International Journal of Innovative Information Systems & Technology Research 14(3):519-532, July-Sept, 2026

© SEAHI PUBLICATIONS, 2026 www.seahipublications.org ISSN: 2467-8562

DOI:10.5281/zenodo.23174480

A Hybridized Machine-Learning Model For Predicting Software Piracy Using Social And Behavioural Characteristics

Timinipre Amabie*; Laeticia Nneka Onyejegbu; Barilee Baridam

Department of Computer Science,
Faculty of Computing,
University of Port Harcourt, Port Harcourt, Rivers State, Nigeria.
***Corresponding author: amabietimi@gmail.com**

ABSTRACT
This study developed a hybridized machine-learning framework for classifying self-reported software piracy using social, demographic and behavioural characteristics. The dataset comprised 553 respondents and 24 questionnaire items from a SurveyPlanet survey on the pricing and consumption of music-production software. Preprocessing included missing-value treatment, geographical grouping, binary target construction and one-hot encoding. Six candidate feature groups yielded 35 binary features. All 63 non-empty combinations were evaluated across Decision Tree depths of 1–9 using five-fold stratified cross-validation, producing 567 configurations. Expertise Level, Age Group, Region, and Importance of Ease of Pirating formed the selected feature combination, and the optimized standalone Decision Tree had a maximum depth of five. The framework also incorporated AdaBoost with depth-one Decision Trees as weak learners. On the held-out test set of 111 observations, the Decision Tree achieved 73.87% accuracy, compared with 72.97% for AdaBoost. For the Pirate class, Decision Tree precision, recall and F1-score were 67.69%, 84.62% and 75.21%, respectively; the corresponding AdaBoost values were 67.19%, 82.69% and 74.14%. The trained classifier was integrated into a Streamlit application. The findings show that the selected characteristics supported classification of self-reported piracy within this dataset, although boosting did not improve accuracy over the optimized standalone tree. The resulting framework provides a practical approach to examining behavioural patterns associated with software piracy.
Keywords: software piracy; machine learning; Decision Tree; AdaBoost; behavioural prediction; classification.

1. INTRODUCTION

Software piracy involves the unauthorized copying, distribution or use of copyrighted software and has been examined through economic, social and behavioural perspectives. Reviews of empirical research show that the field draws on several disciplines and investigates a range of individual and contextual factors (Chavarria et al., 2016; Gomes et al., 2015). Understanding these factors provides a basis for examining patterns associated with software users and their reported piracy behaviour.

Behavioural studies provide a theoretical foundation for this approach. Peace et al. (2003) examined software piracy in the workplace using concepts from the theory of planned behaviour, expected utility theory and deterrence theory. Their model connected piracy intentions with attitudes, subjective norms and perceived behavioural control, while also considering software cost and perceptions of punishment. Al-Rafee and Cronan (2006) investigated factors influencing attitudes towards digital piracy, including moral judgement and social influences. These studies establish the relevance of individual perceptions and social circumstances to the study of piracy.

Lowry et al. (2017) synthesized research on digital piracy using social cognitive theory, bringing together behavioural and environmental factors within a common framework. Mishra et al. (2006) examined software piracy among information technology professionals in organizations. These contributions support the investigation of user characteristics, while the broader reviews by Gomes et al. (2015) and Chavarria et al. (2016) place such investigations within the wider empirical literature. Predictive associations, however, do not by themselves establish that a characteristic causes piracy.

Recent studies extend this behavioural perspective. Sikolia et al. (2025) associated neutralization techniques and dimensions of religiosity with software-piracy intentions in a student sample. Mallick et al. (2024) examined peer influence and awareness through mathematical models of piracy dynamics on networks. These approaches address different outcomes and levels of analysis, but both reinforce the relevance of social context. The present study instead examines whether a defined set of respondent characteristics can classify self-reported software possession.

The present study examines whether survey-derived characteristics can be used together to classify respondents according to their self-reported possession of pirated software. This extends the behavioural perspective into an operational classification task. The target distinguishes respondents reporting a non-zero percentage of pirated software from those reporting none. Accordingly, a prediction represents an estimated survey class; it is not evidence that an individual has infringed copyright.

A Decision Tree classifies observations by partitioning the data according to predictor values. AdaBoost combines a sequence of weak learners, giving greater emphasis during training to observations that preceding learners misclassified. In this study, hybridized refers to the integration of Decision Tree classification and AdaBoost within a common prediction framework. The optimized standalone tree provides the comparison model, while AdaBoost combines multiple shallow trees. The term does not imply that the two final model outputs were combined into a separate classifier.

Tree-based learning is a relevant starting point for structured survey data. Grinsztajn et al. (2022) found that tree-based models remained competitive with neural networks across a benchmark of tabular datasets. Their findings support empirical comparison of candidate models. Reliable comparisons also require separation of model development from testing: Kapoor and Narayanan (2023) documented how data leakage can produce overly optimistic scientific conclusions. In addition, evaluation should examine the errors behind an overall performance score; Chicco and Jurman (2023) emphasized the information provided by sensitivity, specificity and predictive values in binary classification.

The framework evaluates combinations of social, demographic and behavioural characteristics, compares the resulting classifiers and implements the trained model in an interactive application. Its focus is classification from user characteristics. This provides a practical way to assess whether the selected survey variables contain information useful for predicting self-reported piracy.

The aim of this study was to develop a hybridized model for predicting software piracy. The specific objectives were to: (i) design a prediction framework using Decision Tree classification and AdaBoost; (ii) train and test the models using a software-piracy behavioural dataset; (iii) implement the system in Python; and (iv) evaluate classification performance using accuracy, precision, recall, F1-score and confusion matrices.

2. MATERIALS AND METHODS

2.1 Research and System Development Approach

The study adopted a quantitative supervised machine-learning approach for developing the software-piracy prediction model, while Object-Oriented Analysis and Design (OOAD) was used for the analysis and design of the software system. The machine-learning process involved data preprocessing, feature selection, model training, testing and performance evaluation. The Decision Tree and AdaBoost algorithms were used to classify respondents as Pirate or Non-Pirate based on selected social, demographic and behavioural characteristics.

Object-Oriented Analysis and Design was used to describe the structure, functions and interactions of the software system through Unified Modeling Language use-case, class, activity and sequence diagrams.

2.2 Methodology

Object-Oriented Analysis and Design (OOAD) was adopted for the development of the system. The approach organized the analysis and design around the objects, attributes, operations and relationships required by the software-piracy prediction application. The object-oriented development process adopted in this study comprised Object-Oriented Analysis (OOA), Object-Oriented Design (OOD), and Object-Oriented Programming (OOP) as the implementation approach.

1. Object-Oriented Analysis (OOA): This phase focused on analyzing the problem domain and identifying the objects, concepts, processes and requirements relevant to the software-piracy prediction system. It established an understanding of the system requirements and identified the major entities involved in the prediction process, including user characteristics, piracy-related factors, datasets, prediction models and prediction results.

2. Object-Oriented Design (OOD): This phase transformed the requirements identified during analysis into a detailed software design. It involved defining the classes, attributes, methods, relationships and interactions required to implement the system. The design specified how the major system components, including data preprocessing, feature selection, machine-learning model training, model evaluation, prediction and result presentation, interact to fulfil the identified system requirements.

3. Object-Oriented Programming (OOP): OOP represented the implementation approach used to translate the object-oriented design into executable software. The classes and methods defined during the design stage were implemented using a programming language, while objects were instantiated to perform specific operations. OOP provided the implementation foundation for integrating data preprocessing, feature selection, machine-learning algorithms, prediction procedures and system outputs into a functional application.

2.3 System Architecture

Figure 1 illustrates the major processes involved in the software piracy prediction system. The process begins with the dataset, which is divided into 80% training data and 20% test data. The training data are used for model development, validation and optimization, while the test data are reserved for final performance evaluation.

The training data are passed to the Software Piracy Data Analysis Unit, where they are prepared for machine-learning analysis. Data visualization is initially used to examine patterns, distributions and differences among the features. The data are subsequently prepared for modelling through appropriate preprocessing procedures, followed by feature selection.

The selected features are then used to develop and optimize the standalone Decision Tree classifier. Maximum tree depths ranging from 1 to 9 are evaluated using five-fold stratified cross-validation, with a maximum depth of 5 selected for the optimized standalone Decision Tree.

The AdaBoost ensemble is implemented separately using depth-1 Decision Trees, commonly referred to as decision stumps, as weak learners. During successive boosting iterations, greater emphasis is placed on observations misclassified by preceding learners. The implementation uses 50 estimators, a learning rate

of 1.0, and a random state of 42. The predictions of the weak learners are subsequently combined through weighted voting to produce the final ensemble classification.

Following model training, the 20% test dataset, which is not used for model fitting, validation or optimization, is supplied to the trained models for final evaluation. Classification performance is assessed using confusion matrices, accuracy, precision, recall and F1-score.

Overall, the system architecture represents the flow from dataset preparation and analysis through feature selection and model development to final classification and performance evaluation.

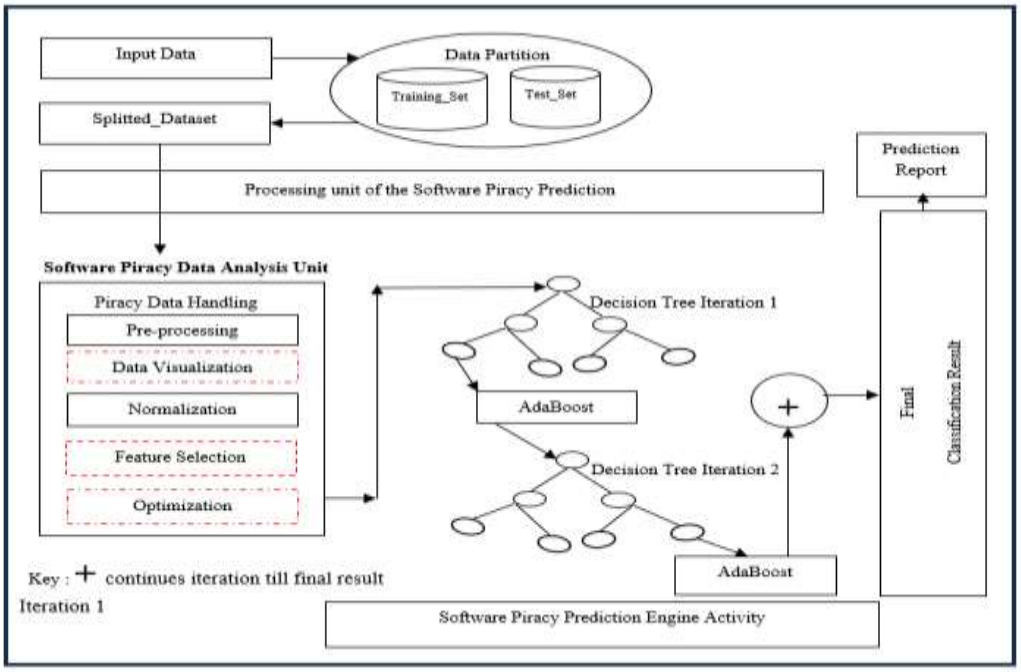

Figure 1. Architecture of the software-piracy prediction system. Source: Authors' system design.

2.4 Unified Modeling Language Design of the System

Four Unified Modeling Language (UML) diagrams were used to represent the design of the Software Piracy Prediction System: Use-Case Diagram, Class Diagram, Activity Diagram and Sequence Diagram. The Use-Case Diagram represents the major functions of the system and the interaction between the user and the system. The Class Diagram represents the structural components of the system and their relationships. The Activity Diagram illustrates the flow of activities involved in the prediction process, while the Sequence Diagram represents the order of interactions among system components during software-piracy prediction.

2.4.1 Use Case Diagram

Figure 2 presents the Use-Case Diagram of the system, which illustrates the principal functions of the software piracy prediction system and the interaction between the User and the system.

The interaction begins when the User selects Enter User Characteristics and provides the information required for software piracy prediction. The characteristics supplied include Expertise Level, Age Group, Region, and Ease of Pirating. These characteristics constitute the selected input features used by the prediction model.

The system proceeds to Validate Input Data to ensure that the required information has been correctly supplied. After validation, the data are preprocessed and encoded into the appropriate format required by the machine-learning model. This stage is necessary because the input characteristics are categorical and

must be transformed into a suitable numerical representation before they can be processed by the trained model.

The processed data are subsequently passed to the Predict Software Piracy use case. At this stage, the trained machine-learning model applies the patterns and relationships learned during model development to generate a software piracy prediction based on the characteristics supplied by the User.

After the prediction has been generated, the system proceeds to Format and Display Result. This converts the prediction output into an understandable form and presents it through the system interface. The View Prediction Result use case represents the User's interaction with the displayed prediction output.

The Exit System use case allows the User to terminate interaction with the application after completing the prediction process.

Overall, the Use-Case Diagram provides a functional representation of the system by showing the major activities involved in software piracy prediction. It demonstrates the progression from the entry of user characteristics through input validation, preprocessing and encoding, prediction, and presentation of the result, while identifying the User as the principal external actor interacting with the system.

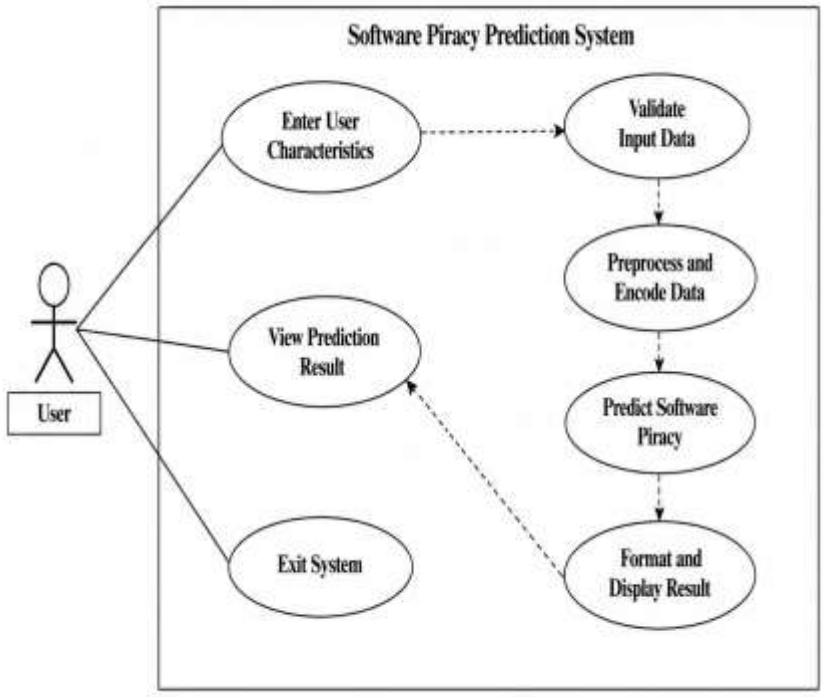

Figure 2. UML use-case diagram of the software-piracy prediction system. Source: Authors' system design.

2.4.2 Class Diagram

Figure 3 presents the class diagram of the system. It shows the major classes, their attributes, operations, and relationships. The major classes are User, InputData, PredictionModel, DecisionTreeModel, AdaBoostModel, PredictionResult, Dataset, and ModelEvaluation.

The **User** class represents the individual who interacts with the system. It contains information relating to the characteristics required for prediction, including Expertise Level, Age Group, Region, and Ease of Pirating. Its operations enable the User to enter the required characteristics, request a prediction, and view the prediction result.

The **InputData** class handles the data supplied for prediction. It is responsible for validating the input, preprocessing the data, and encoding the categorical characteristics into a suitable format for the machine-learning model. The processed data are subsequently made available to the prediction component of the system.

The **PredictionModel** class represents the machine-learning component responsible for software piracy prediction. Its operations include training the model, making predictions, saving and loading the trained model, and evaluating its performance. Once a trained model has been saved, it can be loaded and applied to new input data without retraining the model whenever a prediction is requested.

The **DecisionTreeModel** and **AdaBoostModel** classes represent the two machine-learning algorithms implemented and compared in the study. The Decision Tree represents the standalone classification model, while the AdaBoost model represents the ensemble-learning component in which depth-1 Decision Trees are used as weak learners. These classes contain algorithm-specific attributes and operations required for model training and prediction.

The **Dataset** class represents the dataset used in developing and evaluating the machine-learning models. Its operations include loading, cleaning, preprocessing, and splitting the dataset for model development and evaluation. The class therefore supports the preparation of the data required for training and testing the models.

The **ModelEvaluation** class represents the performance-evaluation component of the system. It contains the evaluation measures used in the study, including accuracy, precision, recall, F1-score, and confusion matrix. These measures are used to assess and compare the classification performance of the trained models and provide information about their predictive effectiveness.

The **PredictionResult** class represents the output produced by the system after processing the supplied characteristics. It stores information relating to the prediction and provides operations for displaying the result. The generated result is subsequently presented to the User through the system interface.

The relationships among the classes illustrate how the different components of the system interact. The User supplies the required characteristics through the InputData component. The supplied data are validated, preprocessed, and encoded before being passed to the prediction component. The PredictionModel uses the trained machine-learning model to generate a PredictionResult, which is subsequently presented to the User. During model development, the Dataset provides the data required for training and testing, while the ModelEvaluation component assesses the performance of the Decision Tree and AdaBoost models using the selected evaluation measures.

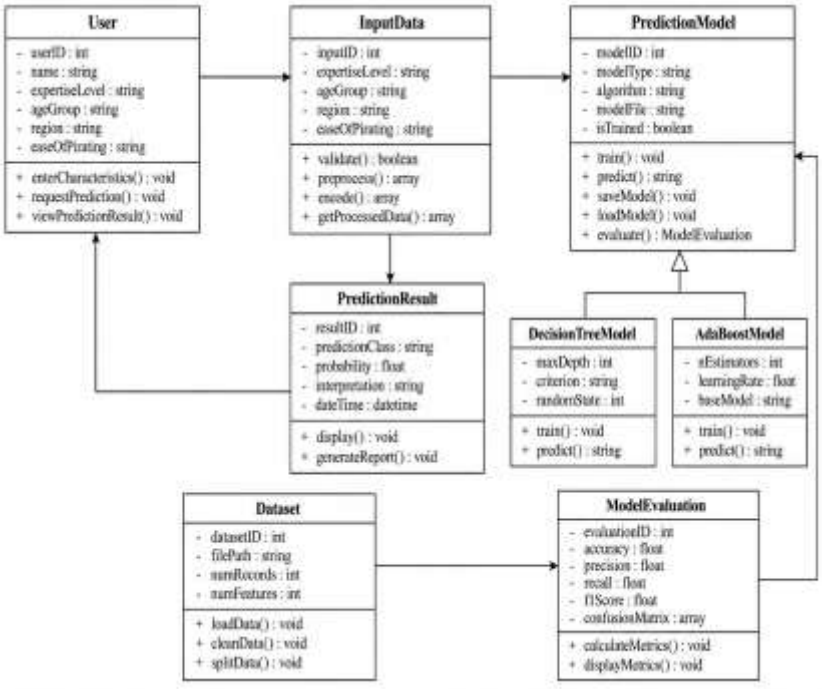

Figure 3. UML class diagram of the software-piracy prediction system. Source: Authors' system design.

2.4.3 Activity Diagram

Figure 4 presents the Activity Diagram of the system, which illustrates the workflow of the software piracy prediction system. It distinguishes the activities performed by the User from those performed by the system.

The process begins when the **User** launches the application and enters the required characteristics. These characteristics include Expertise Level, Age Group, Region, and Ease of Pirating. The entered characteristics are received by the system and validated to ensure that the required input data have been correctly provided.

A decision is made based on the validity of the input data. If the input is invalid, the **User** is required to correct the input data and resubmit the characteristics. This process continues until valid input data are provided. If the input is valid, the system proceeds to preprocess and encode the input data into the appropriate format required by the trained machine-learning model.

After preprocessing and encoding, the system loads the saved trained model and applies it to the processed input data to generate a software piracy prediction. The system then formats the prediction output and displays the result through the application interface.

The **User** subsequently views the prediction result, after which the prediction process terminates. The Activity Diagram therefore demonstrates the flow of control from the initial interaction of the User with the application, through input validation, preprocessing and encoding, to the application of the trained model and the presentation of the final prediction result.

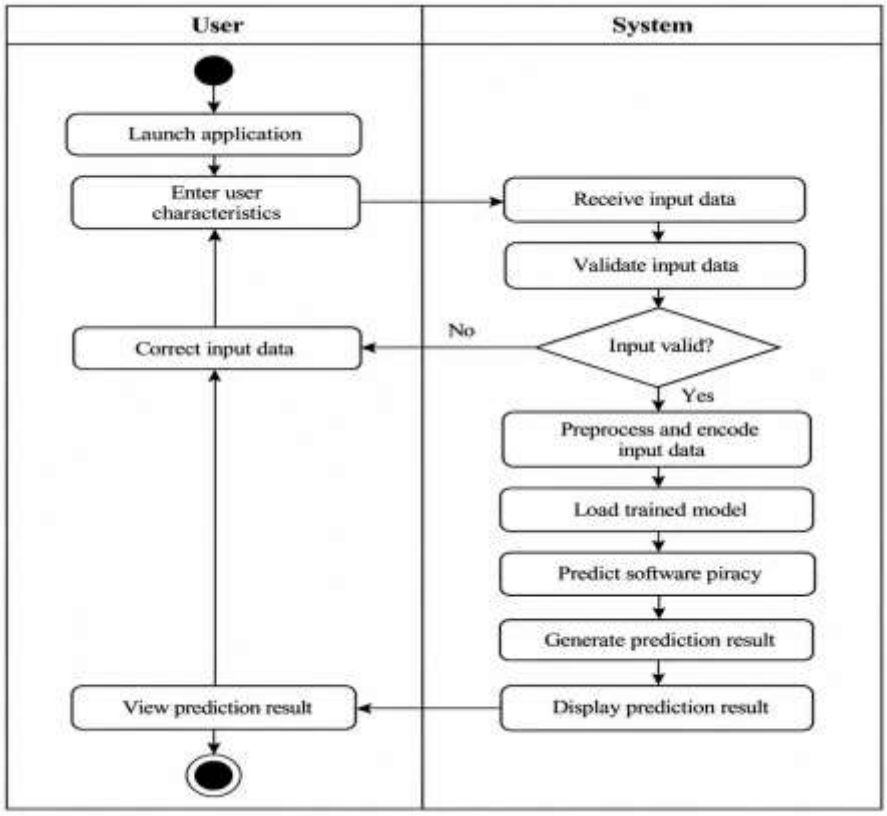

Figure 4. UML activity diagram of the software-piracy prediction system. Source: Authors' system design.

2.4.4 Sequence Diagram

Figure 5 presents the Sequence Diagram of the system. It shows the interactions among the User, User Interface, Input Data Handler, Prediction Model, and Prediction Result components. The vertical lifelines represent the participants involved in the interaction, while the horizontal arrows represent the messages exchanged among them.

The interaction begins when the User enters the required characteristics through the User Interface. The characteristics supplied include Expertise Level, Age Group, Region, and Ease of Pirating. The User Interface receives the characteristics and sends the input data to the Input Data Handler for validation.

The Input Data Handler validates the supplied information to determine whether the required data are valid. Where the input data are invalid, the validation status is returned to the User Interface and the User is prompted to correct the input. The corrected data are then resubmitted. This alternative interaction ensures that invalid or incomplete data do not proceed to the prediction stage.

When the input data are valid, the Input Data Handler preprocesses and encodes the data into the appropriate format required by the trained machine-learning model. The processed data are subsequently passed to the Prediction Model, which applies the trained model to generate the software piracy prediction. The system was developed using Decision Tree and AdaBoost machine-learning algorithms, which were separately implemented and evaluated during model development.

After the prediction has been performed, the Prediction Model returns the prediction output to the Input Data Handler. The resulting prediction is then transferred to the Prediction Result component, where it is formatted into a suitable form for presentation.

The formatted result is returned to the User Interface, which displays the prediction result to the User. The User then views the result, thereby completing the interaction sequence.

The Sequence Diagram demonstrates the chronological flow of information through the system, beginning with the submission of user characteristics and proceeding through input validation, preprocessing and encoding, prediction, result generation, and presentation. It also illustrates how the different components communicate with one another to accomplish the system's primary objective of predicting the likelihood of software piracy based on the selected characteristics.

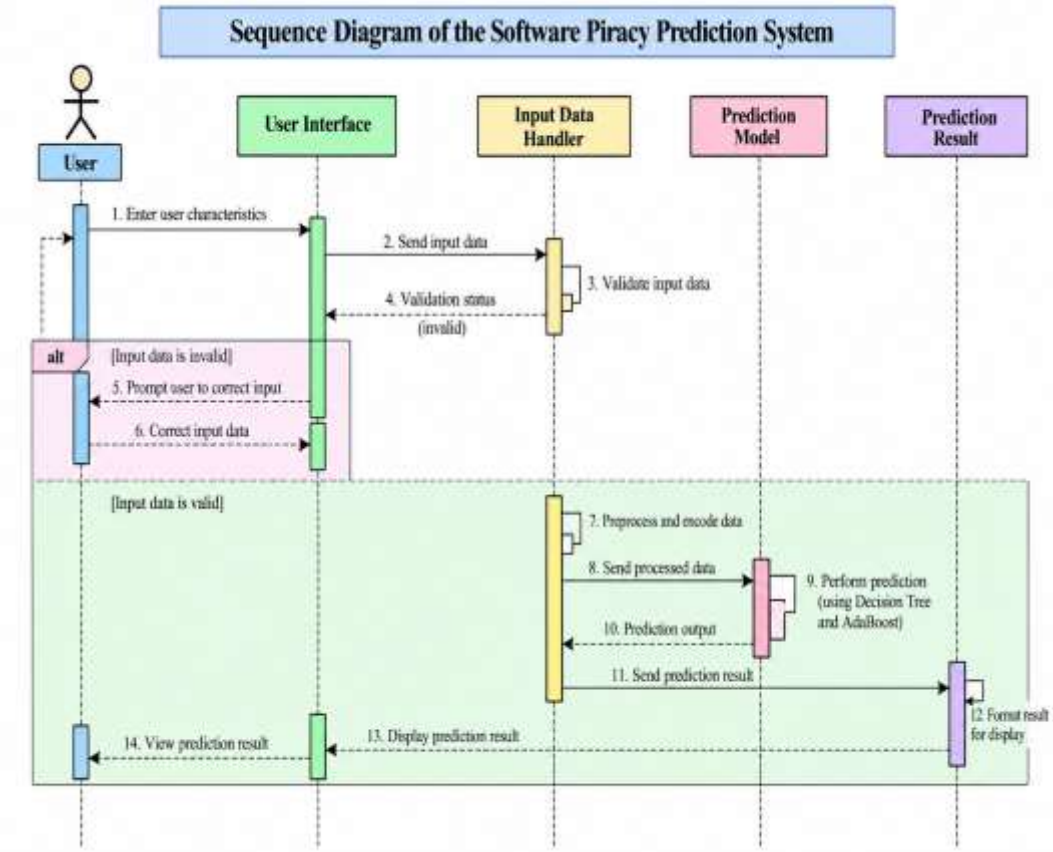

Figure 5. UML sequence diagram of the software-piracy prediction system. Source: Authors' system design.

2.2 Dataset

The dataset originated from a SurveyPlanet survey on the pricing and consumption of music-production software and was exported in comma-separated values format. It comprised 553 respondents and 24 questionnaire items, with each row corresponding to an individual participant.

The dataset included categorical demographic characteristics, information concerning software-consumption behaviour, continuous numerical variables representing perceptions and sentiments, and free-form textual responses. It therefore provided a combination of demographic, behavioural, economic, pricing and software-use information for investigating characteristics associated with software piracy.

2.3 Data Preprocessing and Target Construction

The raw dataset was cleaned, transformed and organized before machine-learning analysis. Missing values were handled through listwise deletion, while categorical variables were transformed into numerical representations suitable for machine-learning algorithms.

The Percentage of Pirated Software variable was converted into a binary target. Respondents reporting a non-zero percentage were assigned to the Pirate class, and those reporting zero were assigned to the Non-Pirate class.

Two variables, Can Afford Everything and Ease of Pirating is Important, were derived from the survey responses. The latter represents the importance respondents attached to ease of pirating and is referred to as Importance of Ease of Pirating in the analysis. Reported countries were grouped into broader geographical regions to reduce the number of categories.

Categorical variables were subsequently transformed using one-hot encoding.

2.4 Exploratory Data Analysis

Exploratory Data Analysis was undertaken to examine distributions and relationships within the dataset before model development. Histograms were used to examine the frequency distributions of variables, while scatter plots were used to explore relationships between selected variables and Percentage of Pirated Software.

The visual analysis provided descriptive information about patterns and differences among respondents and assisted in identifying characteristics that warranted further investigation. Visual patterns were not treated as sufficient evidence of predictive importance; candidate features were subsequently evaluated through machine-learning procedures.

2.5 Candidate Feature Groups

Initial feature selection was informed by the research question, domain relevance and visual differences observed between Pirate and Non-Pirate respondents. Six candidate feature groups were retained.

The groups were Expertise Level (F1), Goal (F2), Age Group (F3), Region (F4), Importance of Ease of Pirating (F5), and Ability to Afford Everything (F6). The last two correspond to the derived variables Ease of Pirating is Important and Can Afford Everything, respectively.

Following one-hot encoding and binarization, these six feature groups produced **35 binary features**.

2.6 Feature-Combination and Decision Tree Optimization

All possible non-empty combinations of the six candidate feature groups were evaluated. With six feature groups, the total number of non-empty combinations was:

$2^6 - 1 = 63$ feature combinations.

Decision Tree maximum depths ranging from 1 to 9 were evaluated for each feature-group combination. Consequently, the optimization procedure examined:

$63 \times 9 = 567$ Decision Tree configurations.

Five-fold stratified cross-validation was applied during optimization. Mean validation accuracy was calculated for each configuration, and the combination producing the highest mean validation accuracy was identified.

The final standalone Decision Tree was therefore configured with a maximum depth of five and a random state of 100.

2.7 Training and Testing Procedure

The dataset was divided into 80% training data and 20% testing data. The training data were used for model development and optimization, while the testing data were retained separately for final performance evaluation. The split used random_state = 100 and shuffle = True, retaining a consistent partition during evaluation of the feature combinations.

Five-fold stratified cross-validation was applied during optimization so that Pirate and Non-Pirate respondents were appropriately represented across the validation folds. Following model selection, the selected configuration was trained using the training subset and subsequently evaluated using the held-out testing subset.

The final evaluation dataset consisted of 111 observations.

2.8 AdaBoost Model

AdaBoost was implemented as the ensemble component of the predictive framework. Unlike the optimized standalone Decision Tree, which used a maximum depth of five, AdaBoost employed shallow Decision Trees with a maximum depth of one as weak learners.

The AdaBoost configuration consisted of a Decision Tree base estimator with maximum depth = 1, 50 estimators, a learning rate of 1.0, and random state = 42.

During training, weak learners were constructed sequentially, with greater emphasis placed on observations incorrectly classified by preceding learners. Their predictions were subsequently combined through weighted voting to produce the final ensemble classification.

This configuration enabled comparison between a single optimized depth-five Decision Tree and an ensemble consisting of multiple shallow Decision Trees.

2.9 Performance Evaluation

Model performance was evaluated using accuracy, precision, recall, F1-score and confusion matrices.

The Pirate class was treated as the positive class in the confusion matrices. Accuracy measured the proportion of all observations classified correctly. For each class, precision measured the proportion of predictions assigned to that class that were correct, and recall measured the proportion of observations in that class identified correctly. The F1-score was the harmonic mean of precision and recall.

These measures were considered collectively because overall accuracy alone does not reveal differences in classification performance between the Pirate and Non-Pirate classes.

2.10 System Implementation

The Software Piracy Prediction System was implemented using the Python programming language. Jupyter Notebook supported data preprocessing, exploratory analysis, feature selection, model experimentation, training and performance evaluation, while Visual Studio was used as an integrated development environment for source-code development and debugging.

Scikit-learn provided the machine-learning functionality required for Decision Tree classification, AdaBoost, train-test splitting, cross-validation and performance evaluation. Pandas and NumPy supported data processing and numerical operations, while Matplotlib and Seaborn supported data visualization.

Joblib was used for model persistence, while Streamlit was used to develop the interactive user interface. The trained model was serialized for reuse within the Streamlit application, enabling predictions to be generated without retraining the model whenever the application was executed.

The application accepts four inputs: Expertise Level, Age Group, Region, and Importance of Ease of Pirating. The saved model processes the inputs and returns a Pirate or Non-Pirate classification.

3. RESULTS

3.1 Feature Selection and Decision Tree Optimization

The search across 63 feature-group combinations and nine maximum tree depths selected Expertise Level, Age Group, Region, and Importance of Ease of Pirating. This combination achieved the highest mean validation accuracy among those evaluated. The final model therefore used four of the six candidate groups, excluding Goal and Ability to Afford Everything from the selected subset.

Selection operated at the feature-group level: the four retained groups contributed their encoded categories. The search consequently identified a combination of respondent characteristics for joint classification, not a ranking of individual encoded predictors.

A maximum depth of five was selected from the tested range of one to nine. Feature-group selection and depth optimization jointly determined the standalone Decision Tree evaluated on the held-out test set. The resulting test performance is presented in Table 1.

The Pirate-class F1-score difference was 1.07 percentage points, and the Non-Pirate F1-score difference was 0.68 percentage points, both favouring the Decision Tree. Its higher accuracy was therefore accompanied by small improvements in the combined precision–recall measure for each class.

Under the tested feature combination and model settings, AdaBoost did not improve classification performance over the optimized standalone tree. This result describes the two fitted configurations on the held-out dataset; the broader interpretation of the difference is considered below.

4. DISCUSSION

4.1 Predictive Value of User Characteristics

The results show that the selected respondent characteristics supported classification of self-reported software piracy, with both models achieving approximately 73% test accuracy. The substantive finding is the predictive value of the combined feature groups within the music-production software survey. This links the behavioural focus of the study to a measurable classification outcome without treating the model as a causal explanation of piracy.

This interpretation is consistent with the attention given to individual perceptions and social circumstances in earlier piracy research (Al-Rafee & Cronan, 2006; Lowry et al., 2017; Peace et al., 2003). The present study evaluates the combined predictive value of selected survey-derived characteristics. Recent evidence also distinguishes explanatory modelling from prediction. Koay et al. (2024) compared seven theoretical models of digital-piracy intentions and assessed both in-sample and out-of-sample predictive performance. Their strongest model drew on the theory of interpersonal behaviour. This supports evaluating behavioural predictors jointly, but their intention outcome and structural equation approach differ from the present classification of self-reported software possession.

Sikolia et al. (2025) provide a further comparison: their analysis associated particular rationalizations and religiosity dimensions with software-piracy intentions. Those constructs were not directly represented by the present four-input model. The two studies therefore address complementary questions—factors associated with intentions and classification of reported possession. Their results should not be combined as if they measured the same behaviour or evaluated the same prediction task.

4.2 Interpretation of the Selected Features

The selected combination brings together expertise, age, geographical region and the importance attached to ease of pirating. Importance of Ease of Pirating represents a respondent's assessment of that consideration; it does not directly measure how easy piracy is. Selection of the four groups reflects their joint performance in the feature search and does not establish an individual importance ranking.

These characteristics should be interpreted as predictors within the fitted models. Membership of a particular age group or region does not establish that a respondent possesses pirated software, and the selected combination does not establish causal effects.

Lowry et al. (2017) identified outcome expectancies, social learning, self-efficacy and self-regulation, and moral disengagement as relevant groups of factors in digital-piracy research. The present inputs represent a narrower operational selection and should not be treated as direct measures of all these constructs. Similarly, exclusion of affordability from the selected subset does not contradict the relevance of cost in Peace et al. (2003); it concerns predictive performance within a particular combination of survey variables.

Regional membership should likewise be interpreted as contextual information, not a direct measure of peer influence. Mallick et al. (2024) explicitly represented social interactions in their network models, whereas the present survey classifier did not model relationships between respondents. That distinction matters: retaining Region does not demonstrate the social-transmission mechanisms examined in a network analysis. It identifies a useful component of the selected predictor set under the present evaluation procedure.

4.3 Comparison of the Classifiers

The optimized Decision Tree exceeded AdaBoost accuracy by 0.90 percentage points (Table 3). Boosting therefore provided no accuracy improvement under the tested configurations. This comparison concerns a depth-five standalone tree and an ensemble of depth-one trees; it does not establish that either algorithm is generally superior.

The empirical advantage of the standalone tree is compatible with the broader case for testing model complexity. Grinsztajn et al. (2022) demonstrated strong tree-based performance on tabular benchmarks, but their comparison with neural networks does not establish a general advantage of single trees over AdaBoost. Here, the depth-five tree and the ensemble of depth-one learners represent different fitted configurations. The observed difference cannot isolate which modelling choice produced the result.

Both classifiers had higher recall for the Pirate class than for the Non-Pirate class, while precision was higher for the Non-Pirate class (Tables 1 and 2). This pattern shows that identifying a large proportion of Pirate-labelled respondents also involved assigning some Non-Pirate respondents to that class. Accuracy alone would not communicate this difference in classification performance.

Rainio et al. (2024) explain that precision and recall describe different aspects of classification and that model comparison requires appropriate statistical evaluation. Here, Pirate recall indicates coverage of respondents reporting piracy, whereas Pirate precision indicates the reliability of that predicted label. Reading both measures alongside the F1-scores clarifies the practical meaning of the results. The observed accuracy difference is a descriptive comparison, not evidence of statistically established superiority.

Chicco and Jurman (2023) stress the importance of examining both outcome classes and predictive values when judging binary classifiers. In the present results, Non-Pirate recall corresponds to specificity when Pirate is the positive class, and Non-Pirate precision corresponds to the negative predictive value. This interpretation makes the asymmetry in Tables 1 and 2 explicit: relatively high Pirate recall coexists with lower precision for Pirate predictions. The classification should therefore be assessed through its full error profile, not the accuracy percentage alone.

4.4 Application and Research Implications

The study combines a systematic search over 63 feature-group combinations and nine tree depths with a comparison of standalone and ensemble classifiers. Integration of the saved model into Streamlit demonstrates how the selected characteristics can be entered through an interface and processed without retraining the model for each prediction.

The application returns a classification based on learned patterns in survey responses. Its outputs can support investigation of software-use behaviour, but they do not verify software licences or establish copyright infringement. Profiles entered into the interface are examples of model operation and should not be interpreted as behavioural rules applying to everyone with the same characteristics.

Reproducible use of the application requires the saved classifier to be accompanied by the feature definitions, category encoding and model settings used during development. Kapoor and Narayanan (2023) showed why separation of training and evaluation information is essential to credible predictive claims. In this study, the documented split and validation procedure provide the basis for interpreting the reported test results. Future extensions should preserve that separation when selecting predictors, adjusting settings or comparing additional models.

Future research could evaluate the framework with independently collected data from other software-use settings, compare additional classifiers and examine whether further behavioural variables improve predictive performance.

5. CONCLUSION

This study developed and implemented a hybridized software-piracy prediction framework using Decision Tree classification and AdaBoost. Six candidate feature groups were evaluated through 567

Decision Tree configurations. Expertise Level, Age Group, Region, and Importance of Ease of Pirating formed the selected combination. The optimized Decision Tree achieved 73.87% test accuracy, compared with 72.97% for AdaBoost, and had a higher Pirate-class F1-score. AdaBoost did not improve performance over the optimized standalone tree in this experiment.

The Python implementation and Streamlit application demonstrate a practical use of survey-derived characteristics for classifying self-reported software piracy. The contribution is an integrated procedure for feature selection, model comparison and interactive prediction. Its classifications describe patterns learned from the survey data and should be interpreted within that research context.

ACKNOWLEDGEMENTS

This study forms part of the PhD thesis of the lead author at the University of Port Harcourt, Nigeria, under the supervision and scholarly guidance of the other authors.

REFERENCES

Al-Rafee, S., & Cronan, T. P. (2006). Digital piracy: Factors that influence attitude toward behavior. *Journal of Business Ethics, 63*(3), 237–259. https://doi.org/10.1007/s10551-005-1902-9

Chavarria, J. A., Andoh-Baidoo, F. K., Midha, V., & Hughes, J. (2016). Software piracy research: A cross-disciplinary systematic review. *Communications of the Association for Information Systems, 38*, 624–669. https://doi.org/10.17705/1CAIS.03831

Chicco, D., & Jurman, G. (2023). The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification. *BioData Mining, 16*, Article 4. https://doi.org/10.1186/s13040-023-00322-4

Gomes, N. D., Cerqueira, P. A., & Almeida, L. A. (2015). A survey on software piracy empirical literature: Stylized facts and theory. *Information Economics and Policy, 32*, 29–37. https://doi.org/10.1016/j.infoecopol.2015.07.008

Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data? *Advances in Neural Information Processing Systems, 35*, 507–520. https://doi.org/10.52202/068431-0037

Kapoor, S., & Narayanan, A. (2023). Leakage and the reproducibility crisis in machine-learning-based science. *Patterns, 4*(9), Article 100804. https://doi.org/10.1016/j.patter.2023.100804

Koay, K. Y., Soh, P. C.-H., Tjiptono, F., Ramayah, T., & Lom, H. S. (2024). Understanding consumers' digital piracy behaviour: Explanation and prediction. *First Monday, 29*(7). https://doi.org/10.5210/fm.v29i7.13553

Lowry, P. B., Zhang, J., & Wu, T. (2017). Nature or nurture? A meta-analysis of the factors that maximize the prediction of digital piracy by using social cognitive theory as a framework. *Computers in Human Behavior, 68*, 104–120. https://doi.org/10.1016/j.chb.2016.11.015

Mallick, D., Gaurav, K., Bhattacharya, S., & Ghosh, S. (2024). Quantitative analysis of social influence and digital piracy contagion with differential equations on networks. *Scientific Reports, 14*, Article 10962. https://doi.org/10.1038/s41598-024-61531-3

Mishra, A., Akman, I., & Yazici, A. (2006). Software piracy among IT professionals in organizations. *International Journal of Information Management, 26*(5), 401–413. https://doi.org/10.1016/j.ijinfomgt.2006.01.004

Peace, A. G., Galletta, D. F., & Thong, J. Y. L. (2003). Software piracy in the workplace: A model and empirical test. *Journal of Management Information Systems, 20*(1), 153–177. https://doi.org/10.1080/07421222.2003.11045759

Rainio, O., Teuho, J., & Klén, R. (2024). Evaluation metrics and statistical tests for machine learning. *Scientific Reports, 14*, Article 6086. https://doi.org/10.1038/s41598-024-56706-x

Sikolia, D., Bahr, G., & Biros, D. (2025). Software piracy: The effects of neutralization and religiosity. *The Journal of the Southern Association for Information Systems, 12*(2), 1–17. https://doi.org/10.17705/3JSIS.00038