CLASS-WEIGHTED EVALUATION METRICS FOR IMBALANCED DATA CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Class distribution skews in imbalanced datasets may lead to models with prediction bias towards majority classes, making fair assessment of classifiers a challenging task. Metrics such as Balanced Accuracy are commonly used to evaluate a classifier's prediction performance under such scenarios. However, these metrics fall short when classes vary in importance. In this paper, we propose a simple and general-purpose evaluation framework for imbalanced data classification that is sensitive to arbitrary skews in class cardinalities and importances. Experiments with several state-of-the-art classifiers tested on real-world datasets from three different domains show the effectiveness of our framework – not only in evaluating and ranking classifiers, but also training them.

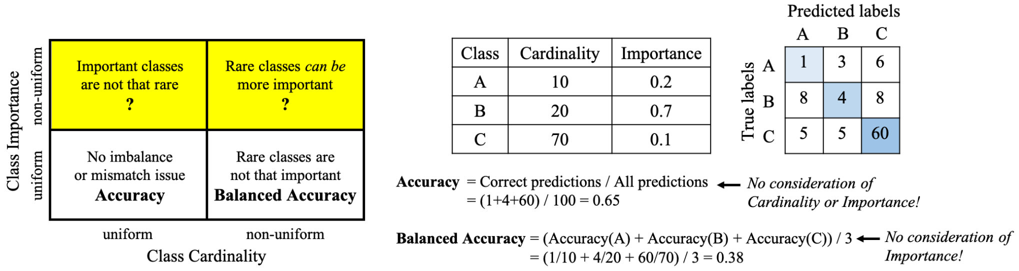
1 INTRODUCTION

For a broad range of machine learning (ML) tasks, predictive modeling in the presence of *imbalanced datasets* – those with severe distribution skews – has been a long-standing problem (He & Garcia, 2009; Sun et al., 2009; He & Ma, 2013; Branco et al., 2016; Hilario et al., 2018; Johnson & Khoshgoftaar, 2019). Imbalanced training datasets lead to models with prediction bias towards majority classes, which in turn results in misclassification of the underrepresented ones. Yet, those minority classes often are the ones that correspond to the most important events of interest (e.g., errors in system logs (Zhu et al., 2019), infected patients in medical diagnosis (Cohen et al., 2006), fraud in financial transactions (Makki et al., 2019)). While there is often an inverse correlation between the class cardinalities and their importance (i.e., rare classes are more important than others), the core problem here is the mismatch between the way these two distributions are skewed: the i^{th} most common class is not necessarily the i^{th} most important class (see Figure 1a for an illustration). In fact, rarity is one of many potential criteria that can determine the importance of a class, which is usually positively correlated with the costs or risks involved in its misprediction. Ignoring these criteria when dealing with imbalanced data classification may have detrimental consequences.

Consider automatic classification of messages in system event logs as an example (Zhu et al., 2019). An *event log* is a temporal sequence of messages that have transpired for a given software system (e.g., operating systems, cyber-physical systems) over a certain time period. Event logs are particularly useful after a system has been deployed, as they can provide the DevOps teams with insights about errors outside of the testing environment, thereby enabling them to debug and improve the system quality. There is typically an inverse correlation between the stability/maturity of a system and the frequency of the errors it produces in its event logs. Furthermore, the message types that appear least frequently in an event log are usually the ones with the greatest importance. A concrete example of this was a rare anomaly in Uber's self-driving car that led to the death of a pedestrian, since the system flagged it as a false positive in its logs (Efrati, 2018). If this event had not been misclassified and dismissed by the system, the pedestrian death in Arizona may have been avoided.

A plethora of approaches have been proposed for building balanced classifiers (Sun et al., 2009; Branco et al., 2016). A fundamental issue that still remains an open challenge is the lack of a generally-accepted methodology for measuring classification performance. The traditional metrics, which are designed to evaluate average case performance (e.g., *Accuracy*) are not capable of correctly assessing the results in presence of arbitrary skew mismatches between class cardinalities and importances. On the other hand, metrics specifically proposed for imbalanced learning are either domain-specific, do not easily generalize beyond two classes, or can not support varying class importance (e.g., *Balanced Accuracy*) (Japkowicz, 2013).

Let us illustrate the problem with the simple example in Figure 1b. The test dataset consists of 100 data items from 3 classes (A, B, C). The greatest majority of the items belong to class C (70), but


Class Importance
non-uniform
uniform
Important classes are not that rare ?
Rare classes *can be* more important ?
No imbalance or mismatch issue **Accuracy**
Rare classes are not that important **Balanced Accuracy**
uniform
non-uniform
Class Cardinality
Predicted labels
True labels

Class	Cardinality	Importance
A	10	0.2
B	20	0.7
C	70	0.1

	A	B	C
A	1	3	6
B	8	4	8
C	5	5	60

Accuracy = Correct predictions / All predictions ← *No consideration of Cardinality or Importance!*
= (1+4+60) / 100 = 0.65

Balanced Accuracy = (Accuracy(A) + Accuracy(B) + Accuracy(C)) / 3 ← *No consideration of Importance!*
= (1/10 + 4/20 + 60/70) / 3 = 0.38

(a) Skew in distributions

(b) Example

Figure 1: Skew in distributions of Class Cardinality or Class Importance, and the potential mismatch between them render existing accuracy metrics unusable in general multi-class prediction problems.

class B (20) has the greatest importance (0.7). In other words, Cardinality and Importance are both non-uniform and in favor of different classes (i.e., representing the top-right quadrant of Figure 1a). The confusion matrix on the right shows the results from a classifier run against this test dataset. Unsurprisingly, the classifier performed the best for the majority class C (60/70 correct predictions). When evaluated using the traditional *Accuracy* metric, neither Class Cardinality nor Class Importance is taken into account. If *Balanced Accuracy* is used instead, we observe the degrading impact of the Class Cardinality skew ($0.38 < 0.65$), but Class Importance is still not accounted for. This example demonstrates the need for a new evaluation approach that is sensitive to both Cardinality and Importance skew, as well as any arbitrary correlations between them. This is especially critical for ensuring a fair comparative assessment across multiple classifiers or problem instances.

Our goal in this paper is to design an evaluation framework for imbalanced data classification, which can be reliably used to measure, compare, train, and tune classifier performance in a way that is sensitive to non-uniform class importance. We identify two key design principles for such a framework:

- *Simplicity:* It should be *intuitive* and *easy* to use and interpret.
- *Generality:* It should be general-purpose, i.e., (i) *extensible* to an arbitrary number of classes and (ii) *customizable* to any application domain.

To meet the first goal, we focus on scalar metrics such as *Accuracy* (as opposed to graphical metrics such as ROC curves), as they are simpler, more commonly used, and scale well with increasing numbers of classes and models. To meet the second goal, we target the more general n-ary classification problems (as opposed to binary), as well as providing the capability to flexibly adjust class weights to capture non-uniform importance criteria that may vary across application domains. Note that we primarily focus on *Accuracy* as our base scalar metric in this paper, as it is seen as the de facto metric for classification problems (Sci). However, our framework is general enough to be extended to other scalar metrics, such as *Precision* and *Recall*. Similarly, while we deeply examine three use cases (log parsing, sentiment analysis, URL classification), our framework in principle is generally applicable to any domain with imbalanced class and importance distributions.

We first provide a brief overview of related work in Section 2. Section 3 presents our new, class-weighted evaluation framework. In Section 4, we show the practical utility of our framework by applying it over: (i) three log parsing systems (Drain (He et al., 2017), MoLFI (Messaoudi et al., 2018), Spell (Du & Li, 2016; 2018)) using four real-world benchmarks (Zhu et al., 2019); (ii) a variety of deep learning models developed for sentiment analysis on a customer reviews dataset from Amazon (Ni et al., 2019); and (iii) an industrial use case for URL classification with real classifiers and datasets from four cyber-security companies. Finally, we conclude in Section 5.

2 Related Work

Imbalanced Data Classification. Imbalanced data is prevalent in almost every domain (Cohen et al., 2006; Batuwita & Palade, 2012; Makki et al., 2019). The growing adoption of ML models in diverse application domains has led to a surge in imbalanced data classification research (He & Garcia, 2009; Sun et al., 2009; He & Ma, 2013; Branco et al., 2016; Hilario et al., 2018; Johnson & Khoshgoftaar, 2019). While the techniques widely vary, they fall under four basic categories: pre-processing training data to establish balance via sampling techniques (Estabrooks et al., 2004; Blaszczynski & Stefanowski, 2015), building custom learning techniques for imbalanced training

Table 1: Notation

Notation	Description
N	total number of data items
C	total number of data item classes
M	number of importance criteria
n_i	true number of data items in class i
p_i	correctly predicted number of data items in class i
f_i	relative frequency of class i
w_i	relative weight of class i
u_i	relative user-defined importance of class i
r_i	relative rarity of class i
$m_{i,j}$	relative weight of class i for importance criteria j
$Accuracy_i$	Accuracy of class i

data (Joshi et al., 2001; Castro & de Pádua Braga, 2013), post-processing predictions from an imbalanced model (Maloof, 2003), and their hybrids (Estabrooks & Japkowicz, 2001). In this paper, we do not propose a new imbalanced learning technique, but a general-purpose performance evaluation framework that could be used in the training and/or testing of models for any technique. Section 4 demonstrates the practical utility of our framework for a variety of real ML use cases.

Evaluation Metrics. Traditional metrics for evaluating prediction performance such as Accuracy, Sensitivity/Specificity (and their combination G-mean), Precision/Recall (and their combination F-Score) were not designed with imbalanced data issues in mind (Japkowicz, 2013). In fact, most of these were originally intended for binary classification problems. To extend them to more than 2 classes, macro-averaging (i.e., arithmetic mean over individual class measurements) is used. Macro-averaging treats classes equally (Branco et al., 2016). Balanced Accuracy is a popular averaging-based approach. There are also probabilistic evaluation approaches that extend Balanced Accuracy with Bayesian inference techniques for both binary and multi-class problems (Brodersen et al., 2010; Carrillo et al., 2014). Close to our work, Cohen et al. (2006) introduced the notion of class weights, yet in the specific context of Sensitivity/Specificity for binary classification in the medical domain. Similarly, Batuwita & Palade (2012) proposed extensions to G-mean for the bio-informatics domain. In addition to these scalar (a.k.a., threshold) metrics, graphical (a.k.a., ranking) evaluation methods such as Receiver Operating Characteristic (ROC) curves or Precision-Recall (PR) curves (and the Area Under the Curve (AUC) for such curves) as well as their extensions to imbalanced data / multi-class problems were also investigated (Weng & Poon, 2008; Japkowicz, 2013). While these methods provide more detailed insights into the operational space of classifiers as a whole, they do not easily scale with use in problems with a large number of classes (Branco et al., 2016).

3 Class-Weighted Evaluation Framework

In this section, we present our new evaluation framework for multi-class learning problems in presence of arbitrary skews among class distributions and/or importances. Our framework builds on and extends commonly used scalar / threshold metrics such as $Accuracy$. These metrics were originally designed for binary classification problems, where there is typically more emphasis on one class (the positive class, e.g., anomalies). To adopt them to multi-class problems where there is no such single-class emphasis, each class' metric can be computed separately and then an overall aggregation (i.e., arithmetic mean) can be performed. For example, $Accuracy$ has been extended to $BalancedAccuracy$ by following this approach. In our framework, we follow a similar aggregation strategy, however, we do it in a more generalized way that allows custom class weights to capture class importance. Furthermore, these class weights can be based on any importance criteria such as rarity, cost, risk, expected benefits, and possibly a hybrid of multiple such criteria. Therefore, it is critical to provide a flexible formulation that allows users or domain experts to adjust the weights as needed by their problem instance. In what follows, we present our new class-weighted evaluation framework in a top-down fashion. Using the basic notation summarized in Table 1, we first formulate the general framework, and then we describe how this framework can be customized to different importance criteria scenarios by specializing the weights in a principled manner. For ease of exposition, we first focus on $Accuracy$ as the underlying performance metric, and then we discuss how our approach can be adopted to other similar metrics. Finally, we end this section with a brief discussion of how our framework can be used in model training.

3.1 Weighted Balanced Accuracy (WBA)

Suppose we are given a test dataset with N data items in it, each of which belongs to one of C distinct classes. Furthermore, each class i contains n_i of the data items in this dataset. Thus:

$$N = \sum_{i=1}^{C} n_i \tag{1}$$

The relative frequency of each class i in the whole dataset is:

$$f_i = \frac{n_i}{N} \tag{2}$$

Assume a classifier that makes a prediction about the class label of each data item in the test dataset, and correctly predicts p_i out of n_i labels for a given class i, where $p_i \leq n_i$. Then, the total number of correct predictions out of all the predictions gives us the overall $Accuracy$ of the classifier:

$$Accuracy = \frac{\sum_{i=1}^{C} p_i}{N} \tag{3}$$

The classifier's $Accuracy_i$ for a given class i (a.k.a., per-class $Recall$ score) can be computed as:

$$Accuracy_i = \frac{p_i}{n_i} \tag{4}$$

$BalancedAccuracy$ is the macro-average of $Accuracy_i$ over all classes in the dataset:

$$BalancedAccuracy = \frac{1}{C} \times \sum_{i=1}^{C} Accuracy_i \tag{5}$$

The above formulation represents the state of the art in how prediction accuracy is evaluated for multi-class classifiers in presence of imbalanced datasets (i.e., those where f_i are not even). While for balanced datasets (i.e., $\forall i, n_i = N/C$ and $f_i = 1/C$) $BalancedAccuracy = Accuracy$, for imbalanced datasets, $BalancedAccuracy$ ensures that the prediction accuracy is not inflated due to high-frequency classes' results dominating over the others'. $BalancedAccuracy$ works well as long as each class is of the same importance, since it is the simple arithmetic mean across per-class accuracy measurements (i.e., each class' accuracy contributes evenly to the overall accuracy). As we discussed in earlier sections, in many real-world classification problems, this assumption does not hold. Rather, classifiers must be rewarded higher scores for their prediction performance on more important classes. In order to capture this requirement, we generalize $BalancedAccuracy$ into $WeightedBalancedAccuracy$ by extending it with per-class importance weights w_i as follows:

$$WeightedBalancedAccuracy = \sum_{i=1}^{C} w_i \times Accuracy_i \tag{6}$$

This simple yet powerful extension enables us to capture both skews and imbalances in class cardinalities as well as importances (i.e., the complete design space in Figure 1a). This general formulation can support any importance criteria for weights as long as $0 \leq w_i \leq 1$ and $\sum_{i=1}^{C} w_i = 1$. In the following subsections, we present general use of WBA with custom weights, for other scalar metrics, as well as in improving model training.

3.2 Weight Customization

In a multi-class setting, not only may the classes carry different importance weights, but also the criteria of importance may vary from one problem or domain to another. We now discuss several types of criteria that we think are commonly seen in applications. This is not meant to be an exhaustive list, but it provides examples and templates that can be easily tailored to different problems.

Importance criteria = User-defined. This is the most general and flexible form of importance criteria. The application designer or domain expert specifies the relative weight of each class based on some application-specific criteria. As an example, the problem might be about classifying images of different types of objects in highway traffic and the user gives higher importance to correct recognition of certain objects of interest (e.g., pedestrians, bikes, animals, etc). We express user-defined relative weight of a class i with u_i, which is simply used as w_i in Equation 6 (i.e., $w_i = u_i$).

Importance criteria = Rarity. It is often the case that the rarer something is, the more noteworthy or valuable it is. In multi-class problems, this corresponds to the case when importance of a class i is inversely correlated with its relative frequency of occurrence (f_i) in the dataset. For example, in system log monitoring, log messages for more rarely occurring errors or exceptions (e.g., denial of service attack) are typically of higher importance. Therefore, a classifier that performs well on detecting such messages must be rewarded accordingly. In our framework, we capture rarity using weights that are based on normalized inverse class frequencies formulated as follows:

$$w_i = r_i = \frac{1}{f_i \times \sum_{j=1}^{C} \frac{1}{f_j}} \tag{7}$$

Multiple importance criteria. In some problems, importance of a class depends on multiple different criteria (e.g., both rarity and a user-defined criteria). To express class weights in such scenarios, we leverage techniques from multi-criteria decision making and multi-objective optimization (Triantaphyllou, 2000; Helff et al., 2016). One of the most basic methods is using normalized weighted sums based on composite weights (Helff et al., 2016). Composite weights can be computed either in additive or multiplicative form (Tofallis, 2014). The multiplicative approach tends to promote weight combinations that are uniformly higher across all criteria, and as such is found to be a more preferred approach in application scenarios similar to ours (Helff et al., 2016; Tofallis, 2014). While we present this approach here, in principle, other approaches from multi-criteria decision making theory could also be used within our framework. Given M different criteria with $m_{i,j}$ denoting the relative weight of class i for criteria j, we can compute the composite weight of a class i as follows:

$$w_i = \frac{\prod_{j=1}^{M} m_{i,j}}{\sum_{k=1}^{C} \prod_{j=1}^{M} m_{k,j}} \tag{8}$$

For example, if we had two criteria, rarity r and user-defined u with weights r_i and u_i for each class i, respectively, then the composite weight for class i would be $w_i = \frac{r_i \times u_i}{\sum_{j=1}^{C} r_j \times u_j}$.

Partially-defined importance criteria. One commonly expected scenario (especially in those classification problems where the number of classes C can be very large) is that not all of the class importance weights might be supplied by the user. For example, in a sentiment analysis use case, the user supplies the weights for all the negative classes, and leaves the others unspecified. Our framework can support such cases by automatically assigning weights to the unspecified classes. The default approach is to distribute the remaining portion of weights evenly across all unspecified classes: (1 - total weights specified) / (number of unspecified classes). If the user prefers an alternative approach (e.g., distribute the remainder based on rarity of the unspecified classes), this can also be easily supported by our framework.

3.3 Metric Customization

The class-weighted evaluation framework presented above focused on the popular *Accuracy* metric as the underlying metric of prediction performance. However, our framework follows a general structure based on the idea of *weighted macro-averaging with customizable weights*, which can essentially be used with any performance metric that can be computed on the basis of a class. For example, the macro-averaging approaches that are already being used for *Precision*, *Recall*, and *F-Score* could easily be extended with our customizable weighing approach by replacing *Accuracy* in our formulas with one of these metrics.

3.4 Model Training Improvement using Class Weights

The customized weights presented in Section 3.2 not only helps to give a user-preferred ranking via the proposed WBA metric, but also helps to improve model training. Recall that ML model training aims to minimize the loss between model predictions and ground truth labels. A common practice is to minimize the sum of all per-sample losses. Using $Loss_i$ to denote the total loss incurred by all samples within class i, the model loss to minimize in training would be: $\mathcal{L} = \sum_{i=1}^{C} Loss_i$.

By applying class importance weights w_i as suggested in Section 3.2, the loss value for important classes takes a larger portion in the final loss value to minimize. This enables the back-propagation process to focus more on optimizing the model parameters for the higher weighted classes, and thus improves their accuracy. The model loss incorporating class importance weights becomes:

$$\mathcal{L} = \sum_{i=1}^{C} w_i \times Loss_i \tag{9}$$

The advancement of popular deep learning frameworks have made this process rather straightforward to implement. For example, the TensorFlow API provides a parameter named `class_weight` which allows user to pass in a JSON data structure that specifies a weight value for each class label (TensorFlow). Similarly, the PyTorch API for cross entropy loss also provides a parameter to pass in class weights (PyTorch). These provide perfect interfaces for the usage of the importance weights in our WBA framework, both to improve a model's learning ability of the important classes as well as raising the overall WBA score of the classification outcomes.

4 Experimental Study

We now present an experimental analysis of our new framework for three application domains. Our goal is to demonstrate the value of WBA compared to existing metrics when evaluating ML models over real-world imbalanced data classification problems. As we will show, often times a traditional metric like $Accuracy$ or $BalancedAccuracy$ will make classifier A seem preferable to classifier B, when in reality classifier B is superior. In addition, we also provide an analysis of how WBA can positively impact, not only the testing of models, but also their training. Details of our experiments (incl. code, data, and examples) can be found in the supplementary file and in the appendix.

4.1 Use Case 1: Learned Log Parsing

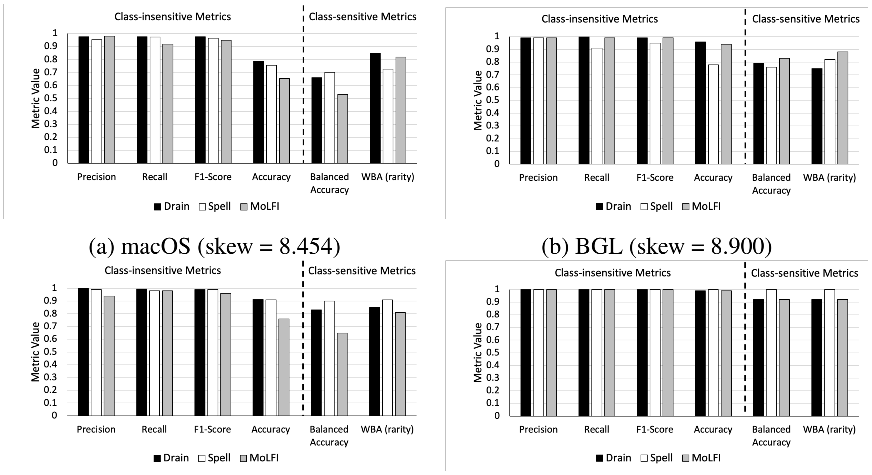
ML-based log parsers are tools that are designed to automatically learn the structure of event logs generated by hardware and software systems to properly categorize them into event classes (e.g., different error types). In our first study, we used WBA to evaluate 3 state-of-the-art log systems: Drain, Spell, and MoLFI (Du & Li, 2016; He et al., 2017; Messaoudi et al., 2018). We start by providing an abbreviated description of our experimental setup.

Log Parsing Systems. *Drain* is a rule-based, online log parsing system that encodes the parsing rules in a fixed-depth parse tree (He et al., 2017). It performs a pre-processing step for each new log message using regular expressions created by domain experts. *Spell*, like Drain, is also rule-based; it principally uses the *longest common subsequence* (LCS) to find new log message classes (Du & Li, 2016). Finally, *MoLFI* casts log parsing as a multi-objective optimization problem and provides a solution based on genetic programming (Messaoudi et al., 2018).

Datasets. We test each aforementioned log message classification system with four real-world datasets taken from a public benchmark (Zhu et al., 2019). Each dataset has 2000 log instances randomly sampled from a larger dataset. The *macOS* dataset contains raw log data generated by the macOS operating system (341 log classes, 237 *infrequent classes* (i.e., those that have fewer occurrences in the dataset than the average number of messages per class), and an average class frequency of 5). The *BlueGene/L (BGL)* dataset is a collection of logs from the BlueGene/L supercomputer system (120 log classes, 101 infrequent classes, and an average class frequency of 16). The *Android* dataset consists of logs from the Android mobile operating system (Zhu et al., 2019) (166 log classes, 127 infrequent classes, and an average class frequency of 16). Finally, the *HDFS* dataset consists of log data collected from the Hadoop Distributed File System (14 log classes, 8 infrequent classes, and an average class frequency of 142). Overall, the first three datasets are highly skewed in class frequencies, whereas the HDFS dataset is relatively much less skewed (see Appendix A.1).

Results. For Drain, Spell, and MoLFI, traditional metrics of $Precision$, $Recall$, *F1-Score*, and $Accuracy$ (named *Parsing Accuracy* in the original papers) were used for training and testing classification performance. None of these metrics are class-sensitive, while in log parsing, messages have in fact varying importance across the classes. The importance criteria is rarity: the more rare an error message is, the more important it is to correctly classify this message. To capture this, we configure the WBA to WBA_{rarity}, which automatically assigns weights to WBA based on the dataset classes' *inverse frequencies*, as described in Section 3.2. Then we evaluate the test results from the 3 parsers over 4 datasets using WBA_{rarity} and compare against traditional metrics in two categories: class-insensitive and class-sensitive, as shown in Figure 2.

WBA_{rarity} vs. Class-insensitive Metrics: The class-insensitive metrics (specifically, F1-Score and Accuracy) agree on how to rank the classification performance of the 3 parsers across all datasets (for macOS and Android, Drain > Spell > MoLFI; for BGL, Drain > MoLFI > Spell; for HDFS, all perform similarly). Since WBA_{rarity} is sensitive to classes' data distribution and importance skews, it makes a completely different judgement. Furthermore, it ranks the techniques differently for each dataset (Drain > MoLFI > Spell in macOS; MoLFI > Spell > Drain for BGL; Spell > Drain > MoLFI for Android; and for HDFS, Spell > Drain and MoLFI). The WBA_{rarity} ranking aligns with our observation on per-class accuracy of the methods. For example, on BGL dataset, although Spell



(a) macOS (skew = 8.454)

(b) BGL (skew = 8.900)

(c) Android (skew = 4.822)

(d) HDFS (skew = 0.202)

Figure 2: WBA vs. class-insensitive & class-sensitive metrics for log parsing: F1-Score & Accuracy agree in all. BA & WBA agree in (c) & (d) only. WBA disagrees with class-insensitive in all.

has the most mis-classified samples and in most classes (and thus the lowest overall Accuracy and Balanced Accuracy), it has very few mis-classified samples for rare classes, making its WBA_{rarity} higher than Drain. On the other hand, MoLFI performs the best on rare classes and thus has the best WBA_{rarity}. A discussion on per-class performance can be found in Appendix A.1. This result validates that WBA_{rarity} provides a more sensitive tool for assessing classification performance.

WBA_{rarity} vs. Balanced Accuracy (BA): As discussed earlier, BA is class-sensitive, but only to distribution imbalance. We can observe the difference between BA and WBA in Figure 2. In macOS and BGL, where the skew is the highest and rarity is more pronounced, the two metrics completely disagree in how they rank the parsers. In contrast, for Android and HDFS, where the skew is lower, there is an overall agreement, although the separation in metric values slightly differ. Of particular importance is the difference seen in Figure 2a. We observe that the best performing model is Spell when scored by BA, and Drain when scored by WBA_{rarity}. The reason for this difference is due to Spell's and Drain's differences in their ability to correctly classify infrequent classes, i.e., those that represent failures and errors that require the most immediate response.

4.2 Use Case 2: Sentiment Analysis

In social media and other user-facing domains like e-commerce sites, it is often useful to understand the view or feelings ("sentiments") associated with users' behavior or preferences. In the second part of our experimental study, we apply WBA in the context of such a sentiment analysis use case, which involves analyzing text-based product reviews from Amazon's e-commerce websites.

Dataset. The dataset consists of customer reviews and ratings of Amazon products (Ni et al., 2019). The task is to classify the reviews into 5 classes (with 1 being the lowest and 5 being the highest review rating a product can get), where ratings constitute the ground truth class labels. There is high class imbalance in this dataset (skew=2.140). As shown in the Frequency column of Table 2, Class 5 with the highest customer rating clearly dominates compared to the others. It is known that the distribution of customer review ratings is typically imbalanced and generally follow a J-shaped distribution (Mudambi & Schuff, 2010; Pavlou & Dimoka, 2006).

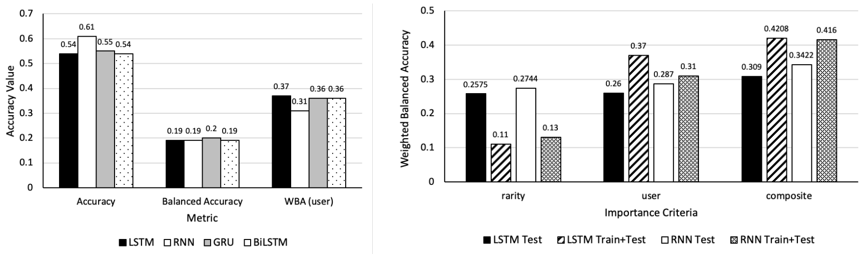
Sentiment Analysis Models. We compare 4 types of recurrent neural networks (RNN), all consisting of an embedding layer with pre-trained word embeddings from (Pennington et al., 2014) followed by a recurrent layer from PyTorch (Subramanian, 2018): RNN, LSTM, GRU, BiLSTM. The hidden state output from the last time step of these are passed to a fully-connected layer with input of 256 neurons and output from 5 neurons.

Results. For this use case, we first worked with a user-defined importance criteria borrowed from published studies suggesting that extreme review ratings (classes 1 and 5) carry more importance (Mudambi & Schuff, 2010; Pavlou & Dimoka, 2006). Thus, we set the weights as shown in Table 2 (shown as WBA(user) or user in Figure 3).

WBA vs. Other Accuracy Metrics: First, we compare WBA(user) with Accuracy and BalancedAccuracy (BA) when used as a metric for both training and testing of the 4 DNN models (Figure 3a). We make a few observations: (i) The class-insensitive Accuracy showcases the imbalance problem

Table 2: Amazon per-class breakdown: Frequencies are highly skewed (skew=2.140); $Accuracy_i$ in each model when both trained+tested with user-defined weights w_i (same weights as in Figure 3a).

Class	**Frequency (f_i)**	**Weights (w_i)**	**LSTM**	**RNN**	**GRU**	**BiLSTM**
1	0.092	0.7	0.19	0.04	0.16	0.17
2	0.052	0	0	0	0	0
3	0.075	0	0	0	0	0
4	0.142	0	0	0	0	0
5	0.639	0.3	0.81	0.96	0.84	0.83



(a) WBA vs. Other Metrics (Train+Test)

(b) WBA: Test vs. Train+Test

Figure 3: Amazon results

in classification, as it favors the RNN model which is heavily biased by the majority class (see $Accuracy_i$ for RNN in Table 2 where class 5 scores 0.96). (ii) The frequency-sensitive BA metric finds all models perform similarly. WBA(user), in contrast, identifies LSTM as the best model. Indeed, Table 2 confirms that LSTM performs relatively the best in predicting the most important class, class 1 (0.19 accuracy). Overall, we find that WBA is capable of capturing importance skews, even when the frequency skew can be high and biased towards less important classes.

Impact of WBA in Model Training: Next we explore the use of WBA not only in model evaluation, but also in training. We focus on two models (LSTM and RNN), and apply WBA only during testing vs. to both training (by extending loss functions of DNNs with weights as in Section 3.4) and testing. Intuitively, if a model is trained being aware of the importance weights, then it should also perform well when tested against the same criteria. To test this hypothesis, we repeated the experiment for 3 alternative importance criteria: (i) rarity ($w_1 = 0.209$, $w_2 = 0.368$, $w_3 = 0.255$, $w_4 = 0.136$, $w_5 = 0.030$), (ii) user-defined (i.e., with weights in Table 2), and (iii) composite of the two ($w_1 = 0.62$, $w_2 = w_3 = w_4 = 0$, $w_5 = 0.38$). In Figure 3b, we observe: (i) Except for rarity, WBA for both LSTM and RNN improves when integrated into model training. This verifies our intuition, and shows that WBA is a useful metric not only for evaluation, but also for training. (ii) When we zoom into rarity, we see that although class 2 is the most important, per-class accuracy for class 5 is much higher for both LSTM and RNN in the *Test-only* case, because both models are still trained heavily biased towards the majority class (5). (iii) Though rarity by itself is not useful in training, when combined with user importance, it visibly improves the WBA scores. This shows that our multi-criteria composition approach is capable of combining importance criteria as intended.

4.3 USE CASE 3: URL CLASSIFICATION

URL classification is a crucial task in the cyber-security industry for managing web traffic. Given a URL, the goal is to categorize the corresponding webpage into several well-defined classes such as benign (i.e., no harmful content), malware, phishing, NSFW (i.e., not safe for work), etc. Given the vast number of URLs available, accurately learning and evaluating a URL classifier can greatly help in automatically categorizing webpages without requiring manual labeling (Vallina et al., 2020).

Dataset. Our dataset contains URLs from several different types of categories: benign (e.g., *news*, *sports*), NSFW (e.g., *drugs*, *gambling*), and malicious (*phishing* and *malware*). These URLs were sampled from a variety of third-party sources so as to minimize potential data bias. Many of the phishing and malware URLs were sampled from *VirusTotal* (VirusTotal), which aggregates various cyber-security vendors' detections into a single site, while benign URLs were sampled from *Alexa's* Top Sites lists (Amazon), which rank websites according to their popularity. We are unable to reveal the exact URLs used in the analysis due to legal reasons, but we tried to ensure that our dataset is representative of the types of URLs encountered in real-world URL filtering scenarios.

URL Classification Services. We tested the performance of URL filtering products from 4 commercial companies, whose names had to be anonymized as A, B, C, D. Since each URL filtering product has its own unique taxonomy of categories, we created mappings from each service's category space to a single shared category space so that we can easily compare the results. Furthermore, since no

Table 3: Evaluating and ranking the URL classification services

Category	**#URLs**	**Rarity** w_i	**User** w_i	**Classification Accuracy**				**Ranking**
				Service A	Service B	Service C	Service D	
benign	16762	0.04	0.05	0.761	0.815	0.661	0.853	DBAC
NSFW	5276	0.14	0.05	0.965	0.804	0.533	0.767	ABDC
malware	1913	0.38	0.8	0.890	0.845	0.602	0.872	ADBC
phishing	1675	0.44	0.1	0.968	0.811	0.521	0.771	ABDC
Accuracy				**0.826**	0.815	0.621	**0.831**	DABC
Balanced Accuracy				**0.896**	0.819	0.579	**0.816**	ABDC
WBA_{rarity}				**0.929**	**0.823**	0.559	**0.812**	ABDC
WBA_{user}				**0.895**	**0.838**	0.593	**0.856**	ADBC

Table 4: Training and evaluating a URLNet model using WBA

Category	Dataset Statistics				Classification Accuracy		
	Train #URLs	**Test #URLs**	**Rarity** w_i	**User** w_i	**Train with no WBA**	**Train with rarity** w_i	**Train with user** w_i
benign	10000	6762	0.04	0.05	0.981	0.300	0.458
NSFW	3150	2126	0.14	0.15	**0**	**0.608**	0.418
malware	1143	770	0.38	0.45	**0.705**	**0.839**	**0.895**
phishing	1000	675	0.44	0.35	0.782	0.788	0.754
Test with Accuracy					0.745	0.435	0.502
Test with Balanced Accuracy					0.617	0.634	0.631
Test with WBA_{rarity}					**0.653**	**0.761**	NA
Test with WBA_{user}					**0.640**	NA	**0.752**

company had access to our test dataset sampled from various third-party sources, no company was able to gain an unfair advantage in this URL classification task.

Results. We submit the sampled URLs from each category to the above services, calculate their per-category accuracy, and compute a final accuracy score for competitive analysis.

Evaluating and Ranking the Classifiers: Table 3 shows our results for Companies A-D. For overall $Accuracy$, D is the best as it has the highest classification accuracy on benign class (i.e., the majority class). With $BalancedAccuracy$, we can see that A is the best, since it has higher accuracy for non-benign classes compared with other services, and this advantage is more evident with WBA_{rarity}, shown as a larger gap between their WBA_{rarity}values. B is better than D when using WBA_{rarity}, but if users consider malicious class to be the most important and apply a very high weight on it (e.g., 0.8), the resulted WBA_{user}metric would show that D is preferable over B.

Improving Model Training with WBA: In this experiment, we adopt our dataset to train a URLNet model (Le et al. (2018)) by applying various WBA weights as suggested in Section 3.2, to show its effectiveness in terms of improving corresponding WBA metric. As shown in Table 4, 60% of the dataset is used for training while the remaining 40% is for evaluation. Table 4 Classification Accuracy columns show the results under different settings. For vanilla training (without class weights), we get a high accuracy for the benign class but low accuracy for the others. The main reason is benign class has the most number of URLs in training data. In order to boost the accuracy for non-benign classes, we apply rarity weights. With this, the accuracy for all non-benign and rarer classes are improved (noticeably, from 0 to 0.608 for the NSFW category). Note that malware content is often considered more harmful among all non-benign categories. For the third experiment, we apply user-defined weights which assign higher weight on the malware class, successfully boosting its accuracy from 0.839 to 0.895. It is clear that applying corresponding class weights in training can significantly improve the related WBA accuracy on the test dataset. For instance, when applying rarity weights, WBA_{rarity}improves from 0.653 to 0.761; while $WBA_{user}$is improved from 0.640 to 0.752 if the same user-defined weights are used in training. This aligns with our observation that increasing the weights in concerned categories can notably improve the accuracy in those categories. As a comparison, neither the Accuracy nor the Balanced Accuracy measure does a good job in ranking the different training methods.

5 Conclusion

In this paper, we presented a simple yet general-purpose class-sensitive evaluation framework for imbalanced data classification. Our framework is designed to improve the grading of multi-class classifiers in domains where class importance is not evenly distributed. We provided a modular and extensible formulation that can be easily customized to different importance criteria and metrics. Experiments with three real-world use cases show the value of a metric based on our framework, Weighted Balanced Accuracy (WBA), over existing metrics – in not only evaluating the classifiers' test results more sensitively to importance criteria, but also training them so.

Ethics Statement

This paper introduces a measurement framework for training and evaluating classification models. The primary goal of this framework is to enable the creation of *more accurate models*, especially in presence of imbalances across class distributions and their importances. As with any technological tool, our work could be subject to misuse. In particular, unfair bias might be introduced to learned models by purposefully misconfiguring the class importance weights. To avoid such ethical risks, we urge the community to use our tool in a responsible manner.

Reproducibility Statement

This research is highly experimental in nature. We provide an appendix section as well as an additional file with supplementary material in order to document all of our experimental artifacts and how they can be used to reproduce the results we reported in this paper. These artifacts include all datasets used in the experiments, as well as code. Except for the URL classification use case, where the URLs cannot be released either because of our contract with VirusTotal, or due to being customer-sensitive information. Nevertheless, we have included the class labels and class predictions for each setting, and corresponding scripts to calculate various WBA metrics which can reproduce our results in the paper. In particular, the appendix section includes extra experimental results to validate our point, as well as the details for each experiment. Our supplementary material contains 3 distinct folders with source code and scripts for the 3 use cases in evaluation. Each folder contains a README file that instructs how to run the code and reproduce the results.

References

Scikit-learn Classification Metrics. `https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics`.

Amazon. Alexa Top Sites. `https://www.alexa.com/topsites`.

Rukshan Batuwita and Vasile Palade. Adjusted Geometric-mean: A Novel Performance Measure for Imbalanced Bioinformatics Datasets Learning. *Journal of Bioinformatics and Computational Biology*, 10(4), 2012.

Jerzy Blaszczynski and Jerzy Stefanowski. Neighbourhood Sampling in Bagging for Imbalanced Data. *Neurocomputing*, 150:529–542, 2015.

Paula Branco, Luís Torgo, and Rita P Ribeiro. A Survey of Predictive Modeling on Imbalanced Domains. *ACM Computing Surveys (CSUR)*, 49(2):1–50, 2016.

Kay Henning Brodersen, Cheng Soon Ong, Klaas Enno Stephan, and Joachim M Buhmann. The Balanced Accuracy and Its Posterior Distribution. In *20th International Conference on Pattern Recognition (ICPR)*, pp. 3121–3124, 2010.

Henry Carrillo, Kay H Brodersen, and José A Castellanos. Probabilistic Performance Evaluation for Multi-class Classification using the Posterior Balanced Accuracy. In *First Iberian Robotics Conference*, pp. 347–361, 2014.

Cristiano Leite Castro and Antônio de Pádua Braga. Novel Cost-Sensitive Approach to Improve the Multilayer Perceptron Performance on Imbalanced Data. *IEEE Transactions on Neural Networks and Learning Systems*, 24(6):888–899, 2013.

Gilles Cohen, Melanie Hilario ad Hugo Sax, Stephane Hugonnet, and Antoine Geissbuhler. Learning from Imbalanced Data in Surveillance of Nosocomial Infection. *Artificial Intelligence in Medicine*, 37(1):7–18, 2006.

Min Du and Feifei Li. Spell: Streaming Parsing of System Event Logs. In *IEEE International Conference on Data Mining (ICDM)*, pp. 859–864, 2016.

Min Du and Feifei Li. Spell: Online Streaming Parsing of Large Unstructured System Logs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 31(11):2213–2227, 2018.

Amir Efrati. Uber Finds Deadly Accident Likely Caused By Software Set to Ignore Objects On Road. *The Information*, 2018. (https://www.theinformation.com/articles/uber-finds-deadly-accident-likely-caused-by-software-set-to-ignore-objects-on-road).

Andrew Estabrooks and Nathalie Japkowicz. A Mixture-of-Experts Framework for Learning from Imbalanced Data Sets. In *International Conference on Advances in Intelligent Data Analysis (IDA)*, pp. 34–43, 2001.

Andrew Estabrooks, Taeho Jo, and Nathalie Japkowicz. A Multiple Resampling Method for Learning from Imbalanced Data Sets. *Computational Intelligence*, 20(1):18–36, 2004.

Haibo He and Edwardo A. Garcia. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 21(9):1263–1284, 2009.

Haibo He and Yunqian Ma (eds.). *Imbalanced Learning: Foundations, Algorithms, and Applications*. Wiley-IEEE Press, July 2013.

Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *IEEE International Conference on Web Services (ICWS)*, pp. 33–40, 2017.

Florian Helff, Le Gruenwald, and Laurent d'Orazio. Weighted Sum Model for Multi-Objective Query Optimization for Mobile-Cloud Database Environments. In *EDBT/ICDT International Workshop on Multi-Engine Data AnaLytics (MEDAL)*, 2016.

Alberto Fernandez Hilario, Salvador García Lopez, Mikel Galar, Ronaldo C. Prati, Bartosz Krawczyk, and Francisco Herrera. *Learning from Imbalanced Data Sets*. Springer, 2018.

Natalie Japkowicz. Assessment Metrics for Imbalanced Learning. In Haibo He and Yunqian Ma (eds.), *Imbalanced Learning: Foundations, Algorithms, and Applications*, pp. 187–206. John Wiley & Sons, 2013.

Justin M. Johnson and Taghi M. Khoshgoftaar. Survey on Deep Learning with Class Imbalance. *Journal of Big Data*, 6(27), 2019.

Mahesh V. Joshi, Vipin Kumar, and Ramesh C. Agarwal. Evaluating Boosting Algorithms to Classify Rare Classes: Comparison and Improvements. In *IEEE International Conference on Data Mining (ICDM)*, pp. 257–264, 2001.

Hung Le, Quang Pham, Doyen Sahoo, and Steven CH Hoi. Urlnet: Learning a url representation with deep learning for malicious url detection. *arXiv preprint arXiv:1802.03162*, 2018.

Sara Makki, Zainab Assaghir, Yehia Taher, Rafiqul Haque, Mohand-Saïd Hacid, and Hassan Zeineddine. An Experimental Study With Imbalanced Classification Approaches for Credit Card Fraud Detection. *IEEE Access*, 7:93010–93022, 2019.

Marcus A. Maloof. Learning When Data Sets are Imbalanced and When Costs are Unequal and Unknown. In *ICML Workshop on Learning from Imbalanced Data Sets*, 2003.

Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. A Search-based Approach for Accurate Identification of Log Message Formats. In *Proceedings of the 26th Conference on Program Comprehension*, pp. 167–177, 2018.

Susan M. Mudambi and David Schuff. Research Note: What Makes a Helpful Online Review? A Study of Customer Reviews on Amazon.com. *MIS Quarterly*, 34(1):185–200, 2010.

Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 188–197, 2019.

Paul A Pavlou and Angelika Dimoka. The Nature and Role of Feedback Text Comments in Online Marketplaces: Implications for Trust Building, Price Premiums, and Seller Differentiation. *Information Systems Research*, 17(4):392–414, 2006.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

PyTorch. PyTorch cross entropy loss API. `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`.

Vishnu Subramanian. *Deep Learning with PyTorch: A Practical Approach to Building Neural Network Models using PyTorch.* Packt Publishing, 2018.

Yanmin Sun, Andrew K. C. Wong, and Mohamed S. Kamel. Classification of Imbalanced Data: A Review. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(4):687–719, 2009.

TensorFlow. TensorFlow class weights. `https://www.tensorflow.org/guide/keras/train_and_evaluate#class_weights`.

Chris Tofallis. Add or Multiply? A Tutorial on Ranking and Choosing with Multiple Criteria. *INFORMS Transactions on Education*, 14(3):109–119, 2014.

Evangelos Triantaphyllou. *Multi-criteria Decision Making Methods: A Comparative Study*. Springer, 2000.

Pelayo Vallina, Victor Pochat, Álvaro Feal, Marius Paraschiv, Julien Gamba, Tim Burke, Oliver Hohlfeld, Juan Tapiador, and Narseo Vallina-Rodriguez. Mis-shapes, mistakes, misfits: An analysis of domain classification services. pp. 598–618, 10 2020. doi: 10.1145/3419394.3423660.

VirusTotal. VirusTotal Documentation. `https://support.virustotal.com/hc/en-us/categories/360000162878-Documentation`.

Cheng G. Weng and Josiah Poon. A New Evaluation Measure for Imbalanced Datasets. In *Proceedings of the 7th Australasian Data Mining Conference (AusDM'08)*, pp. 27–32, 2008.

Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. Tools and Benchmarks for Automated Log Parsing. In *IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 121–130, 2019.

A Appendix

In this appendix, we provide details for the experimental study, including data and code. For further information, please see the supplementary material.

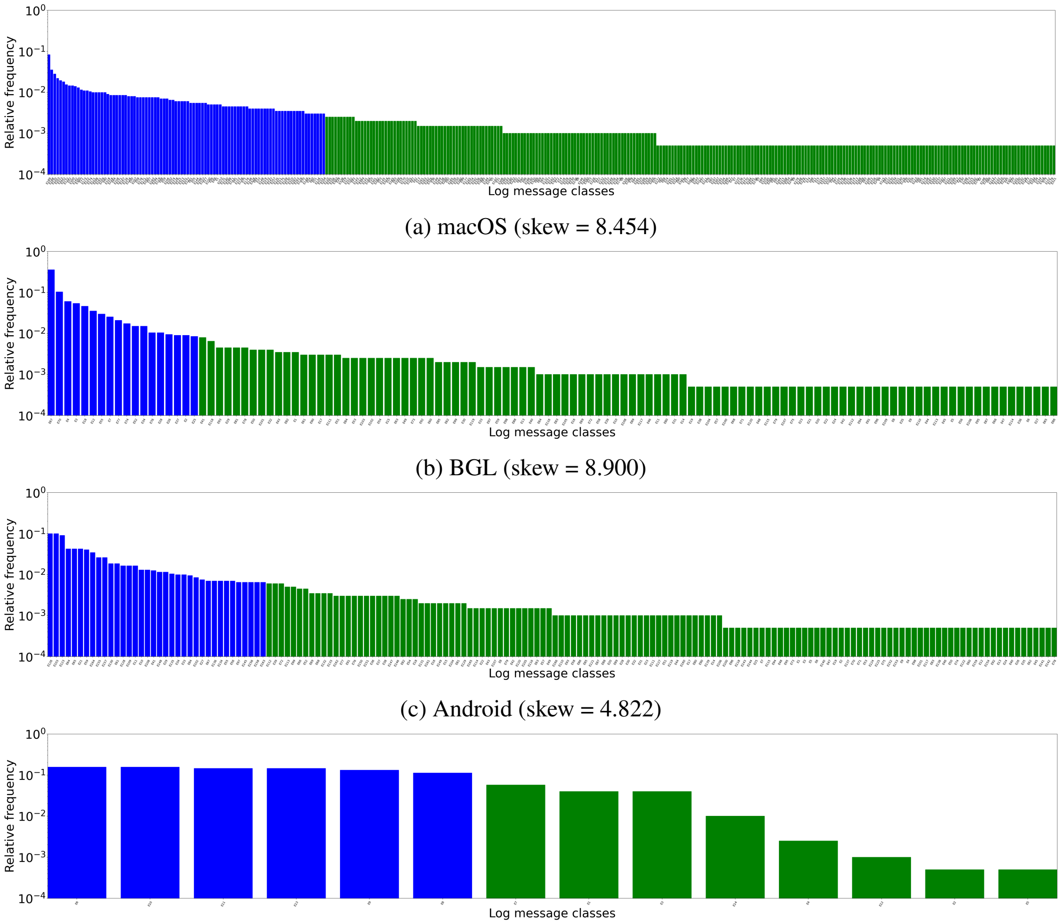
A.1 Details for Log Parsing Experiments

For the three log parsing techniques used in Section 4.1 (Drain, Spell, and MoLFI), we used the implementations provided by the LogPAI team:

`https://github.com/logpai/logparser/`

The four datasets used in these experiments (macOS, BGL, Android, and HDFS) came from the benchmarking data also provided by LogPAI:

`https://github.com/logpai/loghub/`



(a) macOS (skew = 8.454)

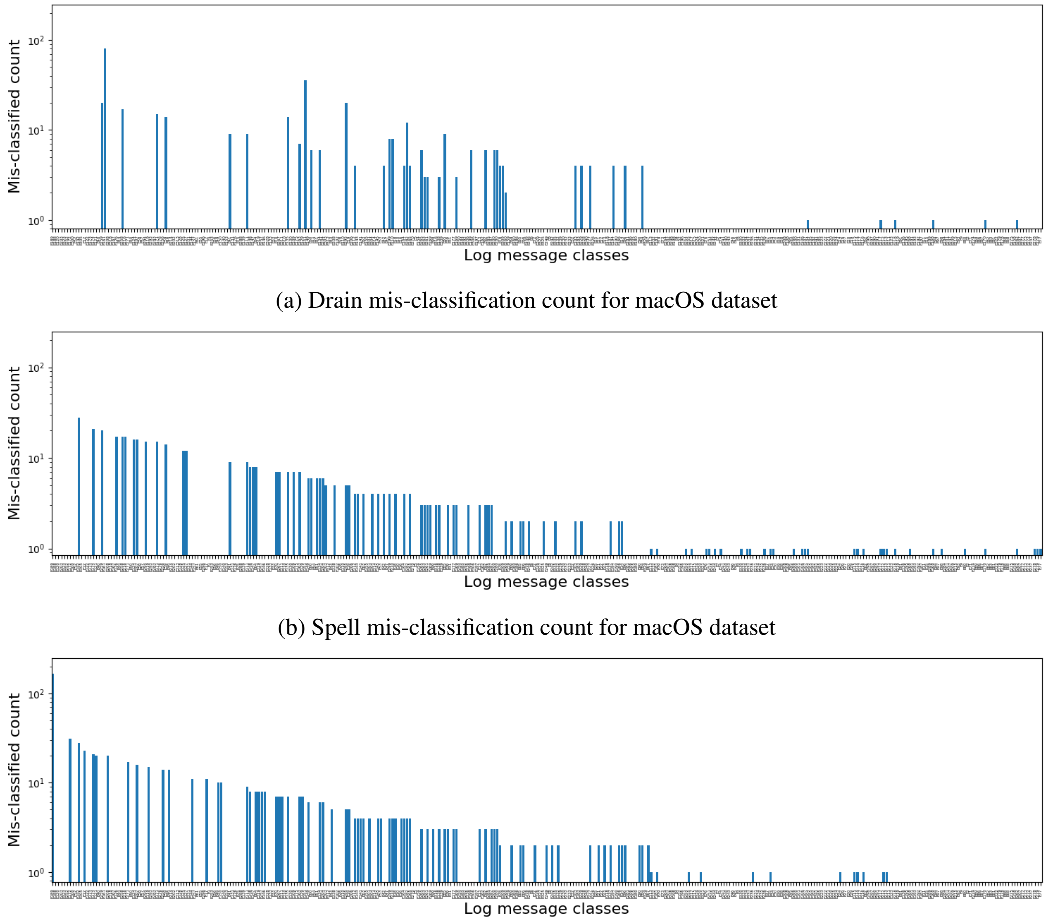
(b) BGL (skew = 8.900)

(c) Android (skew = 4.822)

(d) HDFS (skew = 0.202)

Figure 4: Histograms showing the relative frequencies of log parsing classes for the four experimental datasets: All graphs have their y-axes in log scale; green bars show the *infrequent classes*.

In Figure 4, we show the histograms for the four log datasets together with their skew values. As defined in the Microsoft Excel Documentation, "*Skewness characterizes the degree of asymmetry of a distribution around its mean. Positive skewness indicates a distribution with an asymmetric tail extending toward more positive values, while negative skewness indicates a distribution with an*



(a) Drain mis-classification count for macOS dataset

(b) Spell mis-classification count for macOS dataset

(c) MoLFI mis-classification count for macOS dataset

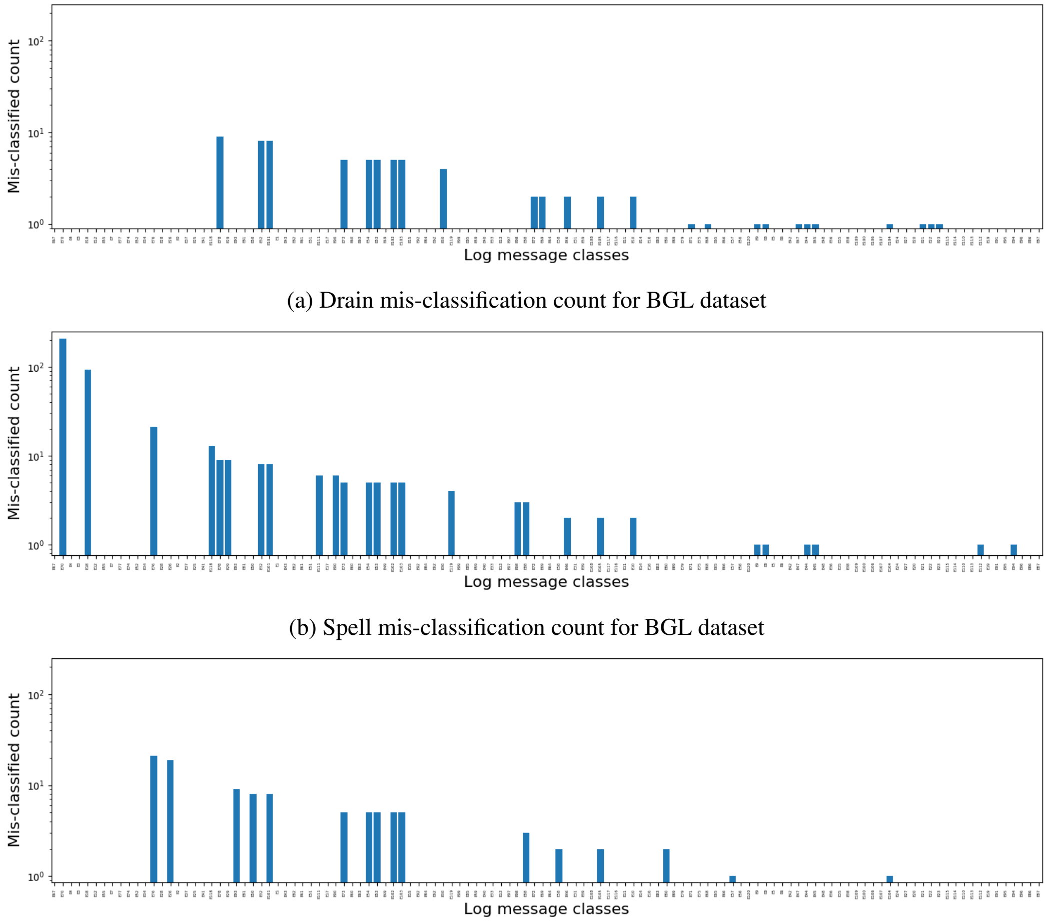
Figure 5: Misclassification count of three different algorithms on the macOS dataset. x-axis is ranked in descending order of class frequencies, as in Figure 4a. Drain performs best on rare classes (right side), which aligns with the WBA_{rarity} ranking shown in Figure 2a.

asymmetric tail extending toward more negative values." [1]. In our context, skew provides a good indication for the degree of imbalance in class cardinality distributions – the larger the skew, the larger the degree of class imbalance.

We also provide data files with class labels (true + predicted) and weights (based on rarity as importance criteria) used in generating the experimental data plotted in Figure 2 as part of our WBA-Evaluator tool implementation included in the supplementary material (can be found under the `WBA-Evaluator/examples/LogParsing/` directory).

A visualization of the per-class mis-classification count for each method on each dataset can be found from Figure 5 to Figure 7. The lower a bar is, the fewer samples being mis-classified in that category. Note that the categories on x axis are ordered in a descending order of class frequencies, i.e., in the same order of Figure 4. As a result, the lower the bars on the right side of each plot, the better the log parser is in classifying infrequent classes for that dataset, and thus the better the WBA_{rarity}score should be. As explained by the legends in each figure, the per-class accuracy in each plot aligns with WBA_{rarity} metrics in Figure 2.

[1] `https://support.microsoft.com/en-us/office/skew-function-bdf49d86-b1ef-4804-a046-28eaea69c9fa`



(a) Drain mis-classification count for BGL dataset

(b) Spell mis-classification count for BGL dataset

(c) MoLFI mis-classification count for BGL dataset

Figure 6: Misclassification count of three different algorithms on the BGL dataset. x-axis is ranked in descending order of class frequencies, as in Figure 4b. MoLFI performs best on rare classes (right side), which aligns with the WBA_{rarity} ranking shown in Figure 2b.

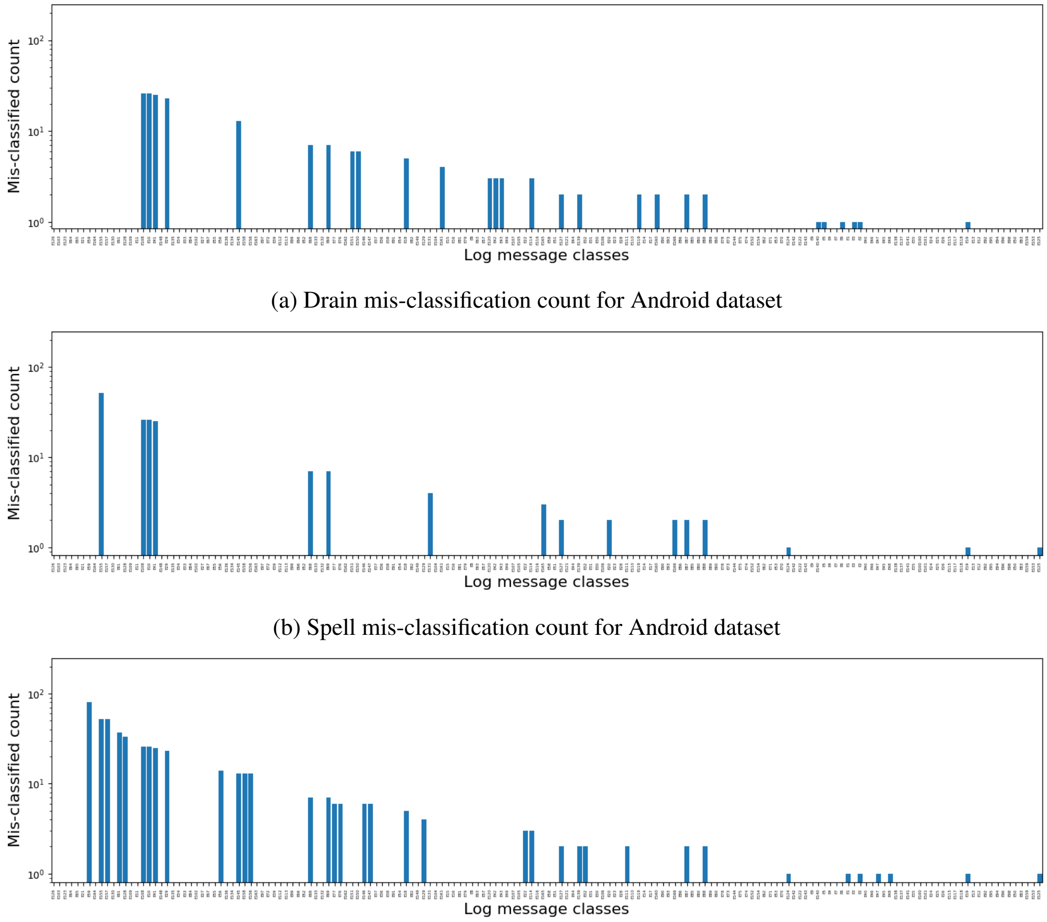
A.2 Details for Sentiment Analysis Experiments

For the sentiment analysis experiments of Section 4.2, we used a sample from the Amazon Customer Reviews dataset provided at:

`https://nijianmo.github.io/amazon/index.html`

In Figure 8, we show the histogram for the Amazon dataset. As described in Section 4.2, we implemented 4 RNN-based classifiers to experiment with this dataset. The code for these classifiers can be found in the supplementary material (under the `AmazonReviewsClassifier/src/` directory) along with a copy of the data (under the `AmazonReviewsClassifier/dataset/` directory).

We also provide the data files with class labels (true + predicted) and weights (user) used in generating the experimental data for LSTM results plotted in Figure 3 and Table 2 as an example. These can be found in our WBA-Evaluator tool implementation included in the supplementary material under the `WBA-Evaluator/examples/Amazon/` directory.



(a) Drain mis-classification count for Android dataset

(b) Spell mis-classification count for Android dataset

(c) MoLFI mis-classification count for Android dataset

Figure 7: Misclassification count of three different algorithms on the Android dataset. x-axis is ranked in descending order of class frequencies, as in Figure 4b. Spell performs best on rare classes (right side), which aligns with the WBA_{rarity} ranking shown in Figure 2c.

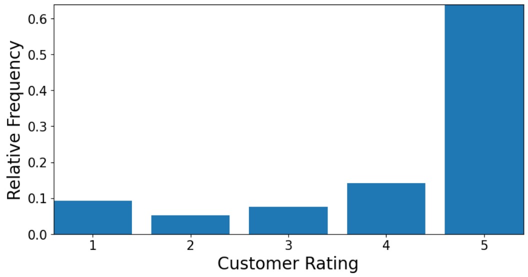


Figure 8: Histogram showing the relative frequencies of the five customer rating classes for the Amazon dataset (skew = 2.140).

A.3 Details for URL Classification Experiments

For the URL classification experiment in Section 4.3, and specifically for the experiment result in Table 4, our implementation is inherited from the URLNet work (Le et al. (2018)). In particular,

Table 5: Training and evaluating a URLNet model using WBA - 2 classes

	Dataset Statistics				Classification Accuracy		
Category	**Train #URLs**	**Test #URLs**	**Rarity w_i**	**User w_i**	**Train with no WBA**	**Train with rarity w_i**	**Train with user w_i**
Benign	10000	6762	0.091	0.2	0.994	0.902	0.950
Phishing	1000	675	0.909	0.8	0.859	0.942	0.861
Test with Accuracy					0.958	0.903	0.942
Test with Balanced Accuracy					0.927	0.922	0.906
Test with WBA_{rarity}					**0.854**	**0.904**	NA
Test with WBA_{user}					**0.868**	NA	**0.879**

Table 6: Training and evaluating a URLNet model using WBA - 3 classes

	Dataset Statistics				Classification Accuracy		
Category	**Train #URLs**	**Test #URLs**	**Rarity w_i**	**User w_i**	**Train with no WBA**	**Train with rarity w_i**	**Train with user w_i**
Benign	10000	6762	0.07	0.1	0.992	0.795	0.177
NSFW	3150	2126	0.22	0.5	**0**	**0.105**	**0.832**
Phishing	1000	675	0.71	0.4	0.855	0.933	0.901
Test with Accuracy					**0.762**	**0.652**	0.374
Test with Balanced Accuracy					0.616	0.611	0.637
Test with WBA_{rarity}					**0.673**	**0.738**	NA
Test with WBA_{user}					**0.441**	NA	**0.794**

we changed the URLNet source code to support multi-class classification (previously binary), and to apply various weights for model training. The URLNet open source repository is available at:

`https://github.com/Antimalweb/URLNet/`

Our implementation and modification can be found at the `WBA-Evaluator/WeightedURLNet/` directory.

In the main context, we show how the proposed WBA weights are able to improve training accuracy for underrepresented classes. Due to page limit, only the evaluation results on 4 classes are listed in Figure 4. Here we include the evaluation results for 2 classes and 3 classes respectively, to further express the effectiveness of WBA weights in improving model training. As shown in Table 5, by applying WBA_{rarity} weights and user-defined weights which all focus more on phishing category, the phishing accuracy is improved accordingly in both cases, with corresponding WBA metric improved as well. In contrast, neither overall accuracy nor Balanced Accuracy is able to show the accuracy improvement in the important class. Similarly, Table 6 shows the experiment results for 3 classes. It should be noted that a vanilla training leads to 0 accuracy on the newly added NSFW category, while another training with rarity weights brings some accuracy to this category, and applying a relatively high user weight for this category boosts its accuracy to over 80%.

A.4 The WBA-Evaluator Tool

In addition to details on our experimental study as described above, we also provide a copy of the WBA-Evaluator tool that implements our customizable, class-weighted evaluation framework described in Section 3. WBA-Evaluator is written in Python and can be found in the supplementary material along with a README that describes how it can be used. In a nutshell, WBA-Evaluator takes as input three files (true class labels, predicted class labels, class weights) and a number of configuration parameters in the form of commandline arguments, and then it generates accuracy scores (BA or WBA) as specified by these arguments. The WBA-Evaluator implementation comes with two subdirectories: `src/` contains the Python source code; `example/` contains all the input files (labels and weights) and scripts in the `scripts/` subfolder to run these. Please see the `README` file for more details. Using this tool, results reported in the paper can be reproduced.