Heart Disease Prediction

Justin Jiang
CSC 84030

***Abstract*—This project aims to find out what factors contribute to heart disease in America. Machine learning models are used to model their performance in classifying heart disease. This is to extract features that help predict what contributes to heart disease. The datasets used for this project are surveys from the BRFSS (Behavioural Risk Factor Surveillance System). The project utilizes the 2022 BRFSS to predict heart disease, while the 2021 survey is used to help with preprocessing the 2022 data. Both surveys contain over 300 questions about a participant's health and routine. Preprocessing includes mapping values, dropping duplicates, dropping columns, filtering data, encoding data, partitioning data, and filling in missing values. This is to help sufficiently run the model. All mentioned methods are done under PySpark. Due to the nature of this project being a classification problem, random forest, logistic regression, and gradient-boosting trees are chosen. All machine learning models are cross-validated and are chosen based on the f1 score. The models performed sufficiently well. The gradient-boosting tree model showed that BMI, height, mental health, physical health, and weight contribute to heart disease. However, BMI is a measurement of both height and weight. Other factors that might significantly contribute to heart disease are a person's hearing. These findings should be researched in medicine to effectively prevent heart disease.**

I. Introduction

Heart disease is the leading cause of death for Americans, with devastating statistics that emphasize its widespread impact. According to the CDC, heart disease is responsible for over 700,000 deaths annually. Although the number of deaths has decreased by 11% over the past decade (2009 to 2019), the toll remains unacceptably high. This decline is evidence of improved awareness of well-known factors, such as diet and lifestyle. However, despite these efforts, heart disease continues to claim lives, suggesting there may be additional, less obvious factors at play. Investigating these hidden contributors—whether genetic, regional, behavioral, or linked to past medical conditions—could unlock critical insights to further reduce the burden of heart disease.

The Behavioral Risk Factor Surveillance System (BRFSS), the world's largest health-related telephone survey, offers a unique opportunity to explore these questions. Conducted annually, the BRFSS gathers extensive data on participants' physical characteristics, medical histories, habits, and health outcomes, providing a rich foundation for predictive modeling and feature analysis. Questions include, have you been vaccinated, when was the last time you had a heart attack, how much vegetables you eat a day, have you been injured for the past few days, and what is your age and height. The survey contains a tremendous number of questions and will be addressed in the data portion.

The dataset used for this project is the 2022 BRFSS dataset, which contains responses from over 445,000 participants on topics ranging from lifestyle choices and medical history to geographic location and health conditions. Using this dataset, the goal is to predict heart disease and identify significant contributing factors. However, the data presents challenges, including missing values and imbalances between participants with and without heart disease. To address these issues, the 2021 BRFSS dataset is used as a reference for imputing missing data in the 2022 dataset. Despite containing more raw data, the 2021 dataset requires extensive preprocessing to align with the 2022 dataset, ensuring consistency across features and values. This preprocessing step involves feature mapping, imputation of missing values (2021 dataset), and encoding categorical responses into numerical formats suitable for machine learning models.

A key aspect of the preprocessing pipeline is the use of Jaccard similarity to handle missing values. Jaccard similarity, computed from sparse vectors of encoded features, is employed to find the most similar records between the 2021 and 2022 datasets. This enables imputation of missing values in the 2022 data using the most similar record from 2021. By completing this step, the 2022 dataset is prepared for machine learning analysis, ensuring both completeness and consistency. To predict heart disease, three machine learning models—random forest, gradient boosting trees, and logistic regression—are applied to the cleaned and preprocessed dataset. Each model is fine-tuned using hyperparameter optimization to maximize predictive performance. Evaluation metrics, including F1 score, are used to identify the best-performing model. The final model to be used for feature importance is determined by a model's AUC. Finally, feature importance analysis is conducted on the final model to uncover which factors have the greatest impact on heart disease prediction.

The significance of this project lies in its potential to inform public health. By identifying lesser-known factors that contribute to heart disease, healthcare professionals and policymakers can design targeted prevention strategies. If specific behaviors or medical histories emerge as critical factors, awareness can be tailored to address those risks. Ultimately, the findings of this project aim to further understand heart disease.

II. Background and Related Work

This project applies several preprocessing techniques to prepare the data for machine learning. String indexing, a method similar to ordinal encoding, assigns numeric values to categorical column entries. One-hot encoding creates new features for each unique value in a categorical column. Vector assemblers combine multiple columns into a single vector. Sparse vectors represent data in a compact format where the position indicates the column and the value at that position indicates the data point. The project utilizes a Spark session configured to optimize resource allocation on Google Colab. The session is initialized and is tailored to leverage Colab's computational power. Key configurations include allocating 24 GB of memory each to the driver and executor, assigning 8 cores to the executor, enabling dynamic resource allocation, and setting the number of shuffle partitions to 100. These settings ensure efficient data processing and scalability within Colab's environment. Other works have used the preprocessed version of the 2022 survey data. However, it uses the version with no NULL values available. The works in Kaggle use the same machine learning models in the project. The performance of these models has been suboptimal, often yielding mediocre results. This underperformance is largely attributed to data imbalance, with a roughly 1:20 ratio between participants who experienced a heart attack and those who did not. The input from other workouts is a filtered preprocess dataset with rows still maintained. There is not a clear baseline from the other works because of the different approaches people took to classifying heart disease.

III. Method

A. Data

Detail the methodology you followed, covering: Originally the source of the two datasets came from the CDC. Both datasets were XPT files. The datasets used are processed as CSV files and obtained from Kaggle. All features are abbreviated and values are encoded. For example, a question that asks about smoking is called _SMOKER3, and "Yes" is represented as "1". A data dictionary is provided for both datasets describing the name of each feature, and the encoded values.

The data uses a preprocessed BRFSS from 2022. The data contains 40 questions (40 columns) and 445,132 records. Each question asks the user about their location, lifestyle, physical health, mental health, medical history, and physical attributes. The target variable for this dataset is the participants who had a heart attack and the predictor variable is the other features. Performing exploratory data analysis, it was shown that there is a huge imbalance between people who had a heart attack and people who didn't have one. 416,959 did not have a heart attack, 25,108 did have a heart attack, and 3,065 people did not respond to having a heart attack. A map showing the number of participants having a heart attack by the state is shown in Fig. 1.

The BRFSS from 2021 will be used as a reference dataset to help preprocess the 2022 dataset. The 2021 dataset will be used to help fill any empty values in the 2022 dataset. Unlike the 2022 dataset, the 2021 is not preprocessed. Preprocessing is needed on the 2021 dataset before processing the 2022 dataset. The 2021 data has 438,693 records and 303 columns. However, some of the questions asked for the 2022 survey are not present in the 2021 survey.

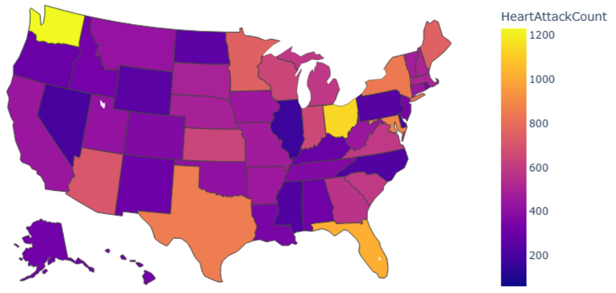


Fig. 1: Participants who had a heart attack by state.

B. Preprocessing

To align the 2022 survey with the 2021 survey, the data dictionaries for both years were carefully reviewed. The 2022 survey included four questions that were not present in the 2021 survey. As a result, four columns from the 2022 survey—Hours of Sleep, Risk of HIV, Removed Teeth, and Tested for COVID—were dropped, as they did not exist in the 2021 survey.

A Kaggle notebook used for preprocessing the 2022 survey was referenced to guide the preprocessing of the 2021 survey data. The feature names used in the 2022 preprocessing notebook were also applied to rename the columns in the 2021 survey. However, a challenge arose because BRFSS changes feature names annually. For instance, the _SMOKER3 column in the 2022 survey could be named _SMOKER2 in the 2021 survey. To address this, the data dictionaries were closely examined to identify corresponding feature names in the 2021 survey. After matching the features, the 2021 data frame was filtered to include only the identified columns. The identified features were renamed to the names in the notebook. For example, _SMOKER2 is renamed to SmokerStatus. Renaming the features is necessary because it allows for the mapping of the encoded values in the 2021 data frame.

All code from the Kaggle notebook is written in Pandas. Since this project is operating utilizing PySpark, the mapping is modified so PySpark can manipulate the mapping. This mapping translates all binary responses, where "1" is "Yes" and "0" is "No". Furthermore, other categorical feature values were translated according to the data dictionary. The mapping (feature names and paired values) is broadcasted for efficient mapping. A UDF (user-defined function) is created where based on a column input and value input, it finds the column in the broadcasted value and returns a value based on the inputted value. After creating the UDF, the UDF is applied to each column so that the encoded values are translated to their mapped value. Features that have binary responses only use the mapping with only “Yes” or “No” as values. For numeric columns, the values are converted according to the same unit of measure in the 2022 data frame. It is unclear what the unit of measure is, but it divides the values by 100.

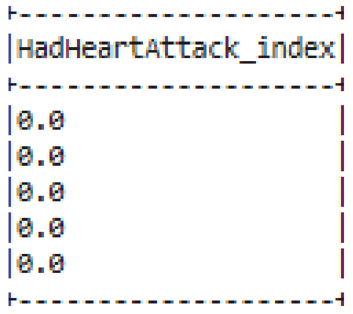
To conserve time, especially for modeling, a state will be observed to predict heart disease. A data frame is created, where it shows the number of heart attacks per state. This is done by selecting the column with state and heart attack, as well as summing the number of times people had a heart attack. After looking at the number of heart attacks in each state, New York was chosen for containing the most number of heart attacks (2,105). It also is the 6th ranked state with the most heart attacks (839) according to Figure 1. This reduces the 2022 data frame size from 445,132 records to 17,800 records. Since the 2021 data frame is still large and makes finding similarities tedious, the size was reduced from 438,693 to 4137. This is from filtering the data frame to contain rows from New York. Then, it was filtered again where it contains rows from participants who had heart attacks and choosing 2,000 random samples from people with no heart attacks. This also addresses any possible data imbalance in the 2021 data frame. 4,105 rows were expected, yet mysteriously there are 32 more. This is later addressed by dropping them later in the project.

A function to create a temporary data frame was created to count the missing values for each feature in a data frame. This temporary data frame was generated by filtering rows with NULL values and calculating the total number of NULL values per feature. The resulting data frame lists all features along with their respective counts of NULL values. The 2021 data frame was used in the function to count the number of NULL values. It was shown that tetanus shots and chest scans contain almost all NULL values. These features were dropped for both the 2021 and 2022 data frames.

There are two steps needed before filling in the missing values in the 2022 data frame with the 2021 data frame. The first step is to fill in the NULL values of the 2021 data frame. To fill in the NULL values for each categorical feature, the NULL values are replaced by the most frequent response for each feature. For numeric columns, it uses the average of the responses. Two dictionaries containing the name of the feature and the most frequent value were created to determine which feature is categorical or numeric. Each dictionary stores features based on the data types of their values. Features containing int or double data types are put in one dictionary, and the rest go to the other. The filled data frame is then de-duplicated to eliminate any duplicate rows in the data frame.

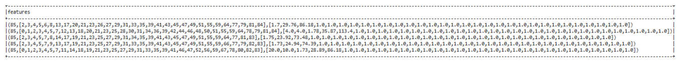
C. Encoding

The second step is to encode the data. The categorical columns are encoded for both data frames. Before one encoding, a string indexer is used. This is to map the values in each categorical feature. This must be done before one-hot encoding. This results in columns that are transformed from string indexer containing a numeric version of the values. After using the string indexer, a one-hot encoder is used to fit the 2022 data frame. The 2022 data frame is fitted because the 2021 data frame should have values that match the mapping of the 2022 data. The one-hot encoder transforms both the 2021 and 2022 data frames to create one-hot encoded columns. During this process, NULL values in the 2022 data frame are retained. After one hot encoding, PySpark’s vector assembler is used to transform the one hot encoded columns into a sparse vector for each row. These sparse vectors will be in a column titled ’feature’. Finally, the 2022 data frame is split into two subsets: one containing rows with NULL values in their sparse vectors and another without NULL values. The subset with NULL values has a new column with the index of the row.



```
+-------------------+
|HadHeartAttack_index|
+-------------------+
|0.0                 |
|0.0                 |
|0.0                 |
|0.0                 |
|0.0                 |
+-------------------+
```

Fig. 2: Example of string index column.



```
|features
+
|(85,[2,3,4,5,6,8,13,17,20,21,23,26,27,29,31,33,35,39,41,43,45,47,49,51,55,59,64,77,79,81,84],[1.7,29.76,86.18,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0])      |
|(85,[0,1,2,3,4,5,7,12,13,18,20,21,23,25,28,30,31,34,36,39,42,44,46,48,50,51,55,59,64,78,79,81,84],[4.0,4.0,1.78,35.87,113.4,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0])|
|(85,[2,3,4,5,7,8,14,17,19,21,23,25,27,29,31,34,35,39,41,43,45,47,49,51,55,59,64,77,81,83],[1.75,23.92,73.48,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0])          |
|(85,[2,3,4,5,7,9,13,17,19,21,23,25,27,29,31,33,35,39,41,43,45,47,49,51,55,59,66,77,79,82,83],[1.73,24.94,74.39,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0])    |
|(85,[0,1,2,3,4,5,7,11,14,18,19,21,23,25,27,29,31,33,35,39,41,46,47,52,56,59,67,78,80,82,83],[20.0,10.0,1.73,28.89,86.18,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0])   |
+
```

Fig. 3: Example of sparse vector and ”feature” column.

D. Filling in NULL values

MinHashLSH is used to find the similarities between the subset with NULL values and the 2021 data frame. Since MinHashLSH uses sparse vectors, the ’features’ column is used in both data frames. The 2021 data frame is fitted because it is the data frame in which the subset should find similarities. A new data frame is created where it contains the index of rows in the subset data, the sparse vectors (from the 2021 data frame) that are similar to a row in the subset, and the Jaccard distance. The data frames produce multiple vectors similar to a row in the subset. The new data frame then sort the rows by the Jaccard distance. Afterward, the data frame creates a new column to contain the maximum Jaccard distance for each unique index in the subset. The data frame is then filtered to contain rows in which each Jaccard distance for each unique index is the maximum Jaccard distance. The temporary data frame is joined with the subset based on the subset index. The resulting data frame is the subset with NULL values with the sparse vectors from the temporary data frame. Finally, the sparse vectors originally from the subset have their NULL values filled in based on the sparse vectors from the joined sparse vectors in their respective rows. The columns from the temporary data set are dropped once the sparse vectors eliminate the NULL values. This result is the subset with NULL values having their NULL values substituted.

The processed subset is unionized with the subset with no NULL values. Any NULL values from the heart attack column that were string-indexed are dropped. Sometimes during the encoding process, more classes are made when using the string indexer (unknown reason as to why). Rows are dropped if they contain a value of more than 1 in the heart attack column because it has to be binary. The final data frame for 2022 contains the columns before encoding, the string-indexed columns, the one-hot encoded columns, and the sparse vectors from assembling the one-hot encoded columns.

E. Model Building

The cleaned 2022 dataset is now ready for model building. The sparse vectors ("features") serve as the predictor variables, while the string-indexed heart attack column is the response variable. The dataset is partitioned into 80% training and 20% testing subsets. Three machine learning models are developed using the same methodology: Random Forest, Logistic Regression, and Gradient Boosting Trees. All three models are configured with "features" being the input, string-indexed heart attack as the label, and prediction being the predictions. Each model is optimized with specific hyperparameter settings:

- **Random Forest:** Number of trees: 20, 50; Maximum depth: 10, 15
- **Logistic Regression:** Regularization parameters (λ): 0.01, 0.1; Elastic net regularization (α): 0, 0.5, 1
- **Gradient Boosting Trees:** Maximum depth: 10, 15; Maximum iterations: 20, 50

The cross-validator method is used to test each combination of parameters for their model. Cross-validation is performed in 3 folds. The models fit the train data and are tested with the test set. The best parameters are determined by their f1 score. The best models have their AUC calculated to determine which best model is the definitive best one.

The AUC is calculated in two ways for the random forest and logistic regression. The first method converts the resulting predictions into a pandas data frame to get the predictions extracted. It extracts the positive class probabilities (heart attack) from the predictions (there was a third class for unknown reasons). It calculates the AUC by using Sklearn's AUC method using the labels and positive class probabilities as input. The other method is used on the models, where it uses PySpark's binary evaluator using AUC as a metric. ROC plots are generated from plotting the false positive rates (fpr) and true positive rates (tpr). Fpr and tpr are obtained during the process of caluclating the AUC in Sklearn.

PySpark has a method to get feature importance from a model. This results in an index of the feature and its importance. The feature importance is sorted. To get the feature name from the index, vector assembler has a method to get the feature name from the index. The ranking of feature importance is done by converting the index into its feature name after sorting the feature importance in descending order.

IV. Results

From looking at Fig 5, it is shown that both the f1 score and AUC are the same. A problem that might cause this is a sudden 3rd class that was created during the machine learning pipeline. The probability from the 3rd class could have caused the evaluation of the metrics to be similar. Another unexplained occurrence is why the metrics indicate the models perform very well. This is especially more confusing with AUC being able to be calculated despite there being 3 classes instead of 2. The metrics from Sklearn should be handled with skepticism because of the unexplained causes of high performing metrics and calculation with a third class.

The Random Forest model achieved its best performance with a maximum of 50 trees and a maximum depth of 15, which were the upper bounds set during the grid search. While these parameters yielded an AUC of 0.844, indicating a strong ability to distinguish between the two classes, it is possible that larger parameter values—such as more trees or greater depth—could further optimize the model's performance. The high dimensionality of the dataset (33 features) implies that the model is learning from a wide array of inputs, which may explain why the feature importance scores appear smaller compared to those in gradient boosting trees. Unlike gradient boosting, which tends to emphasize a small subset of highly predictive features, the Random Forest model distributes importance more evenly across all features, suggesting that the features contribute dependently to the predictions. The plotted ROC curve confirms the model's ability to reliably distinguish between the two classes. This reflects the model's robustness, aided by its use of all 50 trees to reduce variance and improve stability. However, in the context of class imbalance, the model's performance may still be affected by the underrepresentation of the minority class. Increasing the number of trees could provide more insight from the minority class.

Comparing the AUC of the baseline random classifier (AUC of 50), it is shown to be closer to 1 than the baseline. Random Forest model ranks as the best performing model when considering metrics from Sklearn. However, it is ranked the second best using metrics from Pyspark. The feature importance plot for the Random Forest model, while less peaked than that of gradient boosting trees, aligns with the expectation that the Random Forest model spreads importance more evenly across features. The logistic regression model performed with a regularization parameter of 0.01 and an elastic net mixing ratio of 0.5, resulting in an AUC of 0.715. This indicates that the model has a limited ability to distinguish between the two classes, performing only slightly better than a random classifier (AUC = 0.5). The small regularization parameter (0.01) suggests that the model is likely overfitting to the training data, which aligns with the mediocre AUC score. The elastic net mixing ratio of 0.5 indicates a balanced contribution from both L1 (lasso) and L2 (ridge) regularization, allowing the model to penalize complex patterns while still capturing feature relationships. However, this balance may not have been sufficient to improve the model's overall generalization.

The issue of class imbalance likely exacerbated the poor performance of logistic regression. Logistic regression inherently assumes balanced classes and may struggle to properly model the minority class, resulting in biased predictions.

When compared to the baseline random classifier, the AUC of logistic regression is closer to 0.5 than to 1, further emphasizing its limited predictive performance. Among the evaluated models, logistic regression ranks as the worst-performing model in terms of AUC in both Scikit-learn and PySpark, highlighting its inability to effectively model the data.

The gradient boosting tree performed sufficiently well with a maximum depth of 10 and maximum iterations (similar to maximum trees) of 50. These parameters lead to an AUC of 0.79, making the model be able to sufficiently distinguish the two classes. The maximum depth of 10 aligns with gradient boosting being able to learn the most from a subset of data. The maximum of 50 iterations suggests the model needs more iterations to learn the minority class. The model distributes the feature importance more towards a subset of data, unlike a random forest, suggesting that only a subset has more predictive power than others. In the context of class imbalance, the model's performance may still be affected by the underrepresentation of the minority class. Increasing the number of iterations could provide more insight from the minority class.

Model	**Parameters**
Random Forest	numTrees = 50, maxDepth = 15
Logistic Regression	regParam = 0.01, elasticNetParam = 0.5
Gradient Boosting Trees	maxDepth = 10, maxIter = 50

TABLE I: Models parameters after cross-validation

Model	**F1 Score**	**AUC**	**AUC (Sklearn)**
Random Forest	0.917	0.917	0.844
Logistic Regression	0.913	0.913	0.715
Gradient Boosting Trees	0.930	0.930	0.794

TABLE II: Model Performance Metrics

Comparing the AUC of the baseline random classifier, it is shown to be closer to 1 than the baseline. Gradient boosting trees ranks as the second best performing model when considering metrics from Sklearn. However, it is ranked the best using metrics from Pyspark. The gradient boosting tree model performed well with a maximum depth of 10 and 50 iterations (similar to the number of trees), achieving an AUC of 0.79. This indicates that the model is sufficiently capable of distinguishing between the two classes. The maximum depth of 10 aligns with the gradient boosting tree's ability to extract meaningful patterns from a subset of features. However, the number of iterations suggests that the model might require more trees to improve its ability to learn from the minority class, particularly in the context of class imbalance.

The feature importance distribution for gradient boosting shows that the model places greater emphasis on a subset of features, unlike random forests, which distribute importance more evenly. This highlights that only a few features have substantial predictive power. However, class imbalance likely limits the model's ability to fully capture patterns from the minority class. Increasing the number of iterations could help the model better represent the minority class and enhance its predictive performance. When compared to a baseline random classifier (AUC = 0.5), the AUC of 0.79 is significantly closer to 1, confirming the model's ability to effectively separate the two classes. Gradient boosting ranks as the second-best performing model based on metrics from Scikit-learn, while it ranks as the best-performing model when evaluated using metrics from PySpark.

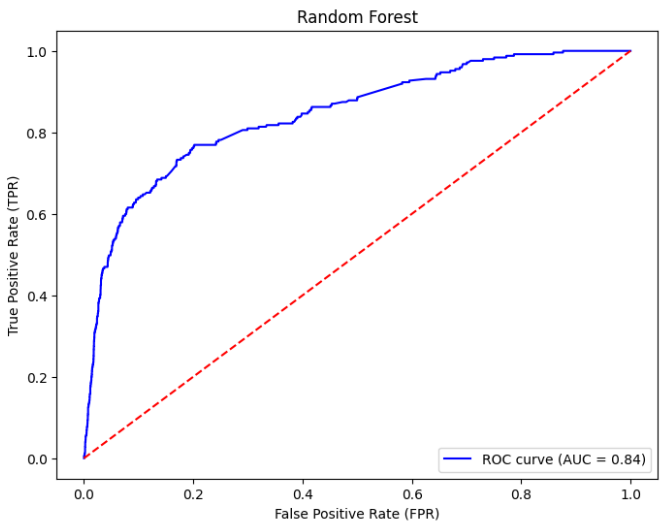


Fig. 4: ROC curve of random forest.

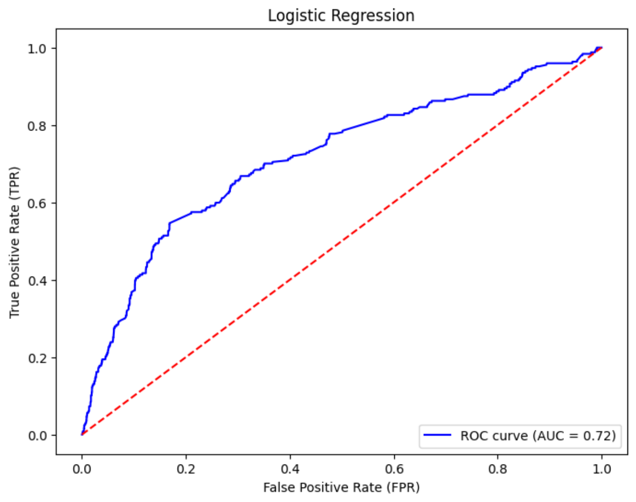


Fig. 5: ROC curve of logistic regression.

Factors contributing to the prediction of heart disease, as shown in Tables 3 and 4, include participants' BMI and any physical or mental conditions experienced in the past 30 days. This is expected, as these factors are known indicators of heart disease risk. BMI, which combines weight and height, is considered a single factor. A higher BMI may suggest an

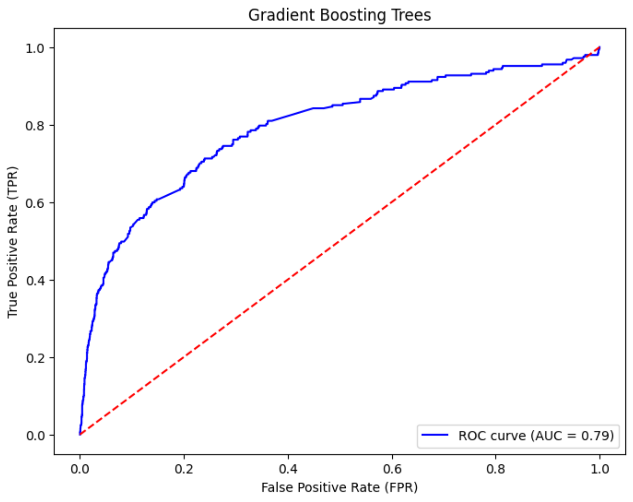


Fig. 6: ROC curve of gradient boosting trees.

unhealthy lifestyle, increasing the likelihood of heart disease.

Physical or mental conditions, whether current or recent, may also be significant because they often indicate ongoing health complications. On the other hand, factors like being hard of hearing are less directly related to heart health. However, it is possible that hearing difficulties are associated with heart disease or that individuals with hearing loss share traits that indirectly contribute to heart disease risk. This line of reasoning might apply to other less obvious factors that help predict the likelihood of heart disease.

Feature	Importance
MentalHealthDays	0.0587
BMI	0.0583
PhysicalHealthDays	0.0582
HeightInMeters	0.0521
WeightInKilograms	0.0477
DeafOrHardOfHearing	0.0349
BlindOrVisionDifficult	0.0320
DifficultyConcentrating	0.0158
DifficultyWalking	0.0158

TABLE III: Feature Importance for Heart Disease Prediction (random forest)

Feature	Importance
BMI	0.1083
HeightInMeters	0.1013
MentalHealthDays	0.0911
PhysicalhealthDays	0.0831
WeightInKilograms	0.0779
DeafOrhardOfHearing	0.0189
LastCheckUpTime	0.0169
SmokerStatus	0.0145
PhysicalActivities	0.0116

TABLE IV: Feature Importance for Heart Disease Prediction (gradient boosting trees)

V. Conclusion

The results demonstrate that the random forest model performed the best in predicting heart disease using Scikit-learn, while the gradient boosting tree model excelled in PySpark. Both models showed robust performance, surpassing results from similar studies. However, skepticism is warranted as many other works rely on accuracy as a metric and use the entire dataset, potentially leading to different estimates. While logistic regression performed the worst among the three models, its results were still superior to those reported in other studies. Key predictive factors identified include BMI and an individual's health condition. Unexpectedly, other debilitating conditions, such as loss of hearing or vision, also emerged as significant contributors to heart disease risk.

Despite these promising findings, the project faced several challenges. The limited resources of Google Colab restricted the ability to compute similarities for the entire dataset, resulting in long runtimes and occasional runtime errors. The time-intensive nature of modeling and hyperparameter tuning—often exceeding an hour—added further constraints. Because of this time was not properly allocated to other parts of the project. Additionally, a persistent issue with the emergence of a third class during preprocessing could not be fully resolved, which may have impacted model performance.

To enhance this work, more features can be selected from the survey to explore its predictive power on heart disease. Leveraging more powerful computational resources would enable the use of more complex models, such as multi-layer perceptrons (MLPs), potentially yielding deeper insights into the importance of specific features. These improvements could lead to even more accurate predictions of heart disease risk.

References

[1] Dariush Bahrami, *CDC BRFSS Survey 2021*, Available at: https://www.kaggle.com/datasets/dariushbahrami/cdc-brfss-survey-2021.

[2] Kamil Pytlak, *Personal Key Indicators of Heart Disease*, Available at: https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease.

[3] Centers for Disease Control and Prevention, *Heart Disease Prevalence*, Available at: https://www.cdc.gov/nchs/hus/topics/heart-disease-prevalence.htm.

[4] Centers for Disease Control and Prevention, *Heart Disease Facts and Stats*, Available at: https://www.cdc.gov/heart-disease/data-research/facts-stats/index.html.