Machine Learning Project Report on Cardiovascular Disease Prediction

Xavier Sánchez Mateus
David Galindo Munté
Maxence Martin

December 17, 2023

1 Introduction

1.1 Subject presentation

This report documents our project's progression in predicting cardiovascular diseases through machine learning techniques. It encompasses our thorough analysis, model training, and exploration of advanced methodologies aimed at enhancing prediction accuracy.

Our journey commenced with an in-depth examination of the dataset, involving meticulous data cleaning and understanding. Subsequently, we delved into training diverse machine learning models, refining them over time. Additionally, we ventured into advanced techniques like ensemble learning, seeking avenues to bolster our predictive capabilities for cardiovascular diseases.

The primary objective of this report is to provide a step-by-step account of our methodologies and findings.

1.2 Presentation of the Data

Feature Name	Feature Type	Data Type	Description
age	Objective Feature	int (days)	Age in days
height	Objective Feature	int (cm)	Height in centimeters
weight	Objective Feature	float (kg)	Weight in kilograms
gender	Objective Feature	categorical code	Categorical code for gender (e.g., 1 for male, 2 for female)
ap_hi	Examination Feature	int	Systolic blood pressure
ap_lo	Examination Feature	int	Diastolic blood pressure
cholesterol	Examination Feature	categorical code	1: normal, 2: above normal, 3: well above normal cholesterol
gluc	Examination Feature	categorical code	1: normal, 2: above normal, 3: well above normal glucose
smoke	Subjective Feature	binary	Binary indicator for smoking (1 for yes, 0 for no)
alco	Subjective Feature	binary	Binary indicator for alcohol intake (1 for yes, 0 for no)
active	Subjective Feature	binary	Binary indicator for physical activity (1 for yes, 0 for no)
cardio	Target Variable	binary	Binary indicator for the presence or absence of cardiovascular disease

Table 1: Presentation of the Data

2 Preliminary Analysis and Data Cleaning

2.1 Preliminary Data Analysis

This section is crucial for understanding our data and, therefore, enhancing the performance of training and prediction. First, to analyze the data properly, we separated the dataset features into categorical and continuous. This allowed us to create various graphs to easily visualize the data.

Initially, we observed the distribution of data based on categorical features. Through these graphs, we noticed that the categorical target "cardio" is balanced, meaning it has an equal number of sick and healthy individuals. There is a significant imbalance in gender data, with twice as many men as women. This could be explained by the fact that men are more affected by cardiovascular diseases, making it easier to find positive cases. However, within each gender, there are equal numbers of sick and healthy individuals. We learned that 80% of our sample exercises, 95% do not drink alcohol, and 91% do not smoke. Additionally, 85% of our sample has a normal glucose level, and 74.8% have a normal cholesterol level.

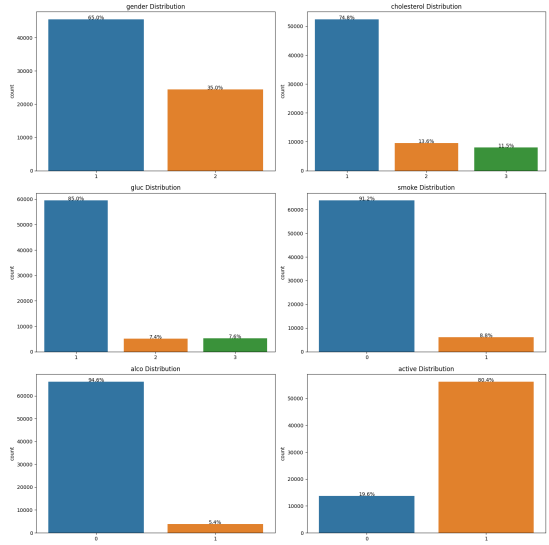

Figure 1: Histogram of Categorical Features

2.2 Cleaning the dataset

The representation of continuous data was more useful. Using boxplots, we identified numerous outliers in the dataset. We adopted two approaches to remove them from the training set. The goal of this removal was to help the model better adapt to the general trend of the data and improve its ability to generalize to new observations.

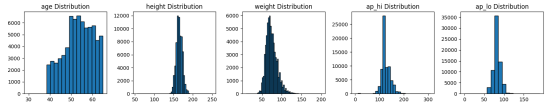

Figure 2: Distribution of Numerical Features

Firstly, we eliminated values that were deemed impossible based on different sources and scientific articles. This was done by setting lower and upper bounds for each category and removing all rows with values outside these bounds. After this, we generated the first dataset, which we would use later for modeling. Before setting it aside, we plotted the histogram of the target feature to check if, after removing rows with impossible values, the dataset remained balanced. In our case, this is particularly concerning because it is likely that abnormal values come from unhealthy individuals, and removing these rows could significantly reduce the number of people with cardiovascular diseases. However, after analyzing the graph, we could see that the dataset remained balanced.

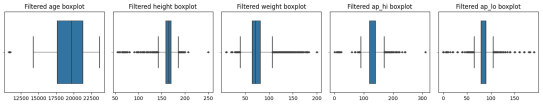

Figure 3: All Boxplots after filtering impossible values

Secondly, we used the dataset without impossible values to build a second dataset, this time without extreme values. For this dataset, we removed outliers in a more conventional way. Specifically, an outlier is considered low if it is below (Q1 - 1.5 * interquartile range) and high if it is above (Q3 + 1.5 * interquartile range). With this manipulation, we obtained our final dataset.

2.3 Principal Component Analysis

Finally, we attempted to reduce the number of features in our dataset by performing Principal Component Analysis (PCA). The idea was to simplify the input features, making the algorithm training faster and

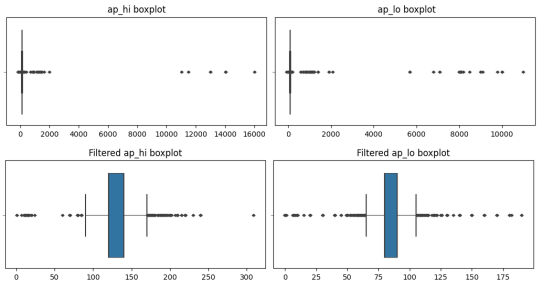

Figure 4: Boxplot of Systolic and Diastolic Pressure Before and After Filtering Impossible Values

allowing us to try higher-degree polynomial transformations. However, after conducting PCA, we found that there was no benefit in performing PCA on our data, as the variance explained by each axis was very low. Moreover, the cumulative explained variance exceeded 97% with 10 components out of 11. Thus, the trade-off between reducing the number of features and preserving information was not worthwhile. In general, the cumulative explained variance graph is nearly linear, without a clear elbow, which is characteristic of a dataset whose dimensionality cannot be reduced without significant information loss.

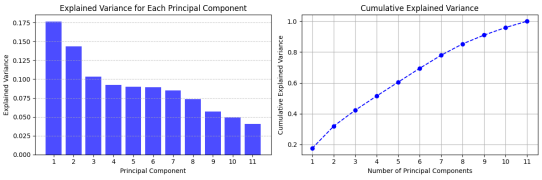

Figure 5: Explained Variance of Each Component and Cumulative Explained Variance

We will train using three datasets: the original dataset serves as a baseline, the second excludes impossible values, and the third removes impossible and extreme values. This approach aims to evaluate the impact of removing outliers on model performance and determine whether excluding outliers improves the results.

3 Training and Model Performance Evaluation

3.1 Preparation of Training and Test Data

Before training our models, we preprocessed our data to facilitate learning for our models. We separated cat-

egorical and continuous features again for different preprocessing.

For categorical features, we transformed them into binary features by adding the necessary features. This was done to give the model more freedom in fitting points based on categories, minimizing potential bias.

For numerical features, we applied a polynomial transformation and then normalized the data. The same transformations were applied to the test set using the saved mean and standard deviation vectors. Originally, we planned to test our classifiers on polynomial-transformed data with degrees 1 to 5. However, due to computational limitations and the lack of GPU support for sklearn, we had to limit our ambitions to degree 2. This polynomial transformation applied to all classifiers except SVM, which can already represent features in a higher-dimensional space through its kernel parameter.

Now that our data is ready, we can proceed to train our models.

3.2 Training Phase: Optimize Models and Find the Best Performing

In our quest for the best classifier, we decided to test the following classifiers: logistic regression, random forest, k-neighbors, Gaussian Naive Bayes, SVM. For each classifier, we defined a set of interesting parameters to vary and see their impact on model performance. Then, for each of these models, we defined a range of values to traverse with more or less fine granularity. The idea was to perform a grid search with cross-validation for each polynomial transformation and each training dataset to find the best possible parameterization under these conditions. The best parameterization is then automatically recorded in our Excel file by our program. Thus, with 4 classifiers to train on 3 different datasets and with 2 different polynomial transformations, we end up with 24 rows of results in our Excel. Additionally, there are 3 rows for SVM. You can find all these results in the Excel file results.xlsx located in the results folder. In simpler terms, these results are recorded in Table 2 below.

3.3 Performance Evaluation

To assess the performance of our models/classifiers, we have a range of metrics at our disposal: accuracy, recall, precision, specificity, and the F-score. Each metric measures a different aspect of the model:

- **Accuracy**: Measures the total number of correct predictions compared to the total number of samples.

Evaluation of models on test data								
Classifier name	Best parameters for (classifier, degree, training dataset)	Degree	Training dataset	Accuracy	Recall	Precision	Specificity	F-Score
LogisticRegression	{'C': 0.001, 'max_iter': 20000, 'penalty': None, 'random_state': 42, 'solver': 'lbfgs'}	1	dataset	0.723	0.679	0.745	0.767	0.71
LogisticRegression	{'C': 0.001, 'max_iter': 20000, 'penalty': None, 'random_state': 42, 'solver': 'newton-cholesky'}	1	impossible	0.731	0.683	0.757	0.78	0.718
LogisticRegression	{'C': 0.1, 'max_iter': 20000, 'penalty': 'l1', 'random_state': 42, 'solver': 'saga'}	1	extreme	0.734	0.676	0.765	0.792	0.718
RandomForestClassifier	{'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100, 'random_state': 42}	1	dataset	0.737	0.674	0.77	0.798	0.72
RandomForestClassifier	{'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100, 'random_state': 42}	1	impossible	0.737	0.678	0.77	0.797	0.721
RandomForestClassifier	{'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 50, 'random_state': 42}	1	extreme	0.737	0.676	0.771	0.798	0.72
KNeighborsClassifier	{'n_neighbors': 100, 'p': 1, 'weights': 'distance'}	1	dataset	0.691	0.654	0.707	0.728	0.679
KNeighborsClassifier	{'n_neighbors': 100, 'p': 1, 'weights': 'uniform'}	1	impossible	0.732	0.674	0.763	0.79	0.716
KNeighborsClassifier	{'n_neighbors': 100, 'p': 1, 'weights': 'uniform'}	1	extreme	0.732	0.675	0.763	0.789	0.716
GaussianNB	{}	1	dataset	0.596	0.596	0.661	0.736	0.605
GaussianNB	{}	1	impossible	0.665	0.554	0.714	0.777	0.624
GaussianNB	{}	1	extreme	0.683	0.607	0.717	0.798	0.657
SVM	{'C': 10, 'gamma': 'scale', 'kernel': 'linear', 'random_state': 42}	1	dataset	0.727	0.64	0.775	0.814	0.701
SVM	{'C': 10, 'gamma': 'scale', 'kernel': 'linear', 'random_state': 42}	1	impossible	0.732	0.647	0.78	0.817	0.707
SVM	{'C': 0.1, 'gamma': 'scale', 'kernel': 'linear', 'random_state': 42}	1	extreme	0.733	0.644	0.784	0.822	0.707
SVM	{'C': 10, 'gamma': 'auto', 'kernel': 'rbf', 'random_state': 42}	1	dataset	0.736	0.701	0.754	0.77	0.727
SVM	{'C': 1, 'gamma': 'scale', 'kernel': 'rbf', 'random_state': 42}	1	impossible	0.738	0.685	0.768	0.791	0.724
SVM	{'C': 1, 'gamma': 'scale', 'kernel': 'rbf', 'random_state': 42}	1	extreme	0.737	0.68	0.768	0.791	0.722
SVM	{'C': 10, 'gamma': 'auto', 'kernel': 'poly', 'random_state': 42}	1	dataset	0.735	0.691	0.759	0.78	0.724
SVM	{'C': 1, 'gamma': 'scale', 'kernel': 'poly', 'random_state': 42}	1	impossible	0.738	0.686	0.765	0.789	0.724
SVM	{'C': 0.1, 'gamma': 'scale', 'kernel': 'poly', 'random_state': 42}	1	extreme	0.706	0.633	0.742	0.779	0.683
LogisticRegression	{'C': 10, 'max_iter': 20000, 'penalty': 'l2', 'random_state': 42, 'solver': 'lbfgs'}	2	dataset	0.734	0.708	0.748	0.76	0.727
LogisticRegression	{'C': 10, 'max_iter': 20000, 'penalty': 'l2', 'random_state': 42, 'solver': 'lbfgs'}	2	impossible	0.736	0.71	0.75	0.762	0.73
LogisticRegression	{'C': 0.001, 'max_iter': 20000, 'penalty': None, 'random_state': 42, 'solver': 'newton-cg'}	2	extreme	0.737	0.708	0.752	0.766	0.728
RandomForestClassifier	{'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 200, 'random_state': 42}	2	dataset	0.739	0.698	0.76	0.779	0.728
RandomForestClassifier	{'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 200, 'random_state': 42}	2	impossible	0.74	0.699	0.762	0.781	0.729
RandomForestClassifier	{'max_depth': 10, 'min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 200, 'random_state': 42}	2	extreme	0.738	0.693	0.762	0.781	0.726
KNeighborsClassifier	{'n_neighbors': 50, 'p': 1, 'weights': 'distance'}	2	dataset	0.707	0.662	0.728	0.751	0.694
KNeighborsClassifier	{'n_neighbors': 100, 'p': 2, 'weights': 'distance'}	2	impossible	0.727	0.674	0.753	0.778	0.712
KNeighborsClassifier	{'n_neighbors': 100, 'p': 2, 'weights': 'distance'}	2	extreme	0.726	0.673	0.754	0.779	0.711
SVM	{'C': 0.1, 'gamma': 'auto', 'kernel': 'poly', 'random_state': 42}	2	dataset	0.765	0.293	0.756	0.895	0.423
GaussianNB	{}	2	impossible	0.716	0.638	0.757	0.794	0.692
GaussianNB	{}	2	extreme	0.721	0.66	0.753	0.782	0.703

Table 2: Results for Each Combination of Classifier, Parameters, Degree, and Training Dataset

- **Recall**: Indicates the model's ability to identify all positive examples.
- **Precision**: Measures the quality of positive predictions made by the model. In other words, it indicates the proportion of instances that the model correctly identified as positive among all instances it predicted as positive.
- **Specificity**: Measures the model's ability to identify all negative examples.
- **F-Score**: Harmonic combination of precision and recall. Provides a balance between the two metrics.

These results are recorded in Table 2 above for each classifier.

The metric to which we generally give the most importance for choosing a model is accuracy. Here, our best accuracy is achieved by a random forest with a polynomial transformation of degree 2 on the "train without impossible values" dataset. However, we notice that most performances display very close scores, except for Gaussian Naive Bayes, which clearly underperforms all other models.

To more easily visualize the performance of a model, we also have its confusion matrix at our disposal. This matrix allows us to quickly assess the performance of a model. Indeed, the stronger the diagonal of the matrix, the better the classifier.

In conclusion, we have ROC (Receiver Operating Characteristic) curves and the Area under the ROC Curve (AUC) as essential tools to evaluate the performance of our binary classification model. ROC curves graphically present the trade-off between true positive rates and false positive rates, providing an intuitive visualization of the model's ability to discriminate between classes. An ideal ROC curve approaches the upper-left corner of the graph, indicating excellent discriminatory ability. Meanwhile, AUC, a quantitative measure between 0 and 1, summarizes the model's performance overall, independently of a specific decision

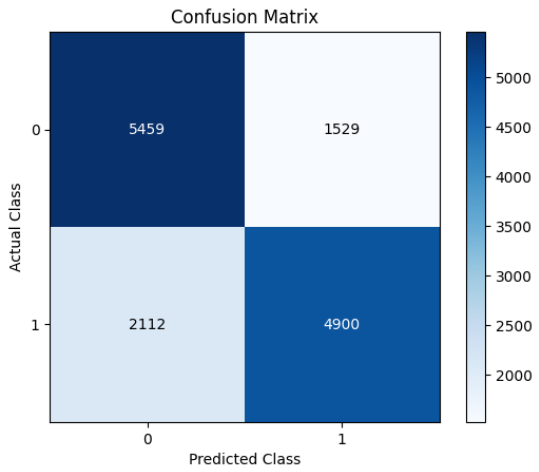

Figure 6: Confusion Matrix of the Best Classifier

threshold. A high AUC indicates superior discriminatory ability, while a value close to 0.5 suggests performance equivalent to random chance. These combined metrics allow for a thorough evaluation of the quality of our models.

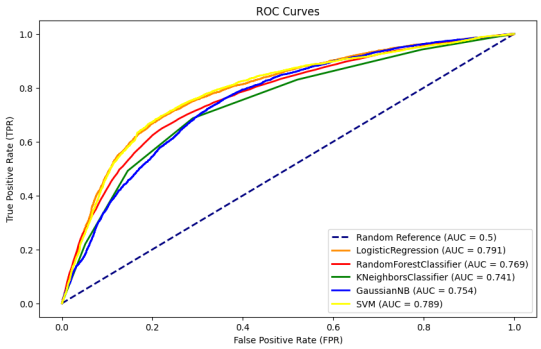

Figure 7: ROC curves for classifiers with default settings, degree 1, and training dataset without extreme values

3.4 Exploring Ensemble Learning Algorithms: Boosting and Voting

In order to try to improve the performance of our previous models we explored the possibilities offered by ensemble learning. So we trained several AdaBoost meta-classifiers with some of our previous base classifiers. We also trained several gradient boosting and voting classifiers with one "hard" vote and one "soft" vote. All results are available in Excel 'results_ensemble_learning.xlsx'. It appears from all these experiments that we still cannot exceed the fateful threshold of 74% with the boosting and voting methods.

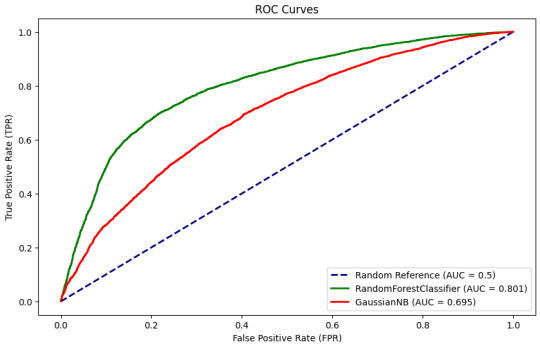

Figure 8: ROC curves of our best and worst classifiers to visualize the difference in performance

4 Conclusion

Despite the efforts invested in this project, the accuracy settled at a somewhat disappointing plateau of 74%. However, upon exhaustive exploration of the Kaggle platform, the primary source of our dataset, it became evident that this performance level might represent the ceiling attainable within the constraints of this specific dataset.

Our project not hitting the expected accuracy shows how tough it is to predict complicated health problems. It really shows how tricky it is to make predictions in healthcare. We found lots of complicated things and uncertainties while trying to predict health issues using the dataset we had. This made it hard to get very precise predictions.

Furthermore, it shows how important good data is for making accurate predictions. The quality of our predictions depends a lot on how good, complete, and relevant the dataset we use is. This teaches us how much better our predictions can be if we pay close attention to getting the right data for healthcare predictions.

In essence, our project journey, though not hitting the accuracy we wanted, shows how hard it is to predict tricky health illnesses. It tells us how important good data is for making predictions that work well. This experience pushes us to keep digging, getting better, and trying new things to make healthcare predictions more accurate.