Comparative Analysis of Classifiers on Select Imbalanced Datasets

Faris B. Mismar, Rebal S. Jurdi, Nibal Salha, and Megha Parhi

Abstract

A number of supervised learning methods have been introduced in the last thirty years. While there have been several studies discussing the performance of supervised learning algorithms, we focus our efforts on evaluating the large-scale performance of five imbalanced datasets on six different machine learning algorithms: K-nearest neighbor, naïve Bayes, multi-layer preceptron, logistic regression, extreme gradient boosting, and random forest classification. We use several performance metrics to evaluate these classifiers: lift, precision, accuracy, positive class accuracy, negative class accuracy, F-1, precision, recall, receiver operating characteristic area under the curve (ROC AUC), and log loss. The imbalanced dataset handling techniques we have used are: under-sampling, over-sampling, and the original dataset. The objectives of this paper are to verify that imbalanced datasets handling techniques can be used to improve classification performance and to show that no one machine learning algorithm can perform well in all of the imbalanced datasets.

Index Terms

comparison, imbalanced, sampling, data mining, SMOTE, XGBoost, MLP, bayes, nearest neighbor, random forest, logistic, lift, ROC AUC, accuracy, principal component.

I. Introduction

In this paper, we analyze six different machine learning classifier algorithms run on five imbalanced datasets with three different imbalanced dataset handling techniques. We carry out this empirical analysis using *Python 3*. We have chosen Python due to the numerous data mining packages it has. It is an interactive and object-oriented language [1].

A. Motivation

The *no-free lunch theorem* in statistics [2] essentially states that there is not one machine learning algorithm universally better than any other. Therefore, since we are not running algorithms on all possible data-generating distributions in the universe, we can expect that some learning algorithms will perform better than the others for a given dataset. While we report nine different scores from two families of metrics: *ranking* and *threshold* metrics, we particularly look into a ranking metric to rank the performance of the imbalance handling techniques. The ranking metric of choice is the ROC AUC since it is a direct trade-off between false and true positives. We also list our findings and conclusions.

We have chosen the problem of imbalanced datasets because there has been lots of attention dedicated to this type of problems and high advancement activity in this field [3]. The ratio of the minority class count to the total set can be so low that a need for a clear method to address highly imbalanced datasets becomes a must, as we shall see.

B. Relevant work

In this recent research paper, which kdnuggets ranked among the top-20 [4], a total of 17 different machine learning algorithms are examined using 121 data sets. Their conclusion was that random forest classifier is the one likely to perform the best[1].

Many publications have surveyed different data mining techniques in different fields. For example, the work in [5] reviewed a mixture of supervised and unsupervised techniques and concluded it is difficult to nominate a single data mining algorithm as the most suitable for the diagnosis and/or prognosis of diseases. There was no clear reference to performance under imbalanced datasets.

In [6], a large-scale empirical comparison between ten supervised learning methods was made, aiming to be an update to the most comprehensive STATLOG study in the mid-1990s. This paper, though comprehensive, did not include extreme gradient boosting classifiers. It assessed these ten methods using eight different performance metrics which they classified into thresholding, ranking, and probability metrics. It used 5-fold cross validation and obtained five trials. We on the other hand considered the 3-fold cross validation to be one trial on which performance metrics have been applied.

More recent work in [7] compared support vector machine (SVM) and K-nearest neighbors (KNN) against decision trees, Naïve Bayes, and multi-layer perceptrons in pseudo-randomly generated data. It was found that SVM and KNN have better performance. Similarly, there was no reference to imbalanced datasets and accuracy was used as a ranking metric in the analysis.

C. Contribution

This paper makes the following specific contributions:

- Evaluating the performance of extreme gradient boosting classifier as an ensemble learning technique for imbalanced datasets along other classifiers.
- Offering alternative measure to accuracy as an optimization objective.
- Measuring various imbalance handling techniques and showing the best in class classifier-imbalance technique combination for select imbalanced datasets.

D. Paper organization

We present the methodology followed including the learning algorithms and the relevant data mining steps in Section II. The imbalanced datasets are detailed in terms of the number of features, the size of the training data, the size of the test data, the positive class ratio, which indicates how imbalanced

[1]We believe this paper could have been published prior to extreme gradient boosting classifiers release even though both were in 2014.

the dataset is, and finally whether the dataset has missing data or not. This section further discusses the performance metrics of choice: ranking and threshold metrics.

Section III discusses how to deal with imbalanced datasets by either generating synthetic over-sampling the minority class data or under-sampling the majority class data.

With the models run on different datasets with different imbalance configurations, results are discussed in Section IV. We further study the performance per technique in Section V. The confusion matrices of the best-in-class classifier-imbalance technique are shown in Section VI and the ROC curves of these best-in-class classifiers are further shown in Section VII. Finally, we conclude the paper in Sections VIII and IX.

II. Methodology

A. Learning algorithms

We select a range of learning algorithms based on their ability to classify these various imbalanced datasets. We search in a reduced hyperparameter space of the models. This reduction in the hyperparameter search space comes at a trade-off with performance.

After splitting the data into a training and a test set—we have chosen 70-30—the search of hyperparameters is done through grid search using a 3-fold cross validation.

Splitting the data is done randomly and therefore we set the random seed to 123. The models and their mnemonics are:

- **K-nearest neighbor (KNN)**: we have chosen K to range from 1 to 20.
- **Naïve Bayes (NB)**: this model assumes conditional independence between feature distributions given its class. The parameters of Gaussian prior are computed online by counting.
- **Multi-layer preceptron (MLP)**: we use Keras and Theano and train the neural networks with gradient descent with backpropagation (sgd) and adaptive moments (adam). The loss function of choice is the binary cross-entropy function. We further vary the number of hidden units (depth) and the dimension of hidden unit (width). The activation functions we have tried are softmax and sigmoid.
- **Extreme gradient boosting (XGB)**: is a famous ensemble learning technique that uses greedy boosting [8]. The authors observed that this model is the most famous in almost all Kaggle classification competitions.
- **Random forest (RFC)**: is also an ensemble learning method which constructs several decision trees from the training data. Then to avoid over-fitting, RFC takes the statistical mode of the predicted classes of these decision trees. We use RFC with bootstrapping aggregation and out of bag sampling to be able to both fit and validate the model while being trained [9]. We use 1,000 estimators for both RFC and XGB.
- **Logistic regression (LR)**: the hyperparameter of choice here is the ridge regression regularization parameter `C` which besides shrinkage also facilitates the bias-variance trade-off [10].

B. Datasets

All these imbalanced datasets are Kaggle competitions with the exception of the fourth dataset below.

1) Credit card transaction fraud (CC): This dataset [11] aims at analyzing anonymized credit card transactions. The data is labeled as fraudulent or genuine.

2) Employee retention (ER): This dataset [12] has an objective of finding why the most experienced employees leave prematurely. The data is labeled by whether an employee left or remained.

3) Medical appointments (MA): This dataset [13] aims at predicting whether a patient would show up to the doctor office or not. This dataset is only slightly imbalanced.

4) AID373 (AID): A primary screen for endothelial differentiation, sphingolipid G-protein-coupled receptor 3. It comes from the UCI ML repository [14]. The outcome from the screening shows either active or inactive compounds.

5) Credit scoring (CS): Yet another Kaggle competition [15] to predict the future credit risk for a credit card user based on whether they were delinquent in the last two years.

Table I: Datasets

Dataset	Mnemonic	Number of features	Training size	Test size	Positive class ratio	Missing data
Credit card transaction fraud [11]	CC	31	199,364	85,443	0.17%	False
Employee retention [12]	ER	10	10,499	4,500	31.25%	False
Medical appointments [13]	MA	13	210,000	90,000	43.36%	False
AID373 [14]	AID	156	47,831	11,957	0.10%	False
Credit scoring [15]	CS	11	105,000	45,000	7.16%	True

Some of these datasets reside in relatively high dimensions. The *curse of dimensionality* plays an important role when attempting to over-sample data with the minority class. More about this to be addressed in Section III.

C. Data sanity check and cleansing

Some datasets have a column called “unnamed: 0” which is essentially the row ID and does not carry any useful information. This column is removed as part of the sanity check and data cleansing.

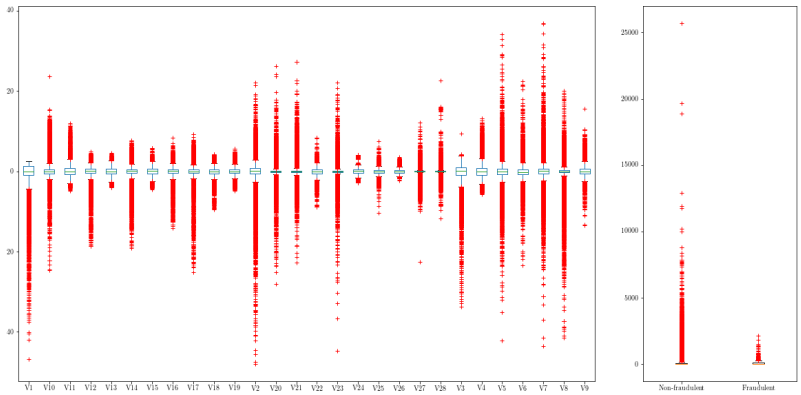


Figure 1: Box plot of all features (*left*) and plot by index of the classes (*right*) in CC

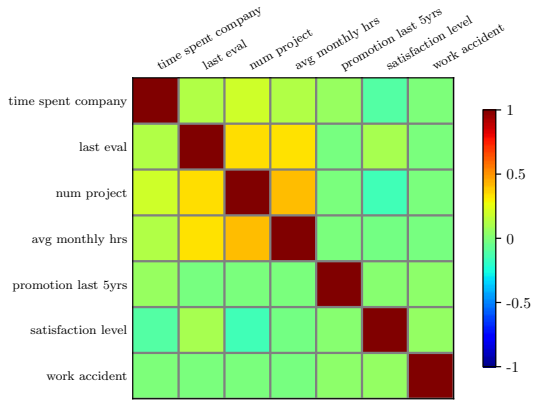


Figure 2: Correlation plot for ER

- **Data visualization and preprocessing**: as shown in Figure 1, outliers above the $Q_3+1.5(Q_3-Q_1)$ threshold or below $Q_1-1.5(Q_3-Q_1)$ threshold have been detected where Q_i is the i-th quartile. These outliers have not been removed. Features with interval data type are plotted in a correlation plot as in Figure 2 where highly correlated features are reduced. Features with nominal and ordinal data types are not included in the correlation plot. We believe that the owners of the datasets preprocessed the data before publishing it. Part of our preprocessing effort has been to perform feature scaling to the training datasets so that they all can be as close to zero mean and unit variance as possible. The same transformation is applied to the test datasets.
- **Feature selection and extraction**: we selected all the relevant features that passed the aforementioned preprocessing step. For feature extraction, principal component analysis[2] (PCA) was used in the CC dataset to keep features anonymized per the problem definition [11].
- **Missing data**: the data points were not missing at random. We have observed that when the debt ratio is high that the income is missing. When a missing data point is found, a new column is added to the dataset as a flag to capture the missing values. However, as there are other points where data exists, many of which are dependent on many other factors, we avoided imputation by mean or median, and have chosen to perform imputation using K-Nearest Neighbor, the Python implementation of which is available in [17].
- **Scaling data**: scaling the training data with unit variance and zero mean has proven important when training the MLP model. Scaling data did not seem to significantly impact the scoring of the other models. The test data is not scaled along with the training data, but it uses the same transformation.
- **Dealing with ordinal data**: we replaced binary ordinal classes with their respective 0 and 1 where the positive class (PC) is always 1 and the negative class (NC) is always 0. For example, an employee may leave the company (1) or stay (0). The positive class is always the *minority class*.

[2]The authors also ran PCA on MNIST [16] and reduced features from 784 to 40 with accurate results using a convolutional neural network and presented this in class.

- **Multi-class problem**: the CS dataset is originally a five-class problem. Since these five classes are ordinal, we relabeled the data so that classes belonging to $\{1,2,3\}$ are recoded as 0 and data in classes $\{4,5\}$ are recoded as 1. Therefore, we are back to an imbalanced binary classification problem.
- **Dropping columns**: in datasets where two columns are related, one is dropped. For example, the MA dataset had a column when the appointment was made and another when the appointment actually is using ISO 8601 formatted timestamp. Both columns were dropped in favor of a third column which was the difference in days between both times [13].

D. Hyperparameter tuning

One important consideration in performing the hyperparameter tuning is to choose a space to search in that does not cause the model to overfit the training dataset, otherwise the training dataset will perform better than the test dataset. We score the models based on the test data.

Also, increasing the size of the search space using a grid search technique [18] comes at the cost of computation, which could span several hours per model on high speed machines. The configuration of the machines used in this data mining technique is shown in Table II. The factors on which this depends are: number of hyperparameters, possible values a hyperparameter can assume, and the cross-validation size.

Table II: Configuration of machines used in data mining

Number	CPU	GPU	RAM	Harddisk type
1*	4 × Intel Core ™ i5 CPUs	–	8 GB	SSD
2	4 × Intel Core ™ i7 CPUs	–	8 GB	SSD
3†	8 × Intel Haswell™ CPUs	1 GPU	7.2 GB	Standard
4‡	8 × *(unknown)* CPUs	1 GPU	*(unknown)*	*(unknown)*

* We have used two instances of this machine.

† This is a Google Cloud Platform (GCP) compute engine.

‡ This machine was offered to us by Prof. Joydeep Ghosh. We did not check the configuration in details.

Table III: Hyperparameters used to tune the models

Model	Hyperparameter	Search range
K-Nearest Neighbors	`K`	$\{1, \ldots, 20\}$
Logistic Regression	`C`	{1e-4, ..., 1e4}
Multi-layer preceptron	`activation`	`softmax, sigmoid`
	`optimizer`	`sgd, adam`
	Hidden layer size (depth) *	{1, 3, 5}
	`output_dim`	{3, 5}
XGBoost	`objective`	`'binary:logistic', 'reg:linear'`
	`max_depth`	{2, 8}
	`min_child_weight`	{1, 6}
	`gamma`	{0, 0.4}
	`subsample`	{0.6, 1}
	`colsample_bytree`	{0.6, 1}
	`lambda`	{0, 1}
	`alpha`	{0, 1}
Random forest	`max_depth`	{3, *dynamic*}
	`criterion`	`entropy, gini`
	`max_features`	$\{10, \ldots, N\}$
	`min_samples_split`	{2, 10}
	`class_weight`	w_0, w_1

* This is a hyperparameter that is not readily available. We added it via Keras wrapper.

In Table III, N is the number of features in the dataset (excluding the label feature). The class weights used in the random forest classifier are computed as follows:

$$
\begin{aligned}
w_0 &\triangleq \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \mathbb{1}_{y_i=0} \\
w_1 &\triangleq \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \mathbb{1}_{y_i=1}
\end{aligned}
\tag{1}
$$

where $\mathbb{1}_{(\cdot)}$ is the indicator function.

E. Performance metrics

These performance metrics can be divided into two groups:

1) Ranking metrics:

- **Log loss**: for a binary classification problem, this is equal to the binary cross entropy. Log loss (actually negative log loss) is a proxy to the ROC AUC since ROC AUC is not a valid optimization objective in several classifiers [19]–[21].
- **Recall score**: is the ratio of the true positives to the total positives.
- **Precision score**: is the ratio of the true positives to the sum of true positives and false negatives. Both precision and recall scores resemble one point on the ROC AUC curve.
- **ROC AUC score**: is the area under the *receiver operating characteristic* curve which is by definition greater than or equal to 0.5 and less than or equal to 1.0 [10].

2) Threshold metrics:

- **Lift**: is a ratio between the percentage of correct classifications of the positive class made by the model to the percentage of actual positive classifications in the *test data*.
- **Accuracy**: is the proportion of the correctly predicted classes with respect to the total test data. While optimizing on accuracy may be convenient, it poses a serious risk in a highly imbalanced dataset. A trivial model predicting all ones in a dataset with 100 zero samples out of 100,000 will have an accuracy of 99.9%!
- **Positive class prediction Accuracy (PC Acc)**: is the accuracy computed over the positive (minority) class.
- **Negative class prediction Accuracy (NC Acc)**: is the accuracy computed over the negative (majority) class.
- **F-1 score**: is a scaled harmonic mean of the precision and recall scores.

These different metrics are used in different fields in matter of preference. For example, information retrieval fields prefer precision and recall while lift is used in the marketing domain [6]. We have chosen the ROC AUC score ranking metric for our ranking of the different algorithms and sampling techniques as we shall see in Section IV since it is a more comprehensive measure that captures how recall and false positive rate (known as $1 -$ sensitivity) trade off for given threshold that varies over *all* possible values on the ROC curve.

III. Dealing with imbalanced datasets

Python facilitates the use of over-sampling and under-sampling techniques [22]. We have used these different approaches in an attempt to assess the performance of the models on the datasets.

A. Over-sampling

Over-sampling the *minority class* means generating more samples for this class. The algorithm we have chosen to use is *synthetic minority over-sampling technique* (SMOTE) described in [23]. It is synthetic because samples are generated from K-nearest neighbors instead of oversampling. We use a default $K = 5$. The major problem with over-sampling is that classification is biased towards the majority class. In fact, the bias is larger for high-dimensional data, due to the *curse of dimensionality* [24].

B. Under-sampling

We under-sample the *majority class* taking $k = 0.01, 0.05, 0.10, 0.20, 0.50$. When $k = 0.50$, it means that the majority class will have labels worth 50% of the data. Therefore, this is when both classes are equal. It should be clear; however, that the 50% under-sampling and SMOTE are not the same: the latter synthesizes samples while the former uses actual samples from the original dataset (without replacement). Moreover, the dataset size is substantially smaller.

Suppose that the majority class (or class 0) has a total of N_0 rows while the minority class (or class 1) has a total of N_1 rows.

For under-sampling, we need to find the majority class sample size $u \leq N_1$ given the minor sample size v. For this we use a proportion computation. If k is the proportion of under-sampling, then we can write:

$$\begin{aligned} k &= \frac{u}{u+v} \\ u &= \left\lceil \frac{vk}{1-k} \right\rceil \end{aligned} \tag{2}$$

which makes it clear that when $k = 0.5$, $u = v$.

In the case where $u > N_1$ the equation above has no solution and the resulting under-sampling of the majority is simply N/A, which we shall see in Section IV.

C. No sampling

The final technique that we use is to analyze the imbalanced data in its original form without any form of sampling. That is, the original imbalanced dataset will be used to train the models. While stratification may work in such cases, we have opted for using the original data as it came from the source.

IV. Performance by dataset

We list the performance metrics by dataset and models for the various imbalanced techniques. For the ROC AUC performance metric, we color the ones that are the highest for a given dataset and model pair across all the different sampling techniques in blue, and generate their ROC curves in Section VII.

A. Original imbalanced data

Table IV: Performance metrics by dataset and model (original imbalanced data)

Dataset	Model	Lift	Accuracy	PC Acc	NC Acc	F-1	Precision	Recall	AUC	Log-loss
CC	KNN	0.0000	0.9981	0.0000	1.0000	0.0000	0.0000	0.0000	0.6403	0.0647
	NB	327.0865	0.9928	0.6125	0.9936	0.2426	0.1512	0.6125	0.9700	0.2474
	MLP	383.8260	0.9993	0.7188	0.9999	0.8014	0.9055	0.7188	0.9818	0.0230
	XGB	437.2279	0.9995	0.8187	0.9999	0.8704	0.9291	0.8187	0.9734	0.0158
	RFC	110.1414	0.9985	0.2062	1.0000	0.3420	1.0000	0.2062	0.9693	0.0513
	LR	337.0993	0.9989	0.6312	0.9996	0.6871	0.7537	0.6312	0.9214	0.0372
ER	KNN	3.8604	0.9400	0.9188	0.9466	0.8794	0.8432	0.9188	0.9725	2.0724
	NB	3.4327	0.6511	0.8170	0.5993	0.5271	0.3891	0.8170	0.7991	12.0504
	MLP	3.6642	0.9471	0.8721	0.9705	0.8870	0.9024	0.8721	0.9689	1.8267
	XGB	4.0761	0.9900	0.9701	0.9962	0.9788	0.9876	0.9701	0.9936	0.3454
	RFC	3.8408	0.9776	0.9141	0.9974	0.9509	0.9909	0.9141	0.9930	0.7752
	LR	1.3809	0.7893	0.3287	0.9332	0.4262	0.6059	0.3287	0.8195	7.2762
MA	KNN	0.2757	0.6846	0.0834	0.9456	0.1381	0.3995	0.0834	0.5591	10.8928
	NB	0.4844	0.6691	0.1466	0.8959	0.2115	0.3794	0.1466	0.5718	11.4286
	MLP	0.0482	0.6975	0.0146	0.9938	0.0283	0.5070	0.0146	0.6033	10.4495
	XGB	0.0522	0.6976	0.0158	0.9935	0.0306	0.5131	0.0158	0.6041	10.4453
	RF	0.0000	0.6973	0.0000	1.0000	0.0000	0.0000	0.0000	0.5965	10.4537
	LR	0.0160	0.6971	0.0048	0.9975	0.0096	0.4567	0.0048	0.5865	10.4633
AID	KNN	83.0347	0.9987	0.0833	0.9996	0.1111	0.1667	0.0833	0.5415	0.0462
	NB	830.3472	0.3599	0.8333	0.3594	0.0026	0.0013	0.8333	0.5964	22.1097
	MLP	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.8057	0.0347
	XGB	166.0694	0.9990	0.1667	0.9998	0.2500	0.5000	0.1667	0.8795	0.0347
	RFC	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.8791	0.0347
	LR	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.4147	0.0347
CS	KNN	0.0050	0.9332	0.0003	1.0000	0.0007	1.0000	0.0003	0.5306	2.3064
	NB	5.6125	0.9155	0.3749	0.9542	0.3721	0.3694	0.3749	0.7526	2.9189
	MLP	3.0926	0.9374	0.2066	0.9898	0.3061	0.5909	0.2066	0.8699	2.1606
	XGB	2.6942	0.9373	0.1800	0.9915	0.2771	0.6018	0.1800	0.8730	2.1667
	RFC	0.0000	0.9332	0.0000	1.0000	0.0000	0.0000	0.0000	0.8594	2.3072
	LR	0.4582	0.9338	0.0306	0.9985	0.0582	0.5935	0.0306	0.8300	2.2849

Table V: Performance metrics by dataset and model (over-sampled data using SMOTE)

Dataset	Model	Lift	Accuracy	PC Acc	NC Acc	F-1	Precision	Recall	AUC	Log-loss
CC	KNN	267.0094	0.9456	0.5000	0.9465	0.0333	0.0172	0.5000	0.7471	1.8781
	NB	380.4884	0.9923	0.7125	0.9928	0.2579	0.1575	0.7125	0.9645	0.2652
	MLP	473.9416	0.9937	0.8875	0.9939	0.3442	0.2135	0.8875	0.9657	0.2187
	XGB	457.2536	0.9990	0.8562	0.9993	0.7675	0.6954	0.8562	0.9799	0.0336
	RFC	457.2536	0.9994	0.8562	0.9997	0.8536	0.8509	0.8562	0.9680	0.0190
	LR	483.9545	0.9830	0.9062	0.9832	0.1666	0.0917	0.9062	0.9761	0.5866
ER	KNN	3.9938	0.8876	0.9505	0.8679	0.8009	0.6920	0.9505	0.9685	3.8838
	NB	3.5426	0.5182	0.8431	0.4167	0.4545	0.3111	0.8431	0.7887	16.6404
	MLP	3.8329	0.9218	0.9122	0.9248	0.8474	0.7911	0.9122	0.9668	2.7017
	XGB	4.0918	0.9902	0.9739	0.9953	0.9793	0.9849	0.9739	0.9941	0.3377
	RFC	4.0840	0.9891	0.9720	0.9945	0.9770	0.9821	0.9720	0.9930	0.3761
	LR	3.4171	0.7593	0.8133	0.7425	0.6166	0.4966	0.8133	0.8253	8.3125
MA	KNN	0.9238	0.6111	0.2796	0.7549	0.3032	0.3312	0.2796	0.5298	13.4338
	NB	2.8370	0.3958	0.8587	0.1949	0.4624	0.3164	0.8587	0.5637	20.8692
	MLP	1.8943	0.5484	0.5733	0.5376	0.4346	0.3499	0.5733	0.5836	15.5984
	XGB	1.1321	0.6387	0.3427	0.7671	0.3647	0.3897	0.3427	0.5904	12.4805
	RFC	0.9172	0.6201	0.2776	0.7688	0.3067	0.3426	0.2776	0.5422	13.1199
	LR	1.9324	0.5549	0.5849	0.5419	0.4430	0.3566	0.5849	0.5847	15.3723
AID	KNN	415.1736	0.8998	0.4167	0.9003	0.0083	0.0042	0.4167	0.7206	3.4606
	NB	830.3472	0.3601	0.8333	0.3596	0.0026	0.0013	0.8333	0.5966	22.1011
	MLP	83.0347	0.9954	0.0833	0.9963	0.0351	0.0222	0.0833	0.7512	0.1589
	XGB	166.0694	0.9990	0.1667	0.9998	0.2500	0.5000	0.1667	0.7609	0.0347
	RFC	166.0694	0.9990	0.1667	0.9998	0.2500	0.5000	0.1667	0.7792	0.0347
	LR	664.2778	0.9051	0.6667	0.9053	0.0139	0.0070	0.6667	0.8152	3.2786
CS	KNN	4.9054	0.6796	0.3277	0.7047	0.1202	0.0736	0.3277	0.5189	11.0680
	NB	12.1165	0.3979	0.8094	0.3685	0.1523	0.0840	0.8094	0.7190	20.7959
	MLP	7.8635	0.8956	0.5253	0.9221	0.4019	0.3255	0.5253	0.8487	3.6067
	XGB	3.4910	0.9374	0.2332	0.9879	0.3325	0.5789	0.2332	0.8713	2.1606
	RFC	3.5508	0.9367	0.2372	0.9867	0.3335	0.5614	0.2372	0.8578	2.1875
	LR	10.8914	0.8378	0.7275	0.8457	0.3747	0.2523	0.7275	0.8679	5.6023

Table VI: Performance metrics by dataset and model (under-sampled data at 1%)

Dataset	Model	Lift	Accuracy	PC Acc	NC Acc	F-1	Precision	Recall	AUC	Log-loss
CC	KNN	6.6752	0.9981	0.0125	1.0000	0.0242	0.4000	0.0125	0.6853	0.0651
	NB	333.7617	0.9930	0.6250	0.9937	0.2500	0.1562	0.6250	0.9656	0.2425
	MLP	440.5655	0.9990	0.8250	0.9993	0.7564	0.6984	0.8250	0.9819	0.0344
	XGB	447.2407	0.9992	0.8375	0.9995	0.8048	0.7746	0.8375	0.9753	0.0263
	RF	393.8388	0.9992	0.7375	0.9997	0.7738	0.8138	0.7375	0.9661	0.0279
	LR	203.5946	0.9984	0.3812	0.9995	0.4692	0.6100	0.3812	0.7914	0.0558
ER	KNN	N/A*								
	NB									
	MLP									
	XGB									
	RFC									
	LR									
MA	KNN	N/A*								
	NB									
	MLP									
	XGB									
	RFC									
	LR									
AID	KNN	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.5573	0.0347
	NB	830.3472	0.3599	0.8333	0.3594	0.0026	0.0013	0.8333	0.5964	22.1097
	MLP	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.7860	0.0347
	XGB	249.1042	0.9987	0.2500	0.9995	0.2857	0.3333	0.2500	0.8523	0.0433
	RFC	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.8686	0.0347
	LR	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.4158	0.0347
CS	KNN	N/A*								
	NB									
	MLP									
	XGB									
	RFC									
	LR									

* This under-sampling size is not possible. Equation 2 has the details.

Table VII: Performance metrics by dataset and model (under-sampled data at 5%)

Dataset	Model	Lift	Accuracy	PC Acc	NC Acc	F-1	Precision	Recall	AUC	Log-loss
CC	KNN	43.3890	0.9960	0.0813	0.9977	0.0703	0.0619	0.0813	0.7134	0.1391
	NB	340.4370	0.9909	0.6375	0.9916	0.2080	0.1242	0.6375	0.9694	0.3141
	MLP	457.2536	0.9985	0.8562	0.9987	0.6749	0.5569	0.8562	0.9777	0.0534
	XGB	453.9159	0.9988	0.8500	0.9990	0.7196	0.6239	0.8500	0.9769	0.0428
	RF	430.5526	0.9991	0.8063	0.9995	0.7725	0.7414	0.8063	0.9639	0.0307
	LR	246.9837	0.9981	0.4625	0.9991	0.4805	0.5000	0.4625	0.8304	0.0647
ER	KNN	N/A*								
	NB									
	MLP									
	XGB									
	RFC									
	LR									
MA	KNN	N/A*								
	NB									
	MLP									
	XGB									
	RFC									
	LR									
AID	KNN	0.0000	0.9982	0.0000	0.9992	0.0000	0.0000	0.0000	0.5151	0.0607
	NB	830.3472	0.3594	0.8333	0.3589	0.0026	0.0013	0.8333	0.5961	22.1271
	MLP	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.8811	0.0347
	XGB	415.1736	0.9964	0.4167	0.9970	0.1887	0.1220	0.4167	0.8797	0.1242
	RFC	249.1042	0.9988	0.2500	0.9996	0.3000	0.3750	0.2500	0.8969	0.0404
	LR	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.4281	0.0347
CS	KNN	N/A*								
	NB									
	MLP									
	XGB									
	RFC									
	LR									

* This under-sampling size is not possible. Equation 2 has the details.

Table VIII: Performance metrics by dataset and model (under-sampled data at 10%)

Dataset	Model	Lift	Accuracy	PC Acc	NC Acc	F-1	Precision	Recall	AUC	Log-loss
CC	KNN	103.4661	0.9867	0.1938	0.9882	0.0516	0.0298	0.1938	0.7094	0.4604
	NB	340.4370	0.9909	0.6375	0.9915	0.2075	0.1239	0.6375	0.9690	0.3149
	MLP	470.6040	0.9965	0.8812	0.9967	0.4837	0.3333	0.8812	0.9859	0.1217
	XGB	467.2664	0.9974	0.8750	0.9976	0.5589	0.4106	0.8750	0.9801	0.0893
	RFC	463.9288	0.9983	0.8688	0.9985	0.6511	0.5206	0.8688	0.9729	0.0602
	LR	280.3598	0.9980	0.5250	0.9989	0.4985	0.4746	0.5250	0.8558	0.0683
ER	KNN	N/A*								
	NB									
	MLP									
	XGB									
	RFC									
	LR									
MA	KNN	N/A*								
	NB									
	MLP									
	XGB									
	RFC									
	LR									
AID	KNN	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.6377	0.0347
	NB	830.3472	0.3585	0.8333	0.3581	0.0026	0.0013	0.8333	0.5957	22.1559
	MLP	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.8537	0.0347
	XGB	415.1736	0.9908	0.4167	0.9914	0.0833	0.0463	0.4167	0.9225	0.3178
	RFC	415.1736	0.9951	0.4167	0.9956	0.1449	0.0877	0.4167	0.9120	0.1704
	LR	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.4165	0.0347
CS	KNN	0.0050	0.9332	0.0003	1.0000	0.0007	0.5000	0.0003	0.5312	2.3072
	NB	6.1105	0.9095	0.4082	0.9454	0.3760	0.3486	0.4082	0.7533	3.1254
	MLP	4.8506	0.9358	0.3240	0.9796	0.4029	0.5325	0.3240	0.8727	2.2159
	XGB	4.6813	0.9366	0.3127	0.9813	0.3974	0.5449	0.3127	0.8735	2.1882
	RFC	2.6793	0.9371	0.1790	0.9913	0.2753	0.5965	0.1790	0.8655	2.1736
	LR	0.0100	0.9328	0.0007	0.9995	0.0013	0.0833	0.0007	0.6657	2.3225

* This under-sampling size is not possible. Equation 2 has the details.

Table IX: Performance metrics by dataset and model (under-sampled data at 20%)

Dataset	Model	Lift	Accuracy	PC Acc	NC Acc	F-1	Precision	Recall	AUC	Log-loss
CC	KNN	150.1928	0.9649	0.2812	0.9662	0.0291	0.0154	0.2812	0.7054	1.2123
	NB	360.4627	0.9896	0.6750	0.9902	0.1953	0.1142	0.6750	0.9686	0.3598
	MLP	473.9416	0.9879	0.8875	0.9881	0.2160	0.1229	0.8875	0.9808	0.4168
	XGB	473.9416	0.9931	0.8875	0.9933	0.3257	0.1994	0.8875	0.9749	0.2377
	RFC	473.9416	0.9949	0.8875	0.9951	0.3939	0.2531	0.8875	0.9747	0.1767
	LR	293.7103	0.9975	0.5500	0.9984	0.4560	0.3894	0.5500	0.8834	0.0849
ER	KNN	N/A*								
	NB									
	MLP									
	XGB									
	RFC									
	LR									
MA	KNN	N/A*								
	NB									
	MLP									
	XGB									
	RFC									
	LR									
AID	KNN	166.0694	0.9423	0.1667	0.9431	0.0058	0.0029	0.1667	0.5678	1.9932
	NB	830.3472	0.3509	0.8333	0.3504	0.0026	0.0013	0.8333	0.5920	22.4188
	MLP	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.7472	0.0347
	XGB	498.2083	0.9370	0.5000	0.9375	0.0157	0.0080	0.5000	0.9169	2.1752
	RF	498.2083	0.9893	0.5000	0.9898	0.0857	0.0469	0.5000	0.8875	0.3697
	LR	0.0000	0.9990	0.0000	1.0000	0.0000	0.0000	0.0000	0.3959	0.0347
CS	KNN	0.1145	0.9290	0.0077	0.9949	0.0142	0.0970	0.0077	0.5407	2.4538
	NB	6.9073	0.8900	0.4614	0.9207	0.3592	0.2940	0.4614	0.7522	3.7986
	MLP	7.8087	0.9166	0.5216	0.9448	0.4551	0.4036	0.5216	0.8690	2.8821
	XGB	7.5548	0.9222	0.5047	0.9521	0.4642	0.4297	0.5047	0.8733	2.6879
	RFC	6.0109	0.9303	0.4015	0.9681	0.4348	0.4741	0.4015	0.8681	2.4085
	LR	0.3635	0.9333	0.0243	0.9983	0.0464	0.5105	0.0243	0.7778	2.3049

* This under-sampling size is not possible. Equation 2 has the details.

Table X: Performance metrics by dataset and model (under-sampled data at 50%)

Dataset	Model	Lift	Accuracy	PC Acc	NC Acc	F-1	Precision	Recall	AUC	Log-loss
CC	KNN	293.7103	0.7012	0.5500	0.7015	0.0068	0.0034	0.5500	0.6816	10.3211
	NB	377.1507	0.9883	0.7063	0.9888	0.1846	0.1062	0.7063	0.9686	0.4034
	MLP	490.6297	0.9630	0.9187	0.9631	0.0851	0.0446	0.9187	0.9836	1.2778
	XGB	487.2921	0.9594	0.9125	0.9595	0.0776	0.0405	0.9125	0.9730	1.4035
	RFC	493.9673	0.9594	0.9250	0.9594	0.0786	0.0410	0.9250	0.9752	1.4031
	LR	487.2921	0.9591	0.9125	0.9592	0.0771	0.0403	0.9125	0.9796	1.4128
ER	KNN	3.9388	0.8849	0.9374	0.8685	0.7949	0.6900	0.9374	0.9603	3.9759
	NB	3.6328	0.5293	0.8646	0.4246	0.4665	0.3194	0.8646	0.8027	16.2566
	MLP	3.8761	0.9160	0.9225	0.9140	0.8394	0.7701	0.9225	0.9653	2.9013
	XGB	4.0918	0.9816	0.9739	0.9840	0.9617	0.9499	0.9739	0.9932	0.6371
	RFC	4.0918	0.9858	0.9739	0.9895	0.9702	0.9666	0.9739	0.9934	0.4912
	LR	3.4837	0.7651	0.8291	0.7451	0.6269	0.5040	0.8291	0.8278	8.1129
MA	KNN	1.4993	0.5829	0.4538	0.6389	0.3971	0.3529	0.4538	0.5636	14.4067
	NB	0.6865	0.6527	0.2078	0.8458	0.2659	0.3690	0.2078	0.5715	11.9962
	MLP	1.8857	0.5768	0.5707	0.5794	0.4494	0.3706	0.5707	0.6016	14.6186
	XGB	1.9973	0.5636	0.6045	0.5459	0.4561	0.3662	0.6045	0.6032	15.0714
	RFC	2.0850	0.5429	0.6311	0.5046	0.4552	0.3561	0.6311	0.5960	15.7880
	LR	1.8853	0.5584	0.5706	0.5531	0.4389	0.3566	0.5706	0.5867	15.2530
AID	KNN	415.1736	0.5828	0.4167	0.5829	0.0020	0.0010	0.4167	0.5318	14.4115
	NB	913.3819	0.2454	0.9167	0.2447	0.0024	0.0012	0.9167	0.5933	26.0643
	MLP	332.1389	0.4727	0.3333	0.4728	0.0013	0.0006	0.3333	0.4455	18.2129
	XGB	664.2778	0.6988	0.6667	0.6989	0.0044	0.0022	0.6667	0.7661	10.4020
	RF	747.3125	0.7302	0.7500	0.7302	0.0055	0.0028	0.7500	0.8528	9.3188
	LR	830.3472	0.7018	0.8333	0.7017	0.0056	0.0028	0.8333	0.8252	10.2980
CS	KNN	7.9731	0.5399	0.5326	0.5404	0.1339	0.0766	0.5326	0.5457	15.8920
	NB	10.2938	0.7092	0.6876	0.7107	0.2400	0.1454	0.6876	0.7575	10.0456
	MLP	11.2798	0.8241	0.7535	0.8291	0.3639	0.2399	0.7535	0.8695	6.0766
	XGB	11.7828	0.7960	0.7871	0.7966	0.3401	0.2169	0.7871	0.8717	7.0468
	RFC	11.8476	0.7906	0.7914	0.7906	0.3356	0.2129	0.7914	0.8674	7.2310
	LR	11.3346	0.8148	0.7572	0.8189	0.3532	0.2303	0.7572	0.8669	6.3983

Some of the datasets show lift scores that are close to zero. For these, we observe that the recall is also close to zero, which means that the model for this dataset and sampling technique is failing to detect the minor class at this given point on the ROC curve. Since the ROC AUC score is above 0.5, it means that the model is doing better in predicting the majority class than the random predictor.

V. Performance by sampling technique

In Section IV, we have highlighted the top ROC AUC results. Here we summarize the results and group them. We further count the number of wins that sampling technique achieved and rank them accordingly.

Table XI: ROC AUC performance by sampling technique

Dataset	Model	Sampling technique	Best ROC AUC
CC	KNN	SMOTE	0.7471
	NB	Original	0.9700
	MLP	Under-sampling at 10%	0.9859
	XGB	Under-sampling at 10%	0.9801
	RFC	Under-sampling at 50%	0.9752
	LR	Under-sampling at 50%	0.9796
ER	KNN	Original	0.9725
	NB	Original	0.7991
	MLP	Original	0.9689
	XGB	SMOTE	0.9941
	RFC	Under-sampling at 50%	0.9934
	LR	Under-sampling at 50%	0.8278
MA	KNN	Under-sampling at 50%	0.5636
	NB	Original	0.5718
	MLP	Original	0.6033
	XGB	Original	0.6041
	RFC	Original	0.5965
	LR	Under-sampling at 50%	0.5867
AID	KNN	SMOTE	0.7206
	NB	SMOTE	0.5966
	MLP	Under-sampling at 5%	0.8811
	XGB	Under-sampling at 10%	0.9225
	RFC	Under-sampling at 10%	0.9120
	LR	Under-sampling at 50%	0.8252
CS	KNN	Under-sampling at 50%	0.5457
	NB	Under-sampling at 50%	0.7575
	MLP	Under-sampling at 10%	0.8727
	XGB	Under-sampling at 10%	0.8735
	RFC	Under-sampling at 20%	0.8681
	LR	SMOTE	0.8679

Based on Table XI, we rank the sampling techniques by the number of top performing contributions as follows:

Table XII: ROC AUC performance ranked by sampling technique

Rank	Technique	Count
1	Under-sampling at 50%	9
2	Original	8
3	Under-sampling at 10%	6
4	SMOTE	5
5	Under-sampling at 20%	1
	Under-sampling at 5%	1
7	Under-sampling at 1%	0
		Total = 30

VI. Confusion matrices

Here we show the normalized confusion matrices for the model with the highest ROC AUC given each dataset using the best sampling technique per dataset.

Table XIII: Best performing models and sampling technique

Dataset	Best scoring model and technique	ROC AUC
CC	MLP on under-sampling at 10%	0.9859
ER	XGB on under-sampling at 50%	0.9941
MA	XGB on original dataset	0.6041
AID	XGB on under-sampling at 10%	0.9225
CS	XGB on under-sampling at 10%	0.8735

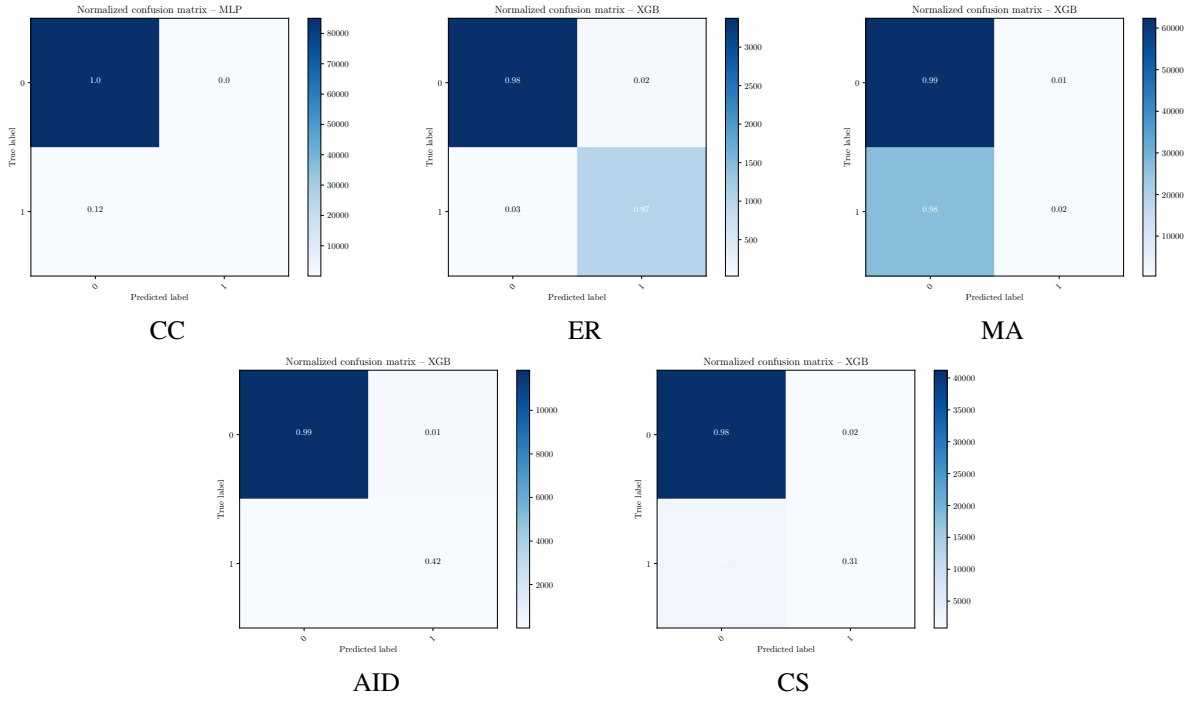
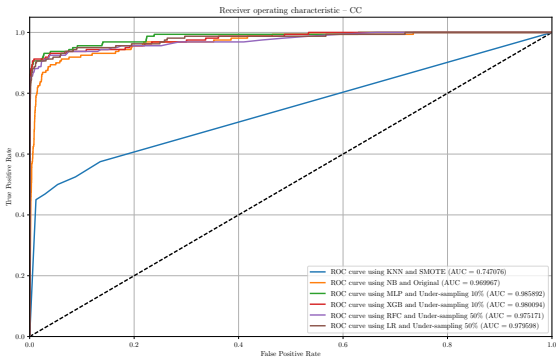


Figure 3: Normalized confusion matrices

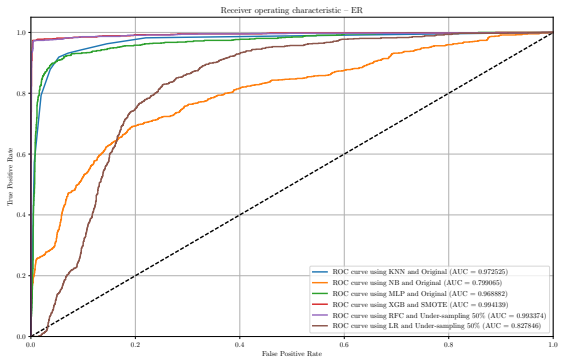
We observe that classification of MA has the best ROC AUC score using the original dataset, aligned with the fact that this dataset is close to being balanced. Therefore, sampling did not help much.

VII. Receiver operating characteristic

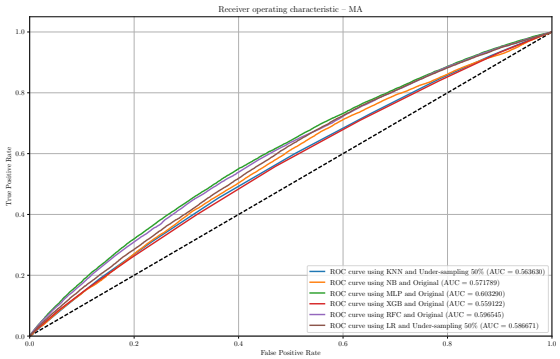
We now show one ROC curve per model for the best in class results using original, over-sampled, and under-sampled data as per the results outlined in Section IV.



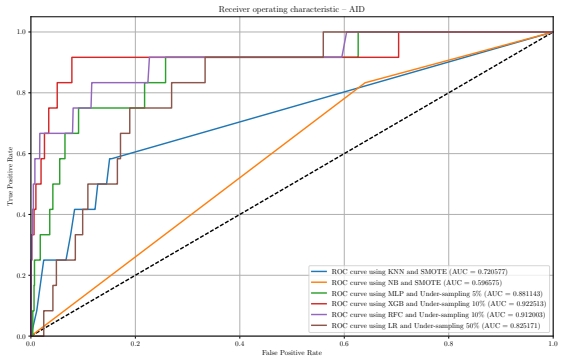
CC



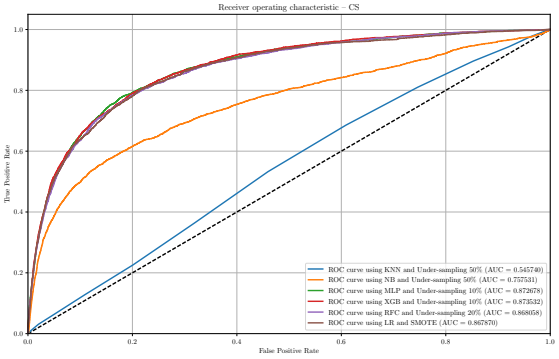
ER



MA



AID



CS

Figure 4: Best in class ROC curves

VIII. Lessons learned

We learn that PCA and NB work well together since features become independent and NB assumes conditional independence of the features distributions given the class predicted. In fact, the combination of PCA and NB proved helpful in the CC dataset. In banking industry, where data flows in streaming fashion, this could help the bank make decisions "on-the-fly" with the condition that the PCA bases vectors are updated periodically.

Further, scaling the features prior to MLP is important so that the weights of the activation functions are comparable.

When the dataset is highly imbalanced, optimizing on predicted class accuracy is a not indicative of performance. We have observed that over-sampled and original imbalanced dataset had tendency to take the longest execution times, and under-sampling tends to finish execution faster due to reduced number of rows.

Optimizing on ROC AUC is ideal for improving ROC; however, ROC AUC is not a readily available optimization metric in MLP, and thus a surrogate log loss metric is used which is equal to the binary cross-entropy in a binary classification problem.

A particular observation about the MA dataset is that its performance scores are significantly lower than the other datasets. We believe searching in the hyperparameter space could help us find better performance values, but due to our desire to have unified range of hyperparameters for all datasets, this variation in performance is deemed to happen.

IX. Conclusion

In conclusion, we believe that dealing with highly imbalanced datasets is possible with a high level of performance. The most important matter that must be adhered to is avoiding the default accuracy metric. ROC AUC or its surrogate log loss metric can prove very helpful in improving the classification performance. All three techniques viewed: under-sampling, original, and over-sampling generated comparable results when ranked. Similarly, all six machine learning models were assessed, with MLP and XGB coming at the top—a finding not captured in current relevant literature.

APPENDIX

The code required to generate all the work done here can be found at https://github.com/farismismar/EE380L/. Here is a brief description of the folder content:

- `mm_top_v5.py`: runs the method referenced in this paper and generates several output files for different sampling techniques. This is written in Python 3.6 and requires Theano, Keras, Pandas, imblearn, scikit-learn, and numpy to run. The dataset folder must be in the same folder as this python file. The first two letters are replaced with the dataset mnemonic.
- `mm_xtrn_metrics.txt`: the output file which contains all the metrics referenced in this paper.

ACKNOWLEDGEMENTS

The authors would like to thank Prof. Joydeep Ghosh for his guidance and advice throughout Spring 2017 and providing us access to a GPU enabled server, which helped us save data mining time.

REFERENCES

[1] "General Python FAQ," https://docs.python.org/3/faq/general.html#what-is-python, [Online; accessed 03-22-2017].

[2] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, Apr 1997.

[3] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, Sept 2009.

[4] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *Journal of Machine Learning Research*, vol. 15, pp. 3133–3181, 2014. [Online]. Available: http://jmlr.org/papers/v15/delgado14a.html

[5] C. Gera and K. Joshi, "A Survey on Data Mining Techniques in the Medicative Field," *International Journal of Computer Applications*, vol. 113, 2015.

[6] R. Caruana and A. Niculescu-Mizil, "An Empirical Comparison of Supervised Learning Algorithms," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 161–168. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143865

[7] J. Zhu, C. Xu, Z. Li, G. Fung, X. Lin, J. Huang, and C. Huang, "An examination of on-line machine learning approaches for pseudo-random generated data," *Cluster Computing*, vol. 19, no. 3, pp. 1309–1321, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10586-016-0586-5

[8] DMLC, "Extreme Gradient Boosting Algorithm," http://xgboost.readthedocs.io/en/latest/model.html, [Online; accessed 04-02-2017].

[9] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[10] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*, ser. Springer Texts in Statistics. Springer New York, 2014. [Online]. Available: https://books.google.com/books?id=at1bmAEACAAJ

[11] "Credit Card Fraud Detection Kaggle Competition," https://www.kaggle.com/dalpozz/creditcardfraud, 2016, [Online; accessed 03-18-2017].

[12] "Human Resources Analytics Kaggle Competition," https://www.kaggle.com/ludobenistant/hr-analytics, 2016, [Online; accessed 04-23-2017].

[13] "Medical Appointment No Shows Kaggle Competition," https://www.kaggle.com/joniarroba/noshowappointments, 2017, [Online; accessed 03-18-2017].

[14] "Scripps Research Institute Molecular Screening Center," http://archive.ics.uci.edu/ml/datasets/PubChem+Bioassay+Data, 2016, [Online; accessed 04-28-2017].

[15] "Give Me Some Credit Kaggle Competition," https://www.kaggle.com/c/GiveMeSomeCredit, 2011, [Online; accessed 04-23-2017].

[16] "Digit Recognizer Kaggle Competition," https://www.kaggle.com/c/digit-recognizer/data, 2014, [Online; accessed 03-18-2017].

[17] "fancyimpute 0.2.0," https://pypi.python.org/pypi/fancyimpute/, 2017, [Online; accessed 04-29-2017].

[18] "scikit-learn developers: Tuning the hyper-parameters of an estimator," http://scikit-learn.org/stable/modules/grid_search.html, 2016, [Online; accessed 04-02-2017].

[19] "Keras documentation," https://keras.io/, 2017, [Online; accessed 03-29-2017].

[20] "Theano 0.9.0 documentation," http://deeplearning.net/software/theano/index.html, 2017, [Online; accessed 03-29-2017].

[21] "Tensorflow documentation," https://www.tensorflow.org/, 2017, [Online; accessed 03-29-2017].

[22] "imbalanced-learn API," http://contrib.scikit-learn.org/imbalanced-learn/api.html, 2016, [Online; accessed 04-18-2017].

[23] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research 16*, pp. 321—-357, 2002. [Online]. Available: https://www.jair.org/media/953/live-953-2037-jair.pdf/

[24] R. Blagus and L. Lusa, "SMOTE for high-dimensional class-imbalanced data," *BMC Bioinformatics*, vol. 14, no. 1, p. 106, 2013. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-14-106