Machine Learning Project 1 - Coronary Heart Disease Detection

Jan Fornieles Zaguirre, Jan Gambús Vinyeta, Cyriac Grégoire
CS-433, EPFL

***Abstract*—In this report we describe the use of a weighed and penalized logistic regression with gradient descent allowing to predict (with some precision) whether an individual will suffer a heart attack based on their answer to the 2015 BRFSS survey. Prehandling of the data, such as feature selection through correlation analysis is done prior to the learning.**

I. Introduction

The Behavioral Risk Factor Surveillance System (BRFSS) conducted a survey in 2015 with the aim of collecting data to be used to predict which patients are likely to suffer of a heart attack and which are not.
Our goal was to use this data as it was intended and determine patient's risks using Machine Learning (ML) algorithm.
We have structured our procedure into two parts, the first explains the the preconditioning of the large dataset in order to make it suitable to our ML approach. In the second we implement the weighed and penalized logistic regression with gradient descent to produce a model that completes the task with good precision.

II. Models and Methods

In our quest to predict the fatal diseases, we have written a Model, which can be found in the file FILENAME of best model, it is structured as follows:

A. Preconditioning the data

1) Categorical vs Numerical features and Cleaning the NaNs: The First step in our preprocessing was to isolate the categorical features (i.e. features that take only a couple of values, such as 'BPMEDS' the feature indicating whether the respondent takes high blood pressure medication. It takes values 1 ('Yes'), 2 ('No'), 7 ('Don't know'), 9 ('Refused') or NaN (There was no answer)).
We simply consider a categorical variable any variable that has less than 10 different values and will deal with them differently than with the numerical ones in future steps.

Secondly we observe, that there is a very large number of features which have NaN entries (Not a Number - corresponds to unanswered questions). This poses two problems, the first being that the features with a very high content of NaNs need a dimension of their own, a parameter of their own and hence increase the computational cost while giving us very little information, the second being we need numbers to implement any ML algorithm.

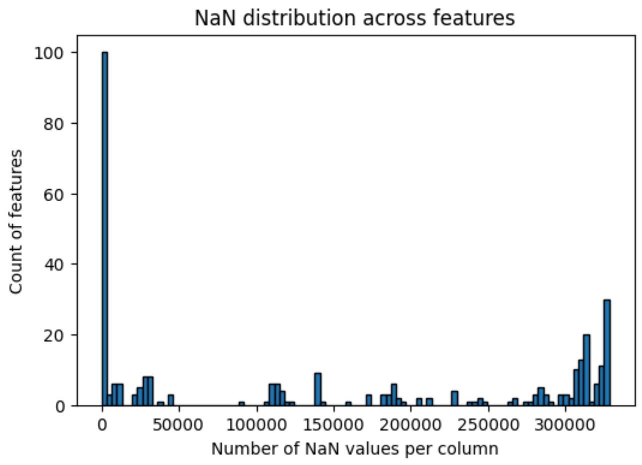

Figure 1. Number of features as a function of the NaN content.

In order to deal with the first point, we simply remove all the features that have a high content of NaNs. The threshold we found most efficient for this (by a simple grid search) was Threshold NaNs (i.e. removing all features with more than Threshold NaNs% NaNs). Figure 1 shows that this corresponds roughly to percentage of the data.
To deal with the second point we compute the median and mode, in the training dataset, of each feature that we keep and replace the NaN value with this median in the case of the numerical features and with the mode in the case of the categorical features allowing us to have numerical values. Here we use the median rather than the mean because it is robust to outliers for the case of numerical features. Since the 'average value' or 'median value' have no meaning for a categorical feature, such as 'BPMEDS' which takes values in $\{1, 2, 7, 9\}$ but could have very well been in $\{1, 2, 3, 4\}$. Choosing the mode allows to keep the structure of the data in a meaningful way.

2) Feature selection: Once we have dealt with the NaNs we will further reduce the number of features by removing those that have a "low impact" on the determination of the result. Indeed when looking at the list of features, we see that the date of interview and questions such as "do you have a landline?" can be found in it. This is useless information, that is not only increasing the computation cost but may also create noise that could impead the determination of a patients survival.

In order to avoid going through the 321 manually to select features (not to mention we are not experts in the field), we follow the two following steps. First we determine which features have many outliers. This is an indication of whether the feature in question is a reliable indicator or whether the subjects of the survey give very different (and thus almost 'random') answers. Here again we must determine a threshold of the percentage of outliers we allow to keep. Trial with various values yields that we use the threshold threshold outliers.
Secondly we would like to compute a feature-target correlation to drop the features with low correlation (such as the phone number). In order to do so we must first retouch the categorical variables. This is done via the one_hot_encode function that takes in the data and splits a categorical feature of k possible values into $k-1$ categorical features each taking values in $\{0,1\}$. This allows to give the same information since the actual value of a categorical feature does not matter but rather the "answer" to the question. (A given feature is 1, we know the corresponding answer was given. If all are 0 the last one was given.) The main advantage of this transformation is that we may now determine a correlation of a given categorical feature with the target. Indeed the correlation of a feature X with the result Y (of covariance σ_X, σ_Y) is given by:

$$\text{corr(X,Y)} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]}{\sigma_X \sigma_Y}$$

Of course we do not have information on the distribution of X and Y but an analogue formula gives the sample correlation coefficient, which requires to compute the mean value of X. This has no meaning as such. However, taking the mean after the 'One Hot Encode' transformation (now always in $[0,1]$) shows how often this answer is given. Hence we are now ready to compute the feature-target correlation for all features and drop the features with correlation bellow threshold corr target.
Finally we compute the correlation matrix to determine which features information is actually readable in another one. Here we take the features with correlation above a threshold of threshold corr (determined via trials) and keep only the one with the highest feature target correlation previously computed.

B. The Learning Model

Since the goal of this project is a classifying task, separating those who did or did not suffer of a heart attack, we have found the use of a version of logistic regression to be most efficient, i.e. optimizing the log-likelihood that our data is observed. We therefore minimize:

$$\text{L}(x) = \frac{1}{N \cdot \sum_{i=1}^{N} \eta_i} \sum_{i=1}^{N} \eta_i \left(log(1 + e^{x_i^T w}) - y_i \cdot x_i^T w \right) + \lambda \, \|w\|_2^2,$$

by using gradient descent. Note that we always consider $w = (w_0, w_1, ..., x_N)^T$ and $x_i = (1, x_0, ..., x_N)^T$. Iterating over a range of λ values we found the optimal λ to be $\lambda^* = 10^{-8}$.
Then we simply compute the probabilities of observing a 0 or 1 for the dataset as follows:

$$\mathbb{P}(y_i = 1 | x_i) = \frac{1}{1 + e^{-x_i^T w}},$$

and predict 1 when this probability exceeds the threshold value of threshold proba. This threshold value was chosen to yield the optimal F1-value.
A particularity, here is that we use weighed logistic regression. This is due to the large imbalance in the data. Indeed about 90% of values are zeroes (people who have not suffered of a heart attack) but this means that predicting more zeroes will give better accuracy. To solve this problem we put a weight η_i on each data point that makes zeros 'count much less' (about 1/9-th) than the ones. This allows us to use all the data without taking the 'lazy route' of declaring ambiguous cases 0. After testing a range of values, the optimal weights seemed to be optimal weight.

In order to get the best refined weights w, we split the data into three parts, a training set, a validation set and a testing set composed of 70%, 15% and 15% of the data respectively. It is important here that we sample each subset in a way that conserves the original composition of the data, i.e. roughly 90% zeroes and 10% ones. Talk about cross validation once it is implemented.

C. Tests that have failed or been less successful

While the main point of this report is to detail the final result of our work, we find it insightful to give a quick overview of some trials that, while not becoming our main work, have lead to improvements of the final model.

1) We call the first BalancedLogReg, it was the first acceptable version of the penalized logistic regression. It had most features above, except weights and the feature selection. Its main weakness was that we had tried to solve the data balancing problem (training and validating on datasets of 1 : 9 ones-zeroes ratio) by simply choosing randomly, 9 times as many zeroes as ones in those sets. Hence resulting in a huge waste of unused data.
2) Feature selection, was another large topic of ours. We have tried various approaches here. The minimalist approach: getting rid of the features with a high NaN content and that's all. This was good, but we improved as explained previously. We also tried handpicking some (around 30) features we would consider very important, such as 'CVDINFR4' which indicates whether a person has already had a heart attack in the past. This

seemed to simply be a much more mediocre version of previous models. We had taken out the noise, but also far too much information.

3) Finally, since this task is a classification task, we had also thought of using the Nearest Neighbours. This would have allowed some choices that a linear or logistic model could never make. Since it is well known that this is not doable in high dimension, we wondered whether using again only a few handpicked features may yield some fruits. We found to either quickly run back into the dimension problem or else that we had lost too much information.

III. Conclusion

To be written at the end.

References