# Machine Learning Project 1

Hamza Morchid, Ilias Merigh, Youssef Belghmi
*Department of Computer Science, EPFL, Switzerland*

***Abstract*—The increasing prevalence of Cardiovascular Diseases (CVDs), such as heart attacks, pose a significant threat worldwide. However, the advent of technologies like machine learning can facilitate early detection and prevention of developing CVDs.**

## I. Introduction

The aim of our study is to use machine learning techniques to analyze the available dataset to create a model assessing the likelihood of an individual developing MICHD based on their lifestyle factors. The data [1] comes from a health survey that collects data on the health behaviors and conditions of U.S. residents. We'll begin with crucial data preprocessing to ensure model efficiency and then implement and compare various binary classification models.

## II. Data preprocessing

### A. Data cleaning

Our dataset includes the following components:

- $x_{train}$: $(328,135$ samples, $322$ features$)$
- $y_{train}$: $(328,135$ samples, $1$ feature$)$
- $x_{test}$: $(109,379$ samples, $322$ features$)$

These datasets are quite large and contain missing values. It's reasonable to question if they contain unnecessary information for our prediction model.

### B. First approach : Manual cleaning

Our initial cleaning approach involves manual identification of important features for classifying individuals with positive MICHD, distinguishing between continuous and categorical features. We prioritized precision when selecting categorical features. For example, we chose _RFHYPE5 over _BPMEDS for hypertension, but occasionally favored more general options, such as _LTASTH1 over _ASTHNOW for asthma. Ultimately, the manual cleaning approach allowed us to reduce $x\_train$ and $x\_test$ to 49 features.

However, after testing our models, this approach didn't yield satisfactory results, as subjective cleaning could lead to the removal of important features that may not be immediately obvious, and therefore causing the model to underfit.

### C. Second approach : Statistical cleaning

Our second filtering approach consists of cleaning the features using basic statistical methods. We observed that most missing values were concentrated within specific columns or rows. Consequently, we decided to automatically remove

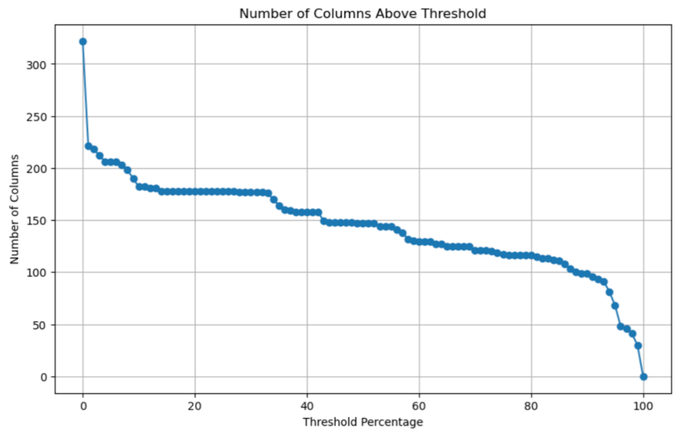


Figure 1. Number of columns with non null values above threshold

columns and rows exceeding a certain threshold of missing data. Figures 1 illustrate the number of columns to be removed based on different threshold percentages. We initially attempted to remove columns with more than 90% missing data but empirically found that the threshold optimizing our scores was 80%. We also removed constant-value columns because they don't contain useful information and cannot be standardized.

Regarding the filter of rows, we discovered that the threshold should be set between 50% and 60%, as going below this threshold led to no row removal. But in the end, we decided to keep all samples as excessive row removal had resulted in decreased scores, confirming that our model was already overfitting.

Finally, we filled in missing values (NaN) using the median of each column, which produced better results than using the mean because the median is not sensitive to outliers. To maintain consistency, all analyses were conducted on $x\_train$ and then applied to $x\_test$. After filtering, cleaning, and standardizing, the datasets retained 202 features, which accounts for 62% of the original dataset.

### D. Class Imbalance

After data cleaning, we examined class balance in $y\_test$, revealing a significant imbalance with 91.17% for -1 and 8.83% for 1. To assess this, we compared two baseline models: a random model (accuracy: 0.5, F1 score: 0.14) and a model always predicting -1 (accuracy: 0.91, F1 score: 0.0). Consequently, due to the class imbalance, we prioritized the

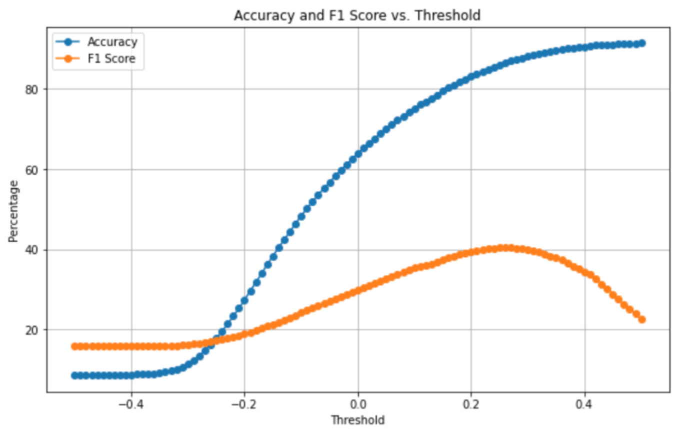


Figure 2. Accuracy and F1 score in function of the Decision boundary of the Least Squares model

F1 score as the key evaluation metric.

## III. Model

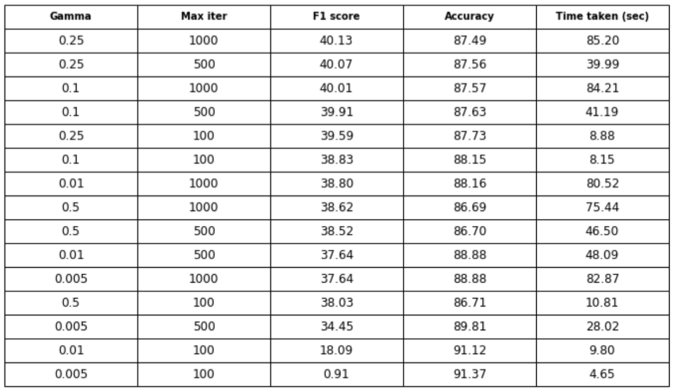
Our model's first step involves standardizing the cleaned data. Standardization, which scales each feature to have a mean of 0 and a standard deviation of 1, prevents bias from features with vastly different magnitudes. Subsequently, instead of using k-fold cross-validation, which can be computationally expensive, we employed the Validation Set method. This approach partitions the data into two disjoint sets, training on 80% of the data and testing on the remaining 20%. This method proved effective due to the large sample size at our disposal. We explored three classification approaches: Least Squares for classification, Logistic Regression with Gradient Descent, and Regularized Ridge Logistic Regression.

### A. *Least Squares for Classification*

Least Squares yielded the most convincing results as it finds optimal weights mathematically, making it both fast and efficient. We focused on determining the optimal threshold for the decision boundary, which classifies predictions as -1 or 1. By testing different threshold values, we find that the best threshold is **0.265** and achieves an accuracy of **86.71%** and an **F1 score of 40.55** in **0.37 seconds**.

### B. *Logistic Regression with Gradient Descent*

Regarding Logistic Regression with Gradient Descent, we also sought an optimal threshold for the decision boundary. Unlike Least Squares, this method iteratively finds the optimal weights, making it more time-consuming. We aimed to identify the optimal hyperparameters, gamma, and num_iters_max. The results, as shown in the Figure 3, highlight the trade-off between precision and performance. The chosen combination (gamma = 0.25, num_iters_max = 100) strikes an excellent balance between the scores and the time performance.



| Gamma | Max iter | F1 score | Accuracy | Time taken (sec) |
|---|---|---|---|---|
| 0.25 | 1000 | 40.13 | 87.49 | 85.20 |
| 0.25 | 500 | 40.07 | 87.56 | 39.99 |
| 0.1 | 1000 | 40.01 | 87.57 | 84.21 |
| 0.1 | 500 | 39.91 | 87.63 | 41.19 |
| 0.25 | 100 | 39.59 | 87.73 | 8.88 |
| 0.1 | 100 | 38.83 | 88.15 | 8.15 |
| 0.01 | 1000 | 38.80 | 88.16 | 80.52 |
| 0.5 | 1000 | 38.62 | 86.69 | 75.44 |
| 0.5 | 500 | 38.52 | 86.70 | 46.50 |
| 0.01 | 500 | 37.64 | 88.88 | 48.09 |
| 0.005 | 1000 | 37.64 | 88.88 | 82.87 |
| 0.5 | 100 | 38.03 | 86.71 | 10.81 |
| 0.005 | 500 | 34.45 | 89.81 | 28.02 |
| 0.01 | 100 | 18.09 | 91.12 | 9.80 |
| 0.005 | 100 | 0.91 | 91.37 | 4.65 |

Figure 3. Results for different hyperparameters values $\gamma$ and $max\_iters\_values$ of Logistic Regression model with Gradient Descent.

### C. *Regularized Ridge Logistic Regression*

To assess the risk of overfitting, we implemented Regularized Ridge Logistic Regression with Gradient Descent, which adds a regularization term to the cost function. We searched for the optimal hyperparameter, lambda, and found that lower regularization led to better results, indicating that regularization wasn't necessary.

## IV. Comparison with Other Libraries

We compared our Logistic Regression results to the Sklearn [2] one and identified differences in performance and runtime. We attributed these disparities to data imbalance and a fixed decision boundary at 0.5. To address this, we explored Data Oversampling and Class Weighting as solutions. Class Weighting produced results similar to ours, validating our approach.

Upon completing our tasks, we experimented with data feature filtering using the SciPy [3] library. We aimed to retain features strongly correlated with the binary target. However, the Point-Biserial Correlation method we employed yielded subpar results because it focuses on linear correlations between continuous variables and a dichotomous target, neglecting categorical data and nonlinear correlations.

## V. Summary

In conclusion, the Least Squares approach demonstrated superior performance compared to Logistic Regression methods. For further project expansion, one could explore techniques such as SVM or kNN to assess whether the dataset's inherent imbalance significantly constrains the F1 score.

## References

[1] CDC - Centers for Disease Control and Prevention. (2016). Behavioral risk factor surveillance system. Retrieved from https://www.cdc.gov/brfss/annualdata/annual2015.html

[2] Scikit-Learn. (n.d.). In Scikit-Learn: Machine Learning in Python. Retrieved from https://scikit-learn.org/stable/index.html

[3] SciPy. (n.d.). In SciPy: Open-Source Scientific Tools for Python. Retrieved from https://www.scipy.org/