The influence of distribution characteristics and data balancing on classification bias in highly unbalanced data sets

Zekiye Erarslan, Manuel Günther, Artemii Redkin, Matteo Zannini

30.09.2023

1 Introduction

In recent years, the field of machine learning has evolved from an emerging science into a widely applied technology, finding applications across various domains such as business, industry, and scientific research. However, as machine learning techniques have gained prominence, a critical challenge has surfaced known as the class imbalance problem. This problem, characterized by a significant disparity in the number of samples between different classes within a dataset, has profound implications for classification performance and decision-making in real-world applications.

It has become increasingly apparent that imbalanced datasets can lead to suboptimal classification results, prompting researchers to explore solutions to mitigate its impact. The class imbalance problem is widespread, affecting a substantial portion of the data mining community. It is essential to understand that class imbalances can manifest in various application domains, including fraud detection, risk management, text classification, and medical diagnosis. In some cases, these imbalances are inherent to the problem, while in others, they arise due to limitations in data collection processes or the need for human intervention in selecting examples for training.

Both data-level and algorithmic-level solutions have been proposed to address class imbalance. These include resampling techniques such as oversampling and undersampling, adjustments to class-specific costs, threshold tuning, and recognition-based learning approaches. Researchers have dedicated considerable effort to developing and refining these methods, aiming to improve the performance and fairness of machine learning models in the face of imbalanced data.

In this report, we illustrate a comparative analysis aiming to highlight the influence of data balancing and other factors in the process of classifying imbalanced datasets. In addition to assessing the performance of established data-level balancing algorithms, our objective includes the exploration of diverse classification scenarios wherein various factors may exert an influence on classification efficacy.

The report is organized into distinct sections, including Methods, Results, and Discussion. Within the Methods section, we comprehensively elucidate our approach, which comprises a four-step pipeline encompassing data generation, balancing, classification, and output analysis. It's noteworthy to emphasize that each step of this methodology is supported by extensive literature research, ensuring a robust foundation for our approach.

In the Results section, we provide the most significant findings derived from the systematic variation of parameters within our defined pipeline. Through rigorous experimentation, we explore the impact of parameter adjustments on the classification outcomes. To facilitate a comprehensive understanding of our findings, we employ data visualization techniques and provide insightful plots that vividly represent the observed results.

In the final section, the Discussion, we delve into a thorough analysis of the results and draw meaningful conclusions from our study. We closely examine what our findings mean in practical terms and how they can be applied in real-world situations. Additionally, we consider areas where our study could be improved or enhanced for future research. By acknowledging both the strengths and limitations of our work, we lay the groundwork for future studies to build upon our findings and advance the field of classifying imbalanced datasets.

2 Methods

This section will describe the different balancing methods and classifiers we used to obtain our results. For each balancer and classifier we will give a short theoretical description as to how it works, what the strengths and weaknesses of the method are, and which challenges we encountered in the implementation.

2.1 Data balancing

As for the data balancing step, here are some important characteristics that determine the usefulness of a balancer:

- The balancer should be generalisable and applicable to different levels of imbalance and subsequent classification methods;
- It should maintain the actual class structure and represent the minority class pattern accurately;
- It should support a correct class identification.

We hereby summarize the main features of the data-level balancing algorithms that we implemented in the pipeline. Typically, these methods are categorised into three main groups: synthetic samplers, resamplers and hybrid samplers.

2.1.1 SMOTE

SMOTE (Synthetic Minority Oversampling Technique) is probably the most known of the balancing methods. SMOTE is an over-sampling technique which generates synthetic minority class samples by considering feature vectors. Given the parameter k, at every iteration the algorithm does the following steps:

For each minority class sample, it calculates the distances between the sample and its k nearest neighbors, with k being selected depending on the needed amount of oversampled instances. A random scaling factor between 0 and 1 is applied to these differences, and the results are added to the original sample's feature vector. This process creates new data points along line segments between features, effectively making the decision boundary of the minority class more inclusive. We decided to include SMOTE in our study as it is a reference method, even though important drawbacks have been detected in previous articles, such as presenting computational complexity quadratic in the size of the minority class and distortion of its distribution due to localization of the selected target points.

2.1.2 Borderline SMOTE

ADASYN is another synthetic sampler

3 Machine Learning Pipeline

This project was focused on two aspects. The first was creating a machine learning pipeline that could facilitate an easy way to conduct experiments on imbalanced data. The second was to use it to conduct some experiments to show it functions and to gain some additional insight into the imbalance problem itself.

Along the way we created three approaches to the pipeline, where the idea with each successive approach was to either improve generality and the potential scope of experiments that could be run, or improve the efficiency of the pipeline computation- or memory-wise to obtain results quicker. The fundamental layout of all three approaches was the same in the sense that each consisted of four or five python classes:

1) A **generator** class that creates the input data for a variety of parameters like the number, dimensionality and imbalance ratio of the samples

2) A **balancer** class that subjects the created data to a chosen balancing method to remedy the class imbalance

3) A **classifier** class that trains a chosen classifier on the output of the balancer class and conducts predictions on the test-set

4) An **metrics** or **assessor** class that applies standard metrics of classification quality to the predictions or guides the flow of the pipeline

The first version of the pipeline is essentially a more elegant packaging for the application of methods from `scikit-learn`, `imblearn`, and some more specialised libraries like `xgboost`. In this version both the balancer and classifier classes are essentially just wrappers that invoke the balancing, training and prediction methods of the imported classes from these libraries. Experiments are conducted directly via for-loop iteration over the experiment parameters.

The second approach was based on two ideas. One is to incorporate a new generator that allows more flexible generation of feature data. The `make_classification` method used in the first approach creates clusters exclusively with Gaussian features and does not allow to specify means or variances. We created an own generator to amplify the control over and the variety of our sample distributions. The other idea was to expand the functionality of the balancer and classifier classes by locating parts of the necessary iterations in a larger experiment in these classes. The hope was to allow brief experiments to be conducted in a more modularised way and improve efficiency. Since the goal of better efficiency was not achieved in this case we created a third approach.

Our third approach was mainly focused on computational efficiency. Stacked for-loop iterations turned out to be fairly slow and required a large amount of time to complete. This is to some extend inevitable as the involved algorithms are complex and the datasets large, but the previous pipelines also created large additional overhead this approach was intended to reduce.

In the following we give a more detailed description of the created pipeline versions.

3.1 First pipeline approach

3.2 Generator Generalisation and Localised Iteration Approach

3.3 Fast Pipeline Approach

Conducting an experiment with the two previous versions of the pipeline involves concatenated for-loop iterations over the respective parameter sets. At each step of an iteration the data is passed between the active class instances of the pipeline and subsequently stored as an instance attribute before transformation. There are two main sources of overhead that these approaches incur due to these operations. For one, every iteration involves the creation and subsequent destruction of large arrays in memory, and while `numpy`'s C implementation guarantees efficient calculations on arrays of fixed size, creation of these arrays incurs significant overhead as large contiguous blocks of memory have to be reserved and released. Secondly, at each step of an iteration the data is passed, i.e. copied, between the individual pipeline components, which again incurs the overhead due to creation and garbage collection.

The key idea of the last pipeline approach is to reduce the impact of these sources of overhead by creating large numpy arrays of adequate dimensions once, giving each component of the pipeline a reference to the data instead of a copy, and applying the algorithms directly to sections of these large arrays.

4 Metrics

¡¡¡¡¡¡¡ HEAD

4.1 Accuracy rate

If benefit and harm are weighted equally, the odds of the threshold is 1:1, or a threshold probability of 50%. This cutoff is by default considered in the calculation of the error rate, which is defined as (FN + FP)/N. The complement is the accuracy rate: (TN + TP)/N. Often FN classifications are more important than FP classifications, which makes the accuracy rate not a sensible indicator of clinical usefulness. Other disadvantages include that the accuracy rate by definition is high for a frequent or infrequent outcome (i.e. imbalanced data).

The accuracy rate is usually calculated at the simplistic cutoff of 50%, but can also be calculated at clinically defendable thresholds. The harm-to-benefit ratio that underpins the choice of the cutoff should then be used to calculate a weighted accuracy, or its complement, the weighted error rate. We can express the TN classifications in units of the TP classifications, such that the weighted accuracy is calculated as (TP + w TN)/n.

The improvement that is obtained by making decisions based on predictions from the model is the difference between the weighted accuracy obtained with the model versus the weighted accuracy of the default policy.

=======

¿¿¿¿¿¿¿ baseline

4.2 Receiver Operator Characteristic (ROC)

Consider a binary classification problem where we have a classifier that outputs a continuous score s(X) for an instance X. We then set a threshold τ to decide the predicted class. If $s(X) > \tau$, we predict the positive class; otherwise, we predict the negative class.

$$TPR(\tau) = \mathbb{P}(s(X) > \tau | Y = 1)$$
$$FPR(\tau) = \mathbb{P}(s(X) > \tau | Y = 0)$$

Here, Y is the true class of an instance. TPR is the probability that the classifier ranks a randomly chosen positive instance higher than a randomly chosen negative instance. Similarly, FPR is the probability that the classifier ranks a randomly chosen negative instance higher than a randomly chosen positive instance.

The ROC curve plots $TPR(\tau)$ against $FPR(\tau)$ for all possible thresholds τ, producing a curve that ranges from $(0, 0)$ to $(1, 1)$. We can interpret a ROC plot as plotting the path of a function

$$f : \mathbb{R} \to [0, 1]^2, \quad \tau \mapsto (TPR(\tau), FPR(\tau))$$

The Area Under the ROC Curve (AUC) then provides a single scalar value that represents the expected performance of the classifier. An AUC of 1 indicates a perfect classifier, while an AUC of 0.5 indicates a classifier that performs no better than random chance.

AUC can also be interpreted in terms of the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance, assuming that one positive and one negative instance are chosen at random. ¡¡¡¡¡¡¡ HEAD The area under the curve (AUC) can be interpreted as the probability that a patient with the outcome is given a higher probability of the outcome by the model than a randomly chosen patient without the outcome Some may consider the interpretation of AUC as straightforward. Others may object that we consider a pair of subjects, one with and one without the outcome, and that such conditioning is a rather artificial situation. Statistically, this conditioning on a pair of patients is attractive, since it makes the area independent of the incidence of the outcome (or event rate), in contrast to R2 or the Brier score, for example. The AUC is a rank order statistic. As a rank order statistic, it is insensitive to errors in calibration such as differences in average outcome. Confidence intervals for the AUC (or c statistic) can be calculated with various methods. Standard asymptotic methods may be problematic, especially when sensitivity or specificity is close to 0% or 100%. Bootstrap resampling is a good choice for many situations.

4.3 Net Benefit

The choice of risk threshold implicitly conveys the adopted relative misclassification costs. It can be derived that the odds of the risk threshold equal the harm-to-benefit ratio, which is the harm of a false positive divided by the benefit of a true positive. For example, if a risk threshold of 20% is used, the odds are 1 to 4. Therefore, a 20% risk threshold assumes that the harm of a false positive is one-quarter of the benefit of a true positive or that 1 true positive is worth 4 false positives: A clinician might express this in terms such as “I would not do more than five biopsies to find one cancer.” Hence, when applying a model to a set of patients, we can correct the number of true positives (TP) for the number of false positive (FP) using the odds w of the risk threshold t:

$$\text{TP} - w\text{FP} = \text{TP} - \frac{\tau}{1-\tau}\text{FP}$$

When dividing by the total sample size N, the Net Benefit is obtained

$$\text{NB} = \frac{1}{N}(\text{TP} - w\text{FP}) = \frac{1}{N}(\text{TP} - \frac{\tau}{1-\tau}\text{FP})$$

The Net Benefit of treat-none is always 0, whereas the Net Benefit of treat-all is positive for risk thresholds below the event rate and negative for risk thresholds above the event rate.

4.4 Net Benefit and Risk Threshold

Suppose r.v. Y with outcome diseased D or healthy $\neg D$. Let $c_{TP}, c_{FP}, c_{TN}, c_{FN}$ then expected cost of predicting $\neg D$ is

$$\mathbb{P}(Y=1)c_{FN} + \mathbb{P}(Y=0)c_{TN}$$

and expected cost for predicting D is

$$\mathbb{P}(Y=1)c_{TP} + \mathbb{P}(Y=0)c_{FP}$$

If we write $T = \mathbb{P}(Y=1)$ we can rewrite to

$$Tc_{FN} + (1-T)c_{TN}$$

and

$$Tc_{TP} + (1-T)c_{FP}$$

One is indifferent about treatment if both expected costs are equal.

$$Tc_{FN} + (1-T)c_{TN} = Tc_{TP} + (1-T)c_{FP}$$
$$\Leftrightarrow$$
$$T = \frac{c_{TN} - c_{FP}}{(c_{TN} - c_{FP}) + (c_{TP} - c_{FN})}$$

It can also be rearranged in a different way

$$Tc_{FN} + (1-T)c_{TN} = Tc_{TP} + (1-T)c_{FP}$$
$$\Leftrightarrow$$
$$\frac{T}{1-T} = \frac{c_{TN} - c_{FP}}{c_{TP} - c_{FN}}$$

In the first group, the costs relate to undertreatment (false-negative classifications) The costs of these false-negative classifications c_{FN} should be compared to the costs of true-positive classifications c_{TP}. The difference $c_{TP} - c_{FN}$ is the net benefit of treating all who have the disease compared to treating none of them. Suppose you have a treatment that causes lots of damage as well as curing the disease. Then this difference might be low, i.e. treating everyone with the dangerous treatment does not give much better outcome than simply not treating anyone. Suppose on the other hand you have a devastating disease like Polio and a low cost treatment like a Polio-vaccine, then that difference will be strongly positive.

In the second group, relevant costs are for those without the event if not treated, who are treated ("overtreated"). The costs of these false-positive classifications (c_{FP}) should be compared to the costs of true-negative classifications (c_{TN}) while $c_{TN} - c_{FP}$ is the harm of treating all who don't have the disease compared to the benefit of not treating any of them. E.g. the cost of not treating anyone without the disease might be 0 but the cost of treating them might be high. Then this value is strongly negative. On the other hand suppose again a low impact vaccine that barely does harm, then that difference may be small negative.

Odds (cutoff) = Harm/Benefit

The ratio in case of e.g. the polio vaccine could be strongly in favour of treating everyone (small cost for those without, high benefit for those with disease).

4.5 Calibration

Another key property of a prediction model is calibration, i.e., the agreement between observed outcomes and predictions.

$$TPR(\tau) = \mathbb{P}(s(X) > \tau | Y = 1)$$

4.6 Mixed

$$\begin{aligned} TPR(\tau) &= \mathbb{P}(s(X) > \tau | Y = 1) \\ FPR(\tau) &= \mathbb{P}(s(X) > \tau | Y = 0) \end{aligned}$$

=======

¿¿¿¿¿¿¿ baseline

4.7 Hypothesis testing

The **Hypothesis-T-Test** class is designed to facilitate hypothesis testing, specifically t-tests, on datasets stored in CSV format. It enables us to compare the effectiveness of different combinations of data balancing and classification methods in terms of key performance measures. The t-test is an appropriate choice in this case, where we want to apply multiple pairwise comparisons of continuous variables (the target measures). Hereby is an overview of its functionality.

The class constructor accepts several essential parameters:

- **target list**: a list of target performance measures;
- **column bal**: the name of the file column addressing the data balancing method
- **column clas**: the name of the file column addressing the classifier method
- **test combination balancers**: a list of tuples, each containing a combination of two balancing methods for comparison
- **test combination classifiers**: a list of tuples, each containing a combination of two classifier methods for comparison

- **alpha**: the test significance level, set by default to 0.05, a typical choice in hypothesis testing.

The core functionality of the class resides in the **perform-t-test** method, which consists of the following steps:

- Initialization of an empty list, **results**, to store the outcomes of the t-tests.
- For each target performance measure, the class iterates through the provided combinations of balancing methods and classifier methods.
- The data is filtered based on the selected balancing or classifier method, creating two distinct groups.
- A t-test is conducted on the two groups, producing statistical values such as the t-statistic and p-value.
- The mean values of the target performance measure are calculated for each group.
- The test results, including the target performance measure, compared methods, mean values, t-statistic, and p-value, are organized into dictionaries and appended to the **results** list.
- After completing all t-tests, the results are compiled into a pandas DataFrame and saved to a CSV file. This file serves as a comprehensive record of the statistical comparisons.

4.8 Linear regression Analysis

Another crucial step in our pipeline is to thoroughly explore the relationships between our target measures and the parameters of interest that we allowed to vary in the preceding stages of our project. To achieve this, we've developed the **Linear Regression Analysis** class. This class empowers us to conduct in-depth investigations into how specific predictor variables influence our chosen target metrics. In the following, we will delve into the functionality of this class and showcase its applications in our data analysis process. After loading a dataset from a specified CSV file and storing it in a pandas DataFrame, the data need to be set up for regression analysis. This is done through the **prepare data** method, which takes two sets of input variables: categorical and continuous regressors. The categorical variables are encoded into binary format and combined with the continuous regressors to create the input matrix **X**. The actual analysis is then implemented in the function **perform linear regression**, which takes the following inputs:

- **target**: the target metric that we want to predict
- **regressors**: a list of predictor variables used in the regression analysis

The function performs the following steps:

- Splits the dataset into training and testing sets.
- Fits a linear regression model to the training data.
- Predicts the target metric for the test data.
- Calculates various regression metrics, including mean squared error, mean absolute error, coefficient values, coefficient of determination (R-squared), F-value, and p-value.
- Stores the results in a pandas DataFrame and returns it.

Finally, the method **plot target vs regressors** repeats the analysis and loops through each regressor in the regressors list. For each regressor, the corresponding predicted values **y values** are calculated as a sum of the intercept and the product between the regression coefficients and the data. Next, a scatter plot is produced to graphically observe the relationship.

References

[1] Nitesh, V., Chawla., Kevin, W., Bowyer., Lawrence, O., Hall., W., Philip, Kegelmeyer. (2002). SMOTE: synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16(1):321-357. doi: 10.1613/JAIR.953

[2] Haibo He, Yang Bai, E. A. Garcia and Shutao Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, 2008, pp. 1322-1328, doi: 10.1109/IJCNN.2008.4633969.