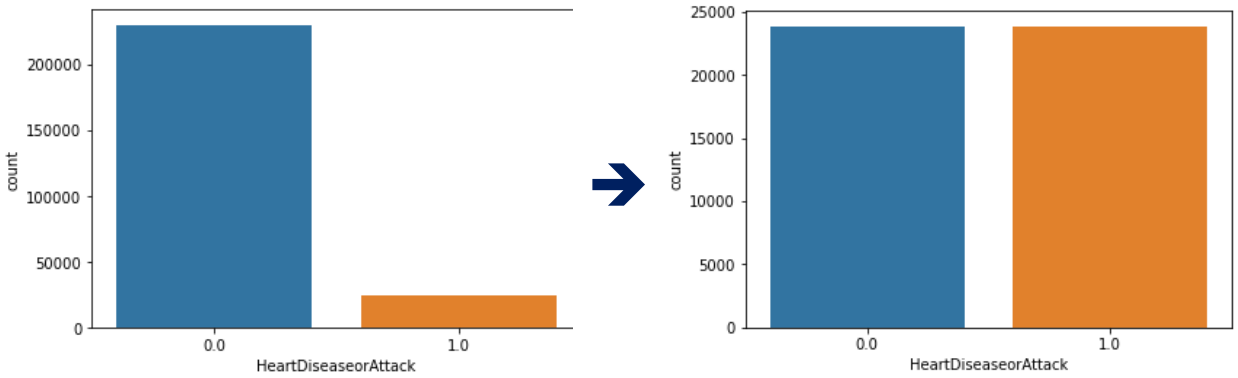
Attached file including code and results: notebook.ipynb

Data and preprocessing

The summary of the data of the file heart_disease_health_indicators_BRFSS2015.csv is the result of the answers of a questionnaire in which people are asked about the factors that are medically considered to increase the possibility of heart diseases. Finally, people are asked whether they have heart disease or not. This collection contains 253,680 rows of information, all of which have a numerical format, but due to the nature of most of its columns, they are categorical variables that are answers to yes/no questions. For example, diabetes, blood pressure, cholesterol, fruit consumption... And on the other hand, there are a number of other variables that know the same situation, but they are multi-class, such as education, income or BMI index and linear age are also reported as belonging to age groups and not the actual number of age. So it can be claimed that all 22 columns of information are categorical. It is important to know this because in the continuation of the work, contrary to the usual routine of neural networks, there is no need for pre-scaling processes. The first column specifies HeartDiseaseorAttack as the target modifier. With a frequency chart, we can understand that the data on the target variable is highly asymmetric. It is also mentioned in the explanation that the target value of 0 means healthy people in this data, there are 229,787 rows in the blank, and there are 23,893 data for target 1, which is sick people. This ratio of 1 to 10 makes learning difficult. If we have a model that recognizes all people as healthy, the model will have good accuracy and loss results, even though it has not done anything! To solve this problem, I use random sampling to make the data symmetric, and for this I used imblearn.under_sampling, which reduced the number of healthy people and made the data set smaller, as well as eliminating the problem of asymmetry. Now we are working with a set of about 50,000 pieces of data.



Why is it not a big problem to throw away this data? The main purpose of this modeling is to detect disease patterns and all the focus is on separating these patterns from healthy patterns. Now, as it was said before, on the 21 dependent variable columns, all values are categorical and they can take a specific number of values. With an estimated calculation, we can see that the size of this data set is large compared to the work that is supposed to be done and it definitely

has a large number of duplicate values. In addition, on these variables, the variety of the healthy pattern is less than the disease pattern, because as mentioned in the data explanation, all of these are risk factors. I put the data set from the pre-processing step in a separate x and y data frame to continue the work.

Method

It was said that the main goal is to recognize and diagnose the disease pattern in people. Heart disease is a fatal disease and naturally the heaviest cost of misdiagnosing a person's life model is the second cost related to medical procedures that are spent on people who are healthy and after more examinations and tests it is determined that they were healthy. I set the first goal in such a way that the model can identify how many percent of them per patient data set, and in the case that all the samples are recognized as patients, this goal is maximally met, but it is not effective because the second cost becomes heavy and This means that all healthy people have to go through the routine of examinations and so on. To reduce the cost of the latter, the goal is determined in such a way that among all the people identified as sick by the model, how many percent are really sick. So, to evaluate the model, I consider these two factors, the first of which can be interpreted as recall and the second as precision, and it is clear that a balance point is necessary to maximize both together, and both cannot be maximized together.

The way it works is that I make several different categories in several stages and each time I try to reconstruct the models that performed well in the previous stage with changes in the architecture of the neural network or increasing the depth or one or two other methods to see the results. Can they improve or not? Because the neural network models are highly dependent on the initialization of the weights, I got 3 results from the 3fold cross validation method every time, so that if I rely on a good result and move forward from one step, I can be sure that the result was not random. Because in the end, maybe the difference between the models that are going to be selected as the final top model candidate will be small. In the last step, every model that has the best results is put in a 10-fold cross validation process, and its best result is reported as the final model and its average.

In order to get a faster understanding of the architecture of each network, I used two helper functions to draw a network-like shape each time. This shape is drawn each time with visualize_model_nn which uses utils_model_nn_config in its heart. Bias is not drawn in the figures.

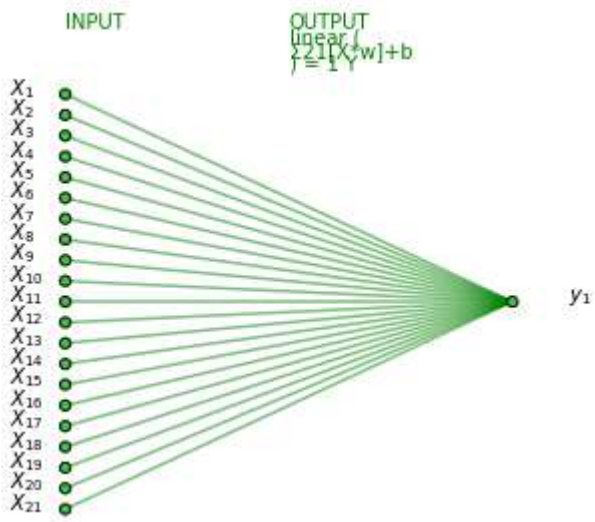
To build models from tensorflow.keras and to categorize test and training in kfold cv from sklearn.model_selection. KFold used.

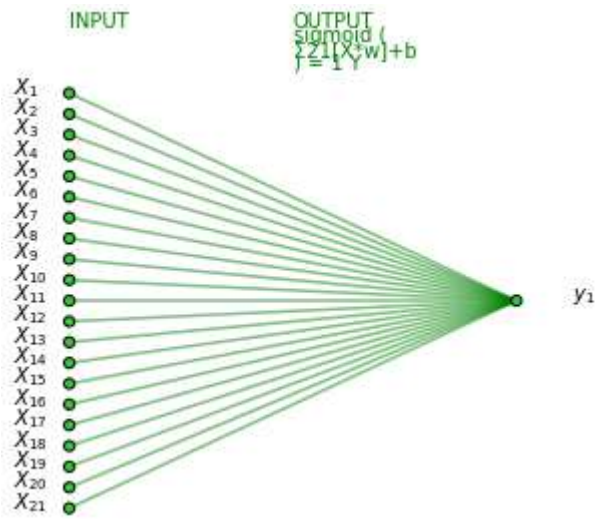
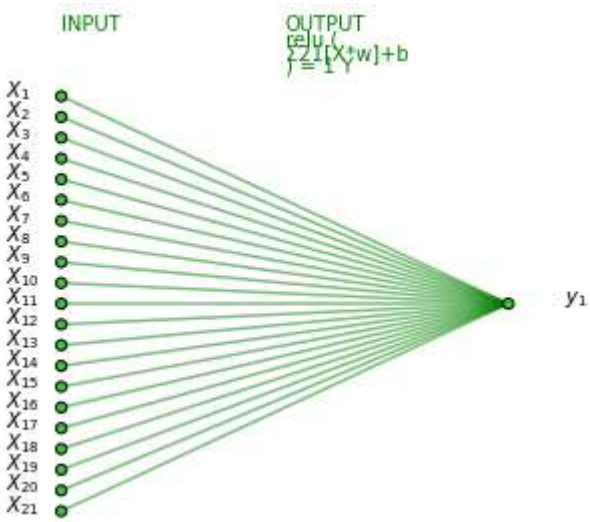
In all the models, I used the Adam method to optimize the weights. The next choice was the decreasing gradient method, which I abandoned due to the discontinuity of the data. For the loss function, I used binary_crossentropy, which is a suitable and recommended choice for two-mode categories with improbable models. All models were trained with epoch = 100 and batch_size = 64.

In each run, 30 percent of the data is kept for evaluation, and the result of each model training is 2 graphs, the black one is equal to loss and the other two are recall and precision on two sets of training and evaluation data on 100 epochs. In addition to this, the final model of each of the 3 trainings is reported with three numbers respectively loss, precision, recall for the training set in the first line and for the test set (on each fold) in the second line. The graphs are drawn using the plot_train_val function.

Understanding more about the data

In the first step, I make 3 perceptron bundles to know most of the data. All of them take 21 inputs with a bias and each uses one of the sigmoid, linear and relu functions as the actuator. It is clear that the data is not linearly separable, the main purpose of this work is to compare the behavior of a perceptron with a different activation function to know which one can better disrupt this non-linearity than for the first model that I make as the activation function of the first layer of the same unit. use.

		Loss	Precision	Recall
model_p1				
	average training	4.52	0.77	0.46
	average test	**4.53**	**0.77**	**0.46**
	results on training	2.25 4.72 6.60	0.72 0.79 0.81	0.85 0.37 0.16
	results on test	2.22 4.68 6.70	0.71 0.78 0.82	0.84 0.37 0.17
Perceptron with linear activation function	The training and evaluation chart has many jumps. Convergence is seen only in the first case. In the other two cases, the results are unreliable. The difference in performance in each training is great.			
model_p2				
	average training	0.49	0.76	0.78
	average test	**0.49**	**0.76**	**0.78**
	results on training	0.49 0.48	0.75 0.76	0.80 0.79

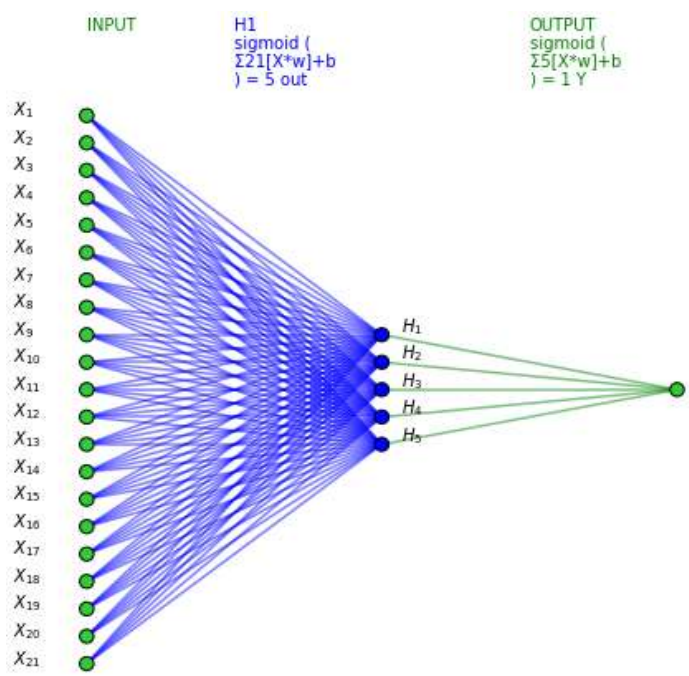
		0.48	0.78	0.74
	results on test	0.48 0.49 0.49	0.76 0.76 0.77	0.80 0.79 0.74
Perceptron with sigmoid activation function	There is convergence in all 3 models. The difference in the criteria of the 3 models has decreased compared to the first case. It gives more reliable results.			
model_p3				
	average training	7.71	0	0
	average test	**7.71**	**0**	**0**
	results on training	7.68 7.74 7.72	0 0 0	0 0 0
	results on test	7.78 7.66 7.70	0 0 0	0 0 0
Perceptron with relu activation function	In all 3 executions, the obtained result is invalid. The training process is not done properly.			

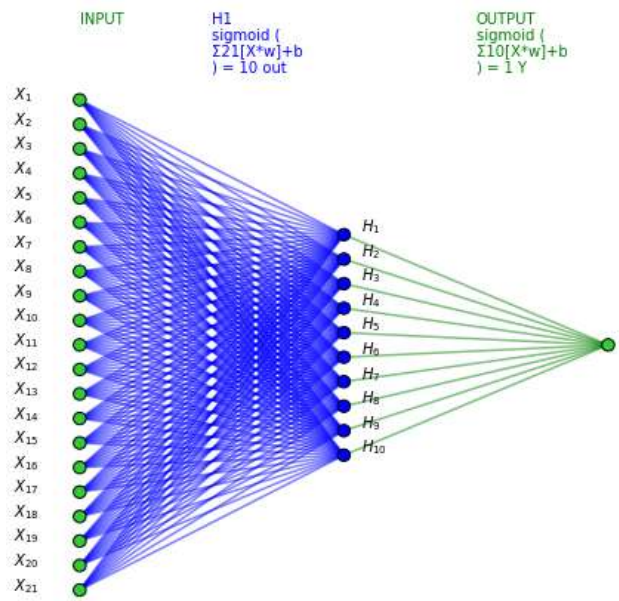
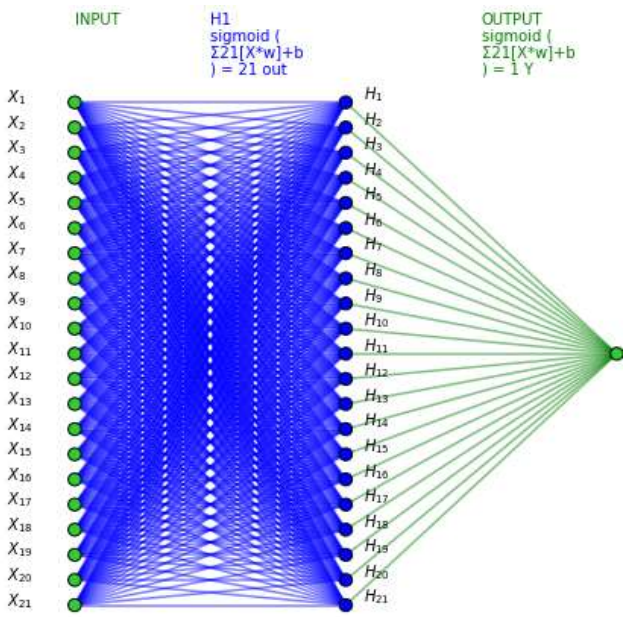
In all of these, the results of testing and training are close to each other, but since these models lack the necessary complexity, I will not do these checks related to fit and error here. The main purpose of this part is that for the next part where I define the first classification neural network, I will use sigmoid for the activation function of the first layer.

Neural network with one hidden layer

In this section, two models model_nn1 and model_nn2 are made. The first question of this section is how much does the presence of a hidden layer improve performance compared to a perceptron? And secondly, how many nodes does this layer have?

The architecture of this section has two node layers (one output and one intermediate or hidden) and one input layer. It is the activator function of the middle sigmoid layer. (The result of the observations of the previous section) The input layer takes 21 dependent variables and the output layer contains an output node that produces the final results with the sigmoid function and performs the task of assigning a class of zero or one (healthy or diseased). In model_nn0, the middle layer has 5 nodes. There is convergence in the training graph of the models and the 3 results are close to each other and are good results to start with. With this observation, I defined model_nn1 and model_nn2 with 10 and 21 nodes in the middle layer to see if increasing this number would improve the results, but the results were very close to each other in both cases and only a little bit of model_nn0 on the recall value Better. Both models of this section will be used in the next sections so that maybe one will be superior to the other in the changes that will be made later, but if we stop here, model_nn1 is preferable because it is simpler (less weights 231 vs. 484).

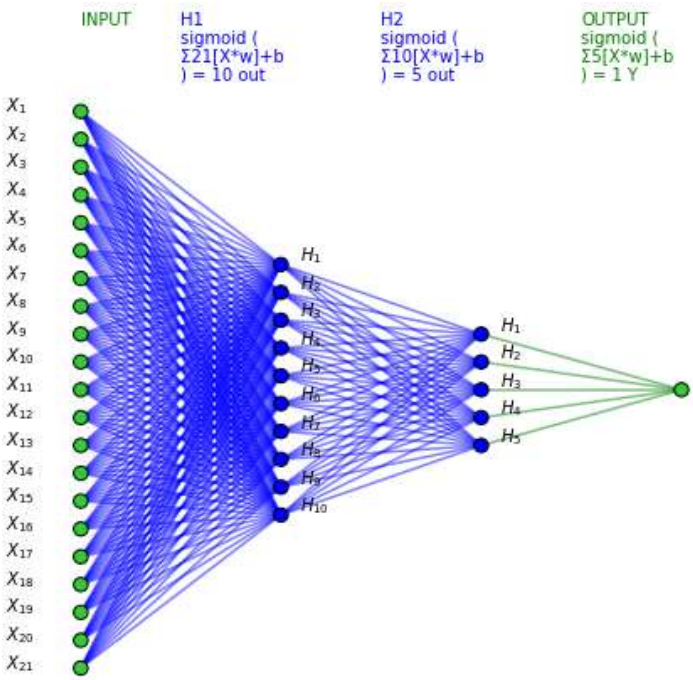
		Loss	Precision	Recall
model _nn0				
	average training	0.47	0.76	0.80
	average test	**0.48**	**0.75**	**0.80**
	results on training	0.48 0.48 0.47	0.75 0.77 0.75	0.82 0.77 0.83
	results on test	0.48 0.48 0.48	0.74 0.77 0.75	0.83 0.76 0.82
model _nn1				
	average training	0.48	0.74	0.83

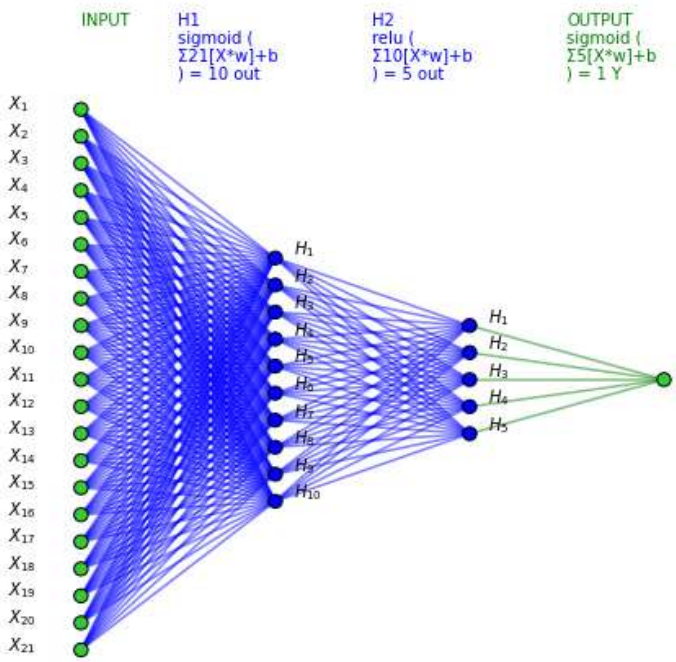
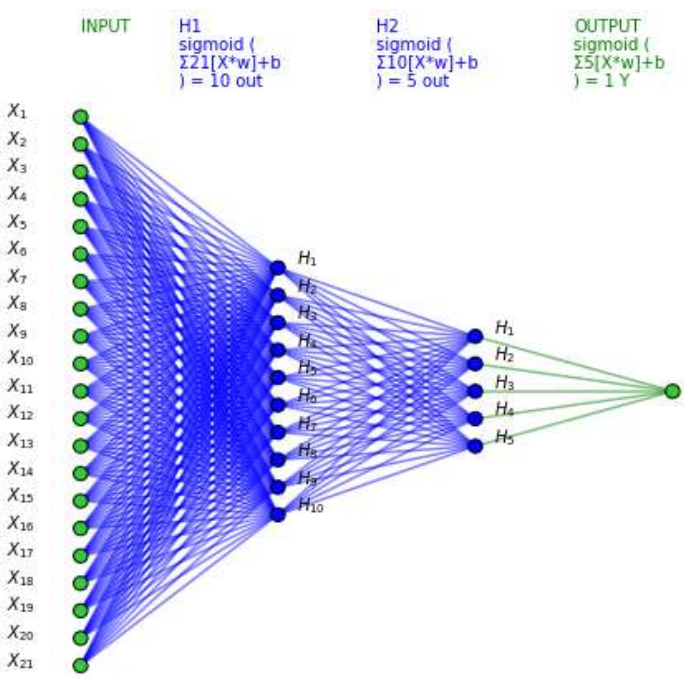
	average test	**0.48**	**0.74**	**0.83**
	results on training	0.48 0.48 0.48	0.75 0.74 0.73	0.81 0.84 0.85
	results on test	0.48 0.48 0.48	0.75 0.74 0.72	0.82 0.83 0.85
mode _nn2				
	average training	0.47	0.75	0.83
	average test	**0.48**	**0.74**	**0.82**
	results on training	0.48 0.47 0.47	0.75 0.74 0.74	0.80 0.83 0.84
	results on test	0.48 0.48 0.48	0.75 0.75 0.73	0.80 0.82 0.84

Neural network with two hidden layers

In this section, I make the network one layer deeper. I put 5 nodes in the second middle layer and train the network on this layer with sigmoid and relu functions. The models of this section are model_dn1 and model_dn2. These two are the result of deepening model_nn1 from the previous section. The results are better on the first one. As can be seen from the results, model_dn1 has a better result on recall, and of course its precision value is lower, but firstly, this difference is less than the value of recall improvement, and secondly, the first goal that was determined in the previous sections is with recall. Another question may arise in this section. If

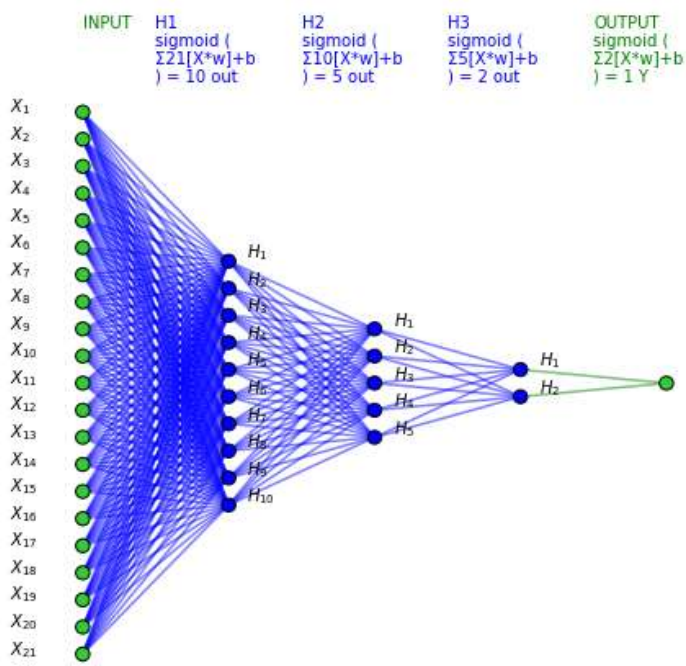
I use the final weights of the model_nn1 layer instead of giving the weights as usual for the random neural network, will the result be better in the end or not? I tried this assumption on model_dn3 and no significant change was seen. In all cases, the training process shows the convergence of the loss function on the graph. If we compare the black line in the test and evaluation chart, none of them show more fit. And in addition, there is a lot of adaptation on the test and training results. From now on, the best result that we got is related to model_dn1 and the effort is whether this result can be improved or not?

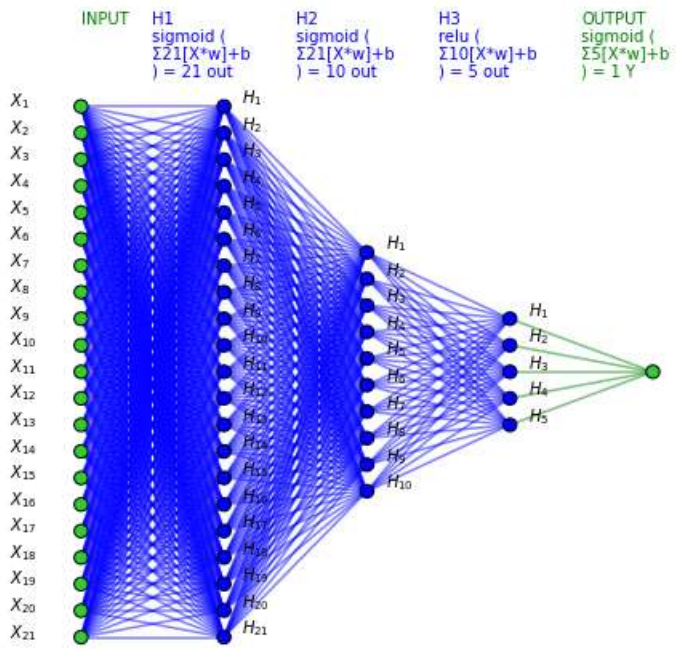
		Loss	Precision	Recall
model _dn1				
	average training	0.48	0.74	0.85
	average test	**0.48**	**0.73**	**0.84**
	results on training	0.48 0.48 0.48	0.73 0.74 0.74	0.86 0.84 0.83
	results on test	0.48 0.48 0.48	0.73 0.73 0.73	0.86 0.84 0.83
Add a second hidden layer to model_nn1 with sigmoid activation function				
model _dn2				
	average training	0.48	0.76	0.80
	average test	**0.48**	**0.75**	**0.80**
	results on training	0.48 0.48 0.48	0.76 0.76 0.75	0.80 0.79 0.82

	results on test	0.48 0.49 0.48	0.76 0.76 0.74	0.79 0.78 0.81
Add a second hidden layer to model_nn1 with the relu activator function				
mode _dn3				
	average training	0.47	0.75	0.82
	average test	**0.48**	**0.75**	**0.81**
	results on training	0.48 0.47 0.48	0.75 0.76 0.74	0.82 0.81 0.83
	results on test	0.48 0.49 0.47	0.74 0.75 0.75	0.82 0.80 0.82
architecture model_dn1 with hidden layer weights model_nn1 as the initial value of the weights of the first layer				

Neural network with three hidden layers

Does deepening the network improve the result? In this first part of the model_dn1 network, which has the best result so far, I deepen it by adding a layer with two nodes. This layer uses the sigmoid function to define the model_ddn1 network. The second architecture that is used with 3 layers, model_ddn2, is the result of adding two layers and deepening the model_nn1 network. This deep network has the same hidden layer model_nn1 including 21 nodes with sigmoid function, and in addition, in the next two hidden layers, it has 10 and 5 nodes with sigmoid and relu functions, respectively. With the architecture of 3 hidden layers, the result was not improved. In model_ddn2, the graph of the evaluation function shows some signs of divergence as it moves forward, but it is not effective. The parameters of this model are many. In the next part, by applying changes on this model, we try to improve the best result. But this part is the end of the work of deepening the network and more than 2 middle layers are not suitable for this problem.

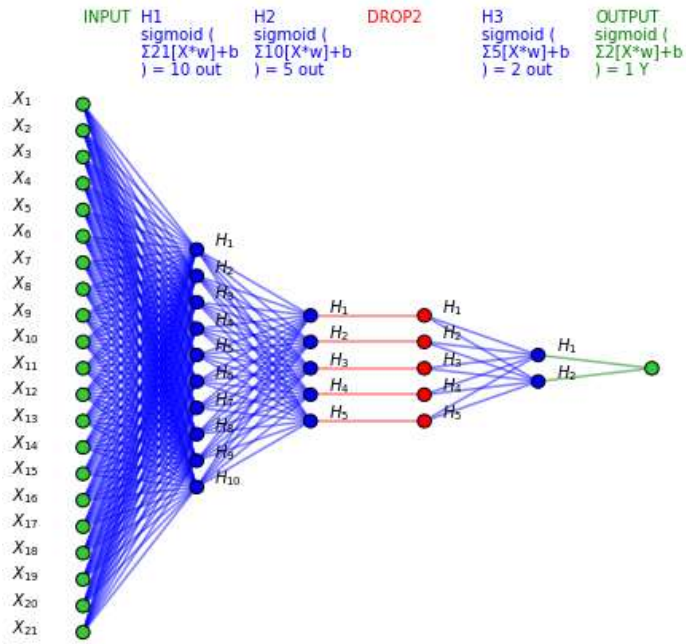
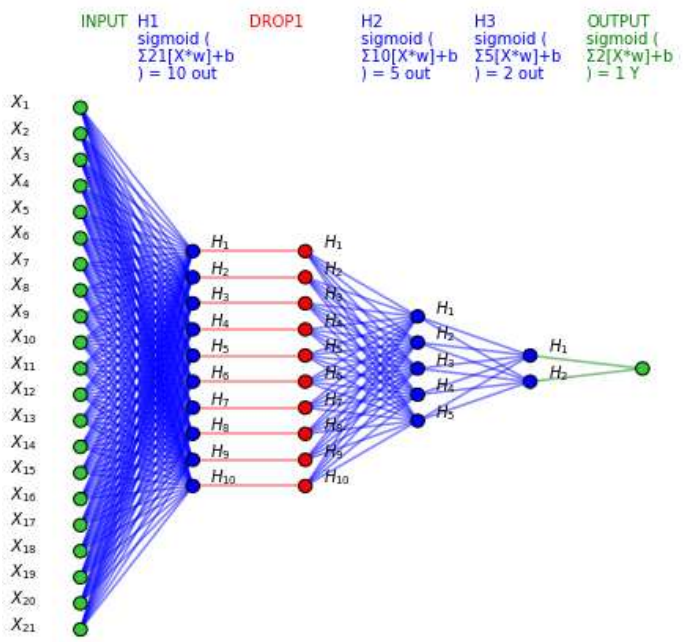
		Loss	Precision	Recall
model _ddn1				
	average training	0.48	0.74	0.83
	average test	**0.48**	**0.74**	**0.83**
	results on training	0.47 0.47 0.48	0.75 0.74 0.74	0.82 0.84 0.84
	results on test	0.48 0.49 0.48	0.74 0.74 0.74	0.82 0.83 0.83
model _ddn2				
	average training	0.47	0.75	0.82
	average test	**0.48**	**0.75**	**0.82**
	results on training	0.47 0.47 0.47	0.75 0.74 0.75	0.81 0.83 0.82

	results on test	0.48 0.48 0.48	0.75 0.74 0.75	0.80 0.83 0.82

Some experiments on the best obtained results:

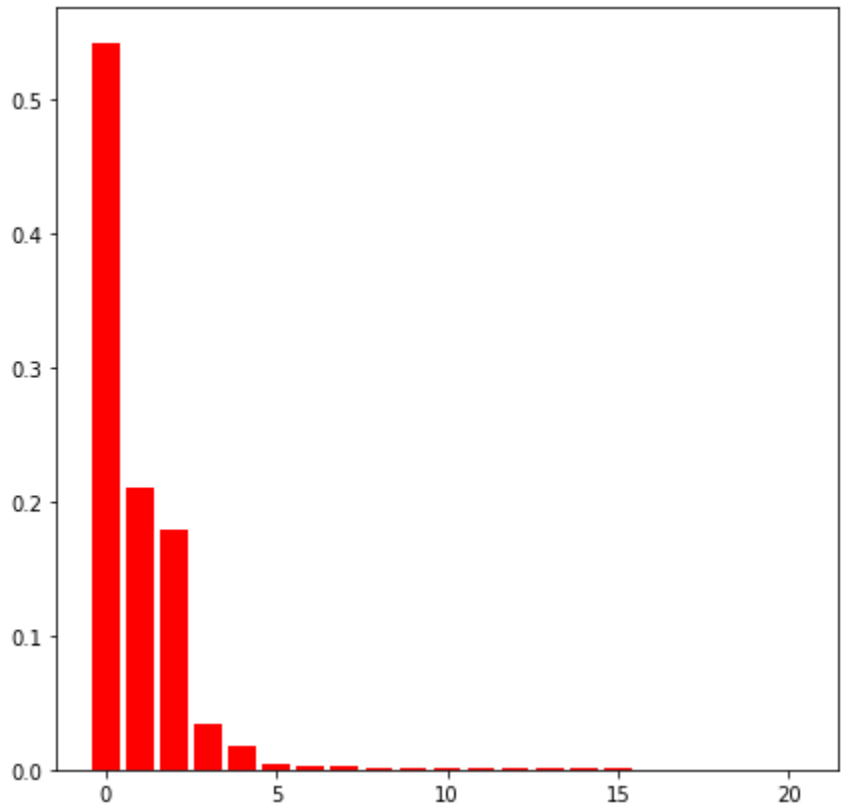
The deepest model that was built and had a good result was the model_ddn1 network, which had 3 hidden layers, but we saw that it was not more efficient than the 2nd network with the same hidden layer. Just to test whether this 3 middle layer architecture improves the result or not, I will make the same again, but this time I will reduce the freedom of the model a little. By adding a dropout layer, first the second layer and then in the first layer and then in the second layer in this architecture, I try to randomly zero the training process as much as one tenth of the input weights to the next layer. This usually works. One way to deal with overfitting is in models that have a lot of complexity, but my point here is that by reducing the computational weights, the architecture with 3 hidden layers will work better than 2 layers, or there will still be no improvement. In both cases, the obtained results are similar and close and are better than model_ddn1, but better performance than model_dn1 is not observed. Both of these models are close to model_dn1, which had 281 weights, in terms of the number of weight parameters (290).

		Loss	Precision	Recall
model _tt1				
	average training	0.48	0.74	0.83
	average test	**0.48**	**0.74**	**0.83**
	results on training	0.48 0.48 0.48	0.74 0.74 0.75	0.82 0.85 0.83

	results on test	0.48 0.49 0.48	.74 0.73 0.75	0.83 0.85 0.82
Add dropout after layer 2				
model _tt2				
	average training	0.48	0.74	0.83
	average test	**0.48**	**0.74**	**0.83**
	results on training	0.49 0.48 0.47	0.74 0.75 0.74	0.83 0.82 0.85
	results on test	0.48 0.48 0.49	0.74 0.75 0.73	0.84 0.81 0.83
Add dropout after layer 1				

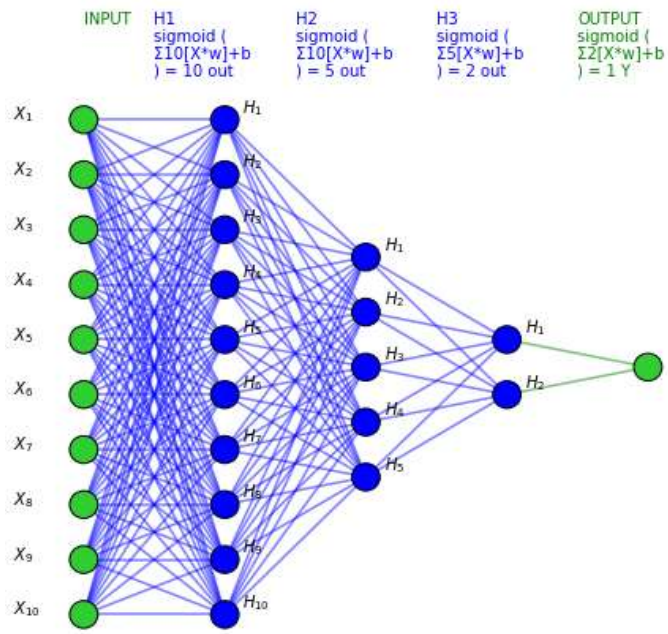
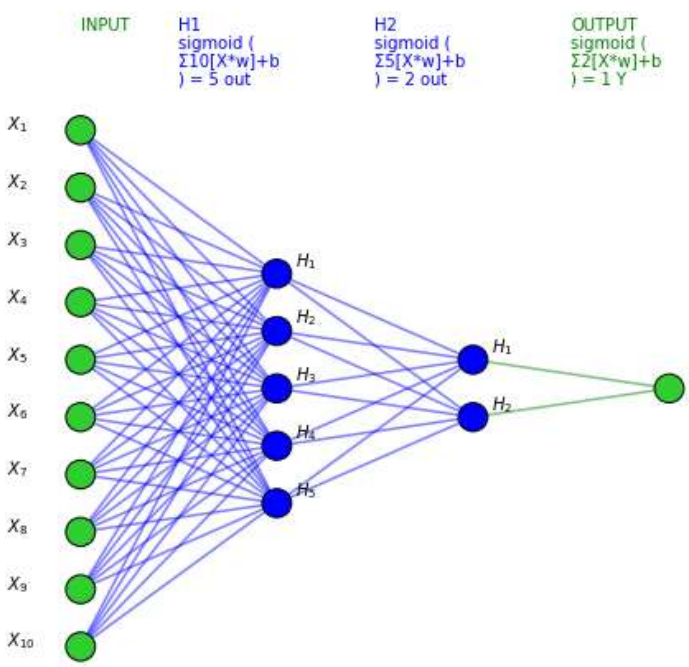
In the experiments section, I will make two other models, the purpose of their construction is to better understand the data and whether it is possible to simplify the best models that have been made so far or not? Here, using PCA transformation, I want to reduce the number of

inputs of model_ddn1 and model_dn1. This may seem strange despite the first explanation of the data being discrete. Especially since the dimensions of the data are not so large. PCA is an unsupervised method and the transformation before entering the information into the neural network is more focused on the relationship between the columns of data and I want to know if among these 21 factors affecting heart disease, meaningful combinations can be found, which is a model with the same optimal efficiency that Here it is obtained to teach but with more simplicity. First, I run pca once on all the data to plot the distribution of variances and know how much I can reduce the input size. Sklearn was used for PCA transformations in this part. The diagram is as follows:



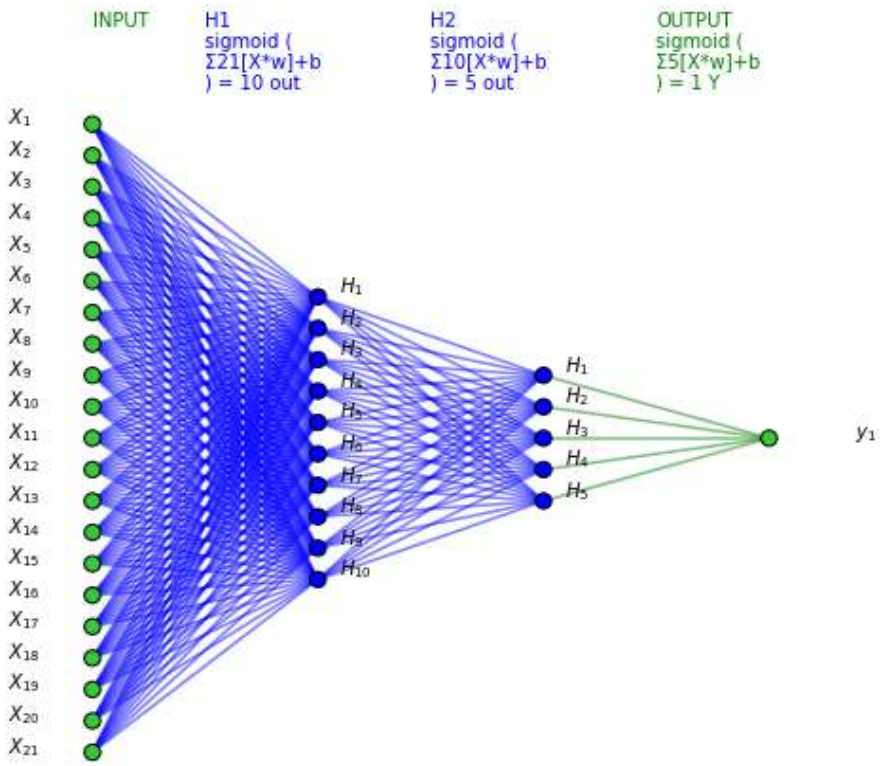
It seems that choosing a number between 5 and 10 would be appropriate. I take the transformation with 10 vectors and this means that my input layer changes from 21 to 10 in the previous models and is halved. Now I make another transformation to the 10-dimensional space and in each of the 3 times of training the model using this first transformation, the training data goes to the new space (each time the PCA transformation is fit with the same set of training data) and then it enters the network And at the end, to evaluate the test data, they are classified with the same transformation in the new space with the network. First, I performed this operation with a 4-layer network (3 hidden layers) which is a modification of model_ddn1, and then with a three-layer architecture (2 hidden layers) modified by model_dn1. The second model with 70 weight is less complicated than all the models that have been made, but interestingly, the difference in its performance with the very complex and

bulky models that were made before is not significant. Although both of these are weaker on both recall and precision.

		Loss	Precision	Recall
model _tt3				
	average training	0.49	0.74	0.82
	average test	**0.49**	**0.73**	**0.81**
	results on training	0.49 0.49 0.49	0.73 0.74 0.74	0.82 0.82 0.82
	results on test	0.49 0.49 0.49	0.74 0.73 0.73	0.82 0.81 0.81
model _tt4				
	average training	0.49	0.73	0.81
	average test	**0.49**	**0.74**	**0.81**
	results on training	0.49 0.49 0.49	0.73 0.74 0.73	0.81 0.81 0.82
	results on test	0.49 0.49 0.49	0.74 0.73 0.74	0.80 0.81 0.81

Selection of the final model

According to the results obtained so far, the best performance on average is on model_dn1, both the amount of complexity and architecture compared to the others and the results obtained are favorable. As the final result of model selection, I run 10-fold cross validation once, and among the 10 results obtained, I select the average as the expected result of this model on this data, and the best result as the final model of this report. In the last step, this selected model will be compared with the results of the work of others who have done modeling in this data.




```
Model: "DeepNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           [(None, 21)]              0
_________________________________________________________________
h1 (Dense)                   (None, 10)                220
_________________________________________________________________
h2 (Dense)                   (None, 5)                 55
_________________________________________________________________
output (Dense)               (None, 1)                 6
=================================================================
Total params: 281
Trainable params: 281
Non-trainable params: 0
```

Precision	Recall	
0.75	0.82	(on 10 fold) Average results
0.74	**0.86**	(best of 10 fold) Best results

Compare with the results of other works

Among the users who had done classification work on this dataset, I mostly came across two cases and the difference between these two usage or non-use of data balancing with random resampling showed the difference in the range of their results. Accuracy was declared in most of the works. Most of the work has been done with linear regression models and tree-based classification methods, and the neural network was little used, in which the recall and precision criteria, which were the bases of my modeling, were not reported. As far as possible, I made a comparison with all kinds of offered models. In each case, I give a brief explanation about the similarities and differences between my method and the method in question. The main weakness of my work in relation to these cases is the lack of attention to statistical methods and tools to better understand the data, in which my work has progressed more with assumptions and conjectures than with a detailed and reliable investigation. And the main difference between my work and these methods is the determination of the two main goals of working with data, which makes the model selection criteria different.

1. https://www.kaggle.com/murattademir/heart-disease-binary-classification

   He has done a lot of statistical work to understand the data, including the use of the correlation matrix and clustering, and calculating the quantiles and variances of the columns and the box diagram, etc. In fact, most of the things that I guessed from visual observation and from the data description information about the data are explained in this work with measurable and accurate calculations and results. Standardization is done on the columns. 3 methods KNeighbors, RandomForest and xgboost are used for classification. The criterion used is accuracy and 90% was reported in all 3 methods. And if precision and recall are calculated from the provided confusion matrices, the following values are obtained:

	Precision	Recall
Kneighbors	0.36	0.08
Randomforest	0.43	0.10
Xgboost	0.52	0.11

   It is clear that my model had better results on these values, because the main goal of this work was to increase accuracy, but my goal, which was explained in the first sections, was to use the model for disease diagnosis.

2. https://www.kaggle.com/abohelal/nueral-network-regression-vs-sklearn-algorithms

The same problem of my guesses against exact statistics is also on this work. Standardization has been done here as before. Both I and here used RandomUnderSampler and used a 4, 4, 1 neural network model with relu functions for the first two layers and sigmoid for the output layer, and for the second architecture, the same was made with a deeper layer with relu in the form of 6, 6, 6, 1, but it is a standard In both networks, the accuracy is reported to be 78%, and I did not calculate and take this into account, and in this work, my recall and precision standards were not taken into account. The results of the networks cannot be compared. The next model is KNeighbors, logisticRegression and decisionTree, which reported the following results:

	Precision	Recall	Accuracy
Kneighbors	0.66	0.60	0.76
logisticRegression	0.69	0.60	0.78
decisionTree	0.53	0.55	0.69

In these 3 models, compared to my final model, no better results were found on the two parameters of precision and recall.

3. https://www.kaggle.com/abohelal/log-knn

Unlike me, he did standardization. Like me, he used RandomUnderSampler and reached the following results with KNeighbors and logisticRegression models. The distinguishable part of this work compared to the previous ones is doing gridsearch and 5fold to find the best k in the knn model.

	Precision	Recall	Accuracy
logisticRegression	0.73	0.79	0.75
KNeighbors	0.76	0.79	0.77

The first regression model on precision works close to my optimal models, and in the nearest neighbor model, the best model that I made has a better result on this criterion, and at the same time, recall is also a good result in both of these, but my work on recall is a more favorable result with 86%.

4. https://www.kaggle.com/mahdishirmohammadi/heart-disease-binary-classification

Without random sampling, to solve the problem of imbalance, it worked with the regression logistic model and worked with more precision to determine the parameters of the modeling, and the final results obtained are as follows:

	Precision	Recall	Accuracy
logisticRegression	0.54	0.12	0.90

Models that are made without correcting the imbalance, like this and the first work models, all have good results on accuracy and precision, about a 50/50 chance, and very low recall.

5. https://www.kaggle.com/davipedrosomartins/pucminas-visualizacao-de-dados

A lot of statistical work has been done on the columns and a lot of effort has been made in terms of data recognition and visualization of this recognition. Finally, a logistic regression classification model with an accuracy of 84 is presented, which is not comparable with the results of my final model due to the difference in criteria.

6. https://www.kaggle.com/funda06/two-layer-nn-heart-disease

Defined a neural network with two layers of 3 and 3 with sigmoid function and defined all stages of construction and definition of model and linear training by back propagation method and performed learning with accuracy and loss criteria. My way of working was mostly to use the keras.Models class and ready and available algorithms in order to more easily search for the best model with my own trial and errors and not with algorithmic search methods. The training of this work is finished with acc = 40%. It is not comparable to my model because of the previous reason.

7. https://www.kaggle.com/mahmoudlimam/undersampling-dimensionality-reduction

He performed well in understanding the data with probability distribution charts and entered the modeling stage with a better understanding of the data than I did. As in my work, RandomUnderSampler is used and random forest is used for classification, which is completely fitted on the training data and has the following result on the test data:

	Precision	Recall	Accuracy
RandomForest	0.79	0.87	0.82

which is completely superior to my model in terms of both precision and recall. Both the neural network and the random forest are models that are bulky and have a lot of complexity, but as seen here, using the forest on these data gives better results.

In the next step, use RobustScaler for pre-processing using SVM and the results on SVM:

	Precision	Recall	Accuracy
SVM	0.75	0.83	0.77

Apart from accuracy, which is absent in my work, my neural network in its average state will perform equivalent to this SVM. In the third step, PCA was used and the data was reduced to two dimensions. In the part where I used pca I worked with 10 vectors and it didn't improve my results. Here too, the pca work is not continued and the goal is only to visualize the data in two dimensions, and the second result of the SVM happens after a reduction of the lda dimension, which is as follows, and except that there is a slight decrease in the rcall,

the rest are unchanged. The aim was to investigate this dimension reduction and its result on the performance of the model.

	Precision	Recall	Accuracy
LDA+ SVM	0.75	0.81	0.78

A significant indicator used in this work was the roc curve.

8. https://www.kaggle.com/mahmoudlimam/model-evaluation-tutorial

He used RobustScaler for preprocessing, checked the distribution charts, but did not perform random sampling and balancing. It uses a decision tree for classification and the result is as follows in 4 iterations.

	Precision	Recall	Accuracy
DecisionTree	0.24	0.27	0.85

The result of regression on precision had better strength. This work has an educational purpose.

9. https://www.kaggle.com/faridrizqis/hertdisease-eda-prediction

Interesting and useful diagrams have been drawn in pre-processing and data recognition. He performed MinMaxScaler on 3 columns and did not do anything with the rest of the columns, which were ordinal, with a two-mode quality. Without changing the default parameters, it performed classification with different models and reported only accuracy on all of them, which is not comparable with my model.

10. https://www.kaggle.com/cienkang/using-logistic-regression

Without pre-processing and balancing etc., we obtained the following results from logistic regression and similar to these results we have already checked in 4. The interesting task is to select the features (of course without statistical checks and only based on speculations like mine about the data) he made two models and trained the same model from all the features and two Roman ones by removing 5 columns in the first one. And finally, he explained the results with the unbalanced data.

	Precision	Recall	Accuracy
logisticRegression	0.51	0.13	0.91
feature selection + logisticRegression	0.51	0.11	0.91

11. https://www.kaggle.com/alexteboul/example-model-easyensembleclassifier

I used the EasyEnsembleClassifier category from the imblearn library and no preprocessing was done. He described the results with many detailed criteria that I will only mention the 3 that were reviewed here:

	Precision	Recall	Accuracy
EasyEnsembleClassifier	0.25	0.79	0.76

Which is an interesting result because all the models I have built so far and studied in other people's work performed better with precision than recall in the blank, which is the opposite here. I think that if my modeling was done without paying attention to the second goal (precision) and went only in the direction of increasing the recall of class 1, it would be close to these conditions.