Medical Science

AIMS Medical Science, 11(2): 58–71.
DOI: 10.3934/medsci.2024005
Received: 27 January 2024
Revised: 11 April 2024
Accepted: 23 April 2024
Published: 07 May 2024

http://www.aimspress.com/journal/medicalScience

Research article

Enhancing cardiovascular disease prediction: A hybrid machine learning approach integrating oversampling and adaptive boosting techniques

Segun Akinola[1], Reddy Leelakrishna[2] and Vijayakumar Varadarajan[3,*]

[1] Johannesburg Business School, University of Johannesburg, ZA-2006, South Africa
[2] Department of Physics, University of Johannesburg, ZA-2006, South Africa
[3] Ajeenkya DY Patil University, Pune, Maharashtra, India

* **Correspondence:** Email: dean.international@adypu.edu.in.

Appendix 1 (All Algorithm)

Algorithm 1 SmoteR(D,tE,o.u,k)

1. Initialize empty sets:
    - rareHD to store instances with high relevance of heart disease (φ(y) > tE) and y = 1
    - rareNHD to store instances with high relevance of no heart disease (φ(y) > tE) and y = 0
    - newCasesHD to store synthetic cases for rareHD
    - newCasesNHD to store synthetic cases for rareNHD o newCases as the concatenation of newCasesHD and newCasesNHD o normCases as the set for under-sampling
2. For each instance (x, y) in D:

- o If φ(y) > tE and y = 1, add (x, y) to rareHD o If φ(y) > tE and y = 0, add (x, y) to rareNHD

3. Generate synthetic cases for rareHD: o newCasesHD = genSynthCases(rareHD, %o, k)
4. Generate synthetic cases for rareNHD:
   - o newCasesNHD = genSynthCases(rareNHD, %o, k)
5. Concatenate newCasesHD and newCasesNHD: o newCases = newCasesHD newCasesNHD
6. Determine the number of cases for under-sampling:
   - o nrNorm = %u of |newCases|
7. Randomly select cases from D{rareHD rareNHD} for under-sampling:
   - o normCases = random sample of nrNorm cases from D{rareHD rareNHD}
8. Return the concatenated synthetic and under-sampled cases:
   - o Return newCases normCases

end

Algorithm 2 for XGB

---

1. Import the essential libraries:
   - o pandas as pd o numpy as np o xgboost as xgb
   - o train_test_split from sklearn.model_selection o accuracy_score from sklearn.metrics
2. Load the heart disease dataset into a pandas DataFrame.
3. Pre-process the data:
   - o Perform data cleaning and handle missing values. o Conduct feature selection based on domain knowledge or statistical techniques.
   - o Normalize or standardize the features if necessary. o Divided the data into training with testing sets using train_test_split() function, considering stratification if needed.
4. Define the XGBoost model:
   - o Set the hyperparameters for the XGBoost model, such as the number of trees, learning rate, and maximum depth.
   - o Optionally, perform cross-validation or grid search for hyperparameter tuning.
5. Train the model: o Fit the XGBoost model using the training data.
6. Evaluate the model:
   - o Make predictions on the testing data using the trained model. o Compute evaluation metrics specific to heart disease prediction, such as accuracy, precision, recall, and F1-score. o Analyze the performance of the model and consider any issues, such as overfitting or underfitting.
7. Interpret the results:

- Investigate the importance of features in predicting heart disease. o Identify any patterns or relationships between features and the target variable.
- Consider the impact of individual features on the model's predictions.

8. Iterate and refine:
    - Based on the results, iterate and refine the model by adjusting hyperparameters, modifying feature engineering techniques, or exploring different algorithms.
    - Consider additional techniques like ensemble methods or addressing class imbalance if necessary.

end

Algorithm 3 for ET

1. Import the necessary libraries:
    - pandas as pd o numpy as np
    - ExtraTreesClassifier starting sklearn.ensemble o train_test_split after sklearn.model_selection o accuracy_score, confusion_matrix from sklearn.metrics
2. Load the heart disease dataset into a pandas DataFrame:
    - data = pd.read_csv('heart_disease_data.csv')
3. Pre-process the data:
    - Separate the features (X) from the target variable (y). o Splitting data's into training with testing sets use train_test_split() function, considering stratification if needed.
4. Define the Extra Trees model:
    - Initialize the ExtraTreesClassifier model.
5. Train the model: o Fit the Extra Trees model using the training data.
6. Make predictions: o Generate predictions for the testing data using the trained model.
7. Evaluate the model:
    - Compute the accurate model by comparing predicted labels with the actual labels.
    - Generate the confusion matrix to assess the outcome of the model.
8. Print the accuracy and confusion matrix:
    - Print the accuracy score.
    - Print the confusion matrix.

End

Algorithm 4 for RF

1. Import the necessary libraries:
   - pandas as pd o numpy as np
   - RandomForestClassifier after sklearn.ensemble o train_test_split after sklearn.model_selection o accuracy_score, confusion_matrix from sklearn.metrics
2. Load the heart disease dataset into a pandas DataFrame:
   - data = pd.read_csv('heart_disease_data.csv')
3. Preprocess the data:
   - Separate the features (X) from the target variable (y). o Divide the data into training with testing sets use train_test_split() function, considering stratification if needed.
4. Define the Random Forest model:
   - Initialize the RandomForestClassifier model.
5. Train the model: o Fit the Random Forest model using the training data.
6. Make predictions: o Generate predictions for the testing data using the trained model.
7. Evaluate the model:
   - Compute the accurate model through comparing the predicted labels with the actual labels.
   - Generate the confusion matrix to assess the performance of the model.
8. Print the accuracy and confusion matrix:
   - Print the accuracy score.
   - Print the confusion matrix.

   End

Algorithm 5 for AdaBoost

1. Input:
   - Training input data: (x1, y1), (x2, y2), ..., (xn, yn), where xi represents the features of the ith patient and yi is the corresponding label (+1 for heart disease present, -1 for heart disease not present). o Number of iterations: T
2. Output:
   - Boosted hypothesis for heart disease prediction: H(x)
3. Step 1: Initialize weights for training samples o Set D1(n) = 1/N for all n, where N is the total number of training samples.
4. Step 2: Iterate T times o For t = 1 to T:

- Train a weak learner ht(x) using the weighted training data.
- ht(x) predicts the presence or absence of heart disease (+1 or -1) based on the patient's features.

5. Step 3: Calculate weighted error rate and weight of the weak learner o Compute the weighted error rate εt using the equation: εt = Σ[n=1 to N] Dt(n) * 1{ht(xn) ≠ yn} / Σ[n=1 to N] Dt(n) o Calculate the weight αt for the weak learner using the equation: αt = 0.5 * ln((1 - εt) / εt)
6. Step 4: Update the weights of the training samples o For n = 1 to N:

   If ht(xn) = yn, then: Dt+1(n) = Dt(n) * exp(-αt)

   If ht(xn) ≠ yn, then: Dt+1(n) = Dt(n) * exp(αt)

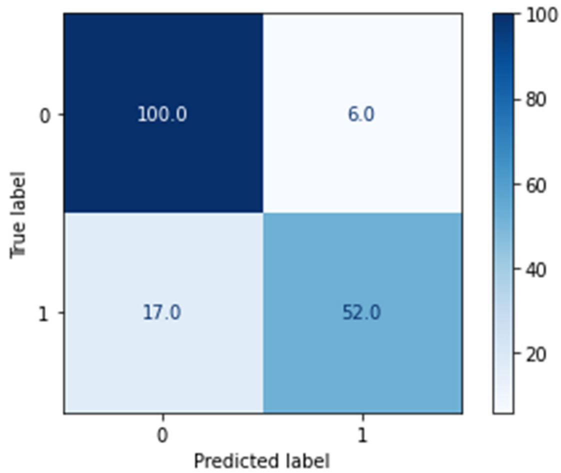
7. Step 5: Normalize the updated weights o Normalize the updated weights Dt+1(n) by dividing them by the sum of all updated weights: Dt+1(n) = Dt+1(n) / Σ[m=1 to N] Dt+1(m)
8. Step 6: Repeat steps 2-5 for T iterations.
9. Step 7: Combine weak classifiers to create the boosted hypothesis o For a new input sample x:

   Calculate the boosted hypothesis H(x) as: H(x) = sign (Σ [t=1 to T] αtht(x))

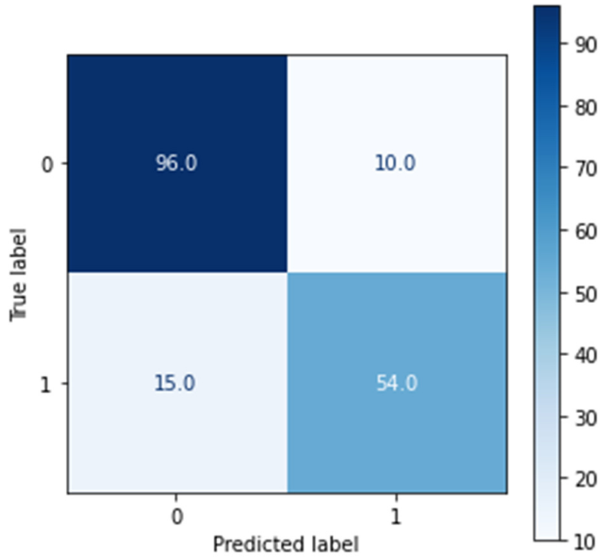
end

Appendix 2 (Validation)

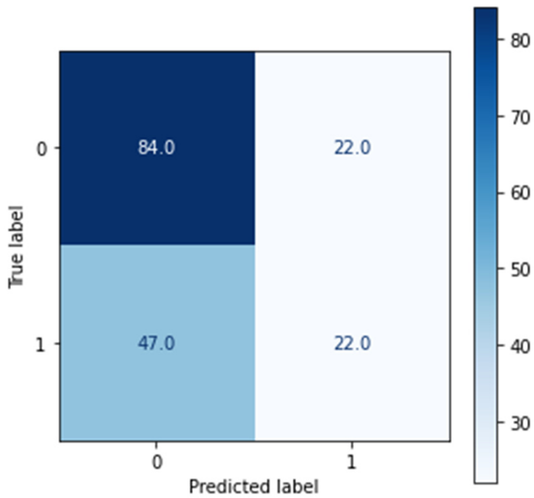
```
================================================
Random Forest Results
================================================
Accuracy [ with RF]: 86.85714285714286 %
Recall  [ with RF]: 75.36231884057972 %
precision [ with RF]: 89.65517241379311 %
MCC [ with RF]: 0.723628103990507
```



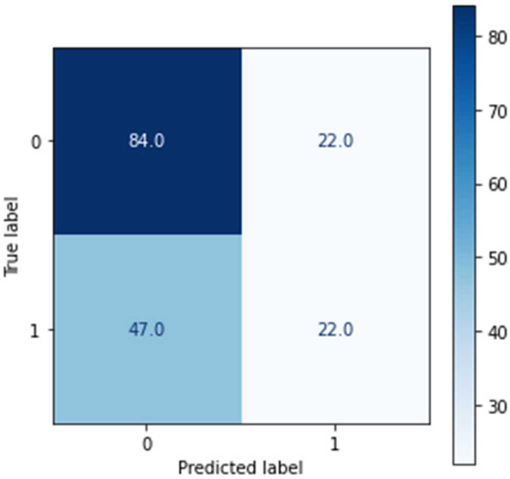

```
=====================================================
AdaBoost Random Forest Results
=====================================================
Accuracy [AdaBoost with RF]: 85.71428571428571 %
Recall [AdaBoost with RF]: 78.26086956521739 %
precision [AdaBoost with RF]: 84.375 %
MCC [ AdaBoost with RF]: 0.6983678802480235
```



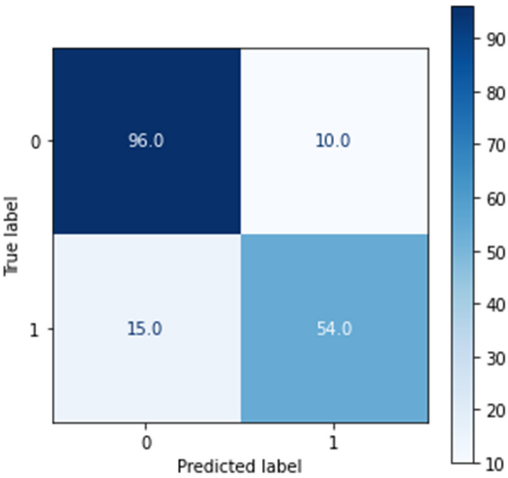

```
==========================================
Logistic Regression Results
==========================================
Accuracy [ with LR]: 60.57142857142858 %
Recall  [ with LR]: 31.88405797101449 %
precision [ with LR]: 50.0 %
MCC [ with LR]: 0.1253674928685844
```



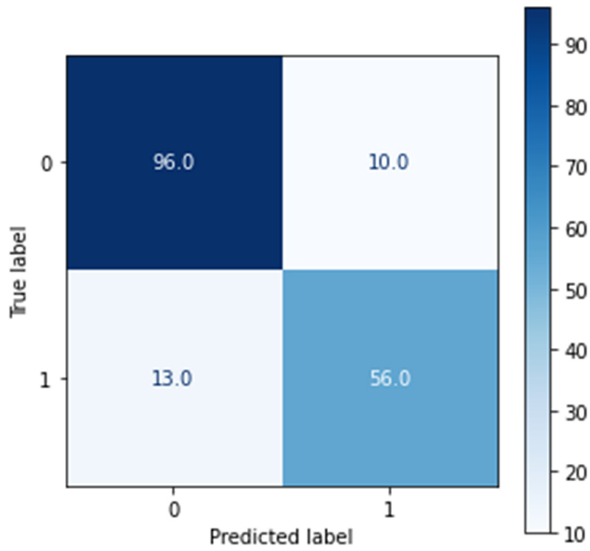

```
==================================================================
AdaBoost LR Results
==================================================================
Accuracy [AdaBoost with LogisticRegression]: 60.57142857142858 %
Recall [AdaBoost with LogisticRegression]: 31.88405797101449 %
precision [AdaBoost with LogisticRegression]: 50.0 %
==================================================================
MCC [ AdaBoost with LR]: 0.1253674928685844
```



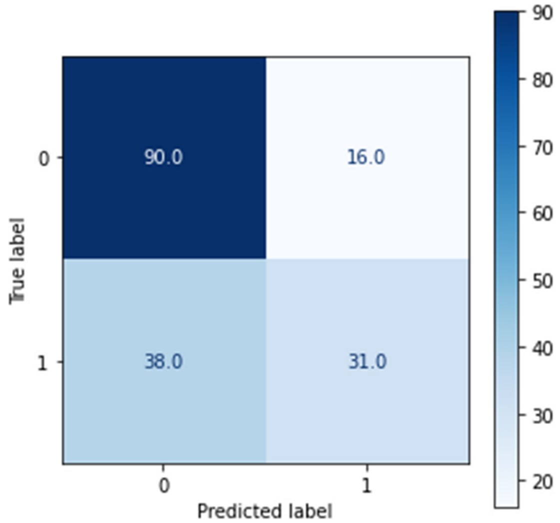

```
==========================================
Extra Tree Results
==========================================
Accuracy [ with ET]: 85.71428571428571 %
Recall  [ with ET]: 78.26086956521739 %
precision [ with ET]: 84.375 %
MCC [ with ET]: 0.6983678802480235
```



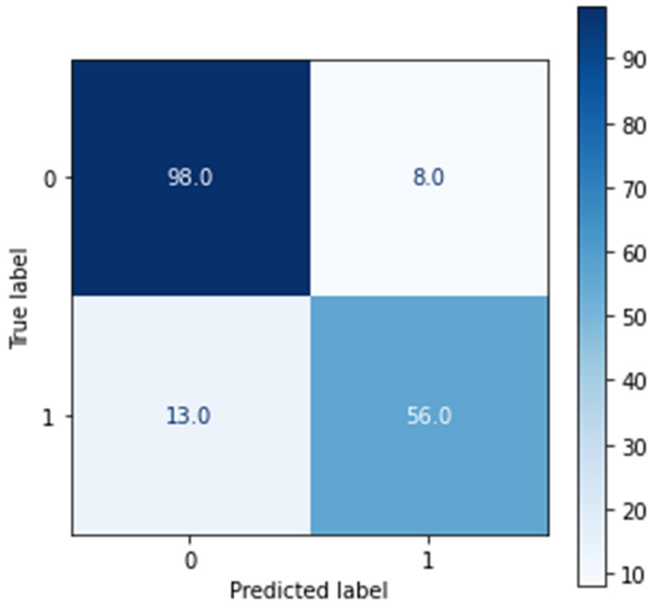

```
====================================================
Extra TreeForest Results
====================================================
Accuracy [AdaBoost with ET]: 86.85714285714286 %
Recall [AdaBoost with ET]: 81.15942028985508 %
precision [AdaBoost with ET]: 84.84848484848484 %
MCC [ AdaBoost with ET]: 0.7232119465299363
```




```
==========================================
XGB Results
==========================================
Accuracy [ with ET]: 69.14285714285714 %
Recall  [ with ET]: 44.927536231884055 %
precision [ with ET]: 65.95744680851064 %
MCC [ with ET]: 0.328945049232401
```




```
================================================
XGB TreeForest Results
================================================
Accuracy [AdaBoost with XGB]: 88.0 %
Recall [AdaBoost with XGB]: 81.15942028985508 %
precision [AdaBoost with XGB]: 87.5 %
MCC [ AdaBoost with XGB]: 0.7469234539641157
```


98.0
8.0
13.0
56.0
0
1
True label
0
1
Predicted label
90
80
70
60
50
40
30
20
10