The Effects of Random Undersampling for Big Data Medicare Fraud Detection

John Hancock, Taghi M. Khoshgoftaar, and Justin M. Johnson
College of Engineering and Computer Science
Florida Atlantic University
Boca Raton, Florida 33431
jhancoc4@fau.edu, khoshgof@fau.edu, jjohn273@fau.edu

***Abstract*—We show it is possible to obtain better classification performance for experiments involving highly imbalanced Big Data with the application of data sampling techniques. We apply Random Undersampling to a publicly available Medicare insurance claims dataset of about 175 million records. We find that Random Undersampling significantly improves the classification performance of Extremely Randomized Trees and XGBoost learners in a Medicare Big Data fraud classification task. We employ Random Undersampling to evaluate performance at multiple minority:majority class ratios. According to the outcome of a Tukey's Honestly Significant Difference test, we find Random Undersampling to the 1:9 or 1:27 class ratios yields the best performance, providing Area Under the Receiver Operating Characteristic scores of over 0.97. Models built with undersampled Big Data require significantly less time to train. Our contribution is to prove the effectiveness of Random Undersampling in classifying Medicare Big Data. Our review of related work shows we are the first to apply Random Undersampling to data on this scale in order to prove one can obtain better performance. To the best of our knowledge, we are the first to perform experiments with the latest Medicare Part D data, made available in 2021.**

***Keywords*—Extremely Randomized Trees, XGBoost, Class Imbalance, Big Data, Undersampling**

I. Introduction

Imbalanced Big Data presents a unique set of challenges for Machine Learning. A significant disparity between the sizes of the classes of data can skew the performance of Machine Learning algorithms. For example, models may overfit to the majority class when one does not employ techniques to address class imbalance. For an in-depth review of studies that address class imbalance, please see [1] and [2]. Random Undersampling (RUS) is an effective technique for addressing class imbalance in Big Data [3] . RUS is the arbitrary removal of some number of instances of the majority class from the training data to force the class ratio of the training data to be at a certain level.

We investigate the impact of RUS on the Area under the Receiver Operating Characteristic Curve (AUC) scores of Extremely Randomized Trees (ET) [4], and XGBoost [5], for classifying highly imbalanced Big Data. The results of our experiments reveal that ET and XGBoost yield better AUC scores when they are trained on data that is preprocessed with RUS than when they are trained on the entire dataset. That is a noteworthy fact since, like many Machine Learning algorithms, less training time is required to train ET and XGBoost with undersampled data than with the full dataset. When working with Big Data, model training times are long enough to push the limits of practicality. Therefore, RUS is an attractive option. The contribution in this work is to illustrate how we apply RUS to Big Data, and in some cases, obtain better classification performance in less time.

We report results for classifying Medicare insurance claims data. To the best of our knowledge, the data are the largest publicly available Medicare claims data. This is because it contains data for all years released by the Centers for Medicare and Medicaid Services (CMS) so far. Medicare is the United States' public health insurance program. The majority of its beneficiaries are aged 65 and older. To facilitate research, CMS periodically releases "Medicare Provider Utilization and Payment Data: Part D Prescriber" data [6]. Hereafter, we will refer to this data as the "Part D" data. At the time of this writing, CMS has released data from 2013 through 2019. The Part D data pertains to medications healthcare providers prescribe for their patients. Each record of the Part D data describes a provider's billing activity with respect to a particular medication and year. The Part D data now contains about 175 million records.

CMS has made seven years worth of data publicly available; therefore, the data is growing at the rate of tens of millions of records per year. The fact that each record represents all of the claims a single provider makes for a particular medication and year gives a sense of the volume of insurance claims CMS receives. The volume of claims makes CMS a target for fraud. In a 2019 report, The Department of Justice states it recovered about $3 billion from Medicare Fraud prosecutions [7]. On the other hand, for 2019, CMS estimates it made $100 billion in improper payments. It is possible that the United States Government could recover more money from fraudsters out of the remaining $93 billion in improper payments. Since the volume of claims sent to CMS is large, there is a great opportunity to leverage Machine Learning for automated fraud detection. Therefore, this study is also a contribution in the area of automated Medicare fraud detection that has the potential to provide significant savings.

We study the ability of two Machine Learning algorithms to detect fraudulent provider activity in the Part D data. The

List of Excluded Individuals and Entities (LEIE), [8], contains data on fraudulent providers. We use the LEIE to label the Part D data. This enables us to approach fraud detection as a supervised Machine Learning task. The remainder of this work is organized as follows: we present our findings from a literature review on related work, then we discuss details on XGBoost and ET. Following that, we explain our process for creating the dataset used in our experiments. Next, there is a section on our experimental methodology, after which we present our results. Finally, we report our conclusions.

II. Related Work

To prepare for our study we searched for work where the authors investigate the impact of RUS on the performance of Machine Learning algorithms to classify Big Data. Our review of related work reveals that we are the first to report findings on a systematic study of the impact of RUS with data on the scale of about 175 million instances. We emphasize our use of ET, since our results show improvement in the performance of ET when undersampling is applied. The improvement in running time that RUS affords is another important fact omitted in some related work. The details we provide on related work show our work is a novel contribution.

Fernandez *et al.* [9] published a study that focuses on the practical application of the Apache Spark [10] and Hadoop [11] distributed processing frameworks. The dataset they use has 12 million instances. Their results show that applying RUS yields significantly better performance. However, the improvement in performance diminishes as they increase the level of parallelism that the Spark and Hadoop frameworks provide. The authors use RUS, but the application of RUS is merely tangential to their study. They apply RUS to balance the data they work with to a 1:1 minority to majority class ratio. We perform experiments where we employ RUS to vary the imbalance ratios to 5 different levels for comparison with the case where no sampling is applied.

Sleeman *et al.* [12] investigate classification of imbalanced datasets with the Apache Spark framework. The largest dataset they work with contains 3 million instances. The Machine Learning tasks for their study involve multi-class classification. They report the performance of Random Forest and Naive Bayes classifiers trained on data preprocessed with multiple sampling techniques, including RUS. We do not see a clear best-performing data sampling technique in the results they report. Their aim seems primarily to present a framework for multi-class classification for researchers to use in future work. We did not find an indication that they executed multiple iterations of cross validation to mitigate the risk of randomly selecting instances from the majority class that could yield unusually high or low performance. They compare RUS to other sampling techniques; however, we see no report of class ratios after sampling techniques are applied. Hence, Sleemen *et al.*'s study is another instance of a work where the authors do not use a dataset that is on the scale of our data, and they do not investigate the impact of RUS at varying levels of imbalance.

Del Río *et al.* also research the application of the Hadoop distributed framework to classify Big Data with sampling techniques [13]. The data used in their study has under six million instances. Their findings are similar to those of Fernandez *et al.* Del Río *et al.* find that RUS improves classification performance, but that the positive effect of RUS degrades as they increase the amount of parallelism in their distributed application. Also, though they report the application of RUS, we do not find that they mention varying class ratios of their data when applying sampling techniques. Therefore, our work serves to accomplish something not addressed in Del Río *et al.*'s study, since we investigate the impact of varying levels of RUS on classification scores.

While we were unable to find studies involving ET, RUS and Big Data, we were able to find a study involving XGBoost, RUS and Big Data. Hancock and Khoshgoftaar investigated the performance of several classifiers, including XGBoost [14]. Their finding is that their best classification results are obtained with classifiers trained on data with RUS applied to make the class ratio 1:1. The results they report are from Medicare Part D datasets that are now out-of-date, since the dataset contains data from 2013 through 2017. The dataset we use here has data from 2013 through 2019. The dataset we work with is larger, having about 175 million instances. Hancock and Khoshgoftaar's dataset has approximately 122 million instances. While they report on the impact of varying undersampling ratios, they do not perform any experiments with the ET classifier. We find ET outperforms XGBoost when RUS is applied. That exposes a gap in Hancock and Khoshgoftaar's work, since ET was available years before they published their study.

Johnson and Khoshgoftaar investigate the impact of RUS on the classification performance of Neural Networks [15]. Their study covers the application of RUS. Their results show RUS performance degrades when the majority class is undersampled to make the minority class to be more than 20% of the training data. They use one-hot encoding to encode categorical features. This encoding technique does not scale to larger datasets since it increases the size of the dataset. One-hot encoding increases the number of features in the dataset in proportion to the number of possible values of the encoded feature. For more detail on encoding techniques for Neural Networks, please see [16]. We use CatBoost encoding to encode categorical features [17]. CatBoost encoding is more amenable to working with Big Data since it does not increase the number of features, but encodes categorical features with a decimal value instead. Furthermore, Johnson and Khoshgoftaar are prevented from experimenting with high-cardinality categorical features with thousands of possible values, since one-hot encoding these values would add too many features to the dataset. Thanks to CatBoost encoding, we are able to work with categorical features that have thousands of possible values. Johnson and Khoshgoftaar report using a dataset with less than five million instances. The data we work with is much larger. Since our dataset is larger, and we wish to encode the drug name feature of the Part D data that has thousands of possible values, the one-hot encoding techniques Johnson and Khoshgoftaar employ are not practical for the data we use here.

In a later work Johnson and Khoshgoftaar perform similar experiments to assess the impact of RUS on the classification performance of Neural Networks [18]. In this study they perform experiments with three datasets. They report results that show applying RUS to make the minority class one percent of the training data yields the best results. As the size of the minority class is made larger in the training data, performance degrades. The datasets they use in this study also have less than five million instances each. Our study differs from this work by Johnson and Khoshgoftaar for the same reasons as their previous work. We report findings for a larger dataset, and we use an encoding technique that supports working with larger datasets.

Related work where RUS is applied to Big Data does not cover many things we cover here. We found several studies that cover the application of RUS to big data. There are no studies that show the improvement of classification results ET achieves when RUS is applied. Moreover, to the best of our knowledge, we are the first to combine CatBoost encoding of categorical features with the ET classifier. Clearly, our study fills some gaps in the literature.

III. Algorithms

We use two classifiers in this study: XGBoost and ET. XGBoost is an implementation of the Gradient Boosted Decision Trees (GBDT) algorithm discovered by Friedman [19]. Chen and Guestrin describe several enhancements they add to GBDT in their seminal work on XGBoost [5]. One improvement they make to GDBT is a modification to its objective function used during ensemble training. Another contribution Chen and Guestrin add to XGBoost is the Weighted Quantile Sketch algorithm. This algorithm provides an approximate method for finding the optimal value for Decision Tree splits during model training. Decision Tree splits are the values in the internal nodes of a Decision Tree that dictate the path through the Decision Tree to the leaf node. The other major improvement in XGBoost, also related to Decision tree splits is so called "Sparsity-aware split finding". As its name implies, Sparsity-aware split finding is a technique for determining the value of splits, which is effective for sparse data.

Geurts *et al.* introduce ET in a 2005 Paper [4]. ET is a variant on Random Forest [20] [21]. Random Forest rests on the premise that the probability of a correct estimate from the majority of a collection of Decision Trees grows as the size of the collection of Decision trees grows. One feature of Random Forest is that it uses a random sample, with replacement, of attributes that it will use for a split as Decision Trees are built. ET introduces more randomness into the Decision Tree growing process by also randomly selecting values for splits. Perhaps surprisingly, this amount of randomness in the model fitting process still manages to yield strong performance, as our results show.

IV. Data Description and Preparation

In the interest of reproducible work, we document the procedure for building the dataset we use for experiments. CMS makes the Part D data available for download on its web site. The latest Part D data became available in 2021. To the best of our knowledge, we are the first to use it in a study. Table I contains descriptions from CMS's data dictionary [22], which we have augmented with information about the categorical features. Each record in the Part D data contains information about how one provider prescribes one drug for one year. There are a total of 22 attributes in the Part D data from CMS. We found that nine of them are useful for Machine Learning experiments. We discard all seven of the features that have names starting with GE65 due to missing values. In addition we discard six additional features to avoid the possibility that the model memorizes the identity of fraudulent providers, either through their National Provider ID (NPI) or their location data. These features are: Prscrbr_NPI, Prscrbr_Last_Org_Name, Prscrbr_First_Name, Prscrbr_City, Prscrbr_State_Abrvtn, and Prscrbr_State_FIPS. Therefore, we discard a total of 13 features, leaving the nine features documented in Table I.

We discard the NPI from the final dataset; however, it is useful for labeling our data. We label our data with a supplementary source, which is the LEIE from the United States Office of the Inspector General (OIG). The OIG publishes the LEIE monthly. It contains data on providers that are prohibited from billing Medicare due to fraud convictions. We filter the list for specific exclusion rules with logic described in [23].

TABLE I
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, 2,272 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

Providers are removed from the LEIE once their exclusion periods are over. Therefore, we use the Internet Archive Tool[1] to label older records. We label all records belonging to a provider that is on the LEIE as fraudulent, from the earliest record to the last record before the end of the exclusion period. Since the LEIE has the NPIs of providers, we can join the Part D data to it, to form a labeled dataset. The labeled dataset contains 173,677,665 instances.

The records labeled as fraudulent form a small part of the total data. The ratio of records labeled fraudulent to not fraudulent is 1:256. Therefore, the data is imbalanced. We apply RUS to address class imbalance. Oversampling is not a practical alternative for the Part D data because it is already a large dataset, and increasing the size of it would make it more difficult to deal with. Undersampling is more attractive since it reduces the size of the dataset, which lowers model training time and memory consumption.

V. Methodology

In further efforts to make our work reproducible, we use publicly available, widely used, open source software to conduct our experiments. We run all experiments as Python programs [24]. XGBoost is an actively maintained library available for download[2]. The ET implementation we use is part of the Scikit-learn Python Machine Learning library [25], which is also freely available.

We use CatBoost encoding to encode categorical features listed in Table I. As mentioned in the related works section, CatBoost encoding is a good choice for encoding categorical features in Big Data since it does not expand the size of the dataset like one-hot encoding, and it supports encoding high-cardinality categorical features. For details on how CatBoost encoding is calculated, please see [17]. For a survey of applications of CatBoost, please see [26]. The Python Category-encoders library provides an object that one may use to encode categorical features in a dataset. Similar to a classifier, one must fit the CatBoost encoder to the dataset before using it to encode categorical features. It is important to fit the encoder on the training data only, otherwise, information about the dependent variable will leak into the encoded features of the test data. Put another way, just as a model should not be fit to the test data, the encoder also must not be fit to the test data.

We also perform ten iterations of five-fold cross validation in every experiment. Therefore, we fit a new CatBoost encoder to the training part of each fold. Then, for the experiments where we apply RUS, we also apply it to the training data only. We apply RUS to make the minority to majority class ratios 1:1, 1:3, 1:9, 1:27, or 1:81. We start with the 1:1 ratio since several authors of related work report using it. Then, we repeatedly multiply the size of the majority class by three.

After encoding and undersampling the training data, we then fit either XGBoost or ET to the training data, and record the AUC score the trained model yields for the test data. The reason for performing ten iterations of five-fold cross validation is to mitigate the risk of data loss due to the random selection of instances from the majority class.

In preliminary experiments, we found XGBoost yields better performance with maximum tree depth set to 24, so we set maximum tree depth for XGBoost to 24 for all experiments. XGBoost also has a "tree_method" parameter that one can use to specify that XGBoost should run on Graphics Processing Unit (GPU) hardware. For both ET and XGBoost, we set and record a unique seed value for the random number generators that they use. This helps to ensure we can reproduce our results. We do not modify any other hyperparameter settings for ET. The important hardware specifications for the system we run experiments on are as follows: the system has an Intel Xeon CPU with 16 cores, 256 GB RAM, and an Nvidia V100 GPU.

To summarize our methodology, we report the total number of experiments performed. We have two classifiers, and six levels of RUS, since we perform experiments where we do not apply RUS as well as with RUS to alter the class ratios. We perform ten iterations of five-fold cross validation. Therefore we conduct $2 \times 6 \times 10 \times 5 = 600$ experiments.

VI. Results

We report AUC scores that are the outcomes of our experiments in Table II. These are the mean AUC scores XGBoost and ET yield for ten iterations of five-fold cross validation. We see the highest mean AUC score for ET in Table II is associated with a class ratio of 1:1. On the other hand, for XGBoost we see the highest mean AUC score is associated with a class ratio of 1:27.

Next, in Tables IV and V, we present the statistical analysis of the experimental outcomes recorded in Table II. The analysis is in the form of a two-factor analysis of variance (ANOVA) test, [27], where classifier and sampling ratio are both factors, followed by *post hoc* Tukey's Honestly Significant Difference (HSD) Tests, [28], to rank the factors by their effect on AUC scores.

TABLE II
Part D Mean and Standard Deviation of AUC with varying levels of RUS (10 iterations of 5-fold cross-validation)

Class Ratio	ET	XGB-24
1:1	0.97625 (0.00033)	0.95292 (0.00059)
1:3	0.97586 (0.00038)	0.96809 (0.00045)
1:9	0.97464 (0.00038)	0.97256 (0.00037)
1:27	0.97244 (0.00037)	0.97363 (0.00037)
1:81	0.96966 (0.00035)	0.97348 (0.00039)
Original (1:256)	0.96746 (0.00038)	0.97273 (0.00042)

Standard deviations are below AUC scores in parenthesis. ET is the Extremely Randomized Trees classifier. XGB-24 is the XGBoost classifier with maximum tree depth set to 24.

[1]http://archive.org/web
[2]https://pypi.org/project/xgboost/

Here we do a two-factor analysis of the performance of XGBoost and ET with RUS as a factor. Table III has the results of the ANOVA test with RUS and classifier (CLF) as factors. Since the Pr(>F), or p-values for both factors in the ANOVA test are practically 0, the result of the ANOVA test shows that both classifier and RUS have a significant impact on performance. Tables IV and V have the results of the HSD tests to group results by RUS level and classifier, respectively. The groups reported in the HSD test results are in alphabetical order by their association with higher AUC scores. Therefore, group 'a' is associated with the highest AUC scores, group 'b' is associated with the second highest AUC scores, and so on.

TABLE III
ANOVA FOR RUS AND CLF AS FACTORS OF PERFORMANCE IN TERMS OF AUC

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
RUS	5	0.00539	0.00108	44.90	*
CLF	1	0.00218	0.00218	90.99	*
Residuals	593	0.01423	0.00002		

* indicates the value is less than 1×10^{-15}.

In Table IV we see the 1:9 and 1:27 RUS levels yield the best performance. Next, we see 1:3 and 1:81 associated with the HSD group 'ab'. The group is designated 'ab' because AUC scores in this group overlap with those in group 'a' and group 'b'. HSD test results show that applying RUS to make the class ratio 1:1 yields the worst performance, and that not applying RUS yields the second worst performance. This is because undersampling to the 1:1 class ratio is too much of a data reduction, and we are discarding too much information about the majority class. In other words reducing to the 1:1 ratio leads to underfitting. On the other hand, training with the full dataset leads to overfitting to the majority class. Results in Table IV are averaged across both ET and XGB-24.

TABLE IV
HSD TEST GROUPINGS OF AUC FOR THE RUS FACTOR

Group a consists of: 1:9, 1:27
Group ab consists of: 1:3, 1:81
Group b consists of: Original (1:256)
Group c consists of: 1:1

Table V has the HSD test results for the classifier factor. We see, across all levels of RUS, ET yields the best performance. Therefore, we find it important to point out that in terms of running time, due to GPU support, the experiments involving XGBoost and the larger datasets ran much faster than the experiments involving ET. Therefore, we find a trade-off between performance and running time. Results in Table V are averaged across all levels the RUS factor.

TABLE V
HSD TEST GROUPINGS AFTER ANOVA OF AUC FOR THE CLF FACTOR

Group a consists of: ET
Group b consists of: XGB-24

We also noticed a significant difference in the time required to fit models with undersampled data versus the original data. XGBoost supports GPU hardware. RUS provides a definite speedup to training time for XGBoost. On the other hand, ET runs on CPUs, and the effect of RUS on the running time of ET is far more significant. A related result of applying RUS to data is decreased memory consumption; therefore, RUS can afford researchers an opportunity to run experiments on systems that will not support working with the entire dataset.

VII. CONCLUSIONS

Our results show that RUS has a significant, positive impact on the performance of ET and XGBoost. That is noteworthy since model training times are reduced when RUS is applied. Moreover, we show XGBoost and ET yield better classification performance when RUS is applied. That is an ideal result, since we obtain better performance with a smaller investment of computing time. Since we do ten iterations of five-fold cross validation, we mitigate the risk that RUS discards important information about the majority class. Over all combinations of classifier and RUS level, we see ET yields the best performance with RUS applied to make class ratios 1:9 or 1:27. To the best of our knowledge, we are the first to conduct experiments to determine the impact of RUS on a experiments with a dataset on the scale of about 175 million instances. Moreover, we are the first to use ET with CatBoost encoding in such a study. Since we see the positive impact RUS has on classification results of Part D data, future work should include similar experiments with more classifiers, and other Big Data datasets. We expect to find results similar to those we report here; that we can obtain better classification results, in less time, when using RUS to address class imbalance in Big Data.

REFERENCES

[1] J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder, and N. Seliya, "A survey on addressing high-class imbalance in big data," *Journal of Big Data*, vol. 5, no. 1, pp. 1–30, 2018.

[2] J. Van Hulse, T. M. Khoshgoftaar, and A. Napolitano, "Experimental perspectives on learning from imbalanced data," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 935–942.

[3] M. Herland, R. A. Bauder, and T. M. Khoshgoftaar, "The effects of class rarity on the evaluation of supervised healthcare fraud detection models," *Journal of Big Data*, vol. 6, no. 1, pp. 1–33, 2019.

[4] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

[5] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016. DOI: 10.1145/2939672.2939785.

[6] The Centers for Medicare and Medicaid Services, *Medicare part d prescribers - by provider and drug*, 2021.

[7] Civil Division, U.S. Department of Justice, *Fraud statistics, overview*, 2020.

[8] LEIE. (2022). Office of inspector general leie downloadable databases. [Online]. Available: https://oig.hhs.gov/exclusions/index.asp.

[9] A. Fernández, S. del Río, N. V. Chawla, and F. Herrera, "An insight into imbalanced big data classification: Outcomes and challenges," *Complex & Intelligent Systems*, vol. 3, no. 2, pp. 105–120, 2017.

[10] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, *et al.*, "Apache spark: A unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[11] Apache Software Foundation, *Hadoop*, version 0.20.2, Feb. 19, 2010.

[12] W. C. Sleeman IV and B. Krawczyk, "Multi-class imbalanced big data classification on spark," *Knowledge-Based Systems*, vol. 212, p. 106 598, 2021.

[13] S. Del Río, V. López, J. M. Benítez, and F. Herrera, "On the use of mapreduce for imbalanced big data using random forest," *Information Sciences*, vol. 285, pp. 112–137, 2014.

[14] J. T. Hancock and T. M. Khoshgoftaar, "Gradient boosted decision tree algorithms for medicare fraud detection," *SN Computer Science*, vol. 2, no. 4, pp. 1–12, 2021.

[15] J. M. Johnson and T. M. Khoshgoftaar, "Medicare fraud detection using neural networks," *Journal of Big Data*, vol. 6, no. 1, pp. 1–35, 2019.

[16] J. T. Hancock and T. M. Khoshgoftaar, "Survey on categorical data for neural networks," *Journal of Big Data*, vol. 7, no. 1, pp. 1–41, 2020.

[17] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: Unbiased boosting with categorical features," *Advances in neural information processing systems*, vol. 31, 2018.

[18] J. M. Johnson and T. M. Khoshgoftaar, "The effects of data sampling with deep learning and highly imbalanced big data," *Information Systems Frontiers*, vol. 22, no. 5, pp. 1113–1131, 2020.

[19] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[20] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[21] T. M. Khoshgoftaar, M. Golawala, and J. Van Hulse, "An empirical study of learning from imbalanced data using random forest," in *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, IEEE, vol. 2, 2007, pp. 310–317.

[22] The Centers for Medicare and Medicaid Services, *Medicare part d prescribers - by provider and drug data dictionary*, 2021.

[23] M. Herland, T. M. Khoshgoftaar, and R. A. Bauder, "Big data fraud detection using multiple medicare data sources," *Journal of Big Data*, vol. 5, no. 1, pp. 1–21, 2018.

[24] G. Van Rossum and F. L. Drake, *Python/c api manual-python 3*, 2009.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[26] J. T. Hancock and T. M. Khoshgoftaar, "Catboost for big data: An interdisciplinary review," *Journal of big data*, vol. 7, no. 1, pp. 1–45, 2020.

[27] G. R. Iversen and H. Norpoth, *Analysis of variance*, 1. Newbury Park: Sage, 1987.

[28] J. W. Tukey, "Comparing individual means in the analysis of variance," *Biometrics*, pp. 99–114, 1949.