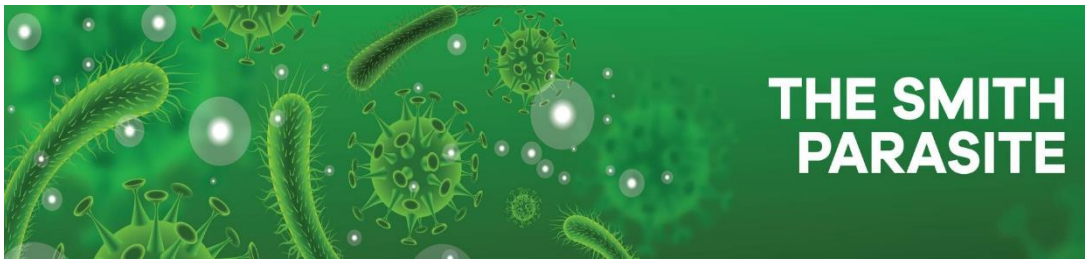

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

MACHINE LEARNING PROJECT

The Smith Parasite - An Unknown Parasitic Disease

Group 08

Mariana Lavinha, number: 20220565

Marta Dinis, number: 20220611

Martim Santos, number: 20220540

Patricia Morais, number: 20220638

Samuel Santos, number: 20220609

December, 2022

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

INDEX

1. Introduction

This report looks into a new disease that has recently been discovered by Dr. Smith, in England. The disease has already affected more than 5000 people, with no apparent connection between them. Our dataset consists of demographic information as well as health and habits information related to each patient.

The main goal of our study is to build a predictive model that can answer the question “Who are the people more likely to suffer from the Smith Parasite?”

In order to achieve this goal, we started by performing data exploration and pre-processing. We performed coherence checks, corrected inconsistent values, identified and handled outliers and imputed missing values. Then we implemented feature engineering, and, finally, feature selection. We obtained satisfactory results on our baseline model, using several classifiers, including decision trees, random forests, neural networks and bagging of decision trees.

Since early in the project, we wanted to have a method to chain several steps, making sure that the output from one step worked consistently as input for the next step. The objective was that we could replace one part and all the rest would work seamlessly.

Professor Carina suggested that we investigated Pipelines from sklearn, and in fact they do precisely this. By implementing a pipeline, it is easier to move from one model to another without risking breaking the code. Pipelines, as the name suggests, join different "pipes" and ensure that data flows from start to finish without leaks. Instead of invoking methods for several features, such as Imputers, Encoders, Regressors, we assemble these elements in a pipeline, and then methods such as fit, transform and predict are invoked at the pipeline level, hiding complexity and ensuring consistency.

In order to setup our pipeline, we developed a ColumnTransformer to create a simple flow where we combine imputers to fill in null values, scalers to standardize numerical features, and encoders to encode categorical features (one-hot encoding and ordinal encoding). This ColumnTransformer encapsulates all these objects in a consistent way. Our pipeline is then, for the most part (exception in Stacking), a combination of this ColumnTransformer with whichever model we’re applying.

Throughout our project, we “use the Pipeline class to simplify the process of building chains of transformations and models.” (Müller & Guido, 2016, Chapter 6), as seen in Figure 12.

Another part of this streamlined process was creating functions to handle data for both train, validation and test data. We developed a function to open the three input files, combine them and handle preprocessing, and the same function is used for train and for test data.

Whenever feasible, we use results from previous methods in the code that follows. For instance, after a RandomizedSearch is performed, we apply a GridSearch whose scope is defined by the best parameters discovered in the RandomizedSearch.

We adapted some code from classes in order to make it fit better our approach. For instance, the myreturn_results function is a variation of the return_results used in class but passing it both the dataframe and the target variable, plus a title for the boxplots, and returning a list with the results.

get_independent_cats function is also an adaptation of function TestIndependence, because we wanted a function that would return a list of categories to drop, instead of just printing it on screen.

2. Exploration

For this phase of our project, we analyzed our data using different Exploratory Data Analysis techniques for classification machine learning algorithms. Our first step is to try to understand our variables simply by looking at the information provided in our dataset and use statistical measures to get some meaningful insights.

2.1. Non-graphical Analysis

The following initial insights were taken:

- The dataset has 800 unique row entries with 18 different variables, 10 of which are **categorical** - *Name, Region, Education, Smoking_Habit, Drinking_Habit, Exercise, Fruit_Habit, Water-Habit, Checkup, Diabetes* - and the other 8, **numeric** - *Birth_Year, Disease, Height, Weight, High_Cholesterol, Blood_Pressure, Mental_Health, Physical_Health;*
- We are dealing with a classification problem due to the fact that our target variable, *Disease*, contains binary values (1 = 'the patient has the disease'; 0 = 'the patient doesn't have the disease');
- *Region* has two values representing the same class ('London', 'LONDON');
- The *Name* variable contains the prefixes Mr. and Mrs. regarding the gender of the patient.
- We checked for null values and discovered 28 records with one feature defined as Null.
- Analyzing the data types, we found out that some were stored as object (region, education, drinking habits, fruit habits, water habits, checkup, diabetes), some as int64 (disease, smoking habits, exercise, height, weight, blood pressure, mental health and physical health) and the rest as float64 (high cholesterol);

2.2. Coherence Check

After the Non-graphical Analysis and, in order to carry out our analysis, we tried to find out if there are inconsistent values in our variables.

First, we analyzed the unique values and tried to notice possible errors before encoding. In the variable *Name* there were two patients with the name 'Mr. Gary Miller'. After analyzing this case we realized that all the information differed and therefore they seem like two different persons. To solve the *Region* values that were representing the same class we used the function *capitalize*() to put all the values in the column 'Region' in lower case and only the first character as uppercase. For the

Mental_Health and *Physical_Health* variables, we decided to establish a range between 0 to 30 days and, for High_Cholesterol, we used a Cholesterol level chart (Figure 7) to check the existence of some extreme values. Thus, we define a range between 0 to 500 to avoid inconsistent values.

2.3. Feature Engineering

Even though feature engineering can be seen as preprocessing, we're documenting it in this section because features created were also subject to analysis. These features are documented in table 1.

a) Gender, obtained from the prefix in the name of the patient. The name itself is certainly not going to help in predicting the disease, but the gender might be relevant.
b) Age, obtained from the birth year. Age is easier to interpret and varies in a more interesting range (23..161 instead of 1855..1993). We also considered that values above 100 years are unrealistic, so we set them as null values to be handled later on.
c) BMI (body mass index, calculated from height and weight).

The next step dealt with visualizing any relations between variables for further creation or transformation. As it can be seen in *Figure 1* and Figure *2*, a pairplot and a correlation heatmap were created to observe the relationship between numeric variables. There could be a relation between height and weight, which is why the previously mentioned variable *BMI* (Body Mass Index) was created. We did not keep this new variable, though, as it didn't help our predictive model.

2.4. Univariate Analysis

A set of histograms was created to present the distribution of all numerical features (Figure 1).

High_Cholesterol has a significantly higher variance than the rest of the variables. Values near 500 are quite high, but nonetheless they're credible values according to Figure 7, so we may consider these as outliers, but we only consider as errors values higher than 500. The remaining outliers will also be analyzed, in particular in the Physical_Health and Blood_Pressure variables.

As for categorical features, we drew some conclusions from the bar charts in the *Figure 3* such as: In the *Education* variable, we can observe that the number of patients with "University incomplete (1 to 2 years)" is much smaller than in the other categories; The number of patients who claim not to consume any type of alcohol is also quite low; as for the *Fruit_Habit* variable, we can observe that most patients eat less than 1 or do not consume fruits every day.

2.5. Multivariate Analysis

For the categorical binary features, we built a grouped bar chart so that we could analyze the relation between each variable and the target, as represented in *Figure 4*.

Our first insights were: most of our data indicates that patients do not smoke, but that does not seem to directly affect the disease since the proportion of patients with the disease is similar in both cases; Patients who exercise seem to be less likely to have the disease and there is a large increase in the number of patients with the disease who do not exercise; As for gender, we can observe that

there are more women with the disease than without. In the case of men, there are more patients without the disease. However, it is important to emphasize that, despite this, the percentage of male patients with the disease continues to be higher than the number of female patients.

For the categorical features, in *Figure 5*, the factors in which we can observe an increase in the number of patients with the disease are:

- *Fruit-habit*: Less than 1. I do not consume fruits every day
- *Checkup*: More than 3 years
- *Diabetes*: I have/had pregnancy diabetes or borderline diabetes and i do have diabetes

One factor that appears to have an influence on the disease is *Diabetes*; patients who claim “Neither I nor my immediate family have diabetes" seem to be less likely to have the disease.

As for numerical features, the most important conclusions to be highlighted from the boxplots in *Figure 6* are: In the variables *Weight* and *High_cholesterol* the data seem to be distributed in a similar way, however, the second one has a much more extreme outlier. In the remaining variables we can highlight the *Age* and *Physical_health* variables with some differences in the data distribution when compared to the target.

3. Preprocessing

3.1 Preprocess function

This function gets a dataframe and does all the manipulation that is to be applied to both train and test data. This ensures consistency and reduces the code that we have to maintain. Handling that is specific to train data (ex. model fitting) is outside this function.

Binary variables are converted in 0/1.

String variables are converted to capitalize case (first letter in each sentence is uppercase).

Feature engineering is performed for birth_year/age and name/gender.

Several potencial inconsistencies are corrected, even though they do not happen in train data, because they could occur in test data. These include agee below 0 or above 100 years, mental and physical health must be between 0 and 30, high_cholesterol must be lower than 500.

3.2 Outliers

“Outliers are values that lie far away from the central tendency of a feature” (Kelleher et al., 2015). It was also done an extensive and detailed outlier process check to our numeric variables (non-binary). This was achieved by checking out each variable individually and inputting logic limits to its observations.

Our first approach consisted in removing outliers from all numerical features using the IQR method. In this case the percentage of data kept after removing the outliers was low as we were losing almost 8% of the data.

When analyzing the Blood_Pressure variable, we used a Blood_Pressure level chart (Figure 8) and, in this case, there wasn't any unusual value. Our boxplot showed some possible outliers but since we are working with a small dataset we decided to keep those values, so we don't incur the risk of losing significant data. As for the variable High_Cholesterol, we decided, considering the previous conclusions and the analysis of Figure 7, to maintain these observations. Even though they were some possible outliers we considered these observations to be important representations of possible groups.

Thus, we proceeded to our final approach, also using the IQR method to remove outliers only for significant numerical features, therefore reducing data loss. From the conclusions mentioned in the feature selection section, we considered significant the numerical variables Mental_Health and Physical_Health so the percentage of data kept after removing outliers is now acceptable (0.9638).

From looking at the boxplots for Height, Weight and Age, all values seem correct and reasonable. There seems to be a potential outlier in a male patient height, but since it's an unusual but possible value, we're going to keep it. It is important to outline that the Age variable went through a slice process when being created, eliminating certain outliers that we would have ended up encountering (age limit of each patient set to below 100 years).

Even though we identified these outliers and removed them from the dataset, we're still going to compare the performance of our predictive models with and without them.

3.3 *Missing values*

In the beginning, only the Education variable had missing values, precisely 13. We should either substitute those values by the mode or drop those values. (Depends on % of missing data)

After doing a transformation of our Birth_Year variable to Age, we replaced with missing values the patients that were older than 100 years since that didn't seem realistic. With this approach the new variable Age got 12 missing values.

We leave our dataset with null values, as we opted to have them inputed inside the pipeline. This enables us to easily change from SimpleImputer to KNNImputer just by changing code in one cell.

3.4 Feature Selection

The next step of our analysis was identifying and selecting a subset of relevant features from a larger set of features to use in our machine learning model. Our goal is to select a set of features that are most relevant to the target variable and that can improve the performance of the model.

Regarding the categorical variables, we used the Chi-Square to test the independence of the variables and check if an independent variable is an important predictor. According to the Chi-Square

test, the categorical variables that we should discard for the final model are Region, Education and Water_Habit.

As for the numerical variables, we used various methods to determine which features to keep in our analysis. Multicollinearity arises when there is a correlation between any two features. In machine learning, it is expected that each feature should be independent of others, i.e., there's no collinearity between them. In that sense, we want to remove features that are highly correlated with other explanatory variables. For the numeric features, in order to understand the relationships between different features in a dataset, we built a correlation matrix as shown in Figure 9. There is no independent variable highly correlated with the target. The correlation doesn't seem particularly high for any set of features, so we must take other approaches for feature selection to get more insights and to decide which variables we should keep. Then we analyze the variance of the features. In this case none of the variables have particularly low variance. So, our next approach consisted of checking if any features weren't significantly correlated with the target variable (using Spearman correlation), in which case it is a prime target for elimination. After the analyses we decided to drop uncorrelated numeric features if the threshold was below 0.25. Thus, we drop Height, Weight, High_Cholesterol, Blood_Pressure and Age. To be sure about our final conclusions, we also resorted to the analysis of the features through the ANOVA test. After looking at various feature selection approaches, we can see that the conclusions are the same when using the ANOVA test and when checking which features are least correlated with the target variable. Therefore, we only kept two numerical features: Mental_Health and Physical_Health.

3.5 *Preparing data*

Before constructing our machine learning models, we need to partition our dataset.

For that we used the train_test_split() method from sklearn library on our merged train dataset so we would end up with a 70/30 split for our training and validation dataset, respectively. This will allow us to have a set to fit our model and a set to evaluate its initial performance with the possibility of further tuning the model's hyperparameters. We also used the stratify parameter to ensure the split sets have balanced target classes.

4. Modeling

Our base model is always a pipeline, consisting of a column transformer object (to impute, scale and encode features) followed by a classifier model. As a proof of concept, we created a base model to test our pipeline processing and check if we obtained consistent values.

Then, we evaluated multiple models in order to understand which models performed better with our data. Most parameters were kept in their default state except for random_state, which we decided to input a constant value and therefore, using the same data, we were able to keep the same results

every time we ran our code. Other parameters were also specified for some of the models, in order to avoid receiving warnings.

The results of our first assessment are presented in figure 7 and the respective boxplots in figure 8. As it can be seen, most of our models got decent to very good F1 scores, with specifically four models standing out: Random Forest; Decision Tree; Bagging Classifier and Neural Networks (MLP). As for the rest of them, it was decided to put them aside due to their insufficient performances compared to the models mentioned above.

The next phase dealt with model optimization by searching and evaluating the best hyperparameter values of each model (Hyperparameter tuning). From the four models that stood out, we decided to only focus on Random Forests and Neural Networks, in this initial phase. This was due to the fact that the Random Forest algorithm is a combination of multiple decision trees and, even though it's less interpretable (visually), it reduces the risk of overfitting, so we decided to focus our time on it. For the Bagging Classifier, the best performance model between the initial ones, we decided to put it aside on this phase and pick it up later on when evaluating Ensemble models.

To find the best hyperparameters for each of these models, Random Forests and Neural Networks, we went through three steps. First, we evaluated hyperparameters to get a sense of possible impacts of different values in the parameter, as shown in Figure 11. We used boxplot visualizations so we could more easily understand these impacts and check the hyperparameter values that performed better. The second step was to do a RandomizedSearchCV to get good ballpark values for hyperparameters, using a wide range of values. Using the values found in the previous step, we took it a step further and, on step three, we also called out a GridSearchCV and observed the results. This fine-tuning was done for parameters very close to the ones obtained through Randomized Search.

These final parameters gave is the best configuration for Random Forest and Neural Network.

We then tested Ensemble models: “Ensemble methods enable combining multiple model scores into a single score to create a robust generalized model” (Swamynathan, 2019, p. 280).

First, we looked at the Bagging Classifier. We wanted to define four hyperparameters; number of sub-models to create, the percentage of data and the percentage of features that each model should receive, and whether to use bootstrapping (ie, using random sampling with repetition). After some experimentation and fine tuning, we established some values to use. We passed six simple algorithms and did not include Random Forest, since it's already an algorithm based on ensemble modeling. We checked results for the standard model and for the “bagged” model. We kept the best Bagging model.

We repeated the process for Adaboosting, whose best model had depth = 3, and GradientBoosting, whose best model had depth=7 and learning rate = 0.3. We saved these best models.

Finally, we created a stacking model, using our optimized random forest and neural network as estimators, and using a logistic regression as the final estimator. This means that both estimators will

be used to make predictions independently, and the final estimator will be responsible for the final prediction.

We then compared the performance of these 6 models, and in the end, we confirmed that the best-performing model is the neural network optimized through randomized/grid search (Figure 12).

5. Assessment

Most assessment was performed in the modeling phase, as expected in a project using CRISP-DM methodology (Fundamentals of Machine Learning for predictive data analytics, Kelleher, John D. & al), in which the preparation, modeling and the assessment phases have strong interactions.

This process took place in several steps of our project and using two different functions we previously created (avg_score and myreturn_results) that calculated the models' f1-score. We used myreturn_results when we were comparing different models and searching for the best hyperparameters, because it allowed us to plot the results of the Stratified KFold cross validation, thus ensuring better visual comprehension, as shown in Figures 11 and 12. On the other hand, we used the function avg_score when we wanted to test our best models and evaluate if they were overfitting.

5.1 Function avg_score

The function avg_score receives a split_method (tipically, K_Fold), a data set, a target variable and a fitted pipeline. The split_method is used to separate the dataset between train and validation. The pipeline's performance is measured for both train and validation data.

The function prints the average score for both train and validation and train, and its difference, thus allowing us to test for overfitting.

5.2 Function myreturn_results

The function myreturn_results receives a list of models, a data set, a target variable and a title.

The function plots boxplots with the results from the models it received. These results consist of the f1-score obtained through the function return_f1score that, in turn, uses the built in sklearn function cross_val_score.

6. Conclusion

The best overall result was obtained using a AdaBoost with hyperparameters base_estimator = DecisionTreeClassifier(max_depth=3)

With this model, we obtained f1 scores of:

Training: 0.997 +/- 0.001

Validation: 0.985 +/- 0.011

Kaggle: 0.97826

7. References

Kelleher, J. D., Namee, B. M., & D'Arcy, A. (2015). *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies* (The MIT Press) (1st ed.). The MIT Press.

Müller, A., & Guido, S. (2016). *Introduction to Machine Learning with Python: A Guide for Data Scientists* (1st ed.). O'Reilly Media.

Swamynathan, M. (2019). *Mastering Machine Learning with Python in Six Steps: A Practical Implementation Guide to Predictive Data Analytics Using Python* (2nd ed.). Apress.

8. Annexes

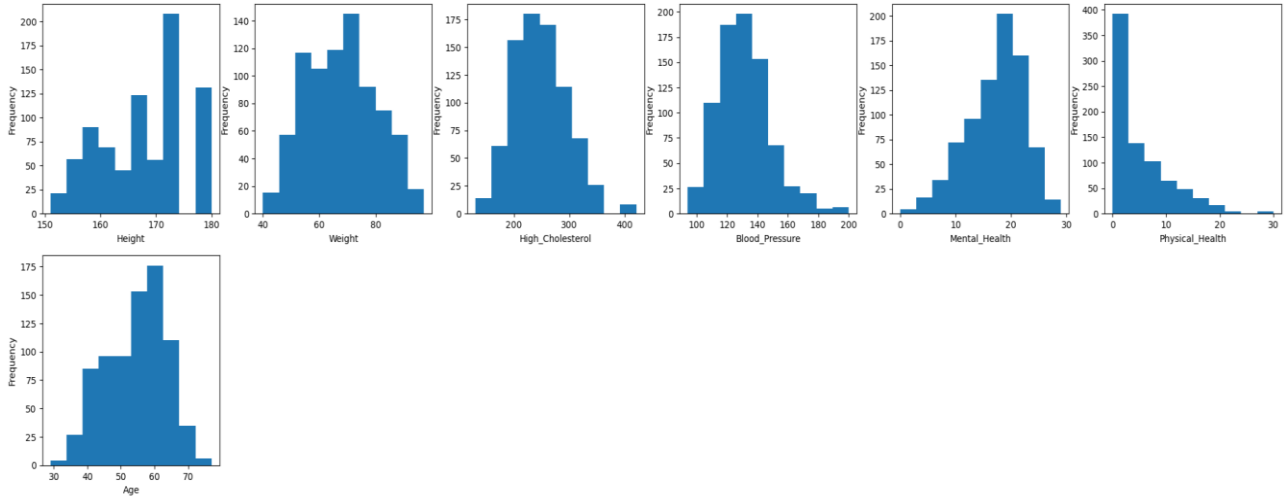

Figure 1: Histograms (numerical variables)

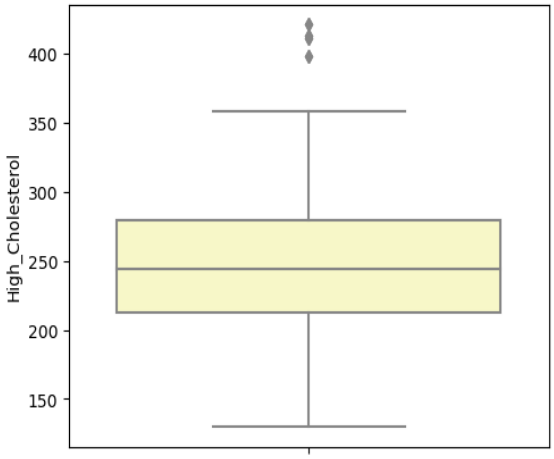

Figure 2: Boxplot High Cholesterol

Figure 3: Bar chart (categorical variables)

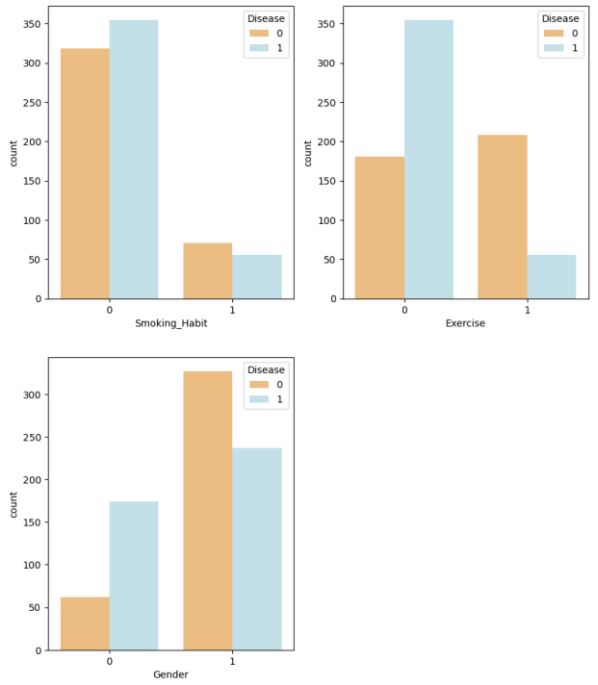

Figure 4: Grouped bar chart (binary features vs target)

Figure 5: Grouped bar chart (ordinal features vs target)

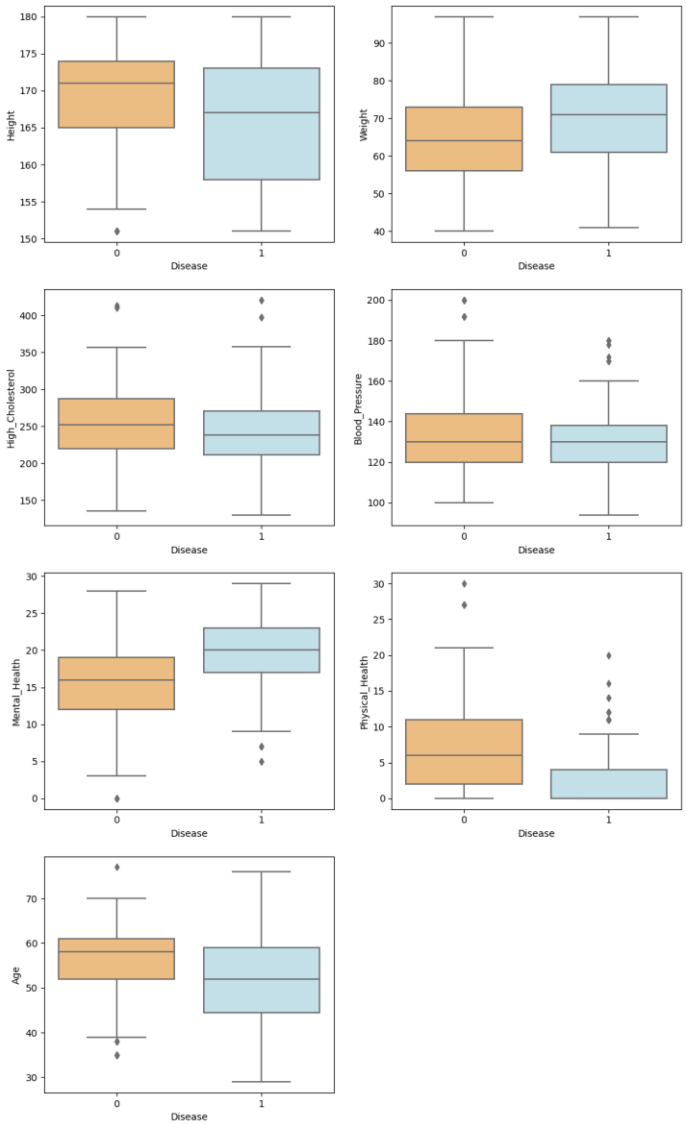

Figure 6: Boxplot (numerical features vs target)

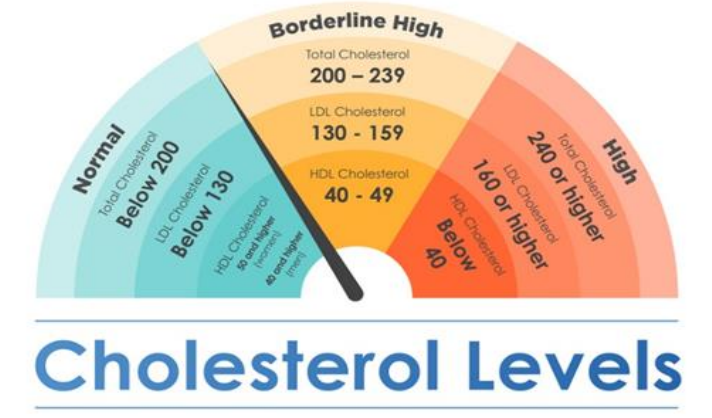

Figure 7: Cholesterol Level chart

Blood Pressure Condition	Systolic / Diastolic pressure (mm Hg)
Hypertension stage 4	210 / 120
Hypertension stage 3	180 / 110
Hypertension stage 2	160 / 100
Hypertension stage 1	140 / 90
High normal blood pressure	130 / 85
Normal blood pressure	120 / 80
Low normal blood pressure	110 / 75
Borderline Hypotension	90 / 60
Serious Hypotension	60 / 40
Very Serious Hypotension	50 / 33

Figure 8: Blood Pressure Level chart

Model	Val
Random Forest	0.980
Decision Tree	0.979
Nearest Neighbors	0.858
Logistic Regression	0.858
MLP	0.954
Gaussian NB	0.847
Linear SVM	0.853
RBF SVM	0.929
SGD	0.814
Bagging	0.987
Ada Boost	0.863

Figure 9: Table with initial validation scores

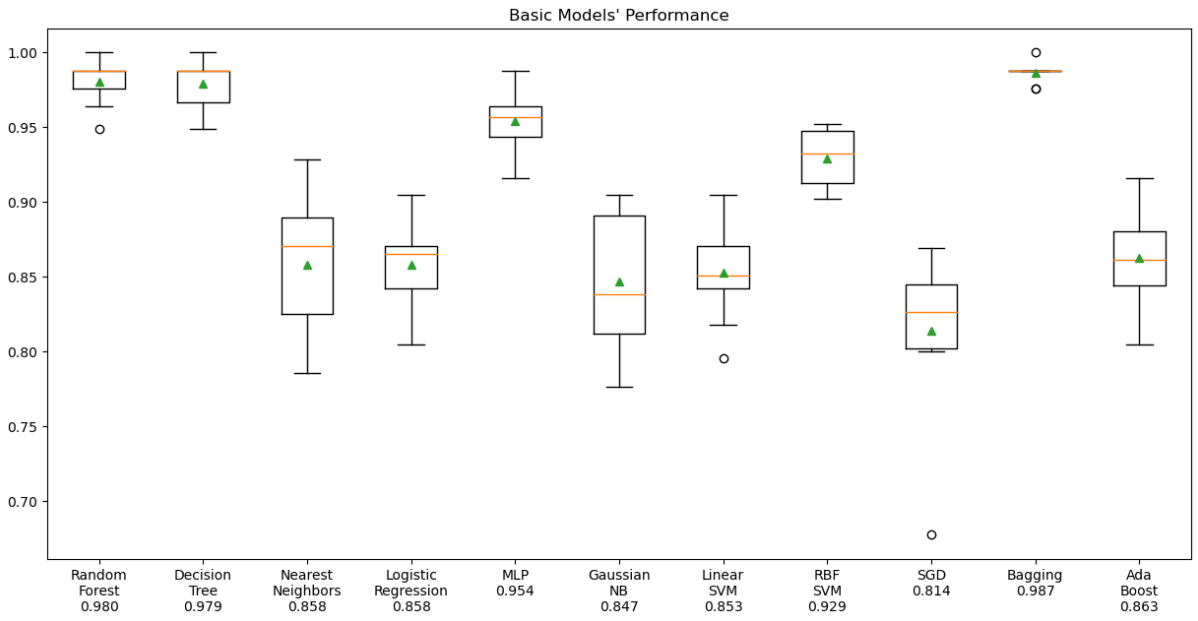

Figure 10: Initial Boxplots with performances of diverse model

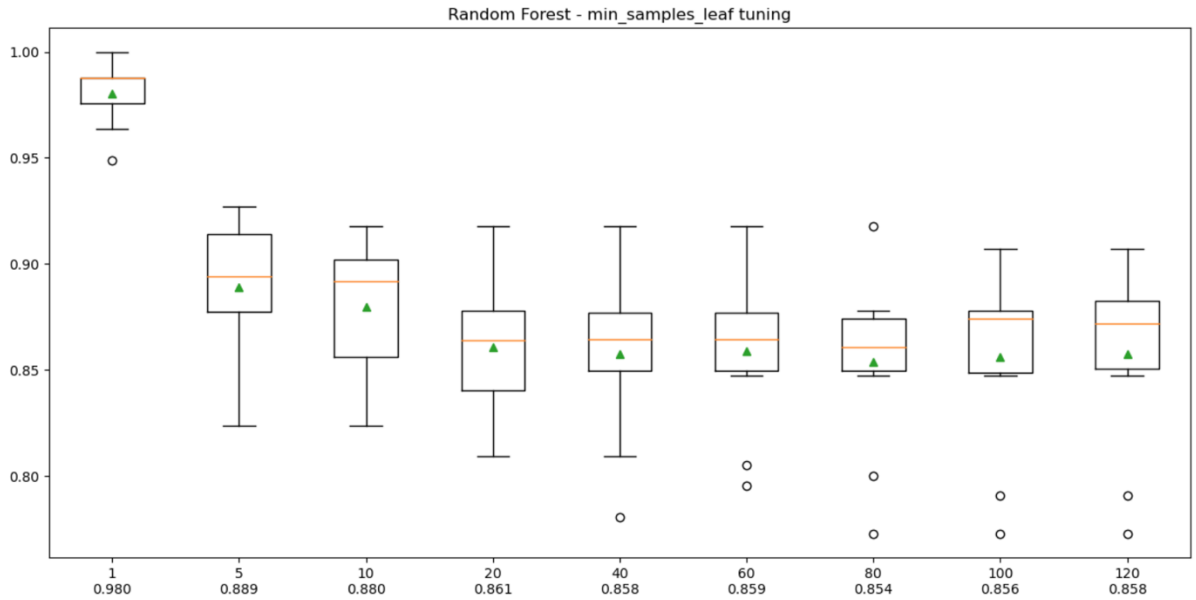

Figure 11: Hyperparameter manual tuning (example rf min_samples_leaf)

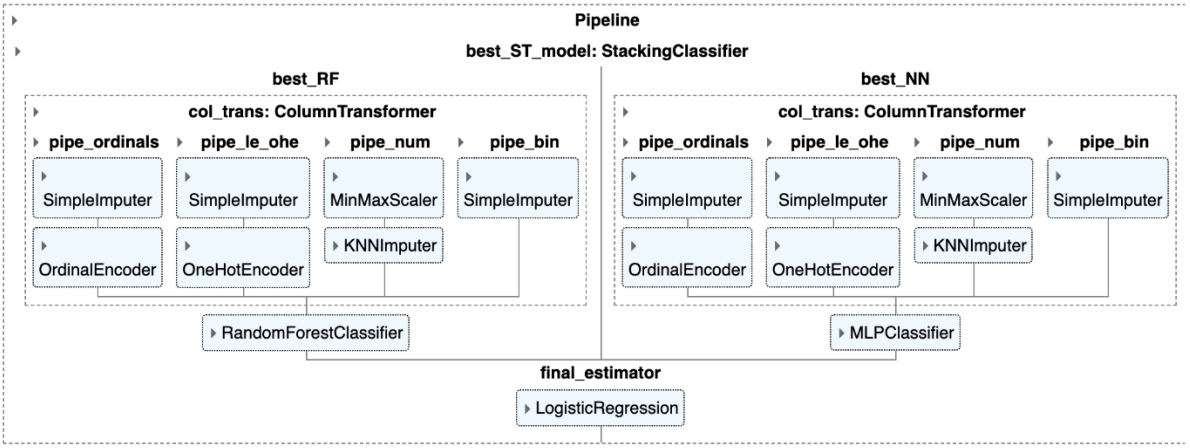

Figure 12: Pipeline for Stacking Classifier

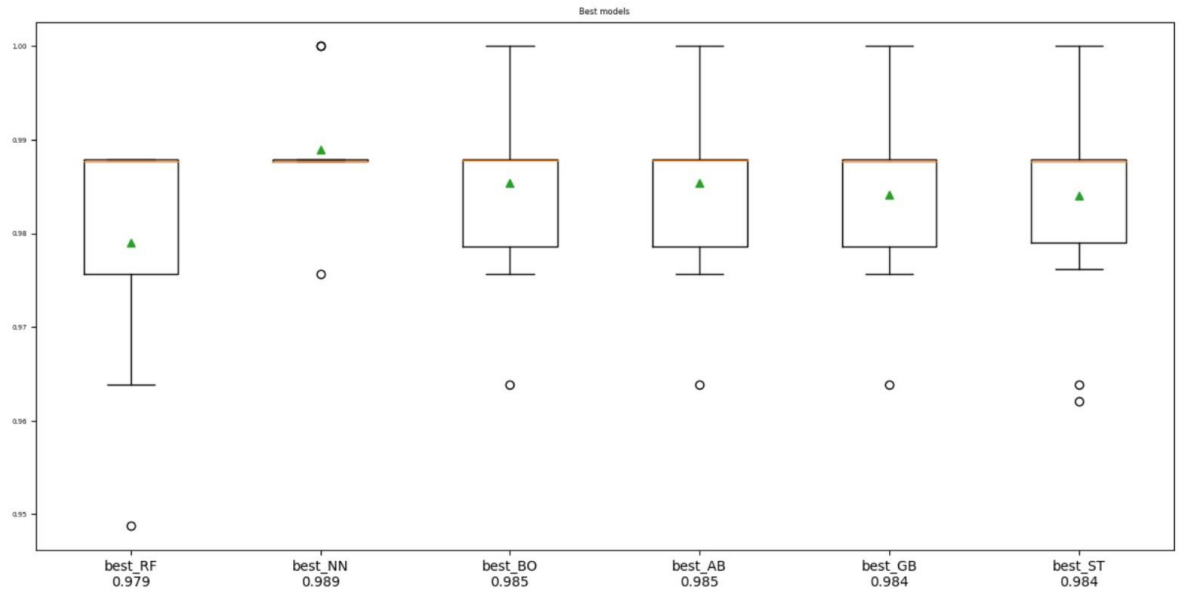

Figure 13 - Best Results

Table 1: Variables created during feature engineering

Variable name	**Calculation**
Gender	If name prefix is “Mr.” then Gender = 0 Else gender = 1
Age	2016 – Year_of_Birth
BMI	Weight/(height/100)**2