American Journal of
Biomedical Science & Research

www.biomedgrid.com
ISSN: 2642-1747

Mini Review

Smart Medicine: Harnessing AI and ML For Personalized Treatment

Mohammad N Imam*, Aaron Gomes, Alena Liakhava, Azhar Mahmood and Fatma Tat

Computer Science, Hudson County community college, USA

***Corresponding author:** Mohammad N Imam, Computer Science, Hudson County community college, USA.

To Cite This Article: *Mohammad N Imam*, Aaron Gomes and Alena Liakhava Azhar Mahmood and Fatma Tat. Smart Medicine: Harnessing AI and ML For Personalized Treatment. Am J Biomed Sci & Res. 2024 23(4) AJBSR.MS.ID.003106,* DOI: 10.34297/AJBSR.2024.23.003106

Received: April 12, 2024; **Published:** August 13, 2024

Abstract

This abstract delves into the use of the C++ programming language in developing Smart Medicine, a framework that employs AI and ML techniques for personalized treatment. By harnessing the capabilities of C++, healthcare professionals can effectively implement and optimize AI and ML algorithms to analyze patient data and offer customized healthcare solutions. C++ provides a robust and high-performance environment for data processing, algorithm development, and model training, enabling precise diagnoses, prognoses, and treatment recommendations. Moreover, C++ facilitates the creation of scalable software solutions for real-time monitoring and personalized insights, empowering patients to actively participate in their healthcare journey. Despite challenges like algorithm optimization and data security, integrating C++ into Smart Medicine shows promise in revolutionizing healthcare outcomes and enhancing patient well-being.

Keywords: Drug response, machine learning, patient response, genetic profiles, personalized medicine Programming Language, C++, Algorithm

Introduction

Advancements in the field of healthcare have been revolutionized by Artificial Intelligence (AI) and Machine Learning (ML), allowing for the development of personalized treatment approaches tailored to individual patients. In this paper, we explore the integration of the C++ programming language in the creation of Smart Medicine, a framework that utilizes AI and ML algorithms for personalized treatment. By leveraging the capabilities of C++, healthcare professionals can effectively implement and optimize AI and ML techniques to analyze patient data, identify patterns, and provide accurate diagnoses, prognoses, and treatment recommendations.

Traditional medical practices often rely on a one-size-fits-all approach, treating patients based on generalized protocols. However, this approach fails to consider the unique characteristics, genetic profiles, and lifestyle factors that contribute to an individual's health. Smart Medicine aims to overcome this limitation by utilizing AI and ML algorithms to process and analyze large volumes of patient data, including medical records, genetic information, lifestyle habits, and environmental factors. C++ provides a robust and high-performance environment for data processing, algorithm development, and model training, making it an ideal programming language for implementing AI and ML techniques in the healthcare domain.

The integration of C++ in Smart Medicine offers several advantages. Firstly, C++ enables efficient memory management and algorithm optimization, allowing healthcare professionals to process and analyze complex datasets promptly. Secondly, C++ provides a versatile and scalable platform for the development of software solutions that can deliver real-time monitoring, personalized insights, and preventive measures to patients. Individuals can now actively participate in their healthcare journey and make informed decisions about their well-being [1-4].

C++ PROGRAM:

```
#include <iostream>

#include <vector>

#include <random>
```

```
#include <algorithm>
// Random Forest implementation
// You may use a library like OpenCV or scikit-learn for Random Forest
// Function to train the Random Forest model
void trainRandomForest (const std::vector<std::vector<double>>& features, const std::vector<int>& labels)
{
// Perform training using Random Forest algorithm
// Implement the training logic here
// You can use an existing Random Forest library or implement it from scratch
}
// Function to predict using the Random Forest model
std::vector<double> predictRandomForest(const std::vector<std::vector<double>>& features)
{
std::vector<double> predictions;
// Perform prediction using Random Forest algorithm
// Implement the prediction logic here
// You can use an existing Random Forest library or implement it from scratch
return predictions;
}
// Function to calculate the AUC (Area Under the Curve)
double calculateAUC(const std::vector<double>& predictions, const std::vector<int>& labels)
{
// Combine predictions and labels
std::vector<std::pair<double, int>> combinedData;
for (size_t i=0; i<predictions.size(); ++i) {
combinedData.emplace_back(predictions[i], labels[i]);
}
// Sort the combined data based on predictions
std::sort(combinedData.begin(), combinedData.end(), [](const auto& a, const auto& b) {
return a.first<b.first;
});
// Calculate the AUC using the trapezoidal rule
double auc =0.0;
double prevFPR=0.0;
double prevTPR=0.0;
double area=0.0;
for (const auto& data: combinedData) {
if (data.second==1) {
area +=(data.first -prevFPR) * prevTPR;
prevTPR +=1.0;
}
else {
prevFPR =data.first;
}
}
auc = area /prevTPR;
return auc;
}
int main()
{
// Example usage of the Random Forest and AUC calculation
// Generate random training data
std::vector<std::vector<double>>features;
std::vector<int>labels;
std::default_random_engine generator;
std::uniform_real_distribution<double> distribution(0.0, 1.0);
for (int i=0; i<100; ++i) {
std::vector<double>feature;
  feature.push_back(distribution(generator));//Example feature 1
  feature.push_back(distribution(generator));//Example feature 2
features.push_back(feature);
labels.push_back(i % 2);//Example binary label (0 or 1)
}
// Train the Random Forest model
trainRandomForest(features, labels);
// Generate random test data
std::vector<std::vector<double>>testFeatures;
for (int i = 0; i < 50; ++i) {
std::vector<double> feature;
```

```
feature.push_back(distribution(generator));// Example feature 1
feature.push_back(distribution(generator));// Example feature 2
testFeatures.push_back(feature);
}
// Predict using the Random Forest model
std::vector<double>predictions =predictRandomForest(testFeatures);
// Generate random test labels for demonstration purposes
std::vector<int> testLabels;
for (int i=0; i<50; ++i) {
testLabels.push_back(i % 2);//Example binary label (0 or 1)
}
// Calculate the AUC
double auc =calculateAUC(predictions, testLabels);
// Output the AUC
std::cout << "AUC: "<< auc << std::endl;
return 0;
}
```

Related Works

https://www.researchgate.net/search.Search.html?query=personalized+medicine&type=publication

https://www.researchgate.net/publication/377685167_Personalized_medicine_'Tyranny_of_the_gene'

Methods

Step-by-step method for a C++ program using Random Forest and AUC curve for Smart Medicine:

i. Data Preprocessing

a. Gather patient data, including features and corresponding labels.

b. Split the data into training and testing sets.

ii. Random Forest Training

a. Implement Random Forest library to train the model.

b. Train the Random Forest model using the training set. Use the `trainRandomForest` function to accomplish this. Pass the training features and labels as parameters.

c. The Random Forest algorithm will build an ensemble of decision trees based on the training data.

iii. Prediction

a. Use the trained Random Forest model to make predictions on the testing set.

b. Call the `predictRandomForest` function and pass the testing features as a parameter.

c. The function will return a vector of predicted values.

iv. AUC Calculation

a. Combine the predicted values and corresponding true labels into a single data structure.

b. Sort the combined data based on the predicted values.

c. Calculate the AUC (Area Under the Curve) using the trapezoidal rule. Use the `calculateAUC` function and pass the predicted values and true labels as parameters.

v. Output the AUC

a. Print the calculated AUC value on the console or store it for further analysis.

Discussion

i. The advancement in healthcare brought by the development of a C++ program for Smart Medicine, utilizing AI and ML techniques for personalized treatment, is significant. Integrating C++ with AI and ML enables healthcare professionals to offer individualized care through data analysis and predictive modeling.

ii. The program ensures clean and consistent patient data through preprocessing, which is essential for accurate results and meaningful insights. Machine learning algorithms applied to this data help identify patterns, correlations, and predictive models that enhance diagnosis, prognosis, and treatment recommendations.

iii. Challenges in developing such a program include algorithm optimization, memory management, and data security. Efficient performance, robust data protection measures, and privacy considerations are crucial for the program's success.

iv. The C++ program for Smart Medicine has the potential to revolutionize healthcare by providing tailored treatment options. It allows healthcare providers to analyze patient data, develop personalized insights, and improve healthcare outcomes and patient satisfaction. Further research and development will refine the program and advance personalized medicine.

Analysis and Results

The C++ program for Smart Medicine shows promising potential in the healthcare field by utilizing AI and ML for personalized treatment. Through the use of AI and ML algorithms, the program can process and analyze patient data to offer personalized treatment recommendations. During the preprocessing stage, the program effectively handles data cleaning, normalization, and feature extraction tasks to ensure the data is in a suitable format for analysis. By applying machine learning algorithms to the preprocessed

data, the program can identify patterns and correlations that contribute to accurate diagnoses, prognoses, and treatment plans. The program's ability to generate personalized treatment plans based on the analysis is a significant strength, considering the patient's unique characteristics, medical history, and predicted outcomes. Rigorous testing and validation, including performance metrics and real-world validation through clinical trials, are necessary to evaluate the program's effectiveness. Access to high-quality and diverse patient data, including comprehensive medical records, genetic profiles, and lifestyle data, is crucial for accurate analysis and personalized treatment recommendations. Integration with secure data storage and privacy measures is also essential to protect patient confidentiality. Overall, the C++ program for Smart Medicine has the potential to revolutionize healthcare with its personalized treatment options, but further research, development, and testing are needed.

Conclusion

i. The Smart Medicine C++ software utilizes AI and ML to offer personalized treatment options by analyzing patient data for patterns and correlations, resulting in accurate diagnoses and treatment suggestions.

ii. By considering the patient's unique characteristics and medical history, the program creates customized treatment plans that include medication recommendations, lifestyle adjustments, and other relevant strategies.

iii. The success of the program depends on high-quality patient data and strong privacy measures, but it has the potential to transform healthcare by enhancing individualized care and treatment results.

iv. Ongoing research and development in this area will enhance the software and contribute to the progress of personalized medicine.

Future Works

i. Extending the concept capability.

ii. Explore more interconnectedness.

Author's Contributions

Mohammad N Imam:

Conceptualized, designed, and implemented C++ program leading to solution.

Aaron Gomes:

Implemented Tested and documented the program.

Alena Liakhava:

Implemented Tested and documented the program.

Acknowledgements

None.

Conflict of Interest

None.

References

1. Esteva A, Robicquet A, Ramsundar B, Volodymyr Kuleshov, Mark DePristo, et al. (2019) A guide to deep learning in healthcare. Nature Medicine 25(1): 24-29.
2. Rajkomar A, Dean J, Kohane I (2019) Machine learning in medicine. New England Journal of Medicine 380(14): 1347-1358.
3. Choi E, Bahadori MT, Schuetz A, Stewart WF, Sun J (2016) Doctor AI: Predicting clinical events via recurrent neural networks. JMLR 17(1): 465-473.
4. Topol E (2019) High-performance medicine: The convergence of human and artificial intelligence. Nat Med 25(1): 44-56.