Predicting age and survival using pulse wave analysis measurements and the tools of machine learning

Alan Le Goallec

December 10, 2020

1 Abstract

In the following, I present 30 age and survival predictors I built on 235,241 pulse wave analysis [PWA] records collected from 207,932 UK Biobank participants aged 39-72 years. PWA is a measure of arterial stiffness that relies on measuring the change of blood volume in the finger using an infrared sensor as the pulse wave propagates through it. I built models on both the raw PWA time series (convolutional neural networks [CNNs]) and the scalar features summarizing the raw data data (elastic nets, gradient boosted machines [GBMs] and fully connected neural networks). I tuned the models using Bayesian hyperparameters optimisation. I found that I could predict age with a testing R-Squared value of 38.0+-0.5% and survival with a testing concordance-index [CI] of 71.4+-0.9%.

In terms of age prediction, I found that non-linear models (e.g gradient boosted machines) significantly outperformed linear models (elastic nets). I also found that leveraging the raw data yielded more accurate age predictors than relying on the sole summary scalar features. I also found that leveraging the time-dependent aspect of the raw signal with a CNN yielded a better predictor than treating the 100 time steps of the raw time series as 100 scalar predictors and training a GBM on these features.

In terms of survival prediction, I found pulse wave analysis records to be poor lifespan predictors. The CI obtained by leveraging both the raw time series, the scalar features and the demographic variables (CI=71.4+-0.9%) was only marginally higher than the CI obtained by predicting survival using age alone (CI=69.5+-0.9%). This is coherent with the performance of other survival predictors built on the UK Biobank cohort[13].

I conclude that leveraging the raw time series can capture more information about arterial health than relying on the scalar features traditionally extracted to summarize the raw signal. New disease predictors built on the time series using the tools of machine learning have the potential to outperform current PWA-based disease predictors.

2 Background

Cardiovascular diseases [CVDs] are diseases of the cardiac and the circulatory system such as stroke, transient ischemic attacks, myocardial infarction, heart failure, angina pectoris [12]. CVDs

are the leading cause of death[14]. With the aging of the global population, changes in lifestyle and risk factors such as lack of physical activity, smoking, hypertension, obesity, diabetes and hyperlipidaemia becoming more common, the prevalence of CVDs is expected to increase in the coming decades[12]. Risk for CVDs can be mitigated by lifestyle changes and medications if detected early[16], and models such as the Framingham score[11] have therefore been built to predict the onset of CVDs5–7[3, 8, 30].

Different data modalities such as health and medical history, lifestyle and environment, blood assays, physical activity, family history, physical measures, psychosocial factors, dietary and nutritional information, and sociodemographics[3] have been leveraged to predict the onset of cardiovascular diseases. Pulse wave velocity analysis [PWA] is one of these modalities[5, 20]. PWA is a non-invasive measure of arterial stiffness. An infrared sensor is clipped on the finger and uses the absorbance to estimate the volume of blood circulating through the artery as a function of time. As the pressure wave initiated by the contraction of the right ventricle propagates through the artery, it stretches it. This deformation of the artery over time can be seen on the pulse wave analysis record (see figure 1). The shape of the wave contains information, such as the position of the pulse wave notch, which can be leveraged to assess arterial stiffness and vascular health[10, 25, 35]. The scalar features extracted from the wave have been used to predict cardiovascular diseases such as hypertension[20] and cardiovascular mortality[17, 22, **?**, 32]. However, to my knowledge, no work has been done to fully leverage the raw time series as a predictor of cardiovascular mortality. In other fields such as ECG analysis, predictors using machine learning algorithms such as convolutional neural networks and recurrent neural networks (LSTMs, GRUs) were able to significantly outperform the previous predictors built on scalar features extracted from the raw ECG signal[29, 28, 4]. I will therefore test if leveraging the raw PWA data with the tools of deep learning also significantly outperforms the control models I will build on the associated summary scalar features.

3 General approach

The goal of my project was to estimate how much information regarding survival could be extracted from the biomarkers of arterial health that are PWA records. Specifically, I wanted to test if leveraging the raw time series outperformed models built on the more commonly used summary scalar features.

The most direct way to estimate the value of a biological predictor to predict survival is to perform survival analysis. Survival analysis is however challenging for two reasons. First, it is technically challenging as it relies on unorthodox modeling, loss functions and performance metrics, as described in further details under "Approach". Second, the sample size available to build survival models is usually several folds smaller than the initial sample size of the dataset, as only a small subset of the participants from whom PWA were recorded have died. For these two reasons, it can be hard to estimate whether failure to successfully predict survival is due to the inherent difficulty of the task or to technical errors in the modeling of the problem. I therefore decided to first tackle a simpler version of the problem to ensure that the models I was building were able to extract information from the PWA records.

When clinical trials are performed to assess if a drug is likely to extend the life expectancy of the patients, comparing the survival rates of participants in the case and the control group is ideal

but not always feasible, for example because of the time constraints. In these situations, softer clinical endpoints such as the onset of particular disease or the levels of specific biomarkers can be monitored instead. Building on this idea, I considered predicting age-related diseases such as hypertension or heart failure. This approach would have allowed me to circumvent the first challenge of survival prediction that is its technical difficulty, as disease prediction is a comparatively simpler task of binary classification. However, it would not have addressed the problem of the smaller sample size, as the prevalence of cardiac failure or angina pectoris is for example very small. I believe that disease prediction is a promising future direction for this project, but I have decided on a third approach.

The concept of biological age is tightly linked to survival. As time passes damage accumulates in our bodies, but this accumulation can happen at different rates for different individuals. Accelerated agers are patients whose body is more damaged than the average body of someone their age. By training a model to predict the participants "chronological age" (the time since they were born), we can teach the model to recognize what an average human looks like at different stages of its life, and we can therefore later use it to detect accelerated aging, which is associated with shorter life expectancy. This third approach is valuable because performing a regression task is technically easier than performing survival analysis, and because chronological age was measured for every participant in the UK Biobank cohort, so I can leverage the full sample size of the dataset.

In conclusion I decided to first train models to predict age to ensure that the algorithms were able to leverage the biological information captured by the PWA records, and that I would only attempt to perform survival analysis after confirming that my pipeline was correctly performing the age prediction task.

4 Methods

4.1 Hardware and Software

I performed the computation for this project on Harvard Medical School's supercomputer O2. CPUs and GPUs (Tesla-M40, Tesla-K80, Tesla-V100) are available on O2 via a Simple Linux Utility for Resource Management [SLURM] scheduler. I coded the project in Python[33] and submitted the jobs on O2 using Bash[15] scripts. I leveraged python packages such as NumPy[24, 34], Pandas[23], Hyperopt[6], TensorFlow[2] and Keras[9]. For age prediction, I used scikit-learn library's ElasticNet and MLPRegressor (neural network) functions, lightgbm's[19] LGBMRegression function, and I built the CNN architectures using TensorFlow-v2. For survival analysis, I used scikit-survival[27] library's CoxPHFitter (elastic net) and GradientBoostingSurvivalAnalysis (GBM) functions, PyTorch[26] and torchtuples' libraries MLPVanilla function (neural network), and I built the CNN architectures using TensorFlow-v2. I adapted the loss function for the CNN survival model from a tutorial by Sebastian Pölsterl: https://github.com/sebp/survival-cnn-estimator/blob/master/tutorial_tf2.ipynb.

4.2 Dataset

I leveraged the UK Biobank[31] [UKB] cohort. The UKB dataset is a large biobank collected from 502,211 participants in the United Kingdom that were aged between 40 years and 74 years at enrollment (starting in 2006). The initial data collection at enrollment is described as "instance

0", and some of the participants were followed up so more data was collected in later instances (instances 1, 2, and 3). Pulse wave analysis data was collected for 170,742 participants in instance 0, 20,229 participants in instance 1, 41,427 participants in instance 2 and 2,843 participants in instance 3, amounting to a total of 235,241 samples for 207,932 participants, aged 39-72 years. The UKB cohort also contains mortality data for 30,203 participants. The sex ratio of the cohort is balanced. The majority of the participants are of White British ethnicity (94%).

4.3 Preprocessing

4.3.1 Features extraction

I extracted the PWA data from the UKB dataset, where it is stored as a "compound" dataset (a string encoding the full time series is stored as one of the columns). I noticed that the first time step value was the same for every participant, so I removed it. The resulting dataset is a 100 time steps time series for each of the participants. Ten samples are displayed in Figure 1.

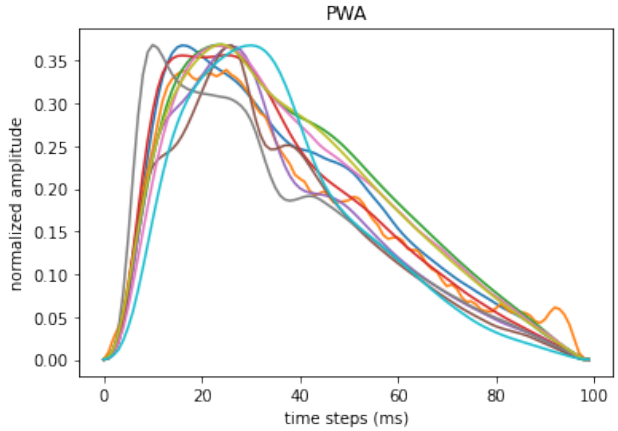

Figure 1: Example of ten pulse wave analysis samples after preprocessing

I also extracted the eight PWA summary scalar features shared by UK Biobank: Absence of notch position in the pulse waveform, Position of pulse wave notch, Position of the pulse wave peak, Position of the shoulder on the pulse waveform, Pulse rate, Pulse wave Arterial Stiffness index, Pulse wave peak to peak time and Pulse wave reflection index.

I extracted three demographic variables: age, sex and ethnicity. I one hot encoded sex and ethnicity. For sex, I used the genetic sex when this information was available (99% of the samples). To obtain a greater precision on the participants age (UKB reports age as an integer), I recalculated it from the participants' date of birth and the date at which they attended the assessment center.

Finally, I preprocessed the survival objects in preparation for the survival analysis. A survival object is a tuple with two variables for each participant. The first variable is called the "status". It is a boolean describing whether the death of the individual has been observed. The second value is called the "time". It is a strictly positive real number describing the time between the data collection (in my case, the PWA data) and the follow up (in my case, the date at which the participant was dead or the death registry was last updated). Participants who did not die before the last update of the death registry are considered "right censored".

4.3.2 Data split

I split the dataset into a training fold (80%), a validation fold (10%) and a testing fold (10%). To prevent information leaking, I ensured that all the samples from each participant were kept in the same data fold. The proportion I chose for the training, validation and testing sample sizes is slightly uncommon, as traditionally the validation and testing set each represent 15-20% of the data. I used 10% for two reasons. First, the initial sample size is very large. Under these conditions, even a small percentage of the dataset is enough to evaluate the performance of the model and tune the hyperparameters, so the bulk of the dataset should be leveraged to better train the model. Second, this 10%-based split generalizes well into a 10 folds cross-validation, which would be needed in the future if I were to generate an unbiased testing prediction for each sample, with the aim to leverage the full sample size of the dataset when testing the association between accelerated aging and genetic and environmental correlates.

4.3.3 Scaling of input and target variables

I normalized all the scalar features by centering and scaling them, using the training means and standard deviations to prevent information leaking "from the future". Normalizing the predictors is important because the elastic net regularization is based on penalizing large regression coefficients. Predictors with large mean and standard deviation can have a huge impact on the regression without having a large regression coefficient associated with them, so they will be less regularized than predictive features with small mean and standard deviation. Normalizing the features correct for this bias. A similar effect can be observed for neural networks when weight decay is used as a regularization technique. In contrast, GBMs are feature scaling invariant since the splits are based on the percentile of the feature distribution. Normalizing the target is particularly important for neural networks when performing regression, because the loss (mean squared error) scales with the square of the standard deviation of the target variable. If the scale of the target is large, the neural network is more likely to suffer from exploding gradients. If it is too small, the neural network is more likely to suffer from vanishing gradients. In contrast, survival analysis is invariant to scaling of the follow-up time.

4.4 Algorithms

4.4.1 Algorithms for scalar features

A large number of algorithms are available to perform regression: linear regression, LASSO regression, ridge regression, elastic net, random forest, gradient boosted machine, XGBoost, support vector regression, K-nearest neighbors, naive Bayes and fully connected neural networks, to name a few. These algorithms can be compared and ranked based on the trade-off they offer between bias and variance, as well as between interpretability and performance.

Elastic net: Linear algorithms such as linear regression, LASSO regression, ridge regression and elastic net are simple models that are unprone to overfitting and highly interpretable by looking at the regression coefficients. Their lack of complexity however prevents them from learning complex patterns such as non-linearities or interactions between the different scalar predictors. They are therefore high bias, low variance, highly interpretable and poorly performing models. I

selected elastic net as a representative of this category, as it represents a tunable compromise between LASSO and ridge regularizations. I planned to use this algorithm to interpret which scalar features were associated with aging and survival.

Neural network: Neural networks sit on the other end of the spectrum. Their large number of trainable weights (high degrees of freedom) makes them high in variance and low in bias, poorly interpretable but highly performing. Deep or wide neural network architectures require large sample size to be successfully trained, and their complexity make them hard to interpret (although model agnostic methods such as LIME, or computing the gradient of each input feature with respect to the loss can be considered). I selected a fully connected neural network as one of my three architectures to test how accurately I would be able to predict age and survival if I was willing to sacrifice interpretability.

I predicted age with a four hidden layers deep neural network (256, 128, 64 and 32 nodes) with ReLU activation layers. I used Adam to perform gradient descent, 16,384 as batch size and early stopping as regularization. For survival prediction, I used the same architecture. I did not perform early stopping, but I added batch normalization and a dropout rate of 10% as regularization.

GBM: Tree-based algorithms such as random forest, gradient boosted machine and XGBoost offer an attractive compromise between the two extremes described above. CART (classification and regression trees) algorithms, such as the ones covered in class or in the third problem set, are high variance low bias algorithms. Their ability to recursively split on the data make them highly flexible, but also highly prone to overfitting. This is the reason why they are rarely used as such. Instead they constitute the building block of ensemble models such as random forest, gradient boosted machine and XGBoost. These algorithms build a large number of trees (e.g forest) and have them vote to predict the target. This ensembling strategy allows them to reduce the variance while conserving a low bias. They are also fairly interpretable, as feature importance can be estimated based on the frequency at which each feature was selected by the different trees of the ensemble. Unlike the regression coefficients of linear models, which are signed values, the feature importance of tree-based algorithms is always positive and can therefore not be used to estimate whether each scalar feature is positively or negatively associated with the target variable. Another drawback of tree-based algorithms is that they are non-parametric model whose size will grow with the training sample size (more splits will be needed to reach node purity) so they tend to become unpractical or to be outperformed by neural networks when very large sample sizes are available. I chose to use a GBM rather than a random forest, as the sequential building of the trees to incrementally improve performance by giving more weight to the samples that were poorly predicted so far suggests that it should outperform the random forest, in average. However GBMs can be very time consuming to train, and even more so to tune as they have a large number of hyperparameters. I therefore chose to use the LightGBM implementation, which makes an approximation to increase the speed of the algorithm several folds.

To predict age, I tuned the hyperparameters of the GBM as described under "Hyperparameters tuning". To predict survival, LightGBM was not available so training the GBM model was too time consuming to tune the hyperparameters. I used the default values: learning rate of 0.1, 100 estimators and maximal depth of three.

4.4.2 Algorithms for time series

Architecture:
To extract the information of the time series and predict survival, I used one-dimensional convolutional neural networks (1D CNNs). I did not test recurrent neural networks architectures such as LSTMs and GRUs because it is my experience that such architectures tend to be difficult to tune and train and lead to suboptimal performance compared to 1D CNNs.

I first built a CNN that was only leveraging the time series data. However, I wanted the model to be able to interpreted the scalar features extracted by the convolutional layers in the context of the demographic variables (sex, ethnicity, as well as age for survival prediction), so I built a second architecture with a "side" fully connected neural network branch to learn from these scalar features. To allow the architecture to predict age and survival as accurately as possible, I also fed the PWA summary scalar features reported by UKB as side scalar predictors along with the demographic variables.

I tested different CNN architectures. They were all composed of three blocks: a convolutional block, a side block and a dense block.

The convolutional block takes the time series (100 time steps) as input and generates scalar features from them. It is composed of several one-dimensional convolutional layers with a kernel size of three, a stride of one, ReLU activation and zero padding to conserve the dimension of the input. After each convolutional layer I used a max pooling layer to halve the temporal dimension (kernel size of two, stride of two). I also added an optional batch normalization layer (see hyperparameters tuning). I tested different number of convolutional layers and ended up using four of them, with respectively 32, 64, 128 and 256 filters. The last layer of the convolutional block is a global max pooling layer to convert each filter into a scalar feature, that are then concatenated to the output of the side block before being fed to the dense block.

The side block consists of a single single dense layer whose input are the scalar features used as side predictors. The dense layer has a number of units of approximately half the number of inputed scalar features (closest power of two), and uses ReLU as activation function. Its output is concatenated to the output of the convolutional block before being fed to the dense block.

The dense block is composed of several dense layers with ReLU as activation function. These dense layers are regularized with two methods. (1) Their weights are L2 regularized (kernel regularization, also known as weight decay) and (2) the dense layers are followed by a dropout layer. Optionally, a batch normalization layer is inserted between the dense layer and the batch normalization layer. I tested different number of dense layers and ended up using two of them with respectively 128 and 64 units. Finally, I added a final dense layer with a single node and linear activation to generate the prediction (age for age prediction and log hazard factor for survival analysis).

The architecture for survival analysis using both the time series as well as all the scalar features available as input can be found in the supplementary figure 1. Alternatively, please refer to the code for the details of the architecture (functions _generate_age_CNN and _generate_surv_CNN).

Compiler:
To predict age, I used mean squared error as the loss, and root mean squared error as the metric. I

used Adam as the optimizer, with gradient clipping (maximum L2 norm of 1.0) and a batch size of 32,768. I used two callbacks: early stopping and reduce LR on plateau. Early stopping is a regularization method that prevents overfitting by stopping the training after the validation performance starts worsening. I monitored the validation loss with a minimum delta of 0 and a patience of 100 (because of the effect of batch normalization further detailed under "Challenges"). The callback restored the best weights after early stopping. Reducing the learning rate on plateau by a factor of ten and a patience of ten helps the model converge if the learning rate was initialized with an excessively large value. In theory Adam is already dynamically tuning the learning rate but in practice I found that the reduce learning rate on plateau callback could help with convergence. Interestingly, the default metric monitored by this callback is the validation loss, which is a mistake. The learning rate should be reduced when the training loss is not decreasing. A worsening validation loss is a sign of overfitting, a problem that will not be addressed by reducing the learning rate. (If anything, it could actually help the algorithm find an even lower minima in terms of training loss in the long run and worsen the overfitting).

4.5 Survival analysis

4.5.1 Difference between survival analysis and regression or classification tasks

As mentioned above, survival analysis is technically more challenging than regression or classification tasks for two main reasons. First, we are not only predicting whether someone will die or not. That would be an easier classification task, but on the long run, everyone dies. So the question is: When will a participant die? It would therefore be tempting to perform a linear regression –another comparatively easier prediction task– to predict the duration of survival after the data was collected from the participant. However, if we adopted this approach we could only train the regression model on the samples for which the participant already died, which would drastically reduce the sample size we could leverage (in my case 6,000 samples instead of 230,000 samples). There is however information contained in the samples for which the participant has not died yet. For example, if there was a follow-up five years after PWA data was collected, and the participant had not died, we might not know their exact date of death, but we already know that this person has survived five years. This is called a "right-censored" sample, and survival analysis models are designed to be able to leverage the information contained in these censored samples. Their complexity come from the fact that they need to perform both a form of classification (did the participant die?) and regression (how long did the participant survive?) at the same time.

4.5.2 The Cox Proportional-Hazards family of survival models

Here I will present the Cox Proportional-Hazards [CoxPH] family of survival models. CoxPH models are semi-parametric models that rely on the assumption of proportional hazard: all participants share the same baseline risk function, that increases over time. The difference of risk between two participants is a scalar factor that does not depend on time. So if CoxPH estimates that the participant A has a higher risk than participant B two months after the assessment, it also estimates that participant will have a higher risk than participant B ten years after the assessment. It also estimates that the risk ratio between participant A and B remains the same over time. So CoxPH models cannot learn to represent situations in which A has a higher risk than B at first, but B is at higher risk than B on the longer term. This is a strong assumption, but it allows CoXPH

models to uncouple (1) the baseline hazard function that is a non-parametric function of time, but does not depend on the predictors since it is shared between all participants from (2) the risk factor for each participant, which does not depend on time, but is a parametric function of the predictors. CoxPH models historically ignored the estimation of the baseline function at first, and focused on estimating the parametric function for the risk factors (Breslow later designed an estimator for the baseline hazard function [1, 21]).

4.5.3 CoxPH: Loss function

As a consequence of the above, the CoxPH loss function does not penalize predicting the wrong average survival time. Instead, it only penalizes the attribution of low risk factors for participants who died quickly, and inversely. In that, it is very similar to a rank-based loss function, but it also quantifies the error: if two participants died after similar follow-up time, the cost for wrongly attributing a lower risk factor to the one who died first is low. This comparison cannot be made between two right-censored individuals since one cannot determine which one will die first. It can easily be made between two deceased participants. Finally, the comparison can only be made between a right-censored and a deceased participant if the follow-up time for the right-censored participant is larger than the follow-up time for the deceased participant. In that case, the loss function penalizes the pairs for which the right-censored participant has a higher risk factor than than the deceased participant, despite the deceased participant dying first.

The paragraph above provides intuition about what the loss is measuring. Formally and mathematically, the loss is called the negative partial likelihood and can be found in the following paper: [18]. The partial likelihood is the product of the likelihood, for each death event, that the deceased participant was the one that actually died considering the risk scores for all the other participants that were still alive at this time. Because of the CoxPH assumption, this likelihood is very similar to a softmax: it is the ratio between the risk factor of the deceased participant, over the sum of the risk scores for all the participants that were still observed (neither dead nor censored) before this death event. For example, let us assume we initially had 100 participants, and the 20th death event was observed after 37 months. The 20th factor in the partial likelihood computation is the ratio between the risk factor for the 20th dead participant, over the sum of the risk factors for the participants that were neither dead (19 of them) nor censored after 37 months. Therefore, the likelihood will be high and the loss will be low if the risk score for the 20th deceased participant was much higher than the risk score for all the other participants that were still observed and did not die. The CoxPH model therefore learn to leverage the predictors to maximize the likelihood by attributing higher risk scores to participants who die first.

The partial likelihood loss function was already implemented in the "off the shelf" packages to perform survival analysis using an elastic net, a GBM or a fully connected neural network. There was however no package available to perform deep survival analysis using CNNs. As explained above, the loss function is complex and highly non-linear, which makes it a challenging task to code it using tensors, especially in its vectorized form. I also did not mention it above for the sake of brevity, but the loss function must be able to handled tied events, which the vanilla version of the partial likelihood does not. Solving this issue further complexifies the implementation of the loss function. After several unfruitful attempts (see Challenges further below), I was able to adapt the code I found in a tutorial by Sebastian Pölsterl: https://github.com/sebp/survival-cnn-estimator/blob/master/tutorial_tf2.ipynb. I was able to further adapt the code to handle multiple

input, so that I could build CNN architectures with a side neural network for scalar features.

4.5.4 CoxPH: Metric

After the model has been train, its performance can be evaluated, which is usually done using the concordance-index [CI] metric. The CI is similar in spirit to the loss function in that it is ranked based: it only penalizes the wrong ordering of the participants predicted survival time and does not take into account the absolute value of the predicted lifespan (which is undefined if the baseline hazard function was not estimated using the Breslow estimator or another method). The CI is defined as the percentage of pairwise comparisons between the participants for which the participant with the highest risk factor died first, and is therefore comprised between 0 and 1. Randomly guessing the risk factors gives a CI of approximately 0.5, so the values observed are usually between 0.5 and 1, with any CI value statistically significantly higher than 0.5 meaning that the model was able to extract information from the predictors to predict survival to some degree. Not all pairwise comparisons can be taken into account in the CI. If two participants are right-censored, one cannot estimate whether the model correctly attributed the higher risk factor to the participant that will die first, so the pair is discarded. If two participants died, the pair can always be used. If one participant died and the other was right-censored (did not), the comparison is only possible if the follow-up time of the right-censored participant is longer than the follow-up time of the dead participant. Specifically, the mathematical expression for the CI can be found in the following paper [7]. I imported the C-Index function from the python package lifelines. To monitor the training of the CNNs, I used the function coded with tensors by Sebastian Pölsterl in his notebook.

4.5.5 CoxPH: Estimation of the risk factor

Historically, the logarithm of the risk factor was initially estimated using a linear regression. This is what the elastic net algorithm is doing, with the small difference that the linear regression is regularized. With the progress of machine learning, the linear regression could be replaced with more complex algorithms. For example in my case, I decided to use a GBM, a dense neural network and a convolutional neural network (with a side neural network for scalar inputs), along with the aforementioned elastic net.

4.6 Hyperparameters tuning

I initially tuned the hyperparameters of my algorithms by sequentially tweaking each of them to identify a range of hyperparameters that would provide decent results. This first approach worked fine for fast algorithms with a limited number of hyperparameters such a elastic net, but it did not scale well for slower algorithms with a large number of interdependent hyperparameters such as GBM. I therefore considered three alternatives: grid search, random search, and Bayesian hyperparameters optimisation.

4.6.1 Grid search

Grid search is a simple way to tune hyperparameters. A list of candidates is listed for every hyperparameter to tune, and every combination is tried. The best hyperparameters combination to

train the final model is selected based on the prediction accuracy on the validation set. Aside from its simplicity, one advantage of grid search is that it can easily be parallelized, since each hyperparameter combination can be tested independently and asynchronously. It however suffers from three drawbacks. First, it scales extremely poorly when the number of hyperparameters to tune is large (e.g for a GBM). Second, if one or several of the hyperparameters have a limited effect of the performance of the model, a large proportion of the time will be spent exploring redundant hyperparameters combinations. For example, if two hyperparameters are useless, and if ten values are tested for each of them, the tuning time will be 100 times longer than it needs to be. In other words, 99% of the hyperparameters combinations tested will be redundant since they will only differ in terms of the useless hyperparameters. Finally, a third drawback of the grid search is that most often, the hyperparameter values tested are sampled periodically. In some rare cases, the performance of the model might be periodic with respect to the hyperparameters values, so a periodic grid search will fail to explore potentially highly performing hyperparameters combinations.

4.6.2 Random search

Random search adresses two of the three drawbacks of grid search by randomly sampling hyperparameters combinations: redundancy of the hyperparameters combinations, and periodicity of the sampling. Because the hyperparameters combination are random, even if some of the hyperparameters have no effect on the performance, each hyperparameter combination will explore new values for the important hyperparameters, so iterations are not wasted anymore on redundant hyperparameters combinations. The randomness of the hyperparameters sampling also ensure that they will not be sampled periodically. Like grid search, random search is fairly simple to code and can easily be parallelized. It suffers from only two drawbacks. First, if the number of combinations tested is not large enough, there is a risk that some zones of the hyperparameters space will not be explored at all, while some other will receive most of the "attention". This risk becomes very large as the number of hyperparameters to combine increases because of the curse of dimensionality. This is a risk that did not exist with the grid search, as the hyperparameters combinations tested were evenly spread in the hyperparameter space. Second, because of its randomness, a large number of hyperparameters combinations will still be wasted in low performance zones of the hyperparameter space.

4.6.3 Bayesian hyperparameters optimisation

Bayesian hyperparameters optimisation addresses the two shortcomings of random search. Instead of predeterministically (grid search) or randomly (random search) exploring the hyperparameter space, the candidate hyperparameter combinations are sequentially chosen based on the performance of the previously explored combinations. There is a trade-off between exploration and exploitation which allows the search to ensure that potentially promising zones of the hyperparameter space are not left unexplored (exploration) and to refine the tuning of the hyperparameters in the most promising zones already identified (exploitation). The way the hyperparameter combinations are selected is by approximating the function that maps hyperparameters combinations to their validation performance using a Gaussian process or Tree Parzen Estimators [TPE]. This function can then be used to identify regions of the hyperparameters space where the incertitude about the potential performance is the highest (exploration) or regions that are most likely to yield high validation performance (exploitation). After a hyperparameters combination has been

tested, its associated validation performance is used to update the surrogate function, which is then used to select the next most promising candidate combination. The two main drawbacks of Bayesian optimization is that (1) it is harder to implement than grid search or random search and (2) it cannot be parallelized in its vanilla version, since the hyperparameter combinations to explore are chosen sequentially based on the results from the previous combinations tested (some modified version of the algorithms allow the search to be parallelized by simultaneously exploring several promising next candidates).

Because of the advantages listed above, I implemented Bayesian optimization using the hyperopt library to tune the hyperparameters of my algorithms. I used TPE as the surrogate function.

4.6.4 Definition of the hyperparameters spaces

For the elastic nets, I tuned the regularization strength (loguniform distribution from 1e-10 to 1) and the ratio between L1 and L2 regularization (uniform distribution between 0 and 1). For the GBM, I tuned the maximum number of tree leaves for a single tree (integer uniformly sampled from 5 to 45), the minimum number of samples per leaf (integer uniform from 100 to 500), the minimum number of hessian weight per leaf (integer uniform from -5 to 4), the subsample ratio of samples for each tree (uniform from 0.2 to 1.0), the subsample ratio of features considered for each tree (uniform from 0.4 to 1.0), the L1 regularization (loguniform from 1e-2 to 1e2), the L2 regularization (loguniform from 1e-2 to 1e2) and the number of trees included in the model (integer uniform from 150 to 450). For the fully connected neural networks, I tuned the initial learning rate (loguniform distribution from 1e-5 to 0.1) and the weight decay (loguniform distribution from 1e-6 to 1e3). For the CNNs, I tuned the learning rate (loguniform distribution from 1e-5 to 1e-3), the weight decay (loguniform distribution from 1e-5 to 1e3), the dropout rate (uniform from 0 to 0.9), the presence of batch normalization after convolutional layers (choice: True/False) and the presence of batch normalization after dense layers (choice: True/False).

For the details about the hyperparameters spaces definition, please also feel free to refer to the "SPACES" dictionary defined under the __init__ function of the Predictions class.

4.6.5 Algorithms for which Bayesian hyperparameter tuning was not an option or failed to perform

For some of the algorithms, Bayesian optimization was not an option because training a model was too time consuming. For example, the GBM for survival analysis took more than 20 hours to train, so a Bayesian optimization with only 10 iterations would have taken a full week. Therefore I did not perform hyperparameters optimization for this algorithm, and I used the default hyperparameters values, as specified in the code.

I was able to perform Bayesian optimization on the fully connected neural network built to predict age. However, after deeper inspection, I realized that I was not providing enough epochs for the model to converge, and that increasing the maximum number of epochs as needed would make the training time too long for Bayesian optimization. I therefore used a different approached and tuned the few hyperparameters sequentially. First, I tuned the learning rate. Since I was using Adam, I found that small learning rates did not prevent the gradient descent from building momentum and still lead to fast convergence, so I used 0.0001 to make sure that the initial learning rate would not be too high, which could have prevented the gradient descent to work. Then I

tuned the architecture by progressively increasing the number of nodes and layers until I found that I was able to overfit on the training set to some degree. Finally, I estimated that considering the large sample size, I did not need to regularize the weights so I set the weight decay to zero. After that I trained the model.

The fully connected neural network built for survival analysis represented a similar challenge. Instead of a full hyperparameter tuning, I kept the same architecture as the one I tuned for age prediction, and I used a function provided by the package to tune the learning rate. Finally, I decided to set the dropout rate to zero, since the large sample size made overfitting fairly unlikely on such a shallow model. After that, I directly trained the model.

The survival CNN model also was too time consuming to train to convergence to be able to afford Bayesian hyperparameter tuning. Instead, I used the hyperparameters I obtained from performing Bayesian hyperparameter tuning on the age predicting model.

4.7 Parallelization of the training and tuning

I built models to predict two phenotypes (age and survival) from seven sets of predictors (demographics, features, PWA, all, demographics+PWA, features+PWA, all+PWA) using four different algorithms (elastic net, GBM, neural network and CNN), for a total of 30 models (the CNN cannot be applied to scalar features, and the other algorithms cannot be applied to time series). Training and tuning these models sequentially would have been prohibitively time consuming, so I parallelized the process. To do this, I first rewrote the code I had initially written as a Jupyter notebook using classes, and I split it into three different scripts: Preprocessing, Training and Postprocessing. I used bash and SLURM to parallely submit the 30 jobs on O2, Harvard Medical School's super computer.

4.8 Performance evaluation

For the prediction of age, I evaluated the performance of the different models using the R-Squared metric. For the prediction of survival, I evaluated the performance of the models using the concordance index metric. For the R-Squared metric, I estimated the confidence interval by bootstrapping the computation 1,000 times. For the concordance index metrics, the computation was prohibitively long so I only bootstrapped 100 times, although proper bootstrapping should be done 1,000-10,000 times. Bootstrapping is a method that consists in sampling with replacement from a dataset several times to simulate the sampling variance that occurred when the dataset was first collected from the total population. If the dataset is homogeneous, the different samples with replacement generated from it will be similar, so the performance estimation will be similar and the standard deviation on the performance will be small. In contrast if the dataset is very diverse, the different samples with replacement might be different and yield different prediction performances, which will result in a wider confidence interval for the performance estimation. It is important to be aware of the fact that the standard deviation computed using bootstrapping only captures a fraction of the total variance of the model, as it do not take into account that fact that retraining a model leads to different performances too. This latter source of variance is often considerably larger and can be estimated using the result of a cross-validation, since a 10-folds cross validation yields 10 different models independently trained (although the overlap in training samples between each two fold is approximately 7/8=87.5%) whose performances can

be used to compute the ”training variance”. The bootstrapping I performed only measures the ”testing variance”, not the ”training variance”.)

4.9 WARNING: Identification of two potentially devastating features/bugs

4.9.1 Batch normalization layers can lead to misleading training loss and performance during the training

tl;dr: The batch normalization layers behave differently during training and during testing. Therefore the training loss and metrics during training can drastically differ from the training loss and metrics during testing. I let the full story below, for those who would like to experiment with it themselves.

As I trained CNN architectures, I observed something strange: the training loss was going down, but the validation loss remained constant for a long time, before gradually increasing until the R-Squared value obtained was as bad as -5,000. This could be interpreted as overfitting, but the fact that the validation loss remained constant at first and never improved below the loss associated with guessing the mean of the distribution for every sample, instead of first decreasing and then increasing was puzzling to me. I therefore built simple architectures for which the problem did not happen and I tried to iteratively add and remove some of the layers to identify the source of the bug. However the problem was not reproducible, which repeatedly mislead me. To minimize the lack of reproducibility, I fixed as many seeds as I could (five of them: os environment seed, numpy library seed, random library seed, tensorflow seed and pytorch seed), but that did not solve the problem, there was still stochasticity in my training process which prevented me from reliably reproduce the problem. Some of my experiments first led me to believe that the problem came from the globalmaxpool layer, which I replaced by a flatten layer. After I later realized it had not actually solved the bug, I started suspecting that the one-dimensional convolution-related layers were the issue, because I had never noticed this problem before using two-dimensional convolutional layers. So I recoded my architecture using two-dimensional convolutional layers, by setting the second spatial dimension to be one. Again, the stochasticity of the bug led me to believe that I had solved the problem, but it was actually not the case. As I dived deeper into the debugging, I noticed something even more strange: during the training of the model, the training loss would go down, along with the training RMSE. However, after the training was complete, when I generated the predictions for the training set myself using model.predict(X_val), I found that the training performance was as bad as the validation performance. I hypothesized that it was maybe due to a shuffling of the predictions and that they were somehow not matching the target. However this hypothesis was not coherent with the highly negative R-Squared values I was observing. I manually inspected the predictions and noticed two puzzling things. (1) Their amplitude was extremely large, and increased during the training. This did not make sense since I had scaled the target variable to have a mean of zero and a standard deviation of one. (2) The sign of all the predictions was the same, which again did not make sense since I had centered the target variable (age) around zero. I therefore hypothesized that this behavior might be due to an overly large learning rate, which prevented the gradient descent from working. However this was not coherent with the fact that the training loss did seem to improve *during the training*, and indeed drastically decreasing the learning rate did not solve the problem. I really did not understand how it was possible for the training loss to decrease during training but to high during model evaluation, so the next experiment I ran was to use the training set as both the train-

ing set and the validation set. Unbelievably, I observed that the training loss on the training set used as the training set indeed decreased during training, whereas the validation loss on this same training set used as a validation set remained constant then increased exponentially. To be able to accelerate the debugging process and look deep into the problem, I worked on a tiny subset of the data, using only 64 samples as my training set, and the problem disappeared. As I increased the number of samples to 256 samples and higher, I observed that the "validation" (still based on the training set) loss would first remain constant and increase, and later decrease if my patience parameter on my early stopping callback function allowed it. After multiple experimentations, I was finally able to identify the culprit: the batch normalization layers. The closer to the final they were, the more likely they were to result in this "pathologic" behavior of the model. The problem was also a lot more likely to happen if the training set was large, and if the learning rate was large. The reason why the batch normalization behaved this way is still unclear to me and I did not find any explanation online, but at least I can explain that batch normalization is not applied the same way during training and validation. During training, the output of the layers are centered and scaled using the values of the batch. During testing, the values used to center and scale are not based on the batch, but are based on values learned by the model during training. This is the reason why the loss for the same samples can be loss during training and high during validation. After a large enough number of epochs, the batch normalization "weights" are properly learned, and learning can happen on the validation set too. One easy fix to my problem would have been to simply get rid of the batch normalization layers. However, using these layers is usually highly recommended, as they accelerate the learning and make the models more generalizable, providing a mild form of regularization. To be able to use them, I needed to be able to increase the patience of my early stopping callback to as least 50 or 200 epochs, but that would have made training prohibitively long. So I cancelled the jobs that I was running on O2 for a different project to increase my fairshare score, I requested GPUs and I registered them to be able to speed the training. This trick worked and I was able to tune some CNN models using batch normalization.

4.9.2 Very bad bug: hyperopt library returns the opposite boolean values for "choice" hyperparameters

I was surprised by the poor performance of some of the CNN models I tuned using the Bayesian optimization pipeline of the hyperopt library. I investigated the verbose output of the function and noticed a very bad bug: for some reason, the boolean hyperparameters selected are always inverted. So, if the best hyperparameter combination found during the exploration was CNN_BatchNormalization=True and DNN_BatchNormalization=False, the hyperparameter combination returned by hyperopt was CNN_BatchNormalization=True and DNN_BatchNormalization=False. The values of all the non-boolean hyperparameters were not affected and did match the values of the best performing hyperparameter combination encountered during the exploration. Therefore I manually inverted the values of the boolean hyperparameter returned by hyperopt, which indeed fixed my bug. I am still very surprised that such an important bug was not caught before. I googled it and found nothing. I wanted to contact the developper of the package but did not find the place to do it.

5 Experiments

5.1 Dataset

I trained, validated and tested my models on 235,241 pulse wave analysis samples from the UK Biobank cohort, aged 39-72 years. I describe the dataset in more detail under 4.4 - Approach - Dataset.

5.2 Results

I built 15 age predictors and 15 survival predictors. For each prediction task I used four different algorithms (elastic net, gradient boosted machine, neural network and convolutional neural network) and seven different sets of predictors. I used four sets of predictors for the algorithms built on scalar features (the demographic variables alone, the scalar features extracted by UKB, the time series time steps used a scalar predictors and the union of all the preceding scalar features) and three sets of predictors for the CNNs (PWA time series + demographics variables as side predictors, time series + UKB scalar features as side predictors, and time series + all scalar features as side predictors). I tuned all age predictors using Bayesian optimisation. I used Bayesian optimisation to tune the elastic net based survival predictor, and manually tuned the remaining survival models. The results can respectively be found in Figure 2 and Figure 3.

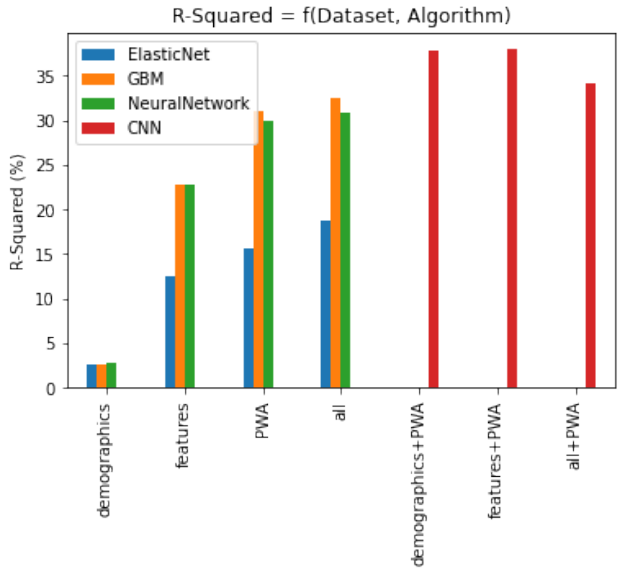

Figure 2: Performance of the different age predictors

5.3 Analysis, critique and interpretation of the performances of the different models

5.3.1 Age prediction performance

Baseline - Demographics predictors: First I generated a baseline prediction by predicting age using the demographics variables only. I found that 2.5% of the variance of age could be explained

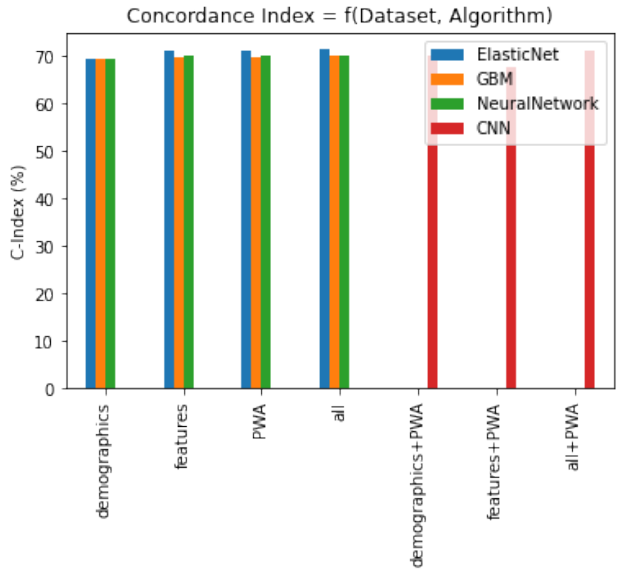

Figure 3: Performance of the different survival predictors

by demographics variable.

Summary features: Then I added the scalar features extracted by UK Biobank as predictors, and found that an elastic net could predict age with a R-Squared value of 12.6+-0.4%. I found that the two non-linear algorithms (GBM and neural network) significantly outperformed the elastic net, with respective R-Squared values of 22.8+-0.5% and 22.7+-0.5%. This higher performance of non-linear models suggests that they leveraged complex interactions and non-linearities between the scalar features that could not be used by the elastic net to predict age.

Raw time series: After establishing this baseline, I predicted age using the full time series and a convolutional neural network. I also added the demographic variables as side predictors, so that the features extracted by the convolutional layers of the CNN could be interpreted differently for male, females and the different ethnicities. This model outperformed the model built on the sole summary scalar features provided by UKB and predicted age with a R-Squared value of 37.8+-0.6%.

Time series steps as scalar features: Because the dimension of the raw time series was small (a single channel of only 100 time steps), I considered treating each time step as a scalar feature. Interestingly, I found that a GBM could reach R-Squared values higher than 30% (R-Squared=31.1+-0.5%) with a lot less training and tuning than a CNN. CNN are built to detect patterns in different places of the signals, which seems well suited to the task as detecting the presence and position of the shoulder of the PWA time series is for example what the features extracted by UKB do. However another key advantage of CNNs is that, by sharing the parameters of the kernel for every position of its application to the precedent layer, they drastically reduce the total number of parameters and allow models to be trained (without convolutional layers, building a complex and deep dense neural network directly on the image would result in a prohibitively large number of parameters). In the specific prediction task that I am performing, the initial dimension is small enough so that this is not a problem. Is is possible that, theoretically, a dense neural network or a

GBM could often perform almost as well as a CNN on the images by treating each pixel in each channel as a distinct scalar feature, but that in practice it can rarely be realistically considered.

CNN built on time series with summary features as side predictors: I then added the scalar features generated by UKB as side predictors to the CNN, and I found that the prediction accuracy could be further increased, although not significantly (R-Squared value=38.0+-0.5%). The modest amplitude of this increment in prediction performance suggests that the features extracted by UKB and the features learned from the time series by the model are similar or correlated, as they only provide weakly complementary information with respect to aging.

All predictors combined: I built a last model what combined a CNN built on the time series and a side neural network built on all the scalar features, including the same time series but treated as scalar features. Although the time series are fed to both the convolutional block and the side neural network, each block processes them and extract features from them in a different way. These features are later combined in the dense block. In theory, this model was the most likely to be the top performer. However, I found that it did not outperform the previously described model (R-Squared=34.1+-0.5). This is likely due to the stochasticity of the training process. Retuning and training the model several times would have probably yielded a similar performance to the best performing model, around 38%.

Prediction accuracy as a function of age: I hypothesized that some age ranges might be easier to predict than others. For example, one could imagine that for females arterial stiffness is relatively constant until menopause, after what it would increase linearly with time. I did not have time to do extensive statistical testing on this end, but I used a scatter plot to visually detect such patterns. In the aforementioned hypothetical menopause example, one would observe that before menopause age (around 50 years old) age prediction would be very difficult, so the data points would be spread away from the regression line. After menopause, part of age variance could be recovered using arterial stiffness, so the data points would get closer to the regression line. The scatter plot can be found under Figure 4. I did not notice any particular pattern, which suggests that arterial aging might be fairly linear over the age range 39-72. The reason why samples aged older than 63 years old are rarer is because these are repeated measure during a follow-up for a subset of the patients.

5.3.2 Survival prediction performance

Baseline - Demographics predictors: Age is an excellent survival predictor and females live on average four years longer than males in the UK, so I first established a baseline prediction by predicting survival using only the demographics variables. I obtained a C-Index of 69.5+-0.9% using an elastic net, a C-Index of 69.3+-1.0% and a C-Index of 69.4+-0.9% using a neural network which suggests that there are no complex interactions and non-linearities between the demographic variables with respect to survival prediction.

Summary features: I then added as predictors the scalars features extracted by UK Biobank, and obtained a C-Index of 71.0+-1.0% using an elastic net. Interestingly, the GBM and the neural network did not perform as well (respectively CI=69.7+-1.0% and CI=69.4+-1.0%), although the difference was not significant. This can be explained by the fact that unlike the elastic net, the GBM and the neural network were very slow to tune, so I could not use Bayesian optimization to tune them. It is also possible that the information linking the feature to survival was linear,

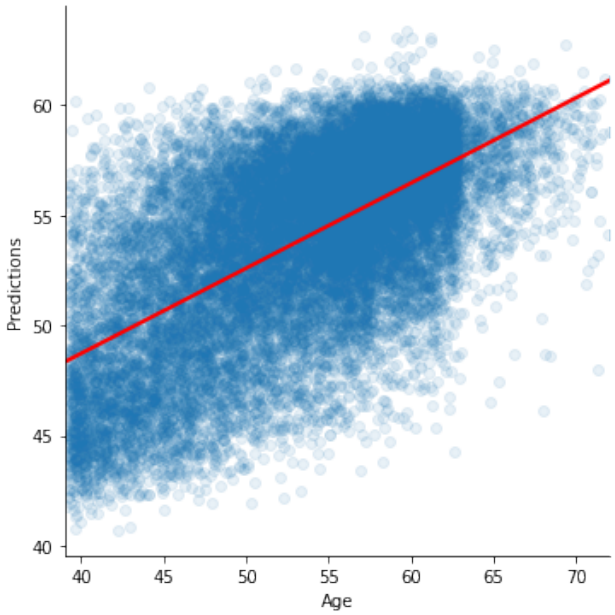

Figure 4: Scatter plot: Prediction = f(Age), Model: CNN trained on the time series, and the summary features as side predictors; R-Squared=38.0+-0.5

in which case a linear regression was better suited to discover and leverage it. In theory a tree based method can learn linear relationships too, by repeatedly splitting the variables in smaller and smaller buckets, but it will be a more challenging task for the algorithm, despite its simplicity for a linear regression.

Raw time series: I then used the same CNN architecture as the one I tuned when I predicted age to predict survival. I obtained the same C-Index of 71.0+-1.0%. I found that leveraging the time series as time series did not outperform the elastic net built on the summary scalar features.

Time series steps as scalar features: I then treated the time series as scalar features, and found that the best performing scalar features-based algorithm was the elastic net again, which predicted survival with a CI of 71.4+-0.9%.

All predictors combined: Finally, I built CNN models that used all the scalar features as side-inputs, including the time series itself, treated as scalar features. In theory, this model was the best suited to predict survival, but in practice it slightly (and not statistically significantly) underperformed with a CI of 71.1+-0.9%. My hypothesis for why complex modeled tended to underperform is that almost of all the survival prediction comes from the age variable. When this variable is diluted with a large number of less informative variables, it becomes harder for complex models to retrieve it. For example, tree based methods can generate variance in the tree they build and ensemble by randomly dropping some of the predictive features in each tree or split. When this happens, age could possibly be dropped which will strongly harm the performance of the model since the bulk of the model's performance depends on this feature. For neural networks, there is a risk that the neuron connection learning from the age side predictor might die (ReLU activation functions let some of the neurons die if the activation is negative, because the gradient is null for negative activations), or the nodes that learned from age might be dropped out. This makes it

harder, although not impossible, for these model to perform as well as the linear regression in this particular context. It is like looking for a needle in a haystack. If the haystack is small and still (elastic net), it will be easier to retrieve the needle than if the haystack is huge and stochastically moving (GBM, neural network, CNN).

5.3.3 Interpretability: Features driving the age predictions

I looked at the linear regression coefficients of the elastic net models and I computed the feature importance for the GBM models to interpret the models and understand which features were key in driving the predictions. There are too many models and features to list them on this paper, so I make them available as supplementary files.

Models built on the sole demographics predictors: For the age predictors built on demographics variables, I found that sex was an important feature and that the regression coefficient was positive. The regression coefficient is associated with being male (baseline for the sex variable is being female), so a positive value intuitive since in the UK Biobank cohort the average male is a couple years older than the average female.

Interpretation of the sex's coefficient in the context of models built on biological features: For the age predictors built on the biological scalar features as well as the demographics variables, I found that sex was the fifth most important feature for the elastic net, and the 8th most important feature for the GBM. In terms of absolute importance, the GBM did not rely on the sex variable as much as the elastic net. The regression coefficient for sex for the elastic net was positive. It is important to note that the interpretation for this coefficient is different than the interpretation for the coefficient of the aforementioned model built on the sole demographics predictors. The interpretation of the regression coefficient for a predictor is: the effect of a change of one unit of the predictor on the target variable, *when every other predictor included is kept at the same value*. The set of other predictors included in the model is different between the model built on the demographic variables and the model that includes the biological features as well, so the interpretation is different. When the arterial stiffness features are included and the model was trained to predict age the interpretation of a positive coefficient for sex is that, if a female and a male have exactly the same arterial stiffness, the male will be predicted to be older (e.g 50 years old) than the female (e.g 45 years old). At first, it could be tempting to conclude that it means that being a male means being older, and that males are aging faster than females, which is coherent with the fact that they have a shorter lifespan in average. As counter-intuitive as it might seem at first, it is important to understand that the interpretation of this coefficient should actually be the exact opposite. The above result shows that the arterial stiffness of a 50 years old male is similar to the arterial stiffness of a 45 years old female. So the interpretation of the coefficient is actually that males are decelerated agers compared to females. I found this finding counter-intuitive because of the shorter male life expectancy, so I first hypothesized that the positive regression coefficient could still be explained by the difference in the mean age of the males and females population in the UK Biobank cohort. I was however not convinced, because the model should have leveraged the other biomarkers to correct for that. I therefore did some literature research on the topic and found that arteries age differently for males and females, and that females are disproportionally affected by the associated cardiovascular disease risk (two folds) after menopause. source: Sex differences in mechanisms of arterial stiffness. Since two thirds of the age range covered by UK Biobank is post-menopausal, I hypothesize that the positive regression coefficient is indeed

a reflection of accelerated arterial aging for females. One way to confirm this hypothesis and to disentangle the two possible explanations would be to remove samples to ensure symmetry in the male and female age distributions and to retrain the elastic net model on this altered dataset. I made similar observations for the models built on the time series as scalar factors. Sex ranked lower because of the shear number of variables included in the model, but the sign of the coefficient remained the same.

Interpretation of the biological predictors' importance: In the model built on the summary scalar the features, I found that the three best predictors selected by the elastic net were Absence of notch position in the pulse waveform, Position of the pulse wave and Position of the shoulder on the pulse waveform. In contrast, the top three features selected by the GBM were Pulse wave Arterial Stiffness index, Pulse wave peak to peak time and Pulse wave reflection index. Interestingly, the top feature selected by the elastic net (Absence of notch position in the pulse waveform) was actually ranked last among the biological features by the GBM. This difference in feature importance evaluation can be explained in three different ways. (1) First and most importantly, features that are non-linearly related to the target variable or whose interaction with other features are complex will not be fully leveraged by the elastic net (hence the lower R-Squared values) and will therefore be deemed more important by the tree based models. (2) Some of the features might be highly correlated. Because of the lasso component (and the ridge component to a lesser degree) of the elastic net regularization, when two features are highly correlated the elastic net tend to select one of them and to strongly decrease the weight of the other. GBMs are stochastic models (based on the hyperparameters I used) and can therefore split on one or the other feature with a higher frequency depending on randomness. This might result in the elastic net ranking one of the two correlated predictors as high and the other as low, while the GBM did the opposite. (3) Finally, I did not assess the statistical significance of the ranking of the predictors. One way to assess that would be to perform a 10 folds cross validation so that I could use each of the 10 models trained to assess the variance in the feature importance. It is possible that, retraining the same algorithm several time, I would find different feature importance and that the variance between the 10 elastic net feature importances and the variance between the 10 GBM models' feature importances would be as high as the variance between the elastic net and the GBMs models, showing that there is actually no statistically significant difference between elastic nets and GBMs in terms of feature importance.

Interpretation of the time steps' importance: In terms of the time steps of the time series treated as scalar features, I found that the elastic net highlighted the time steps towards the middle of the time series (50-60 out of 100), whereas the GBM highlighted earlier time steps (15-25 out of 100).

Identification of the most important predictors categories: When the model was built on both the summary features and the time steps treated as scalar features, the elastic net gave more importance to the time series (the first 77 important predictors are time steps), whereas the GBM gave more importance to the summary features (five of the eight most important predictors are summary features).

5.3.4 Interpretability: Features driving the survival predictions

Interpretation of the importance of the demographics variables: Both the elastic net and the GBM ranked age as the most important predictor and sex as the second most important predictor. This was true for the models trained on the sole demographics variables, on the summary scalar

features, on the time series treated as scalar features, and on all the variables together. Age being ranked as the top predictor is intuitive since mortality risk increases exponentially with age. The positive regression coefficient for sex suggests that a male has on average a higher chance of dying than a female of the same age and with the same arterial health, which is coherent with the shorter lifespan of males.

Interpretation of the biological predictors' importance: The elastic net built on the biological features extracted by UKB ranked all the demographics variables, including all the one-hot encoded ethnicities as more predictive of survival than the biological features, which is coherent with the limited increase in survival prediction performance observed after adding the biological features as predictors. In contrast, the GBM estimated that the pulse rate of a participant was as predictive as sex in terms of survival prediction, which can be explained by the fact that with age the resting pulse tends to increase to compensate for arterial stiffness. Aside from age, sex, pulse rate and pulse wave arterial stiffness index, all the feature importance were estimated to be zero.

Interpretation of the time steps' importance: The elastic net built on the time series steps treated as scalar features highlighted 24 time steps as more important than age. The GBM highlighted the 8th timestep as important, along with age and sex. All the remaining feature importances were either null or negligible compared to the three first ones.

Identification of the most important predictors categories: Finally, the elastic net built on both the summary scalar features and the time steps highlighted 17 time steps as more important than age, and the GBM highlighted age, sex, the pulse rate and the 8th time step.

6 Discussion

6.0.1 Major takeaways

Age - Prediction performance: In conclusion, I was able to predict age with a R-Squared of 38.0+-0.5% by leveraging the raw PWA time series with the tools of deep learning. This model significantly outperformed the models I built on the summary scalar features extracted by UK Biobank (R-Squared=22.8+-0.5%). Notably, leveraging the dataset as a time series (as opposed to treating each time step as a scalar predictor) was difficult but yielded the best performing model. This finding strongly suggests that the medical community would benefit from leveraging the raw PWA signal rather than relying on the summary features when making diagnostics. The quality of medical care can be improved if medical professionals and machine learning practitioners work together to improve the diagnostic tools.

Age - Comparison between algorithms: I found that non-linear algorithms such as the GBM outperformed the performance of the elastic net (GBM: R-Squared=32.5+-0.5%; Elastic net: R-Squared=18.7+-0.5%), which suggests that non-linearities and complex interactions between features link PWA data to aging. This difference can also be observed in the features selected, which differ between the elastic net and the GBM. We found that the fully connected neural network and the GBM performed similarly (Neural network: R-Squared=30.8+-0.6%), but training a successful neural network was a lot more challenging and time consuming: the neural took more than 48 hours to train, tuning not included, whereas tuning and training of the GBM took less than 10 hours. Finally, the CNN was the best performing model (R-Squared=38.0+-0.5%), but tuning and training it required GPUs.

Survival - Prediction performance: In contrast to age prediction, I found that leveraging PWA data only marginally outperformed the models built on the sole demographics variables. Indeed, the demographics variables predicted survival with a C-Index of 69.5+-0.7% whereas the best model predicted survival with a C-Index of 71.4+-0.1%. This model was an elastic net trained on all the scalar features (both the summary features and the time steps treated as scalar features), which suggests that the limited information that could be extracted was likely linear.

6.0.2 Future directions

Several future directions are possible. (1) Although the sample size of the UK Biobank dataset is already massive, the model could benefit from being trained on augmented data to artificially increase the sample size. (2) Second, Bayesian hyperparameter tuning could benefit from being performed with a larger number of iterations, and the neural networks could also benefit from larger architectures, but these would take longer to tune and train. (3) Since I leveraged several architectures and algorithms, building an ensemble model could also increase the performance prediction. (4) When performing survival analysis, I did not filter out the death that were caused by trauma, although these events are only weakly related to biological aging. I do not expect a significant increase in performance in terms of C-Index as death by trauma represent a minority of the death events, but I would still recommend on adding this filtering step. (4) Finally, the results showed that leveraging the raw time series outperformed the models built on the summary scalar features. As a consequence, leveraging the raw time series to build disease predictors is, I believe, the most promising and pressing future direction.

References

[1] Discussion on paper by professor box and professor cox. *J. R. Stat. Soc. Series B Stat. Methodol.*, 26(2):244–252, July 1964.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, and Others. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015.

[3] Ahmed M Alaa, Thomas Bolton, Emanuele Di Angelantonio, James H F Rudd, and Mihaela van der Schaar. Cardiovascular disease risk prediction using automated machine learning: A prospective study of 423,604 UK biobank participants. *PLoS One*, 14(5):e0213653, May 2019.

[4] Zachi I Attia, Paul A Friedman, Peter A Noseworthy, Francisco Lopez-Jimenez, Dorothy J Ladewig, Gaurav Satam, Patricia A Pellikka, Thomas M Munger, Samuel J Asirvatham, Christopher G Scott, Rickey E Carter, and Suraj Kapa. Age and sex estimation using artificial intelligence from standard 12-lead ECGs. *Circ. Arrhythm. Electrophysiol.*, 12(9):e007284, September 2019.

[5] Yoav Ben-Shlomo, Melissa Spears, Chris Boustred, Margaret May, Simon G Anderson, Emelia J Benjamin, Pierre Boutouyrie, James Cameron, Chen-Huan Chen, J Kennedy Cruickshank, Shih-Jen Hwang, Edward G Lakatta, Stephane Laurent, João Maldonado, Gary F Mitchell, Samer S Najjar, Anne B Newman, Mitsuru Ohishi, Bruno Pannier, Telmo Pereira, Ramachandran S Vasan, Tomoki Shokawa, Kim Sutton-Tyrell, Francis Verbeke, Kang-Ling

Wang, David J Webb, Tine Willum Hansen, Sophia Zoungas, Carmel M McEniery, John R Cockcroft, and Ian B Wilkinson. Aortic pulse wave velocity improves cardiovascular event prediction: an individual participant meta-analysis of prospective observational data from 17,635 subjects. *J. Am. Coll. Cardiol.*, 63(7):636–646, February 2014.

[6] James Bergstra, Dan Yamins, and David D Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, volume 13, page 20, 2013.

[7] Adam R Brentnall and Jack Cuzick. Use of the concordance index for predictors of censored survival data. *Stat. Methods Med. Res.*, 27(8):2359–2373, August 2018.

[8] Ruixue Cai, Xiaoli Wu, Chuanbao Li, and Jianqian Chao. Prediction models for cardiovascular disease risk in the hypertensive population: a systematic review, 2020.

[9] François Chollet and Others. keras, 2015.

[10] J N Cohn, S Finkelstein, G McVeigh, D Morgan, L LeMay, J Robinson, and J Mock. Noninvasive pulse wave analysis for the early detection of vascular disease. *Hypertension*, 26(3):503–508, September 1995.

[11] Ralph B D'Agostino, Ramachandran S Vasan, Michael J Pencina, Philip A Wolf, Mark Cobain, Joseph M Massaro, and William B Kannel. General cardiovascular risk profile for use in primary care. *Circulation*, 117(6):743–753, 2008.

[12] Gagan D Flora and Manasa K Nayak. A brief review of cardiovascular diseases, associated risk factors and current treatment regimes. *Curr. Pharm. Des.*, 25(38):4063–4084, 2019.

[13] Andrea Ganna and Erik Ingelsson. 5 year mortality predictors in 498,103 UK biobank participants: a prospective population-based study. *Lancet*, 386(9993):533–540, August 2015.

[14] GBD 2017 Causes of Death Collaborators. Global, regional, and national age-sex-specific mortality for 282 causes of death in 195 countries and territories, 1980-2017: a systematic analysis for the global burden of disease study 2017. *Lancet*, 392(10159):1736–1788, November 2018.

[15] P Gnu. Free software foundation. *Bash (3. 2. 48)[Unix shell program]*, 2007.

[16] Mark Houston. The role of noninvasive cardiovascular testing, applied clinical nutrition and nutritional supplements in the prevention and treatment of coronary heart disease. *Ther. Adv. Cardiovasc. Dis.*, 12(3):85–108, March 2018.

[17] Noriko Inoue, Ryo Maeda, Hideshi Kawakami, Tomoki Shokawa, Hideya Yamamoto, Chikako Ito, and Hideo Sasaki. Aortic pulse wave velocity predicts cardiovascular mortality in middle-aged and elderly japanese men. *Circ. J.*, 73(3):549–553, March 2009.

[18] Christiana Kartsonaki. Survival analysis. *Diagn. Histopathol.*, 22(7):263–270, July 2016.

[19] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. In I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017.

[20] Teemu Koivistoinen, Leo-Pekka Lyytikäinen, Heikki Aatola, Tiina Luukkaala, Markus Juonala, Jorma Viikari, Terho Lehtimäki, Olli T Raitakari, Mika Kähönen, and Nina Hutri-Kähönen. Pulse wave velocity predicts the progression of blood pressure and development of hypertension in young adults. *Hypertension*, 71(3):451–456, March 2018.

[21] D Y Lin. On the breslow estimator. *Lifetime Data Anal.*, 13(4):471–480, December 2007.

[22] Yasutaka Maeda, Toyoshi Inoguchi, Erina Etoh, Yoshimi Kodama, Shuji Sasaki, Noriyuki Sonoda, Hajime Nawata, Michio Shimabukuro, and Ryoichi Takayanagi. Brachial-ankle pulse wave velocity predicts all-cause mortality and cardiovascular events in patients with diabetes: the kyushu prevention study of atherosclerosis. *Diabetes Care*, 37(8):2383–2390, August 2014.

[23] Wes McKinney and Others. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56, 2010.

[24] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[25] Michael F O'Rourke, Alfredo Pauca, and Xiong-Jing Jiang. Pulse wave analysis. *Br. J. Clin. Pharmacol.*, 51(6):507, 2001.

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, High-Performance deep learning library. In H Wallach, H Larochelle, A Beygelzimer, F dAlché-Buc, E Fox, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 8026–8037. Curran Associates, Inc., 2019.

[27] Sebastian Pölsterl. scikit-survival: A library for Time-to-Event analysis built on top of scikit-learn. *J. Mach. Learn. Res.*, 21(212):1–6, 2020.

[28] Sushravya Raghunath, Alvaro E Ulloa Cerna, Linyuan Jing, David P vanMaanen, Joshua Stough, Dustin N Hartzel, Joseph B Leader, H Lester Kirchner, Martin C Stumpe, Ashraf Hafez, Arun Nemani, Tanner Carbonati, Kipp W Johnson, Katelyn Young, Christopher W Good, John M Pfeifer, Aalpen A Patel, Brian P Delisle, Amro Alsaid, Dominik Beer, Christopher M Haggerty, and Brandon K Fornwalt. Prediction of mortality from 12-lead electrocardiogram voltage data using a deep neural network, 2020.

[29] Antônio H Ribeiro, Manoel Horta Ribeiro, Gabriela M M Paixão, Derick M Oliveira, Paulo R Gomes, Jéssica A Canazart, Milton P S Ferreira, Carl R Andersson, Peter W Macfarlane, Wagner Meira, Jr, Thomas B Schön, and Antonio Luiz P Ribeiro. Automatic diagnosis of the 12-lead ECG using a deep neural network. *Nat. Commun.*, 11(1):1760, April 2020.

[30] George C M Siontis, Ioanna Tzoulaki, Konstantinos C Siontis, and John P A Ioannidis. Comparisons of established risk prediction models for cardiovascular disease: systematic review. *BMJ*, 344:e3318, May 2012.

[31] Cathie Sudlow, John Gallacher, Naomi Allen, Valerie Beral, Paul Burton, John Danesh, Paul Downey, Paul Elliott, Jane Green, Martin Landray, Bette Liu, Paul Matthews, Giok Ong, Jill Pell, Alan Silman, Alan Young, Tim Sprosen, Tim Peakman, and Rory Collins. UK biobank: An open access resource for identifying the causes of a wide range of complex diseases of middle and old age, 2015.

[32] Tanvir Chowdhury Turin, Yoshikuni Kita, Nahid Rumana, Naoyuki Takashima, Aya Kadota, Kenji Matsui, Hideki Sugihara, Yutaka Morita, Yasuyuki Nakamura, Katsuyuki Miura, and Hirotsugu Ueshima. Brachial–ankle pulse wave velocity predicts all-cause mortality in the general population: findings from the takashima study, japan, 2010.

[33] Guido Van Rossum and Fred L Drake. *The Python Language Reference Manual*. Network Theory Limited, March 2011.

[34] Stéfan van der Walt, Stéfan van der Walt, S Chris Colbert, and Gaël Varoquaux. The NumPy array: A structure for efficient numerical computation, 2011.

[35] I B Wilkinson, S A Fuchs, I M Jansen, J C Spratt, G D Murray, J R Cockcroft, and D J Webb. Reproducibility of pulse wave velocity and augmentation index measured by pulse wave analysis. *J. Hypertens.*, 16(12 Pt 2):2079–2084, December 1998.