 (0.0007) | 0.9886 (0.0005) | 0.9864 (0.0008) |
| B | XGB | | 0.9023 (0.0014) | 0.9910 (0.0004) | 0.9944 (0.0003) | | |
| B | ET | 0.9812 (0.0008) | | 0.8149 (0.0047) | 0.8860 (0.0014) | | |
| B | CB | | 0.8656 (0.0013) | 0.8970 (0.0012) | | | |
| D | RF-GPU | | | 0.8082 (0.0009) | 0.9520 (0.0008) | 0.9700 (0.0003) | 0.9685 (0.0004) |
| D | XGB | | 0.7563 (0.0007) | 0.9245 (0.0014) | 0.9727 (0.0004) | | |
| D | ET | 0.9675 (0.0004) | | 0.6986 (0.0012) | 0.7223 (0.0018) | | |
| D | CB | | 0.7323 (0.0009) | 0.7485 (0.0008) | | | |

Standard deviations are below mean values in parentheses; blank spaces indicate that classifer/maximum tree depth combination is not attempted

**Table 7** Mean AUC scores of classifiers with varying maximum tree depth and undersampling (ten iterations of fivefold cross-validation)

| Dataset | RF-GPU-32 | XGB-24 | CB-16 | ET |
|---|---|---|---|---|
| B | 0.9886 (0.0005) | 0.9944 (0.0003) | 0.8970 (0.0012) | 0.9812 (0.0008) |
| B RUS 1:1 | 0.9753 (0.0004) | 0.9944 (0.0003) | 0.9307 (0.0009) | 0.9932 (0.0004) |
| B RUS 1:3 | 0.9840 (0.0004) | 0.9922 (0.0003) | 0.9343 (0.0009) | 0.9932 (0.0004) |
| B RUS 1:9 | 0.9885 (0.0004) | 0.9939 (0.0003) | 0.9303 (0.0010) | 0.9926 (0.0004) |
| B RUS 1:27 | 0.9905 (0.0004) | 0.9944 (0.0003) | 0.9215 (0.0013) | 0.9909 (0.0004) |
| B RUS 1:81 | 0.9907 (0.0005) | 0.9946 (0.0003) | 0.9120 (0.0010) | 0.9876 (0.0005) |
| D | 0.9700 (0.0003) | 0.9727 (0.0004) | 0.7485 (0.0008) | 0.9675 (0.0004) |
| D RUS 1:1 | 0.9352 (0.0005) | 0.9529 (0.0006) | 0.7717 (0.0007) | 0.9763 (0.0003) |
| D RUS 1:3 | 0.9547 (0.0002) | 0.9681 (0.0004) | 0.7713 (0.0007) | 0.9759 (0.0004) |
| D RUS 1:9 | 0.9648 (0.0004) | 0.9726 (0.0004) | 0.7657 (0.0008) | 0.9746 (0.0004) |
| D RUS 1:27 | 0.9687 (0.0003) | 0.9736 (0.0004) | 0.7589 (0.0006) | 0.9724 (0.0004) |
| D RUS 1:81 | 0.9696 (0.0004) | 0.9735 (0.0004) | 0.7533 (0.0009) | 0.9696 (0.0004) |

Standard deviations are below mean values in parentheses

finding apparent in Table 7 is that the level of RUS appears to have a small effect on AUC scores for some learners. This is appealing, because, for imbalanced datasets, applying RUS to applied induce higher class ratios reduces the size of the training dataset to a multiple of the size of the minority class. In the next section, we conduct statistical analyses to determine the level of RUS we can apply to obtain optimal performance in terms of AUC.

## Statistical Analysis

We perform ANOVA and Tukey HSD tests to get a better sense of the impact of maximum tree depth and RUS on AUC scores. The outcomes of the tests enable us to make informed conclusions on the effect of the factors. Moreover, they provide guidelines for future research. We perform a total of four analyses. The first two analyses are on experimental outcomes for experiments where RUS is not applied. We do one analysis for experiments with the Part B data, and one analysis for experiments with the Part D data. These analyses are on the same data used to populate Table 6. We provide two more analyses, for experiments involving RUS. Again, the analyses are separated into one for experiments involving the Part B Data, and one for experiments involving the Part D data. The analyses on the outcomes of experiments involving RUS are done on the same data used to populate Table 7.

**Table 8** ANOVA for depth and classifier as factors of performance in terms of AUC

| | Df | Sum Sq | Mean Sq | *F* value | Pr(>*F*) |
|---|---|---|---|---|---|
| Depth | 5 | 0.84 | 0.17 | 1206.27 | * |
| Classifier | 3 | 1.14 | 0.38 | 2733.87 | * |
| Residuals | 591 | 0.08 | 0.00 | | |

*Value is less than $1 \times 10^{-15}$

**Table 9** HSD test groupings after ANOVA of AUC for the depth factor

| |
|---|
| Group a consists of: 32, 48, no limit |
| Group b consists of: 24 |
| Group c consists of: 16 |
| Group d consists of: 6 |

**Table 10** HSD test groupings after ANOVA of AUC for the classifier factor

| |
|---|
| Group a consists of: RF-GPU |
| Group b consists of: XGB |
| Group c consists of: ET |
| Group d consists of: CB |

## Part B Experiments Without RUS: Analysis of Results in Terms of AUC

The first experiments we analyze are the experiments involving maximum tree depth and the Part B data. To get started, we perform an ANOVA test to determine whether maximum tree depth has a significant impact on performance in terms of AUC for classifying the Part B data. The Pr(>F) or *p* values for the *F*-statistics we calculate in the ANOVA tests are practically zero, which indicates that maximum tree depth has a statistically significant impact on performance in terms of AUC (Table 8).

Since the ANOVA test shows that maximum tree depth has a significant impact on performance in terms of AUC, a Tukey HSD test will assign the levels of the maximum tree depth factor into groups that yield equivalent performance. The conclusion we draw from Table 9 is that the larger values of maximum tree depth are associated with the best AUC scores for classifying the part B data.

Classifier is the second factor in the ANOVA test results in Table 8. Since the classifier also has a significant impact on performance in terms of AUC, we conduct an HSD test to rank classifiers in terms of the AUC scores they yield. We report the results of the HSD test in Table 10. We find that the most relevant implication of Table 10 is that the classifier that accommodates only the lowest maximum tree depth of all, CatBoost, is also ranked lowest in the HSD test.

**Table 11** ANOVA for depth and classifier as factors of performance in terms of AUC

| | Df | Sum Sq | Mean Sq | *F* value | Pr(>*F*) |
|---|---|---|---|---|---|
| Depth | 5 | 3.99 | 0.80 | 1004.47 | * |
| Classifier | 3 | 2.90 | 0.97 | 1215.38 | * |
| Residuals | 591 | 0.47 | 0.00 | | |

*Value is less than $1 \times 10^{-15}$

## Part D Experiments Without RUS: Analysis of Results in Terms of AUC

We move on to analyze the impact of maximum tree depth on the classification of the Part D data. We report the results of an ANOVA test to determine the significance of maximum tree depth and classifier in Table 11. The Pr(>*F*) values of the *F*-statistics of the ANOVA tests are practically zero. Thus, both classifier and maximum tree depth have a significant effect on the classification results.

Since classifier and maximum tree depth also have a significant impact on AUC scores when classifying the Part D data, we conduct HSD tests to rank these factors. Table 12 contains the results of an HSD test to rank levels of maximum tree depth in accordance with their impacts on AUC scores. Interestingly, the groups that the HSD test determines for the Part D experiments are the same as the groups that the HSD test determines for the Part B experiments. Therefore, we draw the same conclusion that performance in terms of AUC is positively correlated with maximum tree depth.

Since the *p* value of the *F*-statistic for the classifier value in the ANOVA test results in Table 11 is practically 0, we conduct an HSD test to rank classifiers in terms of their impact on AUC scores. The results of the HSD test for the Part D data in Table 13 are also the same as the results of the HSD test for the Part B data in Table 10. Therefore, the result aligns with the previous findings, for classifying the Part D data. The classifier that yields the lowest AUPRC scores is also the one that allows the lowest possible maximum tree depth.

## Part B RUS Experiments: Analysis of Results in Terms of AUC

As mentioned previously, increasing maximum tree depth has an exponential relationship with potential resource consumption. Therefore, a technique for mitigating the resource consumption is attractive. Since applying RUS

**Table 12** HSD test groupings after ANOVA of AUC for the depth factor

| |
|---|
| Group a consists of: 32, 48, no limit |
| Group b consists of: 24 |
| Group c consists of: 16 |
| Group d consists of: 6 |

**Table 13** HSD test groupings after ANOVA of AUC for the classifier factor

| |
|---|
| Group a consists of: RF-GPU |
| Group b consists of: XGB |
| Group c consists of: ET |
| Group d consists of: CB |

**Table 14** ANOVA for RUS and classifier as factors of performance in terms of AUC

| | Df | Sum Sq | Mean Sq | $F$ value | Pr(>$F$) |
|---|---|---|---|---|---|
| RUS | 5 | 0.02 | 0.00 | 81.75 | * |
| Classifier | 3 | 1.08 | 0.36 | 8796.68 | * |
| Residuals | 1191 | 0.05 | 0.00 | | |

*Value is less than $1 \times 10^{-15}$

shrinks the size of the training data for imbalanced datasets, it is a natural candidate for a way to combat increased resource consumption due to increased maximum tree depth. However, one should be cognizant of any trade-off of performance in terms of AUC for the decreased resource consumption that RUS affords. Therefore, we conduct a statistical analysis to determine whether RUS has a significant impact on performance in terms of AUC. We analyze results for the classifiers and maximum tree depth levels that yield the best performance in terms of AUC that we identify in the previous sections. The first analysis we conduct is an ANOVA test for the effect of RUS and classifier on Medicare fraud detection in the Part B data. The Pr(>$F$) values reported in Table 14 indicate that both classifier and level of RUS have a significant impact on AUC scores.

Since RUS has a significant effect on classification of the Part B data, an HSD test will tell us which levels of the RUS factor are associated with the highest AUC scores. In this case, we see the 1:9 and 1:3 class ratios listed in Table 15 yield the highest AUC scores. This is good news, since it means we can apply RUS to obtain smaller training data, and maintain strong performance in terms of AUC.

Next, we proceed with an HSD test to rank classifiers across all levels of RUS. In Table 16, we find that XGBoost with a maximum tree depth of 24 yields the best performance. Furthermore, the results in Table 16 indicate the classifier that permits the lowest maximum tree depth is associated with the lowest AUC scores as we vary levels of RUS.

### Part D RUS Experiments: Analysis of Results in Terms of AUC

We do a similar analysis on the effect of classifier and RUS on AUC scores for classifying the part D data. Table 17 contains the results of an ANOVA test where classifier and RUS are treated as factors. The Pr(>$F$) values indicate that both factors have a significant impact on classifying the Part D data.

The HSD results in Table 18 indicate that RUS applied to induce a class ratio of 1:9 yields the best AUC scores to classify the Part D data. These results are similar to those for classifying the Part B data; only now the 1:3 class ratio has moved to the group that yields the second-best performance.

We conduct a second HSD test for the classifier factor's effect on AUC scores. The outcome of the test is reported in Table 19. For the Part D data, the result for the group that yields the lowest AUC scores is the same as for the Part B data. However, the best-performing classifier is different. Here, we find that, across all levels of RUS, ET is associated with the best performance.

## Conclusions

The first conclusion we would like to mention involves CatBoost. We were not able to increase the maximum tree depth of the decision trees in the CatBoost classifier beyond 16. This is a much lower maximum tree depth than the other classifiers permit. However, CatBoost plays another important role in our experiments. We use CatBoost encoding to encode the categorical features in the Part B and Part D datasets. Given the strength of the results, we feel that future work to ascertain CatBoost's contribution to the results is appropriate. This can be determined by further experiments including different techniques for encoding categorical features.

Our experimental results and statistical analyses reveal some facts about Medicare fraud detection in the latest, publicly available Part B and Part D data. These findings are a contribution, since they pertain to new data. The first important finding is the effect of maximum tree depth on AUC scores. Our results show that maximum tree depth is positively correlated with AUC scores. The second noteworthy

**Table 15** HSD test groupings after ANOVA of AUC for the RUS factor

| |
|---|
| Group a consists of: 1:9, 1:3 |
| Group ab consists of: 1:27 |
| Group b consists of: 1:1 |
| Group c consists of: 1:81 |
| Group d consists of: No RUS |

**Table 16** HSD test groupings after ANOVA of AUC for the classifier factor

| |
|---|
| Group a consists of: XGB-24 |
| Group b consists of: ET |
| Group c consists of: RF-GPU-32 |
| Group d consists of: CB-16 |

**Table 17** ANOVA for RUS and classifier as factors of performance in terms of AUC

| | Df | Sum Sq | Mean Sq | $F$ value | Pr(>$F$) |
|---|---|---|---|---|---|
| RUS | 5 | 0.01 | 0.00 | 44.45 | * |
| Classifier | 3 | 9.55 | 3.18 | 50352.60 | * |
| Residuals | 1191 | 0.08 | 0.00 | | |

*Value is less than $1 \times 10^{-15}$

**Table 18** HSD test groupings after ANOVA of AUC for the RUS factor

| |
|---|
| Group a consists of: 1:9 |
| Group ab consists of: 1:27, 1:3 |
| Group bc consists of: 1:81 |
| Group c consists of: No RUS |
| Group d consists of: 1:1 |

**Table 19** HSD test groupings after ANOVA of AUC for the classifier factor

| |
|---|
| Group a consists of: ET |
| Group b consists of: XGB-24 |
| Group c consists of: RF-GPU-32 |
| Group d consists of: CB-16 |

result of our study is that one may employ learners with higher maximum tree depth values, and apply RUS to induce class ratios to build models that outperform models trained on the full dataset, in terms of AUC scores. This is good news for researchers experimenting with the Part B and Part D datasets. Fewer training instances, which translate to shorter model training time, can be used to get results similar to, or better than results obtained with the full Part B or Part D datasets. For classifying the Part B data, we find that models trained on data with RUS applied to make the class ratio 1:3 yield the best performance. Similarly, for the Part D data, we find models trained on data with RUS applied to induce the 1:9 class ratio yield the best performance. Since both the Part B and Part D datasets are large and imbalanced, applying RUS reduces the size of the training data by two orders of magnitude. The reduction in the size of the training data mitigates the increased resource consumption that is a by-product of increasing maximum tree depth. In future work, we plan to extend our analysis to other metrics, such as Area Under the Precision–Recall Curve, and other highly imbalanced Big Data.

**Acknowledgements** We would like to thank the Data Mining and Machine Learning Research Group at Florida Atlantic University for their assistance in preparing this manuscript.

## Declarations

**Conflict of Interest** On behalf of all authors, the corresponding author states that there is no conflict of interest.

## References

1. The Centers for Medicare and Medicaid Services: Medicare Physician & Other Practitioners: by provider and service. 2021. https://data.cms.gov/provider-summary-by-type-of-service/medicare-physician-other-practitioners/medicare-physician-other-practitioners-by-provider-and-service. Accessed 9 May 2022.
2. The Centers for Medicare and Medicaid Services: Medicare Part D Prescribers: by provider and drug. 2021. https://data.cms.gov/provider-summary-by-type-of-service/medicare-part-d-prescribers/medicare-part-d-prescribers-by-provider-and-drug. Accessed 18 Feb 2022.
3. Centers for Medicare and Medicaid Services: 2019 Estimated Improper Payment Rates for Centers for Medicare & Medicaid Services (CMS) Programs. 2019. https://www.cms.gov/newsroom/fact-sheets/2019-estimated-improper-payment-rates-centers-medicare-medicaid-services-cms-programs. Accessed 1 Mar 2022.
4. Civil Division, U.S. Department of Justice: Fraud Statistics, Overview. 2020. https://www.justice.gov/opa/press-release/file/1354316/download. Accessed 18 Jan 2022.
5. Bekkar M, Djemaa HK, Alitouche TA. Evaluation measures for models assessment over imbalanced data sets. J Inf Eng Appl. 2013;3(10):27–38.
6. Hancock J, Khoshgoftaar TM. Optimizing ensemble trees for big data healthcare fraud detection. In: 2022 IEEE 23rd international conference on information reuse and integration for data science (IRI); 2022. IEEE. p. 243–49
7. Prokhorenkova L, Gusev G, Vorobev A, Dorogush AV, Gulin A. Catboost: unbiased boosting with categorical features. Adv Neural Inf Process Syst. 2018;31:1–11.
8. Chen T, Guestrin C. Xgboost: a scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining-KDD '16; 2016.
9. Breiman L. Random forests. Mach Learn. 2001;45(1):5–32.

10. Geurts P, Ernst D, Wehenkel L. Extremely randomized trees. Mach Learn. 2006;63(1):3–42.
11. Bauder RA, Khoshgoftaar TM, Hasanin T. Data sampling approaches with severely imbalanced big data for medicare fraud detection. In: 2018 IEEE 30th international conference on tools with artificial intelligence (ICTAI); 2018. IEEE. p. 137–42
12. The Centers for Medicare and Medicaid Services: Medicare Durable Medical Equipment, Devices & Supplies: by Referring Provider and Service. 2021. https://data.cms.gov/provider-summary-by-type-of-service/medicare-durable-medical-equipment-devices-supplies/medicare-durable-medical-equipment-devices-supplies-by-referring-provider-and-service. Accessed 18 Jan 2022
13. Le Cessie S, Van Houwelingen JC. Ridge estimators in logistic regression. J R Stat Soc Ser C (Appl Stat). 1992;41(1):191–201.
14. Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S. Mllib: machine learning in apache spark. J Mach Learn Res. 2016;17(1):1235–41.
15. Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ. Apache spark: a unified engine for big data processing. Commun ACM. 2016;59(11):56–65.
16. Han H, Wang W-Y, Mao B-H. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In: International conference on intelligent computing; 2005. Springer. p. 878–887
17. Lin W, Wu Z, Lin L, Wen A, Li J. An ensemble random forest algorithm for insurance big data analysis. IEEE Access. 2017;5:16568–75.
18. Del Río S, López V, Benítez JM, Herrera F. On the use of mapreduce for imbalanced big data using random forest. Inf Sci. 2014;285:112–37.
19. Herrera VM, Khoshgoftaar TM, Villanustre F, Furht B. Random forest implementation and optimization for big data analytics on lexisnexis'5s high performance computing cluster platform. J Big Data. 2019;6(1):1–36.
20. Genuer R, Poggi J-M, Tuleau-Malot C, Villa-Vialaneix N. Random forests for big data. Big Data Res. 2017;9:28–46.
21. Fauzan MA, Murfi H. The accuracy of xgboost for insurance claim prediction. Int J Adv Soft Comput Appl. 2018;10(2):159–71.
22. Li H, Cao Y, Li S, Zhao J, Sun Y. Xgboost model and its application to personal credit evaluation. IEEE Intell Syst. 2020;35(3):52–61.
23. XingFen W, Xiangbin Y, Yangchun M. Research on user consumption behavior prediction based on improved xgboost algorithm. In: 2018 IEEE international conference on big data (Big Data); 2018. IEEE. p. 4169–175.
24. Johnson JM, Khoshgoftaar TM. Deep learning and data sampling with imbalanced big data. In: 2019 IEEE 20th international conference on information reuse and integration for data science (IRI); 2019. IEEE. p. 175–83.
25. LEIE: Office of Inspector General Leie Downloadable Databases. [Online]. https://oig.hhs.gov/exclusions/index.asp. Accessed 12 Apr 2022
26. Herland M, Khoshgoftaar TM, Bauder RA. Big data fraud detection using multiple medicare data sources. J Big Data. 2018;5(1):1–21.
27. The Centers for Medicare and Medicaid Services: Medicare Physician & Other Practitioners: by Provider and Service Data Dictionary. 2021. https://data.cms.gov/resources/medicare-physician-other-practitioners-by-provider-and-service-data-dictionary. Accessed 28 Jan 2022.
28. The Centers for Medicare and Medicaid Services: Medicare Part D Prescribers: by provider and drug data dictionary. 2021. https://data.cms.gov/resources/medicare-part-d-prescribers-by-provider-and-drug-data-dictionary. Accessed 4 May 2022.
29. Breiman L. Bagging predictors. Mach Learn. 1996;24(2):123–40.
30. Hancock J, Khoshgoftaar TM. Performance of catboost and xgboost in medicare fraud detection. In: 2020 19th IEEE international conference on machine learning and applications (ICMLA); 2020. IEEE. p. 572–79.
31. Friedman JH. Greedy function approximation: a gradient boosting machine. Ann Stat. 2001;29:1189–232.
32. Hancock JT, Khoshgoftaar TM. Catboost for big data: an interdisciplinary review. J Big Data. 2020;7(1):1–45.
33. Van Rossum G, Drake F. Python 3 reference manual createspace. Scotts Valley; 2009.
34. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V. Scikit-learn: machine learning in python. J Mach Learn Res. 2011;12:2825–30.
35. Johnson JM, Khoshgoftaar TM. Hcpcs2vec: Healthcare procedure embeddings for medicare fraud prediction. In: 2020 IEEE 6th international conference on collaboration and internet computing (CIC); 2020. IEEE. p. 145–52.
36. Hancock JT, Khoshgoftaar TM. Survey on categorical data for neural networks. J Big Data. 2020;7(1):1–41.
37. Parameters. Yandex Corporation. https://catboost.ai/en/docs/references/training-parameters/common. Accessed 09 July 2022
38. XGBoost Parameters. XGBoost Developers. https://xgboost.readthedocs.io/en/stable/parameter.html. Accessed 09 July 2022.
39. Hancock JT, Khoshgoftaar TM. Hyperparameter tuning for medicare fraud detection in big data. SN Comput Sci. 2022;3(6):1–13.