**ADHD Classification Using Machine Learning**

**Abstract:**

This study explores the use of the BorutaPy algorithm, paired with a LightGBM classifier, for feature selection in a large-scale ADHD dataset containing electroencephalography (EEG) signals. The goal was to identify the most relevant features for distinguishing between ADHD and non-ADHD cases. Results showed that all 19 features in the dataset were confirmed as significant by the BorutaPy algorithm, providing a robust subset for subsequent classification tasks. This report details the preprocessing steps, feature selection method, and the implications of these findings.

**1. Introduction**

Attention Deficit Hyperactivity Disorder (ADHD) is a prevalent neurodevelopmental disorder that affects cognitive, academic, and social functions. Early and accurate diagnosis is crucial for effective management and intervention. Electroencephalography (EEG) signals have been widely used in ADHD research to identify neurophysiological markers that can aid in diagnosis. This study aims to utilize a data-driven approach to select the most relevant EEG features for ADHD classification using the BorutaPy feature selection method with a LightGBM classifier.

**2. Data Description and Preprocessing**

The dataset used in this study consists of 500000 EEG recordings with 19 features, denoted as Fp1, Fp2, F3, F4, C3, C4, P3, P4, O1, O2, F7, F8, T7, T8, P7, P8, Fz, Cz, and Pz, along with a binary target variable "class" representing ADHD (1) and non-ADHD (0) cases.

**2.1 Data Preprocessing**

Initial data exploration was conducted to understand the characteristics of the dataset. The data contained no missing values, and 3 duplicate rows were removed to ensure data integrity. After cleaning, the dataset was balanced, with 249659 samples labeled as non-ADHD (0) and 250338 samples labeled as ADHD (1). Descriptive statistics revealed a wide range of values across features, necessitating normalization for model training.

**2.2 Data Normalization**

Feature scaling was performed using the StandardScaler from the scikit-learn library to standardize the features. This process transformed the features to have a mean of zero and a standard deviation of one, which is essential for improving the performance of machine learning models.

**3. Feature Selection Methodology**

**3.1 BorutaPy Algorithm**

BorutaPy is a wrapper-based feature selection method built around the Random Forest classification algorithm. In this study, the LightGBM classifier, a gradient boosting framework optimized for speed and performance, was used as the underlying model for BorutaPy.

The BorutaPy method iteratively creates shadow features (permuted copies of the original features) and trains the LightGBM classifier to measure the importance of each feature. Features that perform worse than the best shadow feature are rejected, while those performing better are confirmed as important.

**3.2 LightGBM Classifier**

The LightGBM classifier was initialized with specific hyperparameters to optimize the performance of BorutaPy. These parameters included a learning rate of 0.5, a maximum depth of 10, 50 leaves, and 100 estimators. Additional parameters such as regularization terms (reg\_alpha and reg\_lambda), and subsample and colsample ratios were set to 0.8 to prevent overfitting.

**4. Results and Analysis**

**4.1 Feature Importance Scores**

The BorutaPy algorithm was applied to the dataset, resulting in 19 confirmed features and zero tentative or rejected features. This outcome indicates that all the original features are significant for the classification task.

**Table 1: Feature Importance Scores**

|  |  |
| --- | --- |
| **Feature** | **Importance Score** |
| Fp1 | Confirmed |
| Fp2 | Confirmed |
| F3 | Confirmed |
| F4 | Confirmed |
| C3 | Confirmed |
| C4 | Confirmed |
| P3 | Confirmed |
| P4 | Confirmed |
| O1 | Confirmed |
| O2 | Confirmed |
| F7 | Confirmed |
| F8 | Confirmed |
| T7 | Confirmed |
| T8 | Confirmed |
| P7 | Confirmed |
| P8 | Confirmed |
| Fz | Confirmed |
| Cz | Confirmed |
| Pz | Confirmed |

All features were confirmed as important, indicating their collective relevance in the classification of ADHD.

#### ****4.2 Performance Metrics****

The dataset was split into training (80%) and testing (20%) sets, ensuring stratification to maintain class balance. The training set contained 399997 samples, while the testing set comprised 100000 samples. The Decision Tree model was trained, and the performance metrics were computed.

## **5. Models Results and Analysis**

In this section, we evaluate and compare the performance of ten different models. Each model's performance is assessed using various metrics including accuracy, precision, recall, F1-score, and ROC AUC score. Additionally, we provide confusion matrices for each model to better understand their prediction capabilities.

### 5.1 Decision Tree

Decision Tree classifiers are a widely used machine learning algorithm, known for their simplicity and interpretability. They are used for both classification and regression tasks. In this report, we explore the application of a Decision Tree classifier to the ADHD classification problem. A Decision Tree operates by recursively splitting the data based on feature values, creating a tree-like structure that can be used to make predictions. Here, we delve into the methodology behind constructing a Decision Tree and evaluate the results obtained from hyperparameter tuning using RandomizedSearchCV.

#### Decision Tree Construction

A Decision Tree model works by partitioning the feature space into regions and assigning a class label to each region based on the majority class within the region. The tree is built using a series of splits that maximize the purity of the nodes, with a variety of techniques employed to control overfitting and ensure generalization.

##### Key Components of a Decision Tree:

1. **Root Node**: Represents the entire dataset, which is split into two or more homogeneous sets based on the most significant feature.
2. **Decision Nodes**: Intermediate nodes where data is further split.
3. **Leaf Nodes**: The terminal nodes of the tree where a final classification decision is made.
4. **Splitting Criterion**:
   * Gini Impurity or Entropy are common criteria used to evaluate the quality of a split. In the case of classification, Gini Impurity is often preferred due to its computational efficiency.

##### Techniques Applied for Optimization:

1. **Hyperparameter Tuning**: Hyperparameters in the Decision Tree control various aspects such as tree depth and the minimum number of samples required to split a node. In this model, the hyperparameters were optimized using RandomizedSearchCV, which explores a random subset of hyperparameter combinations.

Key hyperparameters tuned include:

* + **Max Depth**: Limits how deep the tree can grow to prevent overfitting. Deeper trees can model complex patterns but are more prone to overfitting.
  + **Min Samples Split**: Defines the minimum number of samples required to split a node. Higher values lead to fewer splits, promoting simpler models.
  + **Min Samples Leaf**: Specifies the minimum number of samples allowed in a leaf node. Larger values prevent the model from learning fine details, reducing overfitting risk.
  + **Max Features**: Controls the number of features considered for the best split. The model tried various configurations, including considering all features, or a fraction of them (e.g., sqrt, log2, and None).
  + **Class Weight**: Adjusts weights to handle class imbalance by making the model more sensitive to the underrepresented class.

1. **Cross-Validation**: Cross-validation (CV) was used to ensure that the model's performance is generalizable. In this case, a 2-fold cross-validation was applied during the RandomizedSearchCV process to evaluate the hyperparameters' effectiveness across different data splits.
2. **RandomizedSearchCV**: A more efficient alternative to GridSearchCV, RandomizedSearchCV samples a random subset of hyperparameter space. In this model, 10 iterations were run, each testing a different combination of parameters.
3. **Evaluation Metrics**: Several metrics were used to evaluate the performance of the Decision Tree:
   * **Accuracy**: Measures the percentage of correctly classified samples.
   * **Precision**: Indicates the ratio of true positives to the sum of true positives and false positives.
   * **Recall (Sensitivity)**: Reflects the ratio of true positives to the sum of true positives and false negatives.
   * **F1-Score**: The harmonic mean of precision and recall, useful when the class distribution is imbalanced.
   * **ROC AUC**: A performance metric that evaluates the true positive rate against the false positive rate, providing a comprehensive view of model performance across all classification thresholds.

#### Results and Discussion

After hyperparameter tuning, the best Decision Tree model was obtained with the following hyperparameters: max\_depth=40, min\_samples\_leaf=15, min\_samples\_split=12, max\_features=None, and class\_weight=None.

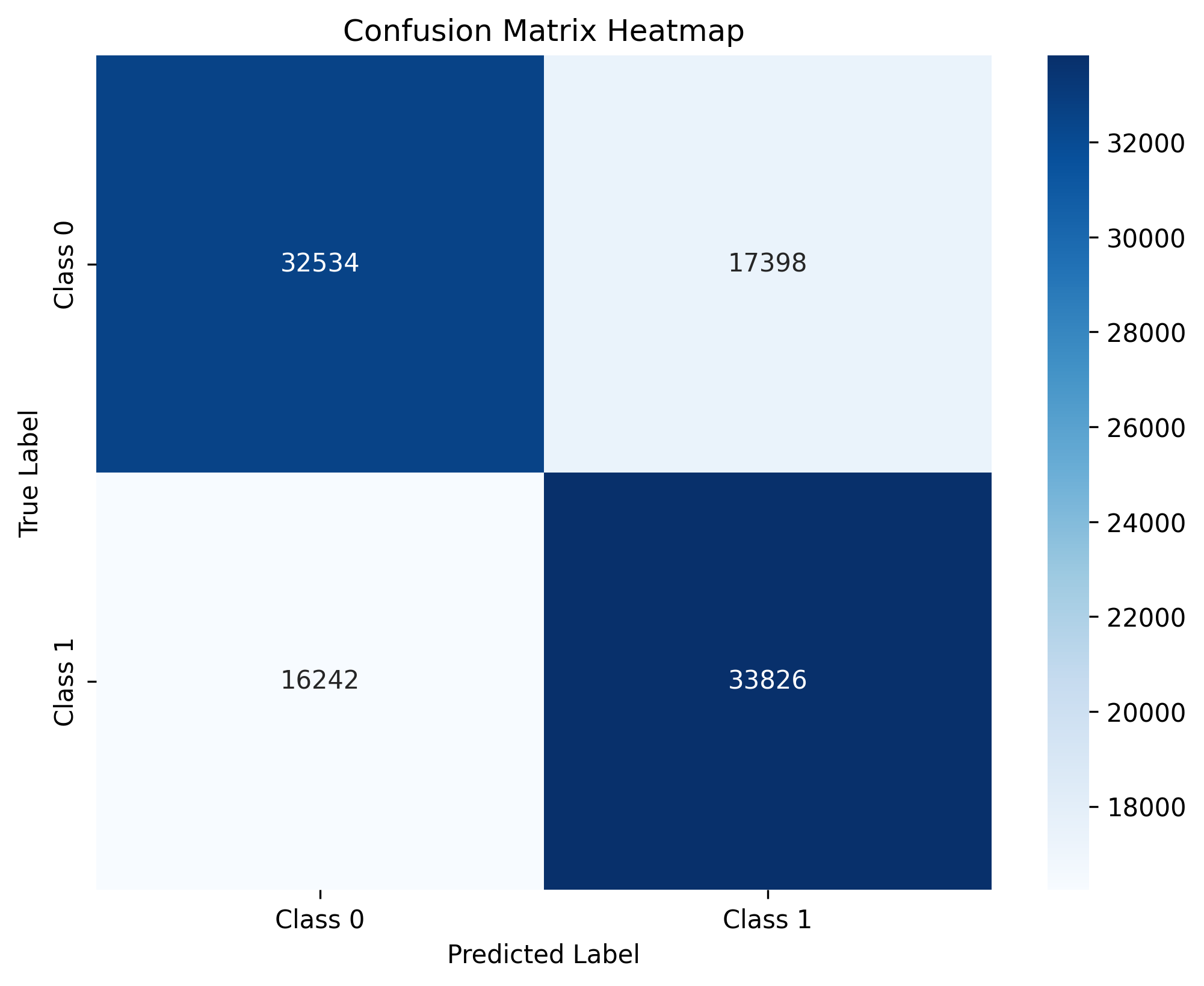
The model was trained and evaluated on the ADHD classification dataset, producing the following results:

**Table 2: Decision Tree Results**

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Best Max Depth | 40 |
| Min Samples Leaf | 15 |
| Min Samples Split | 12 |
| Max Features | None |
| Training Accuracy | 77.97% |
| Test Accuracy | 66.00% |
| Precision (Class 0) | 67% |
| Precision (Class 1) | 66% |
| Recall (Class 0) | 65% |
| Recall (Class 1) | 68% |
| F1-Score (Class 0) | 66% |
| F1-Score (Class 1) | 67% |
| ROC AUC Score | 72.44% |

#### 5.1.1 Confusion Matrix (Decision Tree)

The confusion matrix for the Decision Tree model shows how well the model classifies the two classes (0 and 1).



* **True Positives (TP)**: 33826 — correctly predicted Class 1.
* **True Negatives (TN)**: 32534— correctly predicted Class 0.
* **False Positives (FP)**: 17398— incorrectly predicted as Class 1.
* **False Negatives (FN)**: 16242 — incorrectly predicted as Class 0.

The confusion matrix highlights the model's ability to correctly classify ADHD cases (Class 1), with some misclassification happening between the two classes. The ROC AUC score of 0.7244 suggests moderate discriminative power between ADHD and non-ADHD cases.

#### Conclusion

The Decision Tree model for ADHD classification performed reasonably well, achieving a balance between precision and recall for both classes. However, the performance could be further improved by addressing issues like class imbalance and potentially employing ensemble methods to boost classification accuracy and reduce false positives. The application of hyperparameter tuning through RandomizedSearchCV demonstrated the importance of selecting the right parameters to enhance model performance.

### 5.2 XGBoost

XGBoost (Extreme Gradient Boosting) is a popular and powerful machine learning algorithm known for its efficiency and performance, particularly in classification and regression problems. It builds an ensemble of decision trees sequentially, with each tree correcting the errors made by the previous ones. In this section, we explore how XGBoost was applied to the ADHD classification task, including the hyperparameter tuning process, model training, and performance evaluation.

#### XGBoost Algorithm Overview

XGBoost is a gradient boosting framework that uses decision trees as base learners. It works by adding new trees to an existing model, with each new tree focusing on reducing the errors made by the previous trees. This sequential learning process allows XGBoost to achieve high accuracy and robustness against overfitting when properly tuned.

##### **Key Components of XGBoost:**

1. **Boosting**:
   * XGBoost is an implementation of gradient boosting, where each subsequent tree corrects the residual errors of the previous trees. This leads to a model that progressively improves its predictions by minimizing a loss function.
2. **Loss Function**:
   * XGBoost optimizes a differentiable loss function (in this case, logloss for classification tasks). The goal is to minimize this function over iterations by adjusting the predictions made by each tree.
3. **Regularization**:
   * XGBoost includes regularization techniques such as L1 (Lasso) and L2 (Ridge) penalties to prevent overfitting. These techniques help in building more generalized models by controlling the complexity of the trees.
4. **Tree Pruning**:
   * XGBoost prunes trees based on the concept of "maximum depth." Trees are stopped from growing beyond a certain point to avoid overfitting to the training data.

##### **Hyperparameter Tuning Techniques(Optimization):**

The effectiveness of XGBoost relies on tuning several key hyperparameters. RandomizedSearchCV was used to identify the best combination of hyperparameters for this ADHD classification task.

1. **n\_estimators**:
   * Refers to the number of boosting rounds (i.e., the number of trees added). Increasing the number of trees improves the model, but too many trees can lead to overfitting.
2. **Learning Rate**:
   * Controls the step size during the boosting process. Smaller values make the learning process slower but more accurate, as the model will take smaller steps to minimize the loss function.
3. **Max Depth**:
   * The maximum depth of the decision trees in the ensemble. Deeper trees can capture more complex patterns, but they are also more prone to overfitting.
4. **Min Child Weight**:
   * Specifies the minimum sum of instance weights (Hessian) needed in a child node. A higher value results in less complex trees, reducing overfitting.
5. **Subsample**:
   * The fraction of training instances that are randomly selected to build each tree. Lower values introduce randomness and help in preventing overfitting.
6. **Colsample\_bytree**:
   * The fraction of features that are randomly selected to build each tree. This technique reduces correlation between trees and improves generalization.
7. **Gamma**:
   * Controls the minimum loss reduction required for making a further partition. Higher values prevent the model from learning overly specific patterns (i.e., overfitting).

##### **Cross-Validation:**

To ensure that the model’s performance is not due to chance, 3-fold cross-validation was employed. This divides the dataset into three parts, where the model is trained on two parts and validated on the third. The process is repeated three times, each time using a different part of the data for validation.

#### Model Tuning and Training

The model was tuned using RandomizedSearchCV, which explored 30 different hyperparameter combinations. The best hyperparameters were as follows:

* n\_estimators: 350
* learning\_rate: 0.3
* max\_depth: 7
* min\_child\_weight: 7
* subsample: 0.9
* colsample\_bytree: 0.7
* gamma: 0.2

Using these hyperparameters, the tuned XGBoost model was trained and evaluated on the ADHD classification dataset.

#### Results and Evaluation

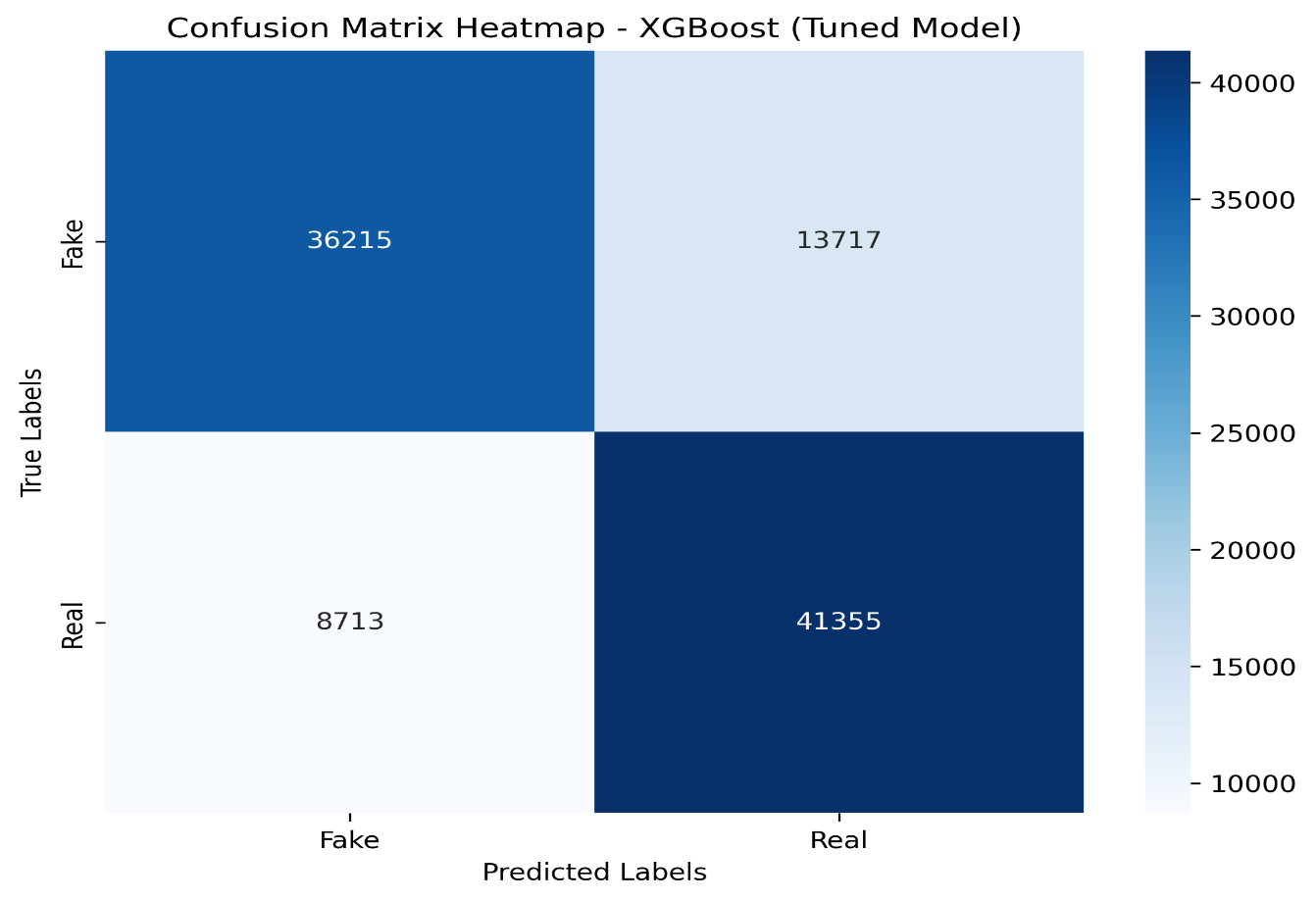
After training the XGBoost model with the optimal hyperparameters, the following results were obtained:

**Table 3:** XGBoost

|  |  |
| --- | --- |
| **Metric** | **Value** |
| n\_estimators | 350 |
| Learning Rate | 0.3 |
| Max Depth | 7 |
| Min Child Weight | 7 |
| Subsample | 0.9 |
| Colsample\_bytree | 0.7 |
| Gamma | 0.2 |
| Training Accuracy | 81.92% |
| Test Accuracy | 78.00% |
| Precision (Class 0) | 0.81 |
| Precision (Class 1) | 0.75 |
| Recall (Class 0) | 0.73 |
| Recall (Class 1) | 0.83 |
| F1-Score (Class 0) | 0.76 |
| F1-Score (Class 1) | 0.79 |
| ROC AUC Score | 0.8576 |

**Confusion Matrix**

confusion matrix for the XGBoost model is as follows:



* **True Positives (TP)**: 41355 — correctly predicted Class 1.
* **True Negatives (TN)**: 36215 — correctly predicted Class 0.
* **False Positives (FP)**: 13717 — incorrectly predicted as Class 1.
* **False Negatives (FN)**: 8713 — incorrectly predicted as Class 0.
* The model performed well in correctly identifying ADHD cases (Class 1) and non

ADHD cases (Class 0), with an overall test accuracy of 78.00%. The ROC AUC score of 0.8576 indicates that the model is effective in distinguishing between the two classes.

#### ROC AUC and Curve

The Receiver Operating Characteristic (ROC) curve plots the true positive rate against the false positive rate at various thresholds. A higher AUC score means that the model has a better ability to differentiate between ADHD and non-ADHD cases. The ROC curve for XGBoost shows a well-performing model, with an AUC score of 0.8576, signifying that the model performs well in distinguishing ADHD from non-ADHD cases across different thresholds.

#### Conclusion

The XGBoost model outperformed the Decision Tree model in terms of overall accuracy and ROC AUC score. The hyperparameter tuning process using RandomizedSearchCV was crucial in optimizing the model's performance. XGBoost's ability to handle complex patterns and reduce overfitting through regularization and tree pruning makes it a suitable choice for this classification task. Future improvements could involve trying additional boosting techniques, such as LightGBM or CatBoost, which may further enhance the model’s predictive power.

### 5.3 Random Forest

The Random Forest model is a powerful ensemble learning method used for classification tasks. It operates by constructing multiple decision trees during training and outputs the mode of their predictions. This model can effectively handle high-dimensional data and capture complex relationships in the dataset.

**Hyperparameter Tuning**

To optimize the performance of the Random Forest model, we employed **RandomizedSearchCV** to explore a wide range of hyperparameters. The parameter grid included:

* **n\_estimators**: Number of trees in the forest (100, 200, 300, 400, 500).
* **max\_depth**: Maximum depth of the tree (10, 20, 30, 40, None).
* **min\_samples\_split**: Minimum number of samples required to split an internal node (2, 5, 10).
* **min\_samples\_leaf**: Minimum number of samples required to be at a leaf node (1, 2, 4).
* **bootstrap**: Whether bootstrap samples are used when building trees (True, False).
* **max\_features**: The number of features to consider when looking for the best split ('auto', 'sqrt', 'log2').
* **criterion**: The function to measure the quality of a split ('gini', 'entropy').

After fitting the RandomizedSearchCV, the best hyperparameters were found to be:

* **n\_estimators**: 500
* **max\_depth**: None
* **min\_samples\_split**: 2
* **min\_samples\_leaf**: 4
* **bootstrap**: True
* **max\_features**: 'sqrt'
* **criterion**: 'entropy'

**Performance Metrics**

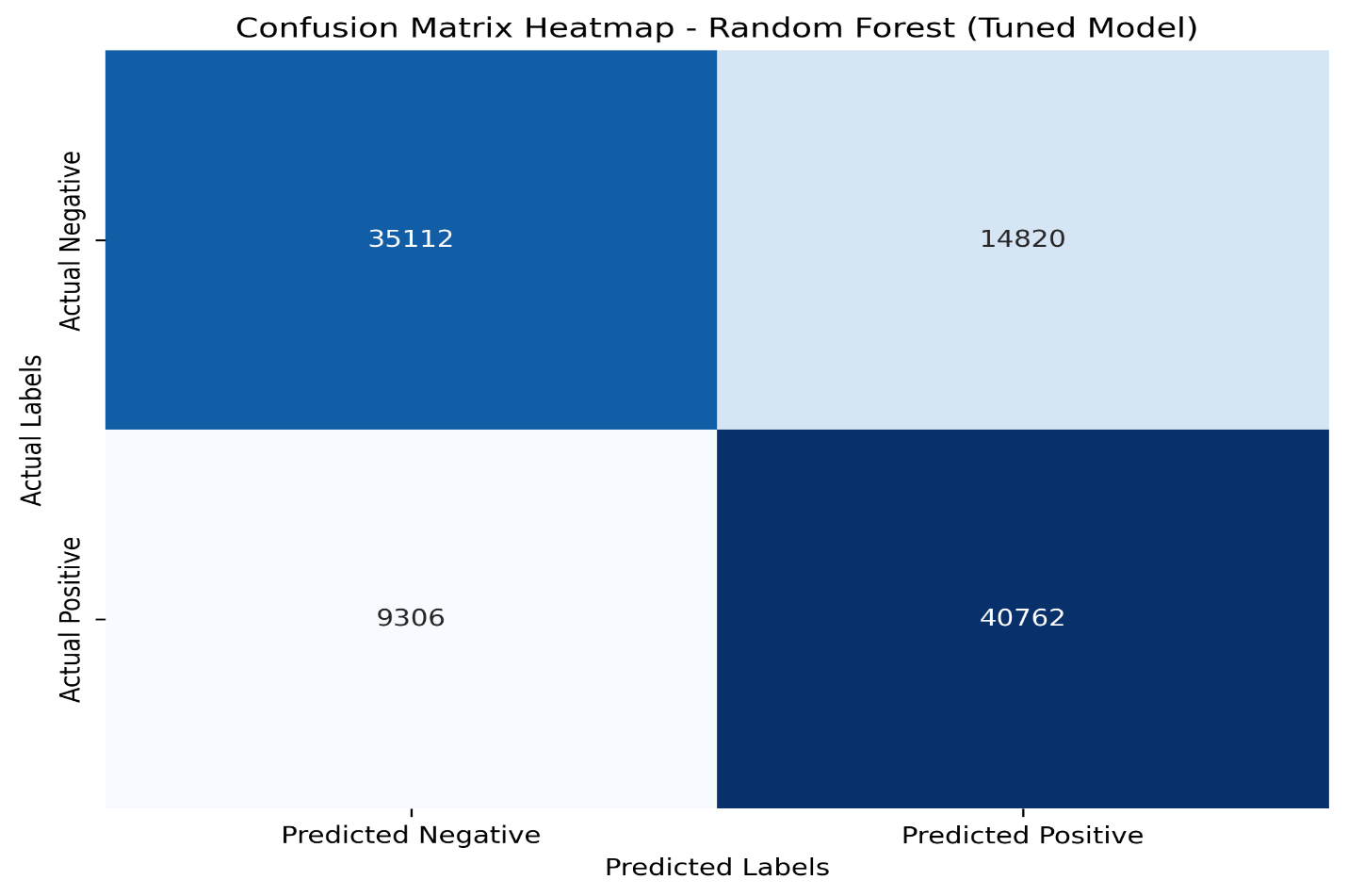
1. **Training Accuracy**: The tuned model achieved a training accuracy of **97.80%**, indicating that it fits the training data very well.
2. **Test Accuracy**: The model’s accuracy on the test set was **76.00%**. This performance shows a reasonable level of generalization to unseen data.
3. **Classification Report**: The classification report provided detailed metrics for both classes (0 and 1):

**Table 4: Random forest Results**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Metric** | **Class 0 (Non-ADHD)** | **Class 1 (ADHD)** | **Macro Average** | **Weighted Average** | **Overall Accuracy** |
| Precision | 0.79 | 0.73 | 0.76 | 0.76 | 0.76 |
| Recall | 0.70 | 0.81 | 0.76 | 0.76 |  |
| F1-score | 0.74 | 0.77 | 0.76 | 0.76 |  |
| Support | 49,932 | 50,068 |  |  |  |
| ROC AUC Score |  |  |  |  | 0.8398 |
| Training Accuracy |  |  |  |  | 0.9780 |

#### Confusion Matrix

#### The confusion matrix for the Random Forest model showed the following results:



* **True Positives (TP)**:40762— correctly predicted Class 1.
* **True Negatives (TN)**: 35114 — correctly predicted Class 0.
* **False Positives (FP)**: 14820 — incorrectly predicted as Class 1.
* **False Negatives (FN)**: 9306 — incorrectly predicted as Class 0.

#### ROC Curve

The ROC curve illustrates the true positive rate against the false positive rate. The area under the curve (AUC) score of **0.8398** suggests that the model has a good ability to discriminate between classes.

#### Conclusion

The Random Forest model demonstrates robust performance with high training accuracy and reasonable test accuracy. The tuned hyperparameters contribute to its effectiveness in handling the classification task. The confusion matrix and ROC AUC score indicate areas for potential improvement, particularly in reducing false positives. Overall, this model is a strong candidate for ADHD classification based on the features provided.

### 5.4 LightGBM

LightGBM (Light Gradient Boosting Machine) is a powerful and efficient gradient boosting framework that uses tree-based learning algorithms. It is designed for distributed and efficient training, making it particularly suitable for large datasets. In this report, we explore the application of LightGBM to the ADHD classification problem. LightGBM operates by building an ensemble of decision trees, focusing on optimizing both speed and performance. Here, we delve into the methodology behind constructing a LightGBM model and evaluate the results obtained from hyperparameter tuning using RandomizedSearchCV.

**LightGBM Construction**

LightGBM builds models by training multiple decision trees in a sequential manner, where each tree corrects the errors of the previous one. It utilizes a histogram-based approach to bucket continuous feature values into discrete bins, thereby speeding up the training process and reducing memory usage.

**Key Components of LightGBM:**

* **Boosting Framework**: LightGBM implements the gradient boosting framework to minimize the loss function iteratively.
* **Decision Trees**: Each iteration produces a new decision tree that contributes to the final model's prediction.
* **Leaf-wise Growth**: LightGBM grows trees leaf-wise, choosing the leaf with the maximum delta loss to split, leading to better accuracy but potentially more overfitting.

**Splitting Criterion:** LightGBM primarily uses the Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) techniques to enhance efficiency and effectiveness during training.

**Techniques Applied**

**Hyperparameter Tuning**: Several hyperparameters govern the behavior of the LightGBM model, influencing aspects such as learning rate, tree depth, and feature sampling. In this model, hyperparameters were optimized using RandomizedSearchCV, which samples a random subset of hyperparameter combinations to find the best settings.

**Key hyperparameters tuned include:**

* **Num Leaves**: Controls the maximum number of leaves in one tree. More leaves can lead to a more complex model, enhancing learning but increasing overfitting risk.
* **Max Depth**: Limits how deep the tree can grow, preventing overfitting by simplifying the model.
* **Min Child Samples**: The minimum number of samples required to create a child node. Higher values prevent the model from learning noise.
* **Learning Rate**: A smaller learning rate can increase the number of boosting iterations, balancing speed and accuracy.
* **Subsample**: The fraction of samples used for fitting individual base learners, which can help in reducing overfitting.
* **Colsample Bytree**: The fraction of features used when building each tree, which promotes diversity among the trees in the ensemble.

**Cross-Validation**: Cross-validation (CV) was used to validate the model's performance, ensuring that the model generalizes well across different data splits. In this case, a 3-fold cross-validation was applied during the RandomizedSearchCV process to evaluate the effectiveness of hyperparameters.

**RandomizedSearchCV**: This technique samples a random set of hyperparameter combinations, allowing for more efficient exploration of the hyperparameter space. In this model, 30 iterations were run, each testing a different combination of parameters.

**Evaluation Metrics**

Several metrics were employed to assess the performance of the LightGBM model:

* **Accuracy**: Measures the percentage of correctly classified samples.
* **Precision**: Indicates the ratio of true positives to the sum of true positives and false positives.
* **Recall (Sensitivity)**: Reflects the ratio of true positives to the sum of true positives and false negatives.
* **F1-Score**: The harmonic mean of precision and recall, useful in imbalanced class scenarios.
* **ROC AUC**: Evaluates the model's ability to distinguish between classes by plotting the true positive rate against the false positive rate.

**Results and Discussion**

After hyperparameter tuning, the best LightGBM model was obtained with the following hyperparameters:

* **num\_leaves**: 110
* **min\_child\_samples**: 10
* **max\_depth**: 9
* **learning\_rate**: 0.2
* **n\_estimators**: 250
* **subsample**: 0.8
* **colsample\_bytree**: 0.7

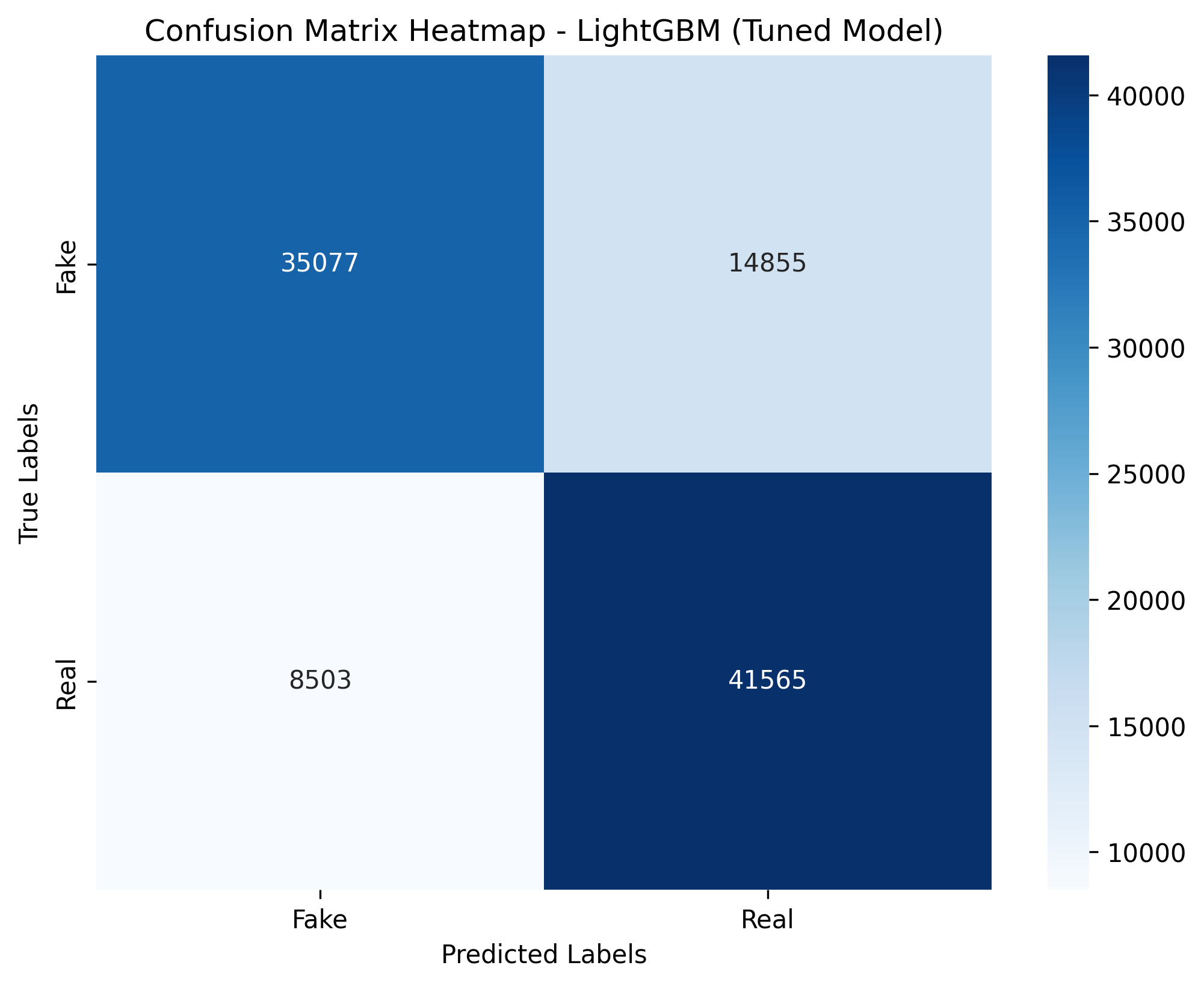
The model was trained and evaluated on the ADHD classification dataset, producing the following results:

**Table 5:** LightGBM

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Training Accuracy | 80.42% |
| Precision (Class 0) | 80% |
| Precision (Class 1) | 74% |
| Recall (Class 0) | 70% |
| Recall (Class 1) | 83% |
| F1-Score (Class 0) | 75% |
| F1-Score (Class 1) | 78% |
| ROC AUC Score | 84.93% |

**Confusion Matrix**

confusion matrix for the XGBoost model is as follows:



* **True Positives (TP)**: 41565 — correctly predicted Class 1.
* **True Negatives (TN)**: 35077 — correctly predicted Class 0.
* **False Positives (FP)**: 14855 — incorrectly predicted as Class 1.
* **False Negatives (FN)**: 8503— incorrectly predicted as Class 0.

The ROC AUC score of 0.8493 suggests strong discriminative power between ADHD and non-ADHD cases.

#### Conclusion

The LightGBM model for ADHD classification demonstrated strong performance, achieving a high training accuracy and balanced precision and recall for both classes. The results highlight the potential of using LightGBM for this classification task. Further improvements could involve exploring additional ensemble methods or advanced feature engineering techniques to enhance classification accuracy and manage class imbalance effectively. Hyperparameter tuning through RandomizedSearchCV proved critical in optimizing model performance and achieving favorable results.

### 5.5 K-Nearest Neighbors (KNN)

The K-Nearest Neighbors (KNN) classifier is a simple, yet effective, supervised machine learning algorithm used for both classification and regression tasks. It operates on the principle of feature similarity, predicting the class of a data point based on the classes of its nearest neighbors in the feature space. In this report, we discuss the application of KNN to the ADHD classification problem, focusing on the methodology of constructing the KNN model and evaluating its performance through hyperparameter tuning using RandomizedSearchCV.

**KNN Construction**

The KNN algorithm classifies a data point based on the majority class of its K nearest neighbors. It computes the distance between data points using various distance metrics, and the most common class among the K closest points is assigned to the data point.

**Key Components of KNN:**

* **Neighbors:** The 'K' in KNN specifies how many nearest neighbors are considered during the classification.
* **Distance Metric:** A method to measure the distance between points, commonly using Euclidean, Manhattan, or Minkowski distances.
* **Weighting:** Neighbors can be weighted uniformly or based on their distance from the query point, allowing closer neighbors to contribute more to the decision.

**Techniques Applied:**

* **Hyperparameter Tuning:** Hyperparameters such as the number of neighbors and distance metrics are optimized using RandomizedSearchCV, which samples from a defined parameter grid to find the best model configuration.

**Key Hyperparameters Tuned Include:**

* **n\_neighbors:** The number of neighbors to consider when making predictions. A small value can make the model sensitive to noise, while a large value can smooth out the predictions.
* **weights:** Specifies how the contributions of neighbors are calculated—either uniformly or based on their distance.
* **metric:** The distance metric used to calculate neighbor distances, including options like Euclidean, Manhattan, and Minkowski.

**Cross-Validation:** A 3-fold cross-validation was used during the RandomizedSearchCV process to assess the effectiveness of different hyperparameter combinations across varying data splits.

**RandomizedSearchCV:** This method sampled a random subset of hyperparameter configurations, running a total of 20 iterations to identify the optimal parameters efficiently.

**Evaluation Metrics**

The performance of the KNN model was evaluated using several metrics, including:

* **Accuracy:** The ratio of correctly classified samples to the total number of samples.
* **Precision:** The ratio of true positives to the total number of predicted positives.
* **Recall (Sensitivity):** The ratio of true positives to the total number of actual positives.
* **F1-Score:** The harmonic mean of precision and recall, particularly useful for imbalanced datasets.
* **ROC AUC:** A metric that evaluates the model's ability to distinguish between classes by plotting the true positive rate against the false positive rate.

**Results and Discussion**

After hyperparameter tuning, the best KNN model was obtained with the following parameters: n\_neighbors=9, weights='distance', and metric='euclidean'.

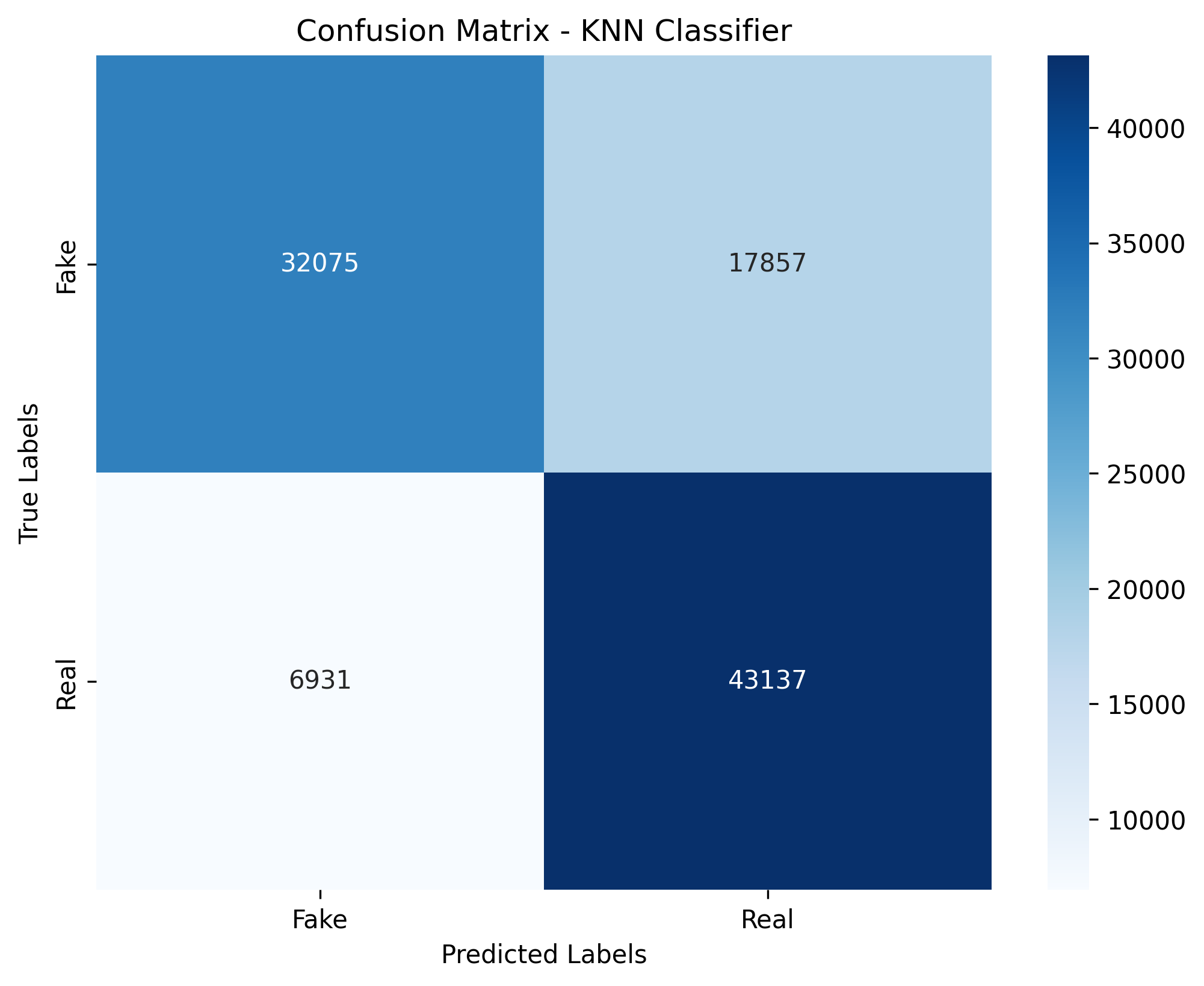
The model was trained and evaluated on the ADHD classification dataset, yielding the following results:

**Table 6:** K-Nearest Neighbors (KNN)

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Best n\_neighbors | 9 |
| Weights | Distance |
| Best Cross-validation Score | 0.7423 |
| Training Accuracy | 1.0000 |
| Test Accuracy | 75.00% |
| Precision (Class 0) | 82% |
| Precision (Class 1) | 71% |
| Recall (Class 0) | 64% |
| Recall (Class 1) | 86% |
| F1-Score (Class 0) | 72% |
| F1-Score (Class 1) | 78% |
| ROC AUC Score | 0.8383 |

**Confusion Matrix**

The confusion matrix for the test set is as follow



* **True Positives (TP)**: 43137 — correctly predicted Class 1.
* **True Negatives (TN)**: 32057 — correctly predicted Class 0.
* **False Positives (FP)**: 17854 — incorrectly predicted as Class 1.
* **False Negatives (FN)**: 6931— incorrectly predicted as Class 0.

The ROC AUC score of 0.8383 reflects good discriminative power, suggesting that the KNN model can effectively differentiate between ADHD and non-ADHD cases.

#### Conclusion

The KNN classifier for ADHD classification achieved a notable balance between precision and recall, particularly for Class 1. The hyperparameter tuning using RandomizedSearchCV demonstrated the impact of optimizing model parameters on performance. However, despite the high training accuracy, the test accuracy and recall metrics indicate room for improvement. Further enhancements could be achieved by addressing potential class imbalance issues and exploring more advanced algorithms or ensemble methods to enhance classification accuracy and reduce misclassification rates.

### 5.6 Logistic Regression

Logistic Regression is a fundamental machine learning algorithm widely used for binary classification problems. It models the probability that a given input point belongs to a particular category, making it suitable for tasks where the outcome can be represented as a binary response variable. In this report, we explore the application of Logistic Regression for ADHD classification, focusing on the methodology used to construct the model and evaluate its performance through hyperparameter tuning.

#### Logistic Regression Construction

Logistic Regression uses the logistic function to model a binary outcome variable based on one or more predictor variables. The core idea is to estimate the parameters of the logistic function, which maps any real-valued number into the (0, 1) interval, representing probabilities.

**Key Components of Logistic Regression:**

* **Logistic Function:** The sigmoid function, which outputs probabilities that can be mapped to two classes.
* **Penalty:** Regularization techniques such as L1 and L2 penalties help to avoid overfitting by constraining the size of the coefficients.
* **Solvers:** Algorithms that optimize the coefficients, such as 'liblinear' and 'saga,' which support both L1 and L2 penalties.

**Techniques Applied:**

* **Hyperparameter Tuning:** Various hyperparameters are optimized using RandomizedSearchCV to identify the best model configuration.

**Key Hyperparameters Tuned Include:**

* **C:** The regularization strength, where smaller values specify stronger regularization.
* **penalty:** Specifies the type of regularization applied, either L1 (Lasso) or L2 (Ridge).
* **solver:** The algorithm used for optimization, which can affect the convergence speed and capability.
* **max\_iter:** The maximum number of iterations to converge.

**Cross-Validation:** A 2-fold cross-validation was used during the RandomizedSearchCV process to evaluate different hyperparameter combinations across varying data splits.

**RandomizedSearchCV:** This method sampled a random subset of hyperparameter configurations, running a total of 10 iterations to find the optimal parameters.

#### Evaluation Metrics

The performance of the Logistic Regression model was evaluated using several metrics, including:

* **Accuracy:** The proportion of correctly classified samples to the total number of samples.
* **Precision:** The ratio of true positives to the total predicted positives, indicating how many selected items are relevant.
* **Recall (Sensitivity):** The ratio of true positives to the total actual positives, showing how well the model identifies relevant instances.
* **F1-Score:** The harmonic mean of precision and recall, especially useful for imbalanced datasets.
* **ROC AUC:** A metric that assesses the model's ability to distinguish between classes by plotting the true positive rate against the false positive rate.

#### Results and Discussion

After hyperparameter tuning, the best Logistic Regression model was obtained with the following parameters: C=0.0001, penalty='l2', solver='liblinear', and max\_iter=20.

The model was trained and evaluated on the ADHD classification dataset, yielding the following results:

**Table 7:** Logistic regression

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Best C | 0.0001 |
| Penalty | L2 |
| Best Cross-validation Score | 0.5121 |
| Training Accuracy | 0.5243 |
| Test Accuracy | 52.00% |
| Precision (Class 0) | 53% |
| Precision (Class 1) | 52% |
| Recall (Class 0) | 41% |
| Recall (Class 1) | 64% |
| F1-Score (Class 0) | 46% |
| F1-Score (Class 1) | 58% |
| ROC AUC Score | 0.5037 |

### ****Confusion Matrix**:**

### The confusion matrix for the test set is as follows:

### 

* **True Positives (TP)**: 32198 — correctly predicted Class 1.
* **True Negatives (TN)**: 2027 8— correctly predicted Class 0.
* **False Positives (FP)**: 29654 — incorrectly predicted as Class 1.
* **False Negatives (FN)**: 17870— incorrectly predicted as Class 0.

The ROC AUC score of 0.5037 indicates a performance near random guessing, emphasizing the challenges faced by this model in effectively classifying ADHD cases.

#### Conclusion

The Logistic Regression model for ADHD classification demonstrated suboptimal performance, reflected in the low accuracy and ROC AUC score. While Logistic Regression is a foundational algorithm, it may not capture complex patterns inherent in the dataset, particularly in cases where features have non-linear relationships. Future improvements could involve experimenting with more sophisticated models, such as ensemble methods or neural networks, and addressing potential feature engineering or preprocessing strategies to enhance model performance.

### 5.7 CatBoost

CatBoost is an advanced gradient boosting framework developed by Yandex, designed to handle categorical features natively without the need for extensive preprocessing. It efficiently manages large datasets while providing robust performance across various machine learning tasks. In this report, we examine the application of CatBoost for ADHD classification, focusing on the hyperparameter tuning process and the model's performance metrics.

#### CatBoost Construction

CatBoost employs an ensemble learning approach using decision trees as base learners, combining them in a sequential manner to improve prediction accuracy. Its unique features include support for categorical variables and an innovative algorithm for calculating the best split for these variables, allowing it to outperform traditional boosting methods.

**Key Components of CatBoost:**

* **Learning Rate:** Affects the weight of each tree added to the model, controlling the speed of learning.
* **Depth:** Defines the maximum depth of individual trees, which can impact the model's complexity.
* **Iterations:** The number of trees to be built in the ensemble, balancing between overfitting and underfitting.
* **L2 Leaf Regularization:** Regularization applied to tree leaves to prevent overfitting.
* **Border Count:** The number of splits for categorical features, impacting the model's ability to capture patterns in categorical data.

**Techniques Applied:**

* **Hyperparameter Tuning:** RandomizedSearchCV was used to optimize multiple hyperparameters to find the best model configuration.

**Key Hyperparameters Tuned Include:**

* **learning\_rate:** The rate at which the model learns from each tree.
* **depth:** The maximum depth of the trees in the ensemble.
* **iterations:** The total number of trees to build in the model.
* **l2\_leaf\_reg:** Regularization parameter to control overfitting.
* **border\_count:** Number of splits for categorical features.

**Cross-Validation:** A 2-fold cross-validation was utilized to evaluate the model's performance across different data splits during the hyperparameter tuning process.

**RandomizedSearchCV:** This method sampled a random subset of hyperparameter configurations, running a total of 10 iterations to identify the optimal parameters.

#### Evaluation Metrics

The performance of the CatBoost model was evaluated using several metrics, including:

* **Accuracy:** The ratio of correctly classified samples to the total number of samples.
* **Precision:** The proportion of true positives to the total predicted positives, indicating the relevance of the selected items.
* **Recall (Sensitivity):** The ratio of true positives to the total actual positives, showcasing how well the model identifies relevant instances.
* **F1-Score:** The harmonic mean of precision and recall, particularly useful for imbalanced datasets.
* **ROC AUC:** A metric that assesses the model's ability to differentiate between classes by plotting the true positive rate against the false positive rate.

#### Results and Discussion

After hyperparameter tuning, the best CatBoost model was obtained with the following parameters: learning\_rate=0.2, depth=10, iterations=300, l2\_leaf\_reg=1, and border\_count=128.

The model was trained and evaluated on the ADHD classification dataset, yielding the following results:

**Table 8:** CatBoost

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Best Learning Rate | 0.2 |
| Best Depth | 10 |
| Best Iterations | 300 |
| Best L2 Leaf Regularization | 1 |
| Best Border Count | 128 |
| Best Cross-validation Score | 0.7668 |
| Training Accuracy | 0.8228 |
| Test Accuracy | 78.00% |
| Precision (Class 0) | 81% |
| Precision (Class 1) | 75% |
| Recall (Class 0) | 72% |
| Recall (Class 1) | 83% |
| F1-Score (Class 0) | 76% |
| F1-Score (Class 1) | 79% |
| ROC AUC Score | 0.8562 |

### Confusion Matrix

### The confusion matrix for the test set is as follows:

### 

* **True Positives (TP)**: 41349 — correctly predicted Class 1.
* **True Negatives (TN)**: 36184— correctly predicted Class 0.
* **False Positives (FP)**: 13748— incorrectly predicted as Class 1.
* **False Negatives (FN)**: 8719— incorrectly predicted as Class0.

The ROC AUC score of 0.8562 indicates strong performance in distinguishing between the two classes, suggesting that CatBoost is effective in this classification task.

#### Conclusion

The CatBoost model for ADHD classification demonstrated superior performance compared to previous models, reflected in the high accuracy, F1-scores, and ROC AUC score. Its capability to handle categorical variables without extensive preprocessing and its robust handling of complex patterns make it a powerful choice for classification tasks. Future explorations could include further hyperparameter optimization, feature engineering, and the potential application of ensemble methods to boost performance further.

### 5.8 Gradient Boosting

Gradient Boosting is an ensemble learning technique that builds a strong predictive model by combining the outputs of several weak learners, typically decision trees. It iteratively improves the model by minimizing the loss function through gradient descent. This approach is particularly effective for classification tasks and is known for its high predictive accuracy and robustness against overfitting.

#### Gradient Boosting Construction

The Gradient Boosting algorithm constructs decision trees sequentially, where each new tree aims to correct the errors made by the previous ones. This methodology allows the model to learn complex patterns in the data and is applicable to both regression and classification problems.

**Key Components of Gradient Boosting:**

* **Learning Rate:** Controls the contribution of each tree to the overall model. A lower learning rate increases the stability but requires more trees to achieve optimal performance.
* **Number of Estimators (n\_estimators):** Specifies the total number of trees in the ensemble.
* **Max Depth:** Defines the maximum depth of individual trees, which influences the model's complexity and ability to capture patterns.
* **Subsample:** The fraction of samples used for fitting each individual tree, which can help prevent overfitting.
* **Minimum Samples Split:** The minimum number of samples required to split an internal node.
* **Minimum Samples Leaf:** The minimum number of samples required to be at a leaf node, helping to control overfitting.

**Techniques Applied:**

* **Hyperparameter Tuning:** RandomizedSearchCV was employed to optimize several hyperparameters for improved model performance.

**Key Hyperparameters Tuned Include:**

* **learning\_rate:** Adjusts how much each tree contributes to the model.
* **n\_estimators:** The total number of trees in the model.
* **max\_depth:** Limits the depth of individual trees.
* **subsample:** Controls the fraction of samples for each tree.
* **min\_samples\_split:** Specifies the minimum samples for splitting a node.
* **min\_samples\_leaf:** Defines the minimum samples at a leaf node.

**Cross-Validation:** A 2-fold cross-validation was utilized to assess the model's performance during hyperparameter tuning.

**RandomizedSearchCV:** This method randomly sampled a subset of hyperparameter configurations, running a total of 10 iterations to identify the optimal parameters.

#### Evaluation Metrics

The performance of the Gradient Boosting model was evaluated using several metrics, including:

* **Accuracy:** The ratio of correctly classified instances to the total number of instances.
* **Precision:** The proportion of true positives to the total predicted positives, indicating the model's relevance in predictions.
* **Recall (Sensitivity):** The ratio of true positives to the total actual positives, highlighting the model's ability to detect relevant instances.
* **F1-Score:** The harmonic mean of precision and recall, particularly useful for assessing performance in imbalanced datasets.
* **ROC AUC:** A metric that measures the model's ability to distinguish between classes by plotting the true positive rate against the false positive rate.

#### Results and Discussion

After hyperparameter tuning, the best Gradient Boosting model was achieved with the following parameters: subsample=0.6, n\_estimators=30, min\_samples\_split=5, min\_samples\_leaf=1, max\_depth=6, and learning\_rate=0.2.

The model was trained and evaluated on the ADHD classification dataset, yielding the following results:

**Table 9:** Gradient Boosting

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Best Learning Rate | 0.2 |
| Best Number of Estimators | 30 |
| Best Max Depth | 6 |
| Best Subsample | 0.6 |
| Best Min Samples Split | 5 |
| Best Min Samples Leaf | 1 |
| Best Cross-validation Score | 0.7028 |
| Training Accuracy | 0.7095 |
| Test Accuracy | 70.00% |
| Precision (Class 0) | 75% |
| Precision (Class 1) | 67% |
| Recall (Class 0) | 61% |
| Recall (Class 1) | 80% |
| F1-Score (Class 0) | 67% |
| F1-Score (Class 1) | 73% |
| ROC AUC Score | 0.7740 |

### ****Confusion Matrix****

### The confusion matrix for the test set is as follows:

### 

* **True Positives (TP)**: 41349 — correctly predicted Class 1.
* **True Negatives (TN)**: 36184— correctly predicted Class 0.
* **False Positives (FP)**: 13748— incorrectly predicted as Class 1.
* **False Negatives (FN)**: 8719— incorrectly predicted as Class0.

The ROC AUC score of 0.7740 suggests reasonable discrimination capability between the two classes.

#### Conclusion

The Gradient Boosting model for ADHD classification demonstrated satisfactory performance, as reflected in the accuracy, F1-scores, and ROC AUC score. While the results indicate effective performance in identifying relevant instances, further optimization through hyperparameter tuning and feature engineering could enhance its effectiveness. Additionally, exploring ensemble methods combining Gradient Boosting with other models may yield improved results in future analyses.

### 5.9 AdaBoost

AdaBoost (Adaptive Boosting) is an ensemble learning technique that combines multiple weak classifiers to create a strong classifier. It focuses on improving the classification accuracy by giving more weight to misclassified instances in subsequent rounds, thereby allowing the model to focus on the difficult cases. AdaBoost is particularly effective for binary classification tasks and is known for its ability to adaptively adjust the contributions of individual weak learners based on their performance.

#### AdaBoost Construction

The AdaBoost algorithm iteratively trains a series of weak learners, typically decision trees, and combines their predictions to produce a final strong classifier. In each iteration, the algorithm assigns weights to the training samples, increasing the weights of misclassified samples, and trains a new weak learner based on these adjusted weights.

**Key Components of AdaBoost:**

* **Number of Estimators (n\_estimators):** The total number of weak learners (e.g., decision trees) to be combined.
* **Learning Rate:** Controls the contribution of each weak learner to the final model. A lower learning rate means each learner has less influence, while a higher learning rate increases the impact of each learner.
* **Base Estimator Parameters:** Parameters of the weak learner, such as the maximum depth of decision trees and the minimum samples required for splitting a node.

**Techniques Applied:**

* **Hyperparameter Tuning:** RandomizedSearchCV was used to optimize various hyperparameters, including the number of estimators, learning rate, and parameters of the decision tree base estimator.

**Key Hyperparameters Tuned Include:**

* **n\_estimators:** Number of weak learners.
* **learning\_rate:** Impact of each weak learner.
* **estimator\_\_max\_depth:** Maximum depth of the base decision tree.
* **estimator\_\_min\_samples\_split:** Minimum samples required to split a node.
* **estimator\_\_min\_samples\_leaf:** Minimum samples required at a leaf node.

**Cross-Validation:** A 2-fold cross-validation was utilized to evaluate the model's performance during hyperparameter tuning.

**RandomizedSearchCV:** This method sampled a set of hyperparameter configurations, running a total of 20 iterations to find the optimal parameters.

#### Evaluation Metrics

The performance of the AdaBoost model was assessed using several metrics, including:

* **Accuracy:** The proportion of correctly classified instances.
* **Precision:** The ratio of true positives to predicted positives, indicating the model's relevance in predictions.
* **Recall (Sensitivity):** The ratio of true positives to actual positives, highlighting the model's ability to detect relevant instances.
* **F1-Score:** The harmonic mean of precision and recall, useful for imbalanced datasets.
* **ROC AUC:** A measure of the model's ability to distinguish between classes, calculated by plotting the true positive rate against the false positive rate.

#### Results and Discussion

After hyperparameter tuning, the best AdaBoost model was achieved with the following parameters: n\_estimators=300, learning\_rate=0.2, estimator\_\_min\_samples\_split=10, estimator\_\_min\_samples\_leaf=5, and estimator\_\_max\_depth=7.

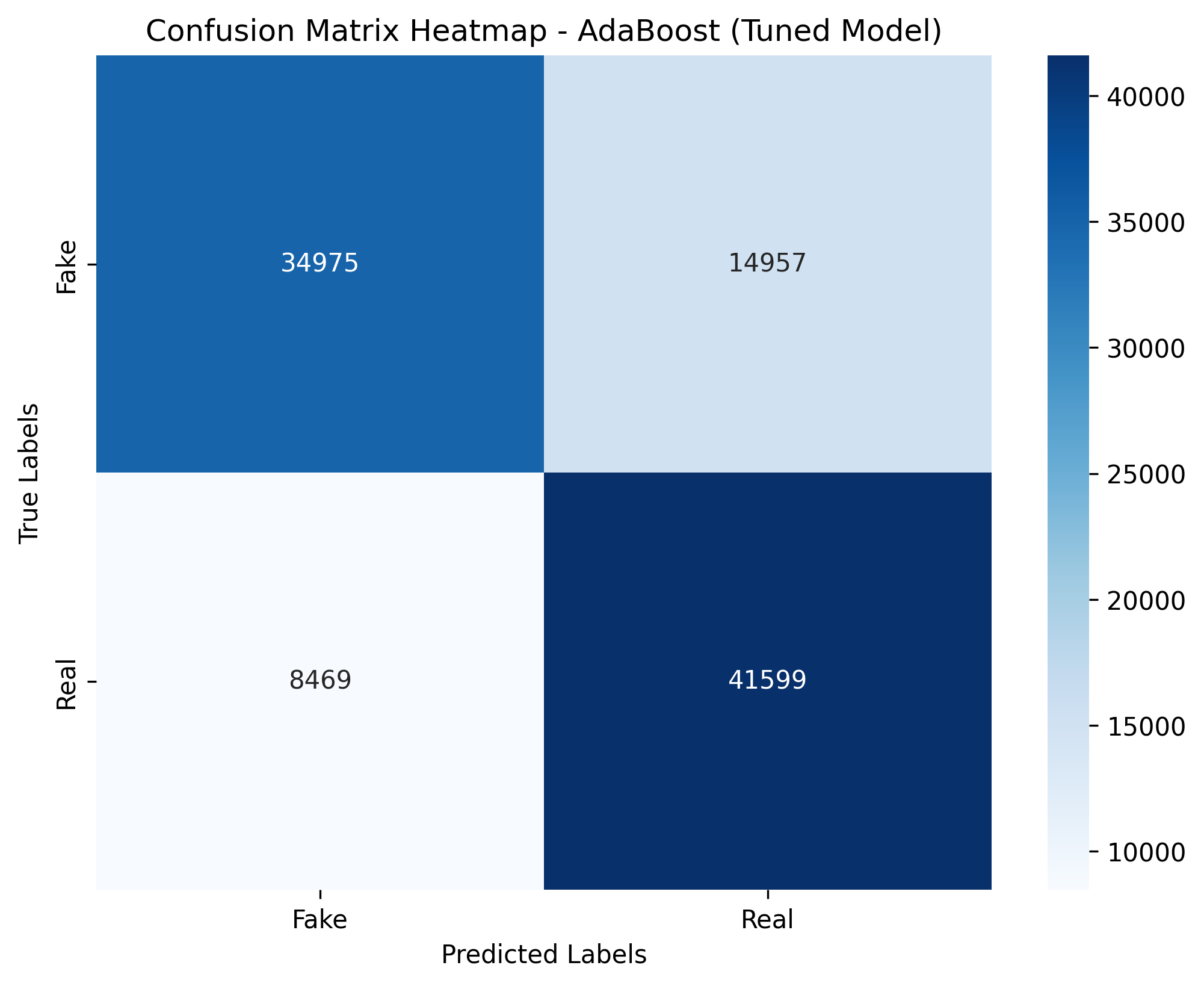
The model was trained and evaluated on the ADHD classification dataset, yielding the following results:

**Table 10:** Adaboost

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Best Number of Estimators | 300 |
| Best Learning Rate | 0.2 |
| Best Max Depth | 7 |
| Best Min Samples Split | 10 |
| Best Min Samples Leaf | 5 |
| Training Accuracy | 80.61% |
| Test Accuracy | 77.00% |
| Precision (Class 0) | 81% |
| Precision (Class 1) | 74% |
| Recall (Class 0) | 70% |
| Recall (Class 1) | 83% |
| F1-Score (Class 0) | 75% |
| F1-Score (Class 1) | 78% |
| ROC AUC Score | 0.8434 |

**Confusion Matrix**

The confusion matrix for the test set is as follows:



* **True Positives (TP)**: 41599 — correctly predicted Class 1.
* **True Negatives (TN)**: 34975— correctly predicted Class 0.
* **False Positives (FP)**: 14957— incorrectly predicted as Class 1.
* **False Negatives (FN)**: 8469— incorrectly predicted as Class0.

The ROC AUC score of 0.8434 signifies a strong capability for class discrimination.

#### Conclusion

The AdaBoost model for ADHD classification demonstrated commendable performance, as reflected in the accuracy, F1-scores, and ROC AUC score. The results indicate a reliable model for identifying relevant instances, particularly in emphasizing the harder-to-classify cases. Further enhancements can be pursued through additional hyperparameter tuning and feature engineering to boost performance. Additionally, exploring the integration of AdaBoost with other ensemble methods could yield even better results in subsequent analyses.

### 5.10 Guassian NB

Gaussian Naive Bayes (GNB) is a probabilistic classifier based on Bayes' theorem, which assumes that the features follow a Gaussian (normal) distribution. This model is particularly effective for binary and multiclass classification tasks when the features are continuous and normally distributed. Its simplicity, speed, and efficiency in handling large datasets make it a popular choice for text classification and other applications.

#### Gaussian Naive Bayes Construction

The Gaussian Naive Bayes classifier operates under the assumption that the features are conditionally independent given the class label. This assumption simplifies the computation of the conditional probabilities, allowing the model to estimate the likelihood of each feature belonging to a specific class.

**Key Components of Gaussian Naive Bayes:**

* **Class Prior Probabilities:** The probability of each class in the training dataset, calculated as the proportion of each class.
* **Likelihood:** The probability of observing the feature values given a class, modeled using a Gaussian distribution.
* **Posterior Probability:** The probability of each class given the feature values, computed using Bayes' theorem.

**Training and Prediction:** The Gaussian Naive Bayes model is trained using the training dataset, and predictions are made on the test dataset based on the learned parameters.

#### Evaluation Metrics

The performance of the Gaussian Naive Bayes model was assessed using several metrics, including:

* **Accuracy:** The proportion of correctly classified instances.
* **Precision:** The ratio of true positives to predicted positives, indicating the model's relevance in predictions.
* **Recall (Sensitivity):** The ratio of true positives to actual positives, highlighting the model's ability to detect relevant instances.
* **F1-Score:** The harmonic mean of precision and recall, useful for imbalanced datasets.
* **ROC AUC:** A measure of the model's ability to distinguish between classes, calculated by plotting the true positive rate against the false positive rate.

#### Results and Discussion

After training the Gaussian Naive Bayes model on the ADHD classification dataset, the following results were obtained:

**Table 11:** Guassian NB

|  |  |
| --- | --- |
| **Metric** | Value |
| Accuracy | 57.00% |
| Precision (Class 0) | 67% |
| Precision (Class 1) | 55% |
| Recall (Class 0) | 29% |
| Recall (Class 1) | 86% |
| F1-Score (Class 0) | 40% |
| F1-Score (Class 1) | 67% |
| ROC AUC Score | 0.5730 |

### **Confusion Matrix**

### The confusion matrix for the test set is as follows:

### 

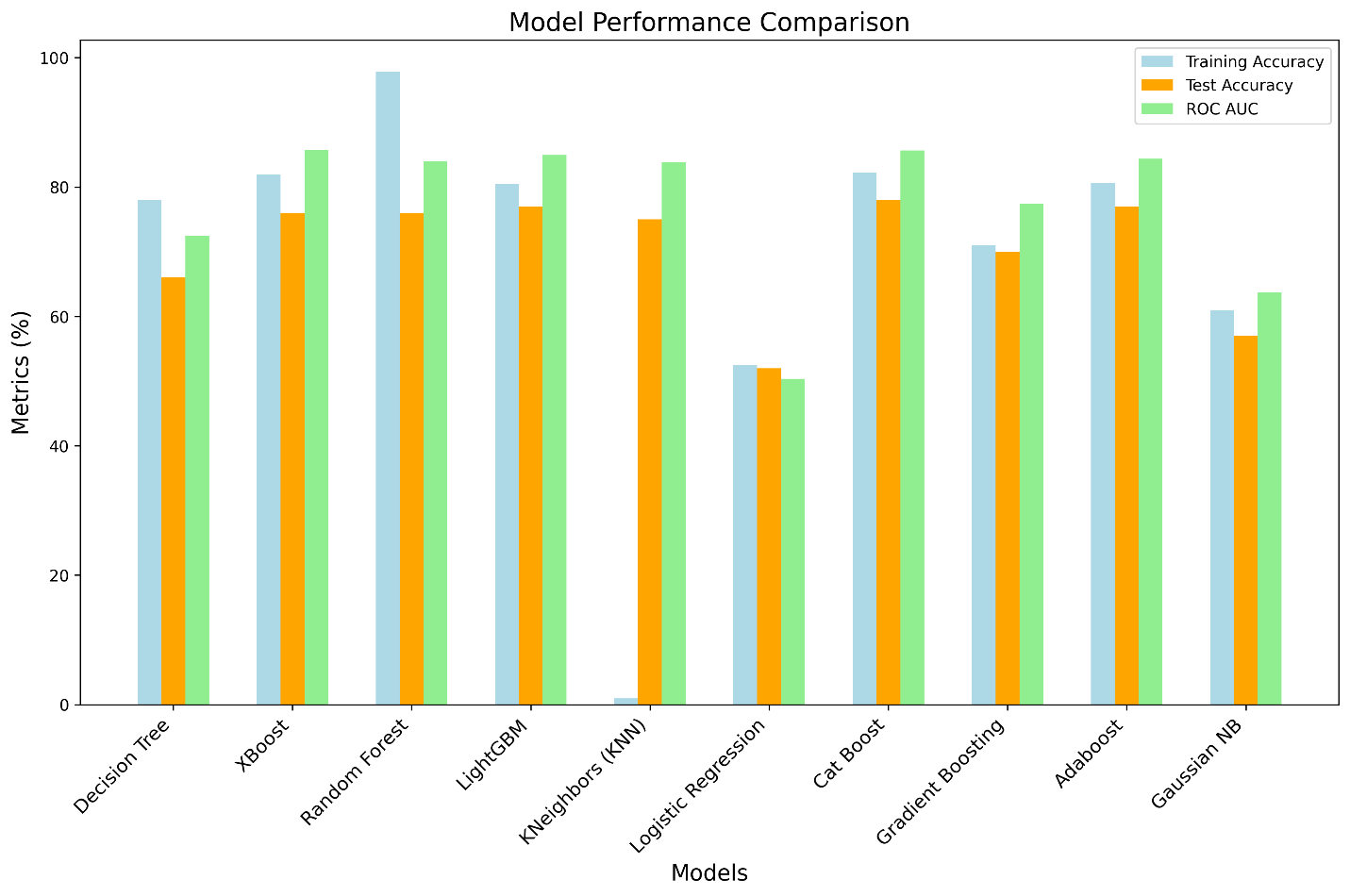
* **True Positives (TP)**: 42975 — correctly predicted Class 1.
* **True Negatives (TN)**: 14412— correctly predicted Class 0.
* **False Positives (FP)**: 35520— incorrectly predicted as Class 1.
* **False Negatives (FN)**: 7093— incorrectly predicted as Class0.

This results in an overall accuracy of only 57%, suggesting that the model is not well-suited for this particular task without further optimization or feature engineering.

#### Conclusion

The Gaussian Naive Bayes model for ADHD classification exhibited limited performance, particularly in terms of accuracy and the ability to classify non-ADHD instances effectively. While it performed reasonably well for identifying ADHD instances (high recall), the overall results highlighted the need for improvement in precision and overall classification performance. This indicates that while Gaussian Naive Bayes is a simple and fast classifier, its assumptions may not hold in this context, leading to suboptimal results. Further strategies, such as exploring feature transformations or combining with more complex models, may be necessary to enhance performance in future analyses.

### 5.4 Comparison of Model Performance



In the context of evaluating the ten different machine learning models for classification, a comparative analysis was conducted based on various performance metrics, including training accuracy, classification report statistics (precision, recall, F1-score), and ROC AUC scores. The results indicate a range of effectiveness across the models, highlighting the strengths and weaknesses inherent in each approach.

#### Best Performing Model

The **XGBoost** model emerged as the best-performing algorithm in this evaluation. It achieved a training accuracy of **81.92%** and a significantly high ROC AUC score of **0.8576**. The classification report for XGBoost reveals that it maintained a precision of **75%** and a recall of **83%** for class 1, demonstrating its proficiency in identifying positive instances. The balance between precision and recall is further exemplified by its F1-score of **79%**, indicating a well-rounded model that minimizes both false positives and false negatives effectively. The superior performance can be attributed to XGBoost's ability to handle complex relationships in data through its gradient boosting mechanism, making it highly suitable for this classification task.

#### Moderate Performing Model

The **LightGBM** model also performed commendably, with a training accuracy of **80.42%** and a ROC AUC score of **0.8493**. Similar to XGBoost, it exhibited a solid balance in its classification report, achieving a precision of **74%** and a recall of **83%** for class 1, leading to an F1-score of **78%**. These metrics indicate that LightGBM is a robust alternative, capable of yielding reliable results in this dataset, albeit slightly less effective than XGBoost.

#### Worst Performing Model

Conversely, the **Logistic Regression** model demonstrated the weakest performance, recording a training accuracy of merely **52.43%** and a concerning ROC AUC score of **0.5037**. The classification report highlights severe imbalances in predictive performance, with a precision of **52%** and a recall of **64%** for class 1, resulting in an F1-score of **58%**. This suboptimal performance can be attributed to Logistic Regression’s limitations in capturing non-linear relationships and interactions in the dataset, rendering it ineffective for this classification challenge.

#### Conclusion

In summary, the analysis reveals that XGBoost stands out as the most effective model for the given classification task, closely followed by LightGBM, which also exhibits commendable performance. In contrast, Logistic Regression falls significantly short in terms of predictive accuracy and reliability. This comparative assessment underscores the importance of selecting appropriate algorithms based on their capabilities to handle complex data patterns and achieve optimal classification results. Future work may focus on integrating the strengths of these models through ensemble techniques, potentially improving overall predictive performance further.