Predicting risk of Coronary Heart Disease using Machine Learning

Dennis Wiersma

2022-11-13

Abstract

Heart disease is one of the most prominent problems in the world, with it demanding one of the highest death tolls in the entire world. It would therefore be of great significance to prevent this condition when early signs can be detected. To achieve such a result a dataset from the Framingham Heart Study was used to train a machine learning model to predict whether a given subject has the risk of developing coronary heart disease within the next ten years. The trained model can make such a prediction with an accuracy of over 89%, a false negative rate of just 5%, an f-score over 0.89, and an area under the ROC curve of over 0.94. These resuls suggest this trained model can predict the risk of developing coronary heart disease fairly accurately.

Abbreviations

ML	Machine Learning
AI	Artificial Intelligence
FHS	Framingham Heart Study
CHD	Coronary Heart Disease
SMOTE	Synthetic Minority Oversampling Technique
FNR	False Negative Rate
AUC-ROC	Area Under the Curve of Receiver Operating Characteristic
SMO	Sequential Minimal Optimization
CSC	Cost Sensitive Classification
PCA	Principle Component Analysis

List of Figures

List of Tables

Contents

1 Introduction

Heart disease is a condition that gets more prevalent by the day. It is one of the most, if not the most, occurring cause of death in this day and age [WHO, 2020]. It is therefore important to know what factors can lead to someone developing heart disease. Thankfully, a lot is already known about what goes into developing this condition. So, if it is known what causes heart disease, then what can we do with this information? The answer is prevention. Possibly just as important as knowing what causes heart disease would be to know whether someone is likely to develop a heart condition in the near future. After all, if one knows they're at risk of succumbing to the number one killer in the world, they can still make an effort to subvert disaster. In the process of gathering such information a patient could seek advice from their physician, who could then evaluate their lifestyle and physical condition. Using this information the medical practitioner can then estimate whether the patient is likely to be subject to the unease of heart disease.

Computer scientists wouldn't be computer scientists if they weren't trying to program someone out of a job, so what if this evaluation by a doctor could be automated by a computer. A piece of software could evaluate data from dozens of patients in mere seconds. This would cause the need for a physician to greatly decrease for this procedure, lowering both cost and barrier to entry.

To achieve this dream of automation a range of techniques were used, the most important of which is Machine Learning (ML) which is a sub field of Artificial Intelligence (AI). Using these ML techniques a model can be trained using a large amount of data. This model can then predict certain values for a given input. For example, such a model could predict whether a patient has the risk of developing heart disease when given information about that patient's medical condition.

As mentioned before, for the training of this ML model a rather large amount of data is required. The dataset which was used for completing this training phase was gathered from the Framingham Heart Study (FHS) and contains information on the risk of subjects developing Coronary Heart Disease (CHD) within the next ten years. Since these models can be fairly specific to the data on which they have been trained, any bias in the dataset will also translate into the model produced from it. This therefore leaves us with the following question to answer: How well can one predict 10 year risk of coronary heart disease in subjects from the Framingham Heart Study?

2 Methodology

For the successful execution of this project a collection of techniques and software were used instructions for which were provided by Hanze University of Applied Sciences [Noback, 2022]. This, together with the process of going from raw data to the classification of brand new data, will be discussed in this chapter.

2.1 Software

An overview of software, packages, and programming languages used for this project. The rest of this chapter will unfold when and where these various tools were used.

Table 1: Software used for this project.

Software	Version	Function
R	4.2.1	Statistical programming
Java	17.0.5	General purpose programming
Weka	3.8.6	Machine learning software workbench

Table 2: R packages used for this project.

R package	Version	Function
knitr	1.40	Dynamic report generation
ggplot2	3.3.6	Data visualisation
ggpubr	0.4.0	Arranging plots
ggbiplot	0.55	PCA plot
corrplot	0.92	Correlation plot
RWeka	0.4-44	Writing data to ARFF

Table 3: Java packages used for this project.

Java package	Version	Function
Gradle	7.4	Project management
Weka API	3.8.0	Weka programming interface
ShadowJar	7.1.2	Fat JAR building

2.2 Data

As mentioned before, training a machine learning algorithms requires data. This data is a subset of the statistics gathered for the FHS and contains over four thousand instances and sixteen variables.

2.2.1 Acquisition

Obtaining the data was a fairly easy process of hunting down an interesting dataset on the Kaggle website that fulfilled a certain set of requirements. These requirements include, but are not limited to:

- Choosing a supervised learning problem. This means the dataset in question is to contain class labels.
- The set contains data from the life sciences. This includes, for example, biology, chemistry, and medicine.
- At least seven attributes are present in the dataset, but preferably more.
- At least several hundred instances are present in the dataset, but preferably more.

At the close of hunting season a Kaggle page called “Logistic regression To predict heart disease” [Dileep, 2019] was chosen for this project. It is the page that contains the subset of data from the FHS. One thing to note about this Kaggle page is that nowhere does it mention how they acquired this data from the FHS project. It simply links to another, by now defunct, Kaggle page. Apart from that it just has a download link to the data, some information, and a not that comprehensive analysis.

2.2.2 Processing

After acquiring the dataset, the first thing is of course to load it into one's software package of choice, which in this case would be R. Once the data has been loaded it is time to take a good look at it, and it is at this point that the first problem arises. This collection of data contains an `education` attribute consisting of values ranging from one through to four. An explanation of what these numbers represent is nowhere to be found. It was therefore decided to drop this variable, since it is not entirely clear what exactly it represents.

Moving on one runs into the second issue, which are the missing units. Just like a description of the `education` characteristic was missing, a lot of units for the given measurements were missing. Reading the Kaggle page and scouring the FHS website yielded no results, but luckily some of the missing units were easily interpreted.

Now that the missing metadata has been dealt with it was time to correctly label the data. Some of the data consisted of binary nominal values but were encoded in numerics. These variables were transformed into R's factor objects. These factors consist of levels like `male` and `female`, or `yes` and `no`. These are perfect for encoding nominal attributes.

While the missing metadata and wrongly encoded data have been resolved, the missing values in our dataset have not been taken care of yet. Resolving these so called `NA` values will be done by reviewing them on an attribute by attribute basis.
First up are the amount of cigarettes smoked per day by a given subject. The initial step was to check whether every instance that had a missing value to their name was currently a smoker. This turned out to be the case, which meant these values could be imputed from the remaining values. This imputation was done by taking the mean number of cigarettes smoked *by subjects that are currently smokers* and assigning this mean value to the gaps in our data.
Next were the missing values for whether the subject was or wasn't on blood pressure medication. After looking at the dataset it was determined that, as one might suspect, a very large majority of subjects were in fact not on blood pressure medication. It was therefore assumed that the instances with the missing values would not be on these types of medication either.
The following attributes were dealt with in bulk since the process of imputation was precisely identical for each of them. The variables in question are:

- Total cholesterol levels
- BMI
- Heart rate
- Glucose levels

Since all of these attributes consist of plain integer data they can easily be imputed. It was decided that the imputation process would simply consist of assigning the missing values the respective attribute's mean values.

Last but not least there's the problem regarding our so called class variable. This is the attribute which tells us whether a subject actually does run the risk of developing CHD in the next ten years. This attribute turned out to be heavily skewed towards subjects without said risk. This could potentially pose a myriad of issues during the training of our ML model. This issue was solved using Synthetic Minority Oversampling Technique (SMOTE). This is an approach where new (synthetic) instances are created which are labelled with the minority class, which would be the `yes` value in this case. The instances with this label were increased by 450% to solve the issue of imbalance.

2.3 Machine Learning

Now that the data was ready, it was finally time to start work on the ML model. A lot of the work of exploring different algorithms, comparing different algorithms, comparing different hyper parameters, and figuring out what worked best was done using the Weka ML software workbench.

2.3.1 Performance

When trying to develop an ML model there are hordes of different algorithms to choose from, each of which perform better when applied in their own specific use case. So then, how does one figure out which algorithm works best for the data at hand? This is where the Weka experimenter comes in. This section of the Weka workbench lets users easily test and compare algorithms and parameters.
Apart from finding a software solution, another question to answer is which scoring methods to use when making such comparisons. In the case of this research project this encompasses the following scoring methods:

- Classification speed
- Accuracy
- False Negative Rate (FNR)
- F-measure
- Area Under the Curve of Receiver Operating Characteristic (AUC-ROC)

These metrics will be used to compare different algorithms, but which algorithms to compare? ML algorithms can be divided into different categories. A few examples of these categories would be rule based, tree based, and lazy learning based algorithms. A selection of algorithms to compare has been carefully curated by ensuring every category of was represented. This resulted in the following algorithms being compared:

- ZeroR
- OneR
- J48
- Random Forest
- IBk
- Naive Bayes
- Simple Logistic
- Sequential Minimal Optimization (SMO)

The results of this comparison prompted the application of Cost Sensitive Classification (CSC). This is a technique where a cost can be applied to instances which are classified a certain way. The classifications that CSC can be applied to are True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) designations. In the context of this research a FN allotment regards a subject that does run the risk of developing CHD but would not be classified as such. It would be best to minimise this outcome, even if it means classifying a number of negative instances as positive. The final cost matrix can be found below:

Table 4: Cost matrix where FN instances are punished more.

		Predicted condition	
		Positive	**Negative**
Actual condition	**Positive**	0 (TP)	2.5 (FN)
	Negative	1 (FP)	0 (TN)

When this initial comparison was concluded, the two best performing algorithms were chosen for optimisation. During this process, each algorithm's hyper parameters were revised such as to squeeze as much performance out of them as possible. The algorithm that performed best after this rigorous process was chosen as the algorithm of choice for the rest of this project. That, however, does not mean this will be the final model. There is still more performance to be gained.
The last step before arriving at the final working model is the testing of so called meta classifiers. For this phase of testing both Bagging and Boosting techniques were applied to best performing algorithm thus far with the intention of creating an even better performing model. This concludes the final step of the ML model construction process.

2.3.2 Interfacing

A working ML model which can classify new instances as being or not being at risk of CHD is nice, but how does one actually go about interacting with this model to achieve such classification? This is another point where Weka comes into play. Apart from it's GUI workbench, Weka also provides an API for building models and classifying new, unknown instances. This API can be interacted with using the Java programming language. Using these tools a wrapper for our ML model was build in Java using Gradle for project management and the Weka API for classifying new instances based on our existing model. All this functionality has been neatly packed into a single JAR file which can be run using one's CLI of choice, as long as a Java install is present. This Java program, as well as all other files produced for the duration of this project, can be found here.

3 Results

The methodology described above generated a plethora of results, which will be showcased in this chapter.

3.1 Data acquisition & preprocessing

During the data acquisition and preprocessing phase the data has been made more insightful using a series of visualisations. A number of these visualisations have been described below.

3.1.1 Correlation

Different variables within a dataset may be correlated with each other. This could potentially pose problems for some ML algorithms since these may simply assume that all attributes are independent from each other. It is therefore important to be aware of whether there are correlations present within the dataset used in this project.

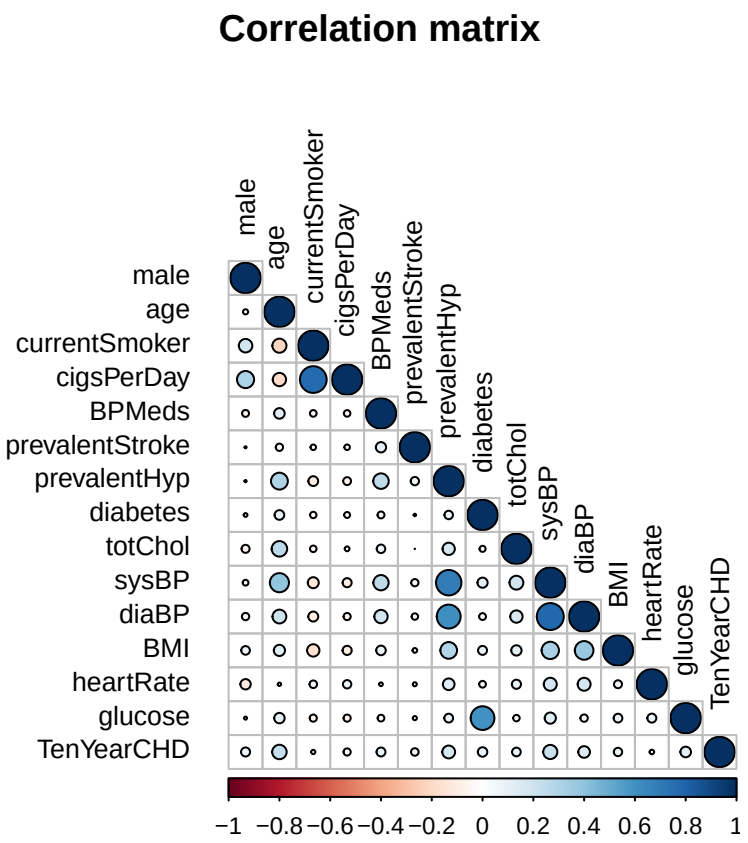

Figure 1: Matrix of degree of correlation between variables.

In this correlation matrix the size of each circle indicates the strength of a given correlation. When the circle is coloured a darker shade of blue it indicates a positive correlation, while a dark shade of red illustrates a negative correlation. Therefore the lighter shades of colour in between demonstrate the lack of a correlation.

When looking at the correlation matrix a few data points immediately jump out. The first one to look at is the strong positive correlation between glucose and diabetes. This correlation has been studied extensively to the point where it is essentially common knowledge that high glucose levels correlate with diabetes. Seeing this correlation within this dataset as well serves as a nice affirmation on the quality of the data.
Another fairly obvious correlation can be found between whether a certain subject is a smoker and the number of cigarettes one smokes per day. These two attributes serve a very similar purpose, which is something to keep in mind during the ML training process.

A very similar situation to the one just described has arisen between systolic and diastolic blood pressure. Since the former refers to the amount of force put on the arteries as the heart beats and the latter to the amount of force on the arteries when the heart is in a state of rest, it might be viable to merge these two attributes if the situation requires it.
Apart from correlating with each other the two variables named above correlate with hypertension as well. This too is a fairly obvious correlation since hypertension denotes the subject has high blood pressure.

3.1.2 Principal Component Analysis

When trying to determine whether our data clusters together one might be inclined to create a scatter plot. This is a valid approach when trying to compare two different variables. Comparing three different variables would already be slightly more challenging since this would require a three dimensional plot. This would become physically impossible however when trying to compare four or more variables, since our mortal souls are limited to a mere three dimensional space. This is where Principle Component Analysis (PCA) comes in. PCA is a technique where the dimensions (variables) are reduced by converting them into Principle Components. These PCs are ordered by their impact on the variation of the data, where PC1 has the highest impact. The first two PCs will be plotted against each other, where every variable has an arrow indicating it's impact on these first two PCs. Since the plotted PCs have the highest impact on the data's variance, and the biggest arrows have the largest impact on the PCs, larger arrows indicate more significant variables. Furthermore arrows pointing the same way indicate positive correlation, while opposing arrows indicate negative correlation.

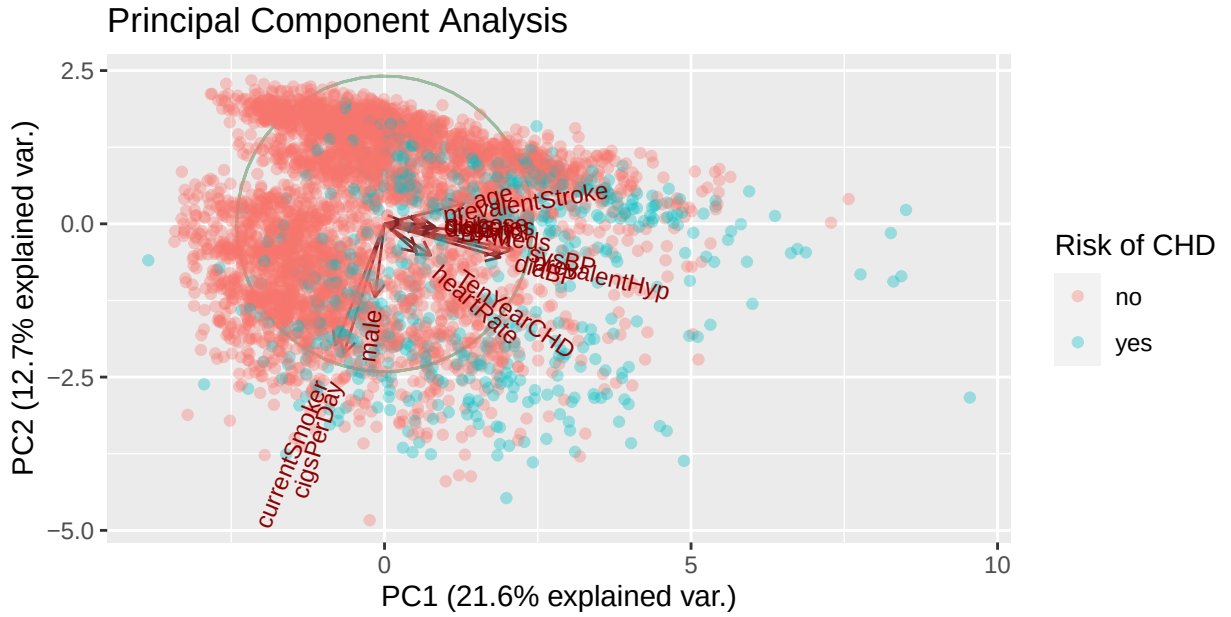

Figure 2: PCA plot charting PC2 against PC1.

In the plot above we can clearly see a contrast between the location of subjects with and without ten year risk of CHD. The subjects with risk of CHD cluster further along the x-axis than subjects lacking this risk. Since PC1 (x-axis) accounts for the larger amount of variation in our data (21.6%) differences along the x-axis are to be considered more significant than differences along the y-axis. We do, however, see some clustering along the y-axis as well.

3.1.3 SMOTE

In an effort to balance out the instances of each of the class variable's labels SMOTE was applied. The initial imbalance and later parity can be observed in the form of this barplot.

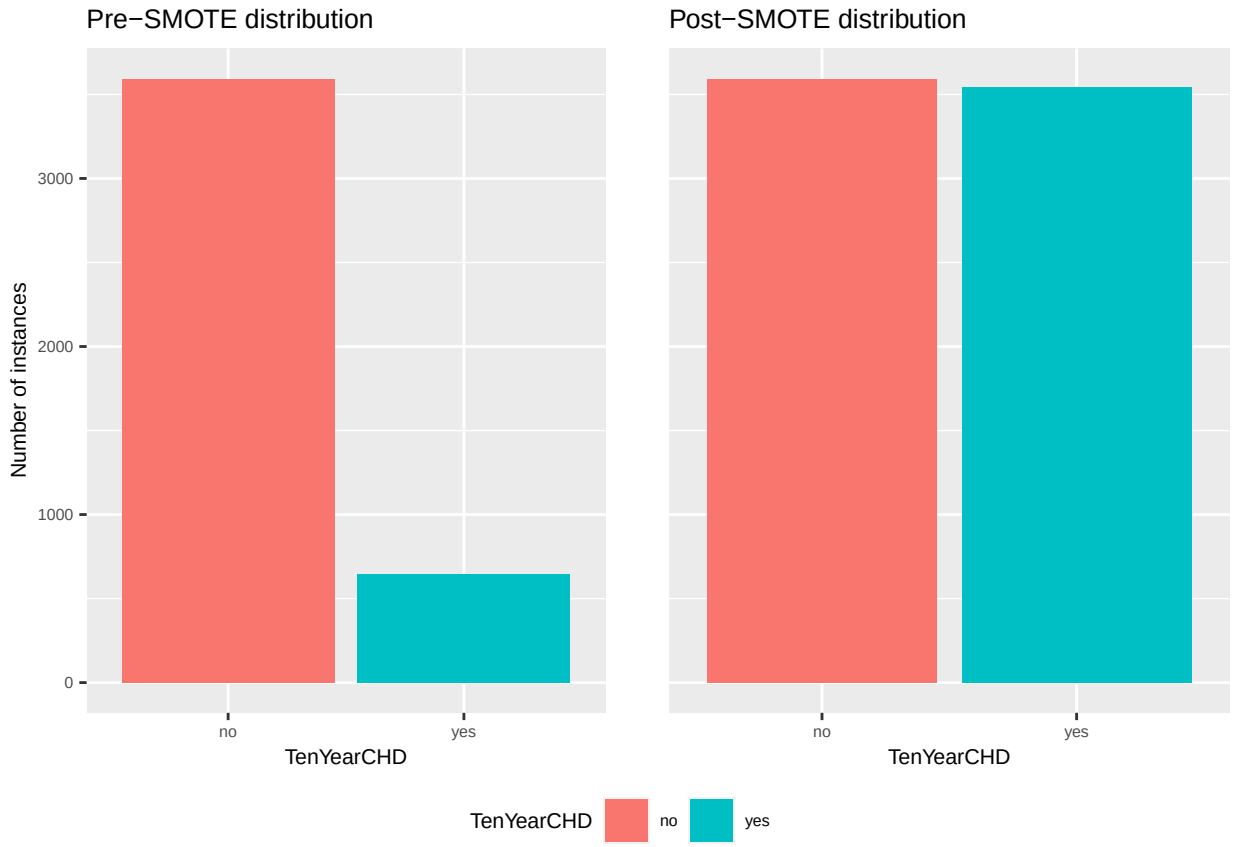

Figure 3: Distribution of class variable pre-SMOTE and post-SMOTE.

As can be observed in the pre-SMOTE distribution found on the left hand side, the number of instances labelled `no` outnumbered the `yes` labelled instances multiple times over. By synthetically creating more instances labelled `yes` these two classes were very nearly equalised, with the negative label still slightly outnumbering the positive label.

3.2 Validation & Performance

This chapter encapsulates the process of measuring and comparing the performance of different ML models. The data found in this chapter is presented using tables.

3.2.1 Default settings

As an initial start all of the chosen algorithms were compared to each other using every algorithm's default settings in Weka. Results for this initial test can be found in the table below.

Table 5: Comparison of ML algorithms using default settings using 10-fold cross validation and using ZeroR as a baseline.

Algorithm	*ZeroR*	*OneR*	*J48*	*RandomForest*	*IBk*	*NaiveBayes*	*SimpleLogistic*	*SMO*
Elapsed time testing	0.00	0.00	0.00	0.03 ∘	0.12 ∘	0.00 ∘	0.00	0.00
Accuracy	50.36	80.28 ∘	74.83 ∘	84.23 ∘	77.06 ∘	62.33 ∘	67.36 ∘	67.17 ∘
False Negative Rate	0.00	0.09 ∘	0.26 ∘	0.17 ∘	0.31 ∘	0.22 ∘	0.33 ∘	0.36 ∘
F-measure	0.67	0.82 ∘	0.75 ∘	0.84 ∘	0.75 ∘	0.67	0.67	0.66
ROC Area	0.50	0.80 ∘	0.77 ∘	0.92 ∘	0.77 ∘	0.70 ∘	0.73 ∘	0.67 ∘

∘, • statistically significant increase or decrease

When taking each metric into account we see that both the OneR rule as well as RandomForest perform quite well, but they are far from perfect. They sport a false negative rate of 9% and 17% respectively. Although this is quite a bit lower than most of the other algorithms, it is still quite dubious to use an algorithm which classifies over $\frac{1}{10}$ of positive instances as negative.

3.2.2 Cost Sensitive Classification

To remedy the issue of false negatives we can apply a the CSC technique mentioned before. This classifier allows one to apply a custom amount of cost to values in the confusion matrix. In this case false negatives are a great sin, so a cost of 2.5 was applied. This is over twice as high as the default cost of only one that applies to false positive values.

Table 6: Comparison of ML algorithms using default settings and cost sensitive classification and 10-fold cross validation and using ZeroR as a baseline.

Algorithm	*ZeroR*	*OneR*	*J48*	*RandomForest*	*IBk*	*NaiveBayes*	*SimpleLogistic*	*SMO*
Elapsed time testing	0.00	0.00	0.00	0.03 ∘	0.12 ∘	0.00 ∘	0.00	0.00
Accuracy	50.36	80.28 ∘	74.65 ∘	80.29 ∘	77.06 ∘	61.80 ∘	61.95 ∘	67.17 ∘
False Negative Rate	0.00	0.09 ∘	0.23 ∘	0.04 ∘	0.31 ∘	0.15 ∘	0.12 ∘	0.36 ∘
F-measure	0.67	0.82 ∘	0.75 ∘	0.83 ∘	0.75 ∘	0.69 ∘	0.70 ∘	0.66
ROC Area	0.50	0.80 ∘	0.75 ∘	0.80 ∘	0.77 ∘	0.62 ∘	0.62 ∘	0.67 ∘

∘, • statistically significant increase or decrease

In these results we can see that some of our best performing algorithms, including our best: the RandomForest, take a hit in accuracy and ROC Area. However, we do get a significantly lower false negative rate in return for that hit. The two best performing algorithms that we are left with look to be OneR and RandomForest. We'll continue our analysis with just RandomForest, since optimising every algorithms would be too time intensive for this project.

3.2.3 Boosting

Boosting is an ensemble learner which builds multiple models. This is an iterative technique which builds new models based on the performance of the previously built models. It does this by assigning extra weight to instances which were misclassified before. This encourages the new model to become a so called expert at instances that were misclassified by earlier models. This algorithm is known in Weka as AdaBoostM1. We will perform this technique with four different numbers of iterations and compare it to a baseline without boosting.

Table 7: Comparison of boosting with RandomForests using a varying number of boost iterations while using cost sensitive classification using 10-fold cross validation and using a RandomForest without boost as a baseline.

Boost setting	*no boost*	*-I 10*	*-I 25*	*-I 40*	*-I 55*
Elapsed time testing	0.03	0.42 ○	1.25 ○	2.49 ○	3.65 ○
Accuracy	80.29	85.25 ○	88.02 ○	88.84 ○	89.28 ○
False Negative Rate	0.04	0.05 ○	0.06 ○	0.06 ○	0.06 ○
F-measure	0.83	0.87 ○	0.89 ○	0.89 ○	0.90 ○
ROC Area	0.80	0.94 ○	0.94 ○	0.93 ○	0.93 ○

○, ● statistically significant increase or decrease

Boosting seems to be a treasure trove of performance improvements! We see statistically significant increases across the board. This is unfortunately at the cost of a false negative rate which is two percentage points higher. One could however argue that this is a sacrifice worth making since it does result in significantly higher values for accuracy, ROC area, and even F-measure.

3.3 Research findings

Let's take a brief look at what the final model looks like. When entering the model settings into Weka it starts off with AdaBoostM1 for which the `numIterations` is set to 55.
Next up is the use of the `CostSensitiveClassifier`. For this the `costMatrix` will be resized to 2x2, the cost of false negatives will be set to 2.5. At last the `minimizeExpectedCost` setting will be set to True.
Last but not least a classifier is passed to this algorithm as well and for that the RandomForest will be used. The `numExecutionSlots` parameter will be set to 8 for full CPU utilisation. This leaves one with a final model which performs as follows:

Table 8: Final model which utilises boosting with RandomForests while using cost sensitive classification using 10-fold cross validation.

Final Model	RandomForest
Accuracy	89.209
False Negative Rate	0.05
F-measure	0.892
ROC Area	0.942

4 Conclusion & Discussion

The contents of this chapter include a thorough discussion of the results showcased in the previous chapter.

4.1 Result based conclusions

During analysis of the correlation matrix (figure 1) it was found that the following variables seem to display a positive correlation:

- Glucose levels & Diabetes
- Current smoker & Number of cigarettes smoked per day
- Systolic blood pressure & Diastolic blood pressure & Hypertension

The contents of the PCA plot (figure 2) reaffirm these correlations as showcased by the arrows in this figure. This visualisation informed on the impact size of the largest principal components as well.

The SMOTE plot (figure 3) showed an exquisite rebalancing of the dataset's class variable. The extra synthetically created instances helped decrease the amount of bias in the data.

Testing every chosen ML algorithm in table 5 using their default settings informed on their potential, but showcased their weaknesses as well. Running this basic comparison resulted in the decision to make use of CSC in table 6 to decrease the number of false negative classifications. Ending on a spectacular increase in performance was the addition of the AdaBoost ensemble learner in table 7. All this led to the final model found in table 8.

4.2 Discussion

Considering the RandomForest model produced over the course of this project, keeping correlated attributes present in the dataset may have introduced bias for these attributes into the model. Since random forests sample some features to build each tree, the information contained in two correlated attributes is effectively sampled twice. This problem is only exaggerated when the data contains a high percentage of correlated features.

Applying SMOTE to balance the dataset's class variable was a good effort to decrease bias. However, as the name suggests, synthetically created data is not quite on the same level as real world data. This artificial pool of data may therefore have influenced the final results the model produces.

When starting the process of ML algorithms, just using the default settings for each of the algorithms may have left some performance on the table. One of the algorithms dropped in this early step may have performed very well if it were optimised more carefully. Optimising that number of algorithms was unfortunately outside of the scope of this project.

4.3 General conclusions and perspective

After sifting through data, modifying the input, and testing and comparing a plethora of ML techniques, one may conclude that one can predict 10 year risk of coronary heart disease in subjects from the Framingham Heart Study fairly well, considering the performance metrics displayed by the finished ML model.

5 Project Proposal

When considering the subject matter for the minor in High Performance / High Throughput Biocomputing an obvious option to continue on this work would be to further improve the model. Better yet, an entirely new model could be made using deep learning techniques. It'd be an interesting undertaking to see if a neural network could be trained to perform just as well or even better than the model produced in this project.

On a related note, it might be interesting to see whether fewer attributes could be used while still achieving the same level of performance. This could play a significant factor when considering at home use of a model like this. Some attributes, like measurements for cholesterol levels, still require a doctor to be involved, while other attributes, like BMI, are perfect for at home evaluation. Eliminating these more difficult to obtain data points would therefore greatly improve the usability of a model.

Further improving upon the at home use of a given model would be developing an easier method of interaction with the model than having to use a CLI. For the minor in Application Design one might consider developing a web application for the classification of brand new data. This would greatly improve the accessibility and usability of a given model by patients.

Bibliography

Dileep. Logistic regression to predict heart disease | kaggle, 2019. URL https://www.kaggle.com/datasets/dileep070/heart-disease-prediction-using-logistic-regression.

Michiel Noback. *ProjectopdrachtThema9 2022*, 2022. URL https://blackboard.hanze.nl/bbcswebdav/pid-5925895-dt-content-rid-81925229_2/courses/sils.bfvt.2206.k01-2223-bin09-intromachinelea/ProjectopdrachtThema9%202022%282%29.pdf.

WHO. The top 10 causes of death. 2020. URL https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death.