IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 29 June 2024, accepted 28 July 2024, date of publication 31 July 2024, date of current version 9 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3436056

RESEARCH ARTICLE

Machine Learning-Based Elephant Flow Classification on the First Packet

PIOTR JURKIEWICZ, BARTOSZ KĄDZIOŁKA, MIROSŁAW KANTOR, JERZY DOMŻAŁ, AND ROBERT WÓJCIK

Institute of Telecommunications, AGH University of Krakow, 30-059 Kraków, Poland

Corresponding author: Piotr Jurkiewicz (piotr.jurkiewicz@agh.edu.pl)

This work was supported by the Project "Precise and Reusable Models of Flow and Flowlet Properties in the Internet" funded by the Polish National Science Centre under Project 2023/49/N/ST7/00532.

ABSTRACT In this paper, we explore the applicability of selected machine learning models to classify incoming flows as elephants or mice on the first packet, using Internet Protocol (IP) and transport layer headers (5-tuple). We show that traditional metrics such as accuracy or F1-score are inadequate for assessing performance in traffic engineering (TE) and quality of service (QoS) applications unless compared at the same traffic coverage. Among the classifiers analyzed, Histogram-based Gradient Boosting with octets-transformed input data provides the best performance, reducing flow operations by a factor of 36.49 and the average number of flow table entries by 16.35, while covering 80% of the traffic.

INDEX TERMS Flows, elephant, classification, traffic engineering, sdn.

I. INTRODUCTION

In the past years, flow-based traffic engineering became a promising solution to handle continuously increasing network demand [1], [2], [3]. In flow-based routing, an individual entry containing the next hop is assigned to each flow in the switch memory. This opens the possibility to route different flows between the same endpoints using distinct paths, bringing multipath load-balancing capabilities. Moreover, paths for subsequent flows can be chosen based on the current or predicted network load distribution, adaptively bypassing overloaded links. Adaptive routing of flows also has greater stability compared to dynamic load balancing in the classic, IP prefix-based, routing [4].

However, the number of simultaneous flows in the network far exceeds the capacity of flow tables in switches [5]. Controller's throughput to handle new flows can also be limited. One solution to these issues could be to create individual entries only for the largest flows, while handling the majority of smaller flows using default approach. This would significantly reduce the number of entries in the tables while still ensuring coverage of a substantial portion of traffic by individual flow-specific entries. These intense flows are referred to as *elephant flows*, while the remaining flows are termed *mouse flows*. Actual distributions of flow lengths and sizes are even more long-tailed than the *Pareto* rule suggests (80/20). Recent analyses indicate that only 0.2-0.4% of flows account for 80% of the total traffic [6], [7].

The challenge remains in how to detect the largest flows. Ideally, flows should be classified along with their first packet to avoid mid-connection reroutings, which can impede transport protocols path state estimations. First packet classification can also reduce controller's load. Flow classification is also used for improving quality of service (QoS), both inside datacenters [8], [9] and in wide-area communication, including inter-datacenter networks [10]. In many QoS applications it is also required to classify flows right from the beginning to redirect them to an appropriate queue or path. Earlier classification also allows a greater share of flow's traffic to be subject of flow type-specific treatment. Such classification can base solely on information contained in packet headers.

There are numerous studies on flow classification, yet most focus on classifying flows not on the first packet, but after observing a number of initial packets. Additionally, existing studies overlook metrics relevant to traffic engineering and QoS. They typically evaluate performance using traditional classification metrics like accuracy or F1-score, which, as we

The associate editor coordinating the review of this manuscript and approving it for publication was Maurizio Casoni.

demonstrate in this paper, are not necessarily good indicators of model performance for the aforementioned applications.

Therefore, in this study we aim to address this gap and explore usage of machine learning algorithms to identify large flows basing on 5-tuples (protocol and source and destination IP addresses and ports) contained in the IP and transport headers. The distinctive feature of our paper is that we focus on metrics relevant from the traffic engineering and QoS perspective, namely: the volume of traffic transmitted by flows classified as elephants after their identification (resulting traffic coverage), the reduction in the number of created individual flow entries (flow table operations reduction), and the average reduction in the number of flow entries in tables (average occupancy reduction). We examine four classifier models available in the `scikit-learn` library [11] with three different representations of the input data (raw, octets, and bits) and 25 different elephant flow thresholds (training traffic coverage values).

II. RELATED WORK

The idea of individually managing elephant flows dates back to 1999, as first proposed by Shaikh et al. [12]. However, it was largely theoretical due to hardware limitations at the time. Recently, the concept has gained renewed interest with the advent of software-defined networking (SDN). In SDN, a central controller with a global network view can efficiently handle large flows. The general approach involves installing wildcard entries for shortest paths and monitoring traffic to detect elephant flows. Upon detection, the controller can compute and implement alternative, non-congested paths for these flows to achieve load balancing.

Hedera [13] is a system designed to dynamically reroute significant flows that exceed a certain threshold. These flows are redirected by the controller to dynamically determined paths based on flow statistics collected by edge devices using OpenFlow counters. Similarly, DevoFlow [14] focuses on elephant flows, using sampling methods and thresholds for detection, although it only evaluates the overall network performance as a measure of efficiency. A comparable system is proposed by Xu et al. [15], using a variation of the Bloom filter to detect elephant flows at edge devices.

The methods mentioned above use straightforward techniques like sampling, counters, and thresholds for detecting large flows. However, advanced machine learning techniques have also been explored. Xiao et al. [16] apply a decision tree to identify elephant flows, focusing on detection accuracy, which might not be the best metric for traffic engineering. Poupart et al. [17] evaluate three machine learning methods for predicting flow size and identifying elephant flows using a dataset of three million flows, analyzing true positive and true negative rates.

Liu et al. [18] propose using a random forest of decision trees to select eight features for a classification model, suggesting a two-level architecture with pre-classification at edge devices and precise classification at the central controller. They classify flows into four types (elephant, cheetah, tortoise, porcupine) and evaluate classification precision and delay. Similarly, Hamdan et al. [19] analyze a two-level classification system with initial classification at switches using the count-min sketch algorithm and final classification at the controller with a decision tree. The switch algorithm is periodically retrained on updated datasets from the controller, using real traffic models but primarily focusing on classification precision.

In 2022, He et al. [20] and Qian et al. [21] introduced sketch-based strategies to improve flow table efficiency. He et al. proposed a streamlined single-level approach, while Qian et al. used TCAM-based storage for elephant flow labels to balance accuracy in identifying elephant and mouse flows. Both methods were evaluated using real ISP packet traces, showcasing their practical utility.

Da Silva et al. [22] introduced a predictive model using Locally Weighted Regression (LWR) to estimate new network flow sizes and durations based on past patterns. In 2022, they enhanced flow management with a hashing mechanism inspired by the Cuckoo Search meta-heuristic [23]. Pekar et al. presented a threshold-agnostic heavy-hitter classification system [24], using template matching to identify elephant flows based on initial packet size distribution.

The CrossBal system [25] uses Deep Reinforcement Learning (DRL) for hybrid load balancing, detecting elephant flows with a three-level mechanism involving threshold-based filtering and rerouting. Wassie et al. [26] employed deep learning with autoencoders, gradient boosting, and autoML predictive algorithms like eXtreme gradient boosting (XGBoost) [27] and gradient boosting machine (GBM) [28] for improved flow management.

These studies typically classify flows after observing several initial packets, but our goal is to identify flows as quickly as possible, ideally from the first packet. Early classification avoids mid-route rerouting, which can disrupt transport layer operations and congestion control algorithms.

Flow classification from the first packet is shown by Durner and Kellerer [29], using features from the 5-tuple and the size of the first packet. Hardegen et al. [30] propose multiclass prediction with a deep neural network based on the 5-tuple of the first packet, similar to their earlier work predicting flow bit rate [31]. In 2023, Gomez et al. [32] assessed various machine learning algorithms for first-packet flow classification, focusing on accuracy rather than impacts on flow tables or traffic coverage. In 2024, Xie et al. [33] introduced a two-stage decision tree system for elephant flow classification based on the first packet's headers, developed in P4 but only tested in an emulator.

Recent works using neural networks for flow classification emphasize QoS rather than traffic engineering. Alkhalidi and Yaseen [34] use a one-dimensional convolutional neural network to classify flows based on packet headers, reducing feature count and processing time while maintaining accuracy. Yaseen et al. [35] classify traffic and assign Differentiated Services Code Point (DSCP) fields with a similar approach, tested within an SDN controller in Mininet.

All the aforementioned studies focus on classification accuracy but overlook the effectiveness of algorithms for traffic engineering goals. Misclassifying the largest flows has a greater impact on traffic coverage than misclassifying smaller flows, which is not accounted for in traditional metrics. Additionally, prior studies do not analyze metrics such as flow table entry reduction or post-classification traffic volume, which are crucial for traffic engineering and QoS.

III. METHODOLOGY

To classify a flow based on its first packet, we utilize classifiers provided by the `scikit-learn` library [11]. It is an open source Python machine learning library, which features various classification, regression and clustering algorithms. Classification is a supervised machine learning method that requires labeled input data for training the model. We use binary classification, meaning both the training labels and the model's output are binary decisions (0/1). In our case, these decisions determine whether a flow is a mouse (0) or an elephant (1).

The primary assumption of this study is that we perform the classification (inference) upon the arrival of the first packet of each flow. Flows classified as elephants (decision 1) are then registered in a memory (flow table) and, from that point on, can be subject to individual treatment. This can involve routing through a flow-specific path for traffic engineering or using specific queues for QoS provisioning. Conversely, flows classified as mice (decision 0) are not treated individually. Their packets can be routed through default shortest paths or using default queues. Therefore, there is no need to store individual per-flow states for mice flows. This approach reduces the number of flows requiring individual processing (reducing controller load and the number of flow table operations) and also reduces the number of entries in flow tables.

In an actual implementation, it would be necessary to ensure that only the first packet of each flow is subjected to inference. For Transmission Control Protoco (TCP), the first packet of each flow can be easily identified using the SYN flag. An alternative, protocol-agnostic solution would be to use a Bloom filter to register hashes of flows classified as mice, avoiding repeated classification of subsequent packets in these flows. However, this is an implementation detail. In this paper, we focus on the isolated problem of the performance of machine learning classifiers, which can be then used in various combinations as building blocks of more advanced systems.

A. DATASET

The dataset plays a crucial role in influencing the performance of any machine learning algorithm. To assess machine learning models, we utilize length and size distributions of flows from a dataset collected over 30 days in a big campus network [6]. For data processing, the package [36] was used.

The aforementioned dataset encompasses over 4 billion flows. The complete set of flow records occupies approximately 278 GB in binary format. Therefore, to train and evaluate our models we use an anonymized subset of that data. The subset covers one hour of traffic, which amounts to 6,517,484 flows and 547 GB of transmitted data. The period was carefully selected to ensure both that it was anomaly-free and that the theoretical reduction rate curve of a perfect elephant classifier calculated for flows during that hour closely resembles that of the mixture derived from the entire 30-day dataset. IP addresses in the published dataset were anonymized using the prefix-preserving Crypto-PAn algorithm [37]. The anonymization does not influence the performance of ML models, as shown in [38].

B. INPUT FEATURES

The input data from the 5-tuple includes the following information: source IP address, destination IP address, source transport port, destination transport port, transport layer protocol number – in total 104 bits. We examine three different representations of the input data:

- **raw**: Header fields are not modified. This results in an input vector consisting of five features: source IP, destination IP, source port, destination port, and protocol, all represented as 32-bit integers.
- **octets**: Header fields longer than 8 bits (IP addresses and ports) are divided into separate octets, resulting in 13 features, each represented as an 8-bit integer. This transformation better captures patterns arising from the hierarchical structure of IP addresses, such as similarities in traffic among hosts in the same subnet, while keeping the number of features significantly lower than the bits format.
- **bits**: Header fields are split into individual bits, resulting in an input vector of 104 features, represented as binary values (0/1).

C. TRAINING LABELS

In the case of binary classification, the model output is a binary decision (0/1). In our scenario, this decision determines whether a flow is a mouse or an elephant, indicating whether to add the flow to the table. Therefore, before starting the training phase, an elephant flow size threshold needs to be established to appropriately label the training dataset.

In our experiment, we assumed 25 different values for the elephant size threshold. These thresholds were not defined directly. Instead, we determined thresholds by selecting a percentage of the largest flows from the training set to achieve a specified coverage of the entire network traffic. For these selected flows, the model is trained with a decision of 1, while for the remaining flows, the decision is set to 0. The coverage values used in the training phase were defined by the following equation:

$$coverage = 1 - \frac{1}{1.37972966146121546^{i}} \quad \text{for } i \in \{1, \ldots, 25\}$$

The equation was chosen deliberately to yield exactly 80% training coverage as one of its values and to produce evenly spaced resulting traffic coverages. This resulted in the following training traffic coverage values:

0.275220, 0.474694, 0.619269, 0.724054, 0.800000,
0.855044, 0.894939, 0.923854, 0.944811, 0.960000,
0.971009, 0.978988, 0.984771, 0.988962, 0.992000,
0.994202, 0.995798, 0.996954, 0.997792, 0.998400,
0.998840, 0.999160, 0.999391, 0.999558, 0.999680

To obtain a curve representing flow table reduction as a function of coverage, the entire training and fitting procedure was repeated for all 25 values of training traffic coverage.

It should be noted that the resulting traffic coverage of a classifier on a validation dataset will be lower than the traffic coverage used for training. This is because traffic generated by inaccurately classified elephant flows (false negatives) will not be covered. Therefore, it is necessary to train models using higher traffic coverage values than those desired during their later operation.

D. TRAINING AND VALIDATION

We perform 5-folds cross-validation by partitioning the dataset into 5 consecutive sets. Each set is then used once as a validation set while the remaining 4 sets form the training dataset. This means the training dataset consist of 5,213,988 flows, while the validation sets consist of 1,303,496 flows. Consequently, each algorithm is trained 5 times for each training traffic coverage. Then, its performance is evaluated on the corresponding validation set. Finally, we calculate the mean performance of all algorithms across all 5 folds, along with the standard deviation.

As mentioned in the Section I, Internet flow size distribution is long-tailed, with only 0.2-0.4% of flows accounting for 80% of the total traffic [6], [7]. Therefore, the training data in our problem is significantly imbalanced. For example, in fold 0 of our dataset with an 80% training traffic coverage, the number of elephant flows is only 1,253, whereas the number of mice flows is 1,302,244. In such cases, the resulting model might be biased towards the more dominant class – in our case, mice – thus reducing classification accuracy for elephants. Moreover, many training algorithms assume that the class distribution of the training dataset is balanced, potentially yielding incorrect results for an imbalanced dataset.

The recommended approach to tackle this issue is to perform class balancing before training, either by sampling an equal number of samples from each class or, preferably, by normalizing the sum of the sample weights for each class to the same value. In our initial attempt, we used the `sample_weight` parameter to equalize the sum of sample weights for mice and elephants during training. This indeed improved the performance of all analyzed models. However, we discovered that assigning sample weights equal to the square root of the flow size (number of bytes transmitted) yields significantly better results. The detailed investigation of this phenomenon remains outside the scope of this paper and will be the subject of further research. Consequently, the training of all models presented in this paper is performed with sample weights equal to:

$$sample\_weight = \sqrt{flow\_size} \tag{1}$$

with the exception of the **KNeighborsClassifier**, which does not support training using explicit sample weights.

E. EVALUATION

The existing literature does not examine metrics such as flow table reduction and post-flow classification traffic volume, which are significant for traffic engineering and QoS. Existing studies predominantly concentrate on classification accuracy, measured through parameters like true positive rate, true negative rate, and the accuracy of mouse/elephant binary classification. Unfortunately, these metrics offer limited insights into the efficency of the analyzed algorithms within our research. **Notably, misclassifying the largest flow in the network has a more substantial impact on traffic coverage than misclassifying a smaller flow**. Traditional metrics do not consider this critical distinction. As shown in Section VI with the example of *KNeighborsClassifier*, conventional metrics can be misleading. A classifier with a high TNR can achieve very high accuracy despite being unable to detect elephant flows effectively.

In response to these limitations, we propose the adoption of specific metrics to evaluate machine learning algorithms within the context of elephant flow detection. Specifically, we suggest assessing the reduction in flow operations, average flow table occupancy reduction, and the fraction of traffic covered. It is important to acknowledge the inherent **tradeoff** between the first two metrics and traffic coverage: **increasing the elephant detection threshold improves flow table reduction but simultaneously lowers the fraction of covered traffic**.

In our experiment, we use a dataset of 6,517,484 flows, representing one hour of traffic. For evaluation purposes, we assume a constant flow arrival rate, equal to the average flow arrival rate during the one hour covered by the whole dataset:

$$\text{FPS} = \frac{6517484}{3600} \approx 1810 \text{ [flows per second]} \tag{2}$$

Each validation fold, consisting of 1,303,496 flows, is therefore attributed to 720 seconds (12 minutes). The start times of all flows belonging to the validation set are equally distributed between 0 and 720 seconds of the experiment. We use real flow duration values, as collected in the dataset. To calculate the average flow table occupancy reduction and traffic coverage, we maintain two numeric arrays: `bytes_sent` and `flow_entries`. The length of both arrays is 720, with initial values set to 0. Upon the arrival of a new flow, we perform an inference to determine whether the flow will be an elephant or a mouse. When a flow is

classified as an elephant, we add 1 to all `flow_entries` fields between the second of flow start and the moment of the flow end plus timeout. We assume the same flow inactive timeout as used in the NetFlow collector when gathering the dataset, that is 15 seconds:

$$flow\_entries_t \mathrel{+}= 1 \quad \forall \quad start \le t \le end + timeout \tag{3}$$

For flows classified as elephants, we also calculate their average rate by dividing the flow size (number of bytes) by the flow duration. We then add the average number of bytes per second (`avg_bps`) transmitted by the flow to all seconds in the `bytes_sent` array during its lifetime:

$$bytes\_sent_t \mathrel{+}= avg\_flow\_bps \quad \forall \quad start \le t \le end \tag{4}$$

We determined that, with the flow arrival rate used in our experiments, approximately 5 minutes are required to achieve stable values of average coverage and occupancy reduction metrics. Therefore, we assume a warm-up time of 300 seconds. This means that to calculate metric values, we use values for seconds in the range between 300 and 720.

Below we provide the formal definition of these metrics:

1) RESULTING AVERAGE COVERAGE

This metric measures the coverage of traffic (number of bytes sent) in the network by flows classified as elephants during the analyzed period. It is calculated by dividing the amount of bytes transmitted by predicted elephant flows in the network in each second by the amount of bytes transmitted in those seconds by all flows.

$$\text{RAC} = \frac{1}{T}\sum_{t=300}^{720}\frac{bytes\_sent\_elephants_t}{bytes\_sent\_all_t} \tag{5}$$

$$T = 720 - 300 = 420$$

2) FLOW OPERATIONS REDUCTION

This metric measures the inverse of the ratio of flows classified as elephants. We call it flow operations reduction because, in the context of traffic engineering applications, where individual entries are created only for flows classified as elephants, this metric indicates by what factor the number of flow entry operations (creation, deletion) will be reduced.

$$\text{Flow Operations Reduction} = \frac{all\_flows}{elephant\_flows} \tag{6}$$

3) AVERAGE OCCUPANCY REDUCTION

This metric measures the average reduction in the number of entries in the flow table during the analyzed period. It is calculated by dividing the number of flow entries in each second in a situation when all flows have individual entries by the number of elephant flow entries when first-packet elephant flow classification is performed.

$$\text{AOR} = \frac{1}{T}\sum_{t=300}^{720}\frac{flow\_entries\_all_t}{flow\_entries\_elephants_t} \tag{7}$$

$$T = 720 - 300 = 420$$

To determine whether traditional machine learning classification metrics correlate with the above metrics proposed by us, we also calculate and present the following metrics:

Definitions: **TP** – number of true positives, **TN** – number of true negatives, **FP** – number of false positives, **FN** – number of false negatives

4) ACCURACY

Accuracy measures the proportion of correctly classified instances among the total number of instances. It is a commonly used metric to evaluate the overall performance of a classifier. High accuracy indicates that the model is performing well on both the positive and negative classes. However, accuracy can be misleading in cases of imbalanced class distributions, as it does not account for the disparity between classes.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{8}$$

5) TRUE POSITIVE RATE (TPR) / RECALL

TPR, also known as Recall or Sensitivity, measures the proportion of actual positive instances that are correctly identified by the classifier. It is crucial for evaluating the ability of the model to detect positive instances, especially in contexts where missing positive cases is costly. A high recall indicates that the classifier successfully captures the majority of positive cases, but it does not account for false positives.

$$\text{TPR} = \frac{TP}{TP + FN} \tag{9}$$

6) TRUE NEGATIVE RATE (TNR) / SPECIFICITY

TNR, or Specificity, measures the proportion of actual negative instances that are correctly identified by the classifier. It is useful for assessing the model's ability to detect negative instances, especially when the cost of false positives is high. A high specificity indicates that the classifier is effective at identifying true negatives, but it does not address false negatives.

$$\text{TNR} = \frac{TN}{TN + FP} \tag{10}$$

7) FALSE POSITIVE RATE (FPR)

FPR measures the proportion of actual negative instances that are incorrectly classified as positive. It is important for understanding the rate of false alarms produced by the classifier, which can be critical in applications such as fraud detection or medical diagnosis. A low FPR indicates that the

model produces few false positives, but it does not provide information on false negatives.

$$\text{FPR} = \frac{FP}{FP + TN} \tag{11}$$

8) FALSE NEGATIVE RATE (FNR)

FNR measures the proportion of actual positive instances that are incorrectly classified as negative. This metric helps to understand how often the model misses positive instances, which can be particularly important in scenarios where false negatives have severe consequences, such as in disease screening. A low FNR indicates that the model successfully captures most positive instances.

$$\text{FNR} = \frac{FN}{FN + TP} \tag{12}$$

9) POSITIVE PREDICTIVE VALUE (PPV) / PRECISION

PPV, or Precision, measures the proportion of true positive instances among all instances that are classified as positive. It is essential for evaluating the accuracy of positive predictions made by the model. High PPV indicates that the model's positive predictions are highly reliable, which is especially important in contexts where false positives are costly or problematic. PPV does not, however, provide information about the model's ability to detect all positive instances.

$$\text{PPV} = \frac{TP}{TP + FP} \tag{13}$$

10) NEGATIVE PREDICTIVE VALUE (NPV)

NPV measures the proportion of true negative instances among all instances that are classified as negative. It evaluates the accuracy of negative predictions made by the model, which is important in contexts where correctly identifying negatives is crucial. High NPV indicates that the model's negative predictions are reliable, although it does not provide information on the model's ability to detect positive instances.

$$\text{NPV} = \frac{TN}{TN + FN} \tag{14}$$

11) FALSE DISCOVERY RATE (FDR)

FDR measures the proportion of positive predictions that are actually false positives. It is useful for assessing the rate at which the model makes incorrect positive predictions, providing a counterbalance to Precision. A low FDR indicates that the majority of positive predictions are accurate, which is crucial in applications where false positives are problematic or costly. FDR is particularly important in fields such as medical diagnostics, where minimizing false positives is critical to avoid unnecessary treatments or anxiety. It helps researchers and practitioners understand the reliability of positive results and can guide decision-making processes in various domains.

$$\text{FDR} = \frac{FP}{FP + TP} \tag{15}$$

12) FALSE OMISSION RATE (FOR)

FOR measures the proportion of negative predictions that are actually false negatives. It helps in understanding how often the model incorrectly predicts negatives, which is critical in contexts where false negatives have severe consequences. A low FOR indicates that most negative predictions are accurate, providing assurance that the model is not missing many positive cases. FOR is especially relevant in scenarios such as disease screening or security applications, where failing to identify a positive case could have significant repercussions. By monitoring FOR, analysts can assess the completeness of their negative predictions and adjust model thresholds or features accordingly.

$$\text{FOR} = \frac{FN}{FN + TN} \tag{16}$$

13) FSCORE

FScore, or F1 Score, is the harmonic mean of Precision and Recall. It provides a single metric that balances the trade-off between those two, making it particularly useful when the class distribution is imbalanced. A high F1 Score indicates that the classifier has a good balance of Precision and Recall, offering a more comprehensive measure of performance than either metric alone. The F1 Score is widely used in machine learning and information retrieval tasks, as it provides a more nuanced evaluation of model performance compared to accuracy alone. It is especially valuable when working with imbalanced datasets, where simple accuracy might be misleading.

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{17}$$

14) INFORMEDNESS (BM)

Informedness, also known as Bookmaker Informedness (BM), measures the probability that the classifier will make an informed decision as opposed to random guessing. It combines True Positive Rate (TPR) and True Negative Rate (TNR) to provide a comprehensive metric of classifier performance. A high BM value indicates that the model is significantly better than random guessing in identifying both positive and negative instances.

$$\text{BM} = \text{TPR} + \text{TNR} - 1 \tag{18}$$

15) MARKEDNESS (MK)

Markedness measures the difference between the True Positive Rate (TPR) and the False Discovery Rate (FDR), reflecting the effectiveness of the classifier in making positive predictions. It combines Positive Predictive Value (PPV) and Negative Predictive Value (NPV) to provide a comprehensive metric of the reliability of the classifier's predictions. High MK indicates that the classifier's predictions are generally accurate and reliable.

$$\text{MK} = \text{PPV} + \text{NPV} - 1 \tag{19}$$

16) MATTHEWS CORRELATION COEFFICIENT (MCC)

MCC is a correlation coefficient between the observed and predicted classifications, taking into account true and false positives and negatives. It provides a balanced measure even with imbalanced class distributions, offering a single metric that reflects the overall performance of the classifier. A high MCC indicates that the model performs well across all classes, making it a robust metric for evaluation.

$$\mathrm{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{20}$$

17) THREAT SCORE (TS)

TS, also known as Critical Success Index (CSI), measures the proportion of correct positive predictions relative to the total number of instances that should have been predicted positive. It is useful for evaluating the performance of classifiers in scenarios where predicting rare events is important. A high TS indicates that the model is effective at identifying positive instances without producing too many false positives.

$$\mathrm{TS} = \frac{TP}{TP + FN + FP} \tag{21}$$

IV. MODELS

In this research, we analyze four classifier models provided by the `scikit-learn` library [11]. The first model is a k-Nearest Neighbors classifier, whereas the remaining three models are boosting-based classifiers. Tree-based classifiers also provided good results. However, to keep the paper concise and the graphs readable and uncluttered, we decided to split their analysis into a separate paper. Nevertheless, in Figure 13 in the Section VI, we provide a comparison between all classifier types, including tree-based ones. Other classifier models available in `scikit-learn` did not reach convergence or provided poor performance, so we did not include them in the comparison.

All the models were trained using their default hyperparameters as defined in `scikit-learn` version `1.4.2`. We conducted a limited number of experiments with hyperparameter optimization; however, the improvements in model performance were insignificant. A detailed analysis of all hyperparameters for all examined models would go far beyond the scope and size constraints of this paper. Therefore, we decided to present only the results for the default hyperparameters for all models, as they provide a good balance between training time, model size, and performance.

Below we provide short descriptions of all classifier models analyzed in this paper:

A. KNeighborsClassifier

k-Nearest Neighbors is an instance-based learning algorithm used for classification and regression. For a given input sample, the algorithm searches for the k nearest samples in the training dataset and assigns the majority class of these neighbors as the prediction. The distance metric used to find the nearest neighbors is typically Euclidean distance, but other metrics can also be used. The algorithm is non-parametric and lazy, meaning it does not require any training phase and directly uses the training data for making predictions. This makes it computationally expensive for large datasets, as it requires calculating distances to all training samples for each prediction.

B. AdaBoostClassifier

Adaptive Boosting [39] is an ensemble technique that combines the outputs of several weak learners to create a strong classifier. It works by fitting a sequence of weak learners, usually decision stumps (trees with one split), to the training data. Each subsequent learner is trained on the weighted version of the dataset, where the weights are adjusted to focus more on the samples that were misclassified by the previous learners. The final prediction is a weighted sum of the predictions of all weak learners, where the weights reflect the performance of each learner.

C. GradientBoostingClassifier

Gradient Boosting [28] is an ensemble method that builds models sequentially, where each new model corrects the errors made by the previous ones. It combines the predictions of multiple weak learners, typically shallow decision trees, in a stage-wise fashion. Each tree is trained to fit the residual errors of the ensemble of previously trained trees. The optimization is performed using gradient descent to minimize a specified loss function. Gradient Boosting can handle various loss functions, making it versatile for different types of tasks.

D. HistGradientBoostingClassifier

It is an efficient implementation [40] of Gradient Boosting that uses histogram-based binning to speed up the training process. It discretizes the continuous features into bins and builds histograms for each feature, which reduces the computational complexity of finding the optimal splits. This method is particularly advantageous for large datasets as it reduces memory usage and improves training speed while maintaining competitive performance. It supports various loss functions and can handle missing values natively.

V. RESULTS

In Figure 1, 2, and 3, we present the complete results of all analyzed metrics across the five validation dataset folds. We selected three **training traffic coverages** (elephant thresholds) from the 25 values examined: 80%, 96%, and 99.2%. The color intensity of the table fields indicates their value relative to other fields in a particular column.

As shown in these figures, the same training traffic coverage can result in vastly different resulting traffic coverages, depending on the classifier type and its performance. To compare classifier performance accurately, it is necessary to do so for the same resulting traffic coverage. Therefore,

we must *normalize* the performance metrics against the resulting traffic coverage. To facilitate this comparison, in Figure 4 we present interpolated metric values for three **resulting average traffic coverage** values: 70%, 80%, and 90%. After interpolation for each fold, the mean and standard deviation across all folds were calculated.

While Figure 4 shows metrics for three selected values of resulting traffic coverage, subsequent graphs illustrate the reduction in flow table operations and average occupancy over a continuous range of resulting traffic coverage between 50% and 100%. It should be noted that the y-axis is on a logarithmic scale. The goal is to achieve maximum reduction while maintaining the highest traffic coverage. This means that the closer the curve is to the top-right corner of the graph, the better the performance of the particular classifier.

The values presented in the graphs were first interpolated for the continuous spectrum of resulting traffic coverage, and then the mean and standard deviation across all folds were calculated. The black *Data* line represents the ideal performance of an ideal classifier, which is able to perfectly distinguish classify flows on the first packet. To obtain it, we first sorted all flows in each validation fold in descending order of size, and then selected the smallest possible number of flows that collectively cover a specified percentage of network traffic. This approach was described as the "first" method in [41].

Graphs are grouped in pairs on each page. The first graph shows the reduction in flow entry operations, while the second graph presents the average reduction in the number of flow table entries during the analyzed period. It is important to note that the y-axis scales differ between the two graphs.

In Figure 5 and 6, the mean values of flow performance metrics for all analyzed classifiers and all three input data representations are shown. Raw, octets, and bits input data formats are shown as continuous, dashed, and dotted lines, resepectively. Figure 7 and 8 present results narrowed down to the **raw** input data representation. In addition to the mean, they also include the standard deviation across all five folds. It is represented as shaded areas around each line. Similarly, Figure 9 and 10 show results for the **octets** input. Finally, Figure 11 and 12 display results for the case when input features were separate **bits** values. Line for *KNeighborsClassifier* is missing on this graph, because we were unable to complete the simulation within the 5-day limit for the bits input data format in case of that classifier.

VI. DISCUSSION

Figure 1, 2, and 3 show results for 80%, 96%, and 99.2% training traffic coverage. It can be seen that resulting average traffic coverage on the validation set is always lower than the traffic coverage defined during training. The difference varies across different models. In the case of the *KNeighborsClassifier*, the gap between training and resulting traffic coverage is the highest. Despite achieving good **accuracy**, this model also has a high **FNR**. This means that traffic generated by inaccurately classified elephant flows (false negatives) is not covered. Misclassifying an elephant flow has a more substantial impact on traffic coverage than misclassifying a smaller flow. However, this difference is not captured by the accuracy metric, which treats false classifications in both classes equally. Examining the remaining models, a general pattern emerges: higher **FNR** results in lower resulting traffic coverage.

The results normalized against the resulting traffic coverage presented in Figure 4 confirm this observation. When compared for the same resulting traffic coverage, the *KNeighborsClassifier* is the worst-performing model in terms of flow operations and table occupancy reductions. It also requires the highest training traffic coverage. Generally, better-performing models can be trained on a lower traffic coverage to achieve the same resulting coverage. Normalization for the same resulting traffic coverage allows for actual comparison of metrics across different models. Only after such normalization does **accuracy** become a meaningful metric. This is because normalization to the same resulting traffic coverage eliminates the previously described inability to account for the much higher influence of false negatives (elephants misclassified as mice) than false positives (mice classified as elephants) in the calculation of the metric. After normalization, **accuracy** becomes correlated with **flow operations reduction** and **average occupancy reduction**. However, this correlation is not linear – incremental improvements in accuracy result in significantly larger improvements in flow table metrics.

Other metrics that, after normalization, are correlated with flow table performance metrics include **TNR** and **NPV**. This seems to contradict the previous emphasis on minimizing **FNR**. If minimizing the **FNR** were crucial, it would mean maximizing **TPR**. However, in coverage-normalized comparisons, it is the **TNR**, not **TPR**, that is more correlated with performance. This is because the disproportionate influence of false negatives (elephants misclassified as mice) is already filtered out during normalization by resulting traffic coverage. In other words, we have already ensured the desired amount of traffic coverage. What we want now is to reduce the number of entries in flow tables, i.e., reduce the number of mice flows classified as elephants. This means minimizing false positives. And minimizing false positives is equivalent to maximizing **TNR** and **NPV**.

Summarizing, we confirmed that traditional classification metrics are not adequate for assessing the performance of elephant flow classification. Their values only become meaningful when compared for the same resulting traffic coverage. This means that results need to be normalized before comparing their **accuracy** metrics.

An alternative approach to normalization against the resulting traffic coverage would be usage using class weights in the calculation of traditional metrics. However, as mentioned in Section III, determining appropriate weights is not straightforward. Figuring out how to calculate flow weights to make the weighted **accuracy** metric reflect the

Training traffic coverage: 80%

	Resulting Average Coverage	True Positives	True Negatives	False Positives	False Negatives	TPR (Recall)	TNR (Specificity)	FPR	FNR	PPV (Precision)	NPV	FDR	FOR	Accuracy	FScore	BM (Informed-ness)	MK (Marked-ness)	MCC	TS	Flow Operations Reduction	Average Occupancy Reduction
KNeighborsClassifier, raw, 0	0.277	299	1301903	341	954	0.239	1.000	0.000	0.761	0.467	0.999	0.533	0.001	0.999	0.316	0.238	0.466	0.333	0.188	2036.71	268.29
KNeighborsClassifier, raw, 1	0.296	299	1301752	257	1189	0.201	1.000	0.000	0.799	0.538	0.999	0.462	0.001	0.999	0.293	0.201	0.537	0.328	0.171	2344.42	280.43
KNeighborsClassifier, raw, 2	0.319	325	1301839	292	1041	0.238	1.000	0.000	0.762	0.527	0.999	0.473	0.001	0.999	0.328	0.238	0.526	0.354	0.196	2112.64	253.97
KNeighborsClassifier, raw, 3	0.332	322	1301780	249	1146	0.219	1.000	0.000	0.781	0.564	0.999	0.436	0.001	0.999	0.316	0.219	0.563	0.351	0.188	2282.83	268.71
KNeighborsClassifier, raw, 4	0.313	346	1301716	418	1016	0.254	1.000	0.000	0.746	0.453	0.999	0.547	0.001	0.999	0.325	0.254	0.452	0.339	0.194	1706.15	257.71
KNeighborsClassifier, octets, 0	0.283	271	1301949	295	982	0.216	1.000	0.000	0.784	0.479	0.999	0.521	0.001	0.999	0.298	0.216	0.478	0.321	0.175	2303.00	305.17
KNeighborsClassifier, octets, 1	0.303	299	1301755	254	1189	0.201	1.000	0.000	0.799	0.541	0.999	0.459	0.001	0.999	0.293	0.201	0.540	0.329	0.172	2357.14	287.20
KNeighborsClassifier, octets, 2	0.292	291	1301914	217	1075	0.213	1.000	0.000	0.787	0.573	0.999	0.427	0.001	0.999	0.311	0.213	0.572	0.349	0.184	2565.94	297.18
KNeighborsClassifier, octets, 3	0.333	291	1301825	204	1177	0.198	1.000	0.000	0.802	0.588	0.999	0.412	0.001	0.999	0.296	0.198	0.587	0.341	0.174	2633.33	285.72
KNeighborsClassifier, octets, 4	0.326	334	1301850	284	1028	0.245	1.000	0.000	0.755	0.540	0.999	0.460	0.001	0.999	0.337	0.245	0.540	0.364	0.203	2109.22	267.73
AdaBoostClassifier, raw, 0	0.360	359	1295277	6967	894	0.287	0.995	0.005	0.713	0.049	0.999	0.951	0.001	0.994	0.084	0.281	0.048	0.117	0.044	177.93	83.03
AdaBoostClassifier, raw, 1	0.386	416	1294603	7406	1072	0.280	0.994	0.006	0.720	0.053	0.999	0.947	0.001	0.993	0.089	0.274	0.052	0.120	0.047	166.64	81.98
AdaBoostClassifier, raw, 2	0.436	460	1299087	3044	906	0.337	0.998	0.002	0.663	0.131	0.999	0.869	0.001	0.997	0.189	0.334	0.131	0.209	0.104	372.00	107.90
AdaBoostClassifier, raw, 3	0.471	485	1298789	3240	983	0.330	0.998	0.002	0.670	0.130	0.999	0.870	0.001	0.997	0.187	0.328	0.129	0.206	0.103	349.93	101.33
AdaBoostClassifier, raw, 4	0.475	547	1294516	7618	815	0.402	0.994	0.006	0.598	0.067	0.999	0.933	0.001	0.994	0.115	0.396	0.066	0.162	0.061	159.64	71.58
AdaBoostClassifier, octets, 0	0.474	533	1292988	9256	720	0.425	0.993	0.007	0.575	0.054	0.999	0.946	0.001	0.992	0.097	0.418	0.054	0.150	0.051	133.16	66.91
AdaBoostClassifier, octets, 1	0.457	531	1292707	9302	957	0.357	0.993	0.007	0.643	0.054	0.999	0.946	0.001	0.992	0.094	0.350	0.053	0.136	0.049	132.56	67.20
AdaBoostClassifier, octets, 2	0.466	511	1293055	9076	855	0.374	0.993	0.007	0.626	0.053	0.999	0.947	0.001	0.992	0.093	0.367	0.053	0.139	0.049	135.97	65.12
AdaBoostClassifier, octets, 3	0.492	521	1293175	8854	947	0.355	0.993	0.007	0.645	0.056	0.999	0.944	0.001	0.992	0.096	0.348	0.055	0.138	0.050	139.04	67.74
AdaBoostClassifier, octets, 4	0.499	573	1293267	8867	789	0.421	0.993	0.007	0.579	0.061	0.999	0.939	0.001	0.993	0.106	0.414	0.060	0.158	0.056	138.08	65.76
AdaBoostClassifier, bits, 0	0.394	433	1287414	14830	820	0.346	0.989	0.011	0.654	0.028	0.999	0.972	0.001	0.988	0.052	0.334	0.028	0.096	0.027	85.40	49.42
AdaBoostClassifier, bits, 1	0.407	441	1288156	13853	1047	0.296	0.989	0.011	0.704	0.031	0.999	0.969	0.001	0.989	0.056	0.286	0.030	0.093	0.029	91.19	56.63
AdaBoostClassifier, bits, 2	0.413	460	1288861	13270	906	0.337	0.990	0.010	0.663	0.034	0.999	0.966	0.001	0.989	0.061	0.327	0.033	0.103	0.031	94.94	53.84
AdaBoostClassifier, bits, 3	0.432	471	1288059	13970	997	0.321	0.989	0.011	0.679	0.033	0.999	0.967	0.001	0.989	0.059	0.310	0.032	0.099	0.031	90.26	52.10
AdaBoostClassifier, bits, 4	0.416	454	1290858	11276	908	0.333	0.991	0.009	0.667	0.039	0.999	0.961	0.001	0.991	0.069	0.325	0.038	0.111	0.036	111.12	58.83
GradientBoostingClassifier, raw, 0	0.448	493	1298784	3460	760	0.393	0.997	0.003	0.607	0.125	0.999	0.875	0.001	0.997	0.189	0.391	0.124	0.220	0.105	329.75	98.25
GradientBoostingClassifier, raw, 1	0.480	558	1299092	2917	930	0.375	0.998	0.002	0.625	0.161	0.999	0.839	0.001	0.997	0.225	0.373	0.160	0.244	0.127	375.11	102.31
GradientBoostingClassifier, raw, 2	0.455	497	1299858	2273	869	0.364	0.998	0.002	0.636	0.179	0.999	0.821	0.001	0.998	0.240	0.362	0.179	0.254	0.137	470.58	109.09
GradientBoostingClassifier, raw, 3	0.503	545	1300107	1922	923	0.371	0.999	0.001	0.629	0.221	0.999	0.779	0.001	0.998	0.277	0.370	0.220	0.285	0.161	528.37	105.32
GradientBoostingClassifier, raw, 4	0.509	612	1299781	2353	750	0.449	0.998	0.002	0.551	0.206	0.999	0.794	0.001	0.998	0.283	0.448	0.206	0.304	0.165	439.63	99.11
GradientBoostingClassifier, octets, 0	0.495	570	1296645	5599	683	0.455	0.996	0.004	0.545	0.092	0.999	0.908	0.001	0.995	0.154	0.451	0.092	0.203	0.083	211.30	78.14
GradientBoostingClassifier, octets, 1	0.532	658	1296842	5167	830	0.442	0.996	0.004	0.558	0.113	0.999	0.887	0.001	0.995	0.180	0.438	0.112	0.222	0.099	223.78	78.62
GradientBoostingClassifier, octets, 2	0.540	629	1296809	5322	737	0.460	0.996	0.004	0.540	0.106	0.999	0.894	0.001	0.995	0.172	0.456	0.105	0.219	0.094	219.04	75.35
GradientBoostingClassifier, octets, 3	0.561	648	1296635	5394	820	0.441	0.996	0.004	0.559	0.107	0.999	0.893	0.001	0.995	0.173	0.437	0.107	0.216	0.094	215.74	74.88
GradientBoostingClassifier, octets, 4	0.539	667	1297484	4650	695	0.490	0.996	0.004	0.510	0.125	0.999	0.875	0.001	0.996	0.200	0.486	0.125	0.246	0.111	245.16	78.62
GradientBoostingClassifier, bits, 0	0.485	560	1293864	8380	693	0.447	0.994	0.006	0.553	0.063	0.999	0.937	0.001	0.993	0.110	0.440	0.062	0.165	0.058	145.81	67.33
GradientBoostingClassifier, bits, 1	0.527	658	1294737	7272	830	0.442	0.994	0.006	0.558	0.083	0.999	0.917	0.001	0.994	0.140	0.437	0.082	0.190	0.075	164.38	70.01
GradientBoostingClassifier, bits, 2	0.524	610	1293671	8460	756	0.447	0.994	0.006	0.553	0.067	0.999	0.933	0.001	0.993	0.117	0.440	0.067	0.171	0.062	143.72	63.11
GradientBoostingClassifier, bits, 3	0.551	635	1294914	7115	833	0.433	0.995	0.005	0.567	0.082	0.999	0.918	0.001	0.994	0.138	0.427	0.081	0.186	0.074	168.19	67.24
GradientBoostingClassifier, bits, 4	0.531	669	1294903	7231	693	0.491	0.994	0.006	0.509	0.085	0.999	0.915	0.001	0.994	0.144	0.486	0.084	0.202	0.078	165.00	67.59
HistGradientBoostingClassifier, raw, 0	0.540	639	1290799	11445	614	0.510	0.991	0.009	0.490	0.053	1.000	0.947	0.000	0.991	0.096	0.501	0.052	0.162	0.050	107.87	41.59
HistGradientBoostingClassifier, raw, 1	0.514	633	1294385	7624	855	0.425	0.994	0.006	0.575	0.077	0.999	0.923	0.001	0.993	0.130	0.420	0.076	0.179	0.069	157.87	51.97
HistGradientBoostingClassifier, raw, 2	0.593	717	1291139	10992	649	0.525	0.992	0.008	0.475	0.061	0.999	0.939	0.001	0.991	0.110	0.516	0.061	0.177	0.058	111.32	41.27
HistGradientBoostingClassifier, raw, 3	0.541	617	1293647	8382	851	0.420	0.994	0.006	0.580	0.069	0.999	0.931	0.001	0.993	0.118	0.414	0.068	0.168	0.063	144.85	46.26
HistGradientBoostingClassifier, raw, 4	0.549	681	1293695	8439	681	0.500	0.994	0.006	0.500	0.075	0.999	0.925	0.001	0.993	0.130	0.494	0.074	0.191	0.069	142.93	43.86
HistGradientBoostingClassifier, octets, 0	0.549	687	1296999	5245	566	0.548	0.996	0.004	0.452	0.116	1.000	0.884	0.000	0.996	0.191	0.544	0.115	0.251	0.106	219.74	70.27
HistGradientBoostingClassifier, octets, 1	0.579	790	1297256	4753	698	0.531	0.996	0.004	0.469	0.143	0.999	0.857	0.001	0.996	0.225	0.527	0.142	0.274	0.127	235.16	70.21
HistGradientBoostingClassifier, octets, 2	0.597	758	1297347	4784	608	0.555	0.996	0.004	0.445	0.137	1.000	0.863	0.000	0.996	0.219	0.551	0.136	0.274	0.123	235.20	69.62
HistGradientBoostingClassifier, octets, 3	0.600	759	1297934	4095	709	0.517	0.997	0.003	0.483	0.156	0.999	0.844	0.001	0.996	0.240	0.514	0.156	0.283	0.136	268.54	72.00
HistGradientBoostingClassifier, octets, 4	0.579	781	1298388	3746	581	0.573	0.997	0.003	0.427	0.173	1.000	0.827	0.000	0.997	0.265	0.571	0.172	0.313	0.153	287.94	75.07
HistGradientBoostingClassifier, bits, 0	0.544	680	1296593	5651	573	0.543	0.996	0.004	0.457	0.107	1.000	0.893	0.000	0.995	0.179	0.538	0.107	0.240	0.098	205.89	71.28
HistGradientBoostingClassifier, bits, 1	0.584	775	1297143	4866	713	0.521	0.996	0.004	0.479	0.137	0.999	0.863	0.001	0.996	0.217	0.517	0.137	0.266	0.122	231.08	72.52
HistGradientBoostingClassifier, bits, 2	0.587	746	1297219	4912	620	0.546	0.996	0.004	0.454	0.132	1.000	0.868	0.000	0.996	0.212	0.542	0.131	0.267	0.119	230.38	71.65
HistGradientBoostingClassifier, bits, 3	0.604	752	1297315	4714	716	0.512	0.996	0.004	0.488	0.138	0.999	0.862	0.001	0.996	0.217	0.509	0.137	0.264	0.122	238.47	73.46
HistGradientBoostingClassifier, bits, 4	0.594	784	1297934	4200	578	0.576	0.997	0.003	0.424	0.157	1.000	0.843	0.000	0.996	0.247	0.572	0.157	0.300	0.141	261.54	74.47

FIGURE 1. Classification performance metrics for the 80% training traffic coverage – full results for all 5 folds.

Training traffic coverage: 96%

	Resulting Average Coverage	True Positives	True Negatives	False Positives	False Negatives	TPR (Recall)	TNR (Specificity)	FPR	FNR	PPV (Precision)	NPV	FDR	FOR	Accuracy	FScore	BM (Informed-ness)	MK (Marked-ness)	MCC	TS	Flow Operations Reduction	Average Occupancy Reduction
KNeighborsClassifier, raw, 0	0.438	2616	1287833	3297	9751	0.212	0.997	0.003	0.788	0.442	0.992	0.558	0.008	0.990	0.286	0.209	0.435	0.301	0.167	220.45	41.87
KNeighborsClassifier, raw, 1	0.460	2694	1287591	2865	10347	0.207	0.998	0.002	0.793	0.485	0.992	0.515	0.008	0.990	0.290	0.204	0.477	0.312	0.169	234.48	42.75
KNeighborsClassifier, raw, 2	0.458	2702	1287967	3201	9627	0.219	0.998	0.002	0.781	0.458	0.993	0.542	0.007	0.990	0.296	0.217	0.450	0.312	0.174	220.82	42.57
KNeighborsClassifier, raw, 3	0.487	2830	1286934	2803	10930	0.206	0.998	0.002	0.794	0.502	0.992	0.498	0.008	0.989	0.292	0.203	0.494	0.317	0.171	231.40	42.19
KNeighborsClassifier, raw, 4	0.468	2730	1287559	3522	9685	0.220	0.997	0.003	0.780	0.437	0.993	0.563	0.007	0.990	0.292	0.217	0.429	0.305	0.171	208.49	42.40
KNeighborsClassifier, octets, 0	0.427	2433	1288340	2790	9934	0.197	0.998	0.002	0.803	0.466	0.992	0.534	0.008	0.990	0.277	0.195	0.458	0.299	0.161	249.57	44.88
KNeighborsClassifier, octets, 1	0.464	2579	1287955	2501	10462	0.198	0.998	0.002	0.802	0.508	0.992	0.492	0.008	0.990	0.285	0.196	0.500	0.313	0.166	256.59	46.02
KNeighborsClassifier, octets, 2	0.462	2543	1288573	2595	9786	0.206	0.998	0.002	0.794	0.495	0.992	0.505	0.008	0.991	0.291	0.204	0.487	0.316	0.170	253.70	45.46
KNeighborsClassifier, octets, 3	0.478	2624	1287492	2245	11136	0.191	0.998	0.002	0.809	0.539	0.991	0.461	0.009	0.990	0.282	0.189	0.530	0.317	0.164	267.71	45.99
KNeighborsClassifier, octets, 4	0.466	2633	1288123	2958	9782	0.212	0.998	0.002	0.788	0.471	0.992	0.529	0.008	0.990	0.292	0.210	0.463	0.312	0.171	233.14	44.10
AdaBoostClassifier, raw, 0	0.681	5566	1233796	57334	6801	0.450	0.956	0.044	0.550	0.088	0.995	0.912	0.005	0.951	0.148	0.406	0.083	0.184	0.080	20.72	12.13
AdaBoostClassifier, raw, 1	0.696	5891	1231456	59000	7150	0.452	0.954	0.046	0.548	0.091	0.994	0.909	0.006	0.949	0.151	0.406	0.085	0.186	0.082	20.09	12.17
AdaBoostClassifier, raw, 2	0.729	6754	1207092	84076	5575	0.548	0.935	0.065	0.452	0.074	0.995	0.926	0.005	0.931	0.131	0.483	0.070	0.184	0.070	14.35	9.21
AdaBoostClassifier, raw, 3	0.724	6476	1225435	64302	7284	0.471	0.950	0.050	0.529	0.091	0.994	0.909	0.006	0.945	0.153	0.421	0.086	0.190	0.083	18.42	11.21
AdaBoostClassifier, raw, 4	0.753	6770	1202298	88783	5645	0.545	0.931	0.069	0.455	0.071	0.995	0.929	0.005	0.928	0.125	0.477	0.066	0.178	0.067	13.64	9.28
AdaBoostClassifier, octets, 0	0.690	5444	1226295	64835	6923	0.440	0.950	0.050	0.560	0.077	0.994	0.923	0.006	0.945	0.132	0.390	0.072	0.167	0.071	18.55	11.43
AdaBoostClassifier, octets, 1	0.701	5885	1219900	70556	7156	0.451	0.945	0.055	0.549	0.077	0.994	0.923	0.006	0.940	0.132	0.397	0.071	0.168	0.070	17.05	11.17
AdaBoostClassifier, octets, 2	0.733	5843	1213810	77358	6486	0.474	0.940	0.060	0.526	0.070	0.995	0.930	0.005	0.936	0.122	0.414	0.065	0.164	0.065	15.67	10.30
AdaBoostClassifier, octets, 3	0.726	6161	1221131	68606	7599	0.448	0.947	0.053	0.552	0.082	0.994	0.918	0.006	0.942	0.139	0.395	0.076	0.173	0.075	17.43	10.88
AdaBoostClassifier, octets, 4	0.740	5782	1209602	81479	6633	0.466	0.937	0.063	0.534	0.066	0.995	0.934	0.005	0.932	0.116	0.403	0.061	0.156	0.062	14.94	9.93
AdaBoostClassifier, bits, 0	0.694	5761	1177259	113871	6606	0.466	0.912	0.088	0.534	0.048	0.994	0.952	0.006	0.908	0.087	0.378	0.043	0.127	0.046	10.90	8.40
AdaBoostClassifier, bits, 1	0.712	5830	1190092	100364	7211	0.447	0.922	0.078	0.553	0.055	0.994	0.945	0.006	0.917	0.098	0.369	0.049	0.134	0.051	12.27	9.02
AdaBoostClassifier, bits, 2	0.723	5834	1181812	109356	6495	0.473	0.915	0.085	0.527	0.051	0.995	0.949	0.005	0.911	0.092	0.388	0.045	0.132	0.048	11.32	8.35
AdaBoostClassifier, bits, 3	0.718	6031	1183521	106216	7729	0.438	0.918	0.082	0.562	0.054	0.994	0.946	0.006	0.913	0.096	0.356	0.047	0.130	0.050	11.61	8.84
AdaBoostClassifier, bits, 4	0.741	5732	1182038	109043	6683	0.462	0.916	0.084	0.538	0.050	0.994	0.950	0.006	0.911	0.090	0.377	0.044	0.129	0.047	11.36	8.64
GradientBoostingClassifier, raw, 0	0.722	6314	1236842	54288	6053	0.511	0.958	0.042	0.489	0.104	0.995	0.896	0.005	0.954	0.173	0.469	0.099	0.216	0.095	21.51	11.89
GradientBoostingClassifier, raw, 1	0.744	6512	1243037	47419	6529	0.499	0.963	0.037	0.501	0.121	0.995	0.879	0.005	0.959	0.194	0.463	0.116	0.231	0.108	24.17	13.69
GradientBoostingClassifier, raw, 2	0.745	6620	1238967	52201	5709	0.537	0.960	0.040	0.463	0.113	0.995	0.887	0.005	0.956	0.186	0.497	0.108	0.232	0.103	22.16	12.31
GradientBoostingClassifier, raw, 3	0.752	7004	1238064	51673	6756	0.509	0.960	0.040	0.491	0.119	0.995	0.881	0.005	0.955	0.193	0.469	0.114	0.231	0.107	22.21	12.25
GradientBoostingClassifier, raw, 4	0.754	6614	1238983	52098	5801	0.533	0.960	0.040	0.467	0.113	0.995	0.887	0.005	0.956	0.186	0.492	0.108	0.231	0.103	22.20	12.07
GradientBoostingClassifier, octets, 0	0.700	5910	1245073	46057	6457	0.478	0.964	0.036	0.522	0.114	0.995	0.886	0.005	0.960	0.184	0.442	0.109	0.219	0.101	25.08	13.81
GradientBoostingClassifier, octets, 1	0.735	6345	1243190	47266	6696	0.487	0.963	0.037	0.513	0.118	0.995	0.882	0.005	0.959	0.190	0.450	0.113	0.225	0.105	24.31	13.61
GradientBoostingClassifier, octets, 2	0.753	6151	1244828	46340	6178	0.499	0.964	0.036	0.501	0.117	0.995	0.883	0.005	0.960	0.190	0.463	0.112	0.228	0.105	24.83	13.26
GradientBoostingClassifier, octets, 3	0.750	6537	1243802	45935	7223	0.475	0.964	0.036	0.525	0.125	0.994	0.875	0.006	0.959	0.197	0.439	0.119	0.228	0.110	24.84	13.47
GradientBoostingClassifier, octets, 4	0.781	6323	1243585	47496	6092	0.509	0.963	0.037	0.491	0.117	0.995	0.883	0.005	0.959	0.191	0.473	0.113	0.231	0.106	24.22	13.21
GradientBoostingClassifier, bits, 0	0.731	6351	1226655	64475	6016	0.514	0.950	0.050	0.486	0.090	0.995	0.910	0.005	0.946	0.153	0.464	0.085	0.198	0.083	18.40	11.04
GradientBoostingClassifier, bits, 1	0.764	6726	1228944	61512	6315	0.516	0.952	0.048	0.484	0.099	0.995	0.901	0.005	0.948	0.166	0.468	0.093	0.209	0.090	19.10	11.50
GradientBoostingClassifier, bits, 2	0.768	6497	1226193	64975	5832	0.527	0.950	0.050	0.473	0.091	0.995	0.909	0.005	0.946	0.155	0.477	0.086	0.203	0.084	18.24	10.84
GradientBoostingClassifier, bits, 3	0.756	6896	1226718	63019	6864	0.501	0.951	0.049	0.499	0.099	0.994	0.901	0.006	0.946	0.165	0.452	0.093	0.205	0.090	18.64	11.05
GradientBoostingClassifier, bits, 4	0.795	6718	1223469	67612	5697	0.541	0.948	0.052	0.459	0.090	0.995	0.910	0.005	0.944	0.155	0.489	0.086	0.205	0.084	17.54	10.61
HistGradientBoostingClassifier, raw, 0	0.780	6783	1232890	58240	5584	0.548	0.955	0.045	0.452	0.104	0.995	0.896	0.005	0.951	0.175	0.503	0.100	0.224	0.096	20.05	11.28
HistGradientBoostingClassifier, raw, 1	0.799	7089	1236477	53979	5952	0.544	0.958	0.042	0.456	0.116	0.995	0.884	0.005	0.954	0.191	0.502	0.111	0.236	0.106	21.35	12.16
HistGradientBoostingClassifier, raw, 2	0.823	6973	1235791	55377	5356	0.566	0.957	0.043	0.434	0.112	0.996	0.888	0.004	0.953	0.187	0.523	0.108	0.237	0.103	20.91	11.54
HistGradientBoostingClassifier, raw, 3	0.837	7584	1230387	59350	6176	0.551	0.954	0.046	0.449	0.113	0.995	0.887	0.005	0.950	0.188	0.505	0.108	0.234	0.104	19.47	10.86
HistGradientBoostingClassifier, raw, 4	0.830	6807	1234243	56838	5608	0.548	0.956	0.044	0.452	0.107	0.995	0.893	0.005	0.952	0.179	0.504	0.102	0.227	0.098	20.48	11.37
HistGradientBoostingClassifier, octets, 0	0.780	6908	1251509	39621	5459	0.559	0.969	0.031	0.441	0.148	0.996	0.852	0.004	0.965	0.235	0.528	0.144	0.276	0.133	28.01	13.48
HistGradientBoostingClassifier, octets, 1	0.813	7382	1252058	38398	5659	0.566	0.970	0.030	0.434	0.161	0.996	0.839	0.004	0.966	0.251	0.536	0.157	0.290	0.144	28.47	14.20
HistGradientBoostingClassifier, octets, 2	0.814	7146	1253290	37878	5183	0.580	0.971	0.029	0.420	0.159	0.996	0.841	0.004	0.967	0.249	0.550	0.155	0.292	0.142	28.95	13.54
HistGradientBoostingClassifier, octets, 3	0.831	7754	1252039	37698	6006	0.564	0.971	0.029	0.436	0.171	0.995	0.829	0.005	0.966	0.262	0.534	0.166	0.298	0.151	28.68	13.44
HistGradientBoostingClassifier, octets, 4	0.840	7276	1251592	39489	5139	0.586	0.969	0.031	0.414	0.156	0.996	0.844	0.004	0.966	0.246	0.555	0.151	0.290	0.140	27.87	13.24
HistGradientBoostingClassifier, bits, 0	0.762	6740	1249481	41649	5627	0.545	0.968	0.032	0.455	0.139	0.996	0.861	0.004	0.964	0.222	0.513	0.135	0.263	0.125	26.94	13.60
HistGradientBoostingClassifier, bits, 1	0.808	7077	1250756	39700	5964	0.543	0.969	0.031	0.457	0.151	0.995	0.849	0.005	0.965	0.237	0.512	0.147	0.274	0.134	27.87	13.90
HistGradientBoostingClassifier, bits, 2	0.807	6820	1251825	39343	5509	0.553	0.970	0.030	0.447	0.148	0.996	0.852	0.004	0.966	0.233	0.523	0.143	0.274	0.132	28.24	13.47
HistGradientBoostingClassifier, bits, 3	0.813	7371	1250602	39135	6389	0.536	0.970	0.030	0.464	0.158	0.995	0.842	0.005	0.965	0.245	0.505	0.153	0.278	0.139	28.03	13.69
HistGradientBoostingClassifier, bits, 4	0.827	7042	1250779	40302	5373	0.567	0.969	0.031	0.433	0.149	0.996	0.851	0.004	0.965	0.236	0.536	0.144	0.278	0.134	27.53	13.56

FIGURE 2. Classification performance metrics for the 96% training traffic coverage – full results for all 5 folds.

Training traffic coverage: 99.2%

	Resulting Average Coverage	True Positives	True Negatives	False Positives	False Negatives	TPR (Recall)	TNR (Specificity)	FPR	FNR	PPV (Precision)	NPV	FDR	FOR	Accuracy	FScore	BM (Informed-ness)	MK (Marked-ness)	MCC	TS	Flow Operations Reduction	Average Occupancy Reduction
KNeighborsClassifier, raw, 0	0.630	26293	1191935	28302	56967	0.316	0.977	0.023	0.684	0.482	0.954	0.518	0.046	0.935	0.381	0.293	0.436	0.357	0.236	23.88	10.25
KNeighborsClassifier, raw, 1	0.641	27308	1190938	27230	58021	0.320	0.978	0.022	0.680	0.501	0.954	0.499	0.046	0.935	0.390	0.298	0.454	0.368	0.243	23.90	10.33
KNeighborsClassifier, raw, 2	0.635	26777	1194302	28081	54337	0.330	0.977	0.023	0.670	0.488	0.956	0.512	0.044	0.937	0.394	0.307	0.445	0.370	0.245	23.76	10.23
KNeighborsClassifier, raw, 3	0.629	28234	1186961	26190	62112	0.313	0.978	0.022	0.687	0.519	0.950	0.481	0.050	0.932	0.390	0.291	0.469	0.369	0.242	23.95	10.17
KNeighborsClassifier, raw, 4	0.651	27898	1176814	46087	52697	0.346	0.962	0.038	0.654	0.377	0.957	0.623	0.043	0.924	0.361	0.308	0.334	0.321	0.220	17.62	9.25
KNeighborsClassifier, octets, 0	0.585	25467	1192599	27638	57793	0.306	0.977	0.023	0.694	0.480	0.954	0.520	0.046	0.934	0.374	0.283	0.433	0.350	0.230	24.55	10.66
KNeighborsClassifier, octets, 1	0.639	26097	1191842	26326	59232	0.306	0.978	0.022	0.694	0.498	0.953	0.502	0.047	0.934	0.379	0.284	0.450	0.358	0.234	24.86	10.73
KNeighborsClassifier, octets, 2	0.619	25691	1195517	26866	55423	0.317	0.978	0.022	0.683	0.489	0.956	0.511	0.044	0.937	0.384	0.295	0.445	0.362	0.238	24.80	10.56
KNeighborsClassifier, octets, 3	0.640	26859	1188001	25150	63487	0.297	0.979	0.021	0.703	0.516	0.949	0.484	0.051	0.932	0.377	0.277	0.466	0.359	0.233	25.06	10.52
KNeighborsClassifier, octets, 4	0.644	26441	1182560	40341	54154	0.328	0.967	0.033	0.672	0.396	0.956	0.604	0.044	0.928	0.359	0.295	0.352	0.322	0.219	19.52	9.56
AdaBoostClassifier, raw, 0	0.920	55277	961125	259112	27983	0.664	0.788	0.212	0.336	0.176	0.972	0.824	0.028	0.780	0.278	0.452	0.148	0.258	0.161	4.15	3.34
AdaBoostClassifier, raw, 1	0.931	57677	961554	256614	27652	0.676	0.789	0.211	0.324	0.184	0.972	0.816	0.028	0.782	0.289	0.465	0.156	0.269	0.169	4.15	3.28
AdaBoostClassifier, raw, 2	0.891	55241	971848	250535	25873	0.681	0.795	0.205	0.319	0.181	0.974	0.819	0.026	0.788	0.286	0.476	0.155	0.271	0.167	4.26	3.33
AdaBoostClassifier, raw, 3	0.912	59528	957392	255759	30818	0.659	0.789	0.211	0.341	0.189	0.969	0.811	0.031	0.780	0.294	0.448	0.158	0.266	0.172	4.13	3.29
AdaBoostClassifier, raw, 4	0.921	56895	933113	289788	23700	0.706	0.763	0.237	0.294	0.164	0.975	0.836	0.025	0.760	0.266	0.469	0.139	0.256	0.154	3.76	3.02
AdaBoostClassifier, octets, 0	0.927	58251	932104	288133	25009	0.700	0.764	0.236	0.300	0.168	0.974	0.832	0.026	0.760	0.271	0.463	0.142	0.257	0.157	3.76	3.03
AdaBoostClassifier, octets, 1	0.933	60910	930072	286096	24419	0.714	0.764	0.236	0.286	0.175	0.974	0.825	0.026	0.760	0.280	0.477	0.149	0.267	0.163	3.73	2.99
AdaBoostClassifier, octets, 2	0.909	59954	917603	304780	21160	0.739	0.751	0.249	0.261	0.164	0.977	0.836	0.023	0.750	0.269	0.490	0.142	0.264	0.155	3.57	2.83
AdaBoostClassifier, octets, 3	0.924	62474	942296	270855	27872	0.691	0.777	0.223	0.309	0.187	0.971	0.813	0.029	0.771	0.295	0.468	0.159	0.273	0.173	3.91	3.10
AdaBoostClassifier, octets, 4	0.924	57529	928535	294366	23066	0.714	0.759	0.241	0.286	0.163	0.976	0.837	0.024	0.756	0.266	0.473	0.139	0.257	0.153	3.70	2.99
AdaBoostClassifier, bits, 0	0.930	62458	867227	353010	20802	0.750	0.711	0.289	0.250	0.150	0.977	0.850	0.023	0.713	0.250	0.461	0.127	0.242	0.143	3.14	2.62
AdaBoostClassifier, bits, 1	0.936	63421	879688	338480	21908	0.743	0.722	0.278	0.257	0.158	0.976	0.842	0.024	0.724	0.260	0.465	0.134	0.249	0.150	3.24	2.68
AdaBoostClassifier, bits, 2	0.915	60525	889964	332419	20589	0.746	0.728	0.272	0.254	0.154	0.977	0.846	0.023	0.729	0.255	0.474	0.131	0.250	0.146	3.32	2.70
AdaBoostClassifier, bits, 3	0.926	67110	878583	334568	23236	0.743	0.724	0.276	0.257	0.167	0.974	0.833	0.026	0.726	0.273	0.467	0.141	0.257	0.158	3.25	2.68
AdaBoostClassifier, bits, 4	0.933	60631	877545	345356	19964	0.752	0.718	0.282	0.248	0.149	0.978	0.851	0.022	0.720	0.249	0.470	0.127	0.244	0.142	3.21	2.65
GradientBoostingClassifier, raw, 0	0.935	58148	988110	232127	25112	0.698	0.810	0.190	0.302	0.200	0.975	0.800	0.025	0.803	0.311	0.508	0.176	0.299	0.184	4.49	3.47
GradientBoostingClassifier, raw, 1	0.924	58400	998635	219533	26929	0.684	0.820	0.180	0.316	0.210	0.974	0.790	0.026	0.811	0.322	0.504	0.184	0.304	0.192	4.69	3.61
GradientBoostingClassifier, raw, 2	0.909	55883	1011648	210735	25231	0.689	0.828	0.172	0.311	0.210	0.976	0.790	0.024	0.819	0.321	0.517	0.185	0.309	0.191	4.89	3.70
GradientBoostingClassifier, raw, 3	0.926	60955	1005040	208111	29391	0.675	0.828	0.172	0.325	0.227	0.972	0.773	0.028	0.818	0.339	0.503	0.198	0.316	0.204	4.84	3.67
GradientBoostingClassifier, raw, 4	0.935	56845	985390	237511	23750	0.705	0.806	0.194	0.295	0.193	0.976	0.807	0.024	0.800	0.303	0.511	0.170	0.294	0.179	4.43	3.44
GradientBoostingClassifier, octets, 0	0.918	58197	982529	237708	25063	0.699	0.805	0.195	0.301	0.197	0.975	0.803	0.025	0.798	0.307	0.504	0.172	0.294	0.181	4.41	3.41
GradientBoostingClassifier, octets, 1	0.924	59104	992967	225181	26225	0.693	0.815	0.185	0.307	0.208	0.974	0.792	0.026	0.807	0.320	0.508	0.182	0.304	0.190	4.59	3.55
GradientBoostingClassifier, octets, 2	0.905	56862	995782	226601	24252	0.701	0.815	0.185	0.299	0.201	0.976	0.799	0.024	0.808	0.312	0.516	0.177	0.302	0.185	4.60	3.52
GradientBoostingClassifier, octets, 3	0.928	62027	992392	220759	28319	0.687	0.818	0.182	0.313	0.219	0.972	0.781	0.028	0.809	0.332	0.505	0.192	0.311	0.199	4.61	3.52
GradientBoostingClassifier, octets, 4	0.936	57502	983247	239654	23093	0.713	0.804	0.196	0.287	0.194	0.977	0.806	0.023	0.798	0.304	0.517	0.171	0.297	0.180	4.39	3.38
GradientBoostingClassifier, bits, 0	0.941	61302	946901	273336	21958	0.736	0.776	0.224	0.264	0.183	0.977	0.817	0.023	0.773	0.293	0.512	0.161	0.287	0.172	3.90	3.10
GradientBoostingClassifier, bits, 1	0.936	62388	954496	263672	22941	0.731	0.784	0.216	0.269	0.191	0.977	0.809	0.023	0.780	0.303	0.515	0.168	0.294	0.179	4.00	3.16
GradientBoostingClassifier, bits, 2	0.919	60449	953718	268665	20665	0.745	0.780	0.220	0.255	0.184	0.979	0.816	0.021	0.778	0.295	0.525	0.162	0.292	0.173	3.96	3.06
GradientBoostingClassifier, bits, 3	0.939	65359	952014	261137	24987	0.723	0.785	0.215	0.277	0.200	0.974	0.800	0.026	0.780	0.314	0.508	0.175	0.298	0.186	3.99	3.13
GradientBoostingClassifier, bits, 4	0.946	60601	943013	279888	19994	0.752	0.771	0.229	0.248	0.178	0.979	0.822	0.021	0.770	0.288	0.523	0.157	0.287	0.168	3.83	3.02
HistGradientBoostingClassifier, raw, 0	0.937	60661	1002099	218138	22599	0.729	0.821	0.179	0.271	0.218	0.978	0.782	0.022	0.815	0.335	0.550	0.196	0.328	0.201	4.68	3.47
HistGradientBoostingClassifier, raw, 1	0.953	62647	997163	221005	22682	0.734	0.819	0.181	0.266	0.221	0.978	0.779	0.022	0.813	0.340	0.553	0.199	0.331	0.205	4.60	3.39
HistGradientBoostingClassifier, raw, 2	0.948	60337	1000906	221477	20777	0.744	0.819	0.181	0.256	0.214	0.980	0.786	0.020	0.814	0.333	0.563	0.194	0.330	0.199	4.63	3.39
HistGradientBoostingClassifier, raw, 3	0.957	65014	998735	214416	25332	0.720	0.823	0.177	0.280	0.233	0.975	0.767	0.025	0.816	0.352	0.543	0.208	0.336	0.213	4.66	3.42
HistGradientBoostingClassifier, raw, 4	0.964	59994	991421	231480	20601	0.744	0.811	0.189	0.256	0.206	0.980	0.794	0.020	0.807	0.322	0.555	0.185	0.321	0.192	4.47	3.33
HistGradientBoostingClassifier, octets, 0	0.947	60169	1019876	200361	23091	0.723	0.836	0.164	0.277	0.231	0.978	0.769	0.022	0.829	0.350	0.558	0.209	0.341	0.212	5.00	3.65
HistGradientBoostingClassifier, octets, 1	0.953	62003	1022233	195935	23326	0.727	0.839	0.161	0.273	0.240	0.978	0.760	0.022	0.832	0.361	0.566	0.218	0.351	0.220	5.05	3.66
HistGradientBoostingClassifier, octets, 2	0.936	59254	1025695	196688	21860	0.731	0.839	0.161	0.269	0.232	0.979	0.768	0.021	0.832	0.352	0.570	0.211	0.346	0.213	5.09	3.64
HistGradientBoostingClassifier, octets, 3	0.952	64049	1021756	191395	26297	0.709	0.842	0.158	0.291	0.251	0.975	0.749	0.025	0.833	0.370	0.551	0.226	0.353	0.227	5.10	3.67
HistGradientBoostingClassifier, octets, 4	0.956	60105	1008437	214464	20490	0.746	0.825	0.175	0.254	0.219	0.980	0.781	0.020	0.820	0.338	0.570	0.199	0.337	0.204	4.75	3.49
HistGradientBoostingClassifier, bits, 0	0.936	62022	999940	220297	21238	0.745	0.819	0.181	0.255	0.220	0.979	0.780	0.021	0.815	0.339	0.564	0.199	0.335	0.204	4.62	3.41
HistGradientBoostingClassifier, bits, 1	0.951	63071	1009004	209164	22258	0.739	0.828	0.172	0.261	0.232	0.978	0.768	0.022	0.822	0.353	0.567	0.210	0.345	0.214	4.79	3.50
HistGradientBoostingClassifier, bits, 2	0.934	60175	1008641	213742	20939	0.742	0.825	0.175	0.258	0.220	0.980	0.780	0.020	0.820	0.339	0.567	0.199	0.336	0.204	4.76	3.45
HistGradientBoostingClassifier, bits, 3	0.951	65916	1005201	207950	24430	0.730	0.829	0.171	0.270	0.241	0.976	0.759	0.024	0.822	0.362	0.558	0.217	0.348	0.221	4.76	3.44
HistGradientBoostingClassifier, bits, 4	0.957	60674	1000358	222543	19921	0.753	0.818	0.182	0.247	0.214	0.980	0.786	0.020	0.814	0.334	0.571	0.195	0.333	0.200	4.60	3.39

FIGURE 3. Classification performance metrics for the 99.2% training traffic coverage – full results for all 5 folds.

	Resulting Average Coverage	Training Coverage	TPR (Recall)	TNR (Specificity)	PPV (Precision)	NPV	Accuracy	FScore	BM (Informed-ness)	MK (Marked-ness)	MCC	TS	Flow Operations Reduction	Average Occupancy Reduction
KNeighborsClassifier, raw	0.7000	0.9952	0.43 ± 0.01	0.95 ± 0.01	0.53 ± 0.06	0.93 ± 0.01	0.89 ± 0.01	0.48 ± 0.03	0.39 ± 0.01	0.46 ± 0.05	0.42 ± 0.03	0.31 ± 0.02	11.31 ± 0.96	6.12 ± 0.44
KNeighborsClassifier, octets	0.7000	0.9953	0.44 ± 0.05	0.95 ± 0.01	0.54 ± 0.05	0.93 ± 0.01	0.89 ± 0.02	0.48 ± 0.04	0.39 ± 0.04	0.47 ± 0.04	0.43 ± 0.03	0.32 ± 0.04	11.16 ± 2.66	6.04 ± 1.09
AdaBoostClassifier, raw	0.7000	0.9569	0.47 ± 0.02	0.95 ± 0.01	0.09 ± 0.01	0.99 ± 0.00	0.95 ± 0.00	0.14 ± 0.01	0.42 ± 0.02	0.08 ± 0.01	0.18 ± 0.00	0.08 ± 0.00	21.96 ± 2.56	12.39 ± 0.93
AdaBoostClassifier, octets	0.7000	0.9548	0.43 ± 0.03	0.95 ± 0.01	0.07 ± 0.01	0.99 ± 0.00	0.95 ± 0.01	0.12 ± 0.01	0.38 ± 0.02	0.07 ± 0.01	0.16 ± 0.01	0.07 ± 0.01	20.36 ± 2.78	12.64 ± 1.45
AdaBoostClassifier, bits	0.7000	0.9558	0.43 ± 0.03	0.92 ± 0.01	0.05 ± 0.00	0.99 ± 0.00	0.92 ± 0.01	0.09 ± 0.00	0.36 ± 0.02	0.04 ± 0.00	0.13 ± 0.00	0.05 ± 0.00	13.21 ± 1.78	9.73 ± 1.13
GradientBoostingClassifier, raw	0.7000	0.9519	0.44 ± 0.03	0.97 ± 0.00	0.12 ± 0.01	1.00 ± 0.00	0.97 ± 0.00	0.19 ± 0.01	0.42 ± 0.02	0.11 ± 0.01	0.22 ± 0.01	0.10 ± 0.01	36.98 ± 5.52	18.41 ± 2.45
GradientBoostingClassifier, octets	0.7000	0.9506	0.42 ± 0.04	0.98 ± 0.01	0.13 ± 0.01	1.00 ± 0.00	0.97 ± 0.01	0.20 ± 0.01	0.39 ± 0.03	0.13 ± 0.01	0.22 ± 0.00	0.11 ± 0.01	44.26 ± 12.40	21.21 ± 4.81
GradientBoostingClassifier, bits	0.7000	0.9464	0.42 ± 0.03	0.97 ± 0.01	0.09 ± 0.00	1.00 ± 0.00	0.97 ± 0.01	0.15 ± 0.00	0.39 ± 0.03	0.09 ± 0.00	0.18 ± 0.00	0.08 ± 0.00	33.18 ± 6.82	18.03 ± 3.01
HistGradientBoostingClassifier, raw	0.7000	0.9275	0.46 ± 0.02	0.98 ± 0.00	0.10 ± 0.01	1.00 ± 0.00	0.98 ± 0.01	0.16 ± 0.01	0.43 ± 0.02	0.09 ± 0.01	0.20 ± 0.01	0.09 ± 0.01	45.87 ± 8.37	21.62 ± 3.22
HistGradientBoostingClassifier, octets	0.7000	0.8984	0.54 ± 0.04	0.99 ± 0.00	0.17 ± 0.02	1.00 ± 0.00	0.99 ± 0.00	0.26 ± 0.02	0.53 ± 0.04	0.17 ± 0.02	0.30 ± 0.02	0.15 ± 0.01	110.24 ± 30.49	34.16 ± 6.20
HistGradientBoostingClassifier, bits	0.7000	0.9043	0.52 ± 0.04	0.99 ± 0.00	0.15 ± 0.01	1.00 ± 0.00	0.99 ± 0.00	0.24 ± 0.02	0.51 ± 0.04	0.15 ± 0.01	0.28 ± 0.02	0.13 ± 0.01	92.38 ± 25.44	31.31 ± 6.05
KNeighborsClassifier, raw	0.8000	0.9982	0.66 ± 0.02	0.89 ± 0.01	0.68 ± 0.05	0.88 ± 0.01	0.83 ± 0.01	0.67 ± 0.02	0.55 ± 0.01	0.56 ± 0.03	0.56 ± 0.02	0.51 ± 0.03	4.05 ± 0.27	2.81 ± 0.14
KNeighborsClassifier, octets	0.8000	0.9977	0.63 ± 0.03	0.91 ± 0.01	0.65 ± 0.04	0.90 ± 0.01	0.85 ± 0.01	0.64 ± 0.03	0.54 ± 0.02	0.55 ± 0.03	0.54 ± 0.02	0.47 ± 0.03	4.90 ± 0.58	3.23 ± 0.29
AdaBoostClassifier, raw	0.8000	0.9673	0.63 ± 0.02	0.90 ± 0.01	0.08 ± 0.00	0.99 ± 0.00	0.90 ± 0.01	0.14 ± 0.01	0.53 ± 0.02	0.07 ± 0.00	0.20 ± 0.01	0.07 ± 0.00	10.65 ± 0.68	7.29 ± 0.49
AdaBoostClassifier, octets	0.8000	0.9712	0.59 ± 0.02	0.90 ± 0.01	0.08 ± 0.01	0.99 ± 0.00	0.89 ± 0.01	0.14 ± 0.01	0.49 ± 0.01	0.07 ± 0.01	0.19 ± 0.01	0.08 ± 0.01	9.54 ± 0.95	6.82 ± 0.50
AdaBoostClassifier, bits	0.8000	0.9742	0.58 ± 0.02	0.87 ± 0.01	0.07 ± 0.01	0.99 ± 0.00	0.86 ± 0.01	0.13 ± 0.01	0.45 ± 0.02	0.06 ± 0.01	0.17 ± 0.01	0.07 ± 0.01	7.26 ± 0.70	5.68 ± 0.49
GradientBoostingClassifier, raw	0.8000	0.9675	0.60 ± 0.02	0.93 ± 0.01	0.11 ± 0.01	0.99 ± 0.00	0.93 ± 0.01	0.18 ± 0.01	0.53 ± 0.01	0.10 ± 0.01	0.23 ± 0.00	0.10 ± 0.00	15.03 ± 1.57	9.33 ± 0.66
GradientBoostingClassifier, octets	0.8000	0.9677	0.57 ± 0.03	0.94 ± 0.01	0.11 ± 0.01	0.99 ± 0.00	0.93 ± 0.01	0.19 ± 0.01	0.51 ± 0.02	0.11 ± 0.00	0.24 ± 0.00	0.10 ± 0.00	16.25 ± 2.67	9.98 ± 1.08
GradientBoostingClassifier, bits	0.8000	0.9666	0.58 ± 0.03	0.93 ± 0.01	0.09 ± 0.00	0.99 ± 0.00	0.92 ± 0.01	0.16 ± 0.01	0.51 ± 0.02	0.09 ± 0.00	0.21 ± 0.00	0.09 ± 0.00	13.56 ± 2.30	8.69 ± 1.10
HistGradientBoostingClassifier, raw	0.8000	0.9547	0.54 ± 0.02	0.96 ± 0.01	0.11 ± 0.01	1.00 ± 0.00	0.96 ± 0.01	0.18 ± 0.01	0.50 ± 0.02	0.10 ± 0.00	0.23 ± 0.01	0.10 ± 0.00	23.70 ± 3.82	12.68 ± 1.39
HistGradientBoostingClassifier, octets	0.8000	0.9547	0.54 ± 0.02	0.97 ± 0.01	0.16 ± 0.01	1.00 ± 0.00	0.97 ± 0.01	0.25 ± 0.01	0.51 ± 0.02	0.16 ± 0.01	0.28 ± 0.00	0.14 ± 0.00	36.49 ± 7.73	16.35 ± 2.46
HistGradientBoostingClassifier, bits	0.8000	0.9577	0.54 ± 0.02	0.97 ± 0.01	0.15 ± 0.01	1.00 ± 0.00	0.97 ± 0.01	0.23 ± 0.01	0.51 ± 0.02	0.15 ± 0.01	0.27 ± 0.00	0.13 ± 0.00	31.47 ± 6.37	15.03 ± 2.26
KNeighborsClassifier, raw	0.9000	0.9992	0.75 ± 0.01	0.76 ± 0.01	0.74 ± 0.03	0.77 ± 0.03	0.76 ± 0.00	0.75 ± 0.02	0.51 ± 0.00	0.51 ± 0.00	0.51 ± 0.00	0.60 ± 0.02	2.11 ± 0.11	1.74 ± 0.07
KNeighborsClassifier, octets	0.9000	0.9990	0.73 ± 0.02	0.81 ± 0.02	0.73 ± 0.03	0.82 ± 0.03	0.78 ± 0.01	0.73 ± 0.01	0.55 ± 0.01	0.55 ± 0.01	0.55 ± 0.01	0.58 ± 0.01	2.48 ± 0.19	1.96 ± 0.10
AdaBoostClassifier, raw	0.9000	0.9892	0.69 ± 0.03	0.81 ± 0.02	0.16 ± 0.03	0.98 ± 0.01	0.80 ± 0.02	0.25 ± 0.04	0.49 ± 0.03	0.14 ± 0.02	0.26 ± 0.02	0.15 ± 0.03	4.70 ± 0.48	3.65 ± 0.34
AdaBoostClassifier, octets	0.9000	0.9875	0.71 ± 0.01	0.81 ± 0.02	0.13 ± 0.02	0.98 ± 0.00	0.80 ± 0.02	0.22 ± 0.03	0.51 ± 0.01	0.12 ± 0.02	0.25 ± 0.01	0.13 ± 0.02	4.70 ± 0.49	3.69 ± 0.38
AdaBoostClassifier, bits	0.9000	0.9887	0.69 ± 0.01	0.77 ± 0.01	0.12 ± 0.01	0.98 ± 0.00	0.77 ± 0.01	0.21 ± 0.01	0.46 ± 0.01	0.11 ± 0.01	0.22 ± 0.01	0.12 ± 0.01	4.05 ± 0.24	3.31 ± 0.19
GradientBoostingClassifier, raw	0.9000	0.9870	0.69 ± 0.01	0.84 ± 0.01	0.15 ± 0.02	0.99 ± 0.00	0.83 ± 0.01	0.24 ± 0.02	0.53 ± 0.01	0.13 ± 0.01	0.26 ± 0.02	0.14 ± 0.02	5.59 ± 0.23	4.27 ± 0.22
GradientBoostingClassifier, octets	0.9000	0.9858	0.68 ± 0.01	0.85 ± 0.02	0.14 ± 0.02	0.99 ± 0.00	0.85 ± 0.02	0.24 ± 0.03	0.53 ± 0.01	0.13 ± 0.02	0.26 ± 0.01	0.14 ± 0.02	6.19 ± 0.89	4.66 ± 0.59
GradientBoostingClassifier, bits	0.9000	0.9856	0.67 ± 0.01	0.85 ± 0.02	0.14 ± 0.01	0.99 ± 0.00	0.84 ± 0.02	0.22 ± 0.02	0.51 ± 0.01	0.12 ± 0.01	0.25 ± 0.01	0.13 ± 0.01	5.99 ± 0.63	4.54 ± 0.44
HistGradientBoostingClassifier, raw	0.9000	0.9796	0.65 ± 0.01	0.90 ± 0.01	0.13 ± 0.01	0.99 ± 0.00	0.90 ± 0.01	0.22 ± 0.01	0.55 ± 0.01	0.12 ± 0.01	0.26 ± 0.01	0.12 ± 0.01	9.28 ± 0.99	6.28 ± 0.52
HistGradientBoostingClassifier, octets	0.9000	0.9798	0.64 ± 0.01	0.92 ± 0.01	0.16 ± 0.01	0.99 ± 0.00	0.91 ± 0.01	0.25 ± 0.01	0.56 ± 0.01	0.15 ± 0.01	0.29 ± 0.01	0.15 ± 0.01	11.15 ± 1.14	7.19 ± 0.58
HistGradientBoostingClassifier, bits	0.9000	0.9810	0.65 ± 0.02	0.91 ± 0.01	0.15 ± 0.01	0.99 ± 0.00	0.90 ± 0.01	0.24 ± 0.01	0.55 ± 0.01	0.14 ± 0.01	0.28 ± 0.00	0.14 ± 0.01	9.60 ± 1.53	6.40 ± 0.81

FIGURE 4. Classification performance metrics normalized (interpolated) for 70%, 80%, and 90% resulting traffic coverage values – mean and standard deviation across all 5 folds.

flow operation reduction metric accurately would be an interesting problem for future research.

When comparing classifier performance, it can be seen in Figure 5 and 6 that the *HistGradientBoostingClassifier* provides the best reduction in flow operations and flow table occupancy across the entire range of resulting traffic coverage between 50% and 100%. As can be seen in Figure 13, with the 80% resulting traffic coverage target, it allows to reduce the number of flow operations by a factor of 36.49 ± 7.73 (mean and standard deviation). It also reduces the average number of individual flow entries by a factor of 16.35 ± 2.46. Both of these results are achieved using the **octets** input data format.

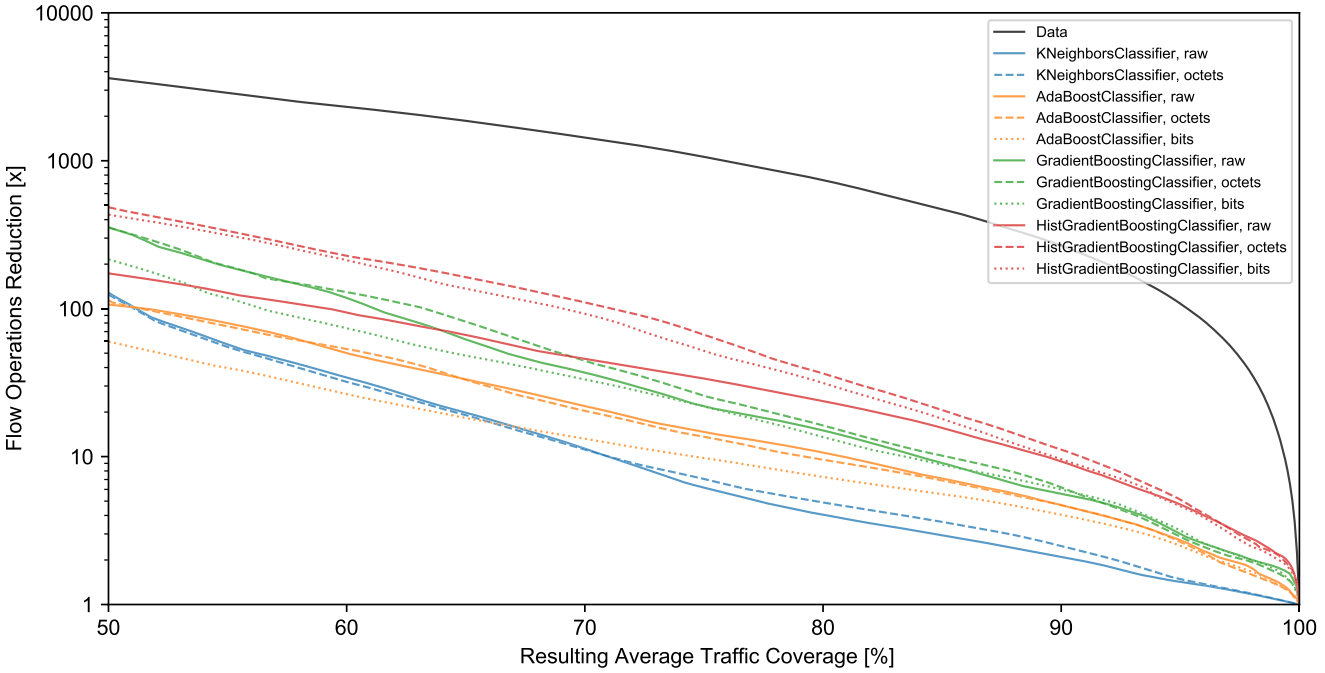

FIGURE 5. Flow operations number reduction factor for the resulting traffic coverage between 50% and 100%. Mean reduction across all five data folds for all input data representations is shown.

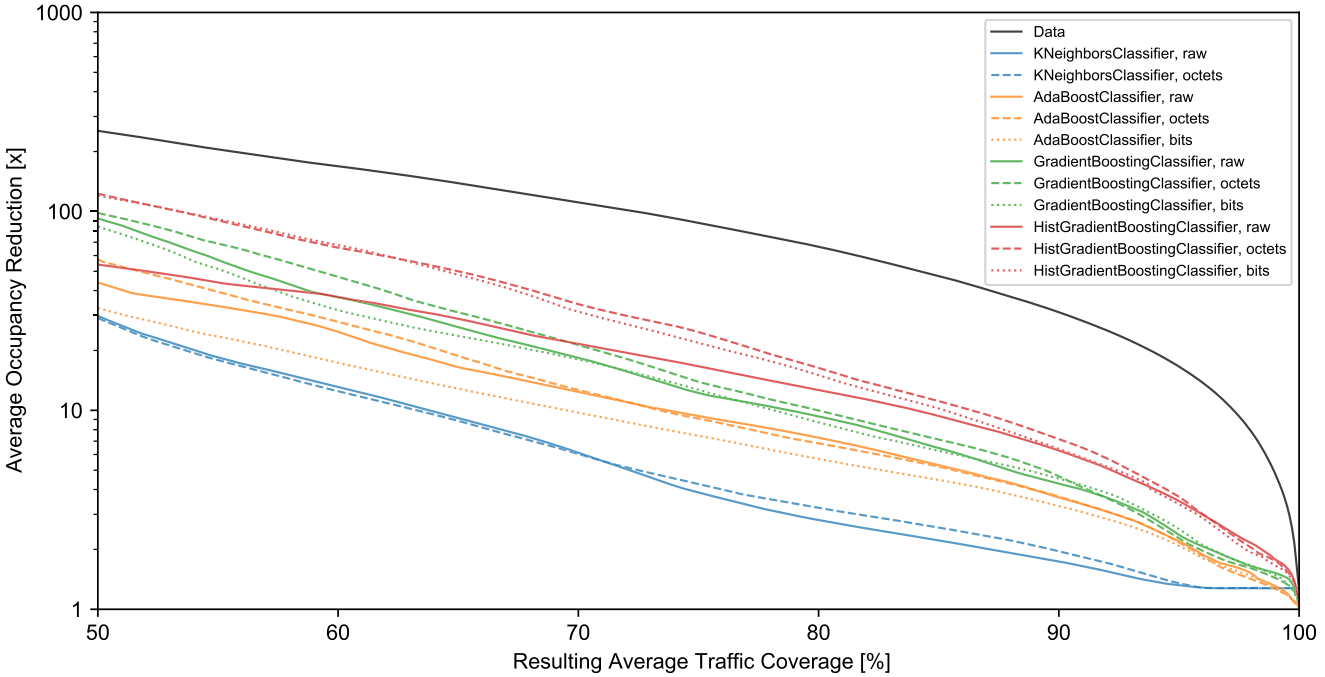

FIGURE 6. Average flow table occupancy reduction factor for the resulting traffic coverage between 50% and 100%. Mean reduction across all five data folds for all input data representations is shown.

The *GradientBoostingClassifier* and *AdaBoostClassifier* models perform worse, achieving maximum flow operations reductions by factors of 10.65 and 16.25, respectively. The lowest performance is achieved by the *KNeighborsClassifier*, which reduces the number of flow operations only by a factor of 4.90 while maintaining 80% traffic coverage. This is likely due to its inability to take sample weights into account during training.

When compared in terms of the traditional **accuracy** metric, the *HistGradientBoostingClassifier* achieves 99%, 97%, and 91.4% accuracy in mouse/elephant classification for resulting traffic coverages of 70%, 80%, and 90%,

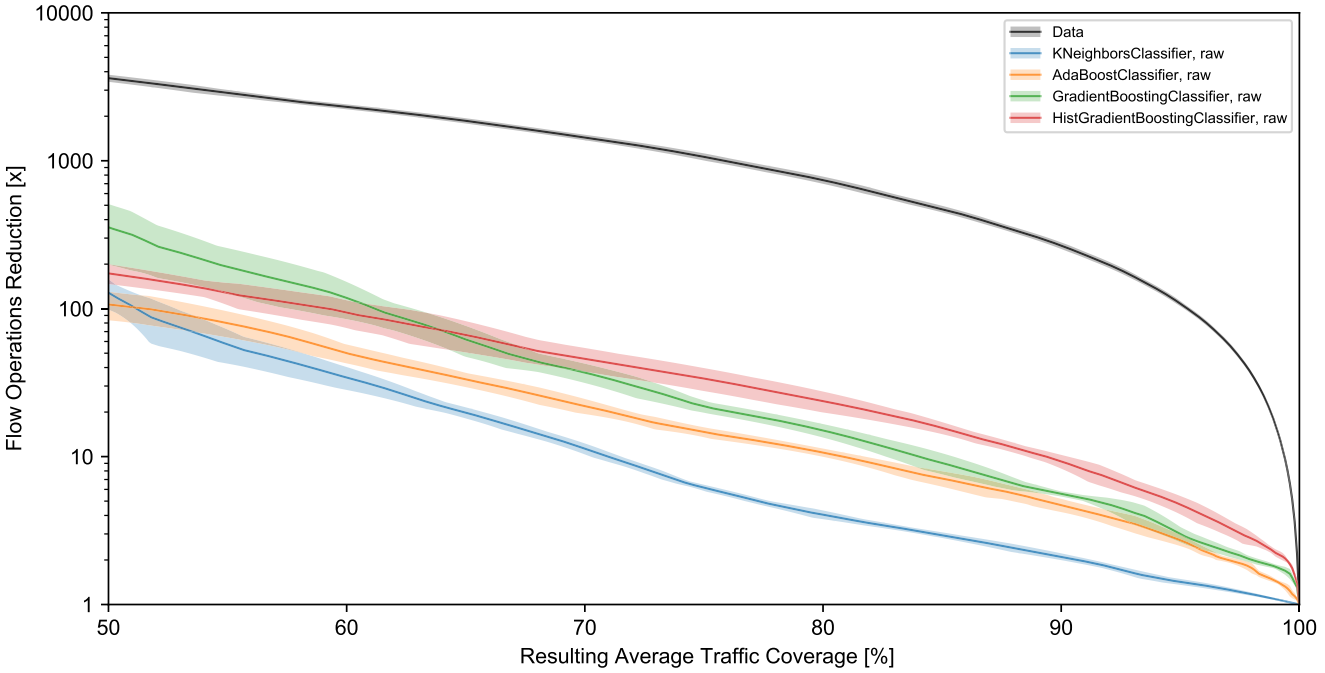

FIGURE 7. Flow operations number reduction factor for the resulting traffic coverage between 50% and 100%. Mean reduction across all five data folds for the raw input data is shown. The shaded area around each line represents standard deviation.

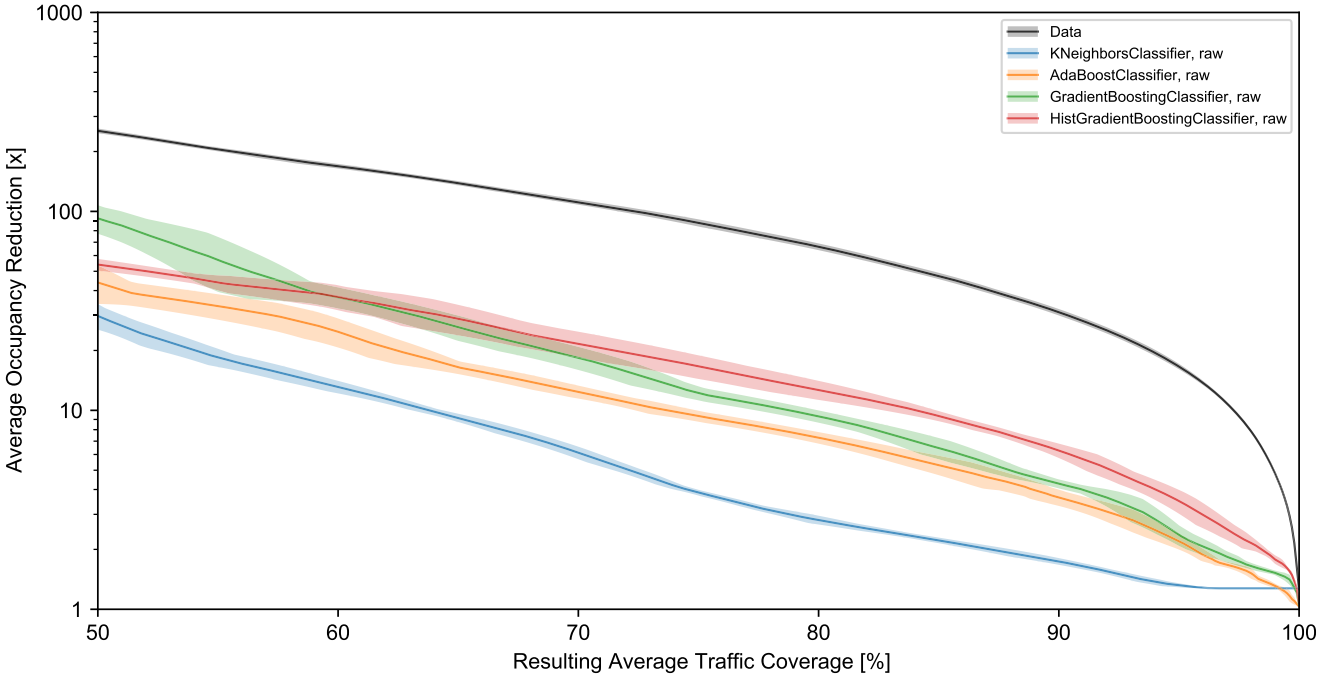

FIGURE 8. Average flow table occupancy reduction factor for the resulting traffic coverage between 50% and 100%. Mean reduction across all five data folds for the raw input data is shown. The shaded area around each line represents standard deviation.

respectively. These results correlate with flow table reduction metrics and are similarly achieved with the **octets** input data format. The *KNeighborsClassifier* achieves an accuracy of 84.8% for the **octets** input format. At first glance, this seems like a good result. However, it translates to significantly lower flow table metrics, as the flow table operations reduction factor is 7.45 times lower. This indicates that elephant flow classifier models need to achieve accuracy higher than 90% to be considered as useful in traffic engineering and QoS applications.

The *HistGradientBoostingClassifier* not only provided better classification performance than the *GradientBoostingClassifier*, but also significantly lower training and inference times due to a more efficient implementation of

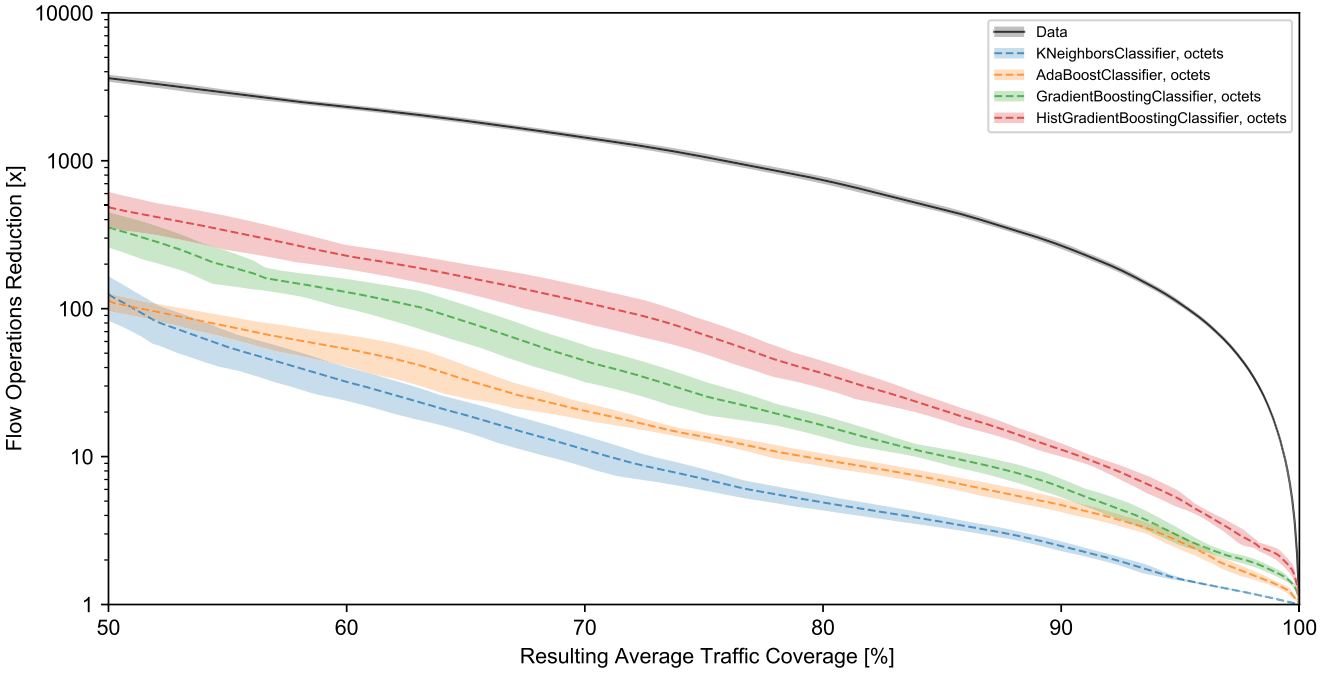

FIGURE 9. Flow operations number reduction factor for the resulting traffic coverage between 50% and 100%. Mean reduction across all five data folds for the octets input data is shown. The shaded area around each line represents standard deviation.

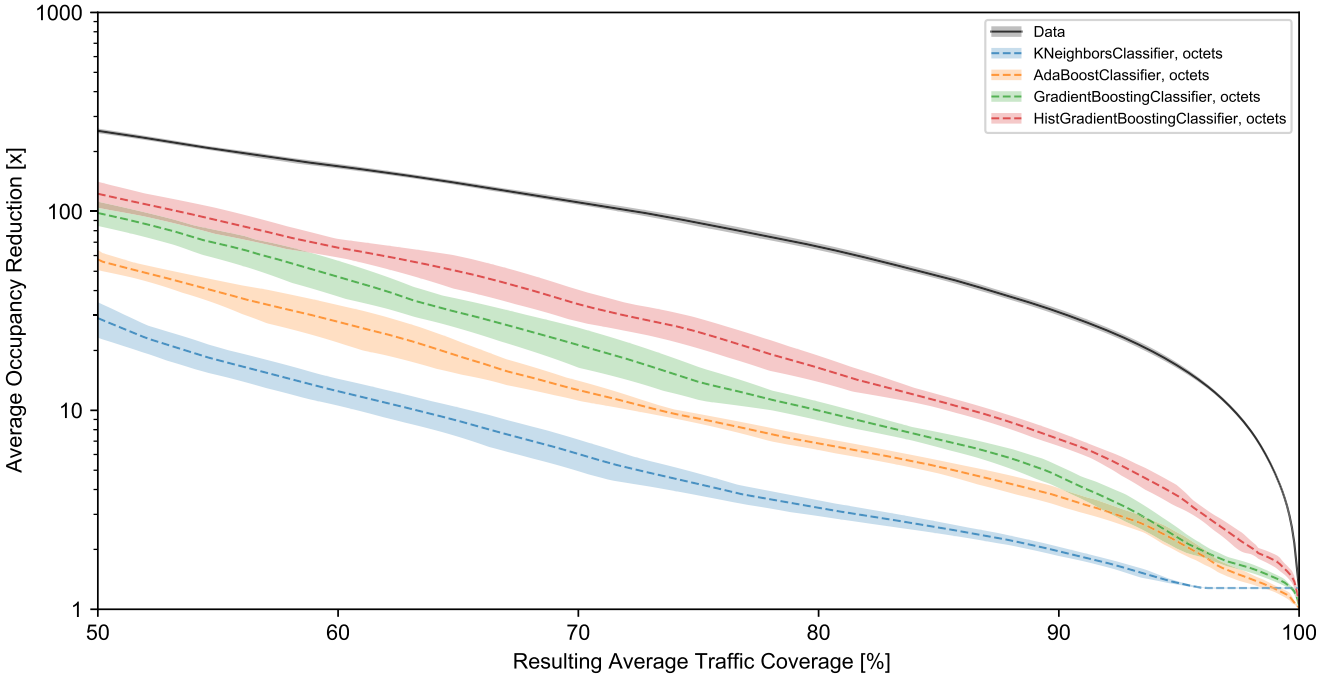

FIGURE 10. Average flow table occupancy reduction factor for the resulting traffic coverage between 50% and 100%. Mean reduction across all five data folds for the octets input data is shown. The shaded area around each line represents standard deviation.

the Gradient Boosting algorithm. On the other hand, the *KNeighborsClassifier* had the slowest training and inference speed. Specifically, for the **bits** input data representation, we were unable to complete the simulation within the 5-day limit, so we do not provide results for that combination.

Differences in model performance based on different input data representations are also evident. Using **raw** header fields resulted in significantly worse classification performance for the *HistGradientBoostingClassifier*. The performance of the model when using **octets** and **bits** input data formats was similar, with **octets** being slightly better. It should be noted

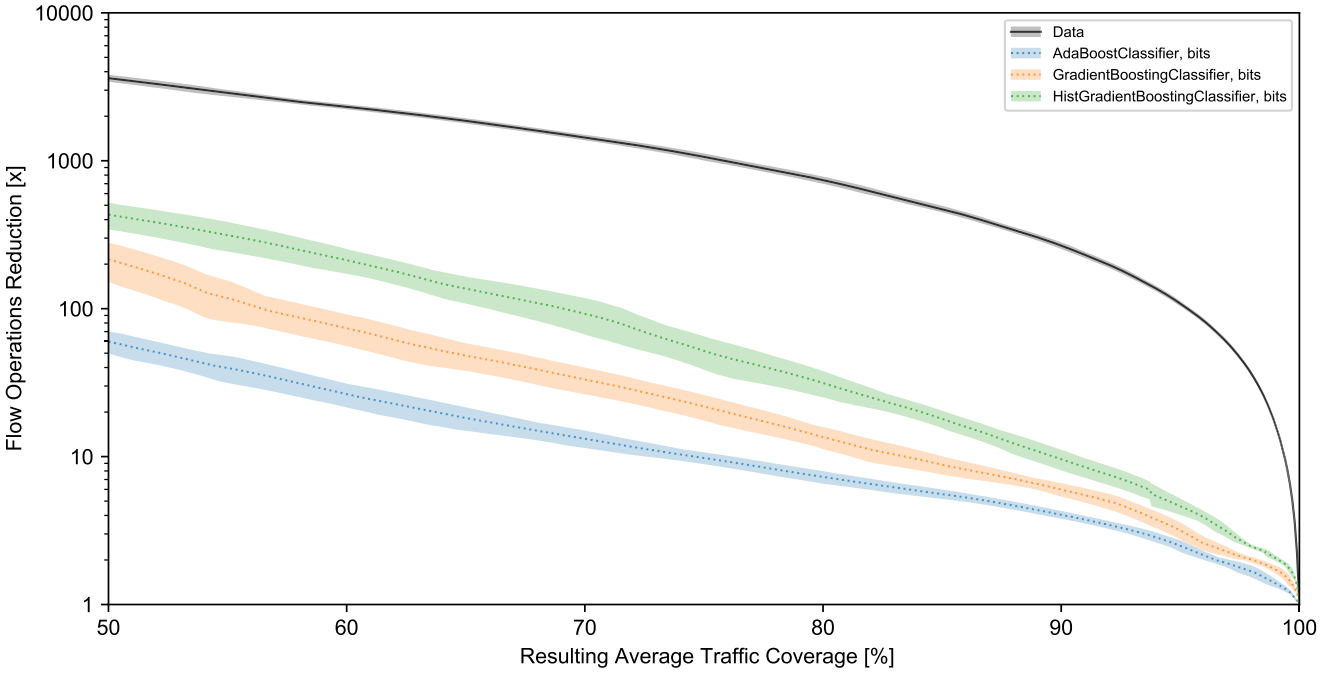

FIGURE 11. Flow operations number reduction factor for the resulting traffic coverage between 50% and 100%. Mean reduction across all five data folds for the bits input data is shown. The shaded area around each line represents standard deviation.

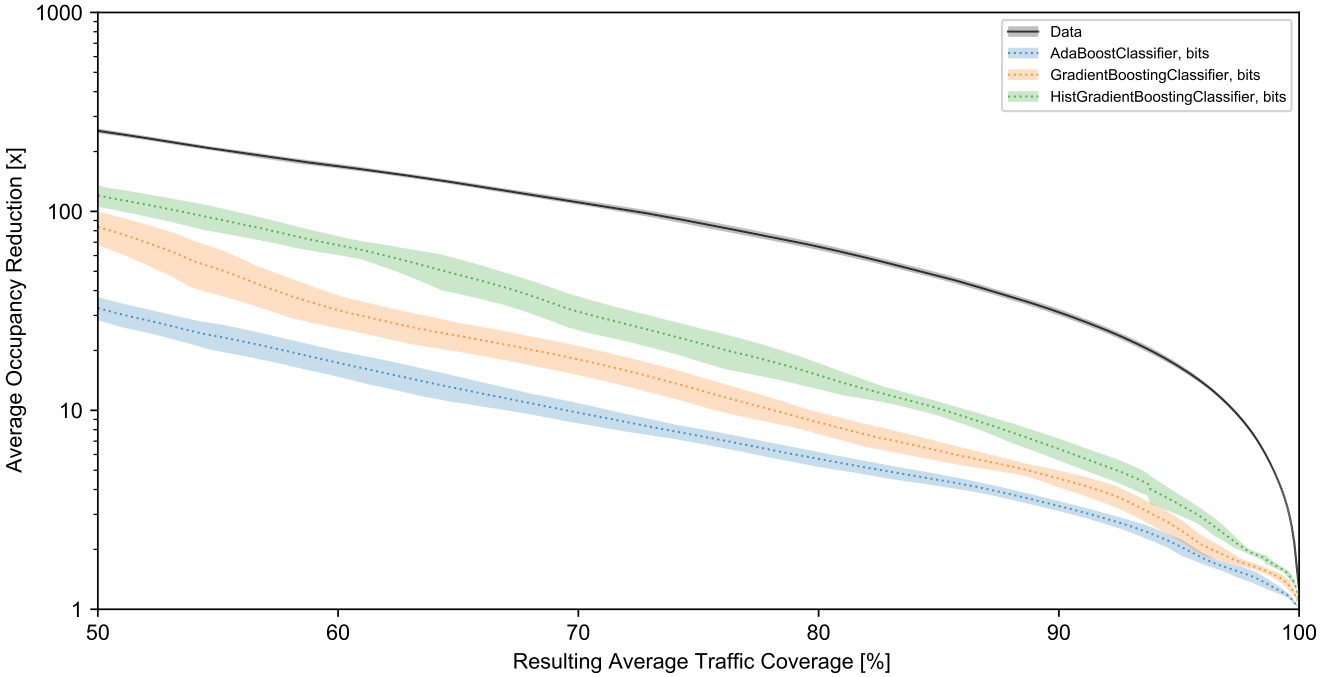

FIGURE 12. Average flow table occupancy reduction factor for the resulting traffic coverage between 50% and 100%. Mean reduction across all five data folds for the bits input data is shown. The shaded area around each line represents standard deviation.

that splitting header fields into octets, instead of separate bits, also results in a significantly lower number of input features (13 vs. 104), which considerably reduces memory usage and improves training and inference speed. Therefore, transforming input header fields into **octets** is the preferable approach.

For other classifiers, the input data format has less influence. Interestingly, in the case of the *GradientBoostingClassifier*, the bits input format performs significantly worse than the other two formats, despite requiring the most memory and computing time. Conversely, ensemble forest-based classifiers, as shown in Figure 13, provided the best

	70%	80%	90%	70%	80%	90%
Data, raw	1437.25 ± 62.11	740.03 ± 41.45	266.91 ± 13.66	111.37 ± 3.30	66.44 ± 2.58	31.14 ± 1.16
KNeighborsClassifier, raw	11.31 ± 0.96	4.05 ± 0.27	2.11 ± 0.11	6.12 ± 0.44	2.81 ± 0.14	1.74 ± 0.07
KNeighborsClassifier, octets	11.16 ± 2.66	4.90 ± 0.58	2.48 ± 0.19	6.04 ± 1.09	3.23 ± 0.29	1.96 ± 0.10
AdaBoostClassifier, raw	21.96 ± 2.56	10.65 ± 0.68	4.70 ± 0.48	12.39 ± 0.93	7.29 ± 0.49	3.65 ± 0.34
AdaBoostClassifier, octets	20.36 ± 2.78	9.54 ± 0.95	4.70 ± 0.49	12.64 ± 1.45	6.82 ± 0.50	3.69 ± 0.38
AdaBoostClassifier, bits	13.21 ± 1.78	7.26 ± 0.70	4.05 ± 0.24	9.73 ± 1.13	5.68 ± 0.49	3.31 ± 0.19
GradientBoostingClassifier, raw	36.98 ± 5.52	15.03 ± 1.57	5.59 ± 0.23	18.41 ± 2.45	9.33 ± 0.66	4.27 ± 0.22
GradientBoostingClassifier, octets	44.26 ± 12.40	16.25 ± 2.67	6.19 ± 0.89	21.21 ± 4.81	9.98 ± 1.08	4.66 ± 0.59
GradientBoostingClassifier, bits	33.18 ± 6.82	13.56 ± 2.30	5.99 ± 0.63	18.03 ± 3.01	8.69 ± 1.10	4.54 ± 0.44
HistGradientBoostingClassifier, raw	45.87 ± 8.37	23.70 ± 3.82	9.28 ± 0.99	21.62 ± 3.22	12.68 ± 1.39	6.28 ± 0.52
HistGradientBoostingClassifier, octets	110.24 ± 30.49	36.49 ± 7.73	11.15 ± 1.14	34.16 ± 6.20	16.35 ± 2.46	7.19 ± 0.58
HistGradientBoostingClassifier, bits	92.38 ± 25.44	31.47 ± 6.37	9.60 ± 1.53	31.31 ± 6.05	15.03 ± 2.26	6.40 ± 0.81
DecisionTreeClassifier, raw	17.48 ± 2.43	6.77 ± 0.49	2.46 ± 0.23	8.60 ± 1.06	4.19 ± 0.32	1.93 ± 0.13
DecisionTreeClassifier, octets	43.91 ± 11.83	10.05 ± 2.00	2.69 ± 0.42	15.32 ± 3.09	5.36 ± 0.79	2.06 ± 0.22
DecisionTreeClassifier, bits	28.12 ± 7.54	7.29 ± 1.74	2.16 ± 0.35	11.46 ± 2.29	4.28 ± 0.74	1.77 ± 0.20
RandomForestClassifier, raw	17.16 ± 1.55	6.80 ± 0.70	2.78 ± 0.18	8.14 ± 0.69	4.09 ± 0.35	2.09 ± 0.10
RandomForestClassifier, octets	53.32 ± 13.67	18.16 ± 4.27	4.69 ± 1.15	15.07 ± 2.71	7.64 ± 1.36	2.99 ± 0.55
RandomForestClassifier, bits	56.61 ± 10.56	20.14 ± 5.30	4.86 ± 0.89	15.53 ± 2.17	8.26 ± 1.61	3.08 ± 0.42
ExtraTreesClassifier, raw	15.41 ± 1.30	6.38 ± 0.45	2.83 ± 0.15	7.60 ± 0.48	3.95 ± 0.24	2.12 ± 0.09
ExtraTreesClassifier, octets	51.22 ± 10.33	18.05 ± 3.41	5.00 ± 0.96	15.21 ± 2.26	7.71 ± 1.03	3.15 ± 0.46
ExtraTreesClassifier, bits	65.54 ± 10.56	23.88 ± 5.27	5.46 ± 1.26	17.33 ± 2.23	9.23 ± 1.53	3.35 ± 0.58
	Flow Operations Reduction (mean ± std) [x]			Average Occupancy Reduction (mean ± std) [x]		

KNeighborsClassifier, raw	0.995203	0.998156	0.999227	0.895 ± 0.006	0.834 ± 0.005	0.760 ± 0.002
KNeighborsClassifier, octets	0.995348	0.997710	0.999001	0.891 ± 0.015	0.848 ± 0.008	0.783 ± 0.010
AdaBoostClassifier, raw	0.956944	0.967316	0.989209	0.947 ± 0.005	0.896 ± 0.006	0.801 ± 0.020
AdaBoostClassifier, octets	0.954838	0.971153	0.987502	0.947 ± 0.006	0.893 ± 0.009	0.801 ± 0.020
AdaBoostClassifier, bits	0.955811	0.974210	0.988707	0.920 ± 0.010	0.863 ± 0.013	0.768 ± 0.012
GradientBoostingClassifier, raw	0.951896	0.967496	0.987044	0.968 ± 0.004	0.929 ± 0.005	0.834 ± 0.005
GradientBoostingClassifier, octets	0.950636	0.967711	0.985753	0.972 ± 0.008	0.935 ± 0.008	0.847 ± 0.019
GradientBoostingClassifier, bits	0.946392	0.966597	0.985609	0.966 ± 0.008	0.923 ± 0.012	0.842 ± 0.015
HistGradientBoostingClassifier, raw	0.927534	0.954748	0.979619	0.976 ± 0.005	0.956 ± 0.008	0.896 ± 0.010
HistGradientBoostingClassifier, octets	0.898353	0.954696	0.979779	0.990 ± 0.004	0.970 ± 0.007	0.914 ± 0.007
HistGradientBoostingClassifier, bits	0.904324	0.957675	0.980999	0.988 ± 0.005	0.966 ± 0.007	0.899 ± 0.015
DecisionTreeClassifier, raw	0.987732	0.995179	0.998701	0.935 ± 0.008	0.868 ± 0.005	0.788 ± 0.008
DecisionTreeClassifier, octets	0.973545	0.992928	0.998534	0.972 ± 0.009	0.901 ± 0.016	0.798 ± 0.014
DecisionTreeClassifier, bits	0.981293	0.995045	0.998996	0.957 ± 0.015	0.873 ± 0.021	0.763 ± 0.017
RandomForestClassifier, raw	0.990804	0.995921	0.998536	0.929 ± 0.004	0.870 ± 0.007	0.812 ± 0.006
RandomForestClassifier, octets	0.983174	0.992212	0.997300	0.970 ± 0.007	0.932 ± 0.012	0.864 ± 0.010
RandomForestClassifier, bits	0.985018	0.992467	0.997235	0.968 ± 0.006	0.933 ± 0.014	0.868 ± 0.008
ExtraTreesClassifier, raw	0.991160	0.995892	0.998432	0.922 ± 0.004	0.864 ± 0.005	0.814 ± 0.004
ExtraTreesClassifier, octets	0.981081	0.991246	0.996793	0.971 ± 0.006	0.935 ± 0.009	0.864 ± 0.010
ExtraTreesClassifier, bits	0.979500	0.989993	0.996603	0.976 ± 0.004	0.946 ± 0.012	0.872 ± 0.013
	Training Coverage (mean)			Accuracy (mean ± std)		

FIGURE 13. Combined comparison of all classifiers for 70%, 80%, and 90% resulting traffic coverage values.

performance with the bits input data. However, a detailed investigation of tree-based classifiers will be the subject of a separate paper.

When compared with results presented in other papers, the *HistGradientBoostingClassifier* outperforms both the *RandomForestClassifier* and *ExtraTreesClassifier*, as shown in Figure 13. It also achieves a significantly higher reduction in the number of flow table operations than the simple neural network classifier model presented in [42], which only reduced flow table operations by a factor of 14.7 under the most optimal hyperparameter configuration.

In summary, the *HistGradientBoostingClassifier* with octets input data representation consistently emerges as the top performer across all traffic coverage levels when considering flow operations reduction and average flow table occupancy reduction. When aiming to cover 80% of network traffic with individual entries, it can reduce the average number of entries in flow tables by a factor of 16.35. This significant reduction enables flow-based traffic engineering in networks with speeds an order of magnitude higher than previously possible with current switches and their flow table capacities. Achieving 80% traffic coverage will still allow for adaptive network load balancing, ensuring high throughput and low packet loss for users. Additionally, the *HistGradientBoostingClassifier* features reduced memory usage and improved training speed compared to other models analyzed. Therefore, it can serve as a strong starting point for future research.

VII. CONCLUSION

In conclusion, traditional metrics such as accuracy do not fully capture the application-specific performance metrics represented by flow operations reduction and average table occupancy reduction. To use traditional metrics as proxies for these application-specific metrics, it is necessary to compare them at the same resulting traffic coverage. Even with such normalization, the relationship between accuracy and flow-specific metrics is not linear. An accuracy higher than 90% (after normalization) needs to be achieved by a

model to consider it useful in traffic engineering and QoS applications.

The choice of data representation (raw, octets, bits) significantly influences classifier performance. For most classifiers, representations using octets and bits lead to higher performance. Particularly, the octets representation is noteworthy because it significantly enhances performance while maintaining a low total number of features, thereby reducing memory usage and computation time.

Among the classifiers analyzed, the Histogram-based Gradient Boosting with octets input data format provides the best reduction in flow operations and flow table occupancy across the entire range of resulting traffic coverage between 50% and 100%. For 80% resulting traffic coverage, it reduces the number of flow operations by a factor of 36.49 and the average number of individual flow entries by a factor of 16.35.

While this study offers comprehensive insights, future work should explore the scalability of these findings to another network environments. Additionally, investigating the temporal stability of classifier performance and the impact of evolving network characteristics on flow operations reduction could provide valuable insights for long-term deployments. As demonstrated in this paper, focusing on boosting-based classifiers with octets data representation appears to be the most promising path to maximizing operational efficiency in network traffic classification tasks and would serve as a solid starting point for more detailed research.

Future research should also include exploring the capabilities of performing real-time inference in networking hardware at the flow arrival rate. It has been shown that it is possible to perform inference using (size-limited) Random Forest models at line rate on P4 switches, without any additional machine learning coprocessors [43]. The question remains whether the same is possible for Histogram-based Gradient Boosting models. We also suspect that such resource-limited models would be more affected by changes in hyperparameters.

Moreover, we found that training classifiers with sample weights proportional to the square root of flow size significantly improves performance compared to using non-weighted training data or class-based sample weight balancing. However, further research is needed in this area, including investigating a range of other possible approaches for calculating sample weights. Additionally, developing a weighted accuracy calculation method to reflect flow-based metrics without the need for normalization against the resulting traffic coverage would be a valuable future research topic.

DATA AVAILABILITY

The anonymized input data and code is available in the GitHub repository: https://github.com/piotrjurkiewicz/flow-models.

REFERENCES

[1] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Comput. Netw.*, vol. 71, pp. 1–30, Oct. 2014.

[2] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[3] A. Mendiola, J. Astorga, E. Jacob, and M. Higuero, "A survey on the contributions of software-defined networking to traffic engineering," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 918–953, 2nd Quart., 2017.

[4] P. Jurkiewicz, R. Wójcik, J. Domzal, and A. Kamisinski, "Testing implementation of FAMTAR: Adaptive multipath routing," *Comput. Commun.*, vol. 149, pp. 300–311, Jan. 2020.

[5] G. Shen, Q. Li, S. Ai, Y. Jiang, M. Xu, and X. Jia, "How powerful switches should be deployed: A precise estimation based on queuing theory," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 811–819.

[6] P. Jurkiewicz, G. Rzym, and P. Borylo, "Flow length and size distributions in campus internet traffic," *Comput. Commun.*, vol. 167, pp. 15–30, Feb. 2021.

[7] P. Megyesi and S. Molnár, "Analysis of elephant users in broadband network traffic," in *Proc. Meeting Eur. Netw. Universities Companies Inf. Commun. Eng.* Chemnitz, Germany: Springer, 2013, pp. 37–45.

[8] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the flow completion time tail in datacenter networks," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Architectures, Protocols Comput. Commun.* New York, NY, USA: Association for Computing Machinery, 2012, pp. 139–150.

[9] Y. Liu, T. Yu, Q. Meng, and Q. Liu, "Flow optimization strategies in data center networks: A survey," *J. Netw. Comput. Appl.*, vol. 226, Jun. 2024, Art. no. 103883.

[10] B. Serracanta, A. Rodriguez-Natal, F. Maino, and A. Cabellos, "Flow optimization at inter-datacenter networks for application run-time acceleration," 2024, *arXiv:2406.12567*.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[12] A. Shaikh, J. Rexford, and K. G. Shin, "Load-sensitive routing of long-lived IP flows," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 215–226, Oct. 1999.

[13] Y. Cao and M. Xu, "Dual-NAT: Dynamic multipath flow scheduling for data center networks," in *Proc. 21st IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2013, pp. 1–2.

[14] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM Conf.*, Aug. 2011, pp. 254–265.

[15] H. Xu, H. Huang, S. Chen, and G. Zhao, "Scalable software-defined networking through hybrid switching," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2017, pp. 1–9.

[16] P. Xiao, W. Qu, H. Qi, Y. Xu, and Z. Li, "An efficient elephant flow detection with cost-sensitive in SDN," in *Proc. 1st Int. Conf. Ind. Netw. Intell. Syst. (INISCom)*, Mar. 2015, pp. 24–28.

[17] P. Poupart, Z. Chen, P. Jaini, F. Fung, H. Susanto, Y. Geng, L. Chen, K. Chen, and H. Jin, "Online flow size prediction for improved network routing," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–6.

[18] W.-X. Liu, J. Cai, Y. Wang, Q. C. Chen, and J.-Q. Zeng, "Fine-grained flow classification using deep learning for software defined data center networks," *J. Netw. Comput. Appl.*, vol. 168, Oct. 2020, Art. no. 102766.

[19] M. Hamdan, B. Mohammed, U. Humayun, A. Abdelaziz, S. Khan, M. A. Ali, M. Imran, and M. N. Marsono, "Flow-aware elephant flow detection for software-defined networks," *IEEE Access*, vol. 8, pp. 72585–72597, 2020.

[20] H. He, Z. Peng, X. Zhou, and J. Wang, "LFOD: A lightweight flow table optimization scheme in SDN based on flow length distribution in the internet," in *Proc. 23rd Asia–Pacific Netw. Operations Manage. Symp. (APNOMS)*, Sep. 2022, pp. 1–6.

[21] Z. Qian, G. Gao, and Y. Du, "Per-flow size measurement by combining sketch and flow table in software-defined networks," in *Proc. IEEE Int. Conf Parallel Distrib. Process. Appl., Big Data Cloud Comput., Sustain. Comput. Commun., Social Comput. Netw. (ISPA/BDCloud/SocialCom/SustainCom)*, Dec. 2022, pp. 644–651.

[22] M. V. B. da Silva, A. S. Jacobs, R. J. Pfitscher, and L. Z. Granville, "Predicting elephant flows in internet exchange point programmable networks," in *Proc. Int. Conf. Adv. Inf. Netw. Appl.* Matsue, Japan: Springer, 2019, pp. 485–497.
[23] M. V. Brito da Silva, A. E. Schaeffer-Filho, and L. Z. Granville, "HashCuckoo: Predicting elephant flows using meta-heuristics in programmable data planes," in *Proc. GLOBECOM IEEE Global Commun. Conf.*, Dec. 2022, pp. 6337–6342.
[24] A. Pekar, A. Duque-Torres, W. K. G. Seah, and O. M. C. Rendon, "Towards threshold-agnostic heavy-hitter classification," *Int. J. Netw. Manage.*, vol. 32, no. 3, p. e2188, May 2022.
[25] B. L. Coelho and A. E. Schaeffer-Filho, "CrossBal: Data and control plane cooperation for efficient and scalable network load balancing," in *Proc. 19th Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2023, pp. 1–9.
[26] G. Wassie, J. Ding, and Y. Wondie, "Traffic prediction in SDN for explainable QoS using deep learning approach," *Sci. Rep.*, vol. 13, no. 1, p. 20607, Nov. 2023.
[27] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. New York, NY, USA: Association for Computing Machinery, Aug. 2016, p. 785.
[28] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.
[29] R. Durner and W. Kellerer, "Network function offloading through classification of elephant flows," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 2, pp. 807–820, Jun. 2020.
[30] C. Hardegen, B. Pfülb, S. Rieger, and A. Gepperth, "Predicting network flow characteristics using deep learning and real-world network traffic," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 4, pp. 2662–2676, Dec. 2020.
[31] C. Hardegen, B. Pfülb, S. Rieger, A. Gepperth, and S. Reißmann, "Flow-based throughput prediction using deep learning and real-world network traffic," in *Proc. 15th Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2019, pp. 1–9.
[32] J. Gómez, V. H. Riaño, and G. Ramirez-Gonzalez, "Traffic classification in IP networks through machine learning techniques in final systems," *IEEE Access*, vol. 11, pp. 44932–44940, 2023.
[33] S. Xie, G. Hu, C. Xing, and Y. Liu, "Online elephant flow prediction for load balancing in programmable switch-based DCN," *IEEE Trans. Netw. Service Manage.*, vol. 21, no. 1, pp. 745–758, Feb. 2024.
[34] N. Alkhalidi and F. Yaseen, "FDPHI: Fast deep packet header inspection for data traffic classification and management," *Int. J. Intell. Eng. Syst.*, vol. 14, no. 4, pp. 373–383, Aug. 2021.
[35] F. A. Yaseen, N. A. Alkhalidi, and H. S. Al-Raweshidy, "SHE networks: Security, health, and emergency networks traffic priority management based on ML and SDN," *IEEE Access*, vol. 10, pp. 92249–92258, 2022.
[36] P. Jurkiewicz, "Flow-models: A framework for analysis and modeling of IP network flows," *SoftwareX*, vol. 17, Jan. 2022, Art. no. 100929.
[37] J. Xu, J. Fan, M. Ammar, and S. B. Moon, "On the design and performance of prefix-preserving IP traffic trace anonymization," in *Proc. 1st ACM SIGCOMM Workshop Internet Meas.* New York, NY, USA: Association for Computing Machinery, 1145, p. 263.
[38] P. Jurkiewicz, "Flow-models 2.0: Elephant flows modeling and detection with machine learning," *SoftwareX*, vol. 24, Dec. 2023, Art. no. 101506.
[39] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.
[40] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Long Beach, CA, USA: Curran Associates, 2017, pp. 1–9.
[41] P. Jurkiewicz, "Boundaries of flow table usage reduction algorithms based on elephant flow detection," in *Proc. IFIP Netw. Conf. (IFIP Networking)*, Jun. 2021, pp. 1–9.
[42] B. Kądziolka, P. Jurkiewicz, R. Wójcik, and J. Domzaol, "Elephant flow classification on the first packet with neural networks," *IEEE Access*, vol. 12, pp. 65298–65309, 2024.
[43] S.-Y. Wang and Y.-H. Wu, "Supporting large random forests in the pipelines of a hardware switch to classify packets at 100-Gbps line rate," *IEEE Access*, vol. 11, pp. 112384–112397, 2023.

PIOTR JURKIEWICZ received the B.S. and M.S. degrees in electronics and telecommunications engineering from the AGH University of Krakow, Poland, in 2012 and 2015, respectively, where he is currently pursuing the Ph.D. degree with the Institute of Telecommunications. His research interests include software-defined networking, flow-based traffic engineering, and multipath and adaptive routing.

BARTOSZ KĄDZIOŁKA received the B.S. and M.S. degrees in electronics and telecommunications engineering from the AGH University of Krakow, Poland, in 2017 and 2019, respectively, where he is currently pursuing the Ph.D. degree with the Institute of Telecommunications. His research interests include software-defined networking, traffic engineering, and network optimization.

MIROSŁAW KANTOR received the M.S. and Ph.D. degrees in telecommunications from the AGH University of Krakow, Poland, in 2001 and 2010, respectively. In 2001, he joined the Institute of Telecommunications, AGH University of Krakow, where he is currently as an Assistant Professor. He has been a reviewer/a TPC member of international journals/conferences. He has actively participated in several European projects and grants supported by the Ministry of Science and Higher Education, Poland. He is the co-author of nearly 40 papers and two books. His research interests include networking and cybersecurity.

JERZY DOMŻAŁ received the M.S., Ph.D., and D.Sc. degrees in telecommunications from the AGH University of Krakow, Poland, in 2003, 2009, and 2016, respectively. Currently, he is a Professor and the Director of the Institute of Telecommunications, AGH University of Krakow. His international trainings were with the Universitat Politecnica de Catalunya, Barcelona, Spain, in April 2005; the Universidad Autónoma de Madrid, Madrid, Spain, in March 2009; and Stanford University, USA, from May 2012 to June 2012. He is the author or co-author of many technical papers, two patents applications, and two books. Especially, he is interested in optical networks and services for future internet.

ROBERT WÓJCIK received the Ph.D. and D.Sc. (Hons.) degrees in telecommunications from the AGH University of Krakow, Poland, in 2011 and 2019, respectively. Currently, he is a Professor with the Institute of Telecommunications, AGH University of Krakow. He was involved in several international EU-funded scientific projects, including SmoothIT, NoE BONE, Euro-NF, smaller, and national projects. He was a leader of three national science projects. He is the co-author of more than 70 research publications, including 21 research papers in JCR journals and several patents. His current research interests include multipath routing, flow-aware networking, quality of service, network neutrality. Recently, he has been working on several collaborative research projects involving specialists from industry and academia.