Detecting Heart Diseases With Machine Learning Algorithms

David Schroeter, Luca Carroz & Paulo Ribeiro
CS-433 Project 1, EPF Lausanne, Switzerland

***Abstract*— With the recent rise of machine learning, new discoveries in the various fields of chemistry, physics, medicine, etc. have been made possible. Thanks to the never ending increase in computational power of our computers, machine learning allows us to better understand our world and to solve complex problems faster and faster. This paper shows how basic machine learning algorithms can help us find causes and predict heart diseases, one of the first cause of death across the world nowadays.**

I. Introduction

Heart disease, a global epidemic with significant morbidity and mortality, poses a formidable challenge in modern healthcare. Timely and accurate diagnosis is crucial for effective intervention and treatment, as cardiovascular diseases are among the leading causes of death worldwide. To address this, the field of data science and machine learning has emerged as a powerful ally, offering the potential to revolutionize the way we detect and predict heart disease.

In this paper, we delve into the realm of predictive modeling, leveraging well known machine learning algorithms to classify individuals as either healthy or at risk of heart disease based on an significant dataset comprising hundreds of thousands of patient records. We explore and evaluate the efficacy of three widely used algorithms: Ordinary Least Squares, Least Square Stochastic Gradient Descent, and Logistic Regression. These models are used to find hidden patterns in the data and help detect heart problems early.

Throughout the course of this research, we will present a comprehensive analysis of the models and methods employed for training, the results obtained, discussions on the implications of our findings, and a concise summary of the project.

II. Model and Methods

Our model is composed of n observations $(\vec{x_i}, y_i)$, $i = 1, 2, \ldots, n$, where $\vec{x_i}$ is a d dimensional vector, where each component of $\vec{x_i}$ contains the value of a measure on patient i (such as its age, weight, sex, etc.). The value of y_i is either 1 if patient i has a cardiovascular disease (CVD) and -1 if patient i is healthy. Our main goal is to learn from those measures in order to be able to predict with good probability whether a new patient might have a CVD or not. Through this learning, we also hope to find which measures are prevalent for predicting CSV (by example, the age, the weight, etc.).

Our problem consists of predicting the value of y given a an observation $\vec{x}$, then we are faced with a classification problem, since y takes only 2 values (1 and -1). Some ML algorithms for binary classification work better with predictions 1 and 0, and we will thus alternate with the two notations, where 0 is the same as -1.

After analyzing our dataset, we notice that most of our features are discrete values, not necessarily ordered in a logical way. Since we will use linear methods, it does not make much sense to use the data as-is. A very clever idea is to map every distinct value of every feature to a risk score, where the risk score is computed as the fraction of data samples diagnosed with a CVD over the total number of samples in the category. By example, a risk score of 0.8 for category $age > 80$ means that 80% of people older than 80 years old have been diagnosed with a CVD. Since our dataset is quite large (more than 300,000 samples, with 321 features each), we cannot take into account all features. We thus decided to analyze our data by computing linear correlation between each feature and the output y, and keep only the few features with the highest correlation. By reducing the size of the observations, we obviously throw away some useful data but we allow ourselves to train more complex models, which is a trade-off we decided to take.

Since our dataset is unbalanced across outputs $y = -1$ and $y = 1$ (ratio 10:1 between healthy and sick people), we decided to under-sample our dataset in order to balance it (i.e. achieve a 1:1 ratio). Under-sampling again makes us loose precious information, but considering the large size of observations it made more sense to under-sample rather than to over-sample, since our computational power is limited. We finally standardized our data (i.e. by centering every feature and dividing by its standard deviation), which concludes the data processing part.

With our cleaned dataset, we first designed a random model that predicts 1 and -1 with a fixed probability, independently from the input $\vec{x}$. We will use this model as a reference point to compare how the other more elaborate models are better than this one. We chose the probability of outputting 1 to be equal to the percentage of sick people in our train dataset, in order to have a meaningful reference model.

We then trained four models based on four algorithms: Least Squares, Ridge regression, Mean Square Stochastic Gradient Descent and Logistic Regression. For each of those algorithms, we use cross validation to find the hyperparam-

eters that maximize the F1-score, since this metric is more important than the usual accuracy here: our main goal is to have as few false negatives as possible (i.e. we want our model to predict that a patient is healthy whereas he/she is sick as infrequently as possible).

1) *Least Squares:* We trained a least square model with a 20-fold, a polynomial expansion of degree 1 up to 6.
2) *Ridge Regression:* We trained a ridge regression with a 20-fold, a polynomial expansion of degree 1 up to 6, and 50 lambdas ranging from 10^{-5} to 10^{2}. As explained above, we chose the best of those hyper-parameters by picking the ones that yield the highest F1-score.
3) *Mean Squared Stochastic Gradient Descent:* We trained a mean square gradient descent with a 20-fold, a polynomial expansion of degree 1 up to 4, a number of gradient descent step equal to 5000, with learning rate 0.0001. After having found the best hyperparameters, we re-trained our model with a bigger learning rate (by a factor 10) to make it converge faster.
4) *Logistic Regression:* We trained a logistic regression with a 20-fold, a polynomial expansion of degree 1 up to 4, a number of gradient descent step equal to 500, with learning rate 0.05.

III. Results

As said before, we looked at the best hyper-parameters of each model. Our algorithms returned the following results :

1) *Least Squares:* A degree of 5.
2) *Ridge Regression:* A degree of 4 and a lambda of 0.000518.
3) *Mean Squared Stochastic Gradient Descent:* A degree of 2
4) *Logistic Regression:* A degree of 3

Once we've determined the best hyperparameters for each model, we proceed to train a final model using a 20-fold cross-validation approach. This process involves training the model over 19 folds and evaluating its performance in terms of accuracy and F1-score on the remaining fold. By following this procedure, we can calculate the average (and variance) performance metrics for each model, allowing for a meaningful comparison. However, before performing this evaluation, it's essential to establish a threshold since our model's predictions are continuous values that need to be categorized within the values $\in$ [-1, 1]. To determine this threshold, we compute the performance metrics across a range of threshold values, spanning from the minimum predicted value to the maximum predicted value. For instance, Figure 1 illustrates our method for finding the optimal threshold for the Least Squares model. This rigorous approach ensures that our model's output is categorized effectively.

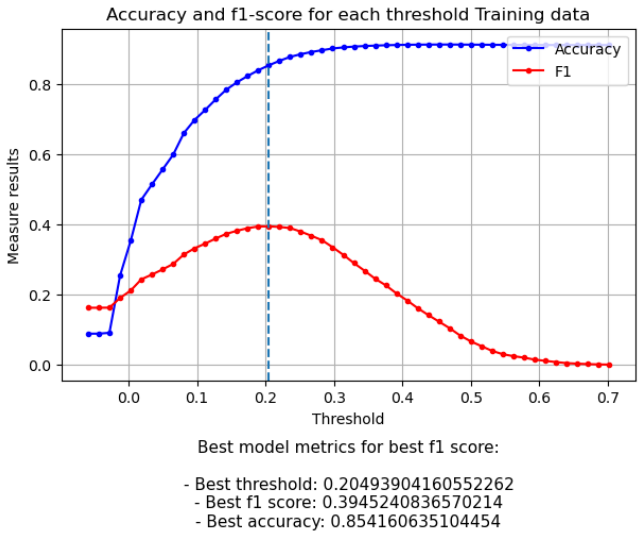

Figure 1. Accuracy and F1-score of all models

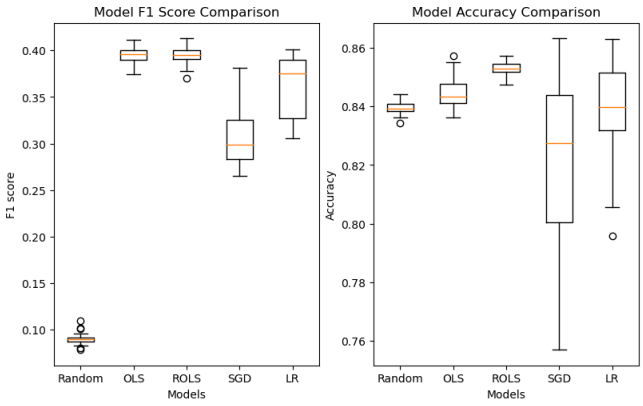

Figure 2. Accuracy and F1-score of all models

Finally, we are able to display boxplots (Figure 2) to compare all four models to each other and also with a random model.

IV. Discussion

Surprisingly enough, the best model for this task turned out to be the ridge regression algorithm, which is not originally designed for binary classification, though logistic regression was almost as good. We noticed that for logistic regression, adding a ℓ_2-regularization always worsen the predictions. We explained that phenomenon by observing that we have a very large number of data entries (a few tens of thousands) compared to the number of features (less than 50 after feature expansion). It is thus almost impossible that the model overfits the data, and regularizing it only slows down its convergence, which increase the loss.