Heart Failure on the Rise in the Smurf Society

By Joris Weets and Jakob Cordua Thibodeau

This paper describes the development of a predictive model for assessing heart failure risk within the Smurf population, from tabular clinical data and heart scan images. The process of finding an optimal model occurred in three distinct phases: establishing a Linear Model (Baseline), exploring and optimizing various Non-Linear Models, and finally, enhancing the best-performing model by integrating the image data using a Convolutional Neural Network (CNN). Ultimately, we perform an analysis of the factors explaining heart failure risks and offer recommendations to the smurf population.

The performance metric used for evaluation across the project is the Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\sum_{i}^{n} \frac{(\hat{y_i} - y_i)^2}{n}}$$

Part I
Linear Model (Baseline)

We obtained a linear model baseline using 3 steps: categorical data transformation, feature scaling, and feature selection.

1 Data transformation

There are four categorical features in the smurf dataset.

Sarsaparilla consumption, *smurfberry liquor* and *smurfin donuts*, having values *Very low, Low, Moderate, High* and *Very high*, were mapped from 0 to 4 (0 being *Very low* and 4 being *Very high*), since the categories have a natural sense of order

Jobs, however, are not ordered. Therefore, a one-hot encoding is prefered since it does not create any sense of hierarchy. One-hot encoding is done by creating six distinct features (one for each type of job) with value 1 if the smurf works on that job and 0 if not.

Finally, all references to images were dropped because they are not included in the baseline model.

1.1 Data scaling

The Smurf dataset contains features on different scales. For algorithms sensitive to feature magnitude, such as Linear Regression and gradient-based methods, a small subset of features could unfairly dominate the learning process. Therefore, all features are standardized using Scikit-Learn's *StandardScaler*. Normalization was also attempted, but it yielded worse results.

2 Feature Selection

Not all features are useful. Feature selection allows us to filter out features that are redundant and/or create noise. Throughout this project (not just part 1), feature selection is done in 2 ways:

1. Greedy Search (Wrapper)
2. Mutual Information (Filter)

2.1 Greedy Search Wrapper

Greedy search is performed using Scikit-Learn's SequentialFeatureSelector. Since linear regression is relatively inexpensive computationally, we perform a backwards search with an uncapped number of possible features. The scoring metric to decide which subset of features is best is RMSE.

Instead of using a separate validation set to evaluate the best subset of features, we use 5-fold cross-validation. We found 5 folds to be a good compromise between robustness and computation complexity.

For linear regression, the ideal subset of features is:

- age
- blood pressure
- cholesterol
- hemoglobin
- height
- potassium
- sarsaparilla consumption
- weight
- administration and governance (job)

2.2 Mutual Information Filter

A simpler method of feature selection is using a filter. Since we want a feature selection adapted for all models, we filter using Mutual Information (MI), which captures statistical relationships (not only linear correlation). In practice, we run SKLearn's mutual_info_regression() on our features relative to the target (risk). We then filter out all features with an MI score lower than 0.01.

The features kept by the filter (and their respective MI scores) are:

- Blood pressure - 0.246
- cholesterol - 0.108
- weight - 0.086
- potassium - 0.059
- age - 0.059
- smurfin donuts - 0.055
- sarsaparilla - 0.043
- vitamin D - 0.039
- height - 0.019
- resource extraction (job) - 0.017
- hemoglobin - 0.015
- Administration and governance (job) - 0.012

3 Baseline predictor

Linear Regression doesn't require model selection since there are no hyperparameters; therefore, we can compare our feature selection methods and select the best model. The following are RMSE scores on test data:

Test RMSE Linear Regression	
No Feature Selection	0.0561
Greedy Search	0.0556
Mutual Information	0.0560

Our Baseline is therefore a Linear model taking greedy search selected features with RMSE 0.0556.

Part II
Non-Linear Models

To try and improve on our baseline, 4 types of non-linear regressors were trained: Polynomial Regression, K-Nearest Neighbors, Neural Network and Random Forests. All models were trained with the 3 feature selection methods mentionned above: No selection, Greedy Search and Mutual Information.

1 Polynomial Regression

The Polynomial regression model is implemented using LinearRegression (just as the Baseline). But right before, we use SKlearn's PolynomialFeatures to create additional features by considering all combinations of the original features up to degree n. The Linear Regression model is then trained on this expanded feature set, making it capable of modeling a non-linear relationship with the input features. We tested $n = 1, 2, 3, 4$, with $n = 2$ giving by far the best results, implying a quadratic relationship between some features.

The following are RMSE scores on test data:

Test RMSE 2nd order Polynomial Regression	
No Feature Selection	0.0435
Greedy Search	0.0419
Mutual Information	0.0415

The optimal Polynomial regression model uses Mutual Information feature selection. Note that the MI filter is applied **before** creating polynomial features. We also tried applying the MI filter after creating polynomial features, however this yielded significantly worse results.

2 Hyperparameter Optimization

Since the next 3 models possess hyperparameters that significantly influence their performance, we performed Grid Search (still with 5-fold cross-validation) to try and find the optimal configuration which minimize the RMSE.

Feature selection is done for **each model** trained using GridSearch since we want to find the best hyperparameters and features combinations. Although this adds significant overhead, we find this solution more sound than to find hyperparameters that fit the full feature set, only to cut down the feature set.

However, with many hyperparameters to test (and especially with Greedy feature selection), Grid Search can be too computationally expensive. Therefore, for Neural Networks and Random Forest, which take more training time per model, we use SKLearn's HalvingGridSearch instead.

HalvingGridSearchCV starts by evaluating all hyperparameter combinations on a small sample of the dataset, discarding poor-performing combinations. Then, remaining combinations are trained on increasingly larger samples of data. This "pruning" approach is less exhaustive than regular GridSearch, but still systematically finds good solutions in less than half the time.

3 K-Nearest Neighbors Regressor

The K-Nearest Neighbors (KNN) model is implemented using SKLearn's KNeighborsRegressor. The hyperparameters tuned for KNN are:

- **n_neighbors**: number of neighbors to consider.
- **weights**: whether all neighbors are weighted uniformly or by distance.
- **metric**: if weights are distance-based, how to calculate distance.

The following are RMSE scores on test data:

Test RMSE KNN	
No Feature Selection	0.0607
Greedy Search	0.0524
Mutual Information	0.0556

The optimal KNN model uses Greedy Search feature selection and the following hyperparameters: **n_neighbors = 8**, **weights = distance** and **metric = euclidian** .

4 Neural Network

The Neural Network (NN) model is implemented using SKLearn's MLPRegressor. The hyperparameters tuned for NN are:

- **hidden_layer_sizes**: Number of hidden layers and number of nodes per layer.
- **activation**: Activation function for all nodes.
- **solver**: Specific gradient descent strategy used.
- **alpha**: L2 Regularization constant that limits weights size.
- **learning_rate_init**: Initial learning rate for sgd or adam solvers.
- **max_iter**: Maximum number of training epochs.

The following are RMSE scores on test data:

Test RMSE NN	
No Feature Selection	0.0426
Greedy Search	0.0396
Mutual Information	0.0411

The optimal NN model uses Greedy Search feature selection and the following hyperparameters: **hidden_layer_sizes** $= (\mathbf{32}, \mathbf{16})$, **activation** $=$ **tanh**, **solver** $=$ **lbfgs**, **alpha** $= \mathbf{0.1}$ and **max_iter** $= \mathbf{200}$. Note that this model has 2 hidden layers.

The **learning_rate_init** is irrelevant since the **lbfgs** solver approximates second order derivatives to decide on the step size and direction.

5 Random Forest Regressor

The Random Forest (RF) model is implemented using SKLearn's RandomForestRegressor. The hyperparameters tuned for RF are:

- **n_estimators**: Number of tree classifiers trained and voting.
- **max_depth**: maximum depth of each tree.
- **min_samples_leaf**: Minimum amount of data points in a node for it to be considered a leaf.
- **max_features**: Maximum number of features to consider when making a split.
- **max_samples**: Fraction of dataset to sample from to train each tree.

The following are RMSE scores on test data:

Test RMSE RF	
No Feature Selection	0.04657
Greedy Search	0.04660
Mutual Information	0.04655

The optimal RF model uses Mutual Information feature selection (although all 3 methods are extremely close in performance) and the following hyperparameters: **n_estimators** $= \mathbf{250}$, **max_depth** $= \mathbf{20}$, **min_samples_leaf** $= \mathbf{1}$,**max_features** $=$ **None** and **max_samples** $= \mathbf{0.8}$.

6 Non-Linear Model Selection

Test RMSE for best tabular data models	
Neural Network	0.0396
Polynomial Regression	0.0415
Random Forest	0.0466
K-Nearest Neighbors	0.0524
Linear Regression (Baseline)	0.0556

All non-linear models, outperform Baseline, and the Neural Network is the current best predictor. We were surprised to see KNN and Random Forest being outperformed by Polynomial Regression. We hypothesize that the small size of our training dataset (only 1000 observations) favours simpler models, whereas KNN and RF require large datasets to be effective.

Part III
Integration of Image Data

So far, we've excluded the images of heart scans. Time to change that. Using Convolutional Neural Networks (CNN), we extract meaningful features from these images to complement the tabular data and enhance our best non-linear model. CNNs perform well in image analysis tasks since they automatically learn hierarchical, translation-invariant features (e.g., borders, textures, anatomical patterns, etc...) from the images' pixels.

1 Image data processing

Before being fed to the CNN model, the images are transformed (using the *torchvision* library) in the following 3 steps:

1. Each image is grayscaled (for formatting reasons since the images are already black and white)
2. Images are converted to Tensors (to work with *torch*)
3. The Tensor values are Normalized and centered around 0. Each value now has a scale of [-1,1]

2 CNN Architecture

The CNN model has the following architecture:

- **Convolutional Block 1:**
  - input channels: 1
  - output channels: *n_channels*
  - Kernel: 3x3
  - Stride: 1
  - Padding: *use_padding*
- **Max Pooling 1:**
  - Kernel: 2x2
  - Stride: 2
  - Padding: False
- **Convolutional Block 2:**
  - input channels: *n_channels*
  - output channels: *n_channels*
  - Kernel: 3x3
  - Stride: 1
  - Padding: *use_padding*
- **Max Pooling 2:**
  - Kernel: 2x2
  - Stride: 2
  - Padding: False
- **Feature Embedding (Linear layer):**
  - Input Features:
    - If *use_padding*: $n\_channels \times 12 \times 12$
    - else: $n\_channels \times 10 \times 10$
  - output: *n_features*

The variables *n_channels*, *n_features*, *use_padding*, *learning_rate*, *batch_size* and *epochs* are tuned, as before, using GridSearch. No Feature selection is used when training the CNN (CNN itself is, in a way, "Selecting" features). Tuning hyperparameters implies testing our features to see if they explain the target. We therefore use a simple linear layer to try and predict the risk of developing heart failure from the generated image features.

The best combination of hyperparameters found is *n_channels = 16*, *n_features = 4* and *use_padding = False*, *learning_rate = 0.0001*, *batch_size = 50* and *epochs = 20*, yielding an average cross-validation RMSE = 0.0738.

In addition to tuning hyperparameters, other architectures were attempted, changing the number of convolution layers and using Average Pooling instead of Max Pooling. However, the architecture stated above performs the best.

3 Tabular and Image Features Fusion

Features created from the image data are then appended to the tabular data. We retrain the Part 2's best model (Neural Network with Greedy Feature Selection) on this new combined dataset.

Since the feature set changed, we thought it best to run another GridSearch, tuning hyperparameters to this new feature set. The best subset of features is found to be:

- *age*
- *blood pressure*
- *cholesterol*
- *hemoglobin*
- *potassium*
- *sarsaparilla*
- *image feature 0*
- *image feature 1*
- *image feature 2*
- *image feature 3*

The best model has the following hyper parameters:

- *activation = 'tanh'*
- *alpha = 0.01*
- *hidden_layer_sizes = (32,16)*
- *max_iter = 200*
- *solver = 'lbfgs'*

Comparing this latest model's test RMSE with previous models gives us:

Test RMSE for best models	
CNN + Neural Network	0.0324
Neural Network	0.0396
Polynomial Regression	0.0415
Random Forest	0.0466
K-Nearest Neighbors	0.0524
Linear Regression (Baseline)	0.0556

The inclusion of convolutional features enhances the model significantly, indicating that the shape and/or position of the heart has an effect on the risk of developing heart failure.

4 Final Prediction

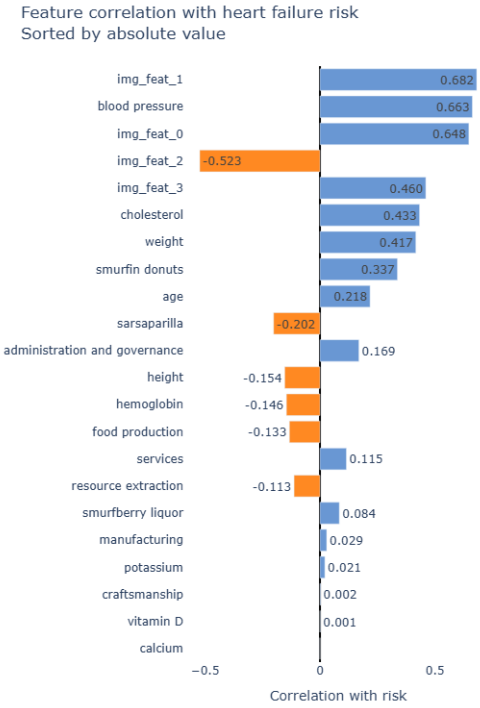
Our final predictions on the unlabeled dataset come from this new combined CNN and NN model. Before outputting these predictions, the NN was trained on the combined X_train and X_test dataset. This allows the models to have more context and therefore produce more robust predictions. However, the downside is that we cannot measure its performance (on the other hand, you dear reader, can :D).

Part IV
Understanding Heart Failure in Smurf Society

Being able to predict heart failure risks is only useful for our Smurfs if we can explain the reasoning behind the predictions. What are the most contributing factors to heart failure risk? How should the affected smurfs change their lifestyle? We can guide the Smurfs using linear correlation, image analysis, and feature importance analysis.

1 Linear Correlation

The figure below shows the correlation coefficients between each feature in the Smurf dataset and the risk of developing heart failure.



We notice that all 4 image features created by CNN present high correlation magnitudes. Since the final layer of the CNN is linear and trained by comparing its output to the risk, high correlation to the target is unsurprising.

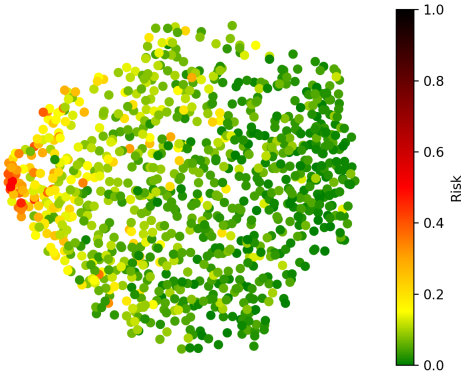
Aside from the image features, the 9 most correlated features to risk exactly match those retained by

the Baseline model's feature selection. Again, this is not surprising, since the baseline only exploits linear relationships. More interestingly, some highly linearly correlated features, such as *weight* and *smurfin donuts* are discarded by the final model. This implies the existence of non-linear relationships, multicollinearity and/or interaction effects between features.

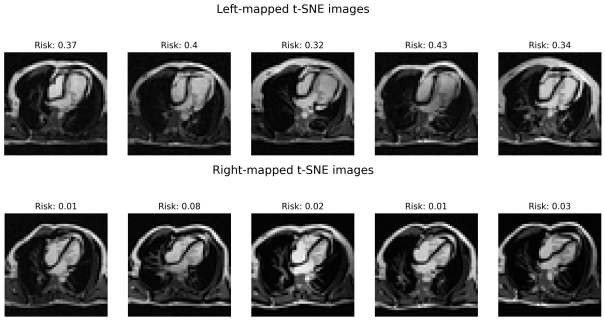
2 Visual Image Analysis

Whereas image features can be understood by the MLP Regressor to make its predictions, finding useful patterns out of them can be more challenging for humans. Therefore, we proceed with a visual analysis of the pictures.

To do so, we perform t-SNE to reduce our 4 image features into 2. t-SNE calculates similarity between all pairs of observations in the 4-dimensional space, then iteratively adjusts their positions in 2D to preserve these similarity relationships using gradient descent. To prevent crowding in the low-dimensional space, it uses a t-distribution. Below is the 2D, t-SNE-transformed dataset of our image features:



Although there isn't a perfect correlation, we observe a clustering of higher-risk hearts on the left and lower-risk hearts on the right. Below are the top 5 leftmost and rightmost heart images in the t-SNE scatter plot.



Between these 2 heart scans sets, there are 2 distinct differences. First, the ratio of the 2 heart ventricles appears more uneven for high-risk hearts, while safer smurfs have more equally proportioned ventricles. Second, some high-risk smurfs have rounder chests, with thicker skin, indicating fat accumulation. Safer smurfs all appear to have skinnier chests with a small indent showing the sternum.

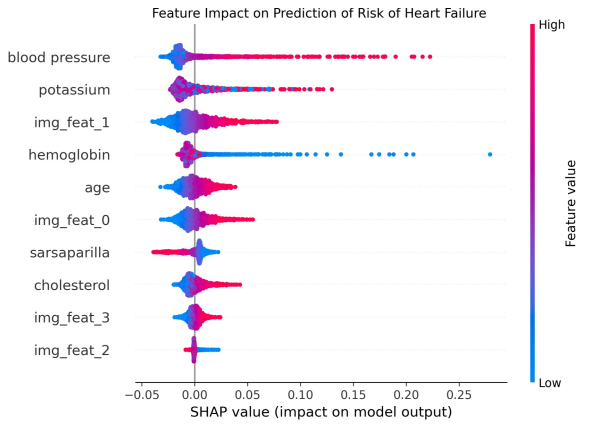
3 Feature Importance (SHAP)

Since CNNs and MLPs are blackbox models, it is very difficult to understand exactly how each connection and step contributes to the final prediction. However, the SHapley Additive exPlanations (SHAP) method helps explain these contributions.

SHAP originates from Game theory principles and quantifies, for every prediction, how much each feature contributed to the prediction. To do so, SHAP compares predictions with and without specific features across all possible combinations of other features, essentially testing the feature's impact in every possible context and outputs a SHAP value. A positive SHAP value means the feature pushed the prediction higher, while a negative value means it pushed it lower, with magnitude indicating importance.

The graph below shows how each feature impacts the model's prediction. In each row, a point represents a smurf, with red representing a high value for that feature and blue representing a low value. Each point's horizontal position represents the strength of the feature influence on the model's prediction for that particular smurf.

For example, a red point in blood pressure represents a smurf with high blood pressure. If that point is positioned to the right, let's say at 0.18, it means that this value of blood pressure increased the prediction of risk by 0.18.



From these SHAP values, we infer the following:

- High *blood pressure* and low levels of *hemoglobin* are enormous contributors to heart failure risk prediction.
- Both very high and very low *levels of potassium* increase heart failure risk prediction.
- High *age* and *cholesterol* also increase heart failure risk prediction, but more moderately
- *Sarsaparilla* consumption reduces heart failure risk prediction.
- High values of *image features 0, 1 and 3* push heart failure risk prediction up, whereas it's low values of *image feature 2* push it up.

4 Recommendations for the Smurf population

Based on our model's predictions and our analysis, we provide the following recommendations to reduce heart failure risk in Smurf society:

4.1 Clinical Interventions

Blood Pressure Management: Blood pressure is one of the most influential predictor of heart failure risk. Smurfs with elevated blood pressure should seek immediate medical attention and implement lifestyle modifications, receiving regular monitoring and appropriate medication when necessary.

Potassium Balance: Smurfs should maintain potassium levels within the normal range through balanced nutrition or supplements if needed.

Hemoglobin Levels: Low hemoglobin, often linked with anemia or poor oxygen-carrying capacity, is linked with increased heart failure risk. Smurfs with low hemoglobin should investigate underlying causes (nutritional deficiencies, chronic disease) and address them appropriately.

Cholesterol Control: Smurfs with high cholesterol should aim to attain healthy levels through diet, exercise, and medical intervention when necessary. Links to consumption of smurfin donuts were not analyzed in this study, however, making an investigation into their nutritional value is recommended.

4.2 Lifestyle Modifications

Increase Sarsaparilla Consumption: Sarsaparilla leaf consumption decreases heart failure risk. It is recommended for all Smurfs to try and incorporate these leaves into their recipes as much as possible.

Weight Management: Smurfs with rounder chests and thicker skin cluster should maintain a healthy weight through balanced nutrition and regular physical activity.

Cardiac Imaging Screening: Given the strong predictive power of imaging features, regular heart scans could serve as an effective early warning system. Smurfs are advised to make regular cardiac check-ups with their local primary care physician.

4.3 Population-Level Actions

The Smurf healthcare system should prioritize:

- Regular blood pressure and hemoglobin screening programs
- Public education campaigns about cardiovascular health
- Accessible cardiac imaging for high-risk populations
- Nutritional programs promoting sarsaparilla consumption and potassium balance
- Exercise and weight management initiatives

By implementing these recommendations, we believe the Smurf population can significantly reduce their collective risk of heart failure and improve cardiovascular health outcomes.

Conclusion

This project developed and optimized a predictive model for heart failure risk in the Smurf population. The Linear Regression Baseline achieved a test RMSE of 0.0556. We found that non-linear models significantly improved prediction accuraccy, with a standard Neural Network using tabular data offering the best performance with an RMSE of 0.0396. By processing image data using a CNN and combining the created image features to the tabular dataset, the Neural Network model achieved the lowest test RMSE of 0.0324, confirming that heart scans provide crucial, information for risk prediction. Analysis using SHAP values and visual inspection of heart scans revealed that high blood pressure, low hemoglobin, and imbalances in potassium, with visual markers such as uneven ventricle ratios and thicker skin/fat accumulation are the most critical predictors. Based on these findings, we provide a set of concrete, evidence-based recommendations for the Smurf population to manage and reduce their collective risk of heart failure.

Appendices

Models hyperparameter search grids

Note: In the code, these grids will look different. They have been modified through several iterations to focus on a broad search at first and tighten the search with time.

Polynomial regression

- Degree: [1,2,3,4]
- interaction_only: [True, False]

KNeighborsRegressor

- n_neighbors: [1-30]
- weights: [uniform, distance]
- metric: [euclidean, manhattan, minkowski]

MLPRegressor

- hidden_layer_sizes: [(32,), (64,), (16, 8), (32, 16), (64, 32), (128,64), (64, 32, 16), (128, 64, 32)]
- activation: ['relu', 'tanh', 'logistic']
- solver: ['adam', 'lbfgs', 'sgd']
- alpha: [0.0001, 0.001, 0.01, 0.1, 1]
- max_iter: [100, 200, 400, 600]
- learning_rate_init: [0.001, 0.005, 0.01, 0.05, 0.1, 1]

RandomForestRegressor

- n_estimators: [50, 100, 200, 400]
- max_depth: [10, 20, None]
- min_samples_leaf: [1, 2, 5, 10]
- max_features: [None, sqrt, 0.5]
- max_samples: [0.6, 0.8, 0.9, 1.0]

CNN

- lr: [0.0001, 0.001, 0.01, 0.1]
- batch_size: [25, 50, 75, 100]
- n_features: [2, 4, 8, 16]
- n_channels: [8, 16, 32, 64]
- epochs: [10, 20, 50]
- use_padding: [True, False]