IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

An Improved Count-Based Classifier for Categorical Data

SANSKRITI SANJAY KUMAR SINGH[1], ALOK CHAUHAN[2]
[1]Student, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, India (e-mail: sanskriti.sanjaykumar2020@vitstudent.ac.in)
[2]Associate Professor, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, India (e-mail: alok.chauhan@vit.ac.in)

Corresponding author: Alok Chauhan (e-mail: alok.chauhan@vit.ac.in).

ABSTRACT The classification of categorical data is a fundamental task in machine learning, with numerous algorithms and techniques available. However, existing approaches often face challenges related to interpretability, scalability, and handling sparse or imbalanced datasets. This study presents an optimized version of the Count-Based Classifier, a novel approach that leverages the simplicity of counting occurrences to perform classification on categorical data. The optimized algorithm addresses the limitations of the original Count-Based Classifier, improving its computational efficiency, robustness, and overall performance. Through a comprehensive evaluation across 21 diverse categorical datasets, the Optimized Count-Based Classifier demonstrates competitive performance, consistently matching or surpassing established classifiers such as Decision Trees, Support Vector Machines, etc. The classifier's inherent interpretability, stemming from its reliance on counting operations, is a valuable asset, particularly in domains where transparency and explainability are crucial. Furthermore, the study explores the classifier's characteristics, including its tendency for overfitting, result consistency, and robustness against label errors. Experimental analyses reveal a low propensity for overfitting, high result consistency, and remarkable resilience to mislabeled data, further solidifying the classifier's practical applicability. The Optimized Count-Based Classifier has been implemented in Python and deployed as a user-friendly package, fostering accessibility and adoption within the machine learning community. By addressing the limitations of traditional approaches and offering a simple yet effective solution, this work contributes to the advancement of count-based classification techniques and their application in real-world scenarios.

INDEX TERMS Algorithm design, classification algorithm, count-based classifier, machine learning, supervised learning.

I. INTRODUCTION

In the realm of supervised learning, a diverse array of classification models exist, each with its own strengths and limitations. While traditional techniques like Naïve Bayes, Decision Trees, and Support Vector Machines have been widely employed, they often face challenges when dealing with categorical data and complex, high-dimensional datasets. More advanced approaches, such as Random Forests, Neural Networks, and Gradient Boosting, have been developed to address some of these limitations, but they may still struggle with issues like overfitting, sensitivity to noise, and computational complexity.

Amidst these challenges, the concept of counting, a fundamental aspect of human cognition, presents a unique solution. By quantifying and analyzing the frequencies of specific attributes, count-based approaches offer a fresh perspective for understanding datasets and extracting valuable insights. The Count-Based Classifier, introduced by Agarwal et al. [1] and inspired by the principles of "Vastu Moolak Ganit" described in the recent Indian philosophy of Madhyasth Darshan by Late Shri A Nagraj [2], harnesses the innate power of counting to make classification decisions for data.

The Count-Based Classifier for Categorical Data operates by iteratively examining attribute values in the training dataset and updating class counts based on occurrences. During classification, it selects the class with the higher count for each attribute value, repeating the process for all attributes and choosing the class with the most votes as the predicted class. While demonstrating efficiency in certain scenarios, the current implementation of the Count-Based Classifier also presents several drawbacks and limitations, such as longer execution times compared to other classifiers, potential scalabil-

ity issues with larger datasets, and over-reliance on individual attribute counts making the classifier susceptible to noise or outliers and miss broader patterns or distributions of classes across the dataset.Thus, offering opportunities for improving its underlying logic and attribute-value relationship handling.

The primary objective of this research is to optimize and refine the existing Count-Based Classifier, improving its computational efficiency, accuracy, and interpretability. Furthermore, this study aims to conduct a comprehensive evaluation of the classifier's characteristics, strengths, weaknesses, and potential real-world applications through carefully designed experiments. Ultimately, the goal is to deploy the optimized classifier as a user-friendly package, contributing to the advancement of count-based classification techniques within the machine learning ecosystem.

II. LITERATURE REVIEW

A. OVERVIEW OF EXISTING CLASSIFIERS

1) Traditional and Ensemble Learning Classifiers

Traditional classification algorithms, such as Naïve Bayes, Decision Trees, and Support Vector Machines, have been widely employed for predictive modeling tasks across various domains. However, these models often face limitations when dealing with categorical data, particularly in scenarios involving sparse or high-dimensional datasets [3]–[6]. Additionally, the complex architectures of some of these algorithms

TABLE 1. Overview of Existing Classification Techniques for Categorical Data

Classifier	Approach	Drawbacks	Training Time Complexity	Testing Time Complexity
Perceptron	A linear classification algorithm that learns a decision boundary to separate different classes. It adjusts weights based on misclassifications to minimize errors [9].	Sensitive to outliers, not suitable for complex data [10].	O(n*d)	O(p*d)
Decision Tree	Uses a tree-like model to make decisions. It recursively splits the data based on feature values, creating a tree structure. Leaf nodes represent class labels [11].	Prone to overfitting, sensitive to small variations in the data [5].	O(n*d*log(n))	O(p*h)
K-Nearest Neighbours (KNN)	Classifies samples based on the majority class among their k-nearest neighbours in the feature space [12].	Sensitive to noisy data, high computation for large datasets [4].	O(1)	O(p*d*k)
Support Vector Machine (SVM)	Finds a hyperplane that best separates classes in the feature space. It maximizes the margin between different classes [13].	Memory-intensive for large datasets, sensitive to noise [14], [15].	O(n2*d)–O(n3*d)	O(p*d*v)
Naive Bayes	Applies Bayes' theorem with the assumption of feature independence. Computes class probabilities based on the product of individual feature probabilities [16].	Assumes independence of features, can be sensitive to irrelevant features [6].	O(n*d)	O(p*d*c)
Logistic Regression	Models the probability of the binary outcome using the logistic function. It fits a linear decision boundary to the data [17].	Assumes linear relationship, sensitive to outliers [18].	O(n*d)	O(p*d)
Multilayer Perceptron (MLP)	A type of neural network with multiple layers. It uses backpropagation to adjust weights and biases, learning complex patterns in the data [19].	Prone to overfitting, sensitive to initial weights [20].	O(n*d*le*i)	O(p*d*l*e)
AdaBoost	Constructs an ensemble by combining weak learners. It assigns higher weights to misclassified samples, focusing on correcting errors [21].	Sensitive to noisy data, can be slow to train [8].	O(n*m*t)	O(m*t)
Random Forest	Builds an ensemble of decision trees. Each tree is constructed using a random subset of features and the final prediction is the majority vote [22].	May overfit noisy data, slow to make predictions [23].	O(n*d*log(n)*t)	O(p*h*t)
Gradient Boosting	Builds a series of weak learners, each correcting the errors of the previous one. It combines them to form a strong predictive model [24].	Prone to overfitting, computationally expensive [7], [25].	O(n*d*log(n)*i)	O(p*h*t)
Extremely Randomized Trees (Extra Trees)	Similar to random forests, but the decision trees are built using random subsets of features and random thresholds at each split [26].	Can have high variance, slow to make predictions [27].	O(n*d*log(n)*t)	O(p*h*t)
eXtreme Gradient Boosting (XGBoost)	A gradient boosting algorithm that uses a regularized objective function and parallel processing [28].	Computationally expensive, sensitive to hyperparameters [7], [29].	O(n*d*log(n)*i)	O(p*h*t)
Light Gradient Boosting Machine (LightGBM)	A gradient boosting algorithm that splits the tree leafwise and employs a histogram-based approach for faster training [30].	May require fine-tuning, sensitive to noisy data [7], [31].	O(n*d*b*i)	O(p*h*t)
Category Boosting (CatBoost)	A gradient boosting algorithm designed for categorical features. It includes regularization techniques [32].	Slow training, may require hyperparameter tuning [7].	O(n*d*log(n)*i)	O(p*h*t)

n = number of training instances, d = number of features, p = number of test instances; h = depth of decision trees, k = number of neighbors, v = number of support vectors, c = number of classes, l = number of hidden layers, e = number of neurons per hidden layer, i = number of boosting iterations/epochs, m = base estimator's time complexity, t = number of estimators/trees in ensemble methods, b = number of bins used for histogram-based splitting.

can lead to challenges in interpretability, hampering their widespread applicability and stakeholder trust.

More advanced techniques, including Random Forests, Neural Networks, and Gradient Boosting algorithms like AdaBoost, XGBoost, LightGBM, and CatBoost, have been developed to address some of the limitations of traditional classifiers. These algorithms leverage ensemble methods and advanced optimization techniques to improve classification performance. However, they may still struggle with specific challenges, such as overfitting, sensitivity to noisy data, and computational complexity, particularly in high-dimensional or large-scale datasets [7], [8].

Table 1 provides a concise overview of various traditional and ensemble learning classifiers for categorical data, highlighting their approach, drawbacks, and computational complexities.

2) Deep Learning Classifiers

Deep learning for categorical data remains an under-explored area in machine learning. TabNet [33] represents an attempt to apply Deep Neural Network (DNN) architecture to categorical data. The model employs gradient-descent based optimization for training and utilizes a sequential attention technique for feature selection, aiming to improve interpretability. TabNet demonstrates some performance improvements by leveraging unsupervised pre-training for prediction tasks.

However, TabNet faces several challenges. Like many deep learning approaches, it typically requires substantial computational resources and large datasets to achieve optimal performance. The model's complexity can make it difficult to interpret, despite efforts to improve transparency through attention mechanisms. While TabNet's soft feature selection technique may offer advantages over simpler architectures, the model's reliance on gradient-based learning can make it sensitive to initialization and prone to local optima. Furthermore, the effectiveness of its self-supervised learning capabilities may vary depending on the specific task and dataset, potentially limiting its generalizability across different domains.

B. EMERGING APPROACHES AND CONSIDERATIONS

Recent research in categorical data classification has explored various techniques to enhance performance and address specific challenges in the field. Association-based methods have gained attention for their ability to leverage relationships within categorical data. These techniques, often based on Association Rule Mining (ARM), aim to uncover frequent patterns or associations that can inform classification decisions [34]. While promising, these methods may face scalability issues when dealing with large-scale datasets containing numerous categorical attributes.

The impact of bias in datasets on classifier performance has emerged as a critical area of study, particularly given the increasing use of Machine Learning in various applications affecting daily life [35]. Researchers are exploring techniques to identify and mitigate unfair discrimination in ML models, often through data preprocessing, algorithm modifications, or post-processing of model outputs. However, addressing bias often involves complex trade-offs between fairness and overall accuracy, presenting a significant challenge in the development of ethical ML systems.

The influence of data preprocessing techniques, such as discretization, on classification outcomes has also been a focus of recent research. Studies have shown that the choice of preprocessing method can significantly affect classifier performance, highlighting the importance of these steps in the overall classification pipeline. This is particularly relevant for algorithms that primarily work with categorical data, as the transformation of continuous attributes to discrete ones can impact the final results [36]–[38].

These emerging approaches and considerations bring light to the ongoing complexities in categorical data classification. They highlight the need for methods that can balance performance improvements with computational efficiency and interpretability. As research in this field progresses, addressing these challenges remains a key objective for developing more robust and widely applicable algorithms for categorical data classification.

C. NEED FOR COUNT-BASED CLASSIFICATION

As mentioned before, current classifiers often face challenges when handling categorical data, particularly in scenarios involving sparse attribute values or high-dimensional feature spaces, because they can lead to poor performance for classifiers, prompting the need for a different approach. Agarwal et al. [1] developed the Count-Based Classifier to address this challenge by focusing on frequency counts of attribute values, enabling it to effectively handle sparse categorical data and ensuring robust and accurate classification outcomes.

Moreover, interpretability is a crucial aspect, especially in domains where stakeholders require transparency in classification decisions. The Count-Based Classifier offers inherent interpretability due to its reliance on counting occurrences of attribute values. This transparency allows users to comprehend the classifier's logic, understand how attributes contribute to predictions, and establishes trust in the model's predictions. This characteristic becomes particularly valuable in applications where interpretability is a priority, such as in healthcare, finance, or legal domains.

Scalability is another key consideration, especially as datasets grow in size and complexity. The Count-Based Classifier's reliance on counting operations makes it well-suited for scalability, particularly in high-dimensional spaces. By efficiently processing and summarizing attribute frequencies, the classifier can handle large-scale categorical datasets without compromising performance. This scalability aspect is crucial for real-world applications dealing with extensive and diverse data.

Furthermore, the Count-Based Classifier facilitates domain-specific insights by quantifying attribute occurrences. This feature becomes invaluable in domains where attribute frequencies hold significant meaning or provide domain-specific insights. By uncovering patterns, trends,

and associations within categorical data, the classifier aids domain experts in making informed decisions and deriving actionable insights. This aspect of the classifier is particularly beneficial for guiding future improvements and applications.

D. EXISTING IMPLEMENTATION OF COUNT-BASED CLASSIFIER

The original implementation of the Count-Based Classifier, proposed by Agarwal et al. [1], revolves around a straightforward algorithm designed to classify categorical data based on attribute frequencies and class counts. However, this implementation also exhibits certain limitations and opportunities for further exploration and optimization. Algorithm 1, as given by Agarwal et al. [1], provides us with the existing Count-Based Classifier's algorithm.

Algorithm 1 Original Count-Based Classifier

```
for each attributeValue in testDataset do
  for each class do
    if present in lookup then
      get classCount
    else
      for each instance in trainingDataset do
        if attributeValue missing then
          ignore
        else
          get classCount
          increment classCount
        end if
      end for
    end if
    update lookup
  end for
end for
predictedClass ← max(attributeValue) // leading in more attributes
if tie then
  predictedClass ← max(totalCount) // leads in total occurrences across all attributes
  if tie then
    predictedClass ← any(class)
  end if
end if
```

The current logic involves calculating the total count of each attribute value for each unique class in the training set. For instance, for an attribute like 'weather,' the attribute class counts may be represented as follows: rainy: [yes: 3, no: 5], sunny: [yes: 8, no: 4], ...

During classification, for each test instance, the algorithm selects the class with the higher count for each attribute value. For example, if a test instance has attributes 'weather: sunny, windy: true, cloudy: false,' the algorithm would look at the class counts for 'sunny' (e.g., [yes: 8, no: 4]) and assign a vote to 'yes' due to its higher count. This process is repeated for each attribute, and the class with the most votes is selected as the predicted class for the test instance. In cases of ties in class votes, the algorithm considers aggregated class counts across attributes to break the tie and make a prediction.

While the current implementation provides a baseline for classification, it has certain limitations and loopholes. For instance, it seems to take a lot of time to run as compared to other classifiers. Moreover, the algorithm's efficiency and scalability may be improved to handle larger datasets and complex attribute-value relationships more effectively. These aspects highlight the need for optimization and refinement in the Count-Based Classifier's logic and implementation.

III. METHODOLOGY

A. OPTIMIZATION OF THE EXISTING COUNT-BASED CLASSIFIER

To enhance the performance and overcome limitations in the original Count-Based Classifier, an optimized version of the algorithm has been developed. This optimized algorithm, as depicted in Algorithm 2, employs a different approach to classification, focusing on efficient tie-breaking mechanisms and improved scalability. It basically reverses the logic of the original count-based classifier, which leads to the classifier capturing the overall distribution and importance of each class and avoid over-reliance on individual attribute counts for initial classification by using attribute-level voting only in case of ties in aggregated class counts. It also ensures that the overall class distributions less affected by localized noise/errors.

In this optimized logic, we first calculate the total count of each class based on the aggregated counts of attribute values across all predictors. If only one class has the maximum total count, we directly predict that class for the instance. Otherwise, we choose the class with the maximum count from the attribute votes.

This inversion of logic aims to address the shortcomings of the original classifier, particularly in handling tie-breaking, over-reliance on individual attribute counts, sesitivity to noise or outliers and improving computational efficiency.

B. CORRECTNESS PROOF FOR THE OPTIMIZED COUNT-BASED CLASSIFIER

To prove the correctness of the Optimized Count-Based Classifier, we utilize the Pigeonhole Principle, a fundamental principle in mathematics. The Pigeonhole Principle states that if there are more pigeons (objects) than pigeonholes (containers), and all the pigeons are placed in the pigeonholes, then at least one pigeonhole must contain more than one pigeon [39].

In the context of the Optimized Count-Based Classifier, we can consider the following:

- The pigeons are the instances in the dataset.
- The pigeonholes are the classes in the target variable.

During the **training phase**, the classifier builds a count lookup table (countdict) that stores the counts of each distinct category within each feature and class label. This count lookup table essentially assigns each instance (pigeon) to its

Algorithm 2 Optimized Count-Based Classifier

```
for each predictor in trainingSet do
  for each class in trainingSet[Target] do
    for each predictorValue in predictor do
      countLookUp[predictorValue, class] ← count(instances containing predictorValue and class)
    end for
  end for
end for
for each instance in testSet do
  for each predictorValue in instance do
    for each class in trainingSet[Target] do
      totalCount[class] ← totalCount[class] + countLookUp[predictorValue, class]
    end for
    attributeVote[predictorValue] ← class having max(countLookUp[predictorValue, class])
  end for
  if only one class has totalCount = max(totalCount[class]) then
    predictedClass ← class having totalCount=max(totalCount[class])
  else
    predictedClass ← class having max(attributeVote)
  end if
end for
```

corresponding class (pigeonhole) based on the feature values and the target class label.

Let us denote the set of input instances as X and the set of class labels as Y. The count lookup table can be formally represented as:

$$countdict : X \times Y \rightarrow \mathbb{N}$$

Where $\mathbb{N}$ is the set of natural numbers (including zero), and $countdict(x, y)$ represents the count of instances with feature values x and class label y in the training data.

During the **prediction phase**, for a new instance $x' \in X$, the classifier calculates the total count for each class $y \in Y$ as:

$$total\_count(y) = \sum countdict(x', y)$$

Where the summation is taken over all feature values x' in the input instance.

Case 1: If there exists a unique class $y^* \in Y$ such that $total\_count(y^*) > total\_count(y)\ \forall\ y \neq y^*$, then the CBC algorithm correctly assigns the instance x' to the class y^*. This can be proved using the Pigeonhole Principle as follows:

- Let n be the total number of instances in the training data. Since each instance belongs to exactly one class, we can consider the instances as pigeons and the classes as pigeonholes.
- According to the Pigeonhole Principle, if there are n pigeons (instances) and k pigeonholes (classes), and $n > k$, then at least one pigeonhole must contain more than (n/k) pigeons.
- In our case, the class y^* with the maximum $total\_count(y^*)$ must contain more than $(n/|Y|)$ instances, where $|Y|$ is the number of classes.

Therefore, y^* is the correct class to assign the instance x'.

Case 2: If there are multiple classes $y_1, y_2, ..., y_m \in Y$ such that $total\_count(y_1) = total\_count(y_2) = ... = total\_count(y_m) = \max(total\_count(y))$, then the CBC algorithm uses the feature votes (*feature_vote*) to break the tie.

Formally, for each feature value x' in the input instance x', the CBC algorithm calculates the most frequent class (vote) as:

$$feature\_vote(x') = argmax(countdict(x', y)); y \in Y.$$

The predicted class is then determined as:

$$predicted\_class = argmax(\sum feature\_vote(x'); y \in Y, x' \in X.$$

This can be proved using the Pigeonhole Principle by considering the feature votes as a secondary level of pigeonholes within each class.

According to the principle, if there are more instances (pigeons) than classes (pigeonholes) in the tied scenario, at least one class (pigeonhole) must have more feature votes assigned to it than the others.

Case 3: If $total\_count(y) = 0\ \forall\ y \in Y$ (i.e., no occurrences for any class in any feature), the CBC algorithm assigns equal probabilities to all classes, which is a valid approach since there is no discriminating information available.

Hence, by making use of the Pigeonhole Principle and the principles of counting and majority voting, the Optimized Count-Based Classifier ensures that instances are assigned to the correct class based on the available information in the count lookup table and feature votes, thus proving its correctness for categorical data classification.

C. IMPLEMENTATION OF OPTIMIZED COUNT-BASED CLASSIFIER

The Optimized Count-Based Classifier has been implemented as a Python module named CategoricalCBC). This module provides functionality for fitting the classifier to training data, predicting class probabilities for test data, and making class predictions based on the optimized logic described above.

The CategoricalCBC class inherits from BaseEstimator and ClassifierMixin in scikit-learn, ensuring compatibility with scikit-learn's API. Key methods include fit for training the classifier, predict_proba for predicting class probabilities, and predict for making class predictions.

Furthermore, the classifier has been thoroughly tested using sample datasets, along with the use of check_estimator method offered by scikit-learn, to ensure its correctness and efficiency. The fit method constructs a count lookup table from the training set, while the predict_proba and predict methods apply the optimized logic to predict class probabilities and make class predictions, respectively.

D. DEPLOYMENT OF OPTIMIZED COUNT-BASED CLASSIFIER

The Optimized Count-Based Classifier has been packaged into a Python module named CountEst. This module is available for installation via the package managers PyPI and conda-forge.

Users can install the CountEst package using the following commands:

- From Python Package Index (PyPI): *pip install countest* [40]
- From Anaconda: *conda install -c conda-forge countest* [41]

IV. EXPERIMENTAL SETUP

To evaluate the performance and characteristics of the Optimized Count-Based Classifier, a comprehensive set of experiments were designed and conducted using a diverse collection of categorical datasets.

A. DATASETS

An extensive search for purely categorical tabular datasets was conducted using various online resources, including the UCI Machine Learning Repository, Kaggle, Google Dataset Search Tool, and ICPSR website. After thorough exploration, 21 datasets meeting the set criteria were selected for evaluating the classifiers' performance. Table 2 provides an overview of these datasets, including their subject areas, number of instances, features, classes, and information about class imbalance.

During the search, a scarcity of open-source, fully categorical datasets with a large number of classes was encountered. As seen in Table 2, most available datasets had only 2-4 classes, with only two datasets having 17 and 15 classes. To maintain the integrity of our evaluation, we refrained from generating synthetic multi-class datasets, as they might not accurately capture the complexities and nuances of real-world categorical data. This limitation is acknowledged, and further evaluation on suitable multi-class datasets will be conducted as they become available.

B. PREPROCESSING

Before conducting the experiments, the datasets underwent the following necessary preprocessing steps:

TABLE 2. Overview of Categorical Datasets being used for the Experiments

Dataset Number	Dataset Name	Subject Area	Number of Instances	Number of Features	Number of Classes	Class Imbalance
1	Mushroom Dataset	Biology	8124	22	2	No
2	Car Evaluation Dataset	Other	1728	6	4	Yes
3	Breast Cancer Dataset	Health and Medicine	286	9	2	Yes
4	Congressional Voting Records Dataset	Social Science	435	16	2	No
5	Tic-Tac-Toe Endgame Dataset	Games	958	9	2	Yes
6	Nursery Dataset	Social Science	12960	8	4	No
7	Soybean (Large) Dataset	Biology	307	35	17	Yes
8	Molecular Biology (Promoter Gene Sequences) Dataset	Biology	106	58	2	No
9	Balance Scale Dataset	Social Science	625	4	3	No
10	Lenses Dataset	Other	24	4	2	No
11	Molecular Biology (Splice-junction Gene Sequences) Dataset	Biology	3190	60	3	Yes
12	SPECT Heart Dataset	Health and Medicine	267	22	2	No
13	Primary Tumor Dataset	Health and Medicine	339	17	15	Yes
14	Chess (King-Rook vs. King-Pawn) Dataset	Games	3196	36	2	Yes
15	Lymphography Dataset	Health and Medicine	148	18	2	Yes
16	Hayes-Roth Dataset	Social Science	160	4	3	No
17	Lung Cancer Prediction Dataset	Health and Medicine	1000	22	3	No
18	Phishing Website Dataset	Computer Science	11054	30	2	Yes
19	Monkey-Pox Patients Dataset	Health and Medicine	25000	9	2	Yes
20	Animal Condition Classification Dataset	Health and Medicine	871	6	2	No
21	Android Malware Detection	Computer Science	4465	241	2	Yes

- **Missing Value Imputation:** Missing values in the datasets were imputed using the KNN Imputer, which estimates missing values based on the nearest neighbors' feature values.
- **Removing Rare Classes:** To ensure a meaningful analysis, instances belonging to classes with fewer than four occurrences were removed from the datasets.
- **Encoding Categorical Variables:** The categorical feature values were encoded using an Ordinal Encoder, which assigns a unique integer value to each category while preserving the ordinal relationship between categories, if present.
- **Handling Class Imbalance:** For datasets exhibiting class imbalance, the Synthetic Minority Over-sampling Technique (SMOTE) was employed to oversample the minority class instances, thereby mitigating the impact of imbalanced class distributions on the classifiers' performance.

C. EXPERIMENTS

A series of experiments were conducted to evaluate the Optimized Count-Based Classifier's performance, computational complexity, robustness, and compare it with existing classification techniques. The experiments are as follows:

1) Comparative Performance Analysis with the Original Count-Based Classifier

To establish a baseline for evaluating the Optimized Count-Based Classifier's performance, a comparative analysis is conducted against the Original Count-Based Classifier proposed by Agarwal et al. [1]. The classifiers are evaluated on the Mushroom, Car Evaluation, and Nursery datasets (i.e. the datasets that the original classifier was evaluated upon) using the metrics of accuracy, recall, precision, F1-score, and time taken for training and testing. Fig. 1 depicts the workflow for this experiment, illustrating the process for a dataset. This is applied across all three datasets.

2) Comparative Time Complexity Analysis with Existing Classifiers

To assess the computational efficiency of the Optimized Count-Based Classifier, a comparative analysis of its time complexity is conducted against various established classifiers. The analysis focused on the theoretical time complexities for both training and testing phases, as reported in the literature review.

3) Comparative Performance Analysis with Existing Classifiers

To evaluate the Optimized Count-Based Classifier's performance and compare it with established techniques, a comprehensive comparative analysis is conducted across the 21 categorical datasets. The Optimized Count-Based Classifier is evaluated against several widely used classifiers, which are Decision Trees, K-Nearest Neighbors, Support Vector Machines, Naive Bayes, Logistic Regression, Multilayer Perceptron, AdaBoost, Random Forests, Gradient Boosting, Extra Trees, and XGBoost.

The classifiers are trained and evaluated on each dataset, and their performance was measured using various metrics, including accuracy, precision, recall, and F1-score. Additionally, the time taken by each classifier for training and testing is to be recorded too. Fig. 2 depicts the workflow for this experiment, illustrating the process for a dataset. This is applied across all 21 datasets.

4) Assessing Classifier Overfitting Tendency

To assess the Optimized Count-Based Classifier's tendency to overfit, a 5-fold cross-validation is performed on the training dataset for each of the 21 datasets. The average training

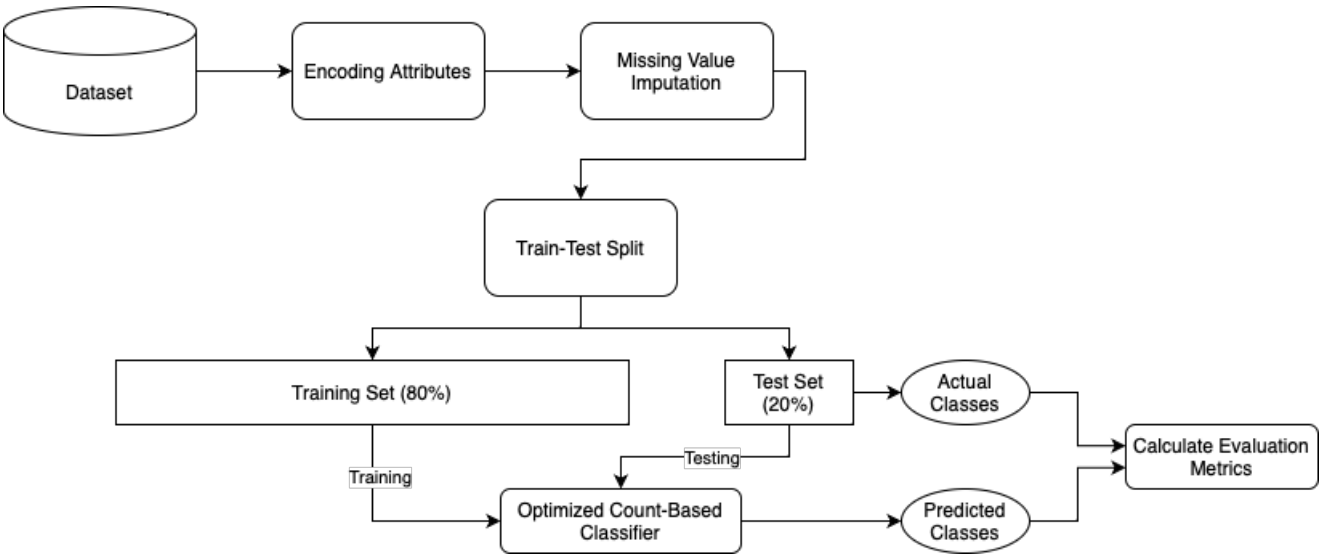

FIGURE 1. Workflow for Experiment 1

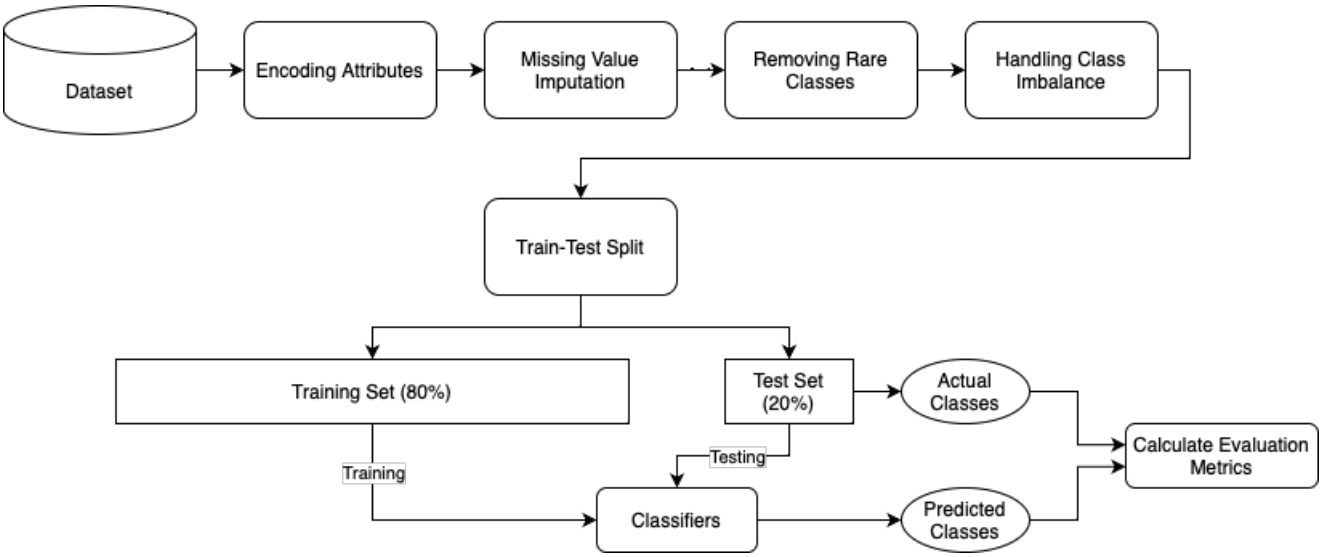

FIGURE 2. Workflow for Experiment 3

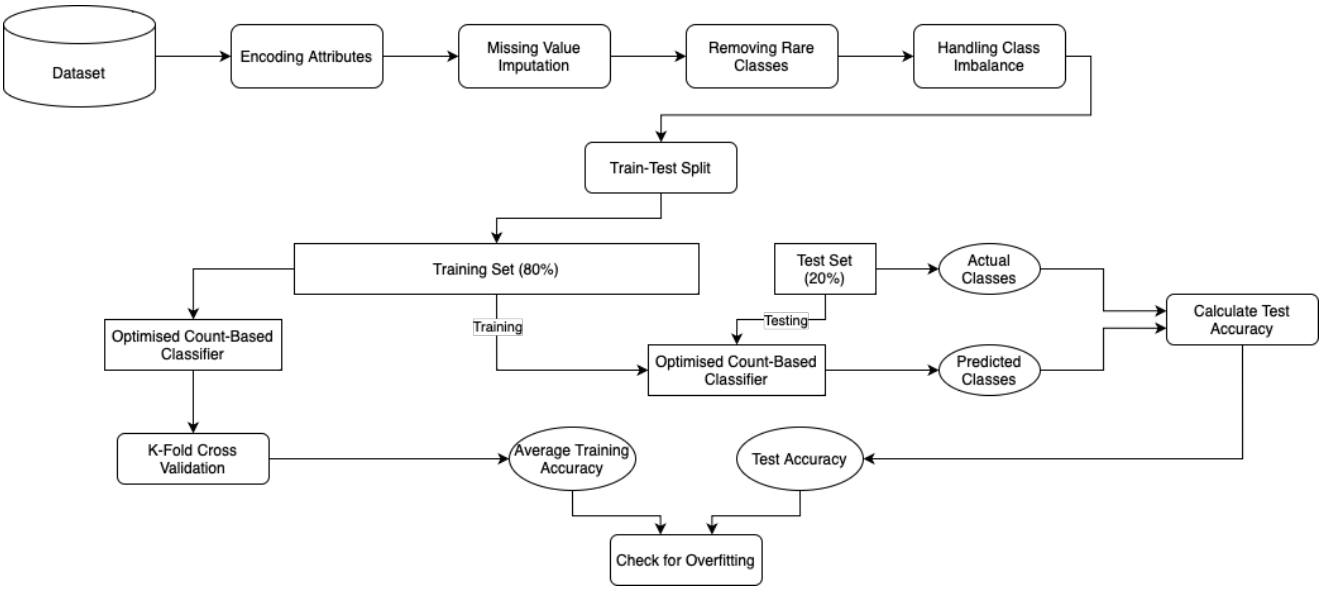

FIGURE 3. Workflow for Experiment 4

accuracy obtained from the cross-validation is compared with the test accuracy obtained by training the classifier on the entire training set and evaluating on the test set. A significant difference between the training and test accuracies would indicate potential overfitting. Fig. 3 depicts the workflow for this experiment, illustrating the process for a dataset. This is applied across all 21 datasets.

5) Assessing Classifier Result Consistency

To evaluate the consistency of the Optimized Count-Based Classifier's results, the classifier is trained and evaluated 100 times for each of the 21 datasets, each time using a different random state for the train-test split (80:20 ratio). The mean and standard deviation of the evaluation metrics (accuracy) across the 100 iterations are then calculated to assess the consistency of the classifier's performance. Fig. 4 depicts the workflow for this experiment, illustrating the process for a dataset. This is applied across all 21 datasets.

6) Robustness Analysis using Cleanlab for Detecting Label Errors

To assess the robustness of the Optimized Count-Based Classifier against label errors, the Cleanlab library is employed. For each dataset, potential label errors are identified using the find_label_issues function from Cleanlab, making use of the Optimized Count-Based Classifier for the identification of label errors. The instances with identified label errors are then removed from the dataset and also corrected in the case

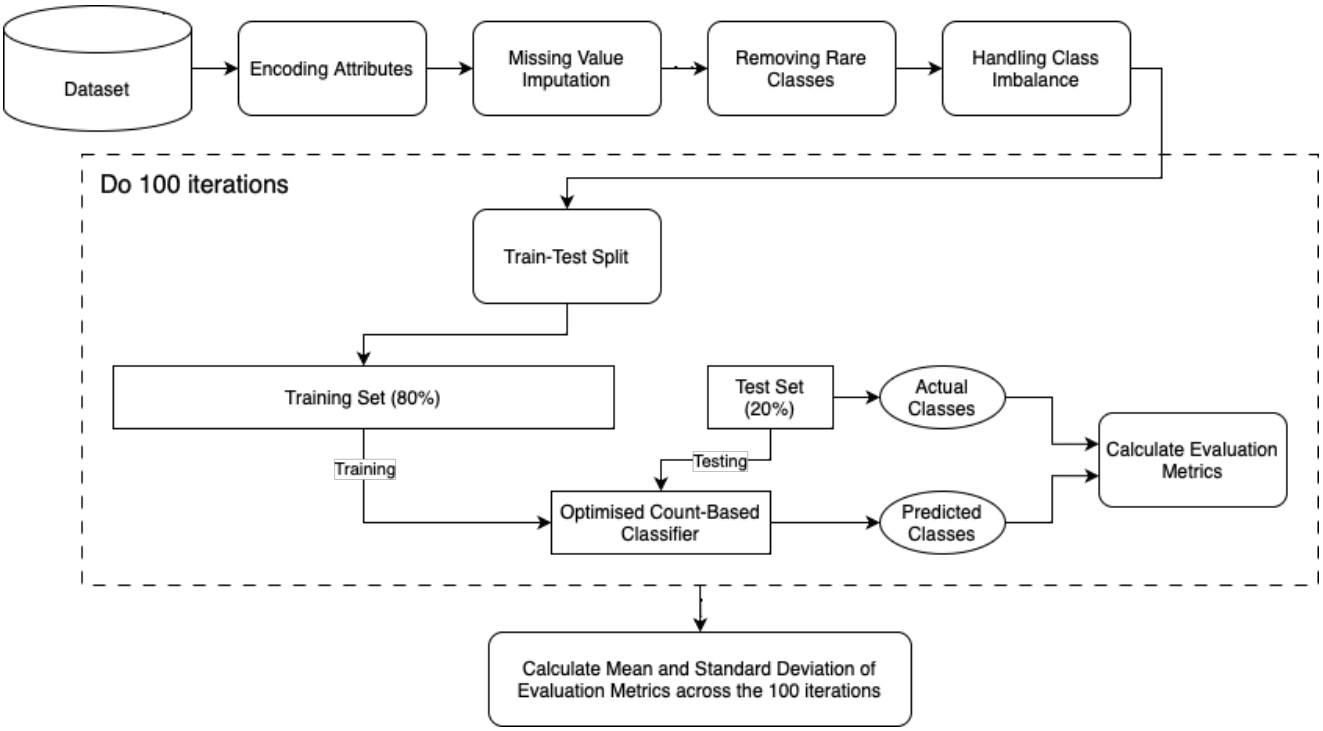

FIGURE 4. **Workflow for Experiment 5**

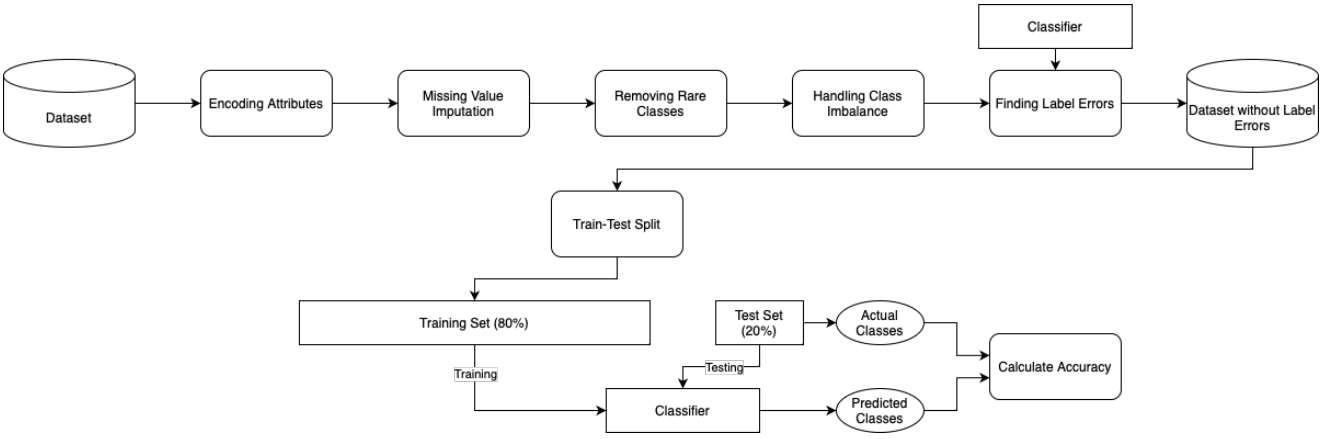

FIGURE 5. **Workflow for Experiment 6**

of binary classification datasets.

Following which the classifier is trained and evaluated on the cleaned dataset. To provide context, the same process was repeated for Decision Tree Classifier, K-Nearest Neighbours Classifier, Support Vector Machine Classifier, Naive Bayes Classifier, Logistic Regression, Multilayer Perceptron, AdaBoost Classifier, Random Forest Classifier, Gradient Boosting Classifier, Extra Trees Classifier and XGBoost Classifier. The performance of the classifiers on the original and cleaned datasets is then compared, and the classifier with the least difference in performance across all datasets is considered the most robust against label errors. Fig. 5 depicts the workflow for this experiment, illustrating the process for a dataset. This is applied across all 21 datasets.

V. RESULTS AND DISCUSSION

The Optimized Count-Based Classifier's performance was extensively evaluated through a series of carefully designed experiments. This section presents the outcomes of these experiments, providing insights into the classifier's capabilities, strengths, and limitations. The experiments encompassed comparative analyses against the Original Count-Based Classifier proposed by Agarwal et al. [1], as well as comparisons with a diverse set of established classification techniques. Additionally, the experiments aimed to assess the optimized Count-Based Classifier's computational complexity,

tendency to overfit, consistency in results, and robustness against label errors.

The evaluation was conducted across a diverse collection of 21 categorical datasets, spanning various subject areas such as biology, health and medicine, games, social sciences, and computer science. These datasets exhibited varying levels of complexity, including differences in the number of instances, features, classes, and class imbalance. By subjecting the Optimized Count-Based Classifier to this diverse range of datasets, a comprehensive understanding of its performance characteristics and applicability in different real-world scenarios was obtained.

The results and subsequent discussions are organized into separate subsections, each focusing on a specific aspect of the evaluation process. These subsections provide detailed analyses, visualizations, and interpretations of the experimental findings, shedding light on the Optimized Count-Based Classifier's performance relative to the Original Count-Based Classifier and other established techniques. Furthermore, the discussions delve into the implications of these findings, highlighting the classifier's strengths, weaknesses, and potential areas for improvement or further research.

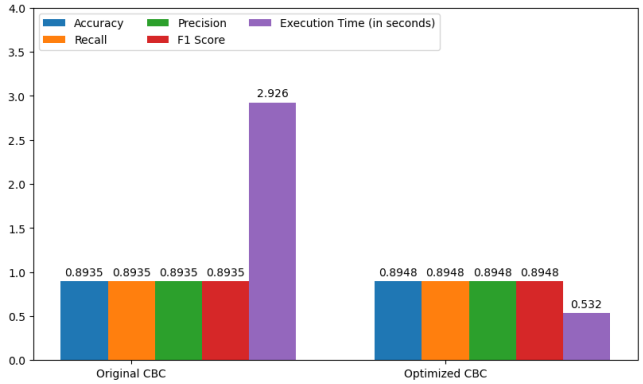

FIGURE 6. Performance Comparison of Optimized and Original Count-Based Classifiers for Mushroom Dataset

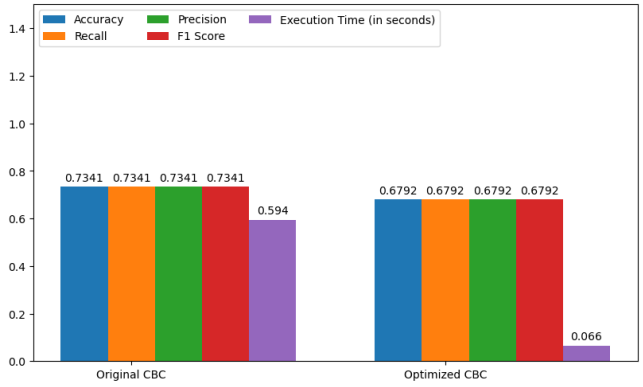

FIGURE 7. Performance Comparison of Optimized and Original Count-Based Classifiers for Car Evaluation Dataset

A. EXPERIMENT 1: COMPARATIVE PERFORMANCE ANALYSIS WITH THE ORIGINAL COUNT-BASED CLASSIFIER

The results of the comparative performance analysis of the optimized Count-Based Classifier with the original Count-Based Classifier for all the three datasets are presented in Table 3 and visualized in the Figs. 6-8.

For the Mushroom dataset, as seen in Table 3 and Figure 6, the Optimized Count-Based Classifier achieved a slightly higher Accuracy, Recall, Precision, and F1-Score of 0.8948 compared to 0.8935 for the Original Count-Based Classifier. However, the most significant improvement was observed in the Execution Time, where the Optimized Count-Based Classifier took only 0.532 seconds, which is approximately 5.5 times faster than the Original Count-Based Classifier's execution time of 2.926 seconds.

For the Car Evaluation dataset, as seen in Table 3 and Figure 7, the Original Count-Based Classifier outperformed the Optimized Count-Based Classifier in terms of Accuracy, Recall, Precision, and F1-Score, with a score of 0.7341 compared to 0.6792. Nevertheless, the Optimized Count-Based Classifier exhibited a faster Execution Time of 0.066 seconds, which is 9 times faster than the Original Count-Based Classifier's execution time of 0.594 seconds.

On the Nursery dataset, as seen in Table 3 and Figure 8, the Optimized Count-Based Classifier significantly outperformed the Original Count-Based Classifier across all evaluation metrics. The Optimized Count-Based Classifier achieved an Accuracy, Recall, Precision, and F1-Score of 0.8673, which is substantially higher than the Original Count-Based Classifier's score of 0.4236. Additionally, the Optimized Count-Based Classifier exhibited a significantly faster Execution Time of 0.606 seconds compared to the Original Count-Based Classifier's execution time of 3.217 seconds.

Overall, the results from Experiment 1 demonstrated that the Optimized Count-Based Classifier consistently achieved faster execution times across all datasets while maintaining comparable or better performance in terms of Accuracy,

TABLE 3. Performance comparison of Optimized and Original Count-Based Classifiers

Dataset	Model	Accuracy	Recall	Precision	F1-Score	Execution Time (s)
Mushroom	Original Count-Based Classifier	0.8935	0.8935	0.8935	0.8935	2.926
	Optimized Count-Based Classifier	0.8948	0.8948	0.8948	0.8948	0.532
Car Evaluation	Original Count-Based Classifier	0.7341	0.7341	0.7341	0.7341	0.594
	Optimized Count-Based Classifier	0.6792	0.6792	0.6792	0.6792	0.066
Nursery	Original Count-Based Classifier	0.4236	0.4236	0.4236	0.4236	3.217
	Optimized Count-Based Classifier	0.8673	0.8673	0.8673	0.8673	0.606

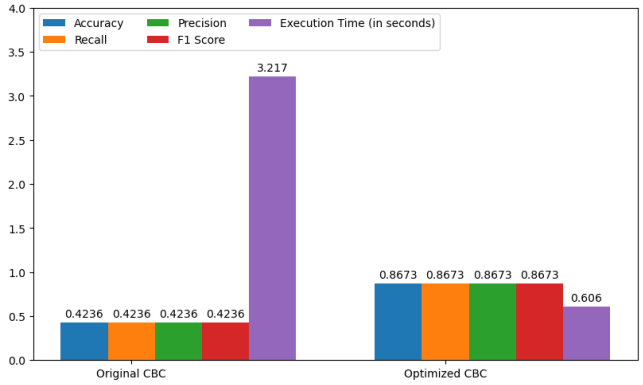

FIGURE 8. Performance Comparison of Optimized and Original Count-Based Classifiers for Nursery Dataset

Recall, Precision, and F1-Score. The performance improvement was particularly notable on the Mushroom and Nursery datasets, where the Optimized Count-Based Classifier exhibited significant improvements in both classification performance and execution time compared to the Original Count-Based Classifier.

B. EXPERIMENT 2: COMPARATIVE TIME COMPLEXITY ANALYSIS WITH EXISTING CLASSIFIERS

The time complexity of the Optimized Count-Based Classifier is determined by the number of features (d), the number of classes (c), the size of the training dataset (n), and the size of the test dataset (p). During the training phase, the time complexity is O(n*d*c), as the classifier must iterate through each feature and class to construct the count lookup table, which has a time complexity of O(n). This analysis assumes that the number of unique values per feature is bounded by a constant, allowing the complexity to be expressed in terms of the feature and class count alone. In the testing phase, the time complexity is O(p*d*c), reflecting the need to process each test instance, consider each feature, and evaluate the predictions against all possible class labels to produce the final output.

Table 4 presents a comparative overview of the training and testing time complexities across the Optimized Count-Based Classifier and various other classification models. From the table, it can be observed that the training time complexity of the Optimized Count-Based Classifier, which is linear in the number of training instances, features, and classes, is comparable to the time complexity of simpler models, such as Perceptron and Naive Bayes.

During the testing phase, the Optimized Count-Based Classifier has a time complexity of O(p*d*c), which is linear in the number of test instances, features, and classes. This is more efficient than some of the more complex models, such as Decision Trees and ensemble methods, which exhibit higher testing time complexities. From the analysis, it is evident that the time complexity of the Optimized Count-Based Classifier is relatively straightforward and does not depend on additional hyperparameters or model-specific factors, unlike some of the more sophisticated classifiers.

Overall, the time complexity analysis demonstrates that the Optimized Count-Based Classifier, while being a relatively simple model, can offer competitive computational efficiency compared to more advanced algorithms, particularly during the testing phase. This makes the Optimized Count-Based Classifier a potentially attractive choice for applications with strict time constraints or limited computational resources.

TABLE 4. Comparative Time Complexity Analysis

Classifier	Training Time Complexity	Testing Time Complexity
Perceptron	O(n*d)	O(p*d)
Decision Tree	O(n*d*log(n))	O(p*h)
KNN	O(1)	O(p*d*k)
SVM	O(n2*d)–O(n3*d)	O(p*d*v)
Naive Bayes	O(n*d)	O(p*d*c)
Logistic Regression	O(n*d)	O(p*d)
MLP	O(n*d*le*i)	O(p*d*l*e)
AdaBoost	O(n*m*t)	O(m*t)
Random Forest	O(n*d*log(n)*t)	O(p*h*t)
Gradient Boosting	O(n*d*log(n)*i)	O(p*h*t)
Extra Trees	O(n*d*log(n)*t)	O(p*h*t)
XGBoost	O(n*d*log(n)*i)	O(p*h*t)
LightGBM	O(n*d*b*i)	O(p*h*t)
CatBoost	O(n*d*log(n)*i)	O(p*h*t)
Optimized CBC	O(n*d*c)	O(p*d*c)

n = number of training instances, d = number of features, p = number of test instances; h = depth of decision trees, k = number of neighbors, v = number of support vectors, c = number of classes, l = number of hidden layers, e = number of neurons per hidden layer, i = number of boosting iterations/epochs, m = base estimator's time complexity, t = number of estimators/trees in ensemble methods, b = number of bins used for histogram-based splitting.

C. EXPERIMENT 3: COMPARATIVE PERFORMANCE ANALYSIS WITH EXISTING CLASSIFIERS

The results of the comparative performance analysis of the Optimized Count-Based Classifier with the existing classifiers, presented in Table 5, paint a compelling picture of the Optimized Count-Based Classifier's capabilities. Across the 21 datasets, the count-based approach demonstrated robust and competitive performance, often rivaling and occasionally outperforming more complex classification models.

One particularly noteworthy finding was the count-based classifier's strong performance on the SPECT Heart dataset, where it achieved the highest scores across all metrics (accuracy: 83.3%, precision: 91.7%, recall: 85.3%, F1: 84.4%) compared to the other classifiers. This suggests that the count-based logic is well-suited for capturing the underlying patterns in certain healthcare and medical datasets, potentially making it a valuable tool for decision support in such domains.

Similarly, on the Monkey-Pox Patients dataset, the Count-Based Classifier tied for the highest performance across all metrics (64.5%) along with the Gradient Boosting classifier. This further highlights the potential of the count-based ap-

TABLE 5. Comparative Performance Analysis Results with Existing Classifiers

Dataset	Evaluation Metric	Best Performing Classifier(s)	Score obtained by Best Performing Classifier(s)	Worst Performing Classifier(s)	Score obtained by Worst Performing Classifier(s)	Score obtained by CBC
Mushroom	Accuracy	DT, MLP, AB, RF, GB, ET, XGB	1.000	CBC	0.895	0.895
	Precision	DT, MLP, AB, RF, GB, ET, XGB	1.000	CBC	0.915	0.915
	Recall	DT, MLP, AB, RF, GB, ET, XGB	1.000	CBC	0.891	0.891
	F1	DT, MLP, AB, RF, GB, ET, XGB	1.000	CBC	0.893	0.893
Car Evaluation	Accuracy	XGB	0.999	LR	0.563	0.596
	Precision	XGB	0.999	LR	0.521	0.632
	Recall	XGB	0.999	LR	0.543	0.598
	F1	XGB	0.999	LR	0.526	0.609
Breast Cancer	Accuracy	ET	0.790	SVM, LR	0.580	0.728
	Precision	ET	0.798	SVM, LR	0.571	0.716
	Recall	ET	0.778	SVM, LR	0.569	0.712
	F1	ET	0.782	SVM, LR	0.569	0.713
Congressional Voting Records	Accuracy	LR, XGB	0.989	NB, CBC	0.897	0.897
	Precision	LR, XGB	0.991	NB, CBC	0.886	0.886
	Recall	LR, XGB	0.984	NB, CBC	0.891	0.891
	F1	LR, XGB	0.987	NB, CBC	0.888	0.888
Tic-Tac-Toe Endgame	Accuracy	RF	0.992	LR	0.578	0.669
	Precision	RF	0.992	LR	0.578	0.670
	Recall	RF	0.992	LR	0.578	0.669
	F1	RF	0.992	LR	0.578	0.669
Nursery	Accuracy	XGB	1.000	LR	0.770	0.867
	Precision	XGB	1.000	LR	0.695	0.916
	Recall	XGB	1.000	LR	0.610	0.763
	F1	XGB	1.000	LR	0.612	0.833
Soybean (Large)	Accuracy	RF	0.985	AB	0.309	0.748
	Precision	RF	0.987	AB	0.305	0.798
	Recall	RF	0.986	AB	0.353	0.709
	F1	RF	0.986	AB	0.327	0.751
Molecular Biology (Promoter Gene Sequences)	Accuracy	DT, SVM, LR, MLP, AB, RF, GB, XGB	1.000	CBC	0.773	0.773
	Precision	DT, SVM, LR, MLP, AB, RF, GB, XGB	1.000	CBC	0.844	0.844
	Recall	DT, SVM, LR, MLP, AB, RF, GB, XGB	1.000	CBC	0.773	0.773
	F1	DT, SVM, LR, MLP, AB, RF, GB, XGB	1.000	CBC	0.760	0.760
Balance Scale	Accuracy	MLP	0.976	DT	0.760	0.896
	Precision	MLP	0.955	DT	0.568	0.931
	Recall	AB	0.954	DT	0.556	0.755
	F1	MLP	0.944	DT	0.560	0.834
Lenses	Accuracy	All Classifiers	0.400	All Classifiers	0.400	0.400
	Precision	All Classifiers	0.333	All Classifiers	0.333	0.333
	Recall	All Classifiers	0.250	All Classifiers	0.250	0.250
	F1	All Classifiers	0.286	All Classifiers	0.286	0.286
Molecular Biology (Splice-junction Gene Sequences)	Accuracy	GB, XGB	0.977	KNN	0.719	0.758
	Precision	GB, XGB	0.977	KNN	0.722	0.818
	Recall	GB, XGB	0.976	KNN	0.704	0.753
	F1	GB, XGB	0.977	KNN	0.711	0.784
SPECT Heart	Accuracy	CBC	0.833	DT	0.648	0.833
	Precision	CBC	0.917	DT	0.515	0.917
	Recall	CBC	0.853	DT	0.522	0.853
	F1	CBC	0.844	DT	0.510	0.844
Primary Tumor	Accuracy	SVM	0.817	AB	0.305	0.623
	Precision	SVM	0.823	AB	0.413	0.666
	Recall	SVM	0.818	AB	0.308	0.622
	F1	SVM	0.812	AB	0.197	0.547
Chess (King-Rook vs. King-Pawn)	Accuracy	DT	0.999	NB, CBC	0.869	0.869
	Precision	DT	0.998	NB, CBC	0.870	0.870
	Recall	DT	0.998	NB, CBC	0.870	0.870
	F1	DT	0.998	NB, CBC	0.870	0.870
Lymphography	Accuracy	SVM	0.939	DT, KNN, LR	0.848	0.879
	Precision	SVM	0.939	DT, KNN, LR	0.850	0.879
	Recall	SVM	0.939	DT, KNN, LR	0.847	0.879
	F1	SVM	0.939	DT, KNN, LR	0.848	0.879
Hayes-Roth	Accuracy	DT	0.875	LR	0.406	0.756
	Precision	DT	0.892	LR	0.479	0.809
	Recall	DT	0.892	LR	0.378	0.751
	F1	DT	0.892	LR	0.406	0.761

DT = Decision Tree Classifier, KNN = K-Nearest Neighbours Classifier, SVM = Support Vector Machine Classifier, NB = Naive Bayes Classifier, LR = Logistic Regression, MLP = Multilayer Perceptron, AB = AdaBoost Classifier, RF = Random Forest Classifier, GB = Gradient Boosting Classifier, ET = Extra Trees Classifier, XGB = XGBoost Classifier, CBC = Count-Based Classifier

TABLE 5. Comparative Performance Analysis Results with Existing Classifiers (Continued)

Dataset	Evaluation Metric	Best Performing Classifier(s)	Score obtained by Best Performing Classifier(s)	Worst Performing Classifier(s)	Score obtained by Worst Performing Classifier(s)	Score obtained by CBC
Lung Cancer Prediction	Accuracy	DT, SVM, LR, MLP, RF, GB, ET, XGB	1.000	AB	0.725	0.930
	Precision	DT, SVM, LR, MLP, RF, GB, ET, XGB	1.000	AB	0.845	0.948
	Recall	DT, SVM, LR, MLP, RF, GB, ET, XGB	1.000	AB	0.667	0.923
	F1	DT, SVM, LR, MLP, RF, GB, ET, XGB	1.000	AB	0.565	0.931
Phishing Website	Accuracy	ET	0.975	CBC	0.902	0.902
	Precision	ET	0.975	CBC	0.905	0.905
	Recall	ET	0.975	CBC	0.899	0.899
	F1	ET	0.975	CBC	0.898	0.898
Monkey-Pox Patients	Accuracy	GB, CBC	0.645	KNN	0.608	0.645
	Precision	GB, CBC	0.645	KNN	0.608	0.645
	Recall	GB, CBC	0.645	KNN	0.608	0.645
	F1	GB, CBC	0.645	KNN	0.608	0.645
Animal Condition Classification	Accuracy	AB	0.994	NB, XGB	0.971	0.989
	Precision	AB	0.997	NB, XGB	0.597	0.994
	Recall	AB	0.750	NB, XGB	0.491	0.738
	F1	AB	0.831	NB, XGB	0.494	0.817
Android Malware Detection	Accuracy	RF, ET	0.993	CBC	0.791	0.791
	Precision	RF, ET	0.993	CBC	0.849	0.849
	Recall	RF, ET	0.993	CBC	0.797	0.797
	F1	RF, ET	0.993	CBC	0.784	0.784

DT = Decision Tree Classifier, KNN = K-Nearest Neighbours Classifier, SVM = Support Vector Machine Classifier, NB = Naive Bayes Classifier, LR = Logistic Regression, MLP = Multilayer Perceptron, AB = AdaBoost Classifier, RF = Random Forest Classifier, GB = Gradient Boosting Classifier, ET = Extra Trees Classifier, XGB = XGBoost Classifier, CBC = Count-Based Classifier

proach in specific problem settings.

The Count-Based Classifier also showed competitive performance on several other datasets. For instance, it achieved the second-highest accuracy (98.9%) on the Animal Condition Classification dataset, very close to the top-performing AdaBoost classifier. Additionally, it performed well on datasets like Congressional Voting Records and Phishing Website, where it was among the top performers.

Further, upon examining the Time Taken Matrix in Fig. 9, it is evident that the Optimized Count-Based Classifier exhibits competitive execution times compared to several complex algorithms, such as AdaBoost, Random Forest, Gradient Boosting, Extra Trees, and XGBoost. While it may not consistently outperform all algorithms in terms of execution time, it demonstrates efficiency in various instances. Notably, the Optimized Count-Based Classifier is the fastest on the Lenses Dataset (Dataset 10) and the second-fastest on the Hayes-Roth Dataset (Dataset 16). This performance highlights its computational efficiency relative to various complex models, although simpler algorithms like Decision Tree and

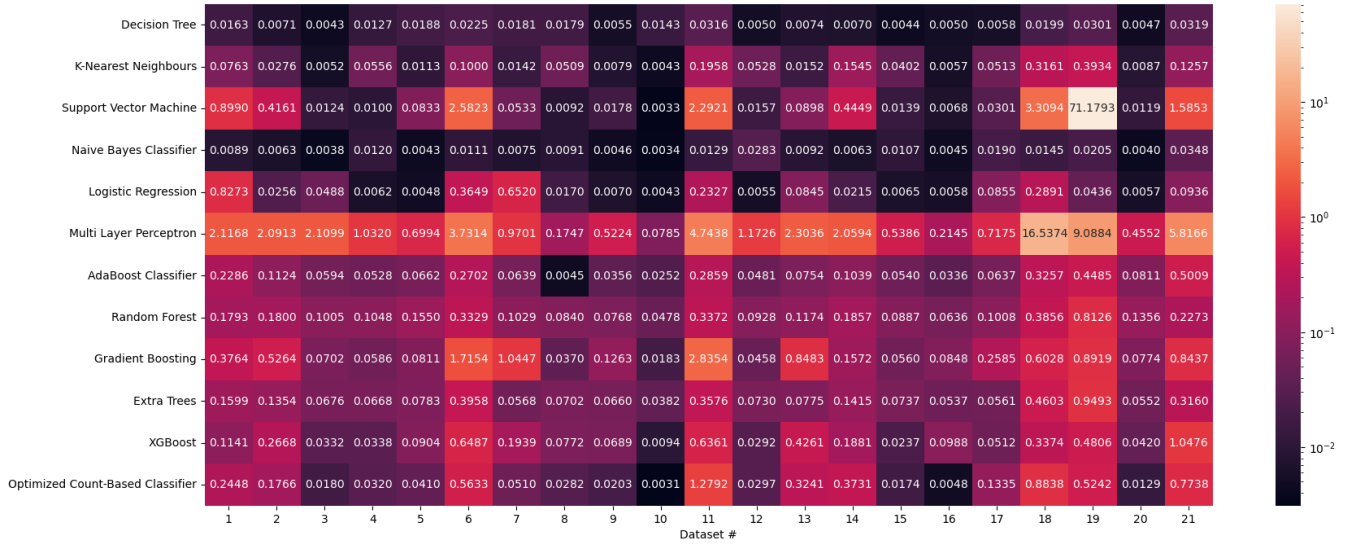

FIGURE 9. Time Taken Matrix for Performance Analysis of the Classifiers

Naive Bayes often show superior execution times across most datasets. It is important to note that, unlike many of the compared algorithms which utilize parallelization techniques, our Optimized Count-Based Classifier has not yet been implemented with parallelization. This suggests potential for further performance improvements in future iterations of the proposed algorithm.

The comparative analysis also revealed some areas where the Optimized Count-Based Classifier could potentially be further enhanced. For example, on the Mushroom dataset and the Android Malware Detection dataset, the count-based approach lagged behind the top-performing models in terms of accuracy. This suggests that there may be opportunities to refine the classifier's logic or explore additional optimization techniques to improve its performance on datasets with higher complexity or dimensionality.

Overall, the results of this experiment highlight the Optimized Count-Based Classifier's robust performance, computational efficiency, and versatility across a wide range of categorical datasets. While there is always room for improvement, the count-based approach has demonstrated its potential as a viable and competitive classification technique, particularly in domains where interpretability and simplicity are desirable characteristics.

D. EXPERIMENT 4: ASSESSING CLASSIFIER OVERFITTING TENDENCY

It can be observed, from the results presented in Table 6 and visualized in Fig. 10, that for most datasets, the cross-validation average accuracy and the test accuracy are relatively close, suggesting that the Optimized Count-Based Classifier does not exhibit a strong tendency to overfit and is able to generalize well to unseen data.

However, there are a few instances where the differences between the cross-validation average accuracy and the test accuracy are more pronounced. For instance, in Dataset 7 (Soybean (Large) Dataset), the cross-validation average accuracy is 69.44%, while the test accuracy is 61.03%, indicating a potential overfitting issue. Similarly, in Dataset 5 (Tic-Tac-Toe Endgame Dataset), the cross-validation average accuracy is 67.14%, but the test accuracy is slightly lower at 64.94%. Also, in Dataset 10 (Lenses Dataset), the cross-validation average accuracy is 65%, while the test accuracy is 40%, although this discrepancy may not necessarily indicate overfitting but rather the limitations imposed by the dataset's size since it comprises of only 24 instances.

Interestingly, for Dataset 12 (SPECT Heart Dataset), the test accuracy of 83.33% is higher than the cross-validation average accuracy of 78.41%, suggesting that the classifier may be underfitting on this dataset. Similarly, in Dataset 3 (Breast Cancer Dataset), Dataset 14 (Chess (King-Rook vs. King-Pawn) Dataset) and Dataset 15 (Lymphography Dataset), the cross-validation average accuracies are lower than the corresponding test accuracies, yet again implying potential underfitting.

TABLE 6. Cross-Validation Average Accuracy and Test Accuracy for the Datasets

Dataset Number	K-Fold CV Average Accuracy (K=5)	Test Accuracy (obtained from previous experiments)
1	0.8989	0.8948
2	0.6095	0.5961
3	0.6821	0.7284
4	0.8879	0.8966
5	0.6714	0.6494
6	0.8640	0.8673
7	0.6944	0.6103
8	0.7853	0.7727
9	0.9000	0.8960
10	0.6500	0.4000
11	0.7553	0.7583
12	0.7841	0.8333
13	0.6337	0.6349
14	0.8303	0.8713
15	0.8449	0.8788
16	0.6415	0.6563
17	0.9175	0.9300
18	0.9069	0.9022
19	0.6328	0.6488
20	0.9756	0.9886
21	0.7826	0.7814

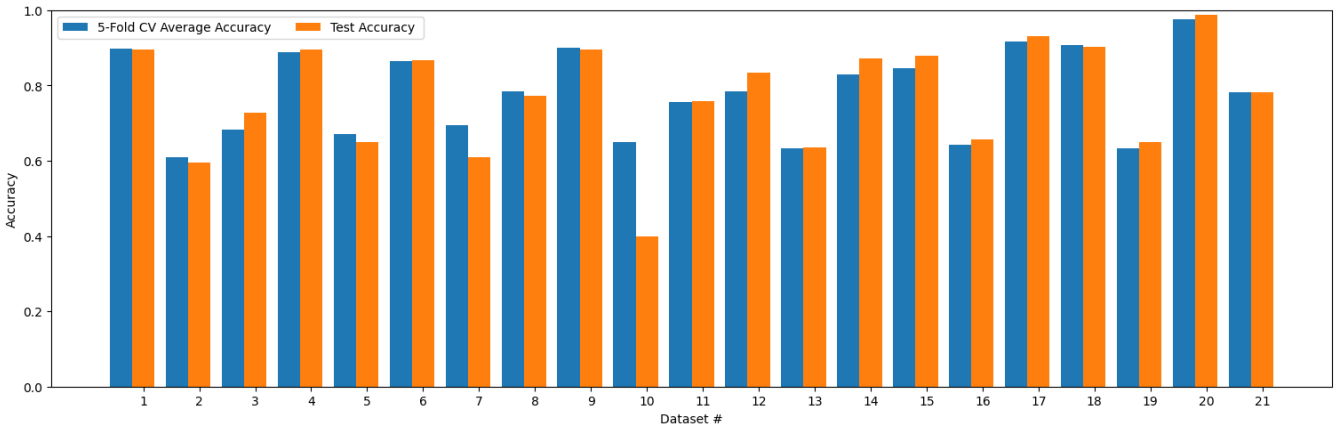

FIGURE 10. Cross-Validation Average Accuracy and Test Accuracy for the Datasets

Nevertheless, these discrepancies range from 1% to 5%, indicating minimal concern. Such variations are expected due to cross-validation's nature as a generalization estimator and possible minor data distribution differences. While not requiring immediate action, the classifier's performance on future unseen data will be closely monitored. Should significant underfitting persist in subsequent applications, strategies such as feature engineering, data augmentation or ensemble learning will be considered for mitigation.

Thus, the results indicate that while the Optimized Count-Based Classifier does not exhibit a strong tendency for overfitting in most cases, there are certain datasets where overfitting or underfitting may occur. It is essential to carefully monitor and address these instances to ensure optimal performance and generalization capabilities.

E. EXPERIMENT 5: ASSESSING CLASSIFIER RESULT CONSISTENCY

Table 7 presents the results of this experiment, including the test accuracy obtained from previous experiments, the mean accuracy across the 100 iterations, and the standard deviation of the accuracy.

Overall, the results indicate that the Optimized Count-Based Classifier exhibits a relatively high degree of consistency in its results. For most datasets, the standard deviation of the accuracy is relatively low, suggesting that the classifier's performance does not vary significantly across different train-test splits.

However, there are a few datasets where the standard deviation is relatively higher, indicating greater variability in the classifier's performance. For example, in Dataset 8 (Molecular Biology (Promoter Gene Sequences) Dataset), the standard deviation of the accuracy is 0.1874, which is relatively high compared to other datasets. Similarly, Dataset 15 (Lymphography Dataset) has a standard deviation of 0.1131, and Dataset 10 (Lenses Dataset) has a standard deviation of 0.1701.

Also, it is noteworthy that for some datasets, such as Dataset 1 (Mushroom Dataset) and Dataset 19 (Monkey-Pox Patients Dataset), the standard deviation is extremely low, suggesting highly consistent performance across different train-test splits.

Thus, in general, while the Optimized Count-Based Classifier exhibits a reasonably high degree of consistency in its results, there are certain datasets where its performance may be more variable. Further investigation and analysis may be required to identify the factors contributing to this variability and explore potential mitigation strategies to enhance the classifier's consistency across different scenarios.

TABLE 7. Previous Test Accuracy along with Mean Accuracy and Standard Deviation across the 100 iterations

Dataset Number	Test Accuracy (obtained from previous experiments)	Mean Accuracy	Standard Deviation
1	0.8948	0.8968	0.0050
2	0.5961	0.6283	0.0242
3	0.7284	0.6775	0.0501
4	0.8966	0.8867	0.0319
5	0.6494	0.6618	0.0280
6	0.8673	0.8700	0.0110
7	0.6103	0.7184	0.0849
8	0.7727	0.7200	0.1874
9	0.8960	0.8938	0.0264
10	0.4000	0.5160	0.1701
11	0.7583	0.7493	0.0321
12	0.8333	0.7939	0.0545
13	0.6349	0.5321	0.0808
14	0.8713	0.7577	0.0764
15	0.8788	0.7558	0.1131
16	0.6563	0.6197	0.0938
17	0.9300	0.9067	0.0267
18	0.9022	0.9084	0.0080
19	0.6488	0.6309	0.0054
20	0.9886	0.9786	0.0101
21	0.7814	0.8973	0.0862

F. EXPERIMENT 6: ROBUSTNESS ANALYSIS USING CLEANLAB FOR DETECTING LABEL ERRORS

The performance of the classifiers on the original and cleaned datasets has been compared in Table 8. As illustrated by Fig. 11, the Optimized Count-Based Classifier demonstrated a commendable level of robustness against label errors across the evaluated datasets. On average, the change in accuracy after removing the identified label errors was only 4.89% and after correcting the label errors was 7.67%, which was among the lowest across all classifiers. This suggests that the Optimized Count-Based Classifier is capable of maintaining its performance even in the presence of noisy or mislabeled data, making it a reliable choice for real-world applications where label quality cannot be guaranteed.

In contrast, some of the more complex classifiers, such as Decision Tree, Logistic Regression, and Multilayer Perceptron, exhibited larger changes in accuracy, as can be observed in Figs. 12-14, indicating a greater sensitivity to label errors. For instance, the Decision Tree Classifier's accuracy improved by an average of 12.79% after label error removal, and the Logistic Regression model's accuracy increased by an average of 28.75% across the datasets.

The Optimized Count-Based Classifier's robustness can be attributed to its inherent simplicity and reliance on counting operations, which are less susceptible to being misled by noisy or erroneous labels compared to more sophisticated machine learning models that might overfit to the training data. This property of the Optimized Count-Based Classifier is particularly valuable in domains where data quality is a concern or where manual label verification is infeasible.

Furthermore, the Optimized Count-Based Classifier was observed to have the lowest average change in accuracy across all datasets, with a mere 4.89% difference between the original and cleaned datasets. This suggests that the Optimized Count-Based Classifier consistently maintained its performance, regardless of the presence of label errors, making it a reliable choice for classification tasks in challenging real-world scenarios.

TABLE 8. Summary of the Performance of Classifiers on Original and Cleaned Datasets

Classifier	Average Accuracy	Average Accuracy after Label Error Removal	Average % Change in Accuracy after Label Error Removal	Average Accuracy after Label Error Correction	Average % Change in Accuracy after Label Error Correction
Decision Tree	0.873	0.955	12.794	0.903	9.707
K-Nearest Neighbours	0.821	0.908	10.669	0.923	15.540
Support Vector Machine	0.880	0.956	12.557	0.935	13.435
Naive Bayes Classifier	0.845	0.962	19.392	0.934	22.803
Logistic Regression	0.804	0.961	28.752	0.964	25.713
Multi Layer Perceptron	0.879	0.968	15.497	0.940	17.217
AdaBoost Classifier	0.775	0.884	14.641	0.929	16.723
Random Forest	0.891	0.971	14.051	0.966	18.338
Gradient Boosting	0.891	0.944	8.695	0.934	13.558
Extra Trees	0.890	0.948	9.332	0.920	9.533
XGBoost	0.892	0.972	13.882	0.970	18.760
Count-Based Classifier	0.783	0.811	4.895	0.843	7.670

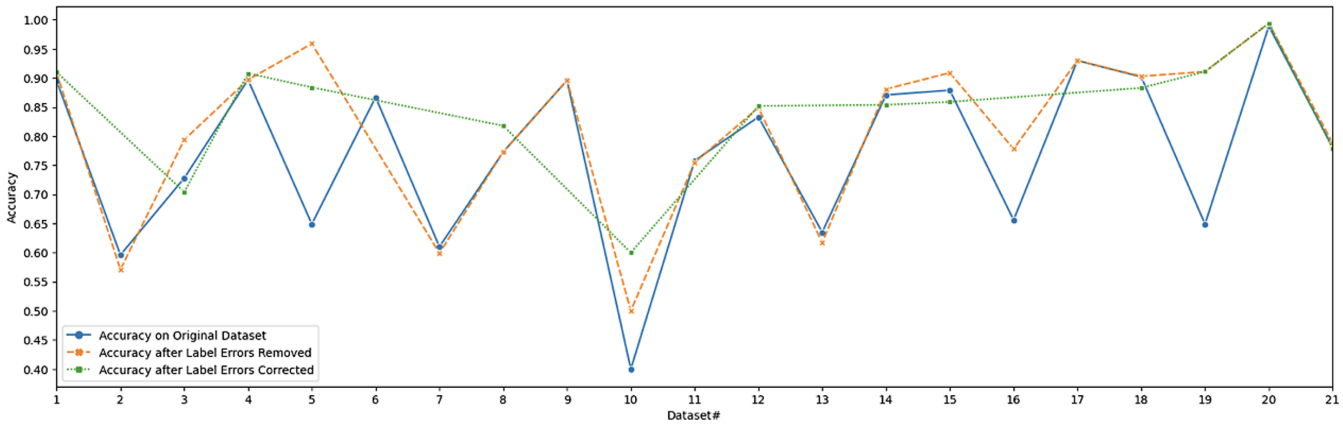

FIGURE 11. Accuracy of Optimized Count-Based Classifier on Original Dataset, Dataset without Label Errors and Dataset with Corrected Label Errors

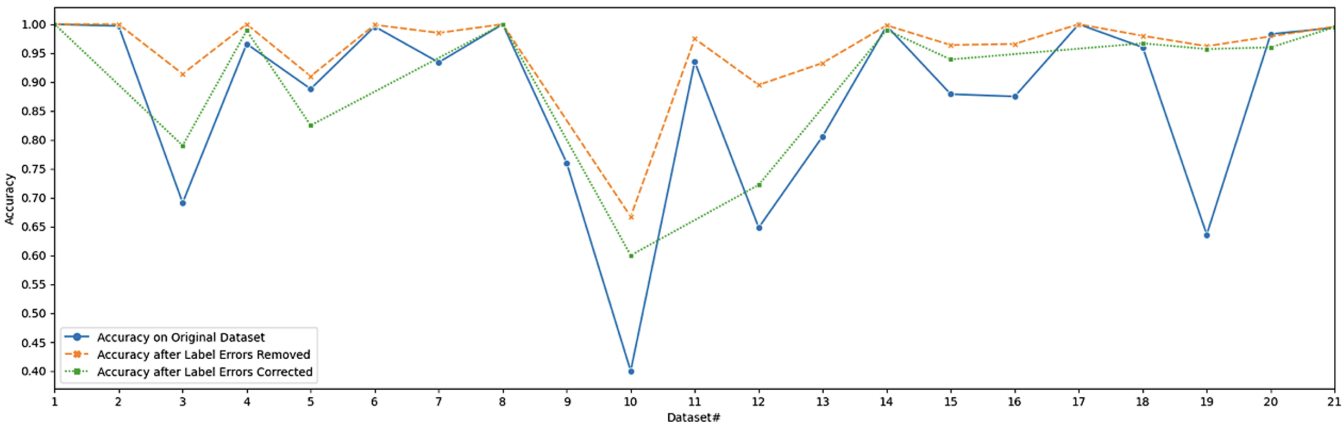

FIGURE 12. Accuracy of Decision Tree Classifier on Original Dataset, Dataset without Label Errors and Dataset with Corrected Label Errors

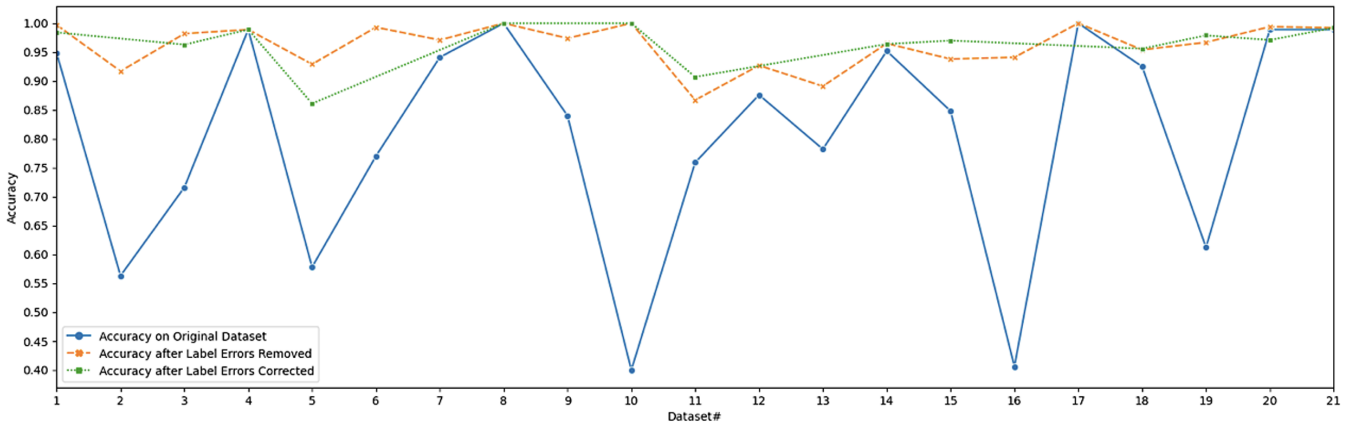

FIGURE 13. Accuracy of Logistic Regression on Original Dataset, Dataset without Label Errors and Dataset with Corrected Label Errors

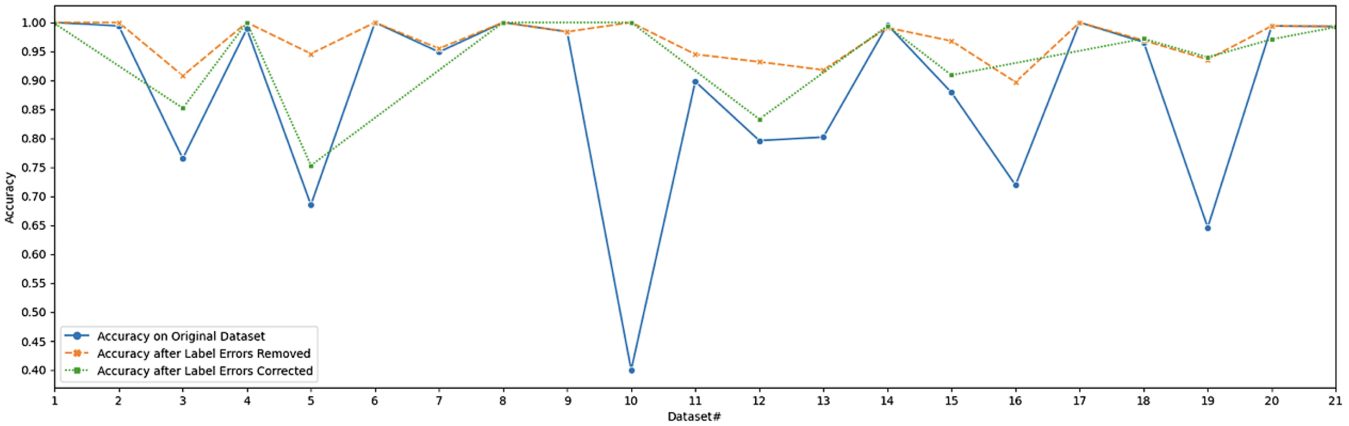

FIGURE 14. Accuracy of Multilayer Perceptron on Original Dataset, Dataset without Label Errors and Dataset with Corrected Label Errors

The robust nature of the Optimized Count-Based Classifier, as demonstrated in this experiment, highlights its potential for practical applications where data quality may be a concern, such as in medical diagnosis, fraud detection, or online content moderation. The classifier's ability to maintain its performance in the face of label errors can be a valuable asset, reducing the burden of extensive data cleaning and quality assurance efforts.

Overall, the robustness analysis using Cleanlab has reinforced the Optimized Count-Based Classifier's resilience to noisy or mislabeled data, positioning it as a viable and reliable choice for classification tasks where data quality may be a concern. This finding, coupled with the classifier's strong computational efficiency and competitive performance, further solidifies its potential for real-world applications in diverse domains.

VI. CONCLUSION

This study successfully optimized the logic and implementation of the Original Count-Based Classifier for categorical data. Through extensive experiments, several notable findings emerged regarding the Optimized Count-Based Classifier's performance and characteristics. The optimization efforts resulted in improved computational efficiency, with faster execution times compared to the original classifier, especially notable in the Mushroom and Nursery datasets. The Optimized Count-Based Classifier demonstrated competitive performance against established classifiers, achieving the highest accuracy on the SPECT Heart Dataset and second-highest on the Animal Condition Classification Dataset, showcasing its potential in health and medicine domains.

Furthermore, the Optimized Count-Based Classifier exhibited robustness against overfitting, exhibiting a low tendency to overfit as cross-validation average accuracies closely aligned with test accuracies for most datasets. Consistent results were observed across multiple iterations with different train-test splits, suggesting reliable and reproducible outcomes. Notably, the classifier demonstrated robustness against label errors, maintaining or modestly improving performance after addressing identified label errors, outperforming more complex models susceptible to noisy data. The Optimized Count-Based Classifier's favorable computational complexity during training and reliance on counting operations contributed to its scalability for handling large datasets efficiently. Moreover, the classifier offered inherent interpretability by quantifying attribute occurrences and their con-

tributions to predictions, a valuable trait for decision-making processes requiring transparency.

While the Optimized Count-Based Classifier demonstrated promising performance, instances of underperformance were observed, particularly in biology and computer science datasets, highlighting areas for potential improvement and optimization of the classifier's logic and implementation. Future work can explore techniques to enhance the classifier's performance in underperforming domains, such as alternative counting strategies, feature engineering, or incorporating domain-specific knowledge. Additionally, extending the classifier's capabilities to handle mixed data types (categorical and numerical) could broaden its applicability. Furthermore, investigating ensemble techniques or hybrid approaches that combine the strengths of the Optimized Count-Based Classifier with other established methods could lead to improved performance and robustness. Parallelization of the implementation could also be explored to further enhance computational efficiency.

This research work serves as a foundation for future investigations, contributing to the advancement of classification techniques for categorical data and their practical applications across diverse domains.

ACKNOWLEDGMENT

The authors express their sincere gratitude to Vellore Institute of Technology, Chennai for providing a conducive environment and continuous support throughout this research endeavor. The authors are also grateful to the administrative staff, fellow researchers and colleagues at VIT for their insightful discussions, collaborative spirit, and constructive feedback, which have undoubtedly enriched and strengthened the outcomes of this research.

REFERENCES

[1] G. Agarwal, C. Goel, C. S. Abhijit, and A. Chauhan, 'Development and Comparative Analysis of an Instance-Based Machine Learning Classifier', SPAST Reports, vol. 1, no. 2, Feb. 2024.

[2] 'Madhyasth Darshan – A direct existential discovery of the universe & human purpose – by a.nagraj'. Accessed: Apr. 12, 2024. [Online]. Available: https://madhyasth-darshan.info/

[3] E. Golinko, T. Sonderman, and X. Zhu, 'CNFL: Categorical to Numerical Feature Learning for Clustering and Classification', in 2017 IEEE Second International Conference on Data Science in Cyberspace (DSC), Shenzhen, China: IEEE, Jun. 2017, pp. 585–594. doi: 10.1109/DSC.2017.87.

[4] S. Zhang, 'Challenges in KNN Classification', IEEE Trans. Knowl. Data Eng., vol. 34, no. 10, pp. 4663–4675, Oct. 2022, doi: 10.1109/TKDE.2021.3049250.

[5] N. A. Priyanka and D. Kumar, 'Decision tree classifier: a detailed survey', IJIDS, vol. 12, no. 3, p. 246, 2020, doi: 10.1504/IJIDS.2020.108141.

[6] I. Wickramasinghe and H. Kalutarage, 'Naive Bayes: applications, variations and vulnerabilities: a review of literature with code snippets for implementation', Soft Comput, vol. 25, no. 3, pp. 2277–2293, Feb. 2021, doi: 10.1007/s00500-020-05297-6.

[7] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, 'A comparative analysis of gradient boosting algorithms', Artif Intell Rev, vol. 54, no. 3, pp. 1937–1967, Mar. 2021, doi: 10.1007/s10462-020-09896-5.

[8] O. Hornyák and L. B. Iantovics, 'AdaBoost Algorithm Could Lead to Weak Results for Data with Certain Characteristics', Mathematics, vol. 11, no. 8, p. 1801, Apr. 2023, doi: 10.3390/math11081801.

[9] F. Rosenblatt, 'The perceptron: A probabilistic model for information storage and organization in the brain.', Psychological Review, vol. 65, no. 6, pp. 386–408, 1958, doi: 10.1037/h0042519.

[10] K.-L. Du, C.-S. Leung, W. H. Mow, and M. N. S. Swamy, 'Perceptron: Learning, Generalization, Model Selection, Fault Tolerance, and Role in the Deep Learning Era', Mathematics, vol. 10, no. 24, p. 4730, Dec. 2022, doi: 10.3390/math10244730.

[11] J. R. Quinlan, 'Induction of decision trees', Mach Learn, vol. 1, no. 1, pp. 81–106, Mar. 1986, doi: 10.1007/BF00116251.

[12] T. Cover and P. Hart, 'Nearest neighbor pattern classification', IEEE Trans. Inform. Theory, vol. 13, no. 1, pp. 21–27, Jan. 1967, doi: 10.1109/TIT.1967.1053964.

[13] C. Cortes and V. Vapnik, 'Support-vector networks', Mach Learn, vol. 20, no. 3, pp. 273–297, Sep. 1995, doi: 10.1007/BF00994018.

[14] D. Meyer, F. Leisch, and K. Hornik, 'The support vector machine under test', Neurocomputing, vol. 55, no. 1–2, pp. 169–186, Sep. 2003, doi: 10.1016/S0925-2312(03)00431-4.

[15] D. A. Pisner and D. M. Schnyer, 'Support vector machine', in Machine Learning, Elsevier, 2020, pp. 101–121. doi: 10.1016/B978-0-12-815739-8.00006-7.

[16] I. Kononenko, 'Comparison of inductive and naive Bayesian learning approaches to automatic knowledge acquisition', Current trends in knowledge acquisition, vol. 8, p. 190, 1990.

[17] J. C. Stoltzfus, 'Logistic Regression: A Brief Primer', Academic Emergency Medicine, vol. 18, no. 10, pp. 1099–1104, Oct. 2011, doi: 10.1111/j.1553-2712.2011.01185.x.

[18] R. D. Joshi and C. K. Dhakal, 'Predicting Type 2 Diabetes Using Logistic Regression and Machine Learning Approaches', IJERPH, vol. 18, no. 14, p. 7346, Jul. 2021, doi: 10.3390/ijerph18147346.

[19] F. Murtagh, 'Multilayer perceptrons for classification and regression', Neurocomputing, vol. 2, no. 5–6, pp. 183–197, Jul. 1991, doi: 10.1016/0925-2312(91)90023-5.

[20] A. Juna et al., 'Water Quality Prediction Using KNN Imputer and Multilayer Perceptron', Water, vol. 14, no. 17, p. 2592, Aug. 2022, doi: 10.3390/w14172592.

[21] R. E. Schapire, 'A brief introduction to boosting', in Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2, in IJCAI'99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 1401–1406.

[22] L. Breiman, 'Random Forests', Machine Learning, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[23] D. A. Fife and J. D'Onofrio, 'Common, uncommon, and novel applications of random forest in psychological research', Behav Res, vol. 55, no. 5, pp. 2447–2466, Aug. 2022, doi: 10.3758/s13428-022-01901-9.

[24] J. H. Friedman, 'Stochastic gradient boosting', Computational Statistics & Data Analysis, vol. 38, no. 4, pp. 367–378, Feb. 2002, doi: 10.1016/S0167-9473(01)00065-2.

[25] C. Bowd et al., 'Gradient-Boosting Classifiers Combining Vessel Density and Tissue Thickness Measurements for Classifying Early to Moderate Glaucoma', American Journal of Ophthalmology, vol. 217, pp. 131–139, Sep. 2020, doi: 10.1016/j.ajo.2020.03.024.

[26] P. Geurts, D. Ernst, and L. Wehenkel, 'Extremely randomized trees', Mach Learn, vol. 63, no. 1, pp. 3–42, Apr. 2006, doi: 10.1007/s10994-006-6226-1.

[27] A. Pagliaro, 'Forecasting Significant Stock Market Price Changes Using Machine Learning: Extra Trees Classifier Leads', Electronics, vol. 12, no. 21, p. 4551, Nov. 2023, doi: 10.3390/electronics12214551.

[28] T. Chen and C. Guestrin, 'XGBoost: A Scalable Tree Boosting System', in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco California USA: ACM, Aug. 2016, pp. 785–794. doi: 10.1145/2939672.2939785.

[29] A. Asselman, M. Khaldi, and S. Aammou, 'Enhancing the prediction of student performance based on the machine learning XGBoost algorithm', Interactive Learning Environments, vol. 31, no. 6, pp. 3360–3379, Aug. 2023, doi: 10.1080/10494820.2021.1928235.

[30] G. Ke et al., 'LightGBM: a highly efficient gradient boosting decision tree', in Proceedings of the 31st International Conference on Neural Information Processing Systems, in NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 3149–3157.

[31] D. D. Rufo, T. G. Debelee, A. Ibenthal, and W. G. Negera, 'Diagnosis of Diabetes Mellitus Using Gradient Boosting Machine (LightGBM)', Diagnostics, vol. 11, no. 9, p. 1714, Sep. 2021, doi: 10.3390/diagnostics11091714.

[32] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, 'CatBoost: unbiased boosting with categorical features', in Proceedings of the 32nd International Conference on Neural Information Processing

Systems, in NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, pp. 6639–6649.
[33] S. Ö. Arik and T. Pfister, 'TabNet: Attentive Interpretable Tabular Learning', AAAI, vol. 35, no. 8, pp. 6679–6687, May 2021, doi: 10.1609/aaai.v35i8.16826.
[34] R. Sharma et al., 'A Novel Framework for Unification of Association Rule Mining, Online Analytical Processing and Statistical Reasoning', IEEE Access, pp. 1–1, 2022, doi: 10.1109/ACCESS.2022.3142537.
[35] R. Sharma, 'On statistical paradoxes and overcoming the impact of bias in expert systems: towards fair and trustworthy decision making', SSRN Journal, 2023, doi: 10.2139/ssrn.4506432.
[36] M. Kaushik, R. Sharma, S. A. Peious, M. Shahin, S. Ben Yahia, and D. Draheim, 'On the Potential of Numerical Association Rule Mining', in Future Data and Security Engineering. Big Data, Security and Privacy, Smart City and Industry 4.0 Applications, vol. 1306, T. K. Dang, J. Küng, M. Takizawa, and T. M. Chung, Eds., in Communications in Computer and Information Science, vol. 1306. , Singapore: Springer Singapore, 2020, pp. 3–20. doi: 10.1007/978-981-33-4370-2_1.
[37] M. Kaushik, R. Sharma, I. Fister Jr., and D. Draheim, 'Numerical Association Rule Mining: A Systematic Literature Review'. arXiv, Jul. 02, 2023. Accessed: Jun. 29, 2024. [Online]. Available: http://arxiv.org/abs/2307.00662
[38] M. Kaushik, R. Sharma, A. Vidyarthi, and D. Draheim, 'Discretizing Numerical Attributes: An Analysis of Human Perceptions', in New Trends in Database and Information Systems, vol. 1652, S. Chiusano, T. Cerquitelli, R. Wrembel, K. Nørvåg, B. Catania, G. Vargas-Solar, and E. Zumpano, Eds., in Communications in Computer and Information Science, vol. 1652. , Cham: Springer International Publishing, 2022, pp. 188–197. doi: 10.1007/978-3-031-15743-1_18.
[39] S. Jukna, 'The Pigeonhole Principle', in Extremal Combinatorics, in Texts in Theoretical Computer Science. An EATCS Series. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 53–75. doi: 10.1007/978-3-642-17364-6_4.
[40] 'CountEst: CountEst is a module containing the implementation of count-based estimators, i.e., supervised learning algorithms that make predictions based on the frequency or count of specific events, categories, or values within a dataset.' Accessed: Apr. 20, 2024. [MacOS, Microsoft:: Windows, POSIX, Unix]. Available: https://pypi.org/project/CountEst/
[41] 'Countest:: Anaconda.org'. Accessed: Apr. 20, 2024. [Online]. Available: https://anaconda.org/conda-forge/countest

SANSKRITI SANJAY KUMAR SINGH is a fourth year student currently pursuing B.Tech degree in computer science and engineering from Vellore Institute of Technology, Chennai, set to graduate in 2024.

Passionate about machine learning, her interests lie in supervised machine learning, deep learning, transfer learning, natural language processing, and causal inference. She has presented her research works at various prestigious conferences, with publications featured in renowned Springer and IEEE indexed chapters.

ALOK CHAUHAN received the M. Sc. degree in Physics from IIT Kanpur in 1985. He has completed the M.Tech degree in computer science from IIT Roorkee in 1988 and the Ph.D. degree from VIT Chennai, in 2020.

He has more than thirty-five years of experience in academics, industry and research. He is currently Associate Professor with the School of Computer Science and Engineering, VIT Chennai, India. His research interests include algorithm engineering and ontological engineering.