Resampling imbalanced data for network intrusion detection datasets

Sikha Bagui* and Kunqi Li

Abstract

Machine learning plays an increasingly significant role in the building of Network Intrusion Detection Systems. However, machine learning models trained with imbalanced cybersecurity data cannot recognize minority data, hence attacks, effectively. One way to address this issue is to use resampling, which adjusts the ratio between the different classes, making the data more balanced. This research looks at resampling's influence on the performance of Artificial Neural Network multi-class classifiers. The resampling methods, random undersampling, random oversampling, random undersampling and random oversampling, random undersampling with Synthetic Minority Oversampling Technique, and random undersampling with Adaptive Synthetic Sampling Method were used on benchmark Cybersecurity datasets, KDD99, UNSW-NB15, UNSW-NB17 and UNSW-NB18. Macro precision, macro recall, macro F1-score were used to evaluate the results. The patterns found were: First, oversampling increases the training time and undersampling decreases the training time; second, if the data is extremely imbalanced, both oversampling and undersampling increase recall significantly; third, if the data is not extremely imbalanced, resampling will not have much of an impact; fourth, with resampling, mostly oversampling, more of the minority data (attacks) were detected.

Keywords: Oversampling, Undersampling, Resampling, Imbalanced Data, Network Intrusion Detection Systems, SMOTE, ADASYN, Artificial Neural Networks, Macro precision, Macro recall

Introduction

Cybersecurity is increasingly becoming a major concern due to the increased reliance on computers and the Internet. In order to detect Cyber-attacks, it is prudent that we build efficient Network Intrusion Detection Systems, and the basis for doing this is to be able to analyze network traffic flow data, termed here as Cybersecurity data, efficiently and quickly. There is an inherent problem with most network traffic flow data or Cybersecurity data—the data is highly imbalanced, that is, there is a disproportionately large amount of good or normal traffic data and, in a most cases, very few attack instances. Even existing benchmark datasets suffer from this problem. Using imbalanced data for machine learning or deep learning algorithms like Artificial Neural Networks (ANN) is a major challenge. Moreover, many of these datasets require multi-class classification.
ANN needs to be trained on historical data, and can be seriously affected by imbalanced proportions in the data. When training data is extremely imbalanced, that is, when one class (or classes) outnumbers the other class(es) by a large proportion, majority data (the class or classes with larger proportions) will have a stronger influence on the ANN model than minority data (the class or classes in lesser proportions). Under these circumstances, the ANN model will recognize majority data well but have poor performance on recognizing minority data.

In most network traffic flow data or Cybersecurity data, benign or normal data makes up a large proportion of the dataset, and attack data makes up only a small proportion of the dataset. If this imbalanced data is used to train an ANN model, the ANN model will have good performance on recognizing the benign data and bad performance on recognizing the attack data. This means that the model will recognize benign data as benign data and might also recognize attack data as benign. Especially in multiclassification, if there are small numbers of certain attack types, the minority attack data may be recognized as benign data or majority attack data. When network traffic cybersecurity data is being used for attack detection, recognizing minority attack data correctly is more important than recognizing majority benign data correctly.

In order to improve performance on classifying imbalanced data, researchers have suggested a number of approaches including resampling, cost sensitive kernel modification methods, and active learning methods [15]. This paper focuses on resampling strategies. The resampling techniques, random undersampling (RU), random oversampling (RO), random undersampling and random oversampling (RURO), random undersampling with Synthetic Minority Oversampling Technique (RU-SMOTE), and random undersampling with Adaptive Synthetic Sampling Method (RU-ADASYN) were applied to six benchmark cybersecurity datasets, KDD99, UNSW-NB15, UNSW-NB17-Ecobee_Thermostat, UNSW-NB17-Danmini_Doorbell (see Footnote 3), UNSW-NB17-Philips_B120N10_Baby_Monitor (see Footnote 3), and UNSW-NB18 [18], before performing classification using ANN. The classification results are evaluated using macro metrics including macro precision, macro recall and macro F-1 score. The training time, which usually forms the major part of the total running time of the algorithm, was also considered. Results of regular ANN using scikit-learn were compared to ANN in the Big Data framework using an EC2 instance of the Spark Machine Learning Library (MLlib) on an EMR Cluster.

The uniqueness of this work can be stated as:

- Applying new resampling technique combinations of random undersampling and random oversampling on imbalanced data. The following unique resampling combinations were used:
  - Random undersampling and random oversampling taken together (RURO).

---

1 http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (Accessed 03-15-2020).

2 “The UNSW-NB15 Dataset Description,” Cyber Range Lab of the Australian Centre for Cyber Security (ACCS), [Online], Available: https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/ (Accessed 09-19-2019).

3 https://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaloT.
- Random undersampling with the random oversampling technique, SMOTE (RU-SMOTE).
- Random undersampling with the random oversampling technique, ADASYN (RU-ADASYN).

- Studying the behavior of the above and other resampling techniques, random undersampling and random oversampling, in the domain of Cybersecurity data, a crucial emerging domain with respect to imbalanced data.
- Applying resampling to classification using ANN.
- The application of all of the above on the Big Data Framework using Spark.

The rest of this paper is organized as follows. A background of the different resampling techniques is presented in the "Resampling techniques implemented" section; this is followed by a section on "Related works"; the following section provides a brief "Description of the datasets" used in this study; the “Experimental design” section presents the study design, followed by the "Evaluation metrics", "Results and discussion"; and finally, the "Conclusion" is presented.

**Resampling techniques implemented**

To address the problem of imbalanced learning, many resampling techniques have been created. Resampling techniques include: oversampling, undersampling, combining oversampling and undersampling techniques, and ensembling sampling. Both oversampling and undersampling are aimed at changing the ratios between the majority classes and minority classes. Combining oversampling and undersampling techniques use both oversampling and undersampling techniques to create a more balanced new dataset. By making the training data more balanced, resampling enables different classes to have relatively the same influence on the outcomes of the classification model. The resampling techniques used in this paper, random undersampling, random oversampling, random undersampling and random oversampling, random undersampling with SMOTE, and random undersampling with ADASYN, are presented next.

Random undersampling refers to the process of reducing the number of samples. Samples from the majority class(es) are randomly picked with or without replacement.\(^4\) After random undersampling, the number of cases (of the majority class) in the dataset decrease, which significantly reduces the training time in a model. However, data points removed by random undersampling may include important information, which may lead to a decrease in classification results. Lemaître et al. [20] presents a scikit learn toolbox to resample training data. In this paper, this toolbox was used to resample training data and Listing 1 presents example scikit learn code for random undersampling.

---

\(^4\) [https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under_sampling.RandomUnderSampler.html#imblearn.under_sampling.RandomUnderSampler](https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under_sampling.RandomUnderSampler.html#imblearn.under_sampling.RandomUnderSampler)
Random oversampling over-samples the minority class(es) by picking samples at random with replacement from the minority class(es) (see Footnote 1). Since oversampling increases the number of cases in the training dataset, random oversampling increases the training time of a model. Random oversampling may also lead to overfitting because
it adds replicated data to the dataset. Listing 2 presents example scikit learn code for random oversampling.

```python
# random_oversampling
print("random oversampling started")
label = data_training['attack_cat']
features = data_training[feature_list3]

from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(sampling_strategy='so', random_state=0)
f_resampled, l_resampled = ros.fit_resample(features, label)

data_resampled = pd.concat([f_resampled, l_resampled], axis=1)
data_train = data_resampled

print("random oversampling finished")
```

**Listing 2. Example Code of Random Oversampling**

Random undersampling and random oversampling uses the two methods together. Synthetic Minority Oversampling Technique (SMOTE), commonly used as a benchmark for oversampling [9, 34], improves on simple random oversampling by creating synthetic minority class samples [4] and addresses the problem of overfitting [5] that can happen with simple random oversampling. This is because the new data points generated by SMOTE are synthetic data points instead of mere duplications. To generate new minority data points, a linear combination of two similar samples from the minority class are used [4]. New feature values are uniformly interpolated between the minority instance and its respective nearest neighbors. SMOTE only considers within class neighbors. Listing 3 presents example scikit learn code for SMOTE oversampling, including the sampling strategy used. In this work, random undersampling is applied in combination with SMOTE, hence this is referred to as RU-SMOTE.
ADASYN [14], a pseudo-probabilistic oversampling technique, uses a weighted distribution for different minority data points according to their level of difficulty in learning. With ADASYN, more synthetic data is generated for minority class examples that are harder to learn as compared to those minority examples that are easier to learn. A fixed number of instances is generated for each minority instance, based on a weighted distribution of its neighbors [1]. Listing 4 presents example scikit learn code for ADASYN oversampling. In this work, random undersampling has been applied in combination with ADASYN, hence it is referred to as RU-ADASYN.

Table 1 presents a brief comparison of Random oversampling, SMOTE and ADASYN.
Related works

Resampling stems from the class imbalance problem. Leevy et al. [19] stressed on the importance of the class imbalance problem and presented a survey of works on the class imbalance problem. The works were mainly divided into data-level methods and algorithm-level methods.

Some of the recent works on algorithm-level methods are: Johnson and Khoshgoftaar [17] examined existing deep learning techniques for addressing class imbalance; Raghuwanshi and Shukla [28] designed a novel BalanceCascade-based kernelized extreme learning machine to handle the problem of class imbalance; Luque et al. [21] presented a new way of measuring imbalance. A set of null-biased multi-perspective Class Balance Metrics were proposed which extended the concept of Class Balance Accuracy to other performance metrics.

There are also several studies on the data-level methods. Several studies have been carried out on comparison of oversampling and undersampling methods for handling the class imbalance problem. Douzas and Bacao [7] developed a conditional version of Generative Adversarial Networks to approximate the true data distribution and generate data for minority classes of various imbalanced datasets. Douzas et al. [8] presented an oversampling method based on k-means clustering and SMOTE which avoids the generation of noise and overcomes imbalances between and within classes.

More [25] reviewed a number of resampling techniques, including random undersampling of the majority class, random oversampling of the minority class, SMOTE, and many other techniques, to handle unbalanced datasets and study their effect on classification.

Amin et al. [2] surveyed six well-known sampling techniques: mega-trend diffusion function (MTDF), SMOTE, ADASYN, couples top-N reverse k-nearest neighbor, majority weighted minority oversampling technique, and immune centroids oversampling technique. Their work showed that the overall predictive performance of MTDF and rules-generation based on genetic algorithms performed the best as compared with the rest of the evaluated oversampling methods and rule-generation algorithms.

Abdi and Sattar [1] looked at different synthetic oversampling techniques and proposed a new oversampling algorithm based on Mahalanobis distance. They showed that their proposed method generates less duplicate and overlapping data points as opposed to other oversampling techniques.

Cieslak et al. [6] used SMOTE to detect network traffic intrusions. Blagus and Lusa [4] investigated the theoretical properties of SMOTE and its performance on high-dimensional data. They considered a two-class classification using Classification and Regression Trees, k-NN, linear discriminant analysis, random forests and support vector

|                         | Random OverSampling | SMOTE                  | ADASYN                  |
|-------------------------|---------------------|------------------------|-------------------------|
| Resampling time         | The shortest of the three methods | Medium                 | The longest of the three methods |
| Each new data point is based on | Only one minority data point | Some minority data points around the new data point | Both minority data points and majority data points |

Table 1 Brief comparison of Random Oversampling, SMOTE and ADASYN
machines (SVM). Wallace et al. [33] also used SMOTE with SVM as the base classifier. Past works have also looked at the effects of dimensionality on SMOTE [16]. Hulse et al. [16] showed that in low-dimensional data, simple undersampling tends to outperform SMOTE. Ertekin et al. [10] and Radivojac et al. [27] evaluated the performance of SMOTE based on the number of samples. Song et al. [29] looked at the class imbalance problem in software detection prediction.

Many works also looked at resampling in the context of Big Data. Fernandez et al. [11] looked at the imbalance problem in the Big Data framework. Basgall et al. [3] developed SMOTE-BD, a fully scalable oversampling technique for imbalanced classification in Big Data Analytics. Terzi and Sagiroglu [30] developed a distributed cluster based resampling for imbalanced Big Data, which was designed to overcome both between-class and within-class imbalance problems in big data. Gutiérrez et al. [13] proposed SMOTE-GPU to efficiently handle large datasets (several millions of instances) on a wide variety of commodity hardware, including a laptop computer. Triguero et al. [32] independently managed the majority as well as minority classes. They undersampled the majority class and took advantage of Apache Spark’s in-memory operations to diminish the effects of the small sample size of the minority class.

In summary, several studies have looked at the class imbalance problem, both in traditional data as well as big data, using various resampling oversampling and undersampling techniques. However, none of the studies have analyzed the application of the resampling techniques, random undersampling and random oversampling (RURO) used together, random undersampling with SMOTE (RU-SMOTE), and random undersampling with ADASYN (RU-ADASYN), using Spark’s ANN multi-class classifier, on imbalanced network traffic cybersecurity data, the work performed in this study. In this study, basically a data-level approach, resampling of the majority and minority classes are handled independently.

Description of the datasets

For experimentation, six popular datasets were used: KDD99 (see Footnote 1), UNSW‑NB15 (see Footnote 2), UNSW-NB17 (Ecobee_Thermostat, Danmini_Doorbell, and Philips_B120N10_Baby_Monitor) (see Footnote 3), and UNSW-NB18 [18]. Next is a description of the datasets.

KDD99
The KDD99 dataset, considered a benchmark cybersecurity dataset for a long time, is a 41 feature dataset. The attack records of this dataset can be classified into four broad categories and 22 subcategories. Table 2 presents the distribution of benign and attack data (in the four broad categories). The data is extremely imbalanced. Benign data makes up almost 20% of the data and the DoS attacks make up almost the other 80% of the data, hence the other attack categories have extremely few case instances.

UNSW‑NB15
The UNSW-NB15 dataset, created by the Cyber Range Lab of the Australian Centre for Cyber Security has 49 features [26]. There are 10 categories (9 attack categories plus 1 benign category). Table 3 presents the distribution of benign and attack data in
UNSW-NB15. Here too, the data is highly imbalanced. Benign traffic makes up 88.5% of the traffic, while the nine attack categories combined make up the other 11.5%. It can be noted that worms make up only 0.0069% of the data, hence there are extremely few cases.

UNSW-NB17

The UNSW-NB17 dataset was generated by 9 IoT devices. There are 9 sub datasets in UNSW-NB17, of which three were arbitrarily selected for this study: Ecobee_Thermostat, Danmini_Doorbell, and Philips_B120N10_Baby_Monitor. Each sub dataset includes two of the most common IoT botnets, Gafgyt and Mirai [22]. Each of the botnets has 5 attack subcategories, hence there are 10 categories of attack traffic and 1 benign category. There are 115 independent features in this data set. The csv files were used, which were extracted from pcap files by Kitsune [23]. Tables 4, 5, and 6 present the distribution of the benign and attack data in these datasets respectively. These datasets are imbalanced, but not as imbalanced as the KDD99 or UNSW-NB15. In these datasets, the Gafgyt_junk and Gafgyt_scan have close of 3% of the data each, but the other attack categories are a little more balanced. And, in these datasets, the benign traffic is not disproportionately high, as is the case in UNSW-NB15.

UNSW-NB18

The UNSW-NB18 BoT-IoT dataset was created by designing a realistic network environment in the Cyber Range Lab of The center of UNSW Canberra Cyber [18]. Table 7 presents the distribution of benign and attack data in this dataset. Here again, the data

---

Table 2 % of benign and attack traffic in KDD99

| Category | % of Traffic (%) |
|----------|------------------|
| Benign   | 19.8590          |
| Probe    | 0.8391           |
| DoS      | 79.2778          |
| u2r      | 0.0011           |
| r2l      | 0.0230           |

Table 3 % of benign and attack traffic in UNSW-NB15

| Category     | % of Traffic (%) |
|--------------|------------------|
| Benign       | 88.5529          |
| Fuzzers      | 0.2016           |
| Reconnaissance| 0.0702          |
| Shellcode    | 0.0089           |
| Analysis     | 0.1069           |
| Backdoors    | 0.0213           |
| DoS          | 0.6528           |
| Exploits     | 1.7773           |
| Generic      | 8.6012           |
| Worms        | 0.0069           |
| Category                  | % of Traffic (%) |
|---------------------------|------------------|
| **Table 4 % of benign and attack traffic in UNSW-NB17_Ecobee** |                  |
| Benign                    | 1.57             |
| Mirai_ack                 | 13.55            |
| Mirai_scan                | 5.17             |
| Mirai_syn                 | 13.97            |
| Mirai_udp                 | 18.12            |
| Mirai_udpplain            | 10.4             |
| Gafgyt_combo              | 6.34             |
| Gafgyt_junk               | 3.63             |
| Gafgyt_scan               | 3.29             |
| Gafgyt_tcp                | 11.37            |
| Gafgyt_udp                | 12.54            |
| **Table 5 % of benign and attack traffic in UNSW-NB17_Doorbell** |                  |
| Benign                    | 4.87             |
| Mirai_ack                 | 10.04            |
| Mirai_scan                | 10.57            |
| Mirai_syn                 | 12.0             |
| Mirai_udp                 | 23.34            |
| Mirai_udpplain            | 8.05             |
| Gafgyt_combo              | 5.86             |
| Gafgyt_junk               | 2.85             |
| Gafgyt_scan               | 2.93             |
| Gafgyt_tcp                | 9.05             |
| Gafgyt_udp                | 10.40            |
| **Table 6 % of Benign and Attack Traffic in UNSW-NB17_Philips** |                  |
| Benign                    | 15.95            |
| Mirai ack                 | 8.29             |
| Mirai_scan                | 9.43             |
| Mirai_syn                 | 10.75            |
| Mirai_udp                 | 19.75            |
| Mirai_udpplain            | 7.36             |
| Gafgyt_combo              | 5.29             |
| Gafgyt_junk               | 2.58             |
| Gafgyt_scan               | 2.54             |
| Gafgyt_tcp                | 8.43             |
| Gafgyt_udp                | 9.63             |
is highly imbalanced. TCP attacks make up approximately 43% of the cases and UDP attacks make up approximately 54% of the cases. In this dataset too, normal traffic makes up only 0.031% of the dataset, hence is very low. This is almost the opposite of the pattern in UNSW-NB15.

**Experimental design**

Figure 1 shows the flow chart of the experimental design. For each dataset, the dataset was split into a training set (70%) and a testing set (30%). Both training as well as testing datasets were pre-processed and standardized. The training dataset was then resampled and the ANN model trained. The test dataset was tested on the ANN model.

For each dataset, six sets of classifications were performed with the following combinations of resampling techniques.

- No resampling (NR).
- Random undersampling (RU).
- Random oversampling (RO).
- Random undersampling and random oversampling (RURO).
- Random undersampling and SMOTE (RU-SMOTE).
- Random undersampling and ADASYN (RU-ADASYN).

Resampling of the majority and minority classes was performed independently, meaning that each category in each dataset was considered individually, rather than taking a fixed % for under or over sampling. Classification was performed using Artificial Neural Networks (ANN) available in Apache Spark. All experiments were run in two modes: (i) on a local machine using Scikit Learn, and (ii) for the Big Data framework, Apache Spark, on Amazon’s Web Service (AWS) EMR cluster. The AWS EMR cluster was setup with 3 nodes (one master nodes and two slave nodes). Each node was an m5.xlarge EC2.

**Apache Spark**

Apache Spark, an open source distributed cluster computing framework, is part of the Hadoop Ecosystem, but has an edge over Hadoop in terms of speed due to it’s in-memory processing architecture. Spark can run up to 100 times faster than Hadoop for data and processes completely residing in-memory [12]. The Spark framework also provides benefits such as scalability and fault tolerance [12], as well as providing a rich set of APIs.
Fig. 1 Flow chart of the experiments
that allow developers to perform many complex analytics operations out-of-the-box. This work took advantage of the Spark Core and Spark MLlib APIs.

Spark Core allows for basic operations on data including mapping, reducing, and filtering. These operations are available in Spark’s primary data structure, Resilient Distributed Datasets (RDDs) [12], which parallelizes computations in a transparent way. Apache Spark’s Machine Learning Library, MLlib, makes machine learning scalable and easy. MLlib provides tools including:

1. ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering.
2. Featurization: feature extraction, transformation, dimensionality reduction, and selection.
3. Pipelines: tools for constructing, evaluating, and tuning ML Pipelines.
4. Persistence: saving and load algorithms, models, and Pipelines.
5. Utilities: linear algebra, statistics, data handling, etc.

The ANN model used in this paper is multilayer perceptron classifier of Spark MLlib.

Artificial Neural Networks
As shown in Fig. 2, ANN is a feed forward neural network in which the information moves from the input layer to hidden layers then to the output layer. A fully connected ANN model was used with the number of neurons in the input layer set to the number of features in the data and the number of neurons in the output layer set to the number of the classes. The intermediary layer used a sigmoid function, where \(i\) is the input [31]:

\[
f(z_i) = \frac{1}{1 + e^{-z_i}}
\]  

(1)

The sigmoid function smoothly puts the input to an output between zero and one. This allows for the interpretation or output of any individual layer to be taken as a probability. The output layer used the softmax function [31]:

![Fig. 2 ANN model used](image)
The softmax function is often used as the activation function for the last layer of a neural network. This activation function turns numbers into probabilities that sum to one. The softmax function outputs a vector that represents the probability distributions to a list of potential outcomes.

Evaluation metrics
In this section, first a discussion of why the macro metrics was used is presented, and then the metrics are presented.

Using macro metrics
For this work, macro precision, macro recall, and macro F1-score were used instead of the micro or weighted metrics to evaluate the results. The macro metrics compute the metrics independently for each class and then take the average, hence all classes, majority as well as minority, are weighted equally.

The micro metrics aggregate the contributions of all classes to compute the average metric, hence results get skewed towards classes with larger case numbers. Micro metrics, in a multi-class setting, with highly imbalanced data, will often produce equal precision, recall and F1-score that is artificially high. The good performance of the majority data overly influences the micro metrics, which is the case for highly imbalanced data.

The weighted metrics compute the averages by taking the class size into account, that is, the number of cases for each class, hence it is the “weighted” average. If a model recognizes majority data correctly but does not recognize minority data correctly, the weighted metrics will be high. Hence, in this case, the weighted metrics does reflect the bad performance of the classifying minority data. Also, the weighted metric may produce an F1-score that is not between precision and recall. Hence, even if the weighted metrics may be good, it was not used for this work.

Since three of the cybersecurity datasets used in this study are highly imbalanced, after resampling, the macro metrics were used as the evaluation metrics in this study. The macro metrics produce relatively lower results than the micro metrics. This is because the macro metrics treat all classes equally, hence the poor performance of the minority classes will lower the macro metrics. But, though the macro metrics reflect the poor performance of classifying minority data, it was deemed that, for these datasets, the macro metrics would better reflect the overall performance of classifying the data.

Metrics formulas
Below are the respective formulas for accuracy, precision, recall and the F1-score. Although the micro, macro and weighted metrics are all computed slightly differently (as discussed in the previous section), all three metrics use the same formulas for calculating precision, recall and the F1-score.

Precision is the positive predictive value, or the percentage of classified attack instances that are truly classified as attack, calculated by \[ \text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \]
Recall or attack detection rate (ADR) is the effectiveness of a model in identifying an attack. The objective is to target a higher ADR. The ADR is calculated by [24]:

\[
\text{Precision} = \frac{TP}{TP + FP} \tag{3}
\]

\[
\text{Recall} = \frac{TP}{TP + FN} \tag{4}
\]

F-measure is the harmonic mean of precision and recall. The higher the F-measure, the more robust the classification model. The F-measure is calculated by [24]:

\[
\text{F}1 - \text{score} = F = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{5}
\]

True Positive (TP) is the number of positive records that were correctly labeled as positive. True Negative (TN) is the number of negative records that were correctly labeled as negative. False Positive (FP) is the number of negative records that were incorrectly labeled as positive. False Negative (FN) is the number of positive records that were correctly labeled as negative.

**Results and discussion**

In this section, first, the classification results for all six datasets, with no resampling, is presented. This will be used as a benchmark for analyzing the results. Then, for each dataset, resampling results and the classification results using the different resampling techniques, are presented. The ANN classification was done in two modes: (i) on the Big Data framework using Spark’s Machine Learning Library; and (ii) using Scikit Learn on a local machine. Observations and discussions follow each set of results.

**Classification with original datasets (no resampling)**

The first set of classifications were done with the original six datasets, that is, with no resampling. These results form the benchmark for the ANN classification results.

Table 8 present the macro precision, macro recall, macro F1 score and training time taken for ANN classification with no resampling on AWS with Spark for all the six datasets. Similarly, Table 9 presents the macro precision, macro recall, macro F1 score and training time taken for ANN classification with no resampling on the local machine for all the six datasets. The testing time was not recorded since the training time is the more significant of the two. Figure 3 graphically presents the macro precision, macro recall,
Observations and discussion

- ANN classification on Scikit-Learn has better performance than ANN classification on Spark. The macro precision, macro recall, and macro F1-score are higher on the ANN classification on Scikit-Learn.
- The ANN classification model on Spark trains faster than the ANN classification model of the local machine. This is expected since Spark is the Big Data framework, hence parallel processing is performed.
- UNSW-NB15 has one category that has the most cases, that is, the benign category comprises almost 88% of the cases, hence this imbalance is causing the low results for this non-resampled dataset. BoT-IoT has two categories that have a large combined total number of cases, TCP (43%) and UDP (54%), hence this imbalance is also causing low results. The results of UNSW-NB17 are pretty high even without resampling, mainly because the three UNSW-NB17 datasets are relatively balanced compared to the other three datasets.

Classification with the Resampled datasets

This section presents the results of the resampling and classification on the six different datasets, KDD99 (see Footnote 1), UNSW-NB15 (see Footnote 2), UNSW-NB17 (Ecobee_Thermostat, Danmini_Doorbell, and Philips_B120N10_Baby_Monitor) (see Footnote 3), and UNSW-NB18 [18]. For all datasets, macro results are presented. For

Table 9  ANN classification results on Scikit-Learn on local machine (no resampling)

|           | KDD99 | NB15   | BoT-IoT   | NB17-Ecobee | NB17-Danmini | NB17-Philips |
|-----------|-------|--------|-----------|-------------|--------------|--------------|
| Macro precision | 0.877317 | 0.53307 | 0.607171 | 0.929664 | 0.924668 | 0.912172 |
| Macro recall   | 0.832509 | 0.412258 | 0.57664  | 0.907989 | 0.903897 | 0.893773 |
| Macro F1 score | 0.842974 | 0.411132 | 0.585591 | 0.876052 | 0.875883 | 0.869574 |
| Training time (s) | 504.875 | 1119.422 | 1696.188 | 2356.359 | 896.3594 | 1385.156 |

Fig. 3  Classification results with no resampling on AWS
two of the datasets, KDD99 and UNSW-NB15, however, the micro metrics were also presented (for AWS runs), but these metrics were not presented for the rest of the datasets because of the artificially high micro results as well as almost equal micro recall, micro precision and micro F1 score. Also, the confusion matrices were presented for the highly imbalanced datasets, since there was little influence on the not highly imbalanced datasets. Also, in the respective resampling sections, for brevity’s sake, only the RU, RO, and RURO are presented, though RU-SMOTE and RU-ADASYN resampling was also done for the classifications.

**Experimentation on KDD99**

The first section presents the resampling of KDD99 and then the classification results are presented. An analysis of the KDD99 results are presented in the observations and discussions section.

**Resampling KDD99**

Table 10 presents the number of samples after before resampling, after RU, after RO, and after RURO and Fig. 4 presents the number of samples before resampling, after RU, and after RO. Before Resampling represents 70% of the original KDD99 dataset, which was used for training the model. From Table 10, it can be noted that u2r had only 40 instances and r2l had only 794 instances before resampling, so oversampling makes a big difference for these two attacks. With RU, the number of instances of benign and DoS were reduced to the number of Probe instances, making all three categories equal, while there was still a low number of instances for u2r and r2l. Hence with RU, the data still appears to be imbalanced overall. With RO, the number of Probe, DoS

### Table 10 Resampling of KDD99

| KDD99 | Number of cases for training model |
|-------|------------------------------------|
|       | Before resampling | RU | RO | RURO |
| Benign | 680,671 | 28,609 | 680,671 | 28,609 |
| Probe | 28,609 | 28,609 | 28,609 | 28,609 |
| DoS | 2,718,787 | 28,609 | 2,718,787 | 28,609 |
| u2r | 40 | 40 | 28,609 | 28,609 |
| r2l | 794 | 794 | 28,609 | 28,609 |
and r2l instances were made the same, although the number of benign and DoS instances were still high. With RURO, the number of instances for each attack were made equal, hence the results were not shown in Fig. 4.

**Classification results for KDD99** Table 11 presents the ANN classification results for KDD99 on AWS using Spark and Table 12 presents the ANN classification results for KDD99 run on the local machine with Scikit-Learn. The results of macro precision, macro recall and macro F1 score are presented for NR, RU, RO, RURO, RU-SMOTE, RU-ADASYN for KDD99. Table 11 also presents the results of the micro precision, micro

| KDD99       | NR     | RU     | RO     | RURO   | RU-SMOTE | RU-ADASYN |
|-------------|--------|--------|--------|--------|----------|-----------|
| Micro precision | 0.99962 | 0.998557 | 0.998661 | 0.995024 | 0.99499  | 0.993675  |
| Micro recall  | 0.99962 | 0.998557 | 0.998661 | 0.995024 | 0.99499  | 0.993675  |
| Micro F1 score| 0.99962 | 0.998557 | 0.998661 | 0.995024 | 0.99499  | 0.993675  |
| Macro precision | 0.735853 | 0.689996 | 0.676256 | 0.601879 | 0.596781 | 0.590074  |
| Macro Recall  | 0.732151 | 0.905783 | 0.918453 | 0.968456 | 0.955936 | 0.955935  |
| Macro F1 score| 0.733986 | 0.74276  | 0.712264 | 0.619913 | 0.619568 | 0.609858  |
| Macro training time (s) | 534 | 78 | 522 | 96 | 96 | 96 |

| KDD99       | NR     | RU     | RO     | RURO   | RU-SMOTE | RU-ADASYN |
|-------------|--------|--------|--------|--------|----------|-----------|
| Macro precision | 0.877317 | 0.710881 | 0.692416 | 0.618828 | 0.625649 | 0.616133  |
| Macro recall  | 0.832509 | 0.88417  | 0.937749 | 0.922727 | 0.960406 | 0.96097   |
| Macro F1 score| 0.842974 | 0.761345 | 0.730075 | 0.651291 | 0.659022 | 0.643459  |
| Training time (s) | 504.875 | 69.71875 | 904.6563 | 329.9844 | 355.3281 | 350.5469  |

![Graph](image_url)  
**UNSW-NB15 NR Vs RU**  
NR vs RU for KDD99 on AWS

and R2L instances were made the same, although the number of benign and DoS instances were still high. With RURO, the number of instances for each attack were made equal, hence the results were not shown in Fig. 4.

**Classification results for KDD99** Table 11 presents the ANN classification results for KDD99 on AWS using Spark and Table 12 presents the ANN classification results for KDD99 run on the local machine with Scikit-Learn. The results of macro precision, macro recall and macro F1 score are presented for NR, RU, RO, RURO, RU-SMOTE, RU-ADASYN for KDD99. Table 11 also presents the results of the micro precision, micro
recall and micro F1 score. The training time of the models was also recorded in Tables 11 and 12. Figure 5 presents a comparison of the micro and macro metrics run on AWS for no resampling and random undersampling. It can be noted that the micro precision, micro recall and micro F1 score were almost equal as well as artificially high, hence the evaluations were based on the macro metrics.

Figure 6 presents the graphical results of the different resampling methods using Spark. The results of the different resampling methods on the local machine show a similar trend, hence are not presented.

| Table 13 Confusion matrix: KDD99 NR |
|------------------------------------|
| **KDD99 NR** | Predicted label |
| 0  | 1  | 2  | 3  | 4  |
| True label |
| 0  | 291,884 | 107 | 22 | 0  | 97 |
| 1  | 161    | 12,329| 3  | 0  | 0  |
| 2  | 45     | 3     | 1,164,535| 0  | 0  |
| 3* | 7      | 0     | 1   | 0  | 4  |
| 4* | 108    | 0     | 0   | 0  | 224 |

| Table 14 Confusion Matrix: KDD99 RU |
|------------------------------------|
| **KDD99 RU** | Predicted label |
| 0  | 1  | 2  | 3  | 4  |
| True label |
| 0  | 290,910 | 647 | 42 | 55  | 456 |
| 1  | 16     | 12,476| 0  | 0  | 1  |
| 2  | 785    | 76    | 1,163,701| 0  | 21 |
| 3* | 2      | 0     | 0   | 7  | 3  |
| 4* | 15     | 0     | 0   | 1  | 316 |
Table 15  Confusion Matrix: KDD99 RO

| KDD99 RO | Predicted label |
|----------|----------------|
|          | 0   | 1   | 2   | 3   | 4   |
| True label |     |     |     |     |     |
| 0         | 290,943 | 150 | 159 | 335 | 523 |
| 1         | 225    | 12,259 | 6  | 3   | 0   |
| 2         | 536    | 6    | 1,164,037 | 1  | 3   |
| 3*        | 2      | 0    | 0   | 8   | 2   |
| 4*        | 15     | 0    | 1   | 1   | 315 |

Table 16  Confusion Matrix: KDD99 RURO

| KDD99 RURO | Predicted label |
|------------|----------------|
|            | 0   | 1   | 2   | 3   | 4   |
| True label |     |     |     |     |     |
| 0         | 286,222 | 608 | 61  | 967 | 4252 |
| 1         | 93     | 12,394 | 0  | 6   | 0   |
| 2         | 1001   | 278  | 1,163,274 | 3  | 27  |
| 3*        | 0      | 0    | 0   | 11  | 1   |
| 4*        | 5      | 0    | 0   | 10  | 317 |

Table 17  Confusion Matrix: KDD99 RU-SMOTE

| KDD99 RU-SMOTE | Predicted Label |
|---------------|----------------|
|               | 0   | 1   | 2   | 3   | 4   |
| True label    |     |     |     |     |     |
| 0         | 286,351 | 984 | 105 | 812 | 3858 |
| 1         | 65     | 12,424 | 0  | 3   | 1   |
| 2         | 1088   | 408  | 1,163,059 | 1  | 27  |
| 3*        | 1      | 0    | 0   | 10  | 1   |
| 4*        | 6      | 0    | 0   | 3   | 323 |

Table 18  Confusion Matrix: KDD99 RU-ADASYN

| KDD99 RU-ADASYN | Predicted label |
|----------------|----------------|
|                | 0   | 1   | 2   | 3   | 4   |
| True label     |     |     |     |     |     |
| 0         | 284,845 | 1180 | 100 | 930 | 5055 |
| 1         | 62     | 12,430 | 0  | 1   | 0   |
| 2         | 1520   | 362  | 1,162,652 | 2  | 47  |
| 3*        | 0      | 0    | 0   | 11  | 1   |
| 4*        | 13     | 0    | 0   | 22  | 297 |

Confusion matrices for KDD99

Tables 13, 14, 15, 16, 17 and 18 show the confusion matrices using the various resampling methods for the AWS runs on Spark. The predicted label vs the true labels
are shown, that is, if it was predicted as attack type 1, was it really attack type 1. The categories that had a really low number of instances are marked with an asterisk and the increases in the minority data identification are in italics.

**Observations and discussion**  Few conclusions that can be drawn from these above sets of results:

- The micro precision, micro recall and micro F1 score were showing very artificially high numbers as well as almost the same results for NR as well as RU (Fig. 5), hence were not considered useful for any further analysis.
- There is almost no overall significant difference between the ANN classification results on AWS and ANN classification results on the local machine in terms of the macro precision and macro recall, and macro F1 score. After oversampling though, it took longer to run on the local machine than on AWS.
- On both AWS and the local machine, when the minority data is increased by oversampling or majority data is decreased by undersampling, the macro precision decreases, and the macro recall increases. Oversampling improves the macro recall significantly. Macro precision decreasing implies that the ratio of the false positive to true positive is going up, and the macro recall increasing implies that the ratio of the false negative to true positive is going down. This means that, for this set of experiments, the false positives are going up and the false negatives are going down.
- The confusion matrices also show an increase in the number of correctly classified cases for the very low minority classes (shown with asterisk) with resampling (results are in italics in Tables 13, 14, 15, 16, 17 and 18), with the best results for RURO and RU-SMOTE. From Table 10 it can be noted that RURO had an equal number of for all the attack types. And, even though the RU still had an imbalanced distribution, it was better than no resampling, and also performed better than no resampling.
- Generally, the F1 score went down for both undersampling and oversampling. It went slightly up only for RU on AWS, but not significantly.
- Except for RO, the training time decreased in all resampling scenarios, for both the local machine as well as AWS, and of course, the training time on AWS was a lot shorter than on the local machine (though it was higher on AWS when no resampling was done).
- From Table 11 (AWS), it can be observed that RURO’s macro recall was the highest, at 96%, while RU-SMOTE and RU-ADASYN’s macro recall were very close, at 95.59%. RU’s macro recall (90.5%) was lower than the recall of the other resampling methods, but a lot better than NR (73%).
- From Table 12 (local machine), it can be observed that, RU-SMOTE and RU-ADASYN performed the best in terms of macro recall, at 96%. RU again had the lowest macro recall of the all the resampling methods (88%), but performed better than NR (83%).
Experimentation on UNSW-NB15

The first section presents the resampling of UNSW-NB15 and then the classification results are presented. An analysis of the UNSW-NB15 results are presented in the observations and discussions section.

Resampling UNSW-NB15  Table 19 presents the number of samples before resampling, after RU, after RO, and after RURO and Fig. 7 graphically presents the data before resampling, after RU, and after RO. Before Resampling represents 70% of the original UNSW-NB15 dataset, which was used for training the model. From Table 19 it can be noted that, with RU, though the number of benign and generic instances were reduced, some of the other attacks like Shellcode, Backdoors and Worms still had a lower number of instances. And overall, with RU, the data was still imbalanced. RO makes the attack instances equal for the rest of the attacks except the benign and generic traffic. The number of benign traffic instances was still very high compared to the rest of the attacks, as shown in Fig. 7. By RURO all the attack instances are made equal.

Classification results for UNSW-NB15  Table 20 presents the ANN classification results for UNSW-NB15 on AWS using Spark and Table 21 presents the ANN clas-
Table 20 ANN Classification results for UNSW-NB15 on AWS with Spark for various resampling methods

| NB15       | NR        | RU        | RO        | RURO     | RU-SMOTE  | RU-ADASYN |
|------------|-----------|-----------|-----------|----------|-----------|-----------|
| Micro precision | 0.985149 | 0.981616 | 0.97705  | 0.967425 | 0.966663 | 0.966274 |
| Micro recall  | 0.985149 | 0.981616 | 0.97705  | 0.967425 | 0.966663 | 0.966274 |
| Micro F1 score | 0.985149 | 0.981616 | 0.97705  | 0.967425 | 0.966663 | 0.966274 |
| Macro precision | 0.337079 | 0.428798 | 0.397854 | 0.35708  | 0.360195 | 0.368759 |
| Macro recall  | 0.320941 | 0.423203 | 0.733399 | 0.744253 | 0.746937 | 0.737735 |
| Macro F1 score | 0.308697 | 0.376054 | 0.423431 | 0.384293 | 0.391973 | 0.3865  |
| Macro training time (s) | 432     | 138       | 600       | 180      | 186       | 186      |

Table 21 ANN classification results for UNSW-NB15 on local machine for various resampling methods

| UNSW-NB15 | NR        | RU        | RO        | RURO     | RU-SMOTE  | RU-ADASYN |
|-----------|-----------|-----------|-----------|----------|-----------|-----------|
| Macro precision | 0.53307  | 0.545088 | 0.425312  | 0.400095 | 0.411192 | 0.399882  |
| Macro recall  | 0.412258 | 0.500229 | 0.760052  | 0.775892 | 0.763106 | 0.764241  |
| Macro F1 score | 0.411132 | 0.449319 | 0.453457  | 0.443606 | 0.449048 | 0.443679  |
| Training time (s) | 1119.422 | 623.2188 | 1682      | 1191.922 | 1498.344 | 1713.953  |

Fig. 8 Comparison of the micro and macro metrics for UNSW-NB15

Fig. 9 ANN classification results for UNSW-NB15 on AWS with Spark for various resampling methods
sification results for UNSW-NB15 run on the local machine with Scikit-Learn. The results of macro precision, macro recall and macro F1 score are presented for NR, RU, RO, RURO, RU-SMOTE and RU-ADASYN for UNSW-NB15. Table 20 also presents the results of the micro precision, micro recall and micro F1 score. The training time was also recorded in Tables 19 and 20 respectively. Figure 8 presents a comparison of the micro and macro metrics on AWS for no resampling and random oversampling. It can be noted that the micro precision, micro recall and micro F1 score were almost equal as well as artificially high, hence the evaluations were done based on the macro metrics.

**Table 22** Confusion Matrix: NB15 NR

| NB15 NR | Predicted label |
|---------|----------------|
|         | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
| True label |
| 0       | 664,828 | 262 | 0  | 0  | 3  | 0  | 2  | 694 | 4  | 0  |
| 1       | 879    | 427 | 0  | 0  | 0  | 0  | 0  | 181 | 1  | 0  |
| 2       | 432    | 11  | 0  | 0  | 0  | 0  | 0  | 95  | 4  | 0  |
| 3*      | 1      | 1   | 0  | 0  | 0  | 0  | 0  | 56  | 0  | 0  |
| 4       | 148    | 66  | 0  | 0  | 0  | 0  | 0  | 17  | 656 | 0  |
| 5*      | 2      | 70  | 0  | 0  | 0  | 0  | 0  | 101 | 0  | 0  |
| 6       | 167    | 85  | 0  | 0  | 0  | 0  | 0  | 93  | 4535 | 34 |
| 7       | 609    | 162 | 0  | 0  | 0  | 0  | 0  | 121 | 12,306 | 14 |
| 8       | 201    | 22  | 0  | 0  | 0  | 0  | 0  | 13  | 1468 | 62,703 |
| 9*      | 3      | 0   | 0  | 0  | 0  | 0  | 0  | 41  | 0  | 0  |

**Table 23** Confusion Matrix: NB15 RU

| NB15 RU | Predicted label |
|---------|----------------|
|         | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
| True label |
| 0       | 660,598 | 1587 | 331 | 2   | 181 | 0  | 14 | 2809 | 271 | 0  |
| 1       | 43     | 1155 | 27  | 0   | 0   | 0  | 1  | 240  | 22  | 0  |
| 2       | 27     | 70   | 201 | 0   | 0   | 0  | 0  | 239  | 5   | 0  |
| 3*      | 0      | 0    | 1   | 5   | 0   | 0  | 0  | 50   | 2   | 0  |
| 4       | 10     | 100  | 0   | 0   | 54  | 1  | 4  | 718  | 0   | 0  |
| 5*      | 0      | 105  | 0   | 0   | 0   | 0  | 0  | 68   | 0   | 0  |
| 6       | 23     | 116  | 20  | 1   | 2   | 1  | 43 | 4584 | 124 | 0  |
| 7       | 64     | 270  | 14  | 0   | 22  | 0  | 45 | 12,673 | 124 | 0  |
| 8       | 20     | 31   | 16  | 2   | 10  | 0  | 8  | 1347 | 62,973 | 0  |
| 9*      | 0      | 2    | 0   | 0   | 0   | 0  | 0  | 41   | 1   | 0  |
### Table 24  Confusion Matrix: NB15 RO

| NB15 RO | Predicted label |
|---------|-----------------|
|         | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
| True label |     |     |     |     |     |     |     |     |     |     |
| 0        | 661,537 | 1337 | 972 | 102 | 929 | 9  | 136 | 503 | 98  | 170 |
| 1        | 76     | 890  | 226 | 6   | 4   | 240 | 16  | 8   | 0   | 22  |
| 2        | 3      | 32   | 434 | 1   | 0   | 5   | 6   | 0   | 5   | 56  |
| 3*       | 2      | 0    | 0   | 55  | 0   | 0   | 1   | 0   | 0   |     |
| 4        | 1      | 0    | 0   | 0   | 262 | 151 | 424 | 49  | 0   | 0   |
| 5*       | 0      | 1    | 3   | 7   | 150 | 5   | 4   | 0   | 0   |     |
| 6        | 66     | 34   | 90  | 79  | 712 | 231 | 2778| 648 | 93  | 183 |
| 7        | 331    | 141  | 243 | 541 | 1131| 394 | 3607| 5342| 187 | 1295|
| 8        | 65     | 54   | 80  | 207 | 109 | 14  | 421 | 505 | 62,784| 168 |
| 9*       | 0      | 1    | 2   | 0   | 0   | 0   | 0   | 2   | 0   | 39  |

### Table 25  Confusion matrix: NB15 RURO

| NB15 RURO | Predicted label |
|-----------|-----------------|
|           | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
| True label |     |     |     |     |     |     |     |     |     |     |
| 0        | 654,173 | 4790 | 2057 | 655 | 1156 | 280 | 161 | 1255 | 548 | 718 |
| 1        | 0      | 982  | 192  | 1   | 1   | 245 | 14  | 19  | 5   | 29  |
| 2        | 0      | 29   | 443  | 0   | 0   | 6   | 3   | 1   | 3   | 57  |
| 3*       | 0      | 0    | 1    | 55  | 0   | 0   | 1   | 0   | 1   |     |
| 4        | 1      | 2    | 1    | 0   | 251 | 154 | 452 | 26  | 0   | 0   |
| 5*       | 0      | 2    | 3    | 7   | 1   | 151 | 5   | 4   | 0   | 0   |
| 6        | 3      | 70   | 100  | 52  | 612 | 259 | 2905| 664 | 45  | 204 |
| 7        | 3      | 283  | 190  | 334 | 1447| 419 | 3769| 5352| 73  | 1342|
| 8        | 1      | 83   | 132  | 63  | 92  | 18  | 441 | 591 | 591 | 301 |
| 9*       | 0      | 2    | 1    | 0   | 0   | 0   | 1   | 0   | 0   | 40  |

### Table 26  Confusion Matrix: NB15 RU-SMOTE

| NB15 RU-SMOTE | Predicted label |
|---------------|-----------------|
|               | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
| True label    |     |     |     |     |     |     |     |     |     |     |
| 0             | 653,429 | 5917 | 1946 | 229 | 1033 | 89 | 172 | 1605 | 687 | 686 |
| 1             | 0      | 1018 | 176  | 0   | 2   | 244 | 13  | 7   | 4   | 24  |
| 2             | 0      | 36   | 447  | 0   | 0   | 7   | 3   | 1   | 3   | 45  |
| 3*            | 0      | 0    | 1    | 55  | 0   | 0   | 1   | 0   | 1   |     |
| 4             | 0      | 3    | 4    | 0   | 298 | 152 | 391 | 39  | 0   | 0   |
| 5*            | 2      | 1    | 2    | 7   | 2   | 148 | 6   | 5   | 0   | 0   |
| 6             | 8      | 87   | 129  | 33  | 909 | 239 | 2581| 640 | 92  | 196 |
| 7             | 8      | 326  | 340  | 201 | 1574| 473 | 3305| 5689| 100 | 1196|
| 8             | 1      | 92   | 141  | 50  | 151 | 23  | 367 | 533 | 62,760| 289 |
| 9*            | 0      | 2    | 0    | 0   | 0   | 1   | 0   | 1   | 0   | 40  |
Confusion matrices for LINSW-NB15

Tables 22, 23, 24, 25, 26 and 27 show the confusion matrices using the various resampling methods for the AWS runs on Spark. The predicted label vs the true labels are shown. The categories that had a really low number of instances are marked with an asterisk and the increases in the minority data identification are in italics.

Observations and discussion

- The micro precision, micro recall and micro F1 score were showing very artificially high numbers as well as almost the same results for NR as well as RO (Fig. 8), hence were not considered useful for any further analysis.
- There is almost no overall significant difference between the ANN classification results on AWS and ANN classification results on the local machine in terms of the macro precision, macro recall, and macro F1 score. After oversampling though, it took longer to run on the local machine than on AWS.
- When the minority data is increased by oversampling or majority data is decreased by undersampling, both the macro precision and macro recall increase, though resampling improves the macro recall significantly. Macro precision increasing implies that the ratio of the false positive to true positive is going down, and the macro recall increasing implies that the ratio of the false negative to true positive is going down. So, for this set of experiments, the true positives went up.
- The confusion matrices also show an increase in the number of correctly classified cases for the very low minority classes (shown with asterisk) with resampling (results are in italics in Tables 22, 23, 24, 25, 26 and 27). Though RU did not perform as well as the other resampling measures, it was at least better than NR (though very
slightly). RURO performed the best, though the other resampling options like RO, RU-SMOTE and RU-ADAYSN performed almost as well.

- With respect to training time, on the local machine, except for undersampling, the training time went up in all scenarios of oversampling. But, on AWS, except for RO, the training time went significantly down. And of course, comparing the local machine to AWS, AWS had a lot lower training time in all cases.
- From Table 20 (AWS), it can be observed that RURO and RU-SMOTE's macro recall were the highest, and very close, at 74.4% and 74.6% respectively. RU's macro recall (42%) was lower than the recall of the other resampling methods, but a lot better than NR (32%).
- From Table 12 (local machine), it can be observed that, RURO's macro recall was the highest at 77.5% and RU-SMOTE and RU-ADASYN's macro recall were pretty close, at 76.4%. Again, RU had the lowest macro recall of the all the resampling methods (50%), but performed better than NR (41%).
Table 29  ANN classification results for UNSW-NB18 (BoT-IoT) on AWS with Spark for various resampling methods

| UNSW-NB18 | NR       | RU       | RO       | RURO      | RU-SMOTE | RU-ADASYN |
|-----------|----------|----------|----------|-----------|----------|-----------|
| Macro precision | 0.571675 | 0.420429 | 0.626432 | 0.460293 | 0.496489 | 0.465322 |
| Macro recall   | 0.451428 | 0.544637 | 0.824786 | 0.858747 | 0.858425 | 0.786073 |
| Macro F1 score | 0.466182 | 0.458041 | 0.586201 | 0.533695 | 0.561807 | 0.509871 |
| Training time (s) | 426      | 84       | 426      | 84        | 84       | 84        |

Table 30  ANN classification results for UNSW-NB18 (BoT-IoT) on local machine for various resampling methods

| UNSW-NB18 | NR       | RU       | RO       | RURO      | RU-SMOTE | RU-ADASYN |
|-----------|----------|----------|----------|-----------|----------|-----------|
| Macro precision | 0.607171 | 0.651555 | 0.707524 | 0.548372 | 0.630532 | 0.473781 |
| Macro recall   | 0.57664  | 0.636077 | 0.862225 | 0.881982 | 0.887849 | 0.767004 |
| Macro F1 score | 0.585591 | 0.608576 | 0.7241   | 0.621413 | 0.688235 | 0.547812 |
| Training Time (s) | 1696.188 | 424.5469 | 910.6094 | 653.6719 | 574      | 635.2813 |

Fig. 11  ANN classification results for UNSW-NB18 (BoT-IoT) on AWS with Spark for various resampling methods

Experimentation on UNSW-NB18

The first section presents the re-sampling of UNSW-NB18 and then the classification results are presented. An analysis of the UNSW-NB18 results are presented in the observations and discussions section.

Resampling UNSW-NB18  Table 28 presents the number of samples before resampling, after RU, after RO, and after RURO and Fig. 10 graphically presents the data before resampling, after RU, and after RO. Before Resampling represents 70% of the original UNSW-NB18 dataset, which was used for training the model. From Table 28 it can be noted that Data Exfiltration and Keylogging had only 4 and 50 instances respectively before resampling, so oversampling makes a big difference for these two attacks. With RU, mainly the number of TCP and UDP attacks, which had the most instances, was reduced. But overall, with RU as well as with RO, the data was still imbalanced. TCP and UDP still have a lot more instances.
### Table 31 Confusion matrix: NB18 NR

| True label | Predicted label | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|------------|----------------|---|---|---|---|---|---|---|---|
| 0          | 66             | 0  | 0 | 0 | 0 | 4 | 52| 6 | 4 |
| 1*         | 0              | 0  | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| 2          | 0              | 0  | 86| 0 | 30| 155| 471| 0 |
| 3*         | 0              | 0  | 0 | 0 | 0 | 19| 3 | 1 | 0 |
| 4          | 0              | 0  | 17| 0 | 144| 4942| 193| 1 |
| 5          | 11             | 0  | 49| 0 | 102| 21,397| 475| 0 |
| 6          | 0              | 0  | 8 | 0 | 49| 1246| 476,116| 0 |
| 7          | 0              | 0  | 0 | 0 | 0 | 2 | 11| 594,894 |

### Table 32 Confusion matrix: NB18 RU

| True label | Predicted label | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|------------|----------------|---|---|---|---|---|---|---|---|
| 0          | 73             | 0  | 0 | 0 | 0 | 2 | 52| 2 | 3 |
| 1*         | 2              | 0  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2          | 3              | 0  | 412| 0 | 76| 208| 43 | 0 |
| 3*         | 6              | 0  | 0 | 0 | 0 | 17| 0 | 0 | 0 |
| 4          | 4              | 0  | 18 | 0 | 1599| 3549| 127 | 0 |
| 5          | 21             | 0  | 17 | 0 | 435| 21,348| 213 | 0 |
| 6          | 87             | 0  | 2042| 0 | 5624| 2528| 467,123 | 15 |
| 7          | 131            | 0  | 0 | 0 | 0 | 64| 17 | 594,695 |

### Table 33 Confusion matrix: NB18 RO

| True label | Predicted label | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|------------|----------------|---|---|---|---|---|---|---|---|
| 0          | 124            | 6  | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1*         | 0              | 2  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2          | 0              | 1  | 627| 16| 0 | 44| 54 | 0 |
| 3*         | 3              | 0  | 1 | 19 | 0 | 0 | 0 | 0 | 0 |
| 4          | 6              | 2  | 54 | 11| 273| 4241| 710 | 0 |
| 5          | 47             | 3  | 298| 39| 14| 20,732| 901 | 0 |
| 6          | 10             | 8  | 233| 112| 20| 1829| 475,207 | 0 |
| 7          | 1              | 0  | 0 | 0 | 0 | 0 | 13 | 594,893 |

Classification results for UNSW-NB18 (BoT-IoT) Table 29 presents the ANN classifi-
Table 34  Confusion matrix: NB18 RURO

| NB18 RURO     | Predicted label |
|---------------|-----------------|
|               | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   |
| True label    |     |     |     |     |     |     |     |     |
| 0             | 129 | 2   | 0   | 0   | 0   | 0   | 0   | 1   |
| 1*            | 0   | 2   | 0   | 0   | 0   | 0   | 0   | 0   |
| 2             | 3   | 0   | 701 | 13  | 0   | 24  | 1   | 0   |
| 3*            | 2   | 0   | 1   | 20  | 0   | 0   | 0   | 0   |
| 4             | 8   | 0   | 53  | 12  | 792 | 4363| 66  | 3   |
| 5             | 42  | 9   | 403 | 35  | 268 | 21,054| 220 | 3   |
| 6             | 206 | 0   | 1385| 126 | 2944| 7220| 464,735| 803 |
| 7             | 16  | 0   | 5   | 0   | 4   | 24  | 594,858|     |

Table 35  Confusion matrix: NB18 RU-SMOTE

| NB18 RU-SMOTE | Predicted label |
|---------------|-----------------|
|               | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   |
| True label    |     |     |     |     |     |     |     |     |
| 0             | 129 | 2   | 0   | 0   | 0   | 0   | 0   | 1   |
| 1*            | 0   | 2   | 0   | 0   | 0   | 0   | 0   | 0   |
| 2             | 4   | 0   | 668 | 7   | 0   | 61  | 2   | 0   |
| 3*            | 2   | 1   | 1   | 19  | 0   | 0   | 0   | 0   |
| 4             | 4   | 1   | 4   | 15  | 1213| 3951| 109 | 0   |
| 5             | 41  | 11  | 335 | 39  | 343 | 21,115| 146 | 4   |
| 6             | 80  | 0   | 1239| 90  | 1939| 7090| 466,243| 738 |
| 7             | 18  | 1   | 0   | 22  | 0   | 5   | 17  | 594,844|

Table 36  Confusion Matrix: NB18 RU-ADASYN

| NB18 RU-ADASYN | Predicted label |
|---------------|-----------------|
|               | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   |
| True label    |     |     |     |     |     |     |     |     |
| 0             | 123 | 2   | 0   | 2   | 0   | 2   | 1   | 2   |
| 1*            | 0   | 2   | 0   | 0   | 0   | 0   | 0   | 0   |
| 2             | 1   | 0   | 596 | 13  | 0   | 132 | 0   | 0   |
| 3*            | 2   | 0   | 13  | 8   | 0   | 0   | 0   | 0   |
| 4             | 17  | 2   | 39  | 11  | 1385| 3731| 107 | 5   |
| 5             | 55  | 10  | 101 | 37  | 152 | 21,517| 160 | 2   |
| 6             | 184 | 0   | 3785| 84  | 1314| 9054| 462,293| 705 |
| 7             | 6   | 0   | 0   | 40  | 0   | 5   | 315 | 594,541|

cation results for UNSW-NB18 (BoT-IoT) on AWS using Spark and Table 30 presents the ANN classification results for UNSW-NB18 (BoT-IoT) on the local machine with Scikit-Learn. The results of macro precision, macro recall and macro F1 score are presented for NR, RU, RO, RURO, RU-SMOTE RU-ADASYN for UNSW-NB18. The training time for the model was also recorded in Tables 29 and 30 respectively. Figure 11 presents
the graphical results of the different resampling methods using Spark. The results of the different resampling methods on the local machine show a similar trend, hence are not presented.

Confusion matrices for UNSW-NB18
Tables 31, 32, 33, 34, 35 and 36 show the confusion matrices using the various resampling methods for the AWS runs on Spark. The predicted label vs the true labels are shown. The categories that had a really low number of instances are marked with an asterisk and the increases in the minority data identification are in italics.

Observations and discussion

- There is almost no overall significant difference between the ANN classification results on AWS and ANN classification results on the local machine in terms of the macro precision, macro recall, and macro F1 score. After oversampling though, it took longer to run on the local machine than on AWS.
- When the minority data is increased by oversampling or the majority data is decreased by undersampling, the macro recall or ADT increases. Oversampling improves the macro recall significantly. The macro precision went up in only one case, in the case of RO. In all other cases, the macro precision decreased. Macro precision decreasing implies that the ratio of the false positive to true positive is going up. Since the macro recall increased, this implies that the ratio of the false negative to true positive is going down. So, in these set of experiments, it can be concluded that the false positives went up and false negatives went down.
- The confusion matrices also show an increase in the number of correctly classified cases for the very low minority classes (shown with asterisk) with RO, RURO, RU-SMOTE, and RU-ADASYN (results are in italics in Tables 31, 32, 33, 34, 35 and 36), though the latter did not do as well as the earlier three resampling methods. It can be noted from Table 32 that RU did not have any effect on these results.
it can be noted that Data_Exfiltration and Keylogging still have very small number of attacks for RU, which is why the ANN classifier could not train effectively for RU.

- Using Spark, only with RO, the training time was the same as the benchmark (which had no resampling), but in all other resampling cases the training time decreased. On the local machine, all cases of resampling had lower times. But again, Spark took a lot less time for training the model than the local machine.
- From Table 29 (AWS), it can be observed that RURO and RU-SMOTE’s macro recall were the highest, and very close, at 85.87% and 85.84% respectively. In this case RU-ADASYN did not perform as well as RURO or RU-SMOTE. RU’s macro recall (54.46%) was lower than the recall of the other resampling methods, but a lot better than NR (45.14%).
- From Table 30 (local machine), it can be observed that, RURO’s macro recall was the highest at 88.78% and RU-SMOTE’s macro recall was pretty close, at 88.1%. In this case, too, RU-ADASYN did not perform as well as RURO or RU-SMOTE. RU,

| Table 38 ANN classification results for NB17-Ecobee on AWS with Spark for various resampling methods |
|--------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|
| NB17-Ecobee | NR | RU | RO | RURO | RU-SMOTE | RU-ADASYN |
|--------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|
| Macro precision | 0.825617 | 0.828213 | 0.805486 | 0.859248 | 0.81543 | 0.798424 |
| Macro recall | 0.874221 | 0.87087 | 0.852089 | 0.853259 | 0.859054 | 0.836491 |
| Macro F1 score | 0.842014 | 0.831575 | 0.821682 | 0.812984 | 0.817154 | 0.793221 |
| Training time (s) | 390 | 198 | 432 | 198 | 198 | 198 |

| Table 39 ANN classification results for NB17-Ecobee on local machine for various resampling methods |
|--------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|
| NB17-Ecobee | NR | RU | RO | RURO | RU-SMOTE | RU-ADASYN |
|--------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|-------------------------------------------------|
| Macro precision | 0.929664 | 0.928654 | 0.951042 | 0.928425 | 0.938275 | 0.909 |
| Macro recall | 0.907989 | 0.905878 | 0.908236 | 0.906522 | 0.904341 | 0.905353 |
| Macro F1 score | 0.876052 | 0.87639 | 0.879551 | 0.872209 | 0.876632 | 0.875529 |
| Training time (s) | 2356.359 | 948.5313 | 1985.313 | 1589.625 | 1427.734 | 1014.813 |
again, had the lowest macro recall of the all the resampling methods (63.6%), but performed better than NR (57.6%).

Experimentation on NB17-Ecobee

The first section presents the re-sampling of NB17-Ecobee and then the classification results are presented. An analysis of the NB17-Ecobee results are presented in the observations and discussions section.

Resampling NB17-Ecobee

Table 37 presents the number of samples before resampling, after RU, after RO, and after RURO and Fig. 12 graphically presents the data before resampling, after RU, and after RO. The Before Resampling column represents 70% of the original NB17-Ecobee dataset, which was used for training the model. Figure 12 shows the imbalance in the data before resampling. In this dataset there were a lower number of benign cases (lower than any of the attacks), and there were no attacks with extremely low number of cases. After RU, the data were more balanced than before resampling, but RO seemed to give the same pattern as before resampling, and the data was balanced for each category with RURO, hence this category was not shown in Fig. 12.

Classification results for NB17-Ecobee

Table 38 presents the ANN classification results for NB17-Ecobee on AWS using Spark and Table 39 presents the ANN classification results for NB17-Ecobee on the local machine with Scikit-Learn. The results of macro precision, macro recall and macro F1 score are presented for NR, RU, RO, RURO, RU-SMOTE, and RU-ADASYN for NB17-Ecobee. The training time (in seconds) for the model was also recorded in Tables 38 and 39 respectively. Figure 13 presents the graphical results of the different resampling methods using Spark. The results of the different resampling methods on the local machine show a similar trend, so they are not presented graphically.

Observations and discussion

- Resampling does not seem to have any effect on macro precision, macro recall or macro F1 score in this dataset. In fact, on AWS, Table 38, it can be observed that
### Table 40 Resampling of NB17-Danmini

| NB17-Danmini | Number of cases for training model |
|--------------|-----------------------------------|
|              | Before resampling | RU | RO | RURO |
| Benign       | 34,474             | 34,474 | 34,474 | 34,474 |
| Mirai_ack    | 1                   | 71,551 | 34,474 | 71,551 | 34,474 |
| Mirai_scan   | 2                   | 75,375 | 34,474 | 75,375 | 34,474 |
| Mirai_syn    | 3                   | 85,704 | 34,474 | 85,704 | 34,474 |
| Mirai_udp    | 4                   | 166,466| 34,474 | 166,466| 34,474 |
| Mirai_udpplain | 5              | 57,708 | 34,474 | 57,708 | 34,474 |
| Gafgyt_combo | 6                   | 41,821 | 34,474 | 41,821 | 34,474 |
| Gafgyt_junk  | 7                   | 20,254 | 34,474 | 20,254 | 34,474 |
| Gafgyt_scan  | 8                   | 20,931 | 34,474 | 20,931 | 34,474 |
| Gafgyt_tcp   | 9                   | 64,280 | 34,474 | 64,280 | 34,474 |
| Gafgyt_udp   | 10                  | 74,244 | 34,474 | 74,244 | 34,474 |

![Resampling NB17-Danmini: Number of Cases](image)

**Fig. 14** Resampling of NB17-Danmini

### Table 41 ANN Classification results for NB17-Danmini on AWS with Spark for various resampling methods

| NB17-Danmini | NR | RU | RO | RURO | RU-SMOTE | RU-ADASYN |
|--------------|----|----|----|------|----------|-----------|
| Macro precision | 0.792128 | 0.812195 | 0.805464 | 0.816337 | 0.824546 | 0.830129 |
| Macro recall   | 0.842704 | 0.861566 | 0.852281 | 0.872596 | 0.87531 | 0.871693 |
| Macro F1 score | 0.806466 | 0.821549 | 0.816367 | 0.82951 | 0.838391 | 0.844219 |
| Training time (s) | 576 | 270 | 594 | 288 | 282 | 288 |

### Table 42 ANN classification results for NB17-Danmini on local machine for various resampling methods

| NB17-Danmini | NR | RU | RO | RURO | RU-SMOTE | RU-ADASYN |
|--------------|----|----|----|------|----------|-----------|
| Macro precision | 0.924668 | 0.945596 | 0.940988 | 0.935772 | 0.925392 | 0.942088 |
| Macro recall   | 0.903897 | 0.905597 | 0.905295 | 0.905451 | 0.905824 | 0.905248 |
| Macro F1 score | 0.875883 | 0.871401 | 0.875896 | 0.870879 | 0.876621 | 0.871806 |
| Training time (s) | 896.3594 | 1625.172 | 1095.422 | 662.0625 | 745.3281 | 1158.844 |
NR and RU performed better than the other resampling methods. On the local machine however, Table 39, NR and the other resampling measures seemed to give almost the same percentage for macro recall.

- Except for RO, the training time is lower than the benchmark in all other cases. And of course, the training time on the local machine is higher than AWS.

**Experimentation on NB17-Danmini**

The first section presents the resampling of NB17-Danmini and then the classification results are presented. An analysis of the NB17-Danmini results are presented in the observations and discussions section.

**Resampling NB17-Danmini**

Table 40 presents the number of samples before resampling, after RU, after RO, and after RURO and Fig. 14 graphically presents the data before resampling, after RU, and after RO. Before Resampling represents 70% of the original NB17-Danmini dataset, which was used for training the model. Figure 14 shows the imbalance in the data before resampling. In this dataset, Gafgyt_junk and Gafgyt_scan had a lower number of cases, but the number of cases were not as low as some of the extremely low number of attacks in KDD99, UNSW-NB15 or UNSW-NB18. After RU the data was more balanced than before resampling, but RO seemed to give the same pattern as before resampling, and the data was balanced for each category with RURO, hence this latter category was not shown in Fig. 14.

**Classification results for NB17-Danmini**

Table 41 presents the ANN classification results for NB17-Danmini on AWS using Spark and Table 42 presents the ANN classification results for NB17-Danmini on the local machine with Scikit-Learn. The results of macro precision, macro recall and macro F1 score are presented for NR, RU, RO, RURO, RU-SMOTE and RU-ADASYN for NB17-Danmini. The training time (in seconds) for the model was also recorded in Tables 41 and 42 respectively. Figure 15 presents the graphical results of the different resampling methods using Spark. The results of the different resampling methods on the local machine show a similar trend, hence were not presented.

---

**Fig. 15** ANN classification results for NB17-Danmini on AWS with Spark for various resampling methods
Observations and discussion

- Resampling does not seem to have any effect on macro precision, macro recall or macro F1 score in this dataset. On AWS, Table 41, NR had a macro recall of 84% while resampling measures had a macro recall of 85–87%. And, on the local machine however, Table 42, NR and the other resampling measures seemed to give almost the same percentage for macro recall, a little above 90%.
- On AWS, except for RO, the training time went down in all cases. On the local machine, however, the time went up for RU, RO, and RU-ADASYN. And again, overall, it took much longer to run on the local machine.
Experimentation on NB17-Philips

The first section presents the resampling of NB17-Philips and then the classification results are presented. An analysis of the NB17-Philips results are presented in the observations and discussions section.

Resampling NB17-Philips  Table 43 presents the number of samples before resampling, after RU, after RO, and after RURO and Fig. 16 graphically presents the data before resampling, after RU, and after RO. Before Resampling represents 70% of the original NB17-Philips dataset, which was used for training the model. Figure 16 shows the imbalance in the data before resampling. In this dataset, Gafgyt_junk and Gafgyt_scan had a lower number of cases, but again, the cases were not as low as some of the attacks in KDD99, UNSW-NB15 or UNSW-NB18. After RU, the data was more balanced (as shown in Fig. 16), but after RO the pattern of distribution of data closely followed the before resampling. After RURO, the number of cases were balanced for each category, hence this was not included in Fig. 16.

Classification results for NB17-Philips  Table 44 presents the ANN classification results for NB17-Philips on AWS using Spark and Table 45 presents the ANN classification results for NB17-Philips on the local machine with Scikit-Learn. The results of macro precision, macro recall and macro F1 score are presented for NR, RU, RO, RURO, RU-SMOTE and RU-ADASYN for NB17-Philips. The training time (in seconds) for the model was also recorded in Tables 44 and 45 respectively. Figure 17 presents the graphical results of the different resampling methods using Spark. The results of the different resampling methods on the local machine show a similar trend, so it was not presented.

| NB17-Philips | NR  | RU  | RO  | RURO | RU-SMOTE | RU-ADASYN |
|--------------|-----|-----|-----|------|----------|-----------|
| Macro precision | 0.912172 | 0.931224 | 0.930606 | 0.892025 | 0.904524 | 0.891446 |
| Macro recall | 0.893773 | 0.903667 | 0.906043 | 0.899702 | 0.907286 | 0.907512 |
| Macro F1 score | 0.869574 | 0.871905 | 0.877093 | 0.863070 | 0.878986 | 0.879480 |
| Training time (s) | 1385.156 | 834.5156 | 1871.328 | 1172.344 | 1643.234 | 1514.453 |

Fig. 17 ANN classification results for NB17-Philips on AWS with Spark for various resampling methods
Observations and discussion

- Resampling does not seem to have any effect on macro precision, macro recall or macro F1 score in this dataset. From both Tables 44 and 45, it can be observed that almost all the resampling methods performed close to NR, on both AWS and the local machine.

- On AWS, except for RO, the training time went down in all cases. On the local machine, however, the time went up for RO, RU-SMOTE and RU-ADASYN. And again, overall, it took much longer to run on the local machine.

Conclusion

Five different forms of resampling were applied to six different datasets. Three of these datasets can be considered highly imbalanced, and the other three datasets can be considered less imbalanced. The high imbalanced datasets were, KDD99, UNSW-NB15, and UNSW-NB18(BoT-IoT). And, the three UNSW-NB17 datasets can be considered less imbalanced. The following conclusions can be drawn from the resampling:

1. Oversampling increases the training time taken while undersampling decreases the training time taken. This is natural because oversampling increases the number of cases in training data, while undersampling decreases the number of cases in training data.

2. In the highly imbalanced datasets, both oversampling and undersampling increase recall significantly. This means that the ratio of the false negatives to the true positives decreases. So, the ANN model recognized more minority data correctly. And this was also shown by the confusion matrices. In some cases, the macro precision
decreases, which means that the ANN model incorrectly recognized more majority data as minority data.

In some cases, the macro precision decreased, meaning that the ANN model incorrectly recognized some majority data as minority data. A summary of the behavior of oversampling and undersampling the highly imbalanced datasets is presented in Table 46.

With no resampling, micro precision and micro recall were high, but the macro precision and macro recall were relatively lower. This is because although the model recognized almost all majority instances correctly, it recognized minority instances incorrectly, which means that the model recognized most minority instances as belonging to the majority class. This made the macro precision and macro recall relatively lower.

With resampling, however, micro precision and micro recall were still high. The macro recall increases after resampling because the model recognizes more minority instances as the minority class, and this was also reflected in the confusion matrices. However, macro precision decreases after resampling because the model also recognizes some majority instances as minority instances. The number of misrecognitions of majority instances is not relatively large in comparison with the number of majority instances. But the number of misrecognitions of majority instances is relatively large in comparison with the number of minority instances, which decreases the precision of minority classes. So, with resampling, generally, it can be stated that more minority instances were recognized correctly. Table 47 presents a summary of the behavior of the recognizing the minority and majority instances in highly imbalanced datasets.

3. Also, for highly imbalanced datasets, NB15 and NB18, from the confusion matrices it appears that RURO performed the best in terms of identifying minority cases,

| Table 48 Oversampling and Undersampling in not extremely Imbalanced Datasets |
|---------------------------|----------------|----------------|
|                           | Time of training steps | Macro precision | Macro recall |
| Oversampling              | Increase           | Almost unchanged | Almost unchanged |
| Undersampling             | Decrease           | Almost unchanged | Almost unchanged |

| Table 49 Summary for recognizing minority and majority instances in not highly imbalanced datasets |
|---------------------------|----------------|----------------|----------------|----------------|----------------|
|                           | Influence on minority’s recall | Influence on minority’s precision | Influence on majority’s recall | Influence on majority’s precision | Overall Influence on recall and precision |
| Recognize more minority instances correctly | Increase slightly | Increase slightly | No influence | No influence | Almost unchanged |
| Recognize more majority instances incorrectly | No influence | Decrease slightly | Little influence | Little influence | Decrease precision |
| Combined influence | Almost unchanged | Almost unchanged | Little influence | Little influence | Almost unchanged |

3. Also, for highly imbalanced datasets, NB15 and NB18, from the confusion matrices it appears that RURO performed the best in terms of identifying minority cases,
though in some cases this was only a small improvement above RU-SMOTE and RU-ADASYN. For KDD99, RURO and RU-SMOTE can be considered to have performed equally well in identifying minority cases.

4. For highly imbalanced datasets, KDD99, NB15 and NB18, in most cases, the RURO and RU-SMOTE performed the best, in terms of macro recall. RU usually did not perform as well as the other resampling measures in terms of macro recall, but performed better than NR. And RO always performed better than RU in terms of macro recall, and sometimes it was comparable to RURO, RU-SMOTE, and RU-ADASYN.

5. If the data is not extremely imbalanced, for example, NB17, resampling makes no difference, as shown in Table 48.

This could be because:

i. Since the data set is not extremely imbalanced, majority data does not have a very strong influence on the model. Minority data has enough influence on the model, hence the model can classify minority data well.

ii. Imbalance may not be the reason for the inaccuracy. Resampling improves the accuracy by reducing the extent of imbalance. If the inaccuracy is not caused by the imbalance, resampling will not be able to improve the accuracy.

Table 49 presents a summary of behavior of recognizing the minority and majority instances in not highly imbalanced datasets.

Abbreviations
AWS: Amazon's Web Service; ANN: Artificial Neural Networks; ADR: Attack Detection Rate; SMOTE: Synthetic Minority Oversampling Technique; ADASYN: Adaptive synthetic sampling method; MLlib: Machine learning library; NR: No resampling; RU: Random undersampling; RO: Random oversampling; RU-RO: Random undersampling and random oversampling; RU-SMOTE: Random undersampling and SMOTE; RU-ADASYN: Random undersampling and ADASYN; TP: True positive; TN: True negative; FP: False positive; FN: False negative; SVM: Support vector machines.

Acknowledgements
This work has been partially supported by the Askew Institute of the University of West Florida.

Author’s contributions
This work was conceptualized by both authors. The programming was done by KL, and both authors participated in the write-up of the paper. Both authors read and approved the final manuscript.

Funding
This work is not funded

Availability of data and materials
Not applicable

Competing interests
None of the authors have any competing interests.

Received: 23 August 2020   Accepted: 4 December 2020
Published online: 06 January 2021

References
1. Abdi L, Sattar H. To combat multi-class imbalanced problems by means of over-sampling techniques. IEEE Trans Knowl Data Eng. 2016;28(1):238–51.
2. Amin A, Anwar S, Adnan A, Navaz M, Howard N, Quadir J, Havalah A, Hussain A. Comparing oversampling techniques to handle the class imbalance problem: a customer churn prediction case study. IEEE Access. 2016;4:7940–57. https://doi.org/10.1109/ACCESS.2016.2619719.
3. Basgall MJ, Hasperué W, Naiouf M, Fernández A, Herrera F. SMOTE-BD: An exact and scalable oversampling method for imbalanced classification in big data. J Comput Sci Technol. 2018;18(03):e23. https://doi.org/10.24215/1666-038.18.e23.

4. Blagus R, Lusa L. SMOTE for High-dimensional class-imbalanced data. BMC Bioinf. 2013; 14:106.

5. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: synthetic minority over sampling technique. J Artif Intell Res. 2002;16:321–57.

6. Cieslak, D. A., Chawla, N. W., & Striiegel, A. (2006). Combating Imbalance in Network Intrusion Datasets. Proc IEEE Int Conf Granular Computing, 2006, Atlanta, Georgia, USA, 732-737.

7. Douzas G, Bacao F. Effective data generation for imbalanced learning using conditional generative adversarial networks. Exp Syst Appl. 2018;91:464–71.

8. Douzas G, Bacao F, Last F. Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE. Inf Sci. 2018;465:1–20.

9. Ertelkin CS. Adaptive oversampling for imbalanced data classification. In: Proceedings of the 28th international symposium on computing and information sciences. 2013, 264:261–9.

10. Ertelkin SE, Huang J, Bottou L, Giles CL. Learning on the border: active learning in imbalanced data classification. In: Proceedings of ACM Conference on information and knowledge management, Lisbon, Portugal, 2007, 127–36.

11. Fernandez A, Rio S, Chawla N, Herrera F. An insight into imbalanced Big Data classification: outcomes and challenges. Complex Intell Syst. 2017;3:105–20.

12. Guller M. Big data analysis with spark. New York: Apress; 2015.

13. Gutierrez PD, Lastra M, Benitez JM, Herrasa F. SMOTE-GPU: big data preprocessing on commodity hardware for imbalanced classification. Prog Artif Intell. 2017;6:347–54. https://doi.org/10.1007/s13748-017-0128-2.

14. He H, Bai Y, Garcia EA, Li S. ADASYN: adaptive synthetic sampling approach for imbalanced learning. In: IEEE international joint conference on neural networks (IEEE world congress on computational intelligence), 2008, p.1322–8.

15. He H, Garcia EA. Learning from imbalanced data. IEEE Trans Knowl Data Eng. 2009;21(9):1263–84.

16. Hulse JV, Khoshgoftaar TM, Napolitano A. Experimental perspectives on learning from imbalanced data. In: Proceedings of the 24th international conference on machine learning, Corvallis, Oregon: Oregon State University, 2007, p. 935–42.

17. Johnson JM, Khoshgoftaar TM. Survey on deep learning with class imbalance. J Big Data. 2019;6:27. https://doi.org/10.1186/s40537-019-0192-5.

18. Koroniots N, Moustafa N, Stithkova E, Turnbull B. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: bot-iot dataset. Fut Gener Comput Syst. 2019;100:779–96. arXiv:1811.00701 v1.

19. LeeV YL, Khoshgoftaar TM, Bauder RA, Seliya N. A survey on addressing high-class imbalance in big data. J Big Data. 2018;5:42. https://doi.org/10.1186/s40537-018-0151-6.

20. Lematre G, Nogueira F, Andias CK. Imbalanced-learn: a Python toolbox to tackle the curse of imbalanced datasets in machine learning. J Mach Learn Res. 2017;18:1–5.

21. Luque A, Carrasco A, Martin A, Heras de las A. The impact of class imbalance in classification performance metrics based on the binary confusion matrices. Pattern Recogn. 2019;192:6–31. https://doi.org/10.1016/j.patcog.2019.02.023.

22. Meidan Y, Bohadana M, Mathov Y, Minsky Y, Breitenbacher D, Shabtai A, Elovic N. N-BaIoT: network-based detection of iot botnet attacks using deep autoencoders. IEEE Pervas Comput. 2018;13(9):1–8.

23. Minsky Y, Doitshman T, Elovic N, Shabtai AJ. Kitsune: an ensemble of autoencoders for online network intrusion detection. In: Network and distributed systems security symposium. 2018.

24. Mohri M, Rostamizadeh A, Talwalkar A. Foundations of machine learning. 2nd ed. Cambridge: MIT Press; 2018.

25. More A. Survey of resampling techniques for improving classification performance in unbalanced datasets. 2018.

26. Moustafa N, Slay J. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). MilCIS. 2015;2015:1–6.

27. Radivojac P, Chawla NV, Dunker AK, Obradovic Z. Classification and knowledge discovery in protein databases. J Biomed Inform. 2004;37(4):224–39. https://doi.org/10.1016/j.jbi.2004.07.008.

28. Raghuwanshi BS, Shukla S. SMOTE based class-specific extreme learning machine for imbalanced learning. Pattern Anal Appl. 2020;187:104814.

29. Song Q, Guo Y, Shepperd M. A comprehensive investigation of the role of imbalanced learning for software defect prediction. IEEE Trans Software Eng. 2019;45(12):1253–69. https://doi.org/10.1109/TSE.2018.2836442.

30. Terzi DS, Sagiroglu S. A new big data model using distributed cluster-based resampling for class-imbalance problem. Appl Comput Syst. 2019;24(2):104–10. https://doi.org/10.1007/s13748-acss-2019-0013.

31. Triguero I, Galar M, Merino D, Salas J, Herrera F. Evolutionary undersampling for extremely imbalanced Big Data classification under Apache Spark. In. 2016 IEEE congress on evolutionary computation (CEC), Vancouver, BC, 2016, p. 640–7. https://doi.org/10.1109/cec.2016.7743853.

32. Wallace B, Small K, Brodley C, Trikalinos T. Class imbalance, redux. In: IEEE 11th international conference on data mining (ICDM), Vancouver, Canada, 2011, p. 754–63.

33. Wang J, Xu M, Wang H, Zhang J. Classification of imbalanced data by using the smote algorithm and locally linear embedding. In: Proceedings of the 8th international conference on signal processing, 2006, p. 1–4.

Publisher’s Note
Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.