Detecting at-risk Heart Disease/Attack Patients with Supervised Learning

Adrian Lam[1], Christopher Chen[1], Rani Saro[1], Abdulrasol Alofi[1], Himmat Toor[1] and Isa Bukhari[1]

[1] Department of Computer Science, University of California - Davis

Abstract :

"Cardiovascular diseases (CVDs) are the leading cause of death globally, taking an estimated 17.9 million lives each year, with 85% of these deaths due to heart attacks and strokes" (World Health Organization). In response to this significant health issue, our project aims to create a machine learning model to accurately and quickly determine an individual's risk of heart disease. Utilizing supervised machine learning, specifically a neural network, we trained our model on 253,680 survey responses from the Heart Disease Health Indicators Dataset on Kaggle. The neural network identifies key biomarkers by modeling both linear and non-linear relationships using a brain-like structure. After processing the data through a series of neurons and activation functions, the model provides a binary prediction of heart disease and heart attack risk. This tool is designed to supplement health assessments and encourage individuals to seek professional medical advice if they are concerned about their heart health. By highlighting responses correlated with an increased risk, we hope to motivate users to consider lifestyle changes and facilitate informed discussions with their healthcare providers.

1 Introduction

1.1 Project Overview

The purpose of this project was to create a machine learning model that could quickly and accurately determine if an individual is at risk of heart disease. We utilize a neural network, a supervised learning machine learning model that is trained on 253,680 survey responses from the Heart Disease Health Indicators Dataset found on Kaggle. The model works by modeling linear/non-linear relationships in key biomarkers using a brain-like structure. After running the data through a series of neurons and activation functions, the model provides a binary prediction indicating the risk for heart disease/attack.

1.2 Our Objective

Our goal with this project is to provide supplemental information regarding an individual's risk of heart disease. We do not aim to take the place of a physician and regardless of the results of the survey, we encourage all individuals to consult a health professional if they are worried about their own risk of heart disease. The intended audience is anybody interested in learning more about their health and risk of heart disease. Since our model highlights responses that a user inputs that are correlated with an increased risk of heart disease, we hope our model helps individuals take a closer look at their lifestyles and maybe think about making changes based on their results. We also advocate for users who get an at-risk prediction to have a conversation with their doctor next time they go for an annual checkup.

1.3 Inspiration for the Project

Initially we considered creating a stock trading AI model because of our interest in the intersection between finance and technology. However, while exploring the different Kaggle datasets, we discovered other datasets that caught our attention. Among those were an eye disease classification dataset, a traffic signs dataset, a hair health dataset, and of course the heart disease health indicator dataset. We explored the importance of each topic to see how we could have the biggest impact and discovered that the leading cause of death worldwide for men and women was heart disease. Cardiovascular disease accounts for nearly a third of all deaths worldwide, it's a big problem so we hoped to create an accessible and accurate model that could help guide people worried about their risk of heart disease to have more information about their risks.

2 Background

2.1 Heart Disease/Heart Attack

Heart disease refers to various conditions affecting the heart, including coronary artery disease, heart failure, and arrhythmias. A heart attack, or myocardial infarction, occurs when blood flow to a part of the heart is blocked, causing damage to the heart muscle. Common risk factors include high blood pressure, high cholesterol, smoking, obesity, and a sedentary lifestyle.

2.2 Weighted Loss Function

A weighted loss function is a type of loss function that assigns different weights to different types of errors. This approach is often used when some types of errors are considered more costly or critical than others. By assigning higher weights to more significant errors, the model is penalized more for these errors, guiding it to focus on minimizing them during training.

2.3 Supervised Learning

Supervised learning is a type of machine learning where a model is trained on labeled data. This means that each training example includes input data and the corresponding correct output. The model learns to map inputs to outputs by minimizing the error between its predictions and the actual outcomes. Common algorithms include linear regression, decision trees, and neural networks.

2.4 Neural Network

A neural network is a machine learning model inspired by the human brain's structure. It consists of layers of interconnected nodes (neurons), each performing a linear or non-linear transformation on the input data. Neural networks are used to model complex patterns and relationships in data, making them suitable for tasks like image recognition and natural language processing.

3 Methods

3.1 Loading the dataset

We are able to preview the contents of the dataset using head - a function to display the first n data points (default n = 5). Looking closely at the dataset, the dataset contains 21 features excluding the target. Some values are consistent with values of either 0 or 1, denoting a binary flag of either true(1) or false(0) while other metrics cannot be measured by a binary flag like Age or BMI. In order to contextualize the non-binary values in relation to the rest of the data we need to utilize some normalization techniques that will appear in later sections. We'll additionally set the random seed to 42 to ensure reproducibility of the results.

3.2 Initial Data Analysis

To get a better understanding of how the features relate to the target we can plot a correlation matrix. A correlation matrix uses a method such as Pearson correlation coefficient to measure how two variables x, y scale linearly with one another. Techniques such as Logistic Regression rely on a strong linear relationship between a given feature and the target to accurately model their relationship and provide accurate predictions. Features with a higher correlation coefficient with the target are more likely to be important in predicting the target. For instance in our dataset, the feature High Blood Pressure (labeled HighBP) has a correlation coefficient of 0.21. Although this is a generally low correlation coefficient, relative to the other features in the dataset, it is the highest. Suggesting HighBP is a necessary component in predicting the target. To reduce the possibility of multicollinearity, we can remove features that have a high correlation with one another. Additionally, we can minimize overfitting by removing features that have a low correlation with the target and are less likely to be important in predicting the target. Finally, we can try to minimize subjective features as much as possible to reduce bias or extra noise in the dataset. Some immediate conclusions we can draw from the correlation matrix: Correlations between features and the target are generally low, many overlapping features exist, and there exist some negative correlations between the feature and the target such as Income or Education.

3.3 Data cleaning and preprocessing

After our initial data analysis, we can remove features characterized by a low correlation with the target, features that have a high correlation with one another, and features that are subjective. We can also remove any missing values in the dataset. We can also normalize the data to ensure that all features are on the same scale. Normalization is important because it ensures that each feature contributes equally to the model. For instance, if we have a feature that ranges from 0 to 1000 and another feature that ranges from 0 to 1, the model will give more weight to the feature that ranges from 0 to 1000. Normalization ensures that the model does not give more weight to a feature simply because it has a larger range. Normalization also ensures features are contextualized with respect to other data observations. We'll utilize a method known as Min-Max scaling to normalize the data. Min-Max scaling scales the data to a fixed range, usually between 0 and 1. This is done by subtracting the minimum value of the feature and dividing by the range of the feature.

3.4 Final Data Analysis

Now, after normalizing features and removing unnecessary features, we can re-verify the validity of our data by replotting the correlation matrix. Although some features still share some correlation, through reasoning we can determine some features are inherently correlated even though they are somewhat independent. For instance, the features HighBP and Cholesterol are correlated because HighBP can be caused by high cholesterol. But removing one of these features would only make our model more inaccurate as they are both independent features with importance to the overall prediction.

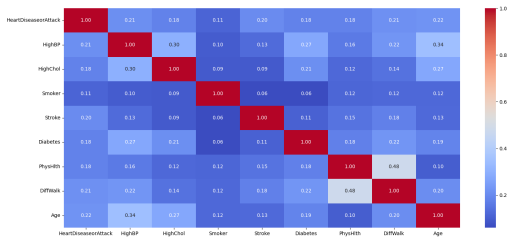

Figure 1. Pearson correlation matrix after data preprocessing

3.5 Data Splitting and Output Label Encoding

After cleaning and preprocessing the data, we can split the data into training and testing sets. We can then encode the target labels to ensure that the model can understand the target labels. We can use a method known as Label Encoding to convert the target labels into a numerical format. Label Encoding is a technique used to convert categorical data into numerical data. We can then train the model on the training data and evaluate the model on the testing data. We'll use a test size of 0.2 to ensure that 20% of the data is used for testing and 80% of the data is used for training. We'll also set the random seed to our predefined value of 42 to ensure reproducibility of the results.

3.6 Training Multiple Classification Models

After splitting the data and encoding the target labels, we can train multiple classification models on the training data. We can use models such as Logistic Regression, Random Forest, and Support Vector Machine to make predictions on the testing data. We can then evaluate the performance of each model using metrics such as accuracy, precision, recall, and F1 score. These metrics provide a measure of how well the model is performing on the testing data. For instance, accuracy measures the percentage of correct predictions made by the model, precision measures the percentage of true positive predictions out of all positive predictions made by the model, while recall measures the percentage of true positive predictions out of all actual positive instances, and F1 score is the harmonic mean of precision and recall. We can use these metrics to determine which model is performing the best on the testing data and make predictions on new data based on the best performing model. Notice we are using an extra parameter "class_weights" in the model training to account for the imbalanced nature of the dataset. This is because the dataset contains more negative samples than positive samples. By using class weights, we can give more importance to the positive samples and ensure that the model is not biased towards the negative samples. In our specific application, we want to try to minimize false negatives while maximizing accuracy. You can read more about why in the discussion section.

3.7 Defining a weighted binary cross-entropy loss function

We can define a weighted binary cross-entropy loss function to account for the imbalanced nature of the dataset. The weighted binary cross-entropy loss function assigns different weights to the positive and negative samples in the dataset. This allows the model to give more importance to the positive samples and ensure that the model is not biased towards the negative samples. Specifically, we'll use the following function when training the Neural Networks in later sections.

3.8 Defining a series of Neural Networks

A library known as Tensorflow simplifies the process of creating Neural Networks. Neural Networks vary in performance based on numerous factors such as the number of layers, the number of neurons in each layer, the activation function, the optimizer, and dropout. First, we can define a set of multiple neural networks - focusing on the number of layers and neurons in each layer in addition to the dropout. For classification tasks, normally activations such as ReLU are used in hidden layers while the sigmoid activation is used in the output layer. The sigmoid activation function is used in the output layer because it outputs a value between 0 and 1, which is suitable for binary classification tasks. ReLU is used in hidden layers because it is computationally efficient and helps the model learn complex patterns in the data. Dropout is used to prevent overfitting by randomly setting a fraction of the input units to 0 at each update during training. This helps the model generalize better to new data and improve performance on the testing data.

3.9 Using Grid Search to find the best Neural Network

We can train the neural networks on the training data and evaluate the performance of each neural network on the testing data using constants for the number of epochs, batch size, and validation split. In the context of training neural networks, an epoch is one complete pass through the training data, batch size is the number of samples processed before the model is updated, and validation split is the fraction of the training data to be used as validation data. All these are known as hyperparameters and can be tuned to improve the performance of the model. We can use a technique known as Grid Search to find the best hyperparameters for the neural networks. Grid Search is a technique used to find the best hyperparameters for a model by searching through a grid of hyperparameters and evaluating the performance of the model on the testing data. Since we defined 11 hyperparameters for each of the 5 neural networks, we'll need to tune 48 hyperparameters in total. So we'll need to train a total of 240 neural network models to find the best hyperparameters. We can then use the best hyperparameters to train the final neural network model and make predictions on new data.

3.10 Evaluating Model Performance

After training the neural networks with hyper-parameter tuning, we can evaluate the performance using metrics such as accuracy, precision, recall, and F1 score. Remember, although some models might have a higher accuracy, we need to consider the other metrics to ensure the model is performing well on the testing data. We can visualize the performance of the model using a confusion matrix. A confusion matrix is a table that is often used to describe the per-

formance of a classification model on a set of data for which the true values are known. The confusion matrix consists of four values: true positive, true negative, false positive, and false negative. We can use these values to calculate metrics such as accuracy, precision, recall, and F1 score. We can also visualize the confusion matrix using a heatmap to get a better understanding of how the model is performing on the testing data. We can then use these metrics to determine if the model is performing well on the testing data and make predictions on new data based on the best performing model.

3.11 Finding the best model

First, we define a custom metric to penalize false positives (FP) and false negatives (FN) more severely by squaring these values: 'Custom Metric = alpha * (FP^2) + beta * (FN^2)'. This formula allows us to adjust the importance of false positives and false negatives using weights 'alpha' and 'beta'. Next, we implement the calculation of this custom metric. We define a function 'calculate_custom_metric' that computes the custom metric based on the confusion matrix of each model. Then, we evaluate all models using the custom metric with specified weights ('alpha = 0.0005' and 'beta = 0.1'). This involves computing the custom metric for each model's confusion matrix. Finally, we select the best model by identifying the one with the lowest custom metric value. We use the following approach: 'min(res, key=lambda x: calculate_custom_metric(res[x]['confusion_matrix'], 0.0005, 0.1))'. Here, we ensure the selected model minimizes false positives and false negatives, with a higher emphasis on minimizing false negatives.

3.12 Ineffectiveness of SMOTEENN due to dataset noise

We attempted to use the SMOTEENN technique to balance the dataset. However, the technique was ineffective due to the presence of noise in the dataset. SMOTEENN is a combination of the SMOTE (Synthetic Minority Oversampling Technique) and Edited Nearest Neighbors (ENN) techniques. SMOTE generates synthetic samples for the minority class, while ENN removes noisy samples from the dataset. However, in our case, the dataset contains noise that cannot be effectively removed by ENN. As a result, the SMOTEENN technique was unable to balance the dataset effectively. Therefore, we need to explore other techniques to balance the dataset and improve the performance of the models.

3.13 Oversampling using SMOTE

We can use a technique known as Synthetic Minority Oversampling Technique (SMOTE) to oversample the positive samples in the dataset. SMOTE is a technique used to balance the class distribution in a dataset by generating synthetic samples of the minority class. This helps improve the performance of the model on the testing data by giving more importance to the positive samples. We can then train the model on the oversampled data and evaluate the performance of the model on the testing data. We can use the same metrics as before to evaluate the performance of the model and make predictions on new data based on the best performing model. In theory, by oversampling the positive samples, we can improve the performance of the model on the testing data and make more accurate predictions on new data.

3.14 Training Multiple Neural Networks with SMOTE

After oversampling the positive samples in the dataset, we can train multiple neural networks on the oversampled data. Notice how we no longer need the class weights since the dataset is now balanced meaning the model will not be biased towards the negative samples.

3.15 Using SMOTE with Weighted Binary Cross-Entropy Loss Function

For experimental purposes, we can combine the use of SMOTE with the weighted binary cross-entropy loss function to see if we can further improve the performance of the model. By using both techniques, we can give more importance to the positive samples and ensure that the model is not biased towards the negative samples.

4 Results

4.1 Model Accuracy and Performance

Our Decision Tree Classifier achieved an accuracy range of 85.1% to 85.3%. While this accuracy is promising, it does not fully represent the model's effectiveness given the imbalanced nature of the dataset (90% negative and 10% positive for heart disease). Realizing the limitation of accuracy, we evaluated the model using additional metrics such as precision, recall, and F1 score. These metrics provided a more nuanced view of the model's performance, particularly in handling the imbalanced dataset. However, the most critical aspect was the model's ability to correctly identify positive cases (high recall), which improved significantly after balancing the dataset with SMOTE. Our final model, Neural Network 4 with Dropout (16-10-0.3), demonstrated superior performance and robustness compared to other models.

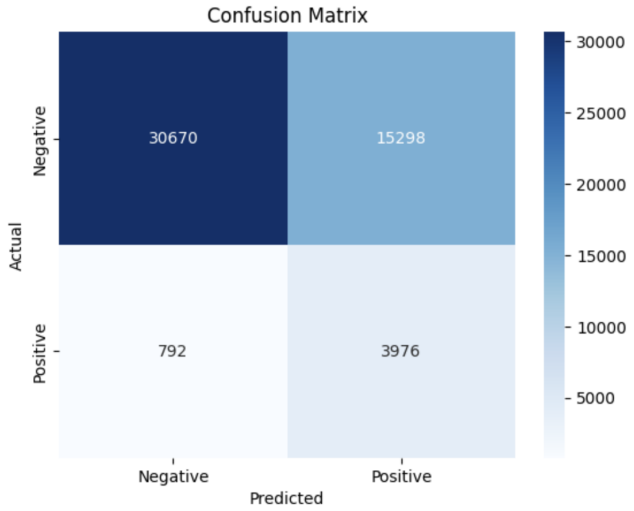

Figure 2. The confusion matrix plot of the chosen model

4.2 Overall

In summary, our project successfully achieved its primary goal of developing a machine learning model to predict heart disease risk. While the initial model's accuracy was promising, the real success lay in our iterative improvements, particularly in addressing dataset imbalance and refining our evaluation metrics. By focusing on metrics beyond accuracy and ensuring our model was interpretable and actionable for users, we created a tool that could significantly impact public health awareness and individual health outcomes. Future work will focus on further enhancing model accuracy, expanding the dataset, and integrating real-time health data for more dynamic predictions.

5 Discussion

5.1 Choosing the performance testing metrics

Getting into this project, we overvalued the importance of using accuracy as a metric to test the results of our AI model. We were under the false implication that the higher the accuracy of our model, the better it is suited to predict heart attack/disease based on input health related parameters for an individual. Using accuracy as the sole testing metric for our project, we began training different models like Decision Tree Classifier, Random Forest, etc to check which model gave us the highest value for accuracy. We set a goal to achieve 95% testing accuracy, and saw progression toward this goal as Decision Tree Classifier gave us an accuracy of 85%, and the Random Forest model gave us an accuracy of 90%. Even though we were satisfied with the results we realized that we should change our focus to maximizing the percentage of true positives for a health care based AI model. We want to minimize the cases where the model predicts that the person does not have a chance of heart attack or heart disease, when the person may actually be at risk. It is much better to use a metric like Recall to test the performance of our AI Model.

5.2 Significance of the results

Analyzing the significance of our results, we see that the recall score of true positives for our model is 83% while the accuracy of the model is 67%. The recall score tells us how good the model is at predicting the percentage of true positive results. The low accuracy means there might be more cases where we falsely predict a person has a chance of heart attack or heart disease when in reality they might not be at risk. We think this is a valid sacrifice to make as it is better to show more caution in our predictions, and a person may still choose to get further diagnosis through a doctor.

5.3 Lessons Learned

The reason why we might not be getting higher scores for some of our performance metrics may be caused by the imbalanced dataset that we chose to use. The whole foundation of the AI Model is based on the choice of dataset, and as we did not consider this as an issue early in the project, we were limited by the scope of improvements we could make. The original kaggle dataset had 90% negative results and 10% positive results for heart attack/disease. This makes it harder to train a dataset to accurately predict positive results due the weight of negative results in training the prediction model. It would be better to do more research in picking a more balanced dataset to train the model for its future iterations.

6 Conclusion

We were able to achieve our goal of creating a machine learning model to determine if the user is at risk of heart disease. This accomplishment was made possible through rigorously experimenting with multiple models and leveraging our knowledge of neural networks. Despite the challenges posed by the imbalanced data from the Kaggle dataset we were able to overcome this obstacle through applying appropriate weights and fine-tuning our hyperparameters. These efforts ensured that our model would minimize false positives and false negatives. A higher emphasis was placed on reducing the number of false negatives as they can lead to missed opportunities for early intervention. Although this model was built with the user's best interests in mind, it is still important to note that this model is not a substitute for professional medical advice. Rather, it should be used as a supplemental resource in conjunction with advice from their doctor and healthcare professionals. We hope that this tool will have a positive impact in reducing the devastating effects of heart disease and encourage users to have informed conversations with their healthcare providers.

Contributions

- **Himmat** - Worked on creating the frontend web page for users to complete the survey, to get the heart attack/disease risk prediction from our AI model. Specifically worked on the results page to display alternating results based on the prediction of the model.
- **Christopher** - Designed the backend application, setting up the http requests to communicate with the front end effectively using FastAPI. Utilized a dataframe to scalar the nonbinary inputs and fed the binary inputs into the model and sent the results in json format to the frontend.
- **Isa** - Created some visuals for the report describing the performance of the model such as a heatmap of the confusion matrix and model structure diagram. Assisted with the backend application implementation using FastAPI.
- **Adrian** - Created the Figma Prototype designs for the front end. I worked on creating the frontend web page for users specifically the survey page to help the users input their own health information for the AI model.
- **Rani** - Developed the front-end application - implemented the landing and survey page and connected the frontend to the backend. Created and trained an initial random forest model during data/model exploration.
- **Abdulrasol Alofi** - Implemented data analysis, data cleaning/preprocessing, generic classification models (Random Forest, Logistic Regression, etc), neural networks. Additionally, made contributions to the front-end and back-end applications - specifically for demo/fullstack.

Acknowledgements

This research received support during the *ECS 170 - Intro. to Artificial Intelligence* course, instructed by Professor Gabe Simmons, MS at the Department of Computer Science, University of California - Davis.