28462000 - Bourez Nicolas
16452000 - Lausberg Ygor

LELEC2870
21th of December 2023

LELEC2870 - Project report

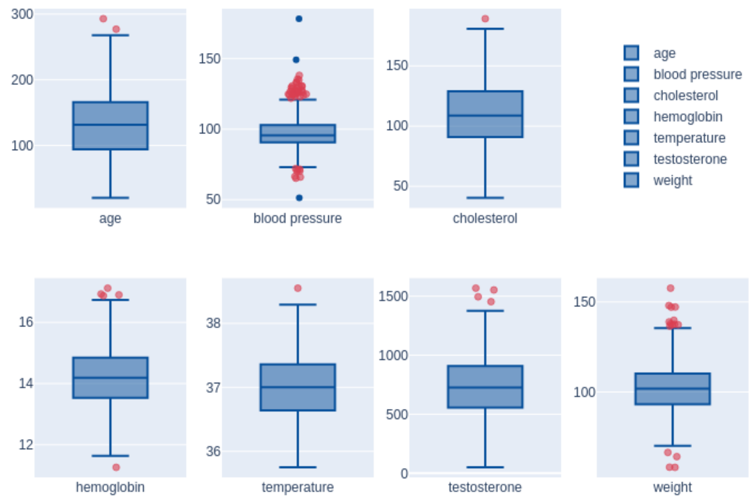
1. Data Engineering

1.1. Checking missing data and mispelling

There were no missing data in the provided dataset, neither as mispelling, so we don't need to perform any data imputation or data cleaning.

1.2. Checking outliers

As outliers can reduce the performance of our model, we decided to check if there were outliers or not in our dataset. We can see on the following box plots that outliers are present for some features.



We could simply discard them to improve our model. But we also need to take the context of our prediction problem into account. We are facing a diagnostic problem, so predicting the diagnose for the majority of the indivuals is important, but what is relevant in this medical context, is to predict the disease of people that seem to not respect the caracteristics of the majority, i.e the *outliers*. We decide to keep them, and to train a model capable of predicting them, even if it will reduce the global performance of our model.

1.3. Dealing with the categorical values

When preparing data for machine learning models, it's crucial to ensure consistency in the input format. This involves separating categorical attributes from continuous ones. In our dataset we can distinguish two kind of categorical features :

- The features which the values gives a certain scale like `sarsaparilla`, `smurfberry liquor` and `smurfin donuts`. we use label encoding to encode the string values as integer from 0 to 4
- The features which the values can be translate into a boolean like `physical activitiy`. Those boolean values are casted to the intigers 0 and 1, 1 being true and 0 being false.
- The features with no scalable values like `bloodtype`. For this feature, we performed one-hot encoding.

1.4. Adding image embeddings

We decided to remove the images filename and to add the image embeddings from the neural network to our dataset. It is a relevant choice as we can see that these embeddings are highly correlated to the target :

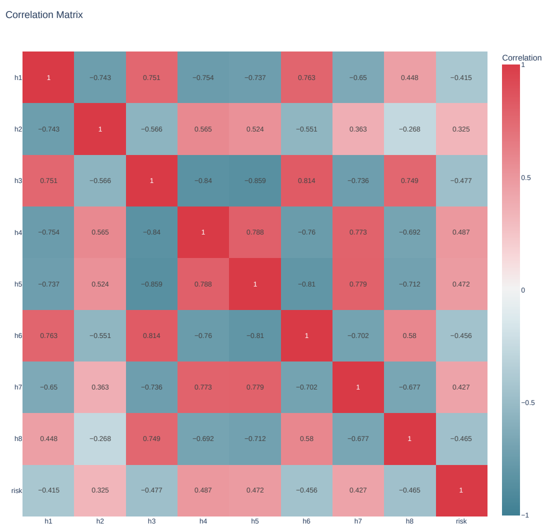


Figure 1: Correlation matrix of the embeddings with the risk

1.5. Standardization of the dataset

We first decide to standardize only the numerical features but we finally decided to standardize every variables as we had better final results. We also decided to standardize the target variables, for the same reasons.

2. Feature selection

We chose two different techniques for selecting the most relevant features :

- At first, we apply a pre-filter based on the **correlation** and the **mutual information**. This filter will give an upperbound of the max number of features to take for the next filter.
- Then, we apply a **wrapper** by chosing the number of features with a range from 5[1] to the upperbound given by the first filter. As this filter is particularly used for the model selection that we explain in details in the next section.

The first filter works as follow :

1. We first calculate the correlation (in absolute value) between each features and the target. We then normalize it.
2. We then caclculate the mutual information between each features and the target[2]. We also normalize it[3]
3. We sum up both metrics and sort in increasing order (see appendix for an example 6).

[1]less than 5 features seem to be ambitious regarding to the correlation and mutual information graph
[2]as it is very random, we run it 1000 times and took the mean
[3]We use normalization to give the same importance to the correlation and the mi

4. The filter consist in dropping the feature one by one (one at each iteration) and calculate the rmse of the model trained on the dataset with only the remaining features (at each iteration). At the end of the loop, it returns the list of features with the least rmse.

The second filter is a just a simple backward selection. We choose the backward because it is the one we had the best results with. Both filters depends on the model, indeed here are the results we got with the different models :

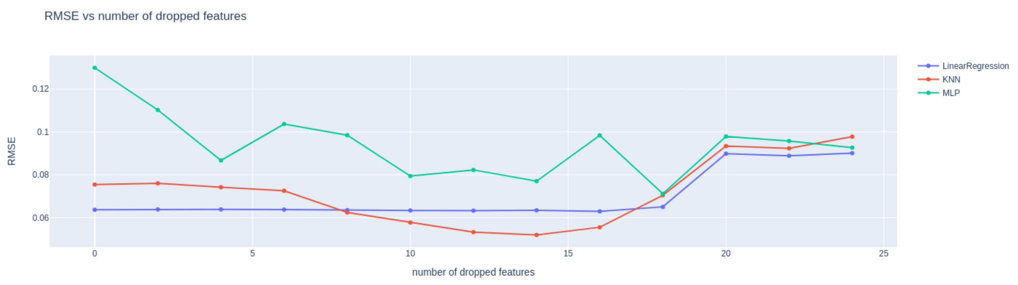


Figure 2: Results of the filter for the 3 models : LinearRegression, KNN and MLP

As the selected features will change after the use of the wrapper (and thus model selection), we will explain them in the next section.

Finally, even if PCA is an accurate method to extract new features and performing feature selection, we wanted our features to stay authentic. Indeed, to interpretation the results, we wanted to compare the selected feeatures with the original set, and show why these features were selected and in what order of importance they influence the risk. If we had chosen to use PCA, it would have been complicated to make links between the selected features and the target.

3. Model selection

To decide which model performs best on our dataset we alternate computation between a model selection grid search and a wrapper. Both of those methods are detailed in the subsections below. We first start by selecting a model based on the features selected in the feature selection phase, then start alternating between the wrapper, that selects a given number of features, and the model selector. The process finishes either when the model is has been degrading for the last 5 iteration or when the model score has converged.

This selects the best model for the selected number of features. Thus to have the best overall performance, we have to compute the best models for every number of features selected between 5 and the number of features selected during the feature selection phase.

We then compare all the computed models to find the overall best model with it's corresponding features. This is done for the K-nearest neighbors, Multilayer perceptron and RandomForest methods[4]. Linear regression doesn't has hyper-parameters to tune.

3.1. Model selection

To select our best hyperparameters, we use the `GridSearchCV` function based on the features selected by the wrapper. We choose a `cv` parameter of 10 cfr. Figure 7, as it is the one we had the best results with.

[4]The wrapper is not used for RandomForest as it is an embedded method

These hyperparameters are selected between a list of parameters that we consider relevant and that are detailed below for each learning method.

- **K-Nearest Neighbors**

  1. `number of neighbors` : 3, 4, 5, 6, 7, 8, 9, 10.
  2. `weights` : uniform, distance
  3. `p` : 1, 2, 3

  We chose these hyperparameters as they're the ones which are the most alike to have a significant impact on the results of the model. Parameters like `algorithms` and `njobs` don't bring a lot of change in our case. Also, after testing our KNN for large amount of neighbours, we decide to give 10 as an upperbound since higher number of neighbours were never improving the model.

- **Multilayer Perceptron**

  1. `hidden layer size` : (2, 10), (4, 10), (2,20), (4,20), (2,100),(10, 10)
  2. `learning rate init` : 0.001, 0.01
  3. `maximum iterations` : 100, 200, 300, 400

  The crucial parameters seems to be the hidden layer size : we didn't chose to large value since we are just performing a regression on a little dataset. Indeed, for a small dataset, using too many neurons or layers might lead to overfitting. Lower learning rates like 0.001 are chosen to prevent overshooting the optimal weights, while 0.01 might converge faster. Max iteration is used for limiting the number of iterations helps prevent also overfitting and controls computational time.

- **Random Forest**

  1. `maximum depth:` 2, 4, 8, 10, 12
  2. `maximum features:` auto, sqrt, log2
  3. `mean samples split:` 2, 4, 8
  4. `number estimators:` 10, 50, 100, 300, 500

  In a regression context with 800 data points, Random Forest parameters like `max depth` guards against over/underfitting, `max features` reduce tree correlations, `min samples split` alters split granularity, while `number of estimators` impacts model robustness and computation. Tweaking these optimizes model performance and generalization.

The scoring function used for the grid search is an adapted RMSE giving more importance to model that over rate the risks. We add a penality to the rmse score if the difference between the prediction and the target is not positive :

```
rmse_custom = rmse + 1.5*sum_neg_distance
```

We did this according to our problem that is used for medical purposes. Indeed, we prefer a good recall, i.e. we prefer over-predicting risk of heart failures than under-predicting risks.

3.2. Results

The best parameters for the models are :

- KNN : `number of neighbors = 6`, `weights = uniform`, `p = 1`.
- MLP : `hidden layer size = (2, 20)`, `learning rate init = 0.001`, `maximum iterations = 200`,
- Random Forest : `maximum depth = 2`, `maximum features = sqrt`, `mean samples split = 4`, `number estimators = 10`,

3.3. Analysis of the selected features

The performance of each model varies in function of the selected features. For example, Linear is performing well when all the features are taken into account, whereas MLP has his best result with taking only 3. But we can still spot some regularities :

- Images embeddings are in general very important, especially h4. This embedding might represent a part of the hart very sensitive to hart problem. These features are also well correlated with the risk, as we seen before.
- However, blood types, hemoglobin, temperature, sarsaparilla, liquor and donuts are way less important. It can also be justified by their weak correlation with the risk
- Finally, features like age, blood pressure, weight and cholesterol are almost always taken in the selection, and every smurf should take these factors into account to avoid hart problems.

4. Final model

4.1. RMSE comparison

Now we have determined the best parameters for each of our regression methods, we have to assert which of them perform bests on our dataset.

For this, we predict the risk of the test set with the 4 different methods. Then, for each model, we calculte the RMSE between the predicted value and the true ones.

To eliminate the randomness of this score, we splitted our training set several times, giving each time a new test set. We repeated this computation 50 times and took the average score to represent the performance of each model. The results are shown on Figure 3.

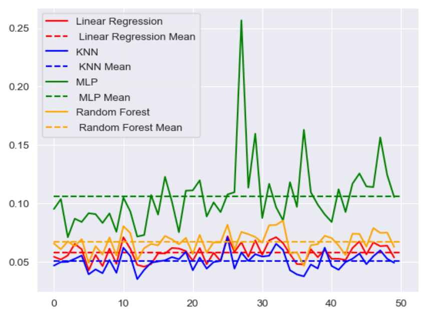
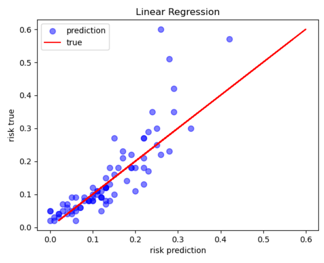


Figure 3: RMSE score for each method

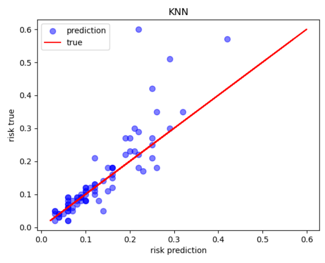
Overall we can observe that our tuned KNN model performs best. Another conclusion we can draw from this graph is that MLP is way less robust, having a big variation, and performs worse. Hereby, we will only consider the other 3 methods that give relatively close results for the rest of our comparison.

4.2. Prediction comparison

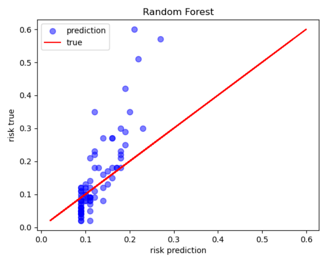
As mentioned before we prefer that our model overrates the risk rather than underrating it. To analyse this we refer to Figure 4, that shows the true value compared to our predicted value. The red line represents the perfect prediction. If a value is situated above this line, it is underrated. Thus we want to have more values under the perfect prediction line.



(a) Linear Regression



(b) K-Nearest Neighbors



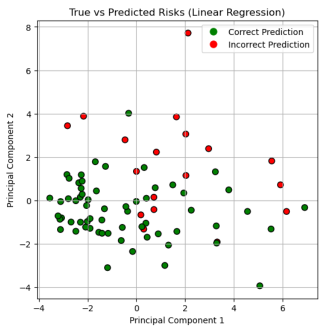
(c) Random Forest

Figure 4: True risks vs predicted risks

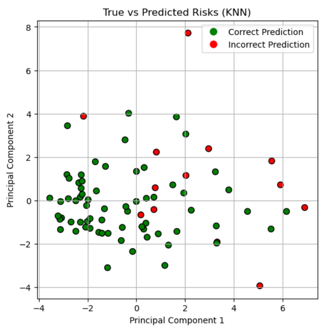
It’s clear that our Linear Regression and K-Nearest Neighbors method follow best the perfect prediction line. K-Nearest Neighbors is slightly better in not underrating the risk too much. For the last comparison, we’re only going to consider those 2 models.

4.3. Right prediction

Here, we compare the number of right predicted risks with a tolerance of 5%. For this we computed the 2 principal components of our dataset, to be able to visualise the test values. The result is shown on Figure 5 .



(a) Linear Regression



(b) K-Nearest Neighbors

Figure 5: True prediction with 5% tolerance

Overall we can conclude that our tuned K-Nearest Neighbors model has the best predictions

4.4. Conclusion

After a tremendous journey in the kingdom of the smurf and their blue medecine, we are happy to replace Papa Smurf’s crystal ball by our well performing model. It might be less accurate than this beautiful artefact, but we did our best and we hope that heart diseases will still just be a bad nightmares for the years to go in the smurf community.

A. example of graph that the first filter is based on

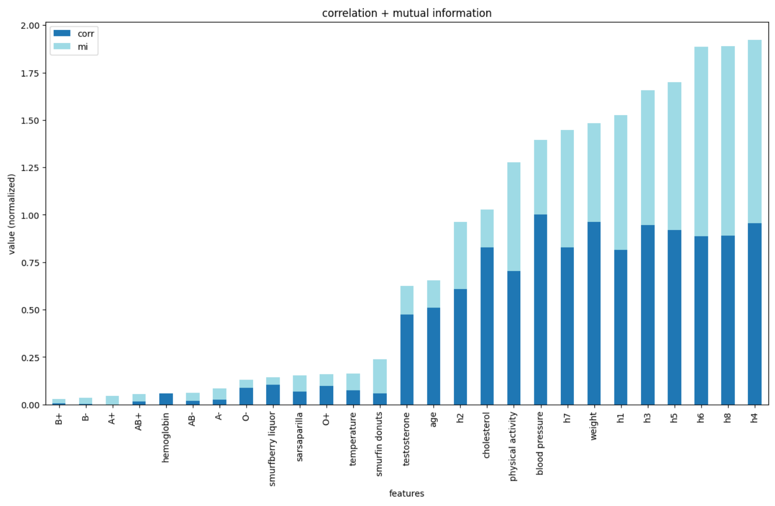


Figure 6: Rank of the features based on both mi and correlation

B. GridSearch CV tuning

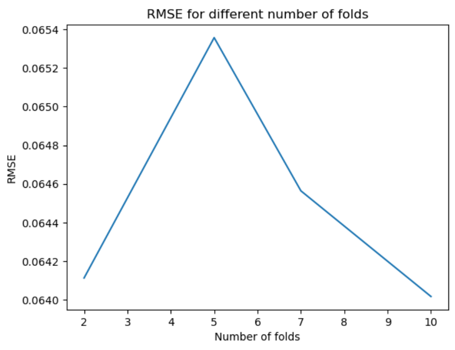


Figure 7: Evolution of the RMSE vs number of k-fold for KNN