2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA) | 978-1-7281-8470-8/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICMLA51294.2020.00095

# Performance of CatBoost and XGBoost in Medicare Fraud Detection

John Hancock and Taghi M. Khoshgoftaar
College of Engineering and Computer Science
Florida Atlantic University
Boca Raton, Florida 33431
jhancoc4@fau.edu, khoshgof@fau.edu

***Abstract*—Due to the size of the data involved, performance is an important consideration in the task of detecting fraudulent Medicare insurance claims. We evaluate CatBoost and XGBoost on the task of Medicare fraud detection, and report performance in terms of running time and Area Under the Receiver Operating Characteristic Curve (AUC). We show that adding a categorical feature for XGBoost and CatBoost improves performance in terms of AUC, and that CatBoost's performance is higher in a statistically significant sense. Moreover, we conduct experiments to find the optimal number of decision trees to use for XGBoost and CatBoost in the task of Medicare fraud detection. This is an important contribution because the number of trees in the ensemble governs overall resource consumption of a Gradient Boosted Decision Tree implementation. We find that with a purely numerical dataset, CatBoost and XGBoost yield nearly equivalent performance in terms of AUC, and XGBoost has a shorter training time. With respect to Medicare fraud detection, to the best of our knowledge, this is the first study to evaluate the performance of CatBoost and XGBoost in terms of running time and AUC on highly imbalanced, Big Data. Our contribution of evaluating running time performance on a large imbalanced dataset benefits researchers looking for more efficient utilization of valuable resources.**

***Keywords*—Gradient Boosted Decision Trees, CatBoost, XGBoost, Class Imbalance, Medicare Fraud**

## 1. Introduction

In an attempt to commit fraud, criminals may target a system with a large volume of transactions. There is a chance that, hidden in a multitude of legitimate transactions, their crimes go unnoticed. One such system is the United States Federal Government's health insurance program, known as Medicare [9]. A large segment of the United States' population is eligible for Medicare. This segment includes people age 65 and over, people with certain disabilities, and others [9]. In 2019, Medicare had 61.2 million beneficiaries [10]. Hence, the agency in charge of the Medicare program, The Centers for Medicare and Medicaid (CMS), handles a large volume of insurance claims, or requests for payment. Some of these claims are fraudulent. To the best of our knowledge, no one knows exactly how much money fraudsters steal from CMS. However, in 2017, the United States Government Accountability Office (GAO) reported, "Although there are no reliable estimates of fraud in Medicare, in fiscal year 2017 improper payments for Medicare were estimated at about $52 billion" [4]. One implication of the GAO's statement is that, whether manual or automated, the resources of CMS are insufficient to detect fraud; otherwise, we would know how much money CMS paid on fraudulent claims. We wish to aid the effort of developing an automated solution to detecting fraudulent Medicare claims. What sets this study apart from previous work is that, to the best of our knowledge, this is the first study to compare the performance of CatBoost [27] and XGBoost [11] in terms of Area Under the Receiver Operating Characteristic Curve (AUC) [7] and running time in the task of Medicare fraud detection.

The question of running time performance of CatBoost versus XGBoost is not settled. In the seminal work on CatBoost, in Appendix C, Prokhorenkova et al. include a table [27, Tab 2], where they report a comparison of average decision tree construction times of XGBoost, CatBoost, and LightGBM [21]. The comparison of these three libraries is reasonable because they are all Gradient Boosted Decision Tree (GBDT) implementations. They report the best average tree construction times as 1.1 seconds for CatBoost, 1.1 seconds for LightGBM, and 3.9 seconds for XGBoost, for a problem involving the Epsilon [25] dataset. The dataset, environment, and hyper-parameter settings one uses for CatBoost and XGBoost impact their relative performance. For example, Spadon et al. report finding XGBoost to be 50 times faster per training iteration than CatBoost [29]. In an earlier study [15], we found CatBoost's performance in terms of AUC is better than XGBoost's, which further motivated us study which learner has better performance in terms of running time. This conflicting data on the running time performance of CatBoost versus XGBoost necessitates further investigation for different classes of applications. We contribute empirical data on the running time, performance in terms of AUC, and optimal ensemble sizes for the class of applications of XGBoost and CatBoost involving highly imbalanced, Big Data.

There exists previous work on the subject of using machine learning for Medicare fraud detection. Therefore,

978-1-7281-8470-8/20/$31.00 ©2020 IEEE
DOI 10.1109/ICMLA51294.2020.00095

in the next two sections we provide some background on related work. After that, in the section on methodology, we cover details on the data we use in our experiments. Following the data description, we explain our sampling techniques, and the configurations of XGBoost and CatBoost employed in our experiments. Then we report and discuss results, and give our conclusions.

## 2. Related Work

There exists a body of research on applying machine learning to the problem of detecting fraud in Medicare insurance claims. Here, we recount key publications in the line of work that leads to this study. These previous publications lead us to investigate whether CatBoost, with its support for working with categorical variables of high cardinality, is a suitable algorithm for Medicare fraud detection. One reason we took CatBoost into consideration is the related work by Johnson and Khoshgoftaar [20], where the authors assess the performance of Nueral Networks in the task of medicare fraud detection. This inspired us to look at GBDT learners for the same task.

We label the medicare claims data from CMS as fraudulent, or not fraudulent by matching the National Provider ID NPI to data in the List of Excluded Individuals and Entities LEIE [24]. The technique of automatically matching records in CMS data by NPI to records in the LEIE is described in Branting et al. [8]. In their work, published in August 2016, the authors derive graphs from publicly available data to predict the presence of a provider in the LEIE. The algorithms produce graphs that give a quantitative notion of either provider geospatial similarity or provider behavioral similarity. The authors then use this similarity to predict whether a provider is in the LEIE. Of particular interest to us is the authors' claim that an ablation analysis of their models' features shows that the features most important for accuracy are those related to geospatial co-location. We find a similar lift in accuracy when we include the National Plan and Provider Enumeration System (NPPES) state as a feature in the CatBoost input data.

In a comparison of XGBoost, LightGBM, and XGBoost, Punmiya and Choe report running time consumption of these learners in the task of Electricity theft detection [28]. Punmiya and Choe use a technique to generate synthetic data to overcome class imbalance, whereas we use random undersampling to overcome class imbalance. Moreover, the size of the dataset in [28] is not entirely clear, but it appears to be much smaller than the one we use. It is our understanding that the datasets they use consist of 5,000 examples, whereas ours consist of millions of examples.

GBDT implementations are central to this study. However, due to space limitations we forgo summarizing GBDT's or the CatBoost and XGBoost implementations of GBDT's. Please see Friedman [13] for further details on Gradient Boosted Decision Trees. For more details on CatBoost please consult [27]. For details on applications of CatBoost in various fields, please see [14]. And, for thorough coverage of XGBoost, please see Chen and Guestrin[11]. For a thorough comparison of learners other than Neural Networks, CatBoost, or XGBoost, in the task of Medicare Fraud detection, please see [23].

## 3. Methodology

In our experiments, similar to [19], [5], [17], [18] and [6], we form a dataset from two sources. The first source is data on claims providers submitted to Medicare. This data is furnished by CMS. The second source is a list of healthcare providers that are excluded from submitting claims to Medicare. This list is the LEIE data. The Office of Inspector General OIG publishes the LEIE data. Both OIG and CMS are organizations within the United Sates Federal Government. In this section, we provide details on these two data sources, how we combine them to form datasets, and how we conduct our experiments.

### 3.1. Data Sources

We use Medicare Part B insurance claims data for the years 2012 through 2015. This data originates from a series of publicly available files CMS publishes. We know of these files collectively as Medicare Provider Utilization and Payment Data: Physician and Other Supplier Public Use Files (PUF) [1]. CMS provides a document [12] that describes all the elements of this Medicare Part B insurance claims data. At a high level this data is in character separated value (csv) format, and there is one file for every year. Every record in the data identifies a provider, primarily by the provider's NPI, and secondarily by several elements that give details on the provider's name, demographics and location. Records also contain a provider type that describes the nature of the provider's practice, for example, ophthalmology. For every procedure the provider has submitted a claim to Medicare for, there is one record in the PUF file for the year. This procedure is represented with a Healthcare Common Procedure Coding System (HCPCS) code. In every record, along with the procedure code, there are some aggregate statistics pertinent to it, such as the number of times the provider performed the procedure for the year, and the average amount the provider bills Medicare for. Please see Table 1 for a description of the subset of features of the Medicare data we use for this study. For details on the Medicare Part B data, please see [17].

The second source of data we use is the LEIE data. The OIG updates the LEIE on a monthly basis. It is also a character separated value file. For a complete description of the LEIE please see [24]. For our purposes, the relevant features of the LEIE are the NPI, the exclusion type, the exclusion date, waiver data and the reinstatement date. In the LEIE file, these elements are named: NPI, EXCLTYPE, EXCLDATE, REINDATE, WAIVERDATE, and WVRSTATE, respectively.

We derive a label for the Medicare Part B data for a particular year from the LEIE data. If an NPI from the Medicare Part B data appears in the LEIE, and the year for the Medicare Part B data is less than the ending year

TABLE 1. FEATURES USED FOR ALL EXPERIMENTS

| Name | Description | Type |
|---|---|---|
| provider_type | Medical provider's specialty (or practice) | Categorical |
| nppes_provider_gender | Provider gender | Categorical |
| nppes_provider_state | The state where the provider is located | Categorical |
| hcpcs_code * | code used to identify the specific medical service furnished by the provider | Categorical |
| line_srvc_cnt | Number of procedures/services the provider performed | Numerical |
| bene_unique_cnt | Number of distinct Medicare beneficiaries receiving the service | Numerical |
| bene_day_srvc_cnt | Number of distinct Medicare beneficiary / per day services performed | Numerical |
| average_submitted _chrg_amt | Average of the charges that the provider submitted for the service | Numerical |
| average_medicare_ payment_amt | Average amount that Medicare paid after deductible and coinsurance amounts have been deducted for the line item service | Numerical |

Asterisk (*) indicates feature used only with CatBoost; Feature descriptions from [12]

of the exclusion period, then we label all Medicare Part B data for that year, having that NPI, as fraudulent. We define the ending year of the exclusion period as the exclusion date, plus the mandatory number of years the exclusion type specifies, unless the waiver date, or the reinstatement date come before. The LEIE does not contain data on which procedures in the Medicare Part B data the provider submitted fraudulent claims for, so we label all records submitted prior to the end of the exclusion period as fraudulent. The labeling technique we describe here is the same technique outlined in [17].

We refer to records labeled as fraudulent as the positive class, and records labeled as non-fraudulent as the negative class. The positive class is very sparse. We follow two different strategies for preparing datasets for XGBoost and CatBoost. In Table 2, we provide positive and negative class counts for the datasets we use as input to CatBoost and XGBoost, as well as the ratios of positive to negative instance counts. Please see Table 2 for class ratio data on the two datasets.

TABLE 2. IMBALANCE RATIOS OF DATASETS

| Dataset | Positive count | Negative count | Ratio, positive count to negative count |
|---|---|---|---|
| Aggregated | 3,691,146 | 1,409 | 0.00038 |
| Not aggregated | 37,224,032 | 31,411 | 0.00084 |

We use four categorical features. The sizes of the sets of possible values for these features is in Table 3.

TABLE 3. COUNTS OF POSSIBLE VALUES OF CATEGORICAL FEATURES

| Feature name | Value count |
|---|---|
| provider_type provider_type | 91 |
| nppes_provider_gender | 3 |
| nppes_provider_state | 61 |
| hcpcs_code | 6,022 |

This concludes our discussion of the nature of the data we work with. In the following three sections, we provide details on how we prepare data for our experiments.

## 3.2. Data for first experiment

We follow the technique outlined in [17] for preparing data for XGBoost and CatBoost in the first experiment. In this experiment, unlike the remaining experiments, we use the same encoding technique for both CatBoost and XGBoost. XGBoost cannot use categorical values in their original form, so we use two techniques to convert features of the Medicare Part B data to a numerical form XGBoost can consume. For a general survey on encoding techniques for categorical data, please see [16]. The first technique we use is one-hot encoding, to represent provider gender, and provider type. However, the procedure (HCPCS) code listed in the Medicare Part B data has thousands of possible values. Therefore, one-hot encoding the HCPCS code without employing a data compression technique, such as a sparse representation, would be prohibitively expensive in terms of memory consumption. However, we aim to make a comparison between XGBoost and CatBoost with the fewest optimizations to either learner possible to get the best sense of their relative performance. Hence, it is outside the scope of this study to gauge the impact of employing data compression techniques, and we do not use the HCPCS code directly as a feature in the first experiment. Instead, we aggregate the Medicare Part B data by NPI, year, gender and provider type, and add the minimum, median, maximum, mean, standard deviation, and sum for the features named: line_srvc_cnt, bene_unique_cnt, bene_day_srvc_cnt, average_submitted_chrg_amt, average_medicare_payment_amt. Including these summary statistics for each of these features adds extra information about the procedure the HCPCS code represents, and therefore serves as a proxy for it.

For the first experiment, because we one-hot encode some categorical features, and add summary statistics for numerical features, our dataset has 126 features. The dataset we prepare for XGBoost and CatBoost has a total of 3,692,555 instances. We convert all data to numerical form before using it as input to CatBoost and XGBoost.

The dataset we use is similar to the dataset Bauder et al. use in [17] where it serves as input to a GBDT implementation. In their work, the authors do not use provider state as a feature for their gradient boosted model. Therefore, we also do not include provider state as a feature in this experiment. Moreover, we use the same data as input to CatBoost and XGBoost in the interest of making a fair comparison.

Therefore, in this experiment, we do not take advantage of CatBoost's automatic handling of categorical features, or of its Ordered Target Statistic encoding of categorical features.

### 3.3. Data for second experiment

For the second experiment, we use the same data from the first experiment, but we do not aggregate it by NPI, year, gender, and provider type. In terms of Table 1 we use all features listed excluding HCPCS code. This means we add the NPPES provider state as a feature. For CatBoost, we do not need to take the pre-processing step to one-hot encode the features for provider type, provider gender or provider state. In this experiment we take advantage of CatBoost's automatic handling of categorical features. According to the CatBoost documentation, the configuration we use causes CatBoost to use Ordered Target Statistics encoding for the provider state, provider gender, and provider type categorical features [3]. Here we take the approach of using minimal enhancements to GBDT libraries we are comparing. Therefore, it is outside the scope of this study to do further experiments to optimize encoding techniques for XGBoost. Hence, for XGBoost we one-hot encode provider gender, provider type, and provider state. In the second experiment we do not use the HCPCS code feature for either CatBoost or XGBoost, therefore, we have no need to add any aggregate statistics to serve as a proxy for HCPCS code as we do in the first experiment. In the following section we discuss datasets we use in experiments to compare the effect of including the HCPCS code feature.

### 3.4. Data for remaining experiments

We perform three experiments with CatBoost that involve the same data we use in the second experiment, but we vary the presence of provider state and HCPCS code to assess the impact of these features on the task of Medicare fraud detection. Therefore, we use three distinct datasets. The data we use is described in Table 1, with the following modifications: in the first dataset we include the HCPCS code feature, in the second we include the provider state feature, and in the third we include both the HCPCS code and provider state features.

CatBoost automatically handles categorical features without our needing to pre-process data. Hence, we are able to easily use the HCPCS code, provider gender, provider type and provider state features of the Medicare Part B data as features for CatBoost. We use the default encoding technique that CatBoost uses in classification tasks, for categorical variables with a set of distinct values larger than the minimum size for one-hot encoding. Please see the section titled "Transforming categorical features to numerical features in classification" in [3] for a detailed discussion on this encoding technique. There is an opportunity for future research to investigate whether we can find a better value for the threshold value CatBoost uses to decide whether to use one-hot encoding, since the threshold is configurable. CatBoost can also use different encoding techniques, for high cardinality categorical variables. The encoding technique is determined by parameters the user specifies. Also, the encoding techniques available depend on the type of machine learning task — classification, regression, or multi-class classification — and the method of transforming categorical features the user specifies when setting CatBoost hyper-parameters. Further details of the various embedding techniques CatBoost uses are outside the scope of this study. Our goal is to establish a relative baseline comparison of the performance of XGBoost and CatBoost. Hence, we are only interested in using hyper-parameter settings for both algorithms as close to their default values as possible. Interested readers should see the documentation in [2].

### 3.5. Data Sampling

Previous studies generally indicate that the best technique for addressing class imbalance for the task of fraud detection in Medicare data is random undersampling [17]. Therefore, we use random undersampling as well. Since [17] reports performance of a GBDT algorithm with a 50:50 ratio, we use the same ratio here. It should be noted, however, that applying random undersampling to this class ratio reduces the size of datasets and risks data loss of the negative class. We repeat experiments ten times to mitigate that risk. Since we apply random undersampling, the size of the data we use for training CatBoost and XGBoost is different from the size of the data we use for testing them. From Table 2, one can surmise that, for the aggregated dataset, we fit CatBoost and XGBoost on a dataset with a size on the order of thousands instances. Similarly, for the dataset that is not aggregated, we fit XGBoost and CatBoost on a dataset with a size on the order of tens of thousands of instances. However, we do not apply random undersampling to the test datasets we use for evaluation. We use stratified 5-fold cross validation so that the test datasets have sizes on the order of 20 percent of the size of the entire dataset. This has an impact on the testing time versus the training time for XGBoost and CatBoost. For XGBoost, due to the size of our training data we see time to test a trained XGBoost model is always longer than the time it takes to train it in our experiments. However, for CatBoost due to the size of the data and model complexity, we find some instances where training time is longer than testing time, and some where that is not the case. This concludes our discussion on data we use in this study; now we move on to cover the algorithms we use.

### 3.6. Models

In order to make a fair baseline comparison, we decided to use hyper-parameters as close to default values as possible for both XGBoost and CatBoost. However, the hyper-parameter in XGBoost and CatBoost that controls the number of trees these GBDT libraries construct has meaningfully different default values. The value of this hyper-parameter has an impact on training time. For XGBoost, the default number of trees it uses in the GBDT ensemble is 100.

For CatBoost, this hyper-parameter has the default value of 1,000. We wish to know the minimum number of trees to use such that a further increase does not yield a statistically significant increase in performance in terms of AUC. Therefore, we conduct three rounds of experiments, one where both CatBoost and XGBoost use ensembles of 100 trees, one where both use 250 trees and one where both use 500 trees. We take the approach of looking for statistically significant improvements in performance in terms of AUC as we increase the number of trees, similar to the approach Khoshgoftaar et al. take in [22]. When we do not see a statistically significant increase in performance in terms of AUC with a commensurate increase in the number of trees, we hypothesize that conducting further experiments with larger numbers of trees will lead to overfitting. We report empirical evidence of overfitting in our results. In order to compare the running times of CatBoost and XGBoost we use the Python `time` library's `process_time` function to measure model fitting and test times. This function only counts the time the process that called it uses, and does not count time that other concurrent processes may take while preempting the processes running our experiments. The hyper-parameters we use for XGBoost are listed in Table 4. The hyper-parameters we use for CatBoost are listed in Table 5.

TABLE 4. XGBOOST HYPER-PARAMETERS

| Hyper-parameter | Value |
|---|---|
| tree_method | gpu_hist |
| objective | binary:logistic |
| learning_rate | 0.1 |
| max_depth | 6 |
| n_estimators | 100, 250 *or* 500 |

For XGBoost and CatBoost, we specify that the algorithms use GPU's because we found using GPU's speeds up the model training process. This makes completion of our experiments in a sensible length of time feasible. For XGBoost, we found that, in order for it to train in a reasonable amount of time, we needed to set the learning rate explicitly to 0.1 and the maximum depth of its constituent decision trees to 6. We set the XGBoost classifier's objective function to binary-logistic since the label in our dataset has two values. In order to control the number of trees XGBoost uses, we set a value for the n_estimators parameter. The hyper-parameter settings we use for CatBoost are those that instruct CatBoost to use GPU's, one that instructs CatBoost on which features are categorical, and one for controlling the number of trees it will construct.

We use the Python application programming interfaces (API's) for both XGBoost and CatBoost. The Python version we use is 3.7.3. We use one GPU to fit both CatBoost and XGBoost models. We use Python Scikit-learn [26] for stratified 5-fold cross validation and AUC calculation.

TABLE 5. CATBOOST HYPER-PARAMETERS

| Hyper-parameter | Value |
|---|---|
| cat_features | provider_type<br>nppes_provider_gender<br>nppes_provider_state |
| task_type | GPU |
| devices | 0 |
| iterations | 100, 250 *or* 500 |

Only in the final experiment do we use all three values listed for the cat_features hyper-parameter. Please see our discussion on data for experiments for details on which categorical features we include.

## 4. Results and Discussion

The tables here summarize the results of our experiments. We use abbreviations to label experiments. XGB-A denotes the experiments where we train XGBoost with aggregated Medicare Part B data. CB-A labels experiments where we use aggregated Medicare Part B data to train CatBoost. XGB-NA-S stands for experiments where we train XGBoost using data that is not aggregated, and we add features for the one-hot encoding of provider state. CB-NA-S indicates experiments where we train CatBoost with data that is not aggregated, and we add a feature for provider state that CatBoost encodes internally. CB-NA-H signifies experiments where we train CatBoost on the data that is not aggregated, and we add a feature for HCPCS code, which CatBoost encodes internally. Finally, CB-NA-HS denotes experiments where we train CatBoost with data that is not aggregated, and we add features for both the HCPCS code and provider state. We perform ten iterations of stratified 5-fold cross validation for XGBoost and CatBoost using the data we describe in section 3.1. For all experiments, we use random undersampling to balance positive and negative classes to a one-to-one ratio. The AUC values in Tables 6, 7, and 8 in this section are the mean of a total of 50 AUC values that we record for each of the five folds and ten iterations of each classifier. For each AUC value we report, we also report the mean training time and mean test time. Columns with the heading "Mean Train" in the Tables 6, 7, and 8 contain the mean training time for the model in the experiment. Training time is the time it takes for the model's fitting process to complete for the current training fold of 5-fold cross validation. Columns with the heading "Mean Test" in Tables 6, 7, and 8 hold the mean test times. Test time is the length of time for the model to classify all the data in the current test fold of 5-fold cross validation, plus the time it takes to compute performance in terms of AUC. For any mean value we report, we give the standard deviation of that same set of values to the right of the mean value in parentheses.

Interestingly, with 100 trees in the ensemble, XGBoost outperforms CatBoost with several datasets. However, we do not investigate the significance of this case because here we are using the default number of trees for XGBoost, 100. This is far below the default value of the number of trees

TABLE 6. CATBOOST AND XGBOOST AUC, TRAINING TIME AND TEST TIME FOR ENSEMBLES WITH 100 TREES

| Name | Mean AUC | Mean Train | Mean Test |
|---|---|---|---|
| XGB-A | 0.7595 (0.0140) | 0.4645 (0.1748) | 7.6346 (0.1816) |
| CB-A | 0.7629 (0.0116) | 1.1936 (0.0282) | 2.2365 (0.0364) |
| XGB-NA-S | 0.8409 (0.0063) | 0.3665 (0.0118) | 16.6041 (0.2253) |
| CB-NA-S | 0.8224 (0.0075) | 1.0392 (0.0255) | 5.4914 (0.1067) |
| CB-NA-H | 0.7636 (0.0094) | 1.0382 (0.0175) | 5.5751 (0.1112) |
| CB-NA-HS | 0.8160 (0.0076) | 1.0650 (0.0203) | 6.1805 (0.1603) |

for CatBoost, 1,000. Hence, training with 100 trees is far below the 1,000 trees that the CatBoost creators implicitly recommend as a starting point by setting it as a default value. With 100 trees, for our datasets, we find XGBoost's training time is about three times as fast as CatBoost's.

TABLE 7. CATBOOST AND XGBOOST AUC, TRAINING TIME AND TEST TIME FOR ENSEMBLES WITH 250 TREES

| Name | Mean AUC | Mean Train | Mean Test |
|---|---|---|---|
| XGB-A | 0.7650 (0.0132) | 1.0381 (0.1844) | 14.3508 (0.3704) |
| CB-A | 0.7705 (0.0128) | 2.5522 (0.0496) | 2.3687 (0.0437) |
| XGB-NA-S | 0.8616 (0.0063) | 0.7725 (0.0156) | 25.5449 (0.5446) |
| CB-NA-S | 0.9080 (0.0046) | 11.7339 (0.2190) | 13.3838 (0.5912) |
| CB-NA-H | 0.7725 (0.0085) | 11.7760 (0.2375) | 14.9020 (0.5553) |
| CB-NA-HS | 0.9115 (0.0042) | 12.2809 (0.2451) | 14.9474 (0.7282) |

When we increase the number of trees to 250, we see CatBoost yields higher AUC values than XGBoost, but CatBoost's training times are much longer. We find support for the conclusion that including summary statistics as a proxy for HCPCS codes is a viable technique since the mean AUC's of CB-A and CB-NA-H are about the same.

TABLE 8. CATBOOST AND XGBOOST AUC, TRAINING TIME AND TEST TIME FOR ENSEMBLES WITH 500 TREES

| Name | Mean AUC | Mean Train | Mean Test |
|---|---|---|---|
| XGB-A | 0.7636 (0.0121) | 1.9024 (0.1922) | 25.9271 (0.6178) |
| CB-A | 0.7744 (0.0115) | 20.3925 (1.1573) | 2.6078 (0.0477) |
| XGB-NA-S | 0.8638 (0.0060) | 1.5486 (0.0233) | 45.5279 (1.1089) |
| CB-NA-S | 0.9080 (0.0043) | 36.8671 (0.7333) | 18.2102 (1.4366) |
| CB-NA-H | 0.7739 (0.0092) | 37.1622 (0.6364) | 22.5459 (1.8125) |
| CB-NA-HS | 0.9107 (0.0038) | 38.6667 (0.4067) | 23.1227 (2.1084) |

In terms of training with 500 trees, we see more of the same things we see with 100 and 250 trees. However, unlike the differences in AUC values between the experiments with 100 and 250 trees, we see CatBoost is not necessarily yielding higher AUC values. In fact, for the CB-NA-HS experiment, with 250 trees CatBoost yields an AUC value of 0.9115, whereas with 500 trees CatBoost yields an AUC value of 0.9107. This is an indication that CatBoost is overfitting when we use 500 trees. We do not find it necessary to conduct an experiment where we use CatBoost's default setting of 1,000 trees, since we find evidence that CatBoost overfits with 500 trees.

## 4.1. Statistical Analysis

Here, we examine the impact of the number of trees on the performance of XGBoost and CatBoost, as well as the impact of the choice of GBDT implementation. Furthermore, we investigate the impact of including more categorical features with the input to CatBoost. To investigate the impact of the number of trees we perform $z$-tests to compare the performance of the same GBDT implementation with the same dataset, but with different numbers of trees. We find that performance does not improve when we use more than 250 trees in the ensemble. The tables that contain the data to support this finding are Tables 9 and 10. Since we find 250 to be the optimum number of trees, we conduct another series of $z$-tests to compare the performance of XGBoost versus CatBoost with the same number of trees (250) on the same datasets. The results of these $z$-tests are in Table 11. Finally, we report the results of $z$-tests for significant differences in terms of AUC when we add the NPPES provider state and HCPCS codes categorical features to CatBoost in table 12.

TABLE 9. $z$-TESTS COMPARING ENSEMBLES OF 100 AND 250 TREES

| Experiment | $p$-value | Decision |
|---|---|---|
| XGB-A | 0.0461 | do not reject |
| CB-A | 0.0019 | reject |
| XGB-NA-S | 0.0000 | reject |
| CB-NA-H | 0.0000 | reject |
| CB-NA-S | 0.0000 | reject |
| CB-NA-HS | 0.0000 | reject |

The majority of results in Table 9 show a significant difference in performance in terms of AUC for classifiers with 100-tree ensembles versus classifiers with 250-tree ensembles. There is one exception for the XGB-A experiment where the difference in the number of trees in the ensemble is not significant at a 99% confidence level. However, the results in Table 10 show that there is no significant difference in performance in terms of AUC when we compare ensembles of 250 and 500 trees.

TABLE 10. $z$-TESTS COMPARING ENSEMBLES OF 250 AND 500 TREES

| Experiment | $p$-value | Decision |
|---|---|---|
| XGB-A | 0.5907 | do not reject |
| CB-A | 0.1085 | do not reject |
| XGB-NA-S | 0.0764 | do not reject |
| CB-NA-H | 0.4100 | do not reject |
| CB-NA-S | 0.9658 | do not reject |
| CB-NA-HS | 0.2992 | do not reject |

The results of the series of $z$-tests in Tables 9 and 10 imply we would not see a significant difference in performance in terms of AUC if we add any more trees

to the ensembles. Therefore, we perform a series of $z$-tests to investigate whether the choice of classifier, XGBoost or CatBoost, is significant. The results of the $z$-tests are in Table 11.

TABLE 11. $z$-TESTS COMPARING CATBOOST AND XGBOOST

| Experiment | $p$-value | Decision |
|---|---|---|
| XGB-A vs CB-A | 0.0344 | do not reject |
| XGB-NA-S vs. CB-NA-S | 0.0000 | reject |

The first result in Table 11 shows that, at a 99% confidence level, the choice of classifier is not significant for the problem of Medicare fraud detection when we train these learners with the aggregated dataset. However, when we use data that is not aggregated, and include provider state as a categorical feature, the second $z$-test result implies that CatBoost outperforms XGBoost in a statistically significant manner. The difference in performance occurs under specific conditions. We let CatBoost use its internal Ordered Target Statistic encoding of the provider state feature, but we use one-hot encoding for XGBoost, since experimenting with various encoding techniques for XGBoost is outside the scope of this study. The final series of $z$-tests we perform is to investigate the impact of varying categorical features for CatBoost.

TABLE 12. $z$-TESTS COMPARING CATBOOST WITH DIFFERENT NUMBERS OF CATEGORICAL FEATURES

| Experiment | $p$-value | Decision |
|---|---|---|
| CB-NA-H vs. CB-NA-S | 0.0000 | reject |
| CB-NA-H vs. CB-NA-HS | 0.0000 | reject |
| CB-NA-S vs. CB-NA-HS | 0.0001 | reject |

Table 12 shows that at a 99% confidence level CatBoost yields significantly different performance when we use either the feature for HCPCS code or the feature for provider state. Moreover, we see a significant difference when we include both provider state and HCPCS code versus including only one of those features. The results we report lead us to several conclusions.

## 5. Conclusion

Our results reflect XGBoost and CatBoost's ability to classify an extremely imbalanced dataset of Medicare claims data, in the task of Medicare fraud detection. We show that CatBoost's built-in handling of categorical features is effective, even for high-cardinality features such as the HCPCS code in Medicare claims data. Moreover, we find a statistically significant increase in CatBoost's performance as we add features for HCPCS code and provider state. Adding these features for CatBoost requires little effort on our part since CatBoost handles encoding categorical features automatically.

As far as performance in terms of AUC is concerned, we find that with the aggregated Medicare Part B data, augmented with summary statistics, the difference in the performance of CatBoost and XGBoost is not statistically significant. However, with data that is not aggregated we find CatBoost yields a mean AUC of 0.9080, whereas XGBoost yields a mean AUC of 0.8616. This difference is statistically significant for both GBDT implementations when we configure them to use 250 trees. On the subject of running time consumption, we find the average training times for XGBoost to be lower than those of CatBoost.

Another aspect of our results is the number of trees in the ensembles that CatBoost and XGBoost use. We find that, for the task of Medicare fraud detection, 250 trees is the optimal number to use in their respective ensembles since we do not see a statistically significant improvement in performance in terms of AUC when we raise the number of trees to 500.

We do not intend to imply that our results give the definitive conclusion that CatBoost is superior to XGBoost in the task of Medicare fraud detection. In future research we intend to apply hyper-parameter tuning techniques to determine if statistically significant differences in performance persist, as well as the costs in terms of resource consumption for different hyper-parameter configurations.

## Acknowledgment

The authors would like to express their gratitude to the reviewers at the Data Mining and Machine Learning Laboratory of Florida Atlantic University for the help in preparing this study.

## References

[1] Medicare provider utilization and payment data: Physician and other supplier, 2019.

[2] Python package training parameters, 2020. [Online]. Available: https://catboost.ai/docs/concepts/python-reference_parameters-list.html#python-reference_parameters-list.

[3] Transforming categorical features to numerical features, 2020. [Online]. Available: https://catboost.ai/docs/concepts/algorithm-main-stages_cat-to-numberic.html#algorithm-main-stages_cat-to-numberic.

[4] S. J. Bagdoyan. Testimony before the subcommittee on oversight, committee on ways and means, house of representatives, 2018.

[5] Richard A. Bauder and Taghi M. Khoshgoftaar. 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)*, pages 11–19, July 2016.

[6] Richard A. Bauder and Taghi M. Khoshgoftaar. A probabilistic programming approach for outlier detection in healthcare claims. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 347–354, Dec 2016.

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

[8] L. K. Branting, F. Reeder, J. Gold, and T. Champney. Graph analytics for healthcare fraud risk estimation. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 845–851, Aug 2016.

[9] Centers for Medicare & Medicaid Services. Get started with medicare. [Online]. Available: https://www.medicare.gov/sign-up-change-plans/get-started-with-medicare.

[10] Centers For Medicare & Medicaid Services. Trustees report & trust funds, 2018.

[11] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016.

[12] CMS Office of Enterprise Data and Analytics. Medicare fee-for-service provider utilization & payment data physician and other supplier, 2017.

[13] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367 – 378, 2002. Nonlinear Methods and Data Mining.

[14] John T. Hancock and Taghi M. Khoshgoftaar. Catboost for big data: an interdisciplinary review. *Journal of Big Data*, 2020.

[15] John T. Hancock and Taghi M. Khoshgoftaar. Medicare fraud detection using catboost. In *2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI)*, pages 97–103. IEEE Computer Society, 2020.

[16] John T. Hancock and Taghi M. Khoshgoftaar. Survey on categorical data for neural networks. *Journal of Big Data*, 7(1):1 – 41, 2020.

[17] Matthew Herland, Richard A. Bauder, and Taghi M. Khoshgoftaar. The effects of class rarity on the evaluation of supervised healthcare fraud detection models. *Journal of Big Data*, 6(1):1, 2019.

[18] Matthew Herland, Richard A. Bauder, and Taghi M.Khoshgoftaar. Approaches for identifying u.s. medicare fraud in provider claims data. *Health Care Management Science*, (1):2, 2020.

[19] Justin M. Johnson and Taghi M. Khoshgoftaar. Deep learning and thresholding with class-imbalanced big data. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 755–762, Dec 2019.

[20] Justin M. Johnson and Taghi M. Khoshgoftaar. Medicare fraud detection using neural networks. *Journal of Big Data*, 6(1):1 – 35, 2019.

[21] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017.

[22] Taghi M. Khoshgoftaar, Moiz Golawala, and Jason Van Hulse. An empirical study of learning from imbalanced data using random forest. *19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007), Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on*, 2:310 – 317, 2007.

[23] Joffrey L. Leevy, Taghi M. Khoshgoftaar, Richard A. Bauder, and NaeemSeliya. Investigating the relationship between time and predictive model maintenance. *Journal of Big Data*, (1), 2020.

[24] LEIE. Office of inspector general leie downloadable databases., 2017. [Online]. Available: https://oig.hhs.gov/exclusions/index.asp.

[25] Pascal Challenge 2008. Epsilon. [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

[26] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[27] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6638–6648. Curran Associates, Inc., 2018.

[28] R. Punmiya and S. Choe. Energy theft detection using gradient boosting theft detector with feature engineering-based preprocessing. *IEEE Transactions on Smart Grid*, 10(2):2326–2329, 2019.

[29] Gabriel Spadon, Andre C. P. L. F. de Carvalho, Jose F. Rodrigues-Jr, and Luiz G. A. Alves. Reconstructing commuters network using machine learning and urban indicators. *Scientific Reports*, 9(1):N.PAG, 2019.