Applied Statistics Project - On the predictability of life course

VEILLON, Juliette
first1.last1@xxxxx.com

ANDRU, Kilian
first2.last2@xxxxx.com

LACOUR, Xavier
first2.last2@xxxxx.com

MASSIN, Keryann
first2.last2@xxxxx.com

May 11, 2023

Abstract

Based on socioeconomic and genetic data, is it possible to predict one's health across time? The question has been discussed in many previous research without satisfying answer. This paper explains how we approached the problem. The data used for our purposes are the ones from the Health and Retirement Study led by the RAND Corporation. We first explain how me chose the global health index, using Stochastic Neighbors Embedding paradigm, before applying a specific kind of Lasso Regression and further machine learning algorithms.

Contents

1 Introduction

The work presented in this document has been realized in the framework of the **Applied Statistics Project** (projet Statapp) by the joint participation of Juliette Veillon, Kilian Andru, Xavier Lacour and Keryann Massin. It is dedicated to the question of the **predictability of life course**, a sociological question approached from a data science perspective.

At the era of overly present information, machine learning reveals links and hidden relationships between variables that used to be hard to tackle. Data Science techniques spread among experts of all fields, and social sciences see major breakthroughs thanks to artificial intelligence. The question of the predictability of life course comes in this context as a challenging one: there is an obvious relationship between age, body-fat ratio and health for instance, but how could one exploit these links and other available information for prediction purpose? For instance, is it possible to forecast one's overall health for the upcoming year based on the socioeconomic and genetic data at our disposal? That is the problem we wish to address with this STATAPP project.

Thanks to our supervisor M. Tropf, we have had access to the Heath and Retirement Study (HRS) led by the RAND Corporation [RAN18], that consists in interviews of seniors in the United States followed over several years (28 years at most). **REMPLIR**

Our main reference for this project is the Fragile Families Challenge [SLKM17]. The format of the database is similar to the HRS one: it comprises 4,242 families, who were interrogated in six waves to collect a total of 12,942 variables over 15 years. The objective was to predict six life outcomes from wave 6 using the previous waves. Several groups of researchers participated in the task and reported both their methods and results. The method which obtained the best results, with an R^2 of 0.232, is described as follows: "Our approach consists of the following steps. First, we do early house-cleaning by dropping variables with more than 60 percent missing and dropping variables with standard deviation smaller than .01. Second, we mean-impute the data and perform LASSO regressions of the outcome variables on all remaining covariates. We then drop any covariate whose coefficient is zero. Third, for the remaining covariates, we identify their originals (i.e., before mean imputation), and apply multiple-imputation using Amelia (which employs EM algorithms). We apply LASSO again for variable-selection using the Amelia-imputed dataset. When applying Amelia, we set M = 5 and pick the third dataset." The second-best method, with an R^2 of 0.229, used an XGBoost algorithm. We used these methods as an inspiration to find our own methods, regarding the structure of our dataset.

2 Presentation and cleaning of the database

2.1 Presentation

Our project is based on multiple databases, which all have the same source: the Health and Retirement Study (HRS). The HRS is sponsored by the National Institute on Aging (grant number NIA *U01AG009740*) and is conducted by the University of Michigan and focuses on Americans over 50. More than $20,000$ are included in the survey. The original goals of the survey were to help facilitate research and to guide policymakers in their decisions.

The first dataset is about participants' lives data (its formal name is RAND HRS Longitudinal File 2018 (V2) [RAN18]). It contains fourteen waves of survey for the moment. At each new wave, people from previous waves must respond again (as the data is longitudinal), and new people representative of the American society are also added. Are considered in the data the respondents themselves, their spouses, and their households. The data here revolves around the socio-economic situation of the respondents but also their health. It is outlined by more than $10,000$ variables.

The two other datasets correspond to the genetic ones **AJOUTER REFERENCE**. They are similar and just differ as one contains people with European ancestry while the other people with African ancestry. The contained genetic data are polygenic scores (PGSs). Their goal is to encapsulate genetic

facts that are linked to a phenotypic trait to try to find the latter. Here, the genetic facts in question are Single Nucleotide Polymorphisms (SNPs). They correspond to DNA sequences where only one particular gene is different from the others in a part of the population wider than 1%. To obtain a PGS, a weighted sum is made with the SNPs linked to the phenotype that one wishes to estimate. The weights are obtained with a meta-analysis done on genome-wide association studies (GWAS) which try to see the link between phenotypes and genes (but not with single genes like with the SNPs).

A crucial information to be pointed out is that our databases contain a lot of missing values. These missing values can be classified in two categories: either the individual has been interviewed during a given wave but he did not answer a question, either the individual has not been interviewed. These two cases lead to missing values of different kind: the first ones might be imputed whereas the second cannot. Indeed, if an individual has not been interrogated at a specific wave, imputing his answers is equivalent to creating them, which is nonsense.

2.2 Initial merges

To conduct our project, we first need to merge the databases. As the people who responded to the genetics survey are also included in the socio-economic one, we only need to add the first to the second by the individual identifier `HHIDPN`. However, this variable does not exist in the genetics databases, so we had to compute it by concatenating the Household Identifier `HHID` with the Person Number `PN`. Then we concatenated the two genetics bases and merged the resulting base with the socio-economic one. As the two genetic bases only differed by the individuals represented (ones with European ancestry and ones with African ancestry) but not by any of the 86 variables of interest, we decided to create an 87^{th} variable `Section_A_or_E` that takes `A` and `E` as modalities and indicates the ancestry of individuals, in order not to lose any information.
Notebook de référence : file_0_merge_Section_A_and_E

2.3 Removal of unusable variables

At this point, our database gathers $n = 42233$ individuals and $p = 15104$ variables. With a size of about 1.15 gigabytes and quadruple in RAM,our laptops could not handle the whole dataset and perform appropriate analysis. Plus, considering that the dataset presents a high missing values rate and that $p \approx n$, machine learning algorithms would not be able to converge in the current state of our data. We need to clean it.

2.3.1 Removal of Spouses' related variables

The subject of the variables always fall into one of the three categories 'Respondent, Spouse, Household'. Because Spouses' related variables amounted to around 7200, we decided to drop them all, considering the fact that 95% of spouses were also interviewed as respondents. At the end of this operation, our database is of size $(n = 42233, p = 7939)$.

2.3.2 Removal of columns with a missing rate too high

The quickest way to efficiently reduce the size of our database is to directly drop useless columns. Our data mainly corresponding to answers to interviews, some columns present a missing rate really high. We calculated for each wave the number of columns for a given missing rate, and got the graph 1 in appendix. The genetic data are available for around 35.97% of the respondents, so we decided to drop all variables that presented a missing rate higher than 654A precision is needed: the vast majority of the columns in our dataset are in fact timed realizations of one global variable. Take `R1MSTAT` for example. It appears that `R1MSTAT, R2MSTAT, ..., R14MSTAT` are the responses of individuals to the same question, say `RwMSTAT`, asked at different waves. Hence, it makes sense for these 14 variables to not be treated separately: if we delete `R12MSTAT`, we shall delete `R1MSTAT, R2MSTAT,` etc. If we keep one column, we keep all of them.
There are 156 variables like `RwMSTAT` with an average missing rate of more than 65%. It corresponds to 1577 columns in our original dataset, and we dropped them all at once. This single operation

drastically reduced the size of the database from $(n = 42233, p = 7939)$ to $(n = 42233, p = 6131)$.

Remark. During the process of cleaning the dataset, we came across a variable named `FILEVER` that were constant (`FILEVER = 'U'`) for all individuals. We decided to drop it.

2.3.3 Removal of health variables

The whole point of our project is to predict a global health index. As explained further on, we used the health variables to construct our index. To remove endogeneity issues, we decided to drop all health variables (even though we used only 27 of the 1983 available to create our index) in the main database, and save them in a separate database. At the end of this operation, our database is of size $(n = 42233, p = 4147)$. No obvious columns removal could be made for now.

3 Definition of the objective

As explained in the introduction, we want to see whether it is possible to forecast someone's health with the socioeconomic and genetic data we have about him at our disposal. We formally need a *Global Health Index*, that could summarize the whole of health-related information into one real number. Our goal will then be to attempt to predict it. This section is dedicated to the creation of this index.

We carefully selected a few quantitative health-variables, from his age and body mass index to the number of overnight stays at hospital or in nursing home he went through over a year. We also took into account qualitative variables indicating whether the respondent had had hypertension, cancer, diabete, heart disease, etc. The problem was to summarize 27 health-variables into one Global Health Index.

The scientific literature points out the benefits of Principal Component Analysis (PCA) in the search of such an index. Therefore our first attempts were to use PCA to summarize the 27 variables into one. However, the index created this way did not present the expected consistency: one's index looked random across time, and the index was awkwardly positively correlated with each health-variable, independently of the actual effect of the variable on one's health.

Hence, we looked for another dimensional reduction algorithm. The t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm has caught our attention, and we decided to apply it on our health data.

3.1 t-SNE algorithm: presentation

The PCA is designed to reduce a dataset of high dimension d into a smaller space of dimension $d' << d$ by finding the *principal components*, that is the d' maximum variance directions, and project the original data in the smaller space. Globally, this is an linear algebraic oriented approach.
On the other hand, the t-distributed Stochastic Neighbor Embedding tries to find similarities in the data using a non-linear probabilistic approach. The overall idea consists in assigning to the original data a probability measure P and another one Q to the reduced data, then trying to alter the reduced data such that the Kullbach-Leibler divergence of the distribution P and Q is minimal.

3.2 Formal details

Formally, considering $(x_1, x_2, \dots x_n) \in \mathbb{R}^{n \times d}$ the original high-dimensional dataset and the reduced dataset $(y_1, y_2, \dots, y_n) \in \mathbb{R}^{n \times d'}$, we compute the similarity of x_j to x_i by the conditional probability $p_{j|i}$ that x_i would pick x_j as its neighbor under a Gaussian distribution centered at x_i, where:

$$p_{i|i} = 0 \quad \text{and} \quad p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_k^2)}, \quad \forall i \neq j \tag{1}$$

with σ_i the bandwidth of the Gaussian kernels.
Once $p_{ij} := (p_{i|j} + p_{j|i})/(2n)$ defined, we also define a probability measure Q on the reduced dataset

following a Student t-distribution:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_k \sum_{\ell \neq k} (1 + \|y_k - y_\ell\|^2)^{-1}} \tag{2}$$

The locations of points y_i are determined by minimizing the Kullbach-Leibler divergence of P from Q, i.e minimizing

$$KL(P\|Q) = \sum_{i \neq j} p_{ij} \ln \left(\frac{p_{ij}}{q_{ij}} \right) \tag{3}$$

with respect to $y_1, y_2, \ldots, y_n$, by a gradient descent.

3.3 Performances

The main objective of this algorithm is to reflect the similarities in the original data. We directly applied the `sklearn.manifold.TSNE` implementation on our health data and checked its consistency. The results are detailed below and in appendix.

The index (denoted *GHI* for *Global Health Index* in the following) we obtained was calculated separately on each wave, only using the health information available at the current wave. Without surprise, the GHI empirically follows a Student t-distribution, and globally takes its values in the interval $[-150, 150]$.

In order to check for the robustness of our index, we conducted an analysis of sensitivity. Formally, an analysis of sensitivity consists in analyzing the robustness of an index to changes in the data or in the assumptions made.
As we will have to estimate an index based on a testing sample of our data, the first question to be answered is the following: does the GHI of an individual change when we do not take into account the whole database in the process? For instance, if one's GHI were equal to 50 when the index has been fitted on the whole database, would it still be equal to 50 if the index were this time fitted on a sample of the data? The figure 2 in appendix demonstrates the robustness of t-SNE algorithm in generating an index not too much data-dependent: the root mean squared error observed for samples of $10,000$ individuals amounts to about 27, for an index that varies between -150 and 150.
For the second test, we deleted values at random in the dataset. As in the previous analysis, we used Root Mean Squared Error to compare the index computed on this altered dataset and the real one. For instance, with 5% of random missing values, we obtained a RMSE of 17. Figure 3 shows the RMSE according to the percentage of random missing values. The robustness of the index to missing values is satisfying, as we can expect there would not be a too high share of missing values when we try to predict the outcome.

In addition, figures 4 and 5 show that the GHI is consistent: highest values of the GHI refer to very fragile health, while lowest ones refer to healthiest individuals. Indeed, an older age will generally correspond to a more fragile health, and so will a high number of overnight hospital stays for instance.

4 First machine learning procedures to format the database

At this point, the dataset we are working on still presents a few thousands variables. Considering that it approaches the number of individuals (that is, $p \approx n$), machine learning algorithms will have struggle converging, or would converge to unsatisfying solutions. Hence, we shall reduce the number of such variables by other means than deletion (every possible removals having already been previously realized).
According to the literature, among the many possible techniques to reduce dimensionality, the Lasso Regression is the suitable one. The idea is to train a Lasso Regression on (X, y), where X are the available data and y is the global health index, to come up with an estimator. The Lasso estimator has the property to shutdown to zero its coordinates when the associated variable does not present a prediction power strong enough. This technique has the advantage of being quick to train and to apply. *A priori*, the estimator we would get from this approach will not be really accurate at predicting, but

it will not matter as we are only interested in its coefficients' nullity.
The library **scikit-learn** in Python would allow us to reach our goal through the `sklearn.linear_model.Lasso` class. However this is bound to fail in our specific case, because X presents many missing values, as discussed in sections before. Hence, we have to adapt the method. For the record, we will present the three methods we thought of, the two first ones having failed.

4.1 Imputation of missing data

To still apply the lasso as discussed before, we could try imputing the missing data. Some imputers are already implemented in **scikit-learn**, such as the `SimpleImputer(strategy="mean")` that naturally imputes missing values in one column with the empiric mean of this column. The problem with this approach is that it does not take into account the inter-variable relationships in imputation at all. To overcome this issue, we would rather use the `IterativeImputer()`, which relies on the correlations and relationships between columns to impute missing values. Its inner algorithm calculates at each iteration a new imputed database $\tilde{X}_t$ according to specific constraints, and tries to minimize the difference $\Delta_t = \tilde{X}_t - \tilde{X}_{t-1}$. When this difference is small enough, the algorithm stops and the solution $\tilde{X}_t$ is returned. This resulting database would present no missing values, and imputed data would be a lot more likely than what the `SimpleImputer()` would obtain. But despite all our attempts, the algorithm did not converge in any way. This is due to the fact that X is too big and the computer had trouble processing. This idea failed.

4.2 Convex conditioned Lasso

Instead of looking for an application in two steps of the Lasso (first imputing the data, then fitting the Lasso), another idea is to directly modify the Lasso optimization program so that it handles missing values by itself, without too heavy imputations. The state-of-art algorithm for that problem is the *CoCoLasso* (which stands for *Convex Conditioned Lasso*). This algorithm badly performs when missing rates are too high, which is our case. We then cannot use it in its first form. However, further researches helped us discover a very similar estimator, that we shall now present.

4.3 Lasso with High Missing Rate

The idea of directly applying the Lasso still being interesting, our final approach is also based on it. According to [TFN19], it is possible to edit the CoCoLasso to make it more appropriate for dataset with high missing rate. Let us dive into the mathematical details.

4.3.1 Why is CoCoLasso not fitted for high missing rate

Consider a linear regression model $y = X\beta + \varepsilon$, where $X \in \mathbb{R}^{n\times p}$, $y \in \mathbb{R}^n$ are the data, $\varepsilon \in \mathbb{R}^n$ a noise and $\beta \in \mathbb{R}^p$ the estimator we wish to determine. We assume that y and each column of X are centered.
If X does not contain missing values, Lasso is applicable, and its optimization program is:

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{2n}\|y - X\beta\|_2^2 + \mu\|\beta\|_1 \tag{4}$$

where $\mu > 0$ is a regularization parameter, $\|\cdot\|_1$ is the ℓ_1 norm and $\|\cdot\|_2$ is the ℓ_2 norm. By the way, the presence of ℓ_1 norm is responsible for the solution $\hat{\beta}$ to have many null coordinates (the property we are interested in).
The equation (4) can be equivalently rewritten as following:

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{2}\beta^T S\beta - \rho^T\beta + \mu\|\beta\|_1 \tag{5}$$

where $S = \frac{1}{n}X^TX$ is the sample covariance matrix of X and $\rho = \frac{1}{n}X^Ty$ is the sample covariance vector of X and y.
In presence of missing values, This means that we can just estimate S and ρ without having to impute

the whole database, and this is in fact a much easier task to achieve. Indeed, we can construct unbiased estimators of S and ρ using the pairwise covariance, that is, $S^{\text{pair}} = (S^{\text{pair}}_{jk})$ and $\rho^{\text{pair}} = (\rho^{\text{pair}}_{j})$ where:

$$S^{\text{pair}}_{jk} = \frac{1}{n_{jk}} \sum_{i \in I_{jk}} X_{ij} X_{jk} \quad \text{and} \quad \rho^{\text{pair}}_{j} = \frac{1}{n_{jj}} \sum_{i \in I_{jj}} X_{ij} y_i \tag{6}$$

for $I_{jk} = \{i \mid X_{ij} \text{ and } X_{jk} \text{ are observed}\}$ and $n_{jk} = \operatorname{card} I_{jk}$. Thus, we can relace S and ρ by S^{pair} and ρ^{pair} in (5).
The major problem here is that S^{pair} may *not* be positive semidefinite (PSD). This would lead (5) to have no real solution and is therefore a fatal issue. The easiest way to resolve this problem is to replace S^{pair} by $\tilde{\Sigma}$, defined as:

$$\tilde{\Sigma} = \underset{\Sigma \succeq 0}{\arg\min} \|\Sigma - S^{\text{pair}}\|_{\max} \tag{7}$$

where $\|A\|_{\max} := \max_{i,j} |A_{ij}|$. Denoting $\Sigma \succeq 0$ a PSD matrix Σ, the optimization problem becomes:

$$\begin{cases} \tilde{\Sigma} & = \underset{\Sigma \succeq 0}{\arg\min} \|\Sigma - S^{\text{pair}}\|_{\max} \\ \hat{\beta} & = \underset{\beta}{\arg\min} \frac{1}{2} \beta^T \tilde{\Sigma} \beta - {\rho^{\text{pair}}}^T \beta + \mu \|\beta\|_1 \end{cases} \tag{8}$$

The system (8) is actually the optimization program solved by the CoCoLasso discussed in the previous section.

As stated before, a high missing rate can deteriorate estimations of the covariance matrix in CoCoLasso: if some pairwise observation numbers n_{jk} are very small, then the corresponding pairwise covariances S^{pair}_{jk} are quite statistically unreliable. As a result, other estimator elements can highly deviate from the corresponding elements in S^{pair}, even if their variables have few missing values. The core issue is that CoCoLasso does not account for the differences in reliability of the pairwise covariance. The next section describes how to overcome this problem.

4.3.2 Advantages of HMLasso over CoCoLasso

To fully understand the advantages oh HMLasso (High Missing rate Lasso) over CoCoLasso, let us first describe the program it solves.
Let Z be the mean imputed data of X. That is:

$$Z_{jk} = \begin{cases} X_{jk} & \text{if } X_{jk} \text{ is observed} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

We then compute the mean imputed covariance matrix, where each coefficient is weighted

$$S^{\text{imp}}_{jk} = \frac{n_{jk}}{n} \cdot S^{\text{pair}}_{jk} \tag{10}$$

Thus, $S^{\text{imp}} = R \odot S^{\text{pair}}$ where $\odot$ denotes the element-wise product and $R = (n_{jk}/n)$ is the matrix of weights. This allows us to take into account the difference of reliability of each coefficients of S^{pair}
We can then state the HMLasso optimization program:

$$\begin{cases} \tilde{\Sigma} & = \underset{\Sigma \succeq 0}{\arg\min} \|W \odot (\Sigma - S^{\text{pair}})\|_2^2 \\ \hat{\beta} & = \underset{\beta}{\arg\min} \frac{1}{2} \beta^T \tilde{\Sigma} \beta - {\rho^{\text{pair}}}^T \beta + \mu \|\beta\|_1 \end{cases} \tag{11}$$

with W the weight matrix of coefficients $W_{jk} = R^{\alpha}_{jk}$ for a fixed constant $\alpha \geq 0$.

This program had the $\|\cdot\|_{\max}$ norm replaced by the ℓ_2 norm and a weight matrix (W) added in the first optimization problem to account for the reliability of each S^{pair}_{jk}. As a result, this implementation is apt to efficiently deal with high missing rates.

4.3.3 Implementation in Python

A research in the literature did not provide us with any already implemented instance of the HMLasso in Python, so we decided to create our own.
The `cvxpy` library provided the solver, and we were able to create a class `HMLasso(mu = 1, alpha = 1)` that fits (X, y) and returns an estimator `beta_opt` useful to drop variables whose coefficient in `beta_opt` are almost 0. Below is an example of what the algorithm does and a sample of its source code:

```
from HMLasso import HMLasso
lasso = HMLasso(mu = 10, alpha = 1) # set values of mu and alpha
lasso.fit(X, y)
estimator = lasso.beta_opt
y_pred = lasso.predict(X_test) # predict X_test using lasso
```

And the source code:

```
class HMLasso():
  def __init__(self, mu=1, alpha=1):
  ...

  def fit(self, X, y):
    self.__verify_centering__(X, y)
    S_pair, rho_pair, R = self.__impute_params__(X, y)
    self.Sigma_opt = self.__solve_first_problem__()
    self.beta_opt = self.__solve_second_problem__()

  def predict(self, X):
    return np.dot(X, self.beta_opt)

  ...
```

The implementation presented a lot of difficulties, one of them regarding the fact that the underlying solver was not able to assert the positive semidefinitiveness of `Sigma_opt` due to its large size, even though the matrix was indeed PSD. As this assertion was an integral part of the resolution process, getting around this check presented many challenges, not to mention that each correction attempt required 30 minutes of simulation. In the final version of the `HMLasso()`, the user is invited to handle such errors when they happen by manually setting `ERRORS_HANDLING = "ignore"`, meaning *'ignore PSD check'*.

5 Second machine learning procedures: prediction

5.1 XGBoost for regression

5.1.1 Prior application of HMLasso

Considering the database left after the removals of unusable variables (section 2.3), we still have more than 40 thousands individuals and $p = 4147$ variables to deal with.
To reduce the number of columns, we applied the HMLasso in a specific way. Uptill now, our individuals are identified by their `HHIDPN`, and the database looks like this:

HHIPDN	$\mathbf{var}_{1,1}$	$\mathbf{var}_{2,1}$	...	$\mathbf{var}_{i,w}$	...	$\mathbf{var}_{i,14}$	...
1010	4	0	...	1	...	3	...
2020	-5	7	...	2	...	0	...
⋮	⋮	⋮	...	⋮	...	⋮	...

where $\text{var}_{i,w}$ is the i-th variable at wave w.
To see which variables were useful in the prediction of the global health index (`GHI`), we decided to look at each variable var_i without considering the wave. Indeed, it would make discutable sense to drop (respectively keep) $\text{var}_{i,5}$ without dropping (respectively keeping) $\text{var}_{i,1}$, $\text{var}_{i,2}$, ..., $\text{var}_{i,14}$. Thus, we altered the database as follows:

HHIPDN	**WAVE**	$\mathbf{var}_1$	$\mathbf{var}_2$	...	$\mathbf{var}_i$	...
1010	1	4	0	...	x	...
2020	1	-5	7	...	x	...
⋮	⋮	⋮	...	⋮	...	⋮
1010	w	x	x	...	1	...
2020	w	x	x	...	2	...
⋮	⋮	⋮	...	⋮	...	⋮
1010	14	x	x	...	3	...
2020	14	x	x	...	0	...

By applying the HMLasso on (X, y) where X is the dataset above and y is the GHI column, we are able to quickly drop hundreds of columns. After these procedures, our dataset comprises $p = 1500$ columns for $n = 42233$ individuals.

5.1.2 Gradient Boosting in a nutshell

XGBoost [CG16] is a gradient boosting algorithm that is used widely by data scientists to achieve state-of-the-art results on many machine learning challenges.
Gradient boosting is a paradigm of machine learning based on the idea of successive upgrades in the prediction function.

Let $X \in \mathbb{R}^{n\times p}$ be our dataset and $y \in \mathbb{R}^n$ the variable of interest. We look for K additive functions $f_1, \dots, f_K$ (often called *weak learners*) to predict the output y:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \quad \forall i = 1, \dots, n \tag{12}$$

In the case of XGBoost, $f_1, f_2, \dots, f_K$ are regression trees.
To learn the set of functions used in the model, we minimize the regularized objective:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{13}$$

where ℓ is a convex differentiable loss function and Ω penalizes the complexity of the model.

In practice, $f_1, \dots, f_K$ are not scalars but functions hence we cannot use traditional optimization methods. Instead, the model is greedily trained. Formally, if $\hat{y}_i^{(t-1)}$ is the prediction given by $\phi_{t-1} = f_1 + f_2 + \cdots + f_{t-1}$, we find f_t by minimizing:

$$\mathcal{L}^{(t)}(\phi) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{14}$$

By proceeding recursively, we obtain at last a suitable approximation $\tilde{\phi}$ of the solution of (13).
XGBoost is sparsity-aware, which means that missing values are not a problem for it, and distributed, which explains why it reveals itself extremely fast to train.

5.1.3 Blabla

The major issue we failed to solve consistently was coping with the fact that some individuals have not responded to every waves. It is a problem because, as a result, some columns present missing values that shall not be imputed: every variables are present for every individuals, even though some individuals have not been interviewed for some variables. We decided to perform our machine learning algorithms on the reduced dataset comprising only individuals present in each wave. These indiviuals amount to $n = 3396$.

To separate the effect of socioeconomic data, genetic data and previous `GHI1, GHI2, ..., GHI13` on the final health index `GHI14`, we performed XGBoost Regression using different data. Here are the results:

Data	R^2 **coefficient**	**standard deviation**
all data	0.360	0.031
only precedent ghi	0.324	0.027
only socioeconomic data	0.216	0.022
only socioeconomic and genetic data	0.207	0.018
only socioeconomic data and precedent ghi	0.374	0.022

Table 1: XGBoost regressors - results over 100 simulations

We obtained the results above by experimentally conducting cross-validation to select the best values of the following hyperparameters:

- `eta`: the learning rate. We tested $\eta \in \{0.1, 0.05, 0.03, 0.01\}$
- `lambda`: the coefficient for ℓ^2 regularization. We tested $\lambda \in \{1, 0.5, 2\}$
- `alpha`: the coefficient for ℓ^1 regularization. We tested $\alpha \in \{0, 0.5, 1\}$
- `max_depth`: the maximum depth of each tree. We tested each value from $\{3, 4, 5, 6\}$.
- `n_estimators`: the number of weak learners (alias K). We tested each value from $\{50, 100, 200\}$.

We were able to calculate a standard deviation by reconducting each simulation 100 times and randomly change the train/test split at each simulation. R^2 column gives the mean of the empirical values.

A Appendix

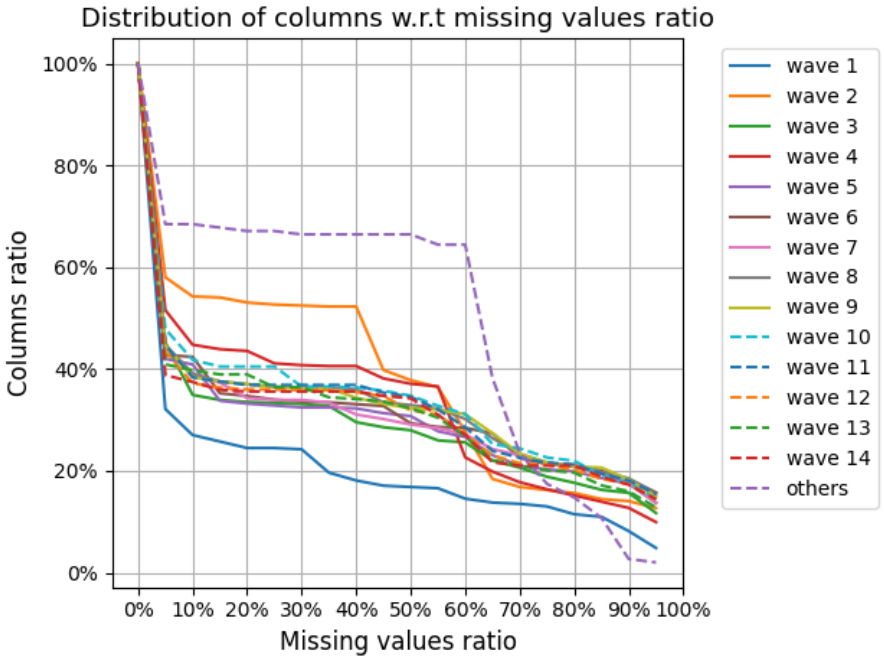

Figure 1: Distribution of columns with respect to missing values ratio

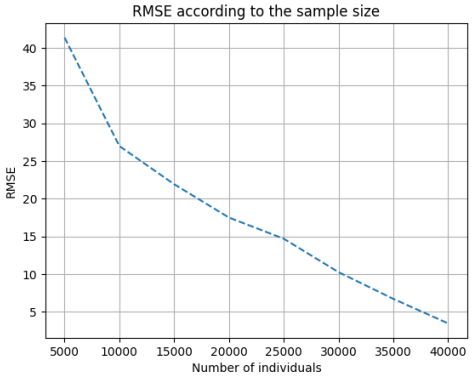

Figure 2: RMSE according to the sample size

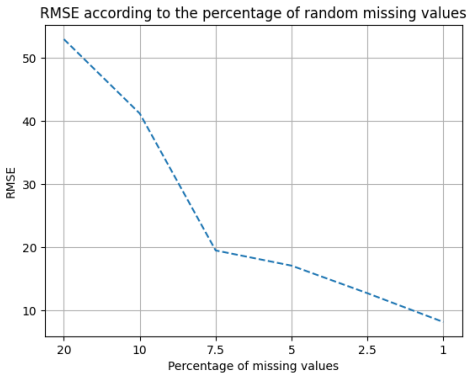

Figure 3: RMSE according to NaN percentage

Figure 1 is meant to be interpreted like this: among the variables collected from the interviews in wave 2, 40% of them present more than 45% missing values.

Figure 2 plots the root mean-squared-error observed between the actual index and the index determined on a sample of the data, size of which being informed on the x axis. Figure 3 gives the same information for each missing values rate.

Figure 4 describes how the GHI is changed when a certain factor is present. The category 0 stands for *no such factor*, category 1 for *there is this factor*, and other categories express a higher level of the current factor. For instance, `DIAB = 0` means that the individual does not present diabete. `CANCR, BACK, HEART, LUNG` stand for the presence of cancer, of pain or health problems in the back, the heart, or the lung. Overall, what we see was expected: when some one has arthrose problems (`ARTHR`), his general health is expected to be lower, hence his GHI to be higher. The unique variable that

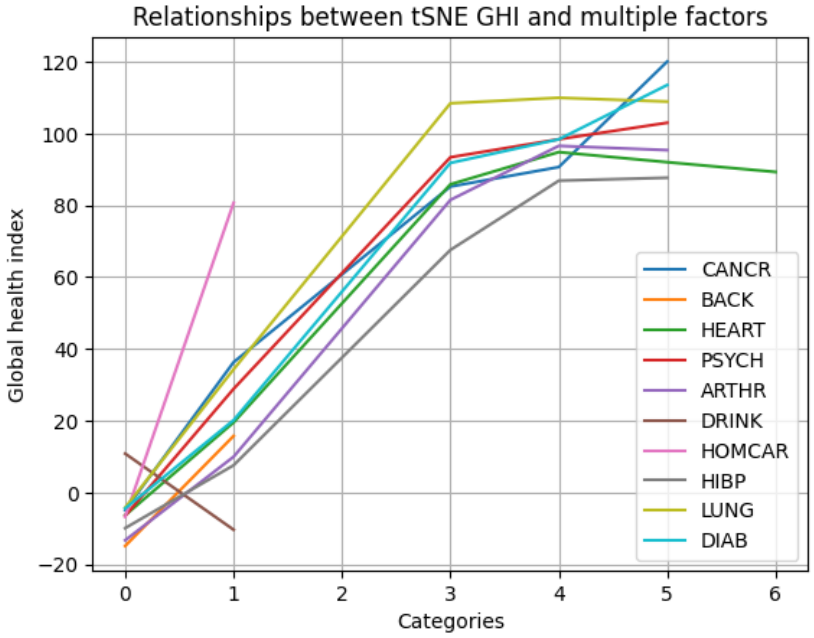

Figure 4: Relationships between the global health index and multiple factors

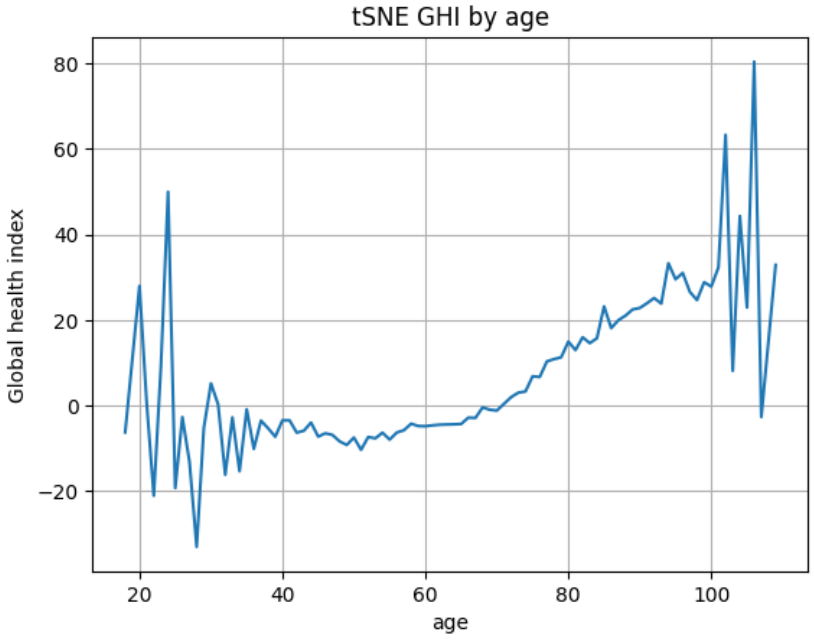

Figure 5: Relationship between the global health index and age

behave differently is `DRINK`, that describes how much one drinks alcohol. But the deviation is marginal.

Figure 5 highlights the expected fact that someone's health is better when he is young.

References

[CG16] Tianqi Chen and Carlos Guestrin. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016.

[RAN18] Rand hrs longitudinal file 2018 (v2). `https://hrsdata.isr.umich.edu/data-products/rand`, 2018.

[SLKM17] Matthew Salganik, Ian Lundberg, Alex Kindel, and Sara McLanahan. Fragile families challenge, 2017.

[TFN19] Masaaki Takada, Hironori Fujisawa, and Takeichiro Nishikawa. Hmlasso: Lasso with high missing rate, 2019.