LELEC2870 - Machine Learning project : Heart failure on the rise in the Smurf society

Guerand Dewell (84792200) , Julie Deprez (69952200)

July 29, 2026

1 Introduction

The goal of this project is to predict the risk of heart failure in the Smurf society. To do so, we have at our disposal data about some smurfs : their age, weight, blood pressure, level of hemoglobin in blood, etc. Using machine learning and a training dataset, we built a model than can predict the risk of heart failure in the Smurf Society. We proceeded by steps : data preprocessing, feature selection, model selection, with both linear and non-linear models. In this report, we presented our choices, results, and chose the model with the best performance. Finally, we integrated image of hearts in the data to see whether it helps the prediction or not.

2 Part 1 : Linear model

2.1 Data preprocessing

Before doing anything, we prepared the the data for analysis. First we checked that there were no missing values (NaN) with the line of code `data.info()`. Then, we encoded the categorical variables "sarsparilla", "smrufberry liquor" and "smurfin donuts" by replacing their values with numeric labels ("very low" $\rightarrow$ 1, "low" $\rightarrow$ 2, "medium" $\rightarrow$ 3, "high" $\rightarrow$ 4, "very high" $\rightarrow$ 5), so that they can be processed by regression. We splited the feature "profession" into six new features, one for each profession present in the dataset. This allowed us to use it as numerical binary variables (one-hot encoding). Then, we checked wether there were outliers that were errors in the samples. When comparing the 95th percentile value with the max value, and the 5th percentile value with the min value for all features, we can identify the presence of some extreme values in the data distributions but it does not seem like there is any measurement or data errors. We thus chose to keep all samples. Finally, we standardized the numerical features to bring all features to a similar scale.

2.2 Feature selection

For feature selection, we focused on filter methods rather than wrapper methods, because the model had not been selected yet, and choosing features based on a specific model could bias the analysis and lead to overfitting.

First, we applied a correlation filter, and visualized the correlation matrix ploting a heatmap. This allowed us to simultaneously evaluate :

1. the correlation between each feature and the target
2. the correlation between the features to detect redundancy

We then used a mutual information filter to detect non-linear dependencies that correlation alone might miss. Thanks to these two filters, we observed that the features "profession_craftsmanship" and "profession_manufacturing" were not really linked with the target (risk of heart failure) because they had 0 mutual information and 0.01 correlation with it. We also identify other features like "Calcium" and other profession which have a 0 mutual information value.

Finally, we used an embedded method (a decision tree) to see whether there were other features to delete and identify other type of relations. It measured the feature importance, and the features "profession_craftsmanship" and "profession_manufacturing" had the lowest. As they also have a low correlation and mutual information with the target, we deleted them. After model selection, we used other methods to ensure that the features that we had removed were the "the correct ones". One of them being an embedded method with our linearRegressor.

After identifying the most relevant predictors, we retained the reduced feature set for subsequent model selection and training.

The figure 1 represents the features importance, according to 3 different methods of feature selection (correlation filter, mutual information filter, and an embedded method). The values obtained the methods are on a scale $[0, 1]$ so that we can identify relevant features.

2.3 Model selection

The first step in model selection was to split our training set into a validation set and a training set. We will use the test set to only predict the final value of our best model.

In the model selection part we tested 3 differents types of Linear regression model : *MyLinearRegressor()*, *Ridge()* and *ElasticNetCV()*.

- *MyLinearRegressor()* : Classic linear regression i.e : $\hat{w} = \underset{w}{argmin} ||y - Xw||_2^2$
- *Ridge()* : Linear regression with penalty on the coefficients i.e : $\hat{w} = \underset{w}{argmin} ||y - Xw||_2^2 + \alpha ||w||_2^2$

Dewell Guerand
Deprez Julie

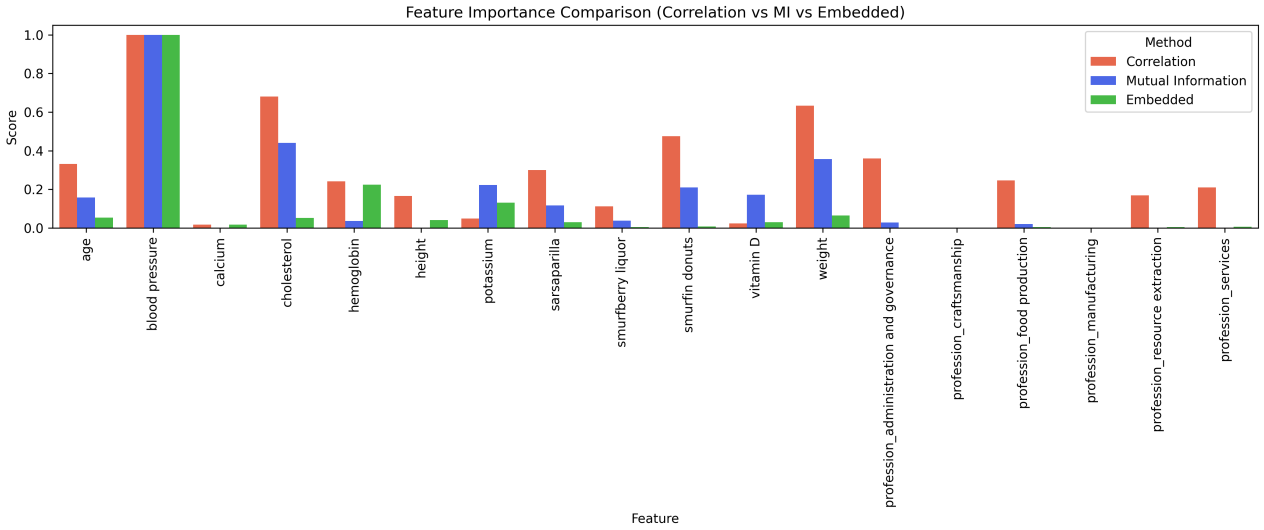


Figure 1: Comparison of feature importance values for differents method.

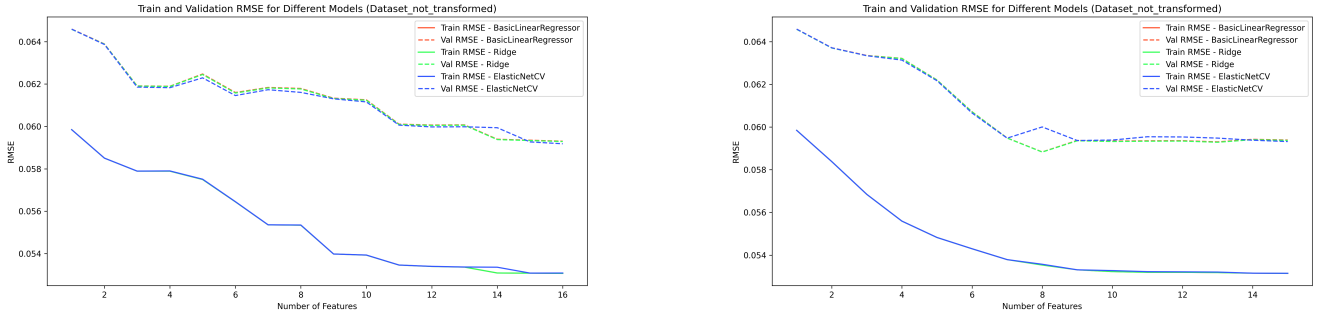
- *ElasticNetCV()* : Linear regression with penalty on the coefficients and reduction of variables i.e : $\hat{w} = \underset{w}{argmin}(||y - Xw||_2^2 + \alpha[\frac{1}{2}(1 - l_1) \times ||w||_2^2 + l_1||w||_1])$

To choose the most relevant model for our dataset, we used RMSE on the valisation set to rank them, meaning that a more effective model will have a smaller prediction error on the validation set. In order to rank the models from best to worst and re-identify the most important features, we first sorted the features according to their importance. We then trained our model by gradually adding the features according to the importance previously determined. We used two types of methods to determine the importance of the features. First, we used a classic method where we trained the model on each feature and calculated the RMSE of the model using only that feature. Then, we used a more advanced technique where we used a wrapper and therefore a backward approach, gradually increasing the number of features that the wrapper could take, which gave us the most important features. This analysis allowed us to identify that all three models tested worked better with all the features and that the best one was elasticNet (see figures 2 and 3).

The elasticNet we used allows its parameters to be adjusted to obtain the smallest RMSE. It is therefore, in a sense, optimal for our dataset. Indeed we obtained un training RMSE of 0.0341 for the train set and 0.0442 for the validation set which enable us to clearly see that there is some overfitting.

Another technique we used to reduce the prediction error of our linear models was an approach based on identifying relationships between variables. We used the "autofeat" library, which enables automated feature engineering.

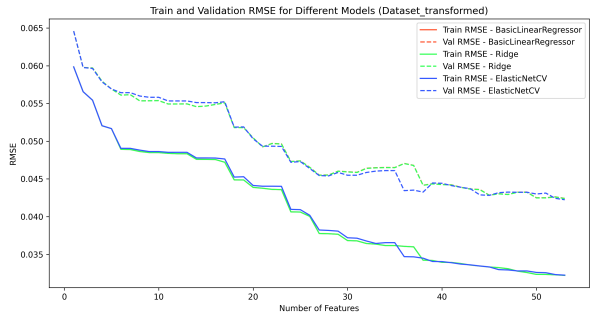
The figures 2 and 3 clearly show that using a wrapper allows our RMSE curve to descend more smoothly and without jumps, as we add features in their true order of importance. This technique identifies relationships between variables even if they are linear, which is not the case with the other technique. Next, we can see directly that by identifying the nonlinear relationships between variables via auto feat, we further improve our error.



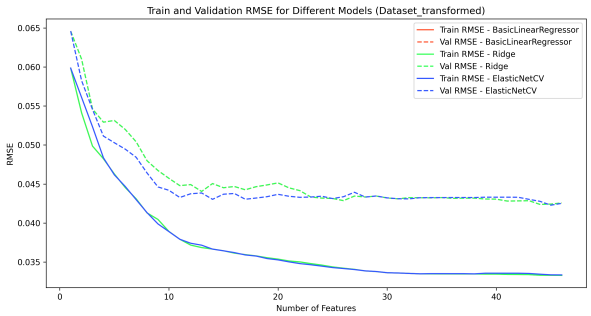
(a) RMSE

(b) Backward

Figure 2: Comparison of linear methods with features being taken in order of RMSE value and with the backward features classifications



(a) RMSE and autofeat



(b) Backward and autofeat

Figure 3: Comparaison of linear method with "autofeat"

3 Part 2 : Nonlinear models

After exploring linear models to model the problem, we tried to improve our technique by using non-linear models. We trained the data on several of them : MLP, RandomForest, GradientBoostingRegressor and KNN. Since non-linear models do not react in the same way to input features as linear models, we processed a new feature selection, differently from in part 1. We used wrapper methods (forward and backward selection) on each model, so that they have each their most adapted features. To fit the data to the models, we proceeded in 3 steps :

1. Hypermarameters tuning with k-fold cross validation
2. Feature selection with a wrapper method
3. A second hyperparameter tuning on the reduced feature set obtained in step 2

Here we have represented the pseudo algorithm that we have used to apply these 3 steps :

Algorithm 1 Algorithme d'optimisation avec hyperparamètres et sélection de features

Input: Ensembles d'entraînement X_{train}, y_{train}
Input: Ensemble de validation X_{val}, y_{val}
Input: Modèle M, grille de paramètres $\mathcal{G}$
Input: Validation croisée k-fold, mode de sélection (`backward`/`forward`)
Input: Option `scaling_target` (booléen)
Output: Modèle optimisé M_{final}, features sélectionnées S^*
`// Normalisation optionnelle de la cible, si MLP`
if *scaling_target* = *True* **then**
  $y_{\text{train}} \leftarrow \text{StandardScaler}(y_{\text{train}})$ $y_{\text{val}} \leftarrow \text{StandardScaler}(y_{\text{val}})$
`// Étape 1 : Premier GridSearch`
$h_1^* \leftarrow \text{GridSearch}(M, \mathcal{G}, X_{\text{train}}, y_{\text{train}}, k\text{-fold})$ $M \leftarrow M(h_1^*)$; `// Appliquer les meilleurs hyperparamètres`
`// Évaluation après premier tuning`
Entraîner M sur $(X_{\text{train}}, y_{\text{train}})$
`// Étape 2 : Sélection de features`
$S^* \leftarrow \text{WrapperSelection}(M, X_{\text{train}}, y_{\text{train}}, \text{mode}, k\text{-fold})$ $X^*_{\text{train}} \leftarrow X_{\text{train}}[S^*]$ $X^*_{\text{val}} \leftarrow X_{\text{val}}[S^*]$
`// Évaluation après sélection de features`
Entraîner M sur $(X^*_{\text{train}}, y_{\text{train}})$ $\hat{y}_2 \leftarrow M.\text{predict}(X^*_{\text{val}})$
`// Étape 3 : Second GridSearch (tuning final)`
$h_2^* \leftarrow \text{GridSearch}(M, \mathcal{G}, X^*_{\text{train}}, y_{\text{train}}, k\text{-fold})$ $M_{\text{final}} \leftarrow M(h_2^*)$; `// Hyperparamètres finaux`
`// Évaluation finale`
Entraîner M_{final} sur $(X^*_{\text{train}}, y_{\text{train}})$ $\hat{y} \leftarrow M_{\text{final}}.\text{predict}(X^*_{\text{val}})$ **if** *scaling_target* = *True* **then**
  $\hat{y}_3 \leftarrow \text{inverse\_transform}(\hat{y}_3)$
$\text{RMSE}_3 \leftarrow \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{\text{val},i} - \hat{y}_i)^2}$ Afficher RMSE_3
return M_{final}, S^*

We decided to begin by tuning the hyperparameters instead of the feature selection, because to do the feature selection, we already need a fully specified model (with fixed hyperparameter valued). Running feature selection on a poorly tuned model could lead to selecting sub-optimal features. Starting with reasonably tuned hyperparameters increases the chance that the selected features are meaningful.

In practice, to find the optimal parameters, we tested the performance of our models for all combinations of parameter sets that we know strongly influence the performance of our models. All of this was done using the GridSearchCV function. In the following table (1), you can find the parameters used as well as the optimal ones found.

Model	**Hyperparameters Tested**	**Best Parameters Found**
MLP Regressor	hidden_layer_sizes: (128,128), (256,256) learning_rate: adaptive, constant, invscaling learning_rate_init: 1e-4, 1e-3, 1e-2 max_iter: 1,2,4,8,16,32,64,128,256,512	hidden_layer_sizes = (256,256) learning_rate = constant learning_rate_init = 0.01 max_iter = 256
Random Forest	n_estimators: 100, 200, 300 max_depth: None, 10, 20, 30 min_samples_split: 2, 5, 10	n_estimators = 100 max_depth = 20 min_samples_split = 2
Gradient Boosting Regressor	n_estimators: 100, 150, 200 learning_rate: 0.05, 0.1, 0.2 max_depth: 3, 5, 7	n_estimators = 150 learning_rate = 0.1 max_depth = 3
KNN Regressor	n_neighbors: 3,5,7,9,11,15,20,25 metric: euclidean, manhattan, minkowski	n_neighbors = 7 metric = euclidean

Table 1: Hyperparameters tested and best parameters found for each nonlinear model.

For hyperparameters tuning, we used k-cross-validation to reduce the risk of overfitting. Indeed, it uses "k" different training and validation sets, so it ensures the model is evaluated on multiple subsets of the data rather than just one random partition. During hyperparameter tuning, k-fold cross validation is used to evaluate each combination of hyperparameters values. The training set is split into k subsets, and the model is trained k times, each time using a different fold as the validation set while the remaining folds serve as the training data. It produces a robust estimate of the model's performance.

For each method, we tried 2 different methods of feature selection : forward and backward selection. In the forward selection, features are added one at a time, and at each step, the feature that provides the most improvement in model performance is selected. In the backward search, all features are initially included, and at each step, the least importance feature is removed. The advantage of forward search is that it's computationally efficient, since it trains models on small subsets, it never needs to train a model on all features at once. The backward search can take much more time, since it requires training a model with all features at the start and retraining it after each removal. However this method may better capture interactions between features, because it evaluates them in the context of the complete set. For the two methods, we tried with all values (from 1 to 18) of the stopping criteron "n_features" to find the optimal features number to select by comparing the RMSEs obtained on the validation set. We compared the final RMSE of the models on the validation set with both feature selection methods in the tabular 2. The number of features selected is noted in parentheses next to the rmse. We can see in the tabular that the performance of models is better when we use backward search as method for feature selection. It means that the 2 methods sometimes don't select the same features for the models. We can compare the differences in the sets of selected features for each model and for each method, and relative to part 1 (see tabular 3).

Models	**Final RMSE**	
	Backward Search	**Forward Search**
MLP	0.0441 (7)	0.044 (7)
Random Forest	0.045 (8)	0.0449 (7)
Gradient Boosting Regressor	0.0399 (18)	0.0401 (17)
KNN	0.0486 (10)	0.0504 (12)

Table 2: Comparison of final RMSE for different models using two types of feature selection: backward and forward search.

By observing the table 3, we remark that some features are kept, no matter the models. We can thus make the hypothesis that these features are really important to predict the heart failure risk : age, blood pressure, hemoglobin, potassium and weight. Furthermore, blood pressure is always the first feature to be kept with the backward of the forward search.

Since gradient boosting regressor is the model we have selected for the rest of this article, we will explain it briefly: Gradient boosting regressor is a model based on the creation of a series of decision trees that aim to correct the errors of the previous ones. Each new tree focuses on minimizing the residual errors—the differences between actual and predicted values—rather than learning directly from the original targets.

Models	Feat. Sel.	Features																	
		F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18
MLP	Forward	x	x			x		x					x	x	x				
	Backward	x	x			x	x	x					x				x		
RandomForest	Forward	x	x		x	x		x	x				x						
	Backward	x	x		x	x	x	x	x				x						
GradientBoost	Forward	x	x	x	x	x	x	x	x	x	x	x	x		x	x	x	x	x
	Backward	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
KNN	Forward	x	x		x	x	x	x					x	x	x	x	x	x	
	Backward	x	x			x	x	x					x		x	x	x	x	
Linear models		x	x	x	x	x	x	x	x	x	x	x	x	x		x		x	x

Table 3: Comparison of selected features by model and feature-selection method (F1 = age, F2 = blood pressure, F3 = calcium, F4 = cholesterol, F5 = hemoglobin, F6 = height, F7 = potassium, F8 = sarsaparilla, F9 = smurfberry liquor, F10 = smurfin donuts, F11 = vitamin D, F12 = weight, F13 = Profession administration and governance, F14 = profession_craftsmanship, F15 = profession_food_production, F16 = profession_manufacturing, F17 = profession_resource_extraction, F18 = profession_services)

4 Part 3 : Integration of image data

In order to add our images to our training, we proceeded as follows:

- First, we took the CNN from the exercise session. We then extracted a number of features n from it and added this new data to the previous data. But then we asked ourselves what the optimal number of features n to extract from the CNN would be. We then iteratively created and trained CNNs with a different number of features to extract and add to our initial dataset, then trained our GradientBoost on them to observe the RMSE. Since the CNN is not deterministic due to the initialization of weights or the shuffling of data, we extracted the best model each time we ran the code. We also ran it several times for each feature to determine the mean and variance.
- Next, we decided to compare this model with a pre-trained "resnet18" model that extracts 512 features from the image and determined that the best model was often the CNN with the optimal number of features extracted previously.

On figure 4, we can observe the distribution of the RMSE as a function of "n_features", and we see that in this case, $n = 2$ achieves the best RMSE. We therefore selected this model and compared it with ResNet. We also train an XGboost on the residuals, which are defined as the difference between the actual expected results and the prediction, in order to further reduce errors. This allows us to correct any remaining systematic errors.

By integrating image data in the model, we went from an error of 0.04 to an error of 0.0336, so it made the error go down from 19%.

Table 4: Comparison of RMSE between classical CNN and a ResNet model

Model	RMSE (Validation)	RMSE (Test)
CNN	0.0336	0.0335
ResNet-50	0.0372	0.0398

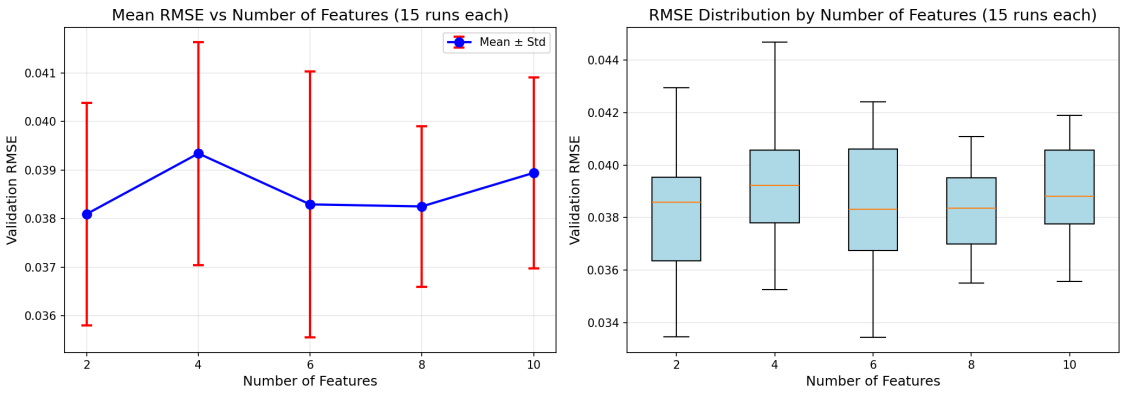


Figure 4: Mean of RMSE in function of n$_f$*eaturesandalsothevariance*

Dewell Guerand
Deprez Julie

5 Part 4 : Understanding heart failure in Smurf Society

To understand heart failure in Smurf Society, we analyzed characteristics of Smurfs, in function of their predicted heart failure. First, we compare extremes : smurfs with y (risk of heart failure) > 0.25 and smurfs with y < 0.02. To do that, for each smurf, we standardized their features, and then we took the mean for each feature for the population with y > 0.25, and the same for population with y < 0.02. You can see the results on figure 5. By analyzing the features that have the biggest differences between these 2 groups, we can formulate hypotheses about the causes of heart failure. The biggest difference is hemoglobin, smurfs with high risk have a higher hemoglobin rate. For calcium and vitamin D, it's the inverse : smurfs with high risk have a lower rate. Smurfs of this group have also a bigger age, blood pressure and cholesterol rate.

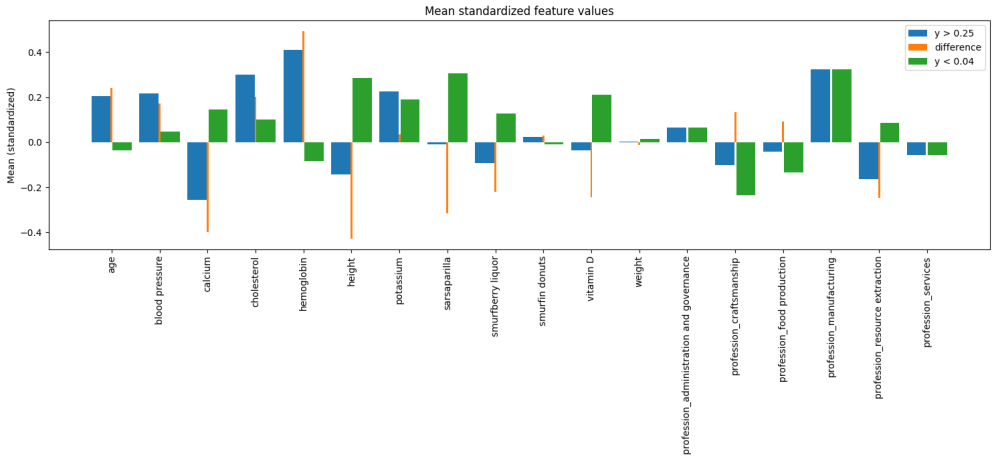


Figure 5: Comparison of standardized features of smurfs with y > 0.25 and smurfs with y < 0.02

We can also observe the characteristics of only the smurfs most at risk, and find their similarities in term of features. If we check at the 5 smurfs most at risk, we can see that they have all weight > 100 kg, and a low fooding rate of sarsaparilla. They all have an age of more than 100 years, except one. In order to determine which Smurfs are most at risk, we have also plotted images of the five Smurfs with the highest risk values for heart failure, as well as those with the lowest risk values.

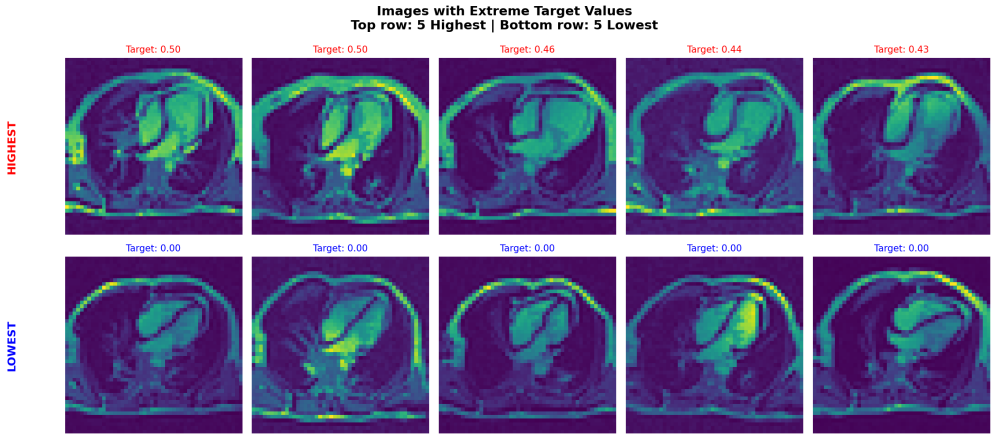


Figure 6: Comparison of the image of low risk vs high risk Smurfs

From these images, we can see that the larger the upper right part of the organ being examined, the greater the risk of heart failure for the Smurf.

6 Conclusion

In conclusion, we studied a lot of possibilities to predict the risk of heart failure in the Smurf society. The best option among what we tested seems to be the non-linear model GradientBoostingRegressor, trained with all the 18 basis features, and with the features corresponding to the image of heart extracted by the model CNN. If we were to improve the project, we would examine other image analysis models as well as other prediction models in order to obtain better results.