Predicting Cardiovascular Disease: A Comprehensive Approach

Jackie Liang, Sudharshan Palaniyappan, Thanush Lingeswaran, Divnoor Chatha

Introduction

How can we predict someone's chances of having a cardiovascular disease? We have created a program which predicts the chances of the user having cardiovascular disease using multiple given data points. The user should know a few important pieces of information about themselves. For example, their age, weight, cholesterol levels, and if they smoke or not etc. The reason why we decided to choose this topic is because heart disease is the leading cause of death in men, women and many racial and ethnic groups (CDC). It's so dangerous that the Centre of Disease Control and Prevention states that "One person dies every 33 seconds in the United States from cardiovascular disease" (CDC). We want to help individuals by using our program to predict their chances of having heart disease by comparing with patients who have been diagnosed by a professional, to potentially save their lives. We know this is important because early diagnosis is extremely important to help control the disease, as "Four out of five people don't know they have heart failure until their symptoms are severe enough to put them in the emergency room" (Booth). Early detection is the difference between life or death as late detection might not be maintainable, and the patient's quality of life will drop massively. This is why we need to focus on this issue, as early detection of the disease will save lives. Overall, the importance of this issue is what strongly motivated us to create such a project, to help detect these issues earlier on. Additionally, it is critical to note that our program should not be mistaken for professional medical advice. To help stress this point, we included an agreement page at the start of the program stating all of the terms and conditions of the program which the user must agree with before proceeding any further. This helps address any ethical issues which may arise as well, as now the program clearly indicates that it should not be mistaken as a professional medical advice tool.

All Datasets

Cardiovascular Disease dataset: The dataset we will use from Kaggle consists of 70,000 entries of patient data we will use to create our model. Each of the following categories is a column in the dataset:

1. Age (int (days))
2. Height (int (cm))
3. Weight (float (kg))

4. Gender (1: male, 2: female)
5. Systolic blood pressure (int)
6. Diastolic blood pressure (int)
7. Cholesterol (1: normal, 2: above normal, 3: well above normal)
8. Glucose (1: normal, 2: above normal, 3: well above normal)
9. Smoking (binary)
10. Alcohol intake (binary)
11. Physical activity (binary)
12. Presence or absence of cardiovascular disease (binary)

Computational Overview

Data Filtration/Aggregation/Transformation

To make our data analysis simpler, we opted to use the pandas library in Python. The pandas library is incredibly efficient at computing large datasets by converting csv files into pandas DataFrames, which are much faster for computations in comparison to if we used Python lists. Through the DataAnalysis class, we were able to do most of the cleaning necessary to get a nice dataset to work with in order to build our model. Starting off, we checked for any missing values and duplicates. Our dataset turned out to have none of these issues, but in the event this was not the case, we had functions handle_missing_data and handle_duplicate_rows to promptly address any discrepancies.

Next, we moved on to addressing issues surrounding invalid data dealing with the columns diastolic and systolic pressure (ap_hi ap_lo). From a biological perspective in a normal cardiovascular system, the pressure in the arteries is never higher during the relaxation phase (diastole) than it is during the contraction phase (systole) so when it is higher, this typically represents an abnormality in your body, like severe hypertension. As a result, including these data points could bring noise into the dataset which would affect our model's reliability/performance and lead to potentially misleading outcomes. So, in order to train our model on more relevant and accurate data, we decided to remove these columns from the data frame. Additionally, we removed outliers from the weight and height columns (via capping the columns between the first and ninety-ninth percentile) in order to prevent the skewing of model results and promote generalization.

Moving forward, other preprocessing steps were essential to simplify some of the input columns we had to work with. This involved changing the age column to years for readability, mapping gender (originally 1 - women, 2 - men) to the transformed (0 - women, 1 - men) for consistency throughout the dataset in terms of binary variables, renaming ap_hi and ap_lo to and the addition of the BMI column (calculated by using person's weight in kilograms and dividing by the square of their height in meters). The inclusion of the BMI column felt necessary as our external research regarding other variables which influence cardio-vascular symptoms showed that a high body mass index was a leading sign of cardio-vascular disease(Held et al.). We also decided to drop the id column in

this preprocessing phase as the column does not provide any meaningful information for analysis or modelling purposes and simply serves as a way to number the unique data points.

Originally, we decided to use linear regression to analyze the relationships between the target variable and the other variables, but we felt this was somewhat redundant after learning about heat maps, which display the relationship between variables quite succinctly and visually. The heat map was displayed using the help of the seaborn library in Python used for data analysis/visualization. From this visual analysis, it was clear that the 3 factors which had the most significant effect on our target were age, cholesterol level, and weight and the 3 factors which had the least significant effect on our target were gender, alcohol intake, and physical activity.

Model Building

For our model-building process, we made the PredictionModel class, which allows us to build, train, and evaluate a decision tree classifier for predicting cardiovascular disease. This was done with the help of the sci-kit library in Python which is an open source library primarily used for machine learning.

Starting with splitting the data, we learned that it is generally good practice to have 60 percent of the data reserved for training, 20 percent for validation, and 20 percent for testing, so this is how we decided the proportion to use when separating the data. Furthermore, setting a random_state parameter (arbitrarily chosen as 56) was essential to ensure reproducibility within the model. If this step had not been taken, the model performance would vary every time we ran it, making the results hard to properly analyze and compare.

After training the model, we make predictions on the validation and test sets and calculate the accuracy of these predictions using the accuracy_score function from scikit-learn. This allows us to evaluate the performance of the model on both the validation and test sets separately. Additionally, we perform cross-validation on the training set using the cross_val_score function to assess the model's performance more rigorously. This helps us ensure that the model's performance is consistent across different subsets of the training data. If the model's performance was not consistent, this suggests the presence of overfitting. To analyze the results of the testing phase, we printed the validation accuracy, testing accuracy, training accuracy, cross-validation scores, the mean cross-validation accuracy, the standard deviation of the cross-validation accuracy, and the classification report for the training set. The classification report in particular provides a summary of the precision, recall, F1-score, and support for each class in the training set, giving us a more detailed understanding of the model's performance.

During the initial iterations of the model, we were getting quite disappointing results, with the test accuracy around the high 50s to low 60s. However, this was expected as we had done nothing to improve the model's performance, so this was a baseline for us to compare against as we made enhancements. Our first step for improvement was trying to adjust the class_weights parameter in DecisionTreeClassifier to see if it would produce better results. This parameter allows us to give more weight to certain classes in the target variable (ex: putting more weight on non-physically active people in comparison to physically active people), which can be useful when dealing with imbalanced datasets. Our dataset had quite large imbalances with some of the binary variables, such as smoke, active, and gender so we hoped to reduce the imbalance to improve the accuracy. However, adjustments in the class_weights parameters did not lead to significant improvements in

model performance. This lack of improvement can likely be attributed to either the variables not being significant for the decision tree's performance or to the inherent capability of decision trees to effectively handle imbalanced datasets.

Moving on, we decided to remove certain input columns to see if they would affect the accuracy. From our data analysis stage, it was clear that gender, alcohol intake, and physical activity had little to no correlation to our target variable so we decided to remove these columns from the dataset. By removing these irrelevant or weakly correlated features, we aimed to simplify the model and potentially improve its performance by reducing noise and focusing on more impactful variables. The result showed a small increase in accuracy scores (2-3 percent), which was encouraging. Additionally, we tried different combinations of input columns, essentially performing manual backward selection which led to similar results. However, removing more than 1 random column at once resulted in a drastic decrease in performance, suggesting that most columns contained valuable information for the model. This method was likely the most tedious part of the model-building process.

Note: In real life, men are more prone to developing cardiovascular disease(Mosca et al.). Our dataset somewhat reflects this ratio of males and females with cardiovascular disease, showcasing the real-world distribution of gender. The inclusion of gender does not significantly impact the performance of our model. Overall, the positives outweigh the negatives, so we decided to include gender in our model, which also plays a key component in our visualizations.

In another attempt to improve model performance, we made the input columns proportionally equivalent, scaling all of them from a range of 0 to 1. This process, known as normalization, is used to ensure that every input column contributes equally to the model, with no single column dominating all the others. Unfortunately, this was another method for improvement which led to either an accuracy decrease or no notable increase, implying that certain columns were already in a suitable range and did not need to be normalized. However, there were columns like age, weight, ap_hi, ap_lo which were not in the same range so it made sense to normalize them from our perspective, implying that there may have been another reason why normalization was not appropriate in this case.

Finally, to account for the overfitting of the model, we had to adjust the max_depth parameter of the DecisionTreeClassifier to find the optimal depth that would prevent overfitting. Our current model was overfitting greatly, with the training accuracy being far better than the testing/validation, so this was a necessary step to take in order to address the overfitting. This concern led to the find_best_depth function being created to determine the optimal max_depth for the decision tree model. The function works by iterating through a range (1 -¿ user int input) of max_depth values calculating the validation accuracy, and storing the best accuracy and the depth which produces the best accuracy. Limiting the max_depth of the model using this function improved the testing accuracy by around 10 percent. However, one limitation of using this function is that it solely relies on the accuracy of the validation set. So, for further analysis and improvement, we made the max_depth_errors function. This function allowed us to analyze the training and validation errors of the Decision Tree model for different max_depth values. By iterating over a list of max_depth values, the function trained the model on the training set and calculated both the training and validation errors. The training error was computed as 1 - model.score(x_train, y_train), and the validation error was calculated similarly. This analysis was crucial in identifying the max_depth value that balanced the bias-variance trade-off, leading to better generalization on unseen data

(in other words, the test set).

Graphical User Interface

For our GUI, we used the Python library tkinter to create an interactive experience for the user. This is created under the Questionnaire class in Project_2_Code. The Tkinter library is essential to create windows that have buttons, sliders, and checkboxes that the user can click to input data, instead of purely typing it into the Python console. When the questionnaire is created, the first thing that pops up is the home page, showcasing the name of the program alongside a start button. Once the start button is clicked, it initializes the questionnaire using the method initialize_questionnaire(self), and it creates buttons via the method create_button. However, to create these buttons, there is a parameter, "command", which controls what the button does when it is clicked. To make this command, we split the buttons up into multiple types, such as yes_no and gender (male and female), and then we call the method on_click, giving the type of the button as its parameter. With this, we successfully made each important button by calling another method to assign the command to each button. However, we also need a slider (for age) and an entry box for text-based questions, but they are manually created and all of these newly created buttons are hidden, except for the agree and disagree buttons. These buttons are not hidden, because the questionnaire starts with the terms and conditions page. Additionally, we used other parameters to help enhance the image of the buttons, which includes text, color, font and size. We also could change the location of each button and text, by using a grid from tkinter.

In addition to tkinter, we used plotly.graph_objects and from plotly.subplots import make_subplots to make our interactive visualizations. First, were the graphs that showcased the difference between the user's inputs and the average cardiovascular patient and the average non-cardiovascular patient of the same gender. These two graphs had the categories of the data and user inputs as their x-axis, while the average value was the y-axis. To do this, we first had to define a new figure, with 1 row and 2 columns (for 2 bar charts), and we also could give these bar charts their names within this function by using the parameter subplot_titles. The reason why we had 2 bar graphs was that we needed 1 bar graph to showcase the values that are not fixed to 1-3 or 1-2, and another to showcase these fixed values. Then, we added traces to this figure by using fig.add_trace, adding 3 bars per graph (representing the differences between the average and user), by using fig.add_trace(go.Bar(...)). By using the parameters within Bar(), we could choose the name of the bar, the x-axis (categories) and the y-axis values (changes for each bar). Then, once we had a total of 6 traces, we could combine the bars using fig.update_layout(barmode = 'group), which combined 3 traces, and the other 3 together, making 2 graphs appear on the same page. In addition to this parameter, we also used title_text to change the main name of the graph. This is how we made one of our visualizations, but we have another, which is the pie charts. The pie charts were created to show the number of individual patients that have either a higher/lower/equal stat to the user (depending on the gender), which we did using a method called analyze_data_for_gender() within the DataAnalysis class. We have 10 pie charts, correlating to each category. Similarly to the bar graphs, we used the same libraries, but we had to use a for loop to go through every category, call analyze_data_for_gender() on each, use plotly.graph_objects to add a pie chart trace to the figure and update the figure to add its unique category name and its positions (rows and columns). Lastly, we update the figure after the loop ends, adding the main title by using the parameter title, turning the legend on by using

showlegend = True, and by adjusting the legend to be closer to the charts by using the legend parameter.

Instructions

Before following the instructions below, please ensure all necessary Python libraries listed in `requirements.txt` are downloaded. Additionally, download and use the zip file found on Markus. Make sure to run `main.py` after completing the previous steps. We have included sample input information should you prefer not to enter your own personal information.

1. You will first be welcomed by the home screen for our program: CardioAlert! Please press the "Start" button to proceed with the program.
2. Next, a list of terms and conditions will appear on the interface in the form of a checklist. Please read each term/condition mentioned and select/click on the corresponding checkbox if you agree with each statement. To proceed further with the program, ensure to select all checkboxes and afterwards select the "I agree" button. A popup stating "You may now proceed" will appear if done correctly. Press "OK" to move forward.
   Note: If not all boxes are checked upon clicking "I agree", the program will prompt you to select all boxes before proceeding. If "I disagree" is selected, the program will exit/quit.
3. You will see the question "Enter your age:" with a slider. Adjust the slider to match your age in years, then press the "next" button to continue.
4. For gender selection, click on the corresponding "Male" or "Female" button.
5. Enter your height in centimeters in the provided text field and press "Submit" to continue.
6. Similarly, enter your weight in kilograms in the designated field and submit.
7. Provide your systolic blood pressure in mmHg in the text field and submit.
8. Enter your diastolic blood pressure in mmHg and press "Submit".
9. Indicate your cholesterol level by entering the corresponding integer (1 for normal, 2 for above normal, 3 for well above normal) in the text field and submit.
10. Enter your glucose level with the corresponding integer in the provided field and submit.
11. Answer whether you are an active smoker by selecting the "Yes" or "No" button.
12. As the last question, you will see the following: "Would you like to see your results? You will receive a general prediction, an estimated probability, and after, you'll be able to compare your results with the data used to train the model visually on a new tab. Once you exit from the results, the questionnaire will close.". You will also see two buttons named "Yes" and "No". Please select the "Yes" button to proceed further with the results. (Please note that if you press the "No" button, the program will close).

13. The first result you will see is a popup of your overall probability prediction of having cardiovascular disease represented as a combination of a bar and a percentage. After carefully reading your prediction/probability chances, please close the popup to proceed further with the in-depth analysis and visualization.

14. Next, you will see two new web-browser tabs that will open on your browser, make sure to click on the first tab, please! On the first tab, you will see two bar graphs, each comparing the user's input statistics, along with the average statistics of someone with cardiovascular disease, and someone without cardiovascular disease. The two bar graphs are separated by responses with true numerical values (weight, height, etc) versus responses with categorical values (cholesterol, etc). On the right-hand side, you can see the legend which helps you understand which colours represent which categories. Additionally, by hovering your cursor over each bar, you can see the specific value of the bar with decimal points, as well as a shortened version of the category it represents as well! The x-axis represents the response categories (age, weight, etc), while the y-axis represents the specific values of each statistic. Additionally, if you click on any of the "legend squares", you can include/exclude the bar of which the category you clicked (average cardiovascular patient, etc). Furthermore, we also added an information category regarding BMI, which we calculated based on the prior user inputs given.

15. On the second page, you will be able to see eight pie graphs, each representing a category of information provided. Each pie graph is then split into six categories:

    - the number of people with the statistic **above** the user input **with** cardiovascular disease
    - the number of people with the statistic **above** the user input **without** cardiovascular disease
    - the number of people with the statistic **below** the user input **with** cardiovascular disease
    - the number of people with the statistic **below** the user input **without** cardiovascular disease
    - the number of people with the statistic **equal to** the user input **with** cardiovascular disease
    - the number of people with the statistic **equal to** the user input **without** cardiovascular disease.

    As visually represented on the pie graphs, you can see the overall percentage of each category of people within each broader category of information (age, weight, etc). Additionally, by hovering over each sub-category in a pie graph, you can see the overall information category, the specific category of people the section is referring to, the total number of people, along with the overall percentage when compared altogether in the pie graph! Overall, both graphs serve as a great visual representation which compares the user inputs to real statistics gathered about individuals with/without cardiovascular disease. Furthermore, we also added a pie graph regarding BMI, which we calculated based on the prior user inputs given.

Sample Input

(a) Age - 40 years old
(b) Gender - Male
(c) Height - 180 cm
(d) Weight - 100 kg
(e) Systolic blood pressure - 150 mmHg
(f) Diastolic blood pressure - 140 mmHg
(g) Cholesterol - 2
(h) Glucose - 1
(i) Smoking - No

Changes to Project Plan

One of the aspects we changed includes recommending future steps to the user according to our model's prediction. We ultimately decided to take out this component from our project due to a few factors. Firstly, we worried that by providing further medical advice based on our model's prediction, the user would ultimately view the program as a tool for medical advice, and follow our instructions rather than visiting a medical professional. Therefore, by only providing our prediction, along with relevant visualizations, this will likely encourage the user to visit a medical professional to follow up on this prediction, take the necessary tests, and consequently follow the medical advice of a trustworthy medical professional, rather than our program.

As briefly mentioned in the computational overview, we also decided not to use linear regression as we felt heat maps provided a simpler and visually appealing way to understand the correlation between variables. In addition, heat maps let us identify patterns /external relationships in the data and they can handle non-linear relationships between variables which linear regression cannot.

Discussion and Further Exploration

The primary objective of this project was to develop a predictive model for cardiovascular disease and make predictions based on its performance. Our final model achieved a test set accuracy of 73 percent, a validation set accuracy of around 72-73 percent and a training set accuracy of 74 percent, which indicates a moderate level of success in terms of prediction of cardiovascular disease.. Additionally, since all of these accuracies are within 2 percent of one another, this suggests the model is not overfitting to the training data.

In terms of precision and recall from our classification report via the validation set, our model was better on average at predicting the absence of cardiovascular disease (true negatives) compared to predicting its presence (true positives). In an ideal situation, these accuracies should have been flipped, as true positives are much more significant, especially when considering the potential seriousness of cardiovascular disease. However, our F1 scores were all above 0.72, indicating that the model achieved a good balance between precision and recall for both classes. The results of our exploration showcase the effectiveness of using machine learning algorithms for disease prediction.

As for limitations, there were quite a few for both the dataset and the GUI. Firstly, for the GUI, although the tkinker library is extremely versatile and effective for creating an interface, we had a lot of trouble with merging it with plotly, in particular, we wanted a plotly graph to show within the tkinter window, but it was extremely hard to do so. We tried to attempt to use some other libraries, such as plotly.io, but we ended up scraping the idea all together and created a graph via Tkinter to work around the limitations. Another limitation we had was attempting to combine both bar graphs and pie charts together, using plotly. However, we realized quickly that, unlike bar graphs, we had to specify a domain parameter when combining multiple pie charts within a single page, but when we attempted to do that with the bar graphs, we would continuously get errors that we couldn't fix. Thus, we decided to separate the bar graphs and pie charts, which we realized made sense because they show two separate ideas. Now, focusing on the pie charts, we stumbled across multiple obstacles, relating to their positions. Since we had to skip the gender pie chart within the for loop (because we wanted to show pie charts for each specific category with respect to the gender of the user, so the gender category is irrelevant), it left a big space where the gender piechart was supposed to be. We attempted to close this gap by shifting the positions over, but it ended up not working, so instead, we decided that bringing the legend closer to the graph would not only be aesthetically pleasing, but it'll also fill in the gap. Thus, although there were some limitations in the graph creation section of our code (using plotly), we ended up overcoming these obstacles by working around them.

In terms of limitations with the dataset, there were some potential limitations in accounting for all cardiovascular risk factors, so the dataset we used for model training and testing may not be representative of the broader population. In particular, take diabetes for example. Diabetes is one of the leading causes of cardiovascular disease "People with diabetes are 2 to 4 times more likely than others to develop cardiovascular disease, (Hopkins). However, within the dataset, there was no category to see the correlation between these two major diseases, thus, it's a major limitation when we also want the prediction to work for a subset of the population. Therefore, although the dataset is comprehensive, it still lacks data that would make the prediction model useful for a broader range of individuals, thereby limiting its applicability and accuracy in the real world.

For further exploration, we could investigate other types of machine learning models to help improve our accuracy and reliability. The reason we decided to use a classification tree for this project is to make trees a fundamental way to represent the data we computed. In the future, we could make improvements by using a more robust dataset, making a linear regression model or support vector machine model and comparing/contrasting it with the classification tree model to see which model predicts the most effectively.

References

[1] Amos, David. *Building Your First Python GUI Application with Tkinter*. Realpython.com, realpython.com/python-gui-tkinter/#building-your-first-python-gui-application-with-tkinter.

[2] Bhatt, Deepak. *Which Blood Pressure Number Matters Most?*. Harvard Health, 1 Apr. 2021, www.health.harvard.edu/heart-health/which-blood-pressure-number-matters-most#:~:text=The%20bottom%20(second)%20number%2C.

[3] Booth, Stephanie. *Early Heart Failure Diagnosis Is Key.* WebMD, 2023, www.webmd.com/heart-disease/heart-failure/early-diagnosis-heart-failure.

[4] CDC. *Heart Disease Facts.* Centers for Disease Control and Prevention, 15 May 2023, www.cdc.gov/heartdisease/facts.htm#:~:text=Heart%20disease%20is%20the%20leading.

[5] Held, Claes et al. *Body Mass Index and Association with Cardiovascular Outcomes in Patients with Stable Coronary Heart Disease – a STABILITY Substudy.* Journal of the American Heart Association, vol. 11, no. 3, Feb. 2022, doi:10.1161/jaha.121.023667.

[6] Mosca, Lori et al. *Sex/Gender Differences in Cardiovascular Disease Prevention.* Circulation, vol. 124, no. 19, Nov. 2011, pp. 2145–54, doi:10.1161/circulationaha.110.968792.

[7] Pandas. *Python Data Analysis Library.* Pydata.org, 2018, pandas.pydata.org/.

[8] Plotly. *Plotly Express in Python.* Plotly.com, plotly.com/python/plotly-express/.

[9] scikit-learn. *Scikit-Learn: Machine Learning in Python.* Scikit-Learn.org, 2019, scikit-learn.org/stable/.

[10] Ulianova, Svetlana et al. *Cardiovascular Disease Dataset.* Www.kaggle.com, 2019, www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset.