Journal of Big Data

RESEARCH Open Access

Evaluating classifier performance with highly imbalanced Big Data

John T. Hancock[1*], Taghi M. Khoshgoftaar[1] and Justin M. Johnson[1]

*Correspondence: jhancoc4@fau.edu

[1] Department of Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, USA

Abstract

Using the wrong metrics to gauge classification of highly imbalanced Big Data may hide important information in experimental results. However, we find that analysis of metrics for performance evaluation and what they can hide or reveal is rarely covered in related works. Therefore, we address that gap by analyzing multiple popular performance metrics on three Big Data classification tasks. To the best of our knowledge, we are the first to utilize three new Medicare insurance claims datasets which became publicly available in 2021. These datasets are all highly imbalanced. Furthermore, the datasets are comprised of completely different data. We evaluate the performance of five ensemble learners in the Machine Learning task of Medicare fraud detection. Random Undersampling (RUS) is applied to induce five class ratios. The classifiers are evaluated with both the Area Under the Receiver Operating Characteristic Curve (AUC), and Area Under the Precision Recall Curve (AUPRC) metrics. We show that AUPRC provides a better insight into classification performance. Our findings reveal that the AUC metric hides the performance impact of RUS. However, classification results in terms of AUPRC show RUS has a detrimental effect. We show that, for highly imbalanced Big Data, the AUC metric fails to capture information about precision scores and false positive counts that the AUPRC metric reveals. Our contribution is to show AUPRC is a more effective metric for evaluating the performance of classifiers when working with highly imbalanced Big Data.

Keywords: Extremely randomized trees, XGBoost, Class imbalance, Big Data, Undersampling, AUC, AUPRC

Introduction

The use of a single metric to draw conclusions on the impact of a factor in classification experiments may lead to mistakes that we wish to help our fellow researchers avoid. Random Undersampling (RUS) is an appealing strategy for mitigating class imbalance in Big Data. It can drastically reduce the size of the training data used during the model training phase of Machine Learning. Less training data translates into faster training times for many Machine Learning algorithms. Therefore, applying RUS may save a researcher time when conducting experiments. Our contribution is to reveal that there is a trade-off for applying RUS that one should consider before concluding that applying RUS is the best choice. We show that RUS may have a positive impact on Area Under the Receiver

Operating Characteristic Curve (AUC) [1] scores. At the same time, we show RUS may have a clear negative impact on Area Under the Precision Recall Curve (AUPRC) [2] scores.

This underscores the importance of evaluating results in terms of more than one metric. Therefore, if performance in terms of AUPRC is important, applying RUS may not be a viable option. We validate these findings using three distinct data sets and five popular ensemble learners in the task of Medicare fraud detection. In our experiments, we apply RUS to induce five different levels of minority:majority class ratios, and classify datasets of varying sizes. The smallest dataset we work with has approximately 12 million instances. We also perform experiments with a dataset that has approximately 68 million instances, and another dataset that has approximately 175 million instances. For each dataset we find the same pattern holds: AUC scores are either not affected, or improved by RUS, and AUPRC scores are degraded.

The application domain of our study is automated Medicare insurance fraud detection. Medicare is the United States public health insurance program, primarily dedicated to individuals aged 65 and over. The organization responsible for the Medicare program is the Centers for Medicare and Medicaid Services (CMS). To foster research, the CMS maintains a repository of publicly available Medicare insurance claims data. We use data from three different sources in this study. Data from each source has unique attributes. The smallest dataset we use is from a section of the CMS website titled "Medicare Durable Medical Equipment, Devices & Supplies—by Referring Provider and Service" (DMEPOS) [3]. We construct a dataset of approximately 12 million instances from this data. The next largest data we use is from a section of the CMS website "Medicare Physician & Other Practitioners—by Provider and Service" (Part B) [4]. We derive a dataset with approximately 68 million instances from the Part B data. Finally, the largest data we use is from a section of the CMS website titled "Medicare Provider Utilization and Payment Data: Part D Prescriber" (Part D) [5]. We compile a dataset of nearly 175 million instances from the Part D data. The CMS regularly adds to each of these data sources, which lends them the aspects of volume and velocity that characterizes Big Data [6].

The volume and velocity of the Part B, Part D, and DMEPOS data also reflects the quantity and speed at which the CMS receives insurance claims from healthcare providers. Currently, it is possible for some dishonest providers to get away with submitting fraudulent claims to Medicare and avoid detection. The volume of claims submitted is large enough that a small fraction of undetected fraudulent claims still translates to large dollar amounts. It is a fact that, in 2019, the Department of Justice was able to recover approximately three billion dollars in fraudulently obtained funds by prosecuting rogue healthcare providers [7]. Nevertheless, there is a degree of uncertainty surrounding how much money the CMS loses due to fraud. The CMS does not report an estimate of funds paid on fraudulent claims. Rather, it reports an estimate of "improper payments." For 2019, the CMS reported it made approximately \$100 billion dollars in improper payments [8]. The CMS defines improper payments to include payments due to fraud, as well as payments due to mistakes on the CMS's part. Reliable, automated fraud detection would provide a means for the CMS to give an estimate of the percentage of improper payments due to fraud. This would in turn provide a stronger justification for law enforcement to pursue the recovery of money stolen by fraudsters. Our application domain is Machine Learning for Medicare fraud detection. Therefore, our work is a contribution towards the ultimate goal of automated Medicare fraud detection. The benefit of better fraud detection is that government can put funds to better use, or lower taxes due to a reduced cost of the program.

There are valid concerns to be raised on the subject of automated fraud detection. Chief among them is the possibility of accusing legitimate providers of fraud when they are innocent. Since we refer to the population of fraudulent providers as the positive class in our classification framework, accusation of innocent healthcare providers is equivalent to a false positive. Clearly, numerous false positives would mean that automated fraud detection would be doing more harm than good. This is why the key finding in our work is important in the field of automated Medicare fraud detection, since it reveals a better way to detect when a classifier that is applied to classify highly imbalanced Big Data yields many false positives. AUC is a popular metric for evaluating highly imbalanced Big Data, for example [9, 10]. However, our research shows AUC might not always provide a complete picture of classification results. Our experimental results show how the AUPRC metric can provide a clearer signal that a model is generating false positives, when compared to AUC. A look at the definitions of the components of AUC and AUPRC reveals why AUPRC is a better herald of false positives. On one hand, AUPRC is calculated by plotting the precision and recall scores a model yields as we vary the output probability threshold for the classification decision from zero to one. Precision is defined as

$$\frac{\text{true positives}}{\text{true positives} + \text{false positives,}}$$

and the definition of recall is

$$\frac{\text{true positives}}{\text{true positives} + \text{false negatives.}}$$

On the other hand, AUC is calculated by plotting true positive rate and false positive rate as the output probability threshold varies from zero to one. The true positive rate is

$$\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

and false positive rate equivalent to

$$\frac{\text{false positives}}{\text{true negatives} + \text{false positives}}.$$

Since true positive rate and recall are actually the same quantity, the difference in AUC and AUPRC must come from the difference between precision and the false positive rate. We see that the false positive rate involves true negatives, whereas precision does not. In highly imbalanced Big Data, where the positive class is the minority class, the true positives in the formula for precision should be small numbers, so that when the number of false positives starts to grow, it can quickly dominate the value of precision. Hence, precision can easily reflect the number of false positives in classifying imbalanced Big Data.

A similar analysis of the terms involved in calculating the false positive rate shows that false positives get drowned out due to the size of the negative class. Since the denominator in the definition of the false positive rate is the size of the negative class, and the size of the negative class is large in imbalanced Big Data, a change in the number of false positives may be difficult to perceive.

To solidify the argument, we give an example with some hypothetical numbers. Let us assume we have a dataset where the size of the negative class is two million. Furthermore, let us assume we have done a classification of the data for some output probability threshold, and we have 1800 true positives and 2000 false positives. Moreover, the sample has 2000 positive instances. From these numbers, we can calculate precision and false positive rate. The precision is

$$\frac{1800}{1800 + 2000} \approx 0.47,$$

and the false positive rate is

$$\frac{2000}{2,000,000} = 0.001$$

Now let us assume that for a different output probability threshold the number of false positives has increased to 4000. Then the new value of precision is

$$\frac{1800}{1800 + 4000)} \approx 0.31,$$

and the false positive rate is

$$\frac{4000}{2,000,000} = 0.002$$

In this example, doubling the total number of false positives decreases the precision score by 0.16, but only increases the false positive rate by 0.001. Since both values are used directly as values of coordinates on curves that occupy the same square with area 1×1 in the x–y coordinate plane, we can compare them directly. Therefore, for this example, we conclude that the increase in false positives has a bigger impact on precision and AUPRC, than on the false positive rate and AUC. Therefore, it is possible that if a factor in some experiments causes a larger number of false positives, AUC will not reflect the impact, but AUPRC will.

We perform a collection of experiments that provides an example of how AUC will not reflect the impact of a factor on classification results, but AUPRC will. One factor tested in this collection of experiments is the minority:majority class ratio in the training data. We apply RUS to induce five different class ratios. The second factor in our experiments is the type of classifier used. We use five popular, open source ensemble learners: CatBoost [11], XGBoost [12], LightGBM [13], Random Forest [14], and Extremely Randomized Trees (ET) [15]. Our results show that, regardless of learner and dataset, AUPRC scores are diminished as the class ratio gets closer to 1:1. The AUC scores for the same experiments do not reflect the relationship RUS has with AUPRC. This leads us to the conclusion that AUPRC scores reveal more about the impact of RUS than AUC scores. Our goal is to provide a thorough justification for this conclusion. Along the way to providing this justification, to the best of our knowledge, we make several novel contributions:

- we are the first to use the Part B, Part D, and DMEPOS data, which became available in 2021, in a peer-reviewed study;
- we are the first to show that classification of this new Big Data should be evaluated in terms of AUPRC,
- we are the first to use five ensemble learners to do Medicare fraud detection,
- we are the first to employ a Random Forest implementation that runs on Graphics Processing units (GPUs) do to Medicare fraud detection, and
- we are the first to use ET to do Medicare fraud detection.

The remainder of this study is organized into the following sections: Related Works, Algorithms, Data Description and Preparation, Methodology, Results, Statistical Analysis, and Conclusions.

Related work

"Data sampling approaches with severely imbalanced big data for medicare fraud detection" is a 2018 study by Bauder et al. [9]. In their study, the authors combine Part B, Part D, and DMEPOS Medicare claims data to form a dataset for Medicare fraud detection via classification. Hence, their study is in the same application domain as ours, albeit with less data than we use, since we use data was not available at the time their study was written. The data they work with has under one million instances. Bauder et al. employ six data sampling techniques in their experiments. The six techniques are RUS, Random Oversampling (ROS), Synthetic Minority Oversampling Technique (SMOTE) [16], two variations of Borderline SMOTE (also covered in [16], and Adaptive Synthetic (ADASYN) [17]. The sampling techniques are used to induce minority:majority class ratios of 1:99, 10:90, 25:75, 65:35, and 50:50. Experiments with the original class ratio of 473:759,267 (approximately 0.00062) are performed as well. For classification experiments, they use Apache Spark [18] implementations of Random Forest, Logistic Regression [19] and Gradient Boosted Trees [20]. To evaluate the performance of the combinations of classifiers and data sampling techniques, the authors use AUC. Bauder et al. conclude that classifiers trained on data with RUS applied to it yield significantly better performance, in terms of AUC, than classifiers trained on data with the original class ratio. Since Bauder et al. prefer RUS to other sampling techniques, we employ only RUS. However, we measure classification results in terms of the AUPRC metric as well. Our results are unique and meaningful, since, on one hand we duplicate Bauder et al.'s results that show an improvement in AUC scores when RUS is applied, but on the other hand, we show that AUPRC scores decline when RUS is used.

Hasanin et al. [21] study the effect of RUS on Geometric Mean [22] and AUC scores. They find RUS has a positive impact on AUC and Geometric Mean scores in the classification of imbalanced Big Data. Here, we investigate the performance of RUS on AUC and AUPRC scores. One advantage AUC and AUPRC have over the Geometric Mean is that they reflect performance over a range of model output probability threshold values, whereas in order to calculate Geometric Mean, one must select a specific output probability threshold. Therefore, a model's Geometric Mean score can only tell us about the performance of the model for one particular threshold value. Another issue that sets our study apart from Hasanin et al. is that the largest dataset used in their study has under 1.7 million instances. The data we work with here is orders of magnitude larger. Hasanin et al. report that they use one-hot encoding for all categorical features. Here we use CatBoost encoding [11], a technique that is more scalable than one-hot encoding since it does not require the introduction of additional attributes to the dataset. For example, to one-hot encode a categorical value that has thousands of possible values, we would need to add thousands of attributes to our dataset. However, CatBoost encoding requires no additional space consumption.

Another study where the authors choose to use Geometric mean as a performance metric is by Del Río et al. In this 2014 study, the authors investigate the impact of RUS on Big Data classification. A second performance metric they use is βf-measure [23]. As a part of their study, experiments are performed as applications of Random Forest to classify various datasets, which are treated with undersampling. The largest dataset they employ in their experiments has less than six million instances. Apart from the size of the dataset, another aspect of our work that sets it apart from Del Río et al. is how they treat their data with RUS. Del Río et al. apply RUS to induce a 1:1 class ratio. We apply RUS to induce five different class ratios, which enables us to report the effect of varying levels of RUS. This is important since we aim to show the effects of RUS on AUC and AUPRC.

Research into the impact of RUS on the classification of Big Data often involves the Apache Spark framework since it is well suited to Big Data. One such study is by Sleeman and Krawczyk [24]. In their experiments, they work with datasets that have at most three million instances. Sleeman and Krawczyk do a thorough job of reporting performance metrics in their study, however, they do not report performance for the AUC or AUPRC metrics. The focus of our study is on the impact of RUS on two well-known metrics, AUC and AUPRC. A separate aspect of our study that differentiates it from Sleeman and Krawczyk's is that we investigate the effect of RUS at different levels. Sleeman and Krawczyk investigate RUS to induce a 1:1 class ratio. We use RUS to induce five class ratios, thus we can treat RUS more thoroughly as a factor in our experiments. The differences between Sleeman and Krawczyk's study and our own imply that our results serve different purposes.

In "Threshold based optimization of performance metrics with severely imbalanced big security data" Calvert and Khoshgoftaar use multiple metrics to evaluate multiple classifiers [25]. The application domain for their study is information systems network security. Hence, their results reveal the ability of Machine Learning algorithms to detect malicious network traffic. Since most of the traffic in their dataset is benign, the classification task is an exercise in the classification of imbalanced data. The data they use in their experiments has approximately 1.7 million instances. To give a sense of the level of class imbalance, the dataset Calvert and Khoshgoftaar use has a minority to majority class ratio of approximately 0.0014. Therefore, their data resembles ours, however we find our dataset is more realistic, since Calvert and Khoshgoftaar generate the malicious traffic in the raw network traffic data they use in their experiments. In a sense our raw data is more natural, since we do not play a role in generating it. Their key finding is that one classifier yields the best performance in terms of AUC, but significantly worse in terms of other metrics. Furthermore, they find one classifier yields the best performance in multiple metrics other than AUC. They claim this result indicates AUC alone cannot identify the best performing model. The principal item that differentiates our study from Calvert and Khoshgoftaar's is that they do not use RUS as a factor in any of their experiments.

One type of classifier we have not discussed yet is Neural Network classifiers. Johnson and Khoshgoftaar document experiments with Neural Network-based classifiers and RUS in [26]. The performance metrics they use to evaluate classification results are AUC, Geometric Mean, True Positive Rate, and True Negative Rate. The application domain for their study is Medicare fraud detection. One thing that sets our study apart from Johnson and Khoshgoftaar's is our use of the AUPRC metric. Our studies have use of the AUC metric in common. Johnson and Khoshgoftaar find that when RUS is applied to their data to induce a class ratio with the minority class occupying more than 20% of the data, AUC scores begin to deteriorate. In a further demonstration on the effect the RUS technique, they show that all metrics reflect worsening performance as RUS is used to grow the proportion of the minority class in the training data. Johnson and Khoshgoftaar work with data similar to ours, however, they apply an aggregation step to prepare their data for experiments. The aggregation reduces the size of their dataset to under five million instances. The aggregation also eliminates the highest cardinality categorical features. They use one-hot encoding for the remainder of the categorical features. For a study on many available options for encoding categorical features, please see [27]. For this study, we selected CatBoost encoding, which has the advantage of supporting much higher cardinality categorical features than can be practical with one-hot encoding. Due to the differences we have listed here, our study represents a contribution that is separate from what Johnson and Khoshgoftaar have to offer.

In a later work, Johnson and Khoshgoftaar use Geometric Mean and AUC to evaluate the performance of Deep Learning algorithms to classify imbalanced Big Data Medicare [28]. They apply RUS to have the minority class constitute larger percentages of the training data. The dataset used in their experiments has approximately five million instances. Results in this study show metrics are even more sensitive to RUS than their previous study. In this study, they find performance, in terms of both metrics, begins to suffer when the minority class becomes more than one percent of the training data. One major difference between our study and Johnson and Khoshgoftaar's is that we use AUPRC, whereas they use Geometric Mean. Also, as in their previous study, Johnson and Khoshgoftaar use an aggregation technique which eliminates high-cardinality categorical features, and then use one-hot encoding to encode remaining categorical features. For reasons mentioned previously, we prefer to apply CatBoost encoding to our categorical features. Hence, our study also differs significantly from this second study by Johnson and Khoshgoftaar.

In "An insight into imbalanced big data classification: outcomes and challenges", Fernandez et al. report on the effects of RUS in the classification of imbalanced Big Data with the Apache Hadoop [29] and Spark distributed computing frameworks. The largest dataset used in their experiments contains approximately 12 million instances. While

Fernandez et al. apply RUS to their data, they induce only the 1:1 class ratio. As stated previously, our experiments involve five different levels of RUS, which enables a more thorough review of the impact of RUS. Fernandez et al. find RUS improves classification performance, in terms of Geometric Mean. Our aim is to compare the impact of RUS on two different performance metrics, AUC and AUPRC, hence to provide results in a domain separate from those of Fernandez et al. Although Fernandez et al. use imbalanced data in their experiments, the data we use is more imbalanced. Their data has an initial minority to majority class ratio of 1:50, whereas ours has an initial ratio of 1:256. Therefore, our study involves a larger, more imbalanced dataset.

In the aptly named "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets", Saito and Rehmsmeier compare classification results in terms of AUC and AUPRC [30]. They perform experiments in the microRNA gene discovery application domain with multiple classifiers and datasets. Their results are similar to ours in that they demonstrate, for varying levels of class imbalance, that AUC curves show good performance. However, for the same experiments, the AUPRC curves reveal stark differences in performance. Saito and Rehmsmeier do not discuss whether their results apply to datasets with high cardinality, categorical features. We show our results apply to datasets, with categorical features that have thousands of possible values. The datasets Saito and Rehmsmeier use are small, all with less than 15,000 instances each. We show the merits of AUPRC as a metric for classifying highly imbalanced Big Data. In addition, we show that as the size of the dataset grows, AUPRC is the more informative metric to assess the impact of experimental factors.

Related works cover concepts that overlap with our study. We find research where RUS is a factor in experiments with highly imbalanced Big Data. However, we do not find a study that reveals insights into the divergent effect of RUS on AUC and AUPRC scores in the classification of highly imbalanced Big Data. We feel our contribution is an important one since it shows that focus on AUC alone can cause one to overlook the negative impact of RUS, and therefore possibly other factors, on classification performance.

Classification algorithms

As a means of ensuring reproducible results, we employ five publicly available, open-source ensemble learners as the classifiers in our experiments. As mentioned previously the learners are: LightGBM, XGBoost, ET, CatBoost, and Random Forest. The learners fall into two distinct families of algorithms. ET and Random Forest are members of the Bagging family of learners. CatBoost, XGBoost, and LightGBM hail from the Gradient Boosted Decision Tree family of Machine Learning algorithms. The advantage to using algorithms that exploit different general techniques is that we can show our results apply to more than just one type of algorithm. Bagging and Gradient Boosted Decision Trees take two different approaches to using a collection of learners to perform classification.

Breiman introduces the Bagging technique for Machine Learning in a 1996 study, [31]. Breiman explains that Bagging can be used in classification and regression problems. Our study involves experiments in binary classification, so we focus on Breiman's treatment of Bagging as it pertains to binary classification. The Bagging technique is based on applying a Machine Learning algorithm (learner) to bootstrap samples of the training data to train a collection (ensemble) of instances of the algorithm. A bootstrap sample is a sample, with replacement, from the training data [32]. After the learners are fit to the bootstrap samples, each learner classifies instances of the test data. The classification result is taken as the classification returned by the majority of learners in the ensemble. A probabilistic argument explains how Bagging may improve classification results.

Assume there is a chance that a poorly performing (weak) learner makes a correct classification more often than not. Then, the chance of a majority of weak learners in an ensemble making the correct classification increases as we increase the number of weak learners. In that case if we treat the ensemble as a classifier it will perform better than one of the constituent learners by itself.

Random Forest relies on the Bagging principle, and adds an enhancement to it. Breiman is also credited with the seminal implementation of Random Forest [14]. Random Forest is an application of the Bagging technique to decision trees, with an addition. In order to explain the enhancement to the Bagging technique, we must first define the term "split" in the context of decision trees. The internal nodes of a decision tree consist of rules that specify which edge to traverse next. The rule is based on a comparison of one numeric value, the split, versus the current value of one of the independent variables in the dataset. Hence, fitting a decision tree to a dataset heavily involves determining the optimal values for splits. The enhancement Random Forest makes to the Bagging approach is to randomly sample a subset of the attributes to use in determining the optimal value for a split. We include Random Forest, since it has been applied successfully to the classification of highly, or severely imbalanced Big Data [33]. As mentioned previously, to the best of our knowledge, this is the first study on Medicare fraud detection where an implementation of Random Forest that runs on Graphics Processing Units (GPUs)[1] is used. In preliminary experiments we found this GPU implementation of Random Forest to have a much faster running time when compared to the CPU based implementation we used previously.

The second classifier from the Bagging family of Machine Learning algorithms we employ is the ET classifier [15]. ET is an extension of Random Forest where we choose the values for splits in the decision tree randomly. In Random Forest, and other decision-tree based learners, splits are usually calculated systematically. For example, one may calculate the optimal value for a split in a decision tree based on some metric that gauges how well the splitting rule divides the training data into subsets that all have the same label. ET does away with systematic ways of determining the values for splits and chooses them randomly. Perhaps surprisingly, our results show ET's random selection of splits can turn out to yield the best performance in classifying highly imbalanced Big Data for Medicare fraud detection.

The remaining classifiers used in our study are descended from the Gradient Boosted Machine algorithm discovered by Friedman [34]. The Gradient Boosting Machine technique is an ensemble technique, but the way in which the constituent learners are combined is different from how it is accomplished with the Bagging technique. The Gradient Boosting Machine technique begins with a single learner that makes an initial set of estimates $\hat{\mathbf{y}}$ of the dependent variable $\mathbf{y}$. The differences (residuals) in the estimates $\hat{\mathbf{y}}$ and

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

$\mathbf{y}$ forms a vector $\mathbf{y} - \hat{\mathbf{y}}$ that we can think of as a new dependent variable that we can estimate with the original independent variables and a second learner. Then, the sum of the output values of the two models will be a more accurate estimate of the dependent variable than the output values of the first model. We can continue to add learners to the ensemble similarly, where each new learner is trained to predict the residuals of the current ensemble. Therefore, each learner we add to the ensemble provides a better estimate of the dependent variable. The Gradient Boosting Machine implementations we use are all enhancements to Friedman's initial proposal. They all involve a specific type of learner, Decision Tree, so we refer to them as Gradient Boosted Decision Trees (GBDTs).

Of the three GBDT implementations we use, XGBoost was the first to be released. Chen and Guestrin released XGBoost in 2016. XGBoost offers several enhancements to the GBDT technique. The first enhancement is an improved loss function used during the training phase. The loss function contains an additional term for regularization to prevent overfitting. Another enhancement XGBoost makes to GBDTs is one that has to do with calculating splits in the constituent decision trees of the GBDT ensemble. Chen and Guestrin introduce the so called "approximate algorithm" which is a technique for estimating optimal values of splits. The approximate algorithm is suitable for distributed environments, as well as applications where the entire dataset does not fit in main memory. A third enhancement in XGBoost is another algorithm for finding splits that works well with sparse data. Sparse data is the type of data that is nearly constant in value with infrequently occurring aberrations. XGBoost can take advantage of sparse data with its "sparsity aware split finding" feature.

Ke et al. released the seminal paper on LightGBM in 2017 [13]. Their goal was to offer a GBDT implementation that yields performance equivalent to XGBoost, while consuming fewer resources. In order to achieve their goal, Ke et al. make two key enhancements to the GBDT technique. The first is Exclusive Feature Bundling (EFB). EFB is a technique for reducing the dimensions of a dataset by combining two features (attributes) of a dataset into a single feature. EFB is an effective technique for sparse data. When two attributes of a dataset exhibit sparsity, and the infrequently occurring values of both attributes are mutually exclusive, they may be safely combined into a single feature without the loss of information. EFB reduces the number of dimensions of a dataset, which helps reduce training time. LightGBM's second enhancement to the GBDT technique is called Gradient-based One-Side Sampling (GOSS). GOSS is a technique for intelligently reducing the number training instances used. GOSS selects instances for training based on their contribution to the loss function that is calculated as part of fitting the GBDT ensemble to the training data. If the instance contributes more than a configurable threshold value to the loss of the model, then it is retained for further iterations of the fitting process. Likewise, instances that contribute less than the threshold amount are set aside. Via GOSS and EFB Ke et al. deliver a GBDT implementation that consumes fewer computing resources.

The third GBDT implementation we use is CatBoost [11]. Prokhorenkova et al. introduced CatBoost in 2018. One may find more information on applications of CatBoost in various domains in [35]. Their motivation for developing CatBoost was to prevent overfitting. The first protection against overfitting that CatBoost makes is Ordered Boosting.

Ordered Boosting is a technique for selecting training instances. There are two steps for adding a decision tree to the GBDT ensemble. The first is to fit the decision tree to the dependent variable in the training data. Multiple decision trees are fit to different samples of the training data. After the trees have been fit, they must be evaluated in order to select the tree that best enhances the overall performance of the ensemble. Under Ordered Boosting one can be sure that training instances used to fit the decision tree will not be used to evaluate it for inclusion into the ensemble. This helps prevent the ensemble from being overfit to the training data. The second kind of overfitting CatBoost offers protections against is through its Ordered Target Statistics method of encoding categorical features. In simple target encoding, a categorical feature is assigned the mean value of the dependent variable that the feature is observed to co-occur with. This strategy for encoding may lead to information leakage in the sense that if the encoded feature co-occurs with different values of the dependent variable in the test data the encoded feature will not be a useful predictor of the dependent variable. To avoid this issue, Ordered Target Statistics is a technique for ensuring that the encoded value for a categorical feature of a given instance is derived from other instances. Put another way, the encoded value of a categorical feature is not allowed to be calculated from the label it appears with. This makes it impossible for the encoded feature value of the instance to be directly related to the value of the dependent variable. This is a protection against what Prokhorenkova et al. call "target leakage".

The five ensemble techniques we employ here lend robustness to our experimental design. Using different learners allows us to rule out the possibility that patterns we see in the results are due to the peculiarities of one model. ET and Random Forest are enhancements to Breiman's Bagging technique. XGBoost, LightGBM, and CatBoost are enhancements to Freidman's Gradient Boosting technique. In the next section we explain how we prepare the data we use as input to these learners.

Data description and preparation

As mentioned in the introduction, the CMS provides the data used in this study. The most recent CMS data we use in constructing all three datasets first became available in 2021. The latest Part B, Part D, and DMEPOS data all spans the years 2013 through 2019. Previous studies on Medicare fraud detection use data that covers fewer years. Moreover, some of the attributes of the latest data are not available in previous studies where older versions of the Part B, Part D, and DMEPOS are used. For example, in "Leveraging lightgbm for categorical big data" [36] we report our DMEPOS data has nine features, whereas one can see in Table 1 our current version of the DMEPOS data has 18 features. Furthermore, in [36], Part B data is reported to have eight features, and Part D data is reported to have seven features. In Table 2 we show that the Part B data we use has 16 features, and in Table 3, we show our Part D Data has 9 features. To the best of our knowledge, we are the first to employ the most recently available data from CMS in a study.

We use the same process for labeling each of the three datasets with data from the List of Excluded Individuals and Entities (LEIE) [37]. The United Sates Office of the Inspector General publishes the LEIE on a monthly basis. If a healthcare provider appears in the LEIE, this means the provider has been convicted of some activity which prevents

Table 1 Features of the DMEPOS Dataset

Feature name	Description
Rfrg_Prvdr_Crdntls	The referring provider's credentials [example: MD]; categorical, 7,315 distinct values
Rfrg_Prvdr_Gndr	The referring provider's gender; categorical, three distinct values
Rfrg_Prvdr_Ent_Cd	Type of entity [individual or organization] reported in NPPES; categorical, 2 distinct values
Rfrg_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
Rfrg_Prvdr_Type_Flag	A flag variable that indicates the source of the Referring Provider Type; categorical two distinct values
BETOS_Lvl	High level grouping of the Berenson-Eggers Type of Service (Berenson-Eggers Type of Service) Classifications into three groups including Durable Medical Equipment, Prosthetic and Orthotic Devices, and Drugs and Nutritional Products; categorical three distinct values
BETOS_Cd	The BETOS classification code assigned to the HCPCS Healthcare Common Procedure Coding System code; categorical, 12 distinct values
BETOS_Desc	Description of the HCPCS code for the specific product or service furnished by the DMEPOS supplier; categorical, 12 distinct values
HCPCS_Cd	HCPCS code for the specific product or service furnished by the DMEPOS supplier; categorical, 1337 distinct values
HCPCS_Desc	Description of the HCPCS code for the specific product or service furnished by the DMEPOS supplier; categorical, 1,618 distinct values
Suplr_Rentl_Ind	Identifies whether the DMEPOS product/service submitted on the supplier's claim is rental or non-rental; categorical 2 distinct values
Tot_Suplrs	Number of suppliers rendering DMEPOS products/services ordered by the referring provider
Tot_Suplr_Benes	Number of beneficiaries associated with the supplier DMEPOS products/services ordered by the referring provider
Tot_Suplr_Clms	Number of DMEPOS claims submitted by the supplier, reflecting products/services ordered by the referring provider
Tot_Suplr_Srvcs	Number of DMEPOS products/services rendered by the supplier
Avg_Suplr_Sbmtd_Chrg	Average of the charges that suppliers submit for DMEPOS products/services
Avg_Suplr_Mdcr_Alowd_Amt	Average Medicare allowed amounts for the DMEPOS product/service rendered by suppliers
Avg_Suplr_Mdcr_Pymt_Amt	Average amount that Medicare paid suppliers after deductible and coinsurance amounts have been deducted for the line item DMEPOS product/service

the provider from submitting insurance claims to Medicare. Records in the LEIE, Part B, Part D, and DEMEPOS data sources have a National Provider ID (NPI) in common. There are different types of exclusions that a provider may fall under. The types of exclusions we consider to be indicative of fraud are those identified by Bauder and Khoshgoftaar [38]. When a provider appears in the LEIE under any of these exclusions, we label all records belonging to that provider in the Part B, Part D, and DMEPOS data as fraudulent.

Exclusions have start and end dates. We label all of a provider's claims data pertaining to dates prior to the end of the exclusion period as fraudulent. Since the Part B, Part D, and DMEPOS data consists of records pertaining to entire years, and exclusion periods end at specific months, we round the end of the exclusion period to the nearest year. Once a provider's exclusion period is over, the provider is removed from the LEIE. The provider may once again submit claims to Medicare, so data that pertains to claims the provider submits after the end of the exclusion period is labeled as not fraudulent. If one

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 7,738 distinct values
HCPCS_Desc	Description of the HCPCS code for the specific medical service furnished by the provider; categorical, 8,252 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

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, 3,907 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, 22,72 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

is compiling a dataset that spans all the available years, the current LEIE will not contain records of providers that were in the LEIE previously. Therefore, one should use a utility such as Internet Archive Tool[2] to retrieve previous versions of the LEIE to label older records of the CMS data.

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

The dataset we derive from the DMEPOS data is the smallest, containing 12,215,370 instances. The original fraction of minority instances in the DMEPOS data is 0.0044. The data pertains to three classes of healthcare expenditures: durable medical equipment, prosthetics orthotics and supplies, and drugs and nutrition-related products. The DMEPOS data has a HCPCS code feature that identifies the expenditure item precisely. Each record in the DMEPOS data is a summary of insurance claims the provider sent to Medicare for the item identified by the HCPCS code for the year. Please see Table 1 for descriptions of the elements of the DMEPOS data we use. The descriptions of the features are copied from the DMEPOS data dictionary [39]. We augment the descriptions with information on the categorical features in the DMEPOS data. Any feature that we document as categorical is encoded with CatBoost encoding during experiments. In the DMEPOS data, as well as data for other parts of the program, there are several attributes that are not suitable for Machine Learning. These are attributes that pertain to the identity of providers, but provide no description of their activities or practices. These are the National Provider ID and other attributes related to the providers' address.

The next largest dataset we use is compiled from the Part B data. The Part B data describes treatments and procedures that a provider performs for patients. One record of the Part B data represents a summary of all the times a provider rendered a particular treatment or procedure for their patients for the year. The treatment or procedure is identified by the HCPCS code listed in the record. Table 2 contains names and descriptions of elements of the Part B data we use in this study. We copy the definitions from the Part B data dictionary [40], and add additional information about the number of possible values for categorical features. Our strategy for adopting elements of the Part B data is similar to the one we employ for the DMEPOS data. We discard the NPI and location data that would serve as the equivalent of a unique identifier for the provider. The final number of instances we obtain for the Part B data is 67,856,547. The fraction of instances of the minority class in the Part B data is 0.0019.

The Part D data comprises the largest of the three datasets that we work with. The Part D data pertains to medications that providers prescribe for their patients. A record of the Part D data relates to one particular medication that a provider prescribed for patients for 1 year. As with the DMEPOS and Part B data, we discard the NPI and location data that could interfere with a Machine Learning Model's ability to generalize. Table 3 summarizes all the Part D data elements we use in this study. Similar to Tables 1 and 2, we copy data definitions from the Part D data dictionary [41], and augment them with information on distinct values of categorical features. Our finalized version of the Part D data contains 173,677,665 records. The fraction of instances in the minority class in the Part D data is 0.0039.

Methodology

To perform experiments, we run programs that train Machine Learning models. Then we employ the trained models to classify the DMEPOS, Part B, or Part D data. The programs are implemented in the Python language [42]. We rely on publicly available, open source libraries to provide implementations of the all classifiers used in this study. One may install any one of them using the standard Python package manager. This lends to the reproducibility of our work.

Due to the stochastic nature of Machine Learning algorithms (learners), in our experiments we do ten iterations of five-fold cross validation. We use unique seeds for the random number generators involved in the experiments to ensure unique initial conditions for every experiment. One round of fivefold cross validation produces one experimental outcome, consisting of an AUC value and an AUPRC value. We average these values across the five rounds and ten iterations. Therefore, Scikit-learn [43] is another library that is vital to the success of our work. It provides functions for performing five-fold cross validation and calculating the AUC and AUPRC metrics after classification is completed.

We use CatBoost encoding [44] for all learners except LightGBM since LightGBM has a built-in function for encoding categorical data. The CatBoost Encoder is another publicly available, open source library one can easily install with a Python package manager. We opted for a general purpose encoding method, since the focus of our study is on RUS and performance metrics. For a study on special purpose encoding techniques in the Medicare fraud detection application domain, please see [45]. The function of the CatBoost Encoder is to convert categorical features to floating point numbers, so they are suitable for use with the learners. The CatBoost encoder must be fit to data before it can be used to encode features. One must take care to fit the encoder to the training data only. Otherwise, the encoded features will contain information about the test data that may cause learners to overfit to the dataset, and yield unrealistically high performance metrics. This is commonly known as target leakage.

After encoding categorical features, before starting classification, we apply RUS to change the class ratio to induce one of the five minority:majority class ratios, 1:1, 1:3, 1:9, 1:27 and 1:81. During our literature review, we found most studies where RUS is applied utilize the 1:1 class ratio. This is our motivation for selecting it. We arrive at the remaining class ratios by iteratively tripling the size of the majority class. As a baseline for determining the effect of RUS, we also perform experiments where we leave the class ratio unchanged. We apply RUS to the training data only. Class ratios of the test data are left unchanged.

We use GPU implementations of Random Forest, XGBoost and CatBoost. In preliminary experiments we found the GPU implementations of these learners to be much faster than their CPU implementations. We attempted to use the GPU implementation of LightGBM, but we found the built-in encoding for Categorical features when run on GPUs is not compatible with high-cardinality categorical data. To the best of our knowledge, we are the first to apply a GPU implementation of Random Forest to the task of Medicare fraud detection in a study. The version Random Forest we use is part of the CuML library.[3]

Also, to the best of our knowledge, we are the first to apply the ET classifier to the task of automated Medicare fraud detection in a peer-reviewed study. ET is available as a component of the Scikit-learn library. Due to the robustness of results ET yields, we recommend it for use in future studies involving imbalanced Big Data. We did not find a GPU implementation of ET that is suitable for working with datasets as large as the ones we work with here. Since ET is based on Random Forest, one avenue for

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

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	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	24	Maximum tree depth
LightGBM		
num_threads	64	Maximum number of execution threads
Random Forest		
max_depth	32	Maximum tree depth
n_streams	8	Number of simultaneous GPU streams active during the fit phase

future research is to modify the CuML implementation of Random Forest to produce a GPU implementation of ET. Therefore, for our experiments we use a CPU implementation of ET.

In preliminary experiments, we found maximum tree depth to be an important hyperparameter to optimize in order to obtain the best results. The Scikit-learn implementations of Random Forest and ET have unlimited maximum tree depth as a default setting. We saw that these classifiers were initially outperforming CatBoost and XGBoost. According to their current on-line documentation, CatBoost and XGBoost both have a default maximum tree depth of 6 [46, 47]. We found it necessary to raise the maximum tree depth of CatBoost to 16 and the maximum tree depth of XGBoost to 24 in order to obtain the best results. The largest value for maximum tree depth that CatBoost will currently permit is 16. Due to resource limitations, we were not able to successfully execute experiments with XGBoost where the maximum tree depth was set to a value greater than 24. Table 4 contains the values of all the changed hyperparameter settings we use in our experiments. Hyperparameter values listed were discovered after doing tuning experiments. Hence, they represent the best settings we discovered for each classifier. We did not find it necessary to modify any hyperparameter settings for ET.

Our experiments are all executed in a distributed computing platform as batch processes. The nodes available to us on the platform have Intel Xeon Central Processing Units (CPUs) with 16 cores, 256 GB RAM per CPU and Nvidia V100 GPUs. A single node has sufficient resources to run any experiment covered here.

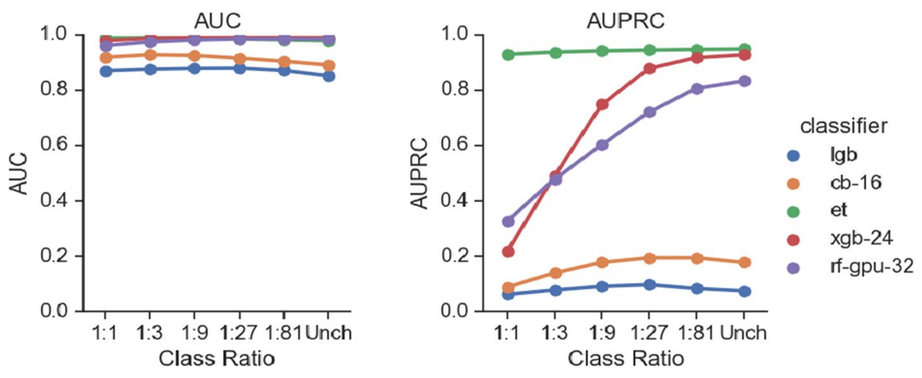

Fig. 1 DMEPOS Data: AUC Scores (left) and AUPRC scores (right)

Table 5 DMEPOS mean and standard deviation of AUC with varying levels of RUS (10 iterations of fivefold cross-validation)

Class ratio	LGB	CB-16	ET	XGB-24	RF-GPU-32
1:1	0.87070	0.91952	0.98862	0.97944	0.96163
	(0.00145)	(0.00134)	(0.00059)	(0.00077)	(0.00082)
1:3	0.87610	0.92829	0.98874	0.98732	0.97472
	(0.00161)	(0.00108)	(0.00072)	(0.00067)	(0.00082)
1:9	0.87922	0.92575	0.98774	0.98933	0.98175
	(0.00169)	(0.00152)	(0.00095)	(0.00069)	(0.00060)
1:27	0.88011	0.91607	0.98536	0.98992	0.98511
	(0.00165)	(0.00174)	(0.00111)	(0.00069)	(0.00066)
1:81	0.87140	0.90471	0.98207	0.98977	0.98549
	(0.00241)	(0.00192)	(0.00124)	(0.00063)	(0.00074)
Unchanged	0.85223	0.89143	0.97917	0.98959	0.98453
	(0.00492)	(0.00207)	(0.00113)	(0.00069)	(0.00104)

Standard deviations are below AUC scores in parenthesis

Results

We report AUC and AUPRC scores for classification results. The values for AUC and AUPRC reported here are mean values computed by averaging 50 experimental outcomes. One round of fivefold cross validation yields one experimental outcome consisting of one AUC and AUPRC score. Since we do 10 iterations of five-fold cross validation, we obtain 50 instances of each metric. We report the same data in graphical and tabular form. The graphical form shows relative performance and the trend in results as the size of the majority class is increased. In addition, we provide the data in tabular form along with standard deviations to give a sense of the spread of AUC and AUPRC scores over the ten iterations of five-fold cross validation.

As stated previously in the section on methodology, we apply RUS to induce a class ratio in the training data. We do not alter the class ratio in the test data. Therefore, scores one sees reported in this section are results for classifying data sampled from its original class ratio. Put another way, models are trained on data with RUS applied, and only evaluated on test data with the original class ratio.

The first results we report are the AUC and AUPRC scores the five classifiers yield for classifying the DMEPOS data. We provide plots of AUC and AUPRC scores in Fig. 1.

Table 6 DMEPOS mean and standard deviation of AUPRC with varying levels of RUS (10 iterations of fivefold cross-validation)

Class ratio	LGB	CB-16	ET	XGB-24	RF-GPU-32
1:1	0.06233	0.08935	0.92987	0.21863	0.32763
	(0.00319)	(0.00453)	(0.00267)	(0.00823)	(0.01293)
1:3	0.07842	0.13984	0.93779	0.49056	0.47900
	(0.00294)	(0.00422)	(0.00258)	(0.00875)	(0.01033)
1:9	0.09192	0.17826	0.94208	0.74880	0.60368
	(0.00305)	(0.00497)	(0.00265)	(0.00968)	(0.00678)
1:27	0.09788	0.19499	0.94493	0.87928	0.72180
	(0.00404)	(0.00510)	(0.00267)	(0.00534)	(0.00474)
1:81	0.08387	0.19489	0.94710	0.91788	0.80715
	(0.00566)	(0.00508)	(0.00301)	(0.00324)	(0.00387)
Unchanged	0.07531	0.17815	0.94866	0.92811	0.83319
	(0.00752)	(0.00474)	(0.00213)	(0.00316)	(0.00547)

Standard deviations are below AUPRC scores in parenthesis

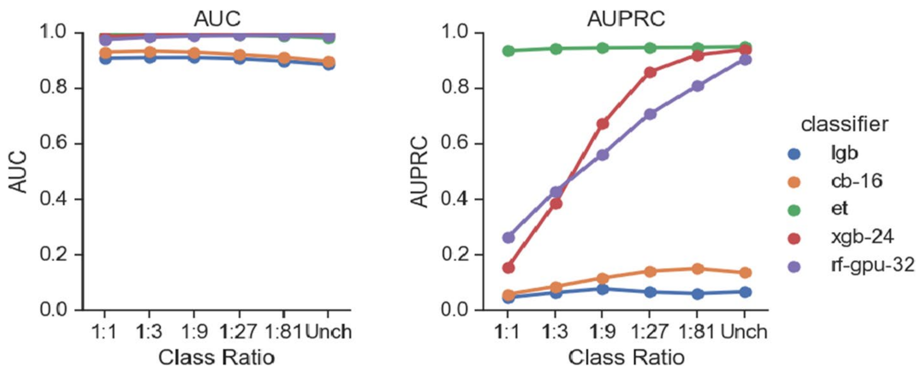

Fig. 2 Part B Data: AUC scores (left) and AUPRC scores (right)

Table 7 Part-B mean and standard deviation of AUC with varying levels of RUS (10 iterations of fivefold cross-validation)

Class ratio	LGB	CB-16	ET	XGB-24	RF-GPU-32
1:1	0.90846	0.93072	0.99316	0.98621	0.97534
	(0.00086)	(0.00087)	(0.00039)	(0.00039)	(0.00039)
1:3	0.91054	0.93427	0.99323	0.99219	0.98404
	(0.00078)	(0.00089)	(0.00037)	(0.00028)	(0.00038)
1:9	0.91087	0.93028	0.99259	0.99394	0.98849
	(0.00102)	(0.00101)	(0.00041)	(0.00025)	(0.00043)
1:27	0.90664	0.92148	0.99085	0.99443	0.99050
	(0.00241)	(0.00127)	(0.00041)	(0.00032)	(0.00041)
1:81	0.89803	0.91200	0.98764	0.99458	0.99069
	(0.00530)	(0.00104)	(0.00051)	(0.00028)	(0.00048)
Unchanged	0.88554	0.89698	0.98118	0.99436	0.98862
	(0.00542)	(0.00121)	(0.00080)	(0.00033)	(0.00051)

Standard deviations are below AUC scores in parenthesis

Table 8 Part-B mean and standard deviation of AUPRC with varying levels of RUS (10 iterations of fivefold cross-validation)

Class ratio	LGB	CB-16	ET	XGB-24	RF-GPU-32
1:1	0.04621	0.05853	0.93517	0.15584	0.26342
	(0.00253)	(0.00238)	(0.00209)	(0.00419)	(0.01037)
1:3	0.06426	0.08606	0.94335	0.38531	0.42815
	(0.00217)	(0.00236)	(0.00160)	(0.00786)	(0.00815)
1:9	0.07818	0.11695	0.94563	0.67304	0.56271
	(0.00513)	(0.00224)	(0.00129)	(0.00795)	(0.00532)
1:27	0.06664	0.14166	0.94682	0.85988	0.70862
	(0.00761)	(0.00309)	(0.00155)	(0.00376)	(0.00746)
1:81	0.06128	0.15084	0.94758	0.92030	0.80848
	(0.01261)	(0.00299)	(0.00150)	(0.00167)	(0.00270)
Unchanged	0.06768	0.13607	0.94990	0.93947	0.90510
	(0.01071)	(0.00254)	(0.00162)	(0.00156)	(0.00182)

Standard deviations are below AUPRC scores in parenthesis

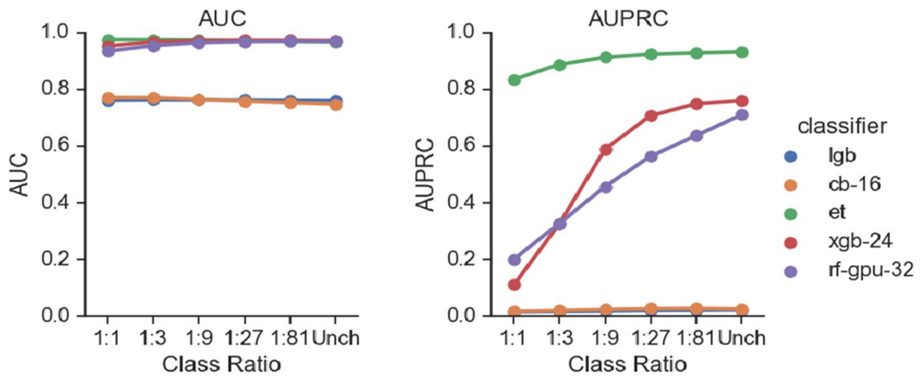

Fig. 3 Part D Data: AUC scores (left) and AUPRC scores (right)

Table 9 Part-D mean and standard deviation of AUC with varying levels of RUS (10 iterations of fivefold cross-validation)

Class ratio	LGB	CB-16	ET	XGB-24	RF-GPU-32
1:1	0.76230	0.77169	0.97625	0.95292	0.93523
	(0.00067)	(0.00075)	(0.00033)	(0.00059)	(0.00046)
1:3	0.76326	0.77125	0.97586	0.96809	0.95465
	(0.00060)	(0.00074)	(0.00038)	(0.00045)	(0.00024)
1:9	0.76317	0.76571	0.97464	0.97256	0.96476
	(0.00055)	(0.00080)	(0.00038)	(0.00037)	(0.00040)
1:27	0.76279	0.75888	0.97244	0.97363	0.96868
	(0.00060)	(0.00062)	(0.00037)	(0.00037)	(0.00030)
1:81	0.76212	0.75329	0.96966	0.97348	0.96963
	(0.00064)	(0.00089)	(0.00035)	(0.00039)	(0.00037)
Unchanged	0.76085	0.74847	0.96746	0.97273	0.96996
	(0.00081)	(0.00076)	(0.00038)	(0.00042)	(0.00031)

Standard deviations are below AUC scores in parenthesis

Table 10 Part-D mean and standard deviation of AUPRC with varying levels of RUS (10 iterations of fivefold cross-validation)

Class Ratio	LGB	CB-16	ET	XGB-24	RF-GPU-32
1:1	0.01567	0.01766	0.83614	0.11168	0.20205
	(0.00015)	(0.00020)	(0.00307)	(0.00158)	(0.00322)
1:3	0.01724	0.02096	0.88826	0.32721	0.32806
	(0.00016)	(0.00021)	(0.00181)	(0.00572)	(0.00322)
1:9	0.01910	0.02451	0.91341	0.58986	0.45801
	(0.00025)	(0.00035)	(0.00103)	(0.00571)	(0.00312)
1:27	0.02095	0.02736	0.92451	0.70815	0.56448
	(0.00031)	(0.00039)	(0.00080)	(0.00375)	(0.00195)
1:81	0.02203	0.02818	0.92925	0.74950	0.63787
	(0.00030)	(0.00044)	(0.00073)	(0.00335)	(0.00283)
Unchanged	0.02294	0.02651	0.93288	0.76105	0.71047
	(0.00037)	(0.00054)	(0.00073)	(0.00406)	(0.00169)

Standard deviations are below AUPRC scores in parenthesis

Due to space limitations, we use the following abbreviations for classifier names in Tables 5, 6, 7, 8, 9, 10 and figures in this section. Classifier names are abbreviated as follows: CatBoost with maximum tree depth set to 16: CB-16, LightGBM: LGB, Extremely Randomized Trees: ET, and the GPU implementation of Random Forest with maximum tree depth set to 32 RF-GPU-32. In Fig. 1 one may notice some facts that hold for other data as well. Mean AUC scores are high, and show little impact of RUS as the size of the majority class increases. However, there is more variance in results in terms of the mean AUPRC metric. ET yields consistently strong performance as RUS is applied to make the majority class larger in the training data. Random Forest and XGBoost yield better performance in terms of AUPRC when we apply RUS to make the size of the majority class larger. CatBoost and LightGBM yield relatively poor performance in terms of AUPRC regardless of how RUS changes.

Next, we report results for classifying the Part B data, starting with Fig. 2. Results are similar to those we obtain for the DMEPOS data. Again, all classifiers show strong performance in terms of mean AUC. However, the mean AUPRC scores reveal a different picture altogether. Consistent with results for DMEPOS, we see ET yields consistently strong AUPRC scores. XGBoost and Random Forest AUPRC scores appear to be in some direct proportion with the size of the majority class in the training data. Also in keeping with their performance in classifying the DMEPOS data, LightGBM and CatBoost yield low AUPRC scores for any level of RUS.

Finally, we report classification results for Part D data. We notice in Fig. 3 that while mean AUC scores are high, with the larger dataset there is a bifurcation of performance in terms of AUC. XGBoost, ET, and Random Forest yield AUC scores that appear noticeably higher than those of LightGBM and CatBoost. For classifying the Part D data, mean AUPRC scores are generally lower than those we see for the Part B or DMEPOS data. However, the same pattern we saw in the results for the Part B and DMEPOS data appears again in the Part D data; ET yields consistently high results, Random Forest and XGBoost scores improve as we apply RUS to increase the size of the majority class, and LightGBM and CatBoost yield low scores relative to the other classifiers.

Table 11 ANOVA for RUS, CLF and Size as factors of performance in terms of AUC

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
RUS	5	0.06	0.01	14.21	*
CLF	4	16.97	4.24	4832.58	*
Size	2	4.70	2.35	2673.69	*
Residuals	4488	3.94	†		

* Indicates value is less than 1×10^{-4}, † indicates value is less than 1×10^{-2}

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

Group a consists of: 1:9, 1:3, 1:27
Group ab consists of: 1:81
Group bc consists of: 1:1
Group c consists of: Unchanged

We find AUPRC scores for CatBoost and LightGBM to be surprisingly low. As a validation step, we performed one additional experiment. We wrote a separate program, completely independent of the program that produced the results above, and we performed an experiment where we trained XGBoost, CatBoost and LightGBM on 80% of the shuffled Part B data without RUS, and use 20% of the shuffled Part B without RUS data as a test set. We then calculated the AUC and AUPRC scores for the three classifiers, and obtained results that align with the previous results we report here. In that experiment, CatBoost yields an AUC score of 0.89744, and an AUPRC score of 0.14076, XGBoost yields an AUC score of 0.99407, and an AUPRC score of 0.93796, LightGBM yields and AUC score of 0.87017, and an AUPRC score of 0.07413. We include these results to show due diligence in ruling out any software bugs causing the extreme differences in AUPRC scores we report.

Statistical analysis

In order to make an informed decision on the results of the previous section, we apply statistical analysis in the form of Analysis of Variance (ANOVA) [48] and Tukey's Honestly Significant Difference (HSD) [49] tests. The ANOVA tests tell us whether a factor has a significant effect on experimental outcomes. When the ANOVA test identifies that a factor has a significant impact, we can then perform an HSD test to rank the levels of the factor in terms of its impact on experimental outcomes. Here, the outcome is either an AUC score or an AUPRC score. For all statistical tests, we use a significance level of $\alpha = 0.01$.

The first outcome we perform analysis for is the AUC score. This is analysis of the variance in AUC scores for all classifiers, all datasets, and all levels of RUS. In all ANOVA tables we present, CLF indicates the classifier factor, RUS indicates the Random Undersampling factor, and Size indicates the number of instances in the dataset before RUS is applied. Please see Table 11 for the ANOVA test results. Since the Pr(>F), or p-values associated with each factor are practically zero, we conclude that all factors have

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

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

Table 14 HSD test groupings after ANOVA of AUC for the size factor

Group a consists of: Medium
Group b consists of: Small
Group c consists of: Large

Table 15 ANOVA for RUS, CLF and size as factors of performance in terms of AUPRC

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
RUS	5	44.65	8.93	713.68	*
CLF	4	501.08	125.27	10011.73	*
Size	2	9.62	4.81	384.50	*
Residuals	4488	56.16	0.01		

* Indicates value is less than 1×10^{-4}

a significant effect on experimental outcomes. Therefore, we can conduct HSD tests to rank the levels of each factor in terms of its impact on AUC score.

HSD test results separate factors into groups such that levels of a factor that have a similar impact on performance are placed into the same group. The group that is associated with the highest value of the experimental outcome is labeled as group 'a', lower ranked groups are labeled with letters that follow in alphabetical order. If there is overlap in the performance associated with two groups, then overlapping groups will be labeled with letters in common. For example, in Table 12, the results for the 1:1 level of the RUS factor are placed in group 'ab', and results for the 1:81 level of the RUS factor are placed in group 'bc'. This implies the intersection of confidence intervals for AUC scores associated with these levels of RUS overlap.

The HSD test results in Table 12 show that applying RUS to induce class ratios of 1:9, 1:3, or 1:27 yields the best performance. This is important to bear in mind, since the HSD tests for the impact of RUS on AUPRC scores will show there is a negative impact on performance.

The next HSD test we undertake is to determine which classifier yields the best performance. The results here are processed for all datasets, and all levels of RUS. Here we see ET and XGBoost yield the best performance (Tables 13, 14).

The final HSD test we can conduct is for the size factor. We have three datasets of varying size. The HSD test results do not reveal a trend on the impact of size. We see that the best AUC scores are associated with the medium size, Part B, dataset. However, the second best AUC scores are associated with the small size, DMEPOS dataset. Finally, the lowest AUC scores are coupled with the largest, Part D dataset.

Table 16 HSD test groupings after ANOVA of AUPRC for the RUS factor

Group a consists of: Unchanged, 1:81
Group b consists of: 1:27
Group c consists of: 1:9
Group d consists of: 1:3
Group e consists of: 1:1

Table 17 HSD test groupings after ANOVA of AUPRC for the CLF 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
Group e consists of: LGB

Table 18 HSD test groupings after ANOVA of AUPRC for the size factor

Group a consists of: Small
Group b consists of: Medium
Group c consists of: Large

Here we begin a series of statistical tests similar to the previous section, only now the experimental outcome we are analyzing is AUPRC instead of AUC. As is the case with the previous analysis of AUC scores, results here are obtained by processing experimental outcome data over all factors: data set size, RUS and classifier, for the Part B, Part D, and DMEPOS data. The factors in Table 15 are the same as in Table 11. Similar to Table 11, the Pr(>F) values for all factors in Table 15 are practically zero, which means each factor has a significant impact on AUPRC scores.

Since all factors have a significant impact on performance, we can conduct a Tukey HSD test to rank the levels of the factors in terms of their impact on performance. Here we see a clear relationship between the class ratio and AUPRC scores. The HSD results in Table 16 show that models built with larger number of majority class instances in the training data are associated with higher AUPRC scores. Since the results here are for AUPRC scores averaged across all datasets and learners, they imply that in general, RUS to induce a class ratio any larger than 1:81 yields worse performance in terms of AUPRC.

Next we report the results of the HSD test to rank the classifier factor. In keeping with the previous results for performance in terms of AUC, ET and XGBoost are associated with the best performance. However, we see in Table 17 that for performance in terms of AUPRC, ET is in a class by itself. ET and XGBoost are grouped together for best performance in terms of AUC.

Interestingly, for performance in terms of AUPRC, we see there is a trend in the size of the dataset and performance in terms of AUPRC. The smallest dataset is associated with the best performance in terms of AUPRC, as Table 18 reveals. Intuitively, the smallest dataset has the smallest negative class size. This could make it so there are fewer false

Table 19 ANOVA for RUS and CLF as factors of performance in terms of AUC

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
RUS	5	0.03	0.01	132.89	*
CLF	4	3.26	0.82	16241.00	*
Residuals	1490	0.07	0.00		

*Indicates the Pr(>F) value is less than 1×10^{-4}

Table 20 HSD test groupings after ANOVA of AUC for the RUS factor

Group a consists of: 1:9, 1:27, 1:3
Group b consists of: 1:81
Group c consists of: 1:1
Group d consists of: Unchanged

Table 21 HSD test groupings after ANOVA of AUC for the CLF 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
Group e consists of: LGB

Table 22 ANOVA for RUS and CLF as factors of performance in terms of AUPRC

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
RUS	5	14.37	2.87	260.65	*
CLF	4	161.84	40.46	3669.03	*
Residuals	1490	16.43	0.01		

*Indicates value is less than 1×10^{-4}

positives to bring down the precision score. It is an interesting question for future work as to whether we might see less of an impact of RUS on performance in terms of AUPRC for small imbalanced datasets than large imbalanced datasets.

Now we move on to analyze the effect of classifier and RUS on experimental outcomes for individual datasets. The first dataset we report on is the DMEPOS dataset. First we present the ANOVA table for the analysis of the variance in AUC scores that the classifier and RUS factors contribute to experimental outcomes. The ANOVA test results in Table 19 show both RUS and classifier factors have a significant impact on AUC scores.

Since both the choice of classifier and RUS have a significant impact on experimental outcomes, we can rank both factors with HSD test. The first test we do is to rank the RUS factors. The result of the HSD test is in Table 20. Similar to the previous case where we looked at the impact of RUS across all three datasets, we see that for the DMEPOS data, RUS to induce class ratios of 1:9, 1:27, and 1:3 yield the best performance in terms of AUC.

Table 23 HSD test groupings after ANOVA of AUPRC for the RUS factor

Group a consists of: Unchanged, 1:81, 1:27
Group b consists of: 1:9
Group c consists of: 1:3
Group d consists of: 1:1

Table 24 HSD test groupings after ANOVA of AUPRC for the CLF 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
Group e consists of: LGB

Table 25 ANOVA for RUS and CLF as factors of performance in terms of AUC

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
RUS	5	0.03	0.01	160.56	*
CLF	4	2.21	0.55	13222.85	*
Residuals	1490	0.06	0.00		

*Indicates the value is less than 1×10^{-4}

Table 26 HSD test groupings after ANOVA of AUC for the RUS factor

Group a consists of: 1:9, 1:3
Group b consists of: 1:27
Group c consists of: 1:1
Group d consists of: 1:81
Group e consists of: Unchanged

Next we rank the classifiers in terms of their impact on AUC scores when classifying the DMEPOS data. This ranking is in Table 21. In this case XGBoost is the top performer; however, the pattern that the two best classifiers are XGBoost and ET continues to hold.

Here we take a look at how the RUS and classifier factors influence the outcome of DMEPOS classification as measured by the AUPRC score. The ANOVA results recorded in Table 22 show both factors have a significant impact on AUPRC scores. Therefore, HSD tests are worthwhile to conduct.

The impact of RUS on AUPRC scores for the classification of DMEPOS data is similar to what we find for the classification over all datasets. The outcome of the HSD test is reported in Table 23. We see not applying RUS at all, and applying it to induce class ratios of 1:81 or 1:27 does not yield a significant difference in the best possible results.

Table 27 HSD test groupings after ANOVA of AUC for the CLF 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
Group e consists of: LGB

Table 28 ANOVA for RUS and CLF as factors of performance in terms of AUPRC

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
RUS	5	18.29	3.66	242.75	*
CLF	4	171.20	42.80	2839.60	*
Residuals	1490	22.46	0.02		

*Indicates the values is less than 1×10^{-4}

Table 29 HSD test groupings after ANOVA of AUPRC for the RUS factor

Group a consists of: Unchanged
Group ab consists of: 1:81
Group b consists of: 1:27
Group c consists of: 1:9
Group d consists of: 1:3
Group e consists of: 1:1

Next we take a look at the impact of the choice of classifier on AUPRC scores for the classification of DMEPOS data. The result of the HSD test in Table 24 reflects the trend that XGBoost and ET yield the best performance.

Here we start the analysis of results for classification of the Part B data. Table 25 contains the results of the ANOVA test for the impact of the classifier and RUS factors on AUC scores recorded for classification of the Part B data. Since the Pr(>F) values for both factors are both practically zero, we conclude both factors have a significant impact on AUC scores.

Since the ANOVA test results in Table 25 show RUS has a significant impact, we conduct an HSD test to determine which level of RUS is associated with the best performance. The result of the HSD test is in Table 26. The HSD test results for all datasets and the DMEPOS dataset associate RUS at the 1:27 level with the best performance. However, for the Part B data, we see that 1:3 and 1:9 are associated with the best performance.

In Table 27 we have the results of the HSD test for the classifier factor. In keeping with their performance across all datasets, and the DMEPOS dataset, ET and XGBoost are the two best performing classifiers.

Now we come to the analysis of classification results for the Part B data. To start the analysis of the impact of RUS and classifier on the AUPRC scores we recorded in our experiments with the Part B data, we ran an ANOVA test. The outcome of the ANOVA test is listed in Table 28. Since the Pr(>F) values are very small, we conclude both the RUS and CLF factors have a significant impact on AUPRC scores.

Table 30 HSD test groupings after ANOVA of AUPRC for the CLF 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
Group e consists of: LGB

Table 31 ANOVA for RUS and CLF as factors of performance in terms of AUC

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
RUS	5	0.01	0.00	45.53	*
CLF	4	15.21	3.80	72803.06	*
Residuals	1490	0.08	†		

*Indicates the value is less than 1×10^{-4}, †indicates the value is less than 1×10^{-3}

Table 32 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: Unchanged
Group d consists of: 1:1

Table 33 HSD test groupings after ANOVA of AUC for the CLF factor

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

Since the ANOVA test shows RUS has a significant effect on AUPRC scores, we can use an HSD test to rank levels of the RUS factor. This will determine which level of RUS yields the best performance. For the Part B data, we see not applying RUS yields the best performance. This result is recorded in Table 29.

The ANOVA test also shows that the classifier has a significant impact on AUPRC scores for the classification of Part B data. Therefore, we conduct an HSD test to rank the classifiers. In Table 30 we have the now-familiar results of ET doing the best in terms of AUPRC scores, and XGBoost coming in second.

Here we begin the analysis of results for the largest of the three datasets, the Part D data. The first set of results we analyze is for the AUC scores recorded for experiments in classifying the Part D data. The resulting ANOVA table is Table 31. It shows that both RUS and classifier choice have a significant impact on performance.

Table 34 ANOVA for RUS and CLF as factors of performance in terms of AUPRC

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
RUS	5	12.38	2.48	248.02	*
CLF	4	170.02	42.51	4256.20	*
Residuals	1490	14.88	0.01		

*Indicates value is less than 1×10^{-4}

Table 35 HSD test groupings after ANOVA of AUPRC for the RUS factor

Group a consists of: Unchanged
Group ab consists of: 1:81
Group b consists of: 1:27
Group c consists of: 1:9
Group d consists of: 1:3
Group e consists of: 1:1

Table 36 HSD test groupings after ANOVA of AUPRC for the CLF 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, LGB

Since the ANOVA test shows that RUS has a significant impact, we can rank the RUS factors by their effect on AUC scores. In Table 32 we see only the 1:9 RUS level is associated with the best performance.

The ANOVA test result for the classifier factor also implies that it has a significant impact on AUC scores. Interestingly, in Table 33 we see that ET is associated with the best performance in terms of AUC. For other datasets, and across all datasets, XGBoost is associated with the best performance.

The last statistical analysis we perform is for the effect of RUS and classifier on AUPRC scores for the classification of the Part D data. The outcome of the ANOVA test is listed in Table 34. The results show both factors have a significant impact on AUPRC scores.

Table 35 shows that not applying RUS yields the best AUPRC scores. This is consistent with results for other datasets. We see consistently that other levels of RUS are associated with lower AUPRC scores for other datasets.

The HSD result for the effect of the classifier on AUPRC scores is in Table 36. As the in cases of other datasets, ET is associated with the highest AUPRC scores for classifying the Part D data as well.

Conclusion

We have presented a thorough review of experiments for Medicare fraud detection in three highly imbalanced Big Data datasets. The three datasets, DMEPOS, Part B, and Part D, range in size from about 12 million to 175 million instances. To the best of our knowledge, we are the first to present a study on the latest versions of all three datasets. The original class ratio of the DMEPOS data is 0.0044, Part B data is 0.0019, and the Part D data is 0.0039. Our primary goal in compiling this study is to show that the AUC metric does not give a clear signal on the negative impact of RUS in the classification of highly imbalanced Big Data. RUS is a tempting technique for addressing class imbalance with Big Data since it lowers resource consumption. However, our results and statistical analyses show that applying RUS to change class ratios appears to have a positive impact on AUC scores. At the same time, for some classifiers, we see a significant drop in AUPRC scores as we apply RUS to make the majority class closer in size to the minority class. Moreover, we find that although other classifiers yield relatively high AUC scores, they yield relatively low AUPRC scores, regardless of the level of RUS used.

For highly imbalanced Big Data, we find that the size of the majority class overwhelms other terms in the calculation of the false positive rate, and therefore hides the detrimental effect of RUS. This in turn hides the effect of RUS in the calculation of AUC as well. Since the AUPRC metric involves recall, and not the false positive rate, the impact of RUS is easier to detect in AUPRC scores. Moreover, as Table 18 illustrates, the size of the majority class magnifies the impact on AUPRC scores. Therefore, the size of the imbalanced dataset is an important consideration when selecting a performance metric. Two interesting avenues for future work are investigation into whether RUS has a negative impact on the classification of smaller imbalanced Big Data, and whether there exist any classifiers that exhibit the same robustness to RUS that ET does.

The choice of classifier has a significant effect on our experimental outcomes. XGBoost and ET are consistently the best classifiers, both for performance in terms of AUC and AUPRC. It is interesting to note that were we to select XGBoost and ET on the basis of their performance in terms of AUC, we would find they both give strong performance for all levels of RUS. However, the AUPRC scores would more accurately inform which classifier performs better. Overall, we find ET is the more robust of the two classifiers, since it is the least impacted by RUS. To the best of our knowledge, we are the first to apply ET in the domain of Medicare fraud detection. All classifiers appear to do well when we look at AUC scores. However, when we look at AUPRC scores we see differences in classifier performance. XGBoost and Random Forest show improved performance as the class ratio reduces from 1:1 to its original highly imbalanced ratio. CatBoost and LightGBM yield relatively poor performance in terms of AUPRC. The ensemble technique for classifiers does not appear to have an effect on outcome, since LightGBM, CatBoost, and XGBoost are all GBDT implementations, and Random Forest and ET are applications of a bagging technique. Moreover, we see the performance of CatBoost and LightGBM, in terms of AUPRC, appears to diminish also as the size of the dataset increases. Overall, we see RUS has the smallest effect on ET. We conjecture that ET's random split selection makes it robust to the random deletion of instances of the majority class that we perform when doing RUS. Our results can be summarized as follows: in the classification of the highly imbalanced Part B, Part D, and DMEPOS Big Data, AUPRC shows that RUS can have a detrimental effect on performance, whereas AUC does not show this detrimental effect, and ET's performance in terms of AUC is more robust to RUS than other learners.

Abbreviations

ADASYN	Adaptive synthetic sampling technique
ANOVA	Analysis of variance
AUC	Area Under the Receiver Operating Characteristic Curve
AUPRC	Area Under the Precision Recall Curve
CMS	Centers for medicare and medicaid services
DMEPOS	Durable Medical Equipment Prosthetics Orthotics and Supplies
EFB	Exclusive feature bundling
GOSS	Gradient based one-side sampling
GPU	Graphics processing unit
HSD	Honestly significant difference
RUS	Random Undersampling
SMOTE	Synthetic minority oversampling technique

Acknowledgements
The authors would like to thank the various members of the Data Mining and Machine Learning Laboratory, Florida Atlantic University, for their assistance with the reviews.

Author contributions
JTH conducted experiments, and wrote the manuscript. JMJ prepared all datasets, and provided reviews and input for the manuscript. TMK provided oversight of experiments, coordinated research, and provided input for the manuscript. All authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 25 September 2022 Accepted: 28 March 2023
Published online: 11 April 2023

References

1. Bekkar M, Djemaa HK, Alitouche TA. Evaluation measures for models assessment over imbalanced data sets. J Inf Eng Appl. 2013;3:10.
2. Boyd K, Eng KH, Page CD. Area under the precision-recall curve: point estimates and confidence intervals. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer: New York; 2013. p. 451–66.
3. The Centers for Medicare and Medicaid Services: Medicare Durable Medical Equipment, Devices & Supplies – by Referring Provider and Service. 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 2021.
4. The Centers for Medicare and Medicaid Services: Medicare Physician & Other Practitioners – by Provider and Service. https://data.cms.gov/provider-summary-by-type-of-service/medicare-physician-other-practitioners/medicare-physician-other-practitioners-by-provider-and-service. 2021.
5. The Centers for Medicare and Medicaid Services: Medicare Part D Prescribers – by Provider and Drug. https://data.cms.gov/provider-summary-by-type-of-service/medicare-part-d-prescribers/medicare-part-d-prescribers-by-provider-and-drug 2021.
6. De Mauro A, Greco M, Grimaldi M. A formal definition of big data based on its essential features. Library Review 2016.
7. Civil Division, U.S. Department of Justice: Fraud Statistics, Overview. https://www.justice.gov/opa/press-release/file/1354316/download 2020.
8. Centers for Medicare and Medicaid Services: 2019 Estimated Improper Payment Rates for Centers for Medicare & Medicaid Services (CMS) Programs. https://www.cms.gov/newsroom/fact-sheets/2019-estimated-improper-payment-rates-centers-medicare-medicaid-services-cms-programs 2019.
9. 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;137–142. IEEE
10. Zuech R, Hancock JT, Khoshgoftaar TM. Detecting web attacks using random undersampling and ensemble learners. J Big Data. 2021;8(1):1–20.

11. Prokhorenkova L, Gusev G, Vorobev A, Dorogush AV, Gulin A. Catboost: unbiased boosting with categorical features. Adva Neural Inf Process Syst. 2018;31:8.
12. Chen T, Guestrin C. Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16 2016. https://doi.org/10.1145/2939672.2939785.
13. Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu T-Y. Lightgbm: A highly efficient gradient boosting decision tree. Adva Neural Inf Process Syst. 2017;30:3146–54.
14. Breiman L. Random forests. Mach Learn. 2001;45(1):5–32.
15. Geurts P, Ernst D, Wehenkel L. Extremely randomized trees. Mach Learn. 2006;63(1):3–42.
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. Springer: Berlin; 2005. p. 878–87.
17. He H, Bai Y, Garcia EA, Li S. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In: 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), 2008;1322–1328. IEEE
18. Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ, et al. Apache spark: a unified engine for big data processing. Commun ACM. 2016;59(11):56–65.
19. Le Cessie S, Van Houwelingen JC. Ridge estimators in logistic regression. J R Stat Soc. 1992;41(1):191–201.
20. Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S, et al. Mllib: Machine learning in apache spark. J Mach Learn Res. 2016;17(1):1235–41.
21. Hasanin T, Khoshgoftaar TM, Leevy JL, Bauder RA. Severely imbalanced big data challenges: investigating data sampling approaches. J Big Data. 2019;6(1):1–25.
22. Kuncheva LI, Arnaiz-Gonzalez A, Díez-Pastor J-F, Gunn IA. Instance selection improves geometric mean accuracy: a study on imbalanced data classification. Prog Artif Intell. 2019;8(2):215–28.
23. He H, Garcia EA. Learning from imbalanced data. IEEE Trans Knowl Data Eng. 2009;21(9):1263–84.
24. Sleeman WC IV, Krawczyk B. Multi-class imbalanced big data classification on spark. Knowl-Based Syst. 2021;212: 106598.
25. Calvert CL, Khoshgoftaar TM. Threshold based optimization of performance metrics with severely imbalanced big security data. In: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), 2019. p. 1328–34.
26. Johnson JM, Khoshgoftaar TM. Medicare fraud detection using neural networks. J Big Data. 2019;6(1):1–35.
27. Hancock JT, Khoshgoftaar TM. Survey on categorical data for neural networks. J Big Data. 2020;7(1):1–41.
28. Johnson JM, Khoshgoftaar TM. The effects of data sampling with deep learning and highly imbalanced big data. Inf Syst Front. 2020;22(5):1113–31.
29. Apache Software Foundation: Hadoop. https://hadoop.apache.org.
30. Saito T, Rehmsmeier M. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. PloS one. 2015;10(3):0118432.
31. Breiman L. Bagging predictors. Machine learning. 1996;24(2):123–40.
32. Efron B, Tibshirani RJ. An Introduction to the Bootstrap. Boca Raton: CRC Press; 1994. p. 5–6.
33. Hasanin T, Khoshgoftaar TM. The effects of random undersampling with simulated class imbalance for big data. In: 2018 IEEE International Conference on Information Reuse and Integration (IRI), 2018. p. 70–9.
34. Friedman JH. Greedy function approximation: a gradient boosting machine. Ann Stat. 2001;34:1189–232.
35. Hancock JT, Khoshgoftaar TM. Catboost for big data: an interdisciplinary review. J Big Data. 2020;7(1):1–45.
36. Hancock JT, Khoshgoftaar TM. Leveraging lightgbm for categorical big data. In: 2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService), 2021. p. 149–154.
37. LEIE: Office of Inspector General Leie Downloadable Databases. https://oig.hhs.gov/exclusions/index.asp.
38. Bauder RA, Khoshgoftaar TM. A novel method for fraudulent medicare claims detection from expected payment deviations (application paper). In: 2016 IEEE 17th International Conference on Information Reuse and Integration (IRI), 2016. p. 11–19.
39. The Centers for Medicare and Medicaid Services: Medicare Durable Medical Equipment, Devices & Supplies – by Referring Provider and Service Data Dictionary. https://data.cms.gov/resources/medicare-durable-medical-equipment-devices-supplies-by-referring-provider-and-service-data-dictionary 2021.
40. The Centers for Medicare and Medicaid Services: Medicare Physician & Other Practitioners – by Provider and Service Data Dictionary. https://data.cms.gov/resources/medicare-physician-other-practitioners-by-provider-and-service-data-dictionary. 2021.
41. The Centers for Medicare and Medicaid Services: Medicare Part D Prescribers – by Provider and Drug Data Dictionary. https://data.cms.gov/resources/medicare-part-d-prescribers-by-provider-and-drug-data-dictionary 2021.
42. Van Rossum G, Drake FL. Python/C Api Manual-Python 3. CreateSpace 2009.
43. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, et al. Scikit-learn: Machine learning in python. J Mach Learn Res. 2011;12:2825–30.
44. McGinnis W. Category Encoders. https://contrib.scikit-learn.org/category_encoders/.
45. Johnson JM, Khoshgoftaar TM. Medical provider embeddings for healthcare fraud detection. SN Computer Sci. 2021;2(4):276.
46. XGBoost Parameters. XGBoost Developers. https://xgboost.readthedocs.io/en/stable/parameter.html Accessed 9 Jul 2022.
47. Parameters. Yandex Corporation. https://catboost.ai/en/docs/references/training-parameters/common. Accessed 9 Jul 2022.
48. Iversen GR, Norpoth H. Analysis of Variance, vol. 1. Newbury Park: Sage; 1987.
49. Tukey JW. Comparing individual means in the analysis of variance. Biometrics. 1949;56:99–114.